NYU Derin Öğrenme — LeCun & Canziani (DLSP20)

Enerji-Tabanlı Modellerden Öz-Denetimli Öğrenmeye — Türkçe Notlar

Phase 2

2026-06-07

İçindekiler

1 Önsöz

2 Bu kitap nedir?

Bu, **NYU Deep Learning — Spring 2020 (DLSP20)** ders serisinin Türkçe ders notlarıdır. Ders iki hocalıdır: **Yann LeCun** (Lecture — teorik, araştırma-sınırı bakışı) ve **Alfredo Canziani** (Practicum — pratik, canlı PyTorch). Hedef, videoları izlerken paralel okunabilecek; sonradan tek başına da yeterli olabilecek bir referans seti üretmek.

Bu, Phase 2'nin **en teorik** kursudur. LeCun'un omurga teması tek cümlede özetlenebilir: bir sinir ağını "girdiyi çıktıya çeviren fonksiyon" olarak görmek kısıtlayıcıdır; asıl güçlü çerçeve, cevapları bir **enerji fonksiyonunun minimumları** yapan **enerji-tabanlı modellerdir** (energy-based models, EBM). Kurs, perceptron'dan başlayıp bu fikre — ve oradan öz-denetimli öğrenmeye (self-supervised learning), graf ağlarına ve yapılandırılmış tahmine — tırmanır.

Her hafta bir **Builder Notu** katmanı taşır: kavramın hem **geriye** (lineer cebir, olasılık, calculus — kardeş kurslar) hem de **ileriye** (production, modern araştırma; EBM → JEPA gibi post-2020 köprüler) bağlantısı. Bu seriyi "ezberlenecek DL teorisi" olarak değil, **temelden kurarak anlama** disipliniyle okuyoruz: önce sezgi, sonra matematik, sonra kod.

> **Kaynak**
>
> - **Ders:** NYU-DLSP20 (atcold.github.io) — Yann LeCun & Alfredo Canziani (NYU Courant / Center for Data Science)
> - **YouTube playlist:** NYU Deep Learning SP20
> - **Sürüm:** Spring 2020 (DLSP20) — COVID-19 dönemi kaydı (Mart 2020)
> - **Çeviri ve genişletme:** Phase 2 (TR + ML Builder köprüleri)

3 Nasıl Okumalı

Sıralı oku. Kurs kümülatiftir — her hafta bir öncekinin kurduğu kavramı ve **dili** kullanır. İlk hafta derin öğrenmenin ne olduğunu ve lineer cebir motivasyonunu kurar; ortadaki haftalar ConvNet / RNN / optimizasyon temelini atar; **Hafta 7-9 kursun teorik omurgasıdır** (EBM, autoencoder, VAE, GAN — hepsi tek enerji çerçevesinde); son haftalar SSL, NLP/Transformer, GCN ve büyük sentezi (varyasyonel serbest enerji = VAE) getirir.

Her hafta **iki hocayı tek sayfada** birleştirir: önce **LeCun** (Lecture) büyük resmi ve teoriyi çizer, bir **geçiş** köprüsü gelir, sonra **Canziani** (Practicum) aynı fikri somut koda ve geometriye oturtur. Quote'lar hocaya göre ayrı işaretlenir (**— LeCun, X:XX / — Canziani, X:XX**), misafir hocalar adıyla verilir.

> Pratik bir tavsiye
>
> Önce videoyu izle (Lecture ≈1.5–2 saat + Practicum ≈50–60 dk), sonra buradaki Türkçe haftayı oku, en sonunda **kontrol sorularını** toggle'ı açmadan cevapla ve **PyTorch egzersizlerini** çalıştır. LeCun "neden"i, Canziani "nasıl"ı verir; ikisi birleşince kavram yerine oturur.

4 14 Hafta + 1 Bonus

Kurs perceptron'dan başlayıp modern araştırma-sınırına tırmanır. Her hafta = LeCun Lecture + Canziani Practicum (misafir hocalar parantezde).

Hafta	Konu	Hoca(lar)
1	Derin Öğrenmeye Giriş ve Lineer Cebir Motivasyonu	LeCun + Canziani
2	Gradient Descent, Backprop ve Yapay Sinir Ağları	LeCun + Canziani
3	Evrişimli Ağlar (ConvNets) ve Doğal Sinyaller	LeCun + Canziani
4	Konvolüsyon Cebiri (Toeplitz) ve Optimizasyon	Canziani + Defazio (konuk)
5	1B/2B Konvolüsyon ve Otomatik Türev (autograd)	Canziani
6	Diziler — RNN, LSTM ve Attention	LeCun + Canziani
7	Enerji-Tabanlı Modeller (EBM) ve Autoencoder	LeCun + Canziani
8	Karşıtsal SSL, Sparse Coding ve VAE	LeCun + Canziani
9	Sparse Coding, Dünya Modelleri ve GAN	LeCun + Canziani
10	Görüde Öz-Denetimli Öğrenme (PIRL/MoCo) ve PPUU	Misra (konuk) + Canziani
11	Aktivasyon/Kayıp Fonksiyonları ve PPUU	LeCun + Canziani
12	NLP — Transformer Dil Modelleri ve Attention	Lewis (konuk) + Canziani
13	Graf Evrişimli Ağlar (GCN)	Bresson (konuk) + Canziani
14	Yapılandırılmış Tahmin ve Düzenlileştirme	LeCun + Canziani
Bonus	Transfer Learning (PyTorch Lightning)	Falcon (konuk)

Not: Phase 2 pilotu Hafta 1 ile başlar. Diğer haftalar aynı şablonla eklenecektir.

5 Notasyon

- **Afin katman:** $\mathbf{y} = W\mathbf{x} + \mathbf{b}$ — matris çarpımı (lineer) + öteleme (bias)
- **Doğrusal çöküş:** $W_2(W_1\mathbf{x}) = (W_2W_1)\mathbf{x} = W'\mathbf{x}$ — nonlinearity olmadan derinlik anlamsızdır
- **Enerji fonksiyonu:** $F(x, y)$ — düşük enerji = uyumlu (x, y); çıkarım $= \arg\min_y F(x, y)$
- **Enerji $\leftrightarrow$ olasılık:** $P(y) \propto \exp(-\beta E)$, yani energy $= -\log p$ (sabite kadar) — Boltzmann köprüsü (Stat 110)
- **SVD:** $A = U\Sigma V^\top$ — bir matris = rotation · scaling · rotation; tekil değer $\approx 0 \Rightarrow$ boyut ezme
- **Serbest enerji (Hafta 14):** $F = \langle E \rangle - T \cdot H(q)$ — varyasyonel sentez = VAE

Tüm matematik MathJax 3 ile render ediliyor.

6 Builder Eksen — İki Yön

Her hafta bu bağlantı katmanını taşır

NYU DL kavramı	Köprü
Perceptron / afin katman $W\mathbf{x} + \mathbf{b}$	**Geriye:** 18.06 matris-vektör + Phase 1 DL
Lineer dönüşüm tipleri + SVD	**Geriye:** 18.06 (Ders 29 SVD) + 18.065 ileri
EBM enerji $\rightarrow$ olasılık	**Geriye:** Stat 110 Boltzmann, $-\log p$
Backprop / Lagrangian	**Geriye:** Calculus zincir kuralı + variational
Manifold hipotezi	**Geriye:** 18.06 altuzay + Stat 110 dağılım geometrisi
EBM	**İleriye:** JEPA / I-JEPA / V-JEPA (post-2020)
Contrastive SSL	**İleriye:** BYOL, Barlow Twins, VICReg, MAE (post-2020)
GCN	**İleriye:** graph transformers, geometric DL

Bir tek şey

Derin öğrenme sihir değildir. Elle tasarlanan öznitelik çıkarıcısını çöpe atıp temsili **doğrudan veriden** öğrenir; geometrik olarak bu, veri uzayını doğrusal dönüşümler ($W\mathbf{x} + \mathbf{b}$) ve doğrusal-olmamalarla yeniden şekillendirip kıvrık veri manifoldunu ayrıştırılabilir hale getirmektir. LeCun bunu cebirle anlatır, Canziani geometriyle gösterir.

Anakronizm notu (önemli)

DLSP20 **Mart 2020'de** çekildi. Modern non-contrastive SSL (BYOL, Barlow Twins, VICReg), MAE ve JEPA/I-JEPA/V-JEPA **bu kurstan sonra** doğdu — derslerde **yokturlar**. Bu terimler yalnızca "ileriye köprü" Builder Notu olarak, **(post-2020, kursta yok)** damgasıyla kullanılır. Kursta gerçekten geçen SSL yöntemleri: PIRL, ClusterFit, MoCo, contrastive methods, sparse coding.

7 Yazım Kuralları

- **Türkçe terminoloji + parantez içinde İngilizce orijinal** ilk geçtiğinde: “enerji-tabanlı modeller (energy-based models, EBM)”, “öz-denetimli öğrenme (self-supervised learning, SSL)”, “manifold hipotezi (manifold hypothesis)”.
- **Hoca alıntıları** İngilizce orijinal hâliyle, blockquote içinde, zaman damgasıyla ve hoca etiketiyle verilir (**— LeCun, X:XX / — Canziani, X:XX /** misafirler adıyla).
- **Builder Notu** callout’ları her ana bölüm sonunda; köprüyü (geriye + ileriye) buraya yazıyoruz.
- **Öğretim kodu** (Canziani’nin PyTorch parçaları) görünür `python` bloklarında durur; **figürler** ise kavramı üreten executable hücrelerdir (kod gizli — `echo:false` — yalnızca görsel görünür; bu kurs math-ağırdır, kod ikincildir).
- **Kontrol Soruları** collapse’lu — cevap kapalı başlar, okur kendi düşündükten sonra açar.
- **Egzersizler** cevapsız — en az bir PyTorch / elle-türetme egzersizi.

> ⚠ Bu kitap hocaların yerine geçmez
>
> Tek başına bu set yetmez — LeCun’un kavramsal çerçevesinin ve Canziani’nin canlı kodlama sezgisinin yerine geçemez. Önce videoyu izle, sonra ilgili haftayı oku, son olarak kodu **kendin yaz**. Set videoyu **destekler**, ikame etmez.

8 Derin Öğrenmeye Giriş ve Lineer Cebir Motivasyonu

NYU'nun iki hocalı ritmi: LeCun cebirle 'temsili veriden öğren' der, Canziani geometriyle 'ağ uzayın kumaşını gerer' diye gösterir — aynı gerçeğin iki dili

> Bölüm bilgisi
>
> - **LeCun'un Lecture videosu:** YouTube — Lecture 01: History, motivation, and evolution of Deep Learning (≈99 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Practicum 01: Classification, linear algebra, visualization (≈52 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈35 dk

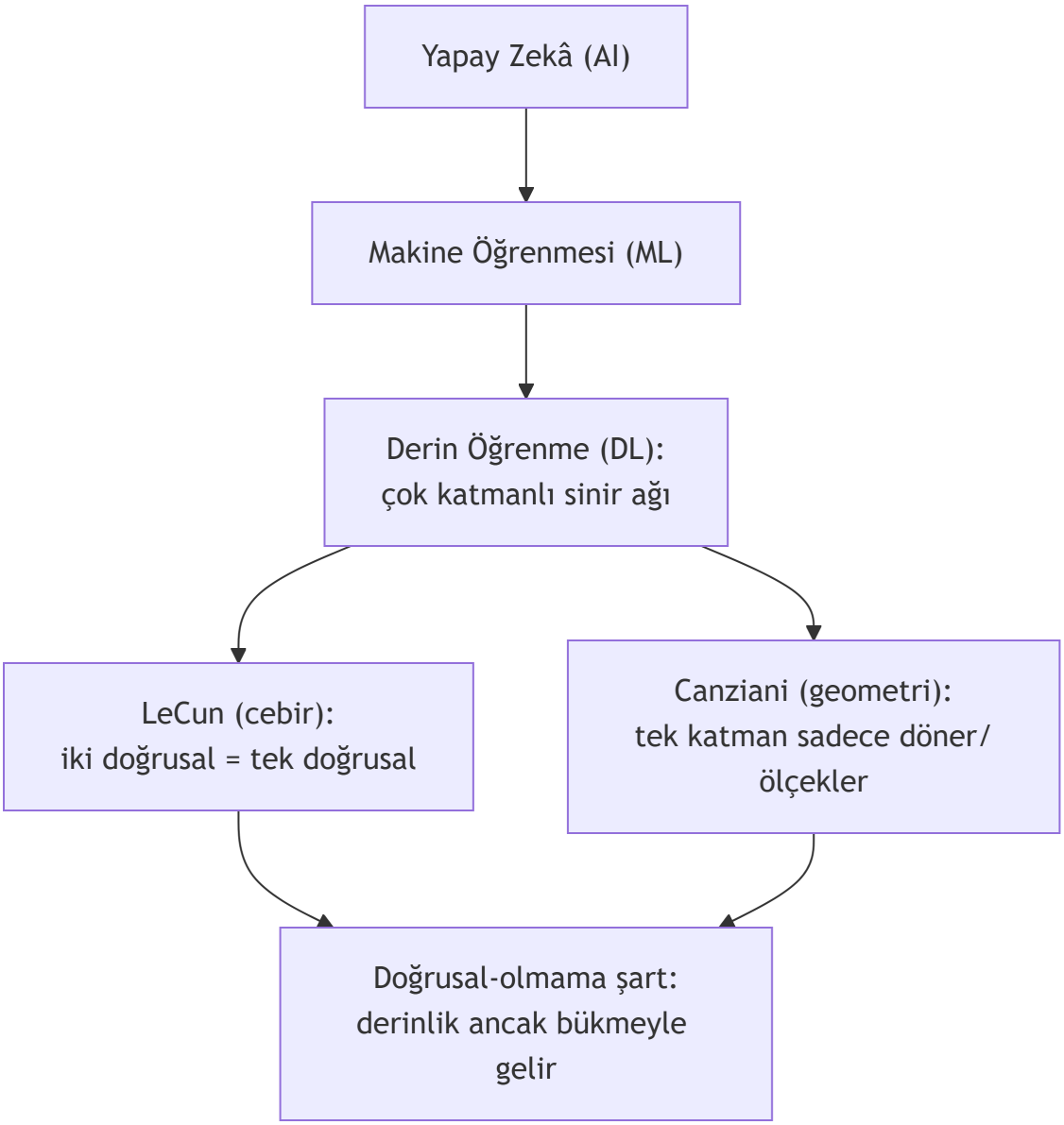
8.1 Bu Derste Ne Var?

Bu, NYU Deep Learning'in ilk haftası — ve kursun **iki hocalı ritmini** ilk kez görüyorsun. Önce **Yann LeCun** (Lecture, Pazartesi akşamı) büyük resmi çizer: derin öğrenme nedir, nereden geldi, neden şimdi patladı. Sonra **Alfredo Canziani** (Practicum, Salı akşamı) aynı fikirleri somut lineer cebire ve PyTorch'a oturtur. LeCun "neden"i, Canziani "nasıl"ı anlatır.

LeCun'un büyük fikri tek cümlede: geleneksel örüntü tanıma (pattern recognition) bir görevi çözmek için **insan eliyle tasarlanmış öznitelik çıkarıcısı** (feature extractor) ister; derin öğrenme bu temsili (representation) **doğrudan veriden** — uçtan uca, katman katman — öğrenir. Canziani ise bunun geometrisini gösterir: bir görüntü, milyonlarca boyutlu bir uzayda tek bir **nokta**dır; bir sinir ağı, bu uzayın "kumaşını" gere gere benzer noktaları ayrılabilir hale getiren bir dönüşümdür.

Bu haftanın üç ana fikri:

1. **Derin öğrenme = temsili veriden öğrenmek.** Elle öznitelik tasarlamak yerine, ağ kendi özniteliklerini katman katman kurar.
2. **"Derin" kelimesi çok katmanlılıktan gelir** — başka bir gizem yok. Her katman ayarlanabilir parametreler + bir **doğrusal-olmama** (nonlinearity) içerir.
3. **Doğrusal-olmama şart.** İki ardışık doğrusal katman tek bir doğrusal katmana çöker (LeCun ve Canziani aynı noktayı ayrı ayrı vurgular).



Builder Notu — İki Yön: Geriye ve İleriye

Geriye (önkoşul kurslar):

- **Perceptron = Wx + b** → Phase 1 DL Ders 1 perceptron + 18.06 matris-vektör çarpımı. Canziani'nin "ağ = matrisler + doğrusal-olmama" çerçevesi doğrudan lineer cebirdir.
- **Lineer dönüşüm tipleri (rotation/scaling/shearing/reflection) + SVD** → 18.06 (Ders 21 özdeğer, Ders 29 SVD) + Phase 2 18.065 ileri.
- **Görüntü = yüksek-boyutlu uzayda nokta, veri düşük-boyutlu altuzayda → manifold hipotezi**; 18.06 altuzay + Stat 110 çok-değişkenli dağılım.

İleriye (production / research):

- LeCun'un "en sevdiği konu" olarak açtığı **enerji-tabanlı modeller (EBM)** — bu kursun omurgası;

Hafta 7'de başlar, bugün JEPA'ya uzanır (ileriye köprü, post-2020).
- Öznitelik öğrenme hiyerarşisi → **öz-denetimli öğrenme (SSL)** (Hafta 8-10) ve **graf ağları (GCN)** (Hafta 13).
- Manifold sezgisi → temsil öğrenme (representation learning) ve boyut indirgemenin tüm modern kullanımları.

! Tek Bir Cümle

Derin öğrenme, elle tasarlanan öznitelik çıkarıcısını çöpe atar ve temsili doğrudan veriden öğrenir; ve bu öğrenme, geometrik olarak veri uzayını doğrusal dönüşümler + doğrusal-olmamalarla yeniden şekillendirmekten ibarettir.

8.2 (LeCun) Derin Öğrenme Nedir? Temsili Veriden Öğrenmek

LeCun ilk dersi kasıtlı olarak yüzeysel ve geniş tutuyor: "bu ilk ders gerçekten geniş bir giriş olacak — derin öğrenmenin ne olduğu, ne yapabildiği ve ne yapamadığı." Tüm kursun yayını burada çiziliyor, sonra her konuya tek tek dönülecek.

Çekirdek fikir şu: yapay zekâ (AI) ⊃ makine öğrenmesi (ML) ⊃ derin öğrenme (DL). DL, ML'in **çok katmanlı sinir ağı** kullanan alt-kümesidir. Ama LeCun'un asıl vurgusu mimari değil, bir *düşünce değişimi*: geleneksel sistemde bir görevi çözmek için **temsili (representation) insan eliyle** tasarlarsın; derin öğrenmede bu temsili **veriden öğrenirsin**.

Bunu somutlaştırmak için LeCun klasik **örüntü tanıma (pattern recognition)** şemasını çiziyor (Bölüm 3'te ayrıntılı). Şimdilik tek cümle: derin öğrenmenin tüm vaadi, "iyi öznitelikler (features) nelerdir?" sorusunu mühendisten alıp **öğrenme algoritmasına** devretmektir.

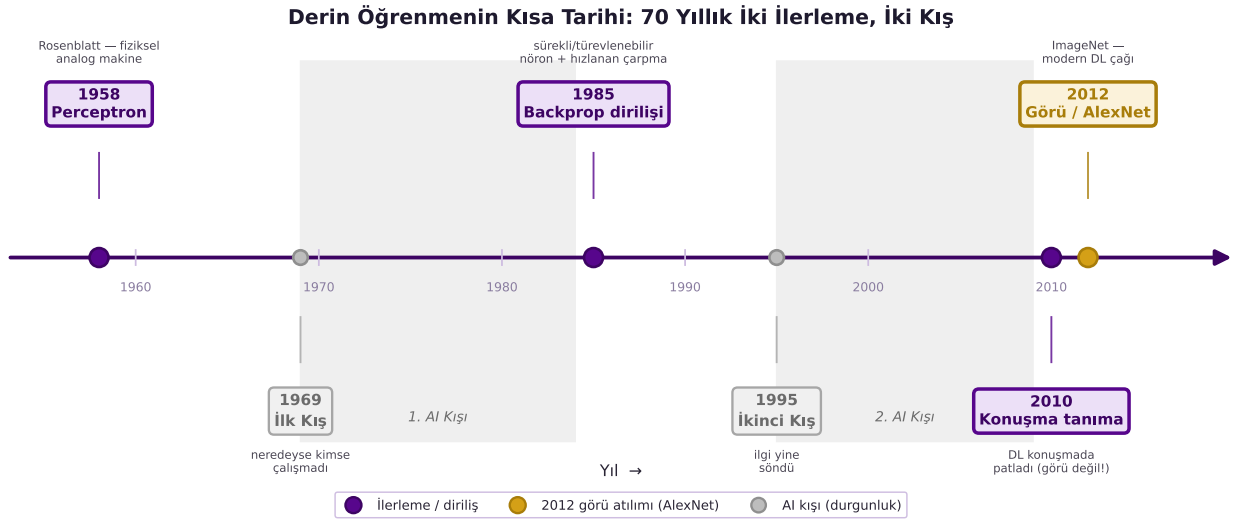
Builder Notu — Temsil Öğrenme Nereye Bağlanır

Geriye (Phase 1 DL): "Temsili veriden öğrenmek" fikri MIT 6.S191 Ders 1'in de açılışıydı (elle mühendislik vs öğrenilen hiyerarşi). NYU bu fikri çok daha teorik bir zemine — enerji-tabanlı modeller ve öz-denetimli öğrenmeye — taşıyacak.
İleriye: "Hangi temsil iyi?" sorusu, modern **representation learning** ve **foundation model** çağının tam kalbidir. Bugün bir modelin değeri, büyük ölçüde öğrendiği temsilin kalitesiyle ölçülür.

8.3 (LeCun) Kısa Tarih: Rosenblatt'ın Perceptron'undan Bugüne

LeCun derin öğrenmenin 70 yıllık inişli çıkışlı tarihini anlatıyor — ve bu tarih, alanın neden "şimdi" patladığını açıklıyor.

1950'ler — Perceptron. Frank Rosenblatt, ağırlıkları öğrenmeyle değişen basit sinir ağlarını hayal etti. İlk perceptron bir yazılım değildi:



Şekil 8.1: Derin öğrenmenin yaklaşık 70 yıllık tarihi: 1958 Perceptron'dan 2012 AlexNet'e iki büyük ilerleme (violet/gold vurgulu) ve aralarındaki iki AI kışı (soluk gri) yatay zaman ekseni üzerinde.

> "this was a physical analog computer, it was not a three-line Python program… it was a gigantic machine with wires and optical sensors… the weights were potentiometers, potentiometers had motors on them so they could rotate for the learning algorithm." — LeCun, 17:18

Yani ağırlıklar, motorla dönen fiziksel potansiyometrelerdi. Sinir ağı fikri 1940'larda doğdu, 1950'lerin sonunda yükseldi.

1969-1984 — İlk kış. Dönemin algoritmaları yalnızca çok basit örüntü tanıma yapabiliyordu. LeCun'un deyişiyle, kabaca 1969 ile 1984 arasında dünyada neredeyse hiç kimse sinir ağları üzerinde çalışmadı — birkaç izole araştırmacı dışında, çoğu Japonya'da (kendi içine kapalı bir araştırma ekosistemi olduğu için modalara kapılmadılar).

1985 — Backprop ile diriliş. Çok katmanlı ağları eğiten **backpropagation** ortaya çıktı. Peki neden daha önce değil? LeCun iki sebep veriyor:

1. **Yanlış nöronlar.** 60'larda kullanılan McCulloch-Pitts nöronları ikiliydi (binary). Gradient descent için **sürekli, türevlenebilir** bir aktivasyon gerekir; bu fikre kimse varmamıştı.
2. **Çarpma pahalıydı.** İkili nöronlar çarpma gerektirmez (yalnızca toplama). Sürekli nöronlar ağırlık × aktivasyon çarpımı ister; 1980'lerden önce kayan-nokta çarpımı çok yavaştı. Donanım ancak 80'lerin ortasında yeterince hızlandı.

1995-2010 — İkinci kış ve diriliş. 1985-1995 dalgası 1995'te söndü. Asıl geri dönüş **2010 civarı konuşma tanımayla** oldu (ImageNet'le değil!): "it didn't start with ImageNet, it started with speech recognition around 2010." 18 ay içinde her büyük oyuncu sinir ağlı konuşma tanıma sistemleri kurdu. 2012-2013'te aynı devrim **bilgisayarlı görüde** (AlexNet) yaşandı.

> Builder Notu — Türevlenebilirlik ve Donanım
>
> **Geriye:** Backprop'un neden 1985'i beklediği — "sürekli, türevlenebilir aktivasyon" — doğrudan Calculus türev/zincir kuralıdır (Phase 1 Calculus Ders 4). İkili → sürekli nöron geçişi, gradient descent'in

ön koşuludur.
İleriye: "Donanım yeterince hızlanınca alan patladı" gözlemi bir builder gerçeğidir: GPU/TPU throughput, modern DL'in motorudur. Aynı kalıp bugün de geçerli — ölçek (compute) çoğu sıçramanın arkasındadır (scaling laws).

8.4 (LeCun) Örüntü Tanıma ve Öznitelik Çıkarıcısı (Feature Extractor)

LeCun, derin öğrenme öncesi dünyayı anlatıyor — çünkü derin öğrenmenin ne yıktığını ancak böyle görürsün. Kabaca dört on yıl boyunca örüntü tanımanın standart şeması iki aşamalıydı:

girdi → öznitelik çıkarıcısı (feature extractor) → eğitilebilir sınıflandırıcı (classifier)

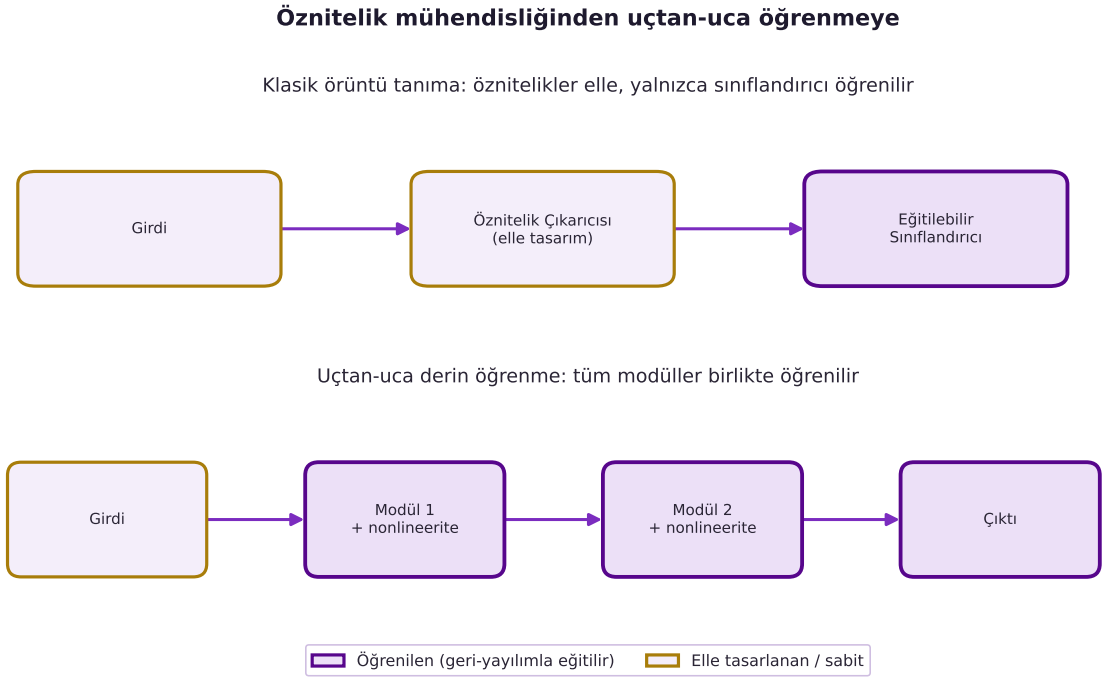
Öznitelik çıkarıcısı, girdiden görevle ilgili karakteristikleri çeker ve bir **öznitelik vektörü** üretir (sayılardan oluşan bir liste). Bu vektör, eğitilebilir bir sınıflandırıcıya verilir. Perceptron durumunda sınıflandırıcı basitçe bir **ağırlıklı toplam + eşik** hesaplar:

$$\hat{y} = g(\mathbf{w}^\top \mathbf{x} + b)$$

Buradaki kritik sorun şu: öznitelik çıkarıcısını **elle tasarlaman** gerekir. Bir yüzü tanımak istiyorsan "göz nasıl tespit edilir? — herhalde bir yerde koyu bir daire vardır" gibi kuralları sen yazarsın.

> "the entire literature of pattern recognition… was focused on this part, the feature extractor: how do you design a feature extractor for a particular problem?" — LeCun, 26:46

LeCun'un vurgusu: on yıllar boyunca tüm emek bu elle-tasarım aşamasına gitti, sınıflandırıcıya çok az.



Şekil 8.2: Klasik örüntü tanıma (elle tasarlanan öznitelik çıkarıcısı + eğitilebilir sınıflandırıcı) ile uçtan-uca derin öğrenme (tüm modüllerin geri-yayılımla birlikte öğrenildiği) boru hatlarının karşılaştırması.

Builder Notu — Wx+b: Perceptron Cebiri

Geriye (18.06): Perceptron'un çekirdeği `w^T x + b` — bir **nokta çarpımı + öteleme**, yani 18.06'nın temel işlemi. Bir katmanda bu, matris-vektör çarpımına (Wx + b) genişler.
İleriye: "Öznitelik mühendisliği" bugün hâlâ klasik ML'de (tablo verisi, XGBoost) yaşıyor; ama görü/dil/ses gibi yüksek-boyutlu alanlarda derin öğrenme onu tamamen değiştirdi. Bu ayrım, bir problemde "deep mi, klasik ML mi?" kararının temelidir.

8.5 (LeCun) Derin Öğrenme = Uçtan Uca Öğrenme (ve Neden Doğrusal-Olmama Şart)

Derin öğrenmenin getirdiği fikir: iki aşamalı (biri elle kurulmuş) süreci bırak, **tüm görevi uçtan uca öğren**. Sistemi bir **modüller dizisi (cascade)** olarak kur; her modülün ayarlanabilir parametreleri ve bir **doğrusal-olmaması (nonlinearity)** vardır; bu katmanları üst üste istifle.

> "the only reason for the deep word in deep learning is the fact that there are multiple layers — there is nothing more to that." — LeCun, 28:14

İlk modülün parametrelerini, çıktıyı istediğin yöne yaklaştıracak biçimde nasıl ayarlarsın? İşte **backpropagation** bunu yapar (Hafta 2).

Peki neden her modül **doğrusal-olmayan** olmak zorunda? LeCun'un cevabı kesin: iki ardışık doğrusal modül tek bir doğrusal modüle çöker. İki doğrusal fonksiyonun bileşkesi yine doğrusaldır:

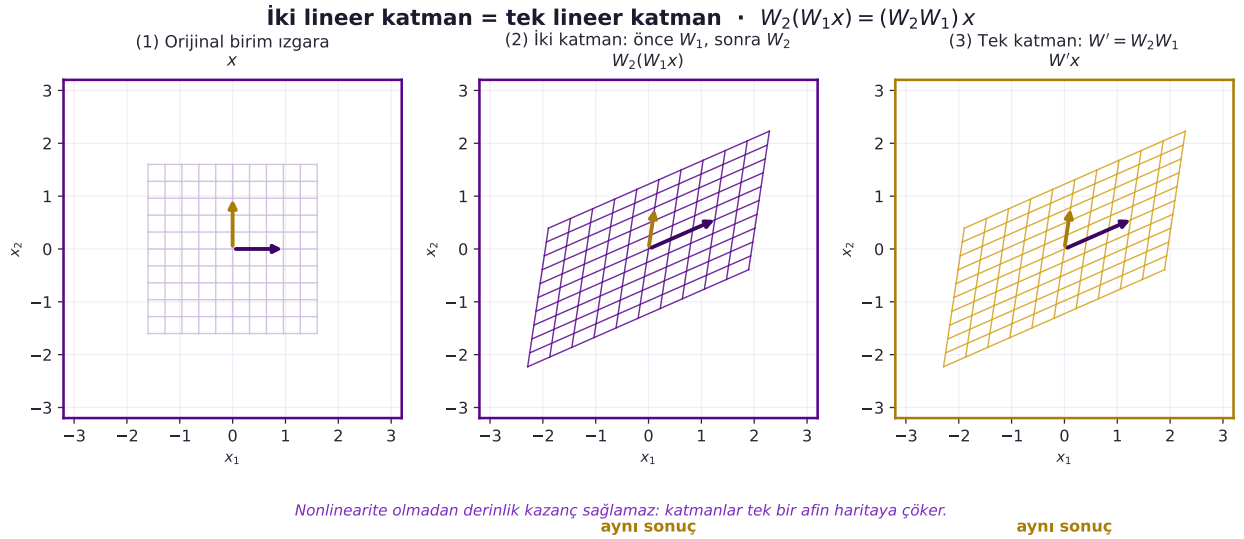
$$W_2\,(W_1\mathbf{x}) = (W_2 W_1)\,\mathbf{x} = W'\mathbf{x}$$

> "if you have two successive modules and they're both linear, you can collapse them into a single linear… there's no point having multiple layers if those layers are linear." — LeCun, 28:48

Yani derinlik ancak katmanlar arasına doğrusal-olmama girerse anlam kazanır. **Bu noktayı aklında tut** — Canziani aynı fikri Practicum'da uzay-germe animasyonuyla *gösterecek* (Bölüm 9). İki hoca, aynı gerçeği iki dilde anlatıyor: LeCun cebirle, Canziani geometriyle. Şekil 8.3 bunu birim ızgara üzerinde gösterir: önce W_1 sonra W_2 uygulamak ile tek matris $W' = W_2 W_1$ uygulamak aynı sonucu verir.

Builder Notu — Doğrusal Çöküş ve Uçtan Uca

Geriye (18.06 + Phase 1 DL): "Doğrusal ∘ doğrusal = doğrusal" tam olarak 18.06'nın bileşke lineer dönüşüm kuralıdır. Phase 1 DL Ders 1 de aynı çöküşü göstermişti — NYU bunu hem LeCun hem Canziani ile pekiştiriyor.
İleriye: Uçtan uca öğrenme (end-to-end), modern sistemlerin varsayılanıdır: ham pikselden/token'dan çıktıya tek bir türevlenebilir hat. "Modül + nonlinearity + istifle" reçetesi, bütün mimarilerin (CNN, transformer, GNN) ortak iskeletidir.



Şekil 8.3: İki ardışık lineer katmanın tek bir matrise çökmesi: $W_2(W_1x)$ ile $(W_2W_1)x$ aynı dönüşümü verir, bu yüzden nonlinearite olmadan derinlik kazanç sağlamaz.

8.6 (LeCun) Optimizasyon, “Kimse Anlamıyor” ve Enerji-Tabanlı Modeller

LeCun derin öğrenmenin nasıl çalıştığına geçiyor: öğrenme neredeyse her zaman **optimizasyon**dur, derin öğrenme ise neredeyse her zaman **gradient-tabanlı** optimizasyondur. Convex (dışbükey) durumda optimizasyon kuralları iyi anlaşılmıştır; ama derin öğrenmede maliyet fonksiyonu **convex değildir** — yerel minimumları, eyer noktaları vardır. Bu yüzden hedef fonksiyonun geometrisini anlamak önemlidir. Sonra LeCun’un en sevdiği itiraf geliyor:

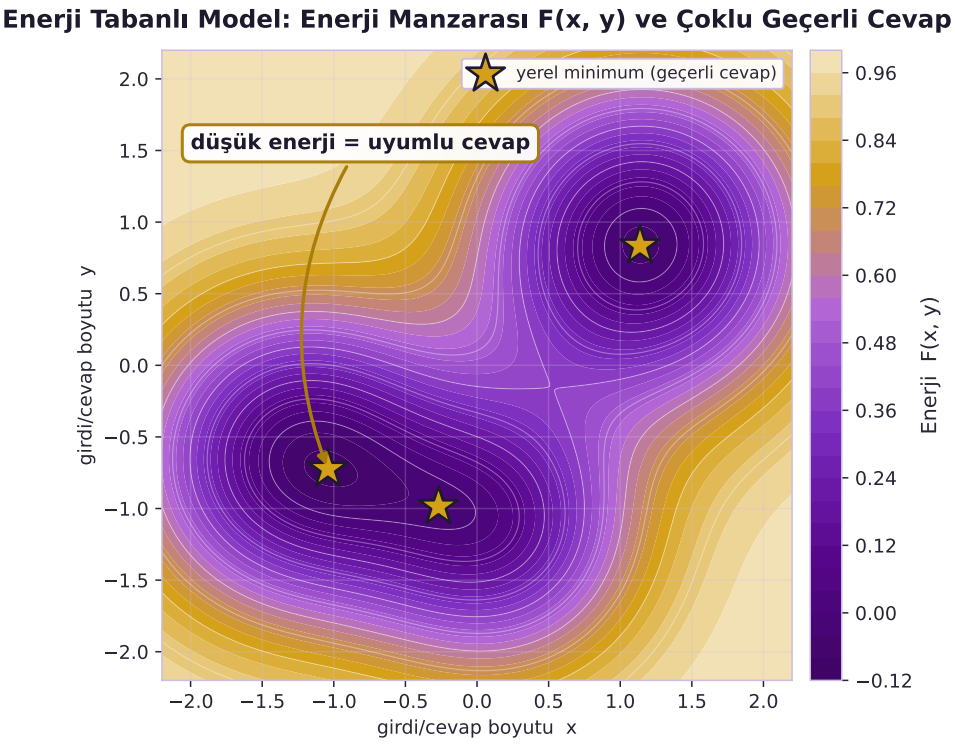
> “it’s important to understand the geometry of the objective function… but the big secret here is that nobody actually understands. So it’s important to understand that nobody understands.” — LeCun, 7:02

Yani alan, işe yarayan ama tam teorik temeli olmayan **sezgi + biraz teori + bol ampirik arayış** karışımı tricklerle ilerliyor: initialization, normalization, regularization (dropout), gradient clipping, momentum, ve egzotik bir konu olan **Lagrangian backprop** (Hafta 14’te döner).

Sonra LeCun kursun omurgasını açıyor — en sevdiği konu, **enerji-tabanlı modeller (EBM)**:

> “energy based models — this is sort of a general formulation of a lot of different approaches to learning, whether they are supervised, unsupervised, self-supervised.” — LeCun, 7:50

Fikir şu: bir sinir ağını “girdiyi çıktıya çeviren fonksiyon” olarak görmek kısıtlayıcıdır — bir girdiye yalnızca tek çıktı üretir. Oysa çoğu problemde bir girdiye **birden çok geçerli cevap** vardır. EBM bunu çözer: cevapları bir **enerji fonksiyonunun minimumları** yap; çıkarım (inference) bu enerjiyi minimize eden değerleri *arar*. Bu, sinir ağlarıyla “akıl yürütme”yi (reasoning) modellemenin bir yoludur.



Şekil 8.4: Enerji tabanlı modelin F(x, y) enerji manzarası: koyu violet kuyular düşük enerjili yerel minimumları (gold yıldız) işaretler ve çoklu minimum bir girdiye birden çok geçerli cevabın karşılığıdır.

Builder Notu — Enerji, Olasılık ve JEPA Köprüsü

Geriye (Stat 110 + Calculus): EBM'nin enerji → olasılık köprüsü Boltzmann dağılımıdır ($P(y) \propto \exp(-\beta E)$); energy = −log p (Stat 110). "Non-convex geometri" ise Calculus ikinci türev/Hessian dünyasıdır. Bu kavramlar Hafta 7'de derinleşecek (Yazım Kılavuzu §4.J kurs terimleri).
İleriye: EBM, LeCun'un bugünkü araştırma programının (JEPA, I-JEPA, V-JEPA — post-2020 ileriye köprü) tohumu. "Çıkarım = enerji minimizasyonu" fikri, modern dünya-modeli ve planlama yaklaşımlarının temelidir.

8.7 Geçiş: LeCun'dan Canziani'ye

LeCun büyük resmi çizdi — derin öğrenme temsili veriden öğrenir, katmanlar doğrusal-olmamayla anlam kazanır, ve tüm bunlar bir optimizasyon problemidir. Şimdi **Alfredo Canziani** Practicum'da sahneye çıkıyor ve aynı fikirleri **somut lineer cebire** indiriyor. LeCun'un soyut "temsil öğrenme"si, Canziani'nin elinde gözle görülür bir şeye dönüşüyor: **uzayda noktaları hareket ettirmek.** Üslup da değişiyor — Canziani interaktif, esprili, İtalyan:

> "I have a very strong Italian accent, you know — 'mamma mia!'… if you do not understand something, it's almost 99.9% my fault." — Canziani, 0:33

8.8 (Canziani) Sınıflandırma: Görüntü = Yüksek-Boyutlu Uzayda Bir Nokta

Canziani sınıflandırmayı (classification) geometriyle anlatıyor. Bir megapiksellik (1000 × 1000) renkli bir fotoğraf düşün: her piksel RGB olduğundan toplam 1000 × 1000 × 3 = **3 milyon sayı**. Yani bu görüntü, **3 milyon boyutlu bir uzayda tek bir nokta**dır.

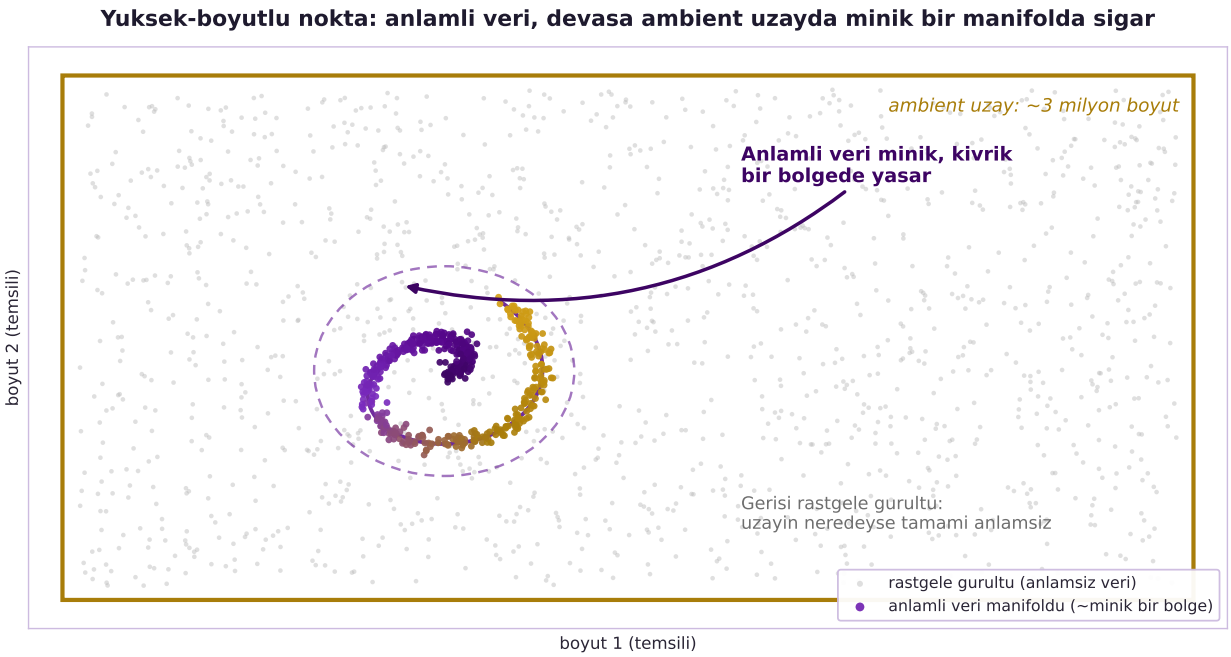
Bu uzay akıl almaz büyüklükte. İçinde rastgele dolaşırsan hiçbir anlamlı şey görmezsin. Bir köpek fotoğrafı bir noktada, bir kedi fotoğrafı başka bir noktada durur — ve Canziani sınıfa soruyor: kedi noktası köpek noktasına yakın mı, uzak mı? Cevap: **çok yakın.**

> "everything that actually makes sense is here (closes hand) — everything else is just trash." — Canziani, 5:45

Yani anlamlı görüntüler (kediler, köpekler, gerçek sahneler) bu devasa uzayın **minik bir bölgesinde** toplanmıştır; geri kalan her yer rastgele gürültüdür. Sınıflandırma problemi şudur: bu sıkışık noktaları al, uzayda **ayrıştırılabilir** hale gelecek biçimde hareket ettir, sonra aralarına bir sınır çiz.

Builder Notu — Vektör Uzayı ve Manifold

Geriye (18.06 + Stat 110): "3 milyon boyutlu uzayda nokta" doğrudan 18.06'nın vektör uzayıdır; "anlamlı veri minik bir bölgede" gözlemi ise **manifold hipotezi**nin (Bölüm 7) sezgisel ifadesidir. Yüksek boyutta noktaların dağılımı Stat 110 çok-değişkenli dağılımla bağlanır.
İleriye: "Veri, ambient uzayın küçük bir manifoldunda yaşar" fikri; boyut indirgeme, autoencoder (Hafta 7), ve üretken modellerin (VAE/GAN, Hafta 8-9) hepsinin dayandığı temel varsayımdır.



Şekil 8.5: Yuksek-boyutlu nokta: devasa ambient uzayda (~3 milyon boyut) anlamli veri yalnizca minik, kivrik bir manifolda sigarken uzayin geri kalani rastgele gurultudur.

8.9 (Canziani) Manifold Sezgisi: Uzayın Kumaşını Germek

Canziani spiral demosunu açıyor: bir spiralin beş kolu, her kol farklı renkte (sınıf). Girdi sadece (x, y) koordinatları — renk yok. Ağdan istenen: noktaları renge göre ayırmak. Ekrandaki animasyonda ağ, uzayı *fiziksel olarak gerip büküyor*:

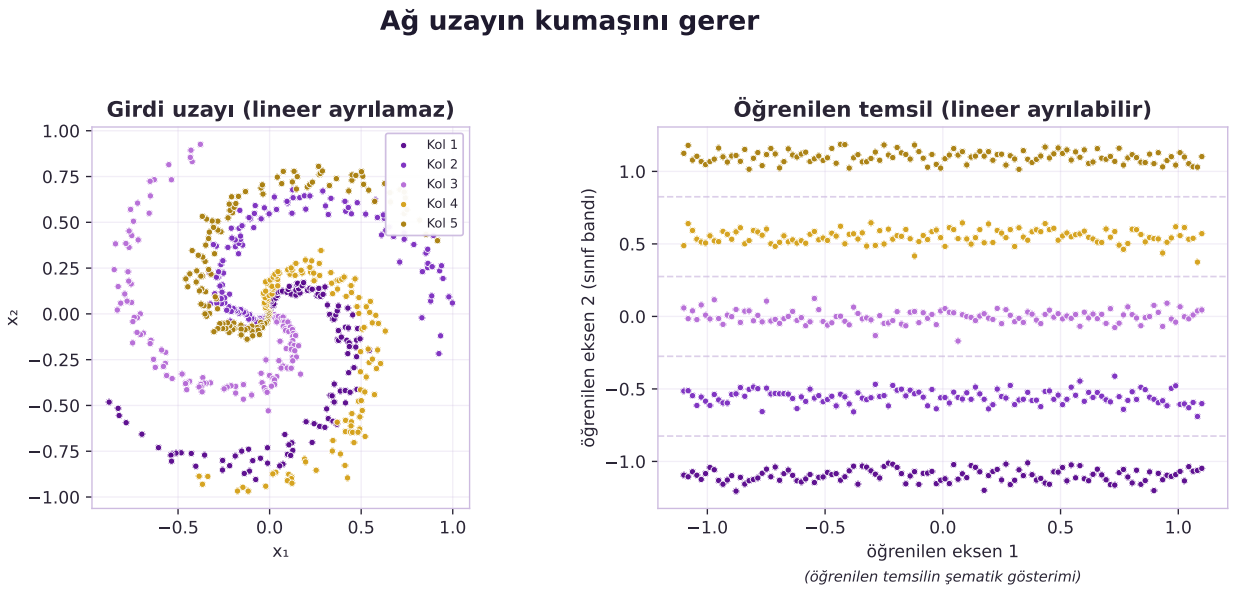
> "it takes the space… and it performs like a stretching of the space fabric." — Canziani, 15:11

Ağ, uzayın kumaşını öyle bir geriyor ki sonunda aynı renkten noktalar aynı alt-bölgede toplanıyor — yani **lineer olarak ayrılabilir** hale geliyorlar. O noktada basit bir lojistik regresyon (düz bir sınır) işi bitiriyor. Bu, LeCun'un "temsili öğrenmek" dediği şeyin gözle görülür hâlidir: iyi bir temsil, problemi kolay (ayrılabilir) yapan bir uzaydır.

Şekil 8.6 solda iç içe geçmiş 5-kollu spirali (lineer ayrılamaz), sağda ağın açtığı paralel sınıf bantlarını (lineer ayrılabilir) yan yana koyar. Bu, **manifold hipotezi**nin canlı gösterimi: veri, yüksek-boyutlu uzayda kıvrılmış düşük-boyutlu bir yüzey (manifold) üzerinde yaşar; öğrenme, bu manifoldu açıp düzleştirmektir. (LeCun de dersini manifold ile kapatır: bir çembere topolojik olarak denk, kıvrık bir manifoldu PCA gibi doğrusal bir yöntem **bulamaz** — çünkü düz değildir.)

Builder Notu — Germe = Lineer Dönüşüm + Bükme

Geriye (18.06): "Uzayı germek/bükmek" bir dizi **lineer dönüşüm** (matris) + doğrusal-olmamadır (Bölüm 8). PCA'nin kıvrık manifoldu bulamaması, doğrusal yöntemlerin sınırını gösterir — 18.06 SVD'nin (Ders 29) neden tek başına yetmediğinin sezgisi.
İleriye: Bu "açıp düzleştirme" görüşü; representation learning, t-SNE/UMAP görselleştirme ve autoen-



Şekil 8.6: 5-kollu spiral girdi uzayı (solda, lineer ayrılamaz) ile ağın öğrendiği temsil (sağda, paralel sınıf bantlarına açılmış, lineer ayrılabilir) yan yana — ağın manifoldu nasıl 'gerdiğini' gösteren şematik karşılaştırma.

coder'ların ortak dilidir.

8.10 (Canziani) Lineer Dönüşümler ve SVD

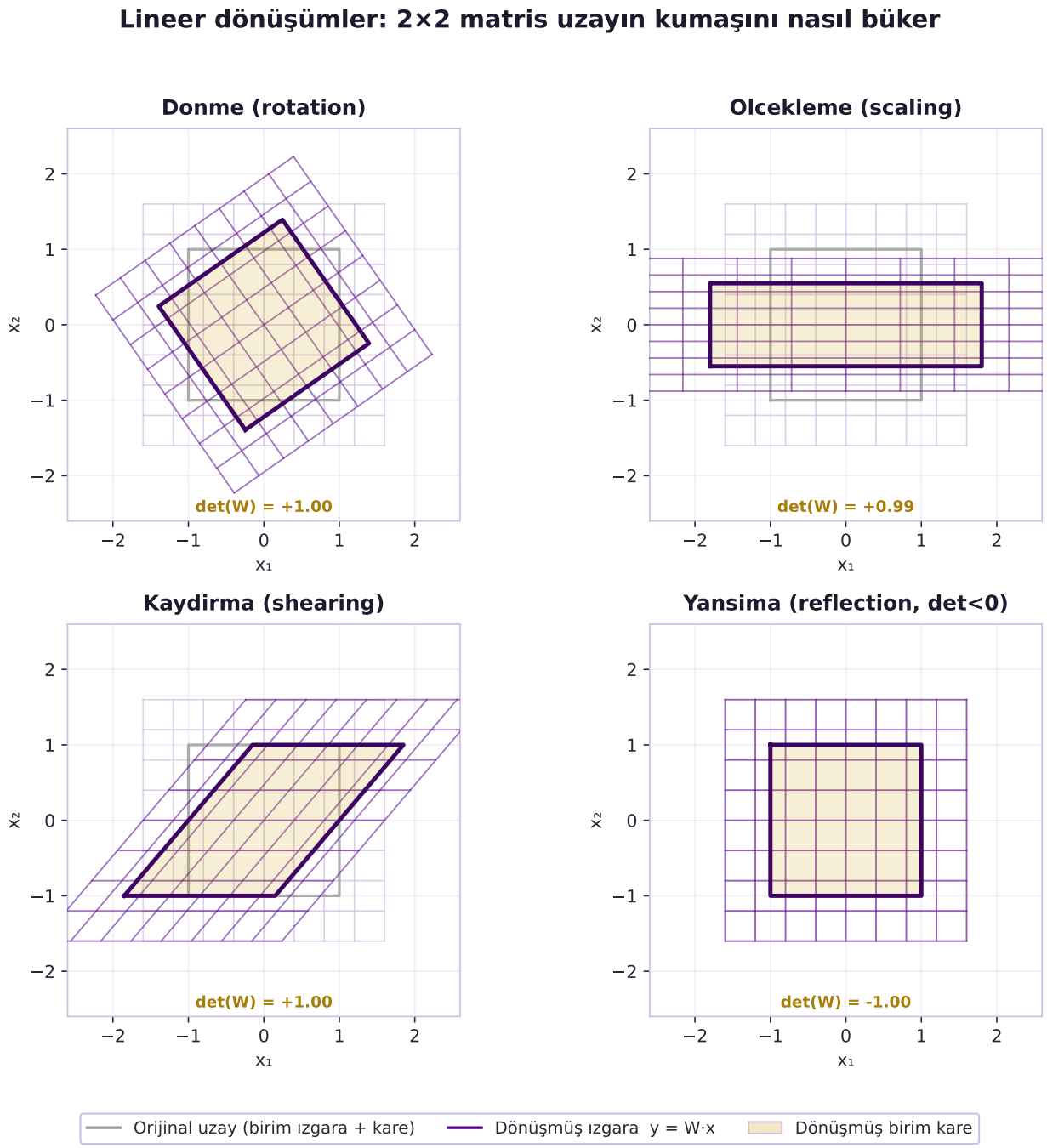
Peki uzayda noktaları nasıl hareket ettiririz? Canziani lineer cebiri interaktif soruyor: matris çarpımı ne yapar? Cevap — bir **lineer dönüşüm**. Dört temel tip var (Şekil 8.7):

1. **Rotation (döndürme)** — matris ortonormalse.
2. **Scaling / stretching (ölçekleme)** — köşegen matrisle.
3. **Shearing (kaydırma)** — eksenleri eğer.
4. **Reflection (yansıma)** — determinant negatifse.

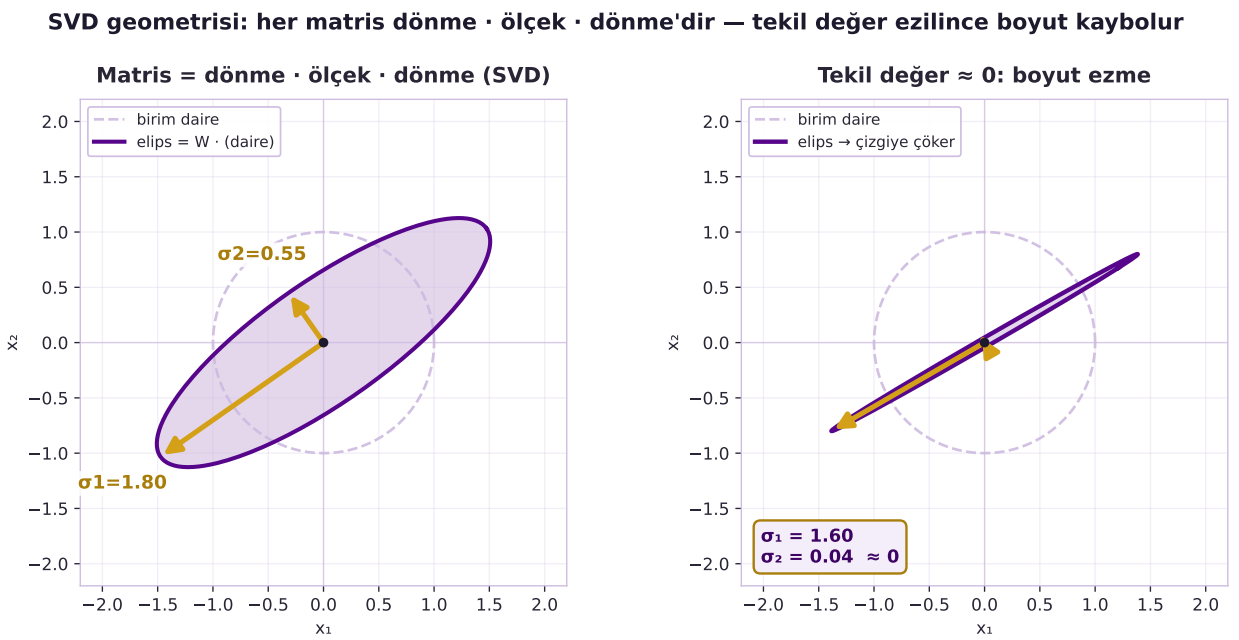
Bir uyarı: **öteleme (translation) lineer DEĞİLDİR.** Canziani'nin testi: bir dönüşüm lineerse 0'ı 0'a götürmeli; öteleme bunu yapmaz. Öteleme eklersek dönüşüm **afin (affine)** olur. Bir sinir ağı katmanı tam olarak budur — matris çarpımı (lineer) + öteleme (bias):

$$\mathbf{y} = W\mathbf{x} + \mathbf{b}$$

Noktaları ayırmak için Canziani'nin reçetesi: önce öteleme ile aşağı indir, sonra köşegen matrisle (uygun **tekil değerlerle**, singular values) ger. Matrisi köşegenleştirmek için araç **SVD (singular value decomposition)**: bir matris = rotation · scaling · rotation. Tekil değerlerden biri sıfıra yakınsa matris bir boyutu neredeyse "öldürür" (ezme). Şekil 8.8 bunu gösterir: solda birim daire bir elipse ("patatese") döner (ana eksenler = tekil değerler), sağda bir tekil değer sıfıra yaklaşınca elips bir çizgiye çöker.



Şekil 8.7: Dört temel lineer dönüşümün (donme, olcekleme, kaydirma, yansima) birim ızgara ve birim kare üzerindeki geometrik etkisi; determinant işareti yansımanın yönelimi tersine çevirdiğini gösterir.



Şekil 8.8: Birim dairenin bir matrisle çarpılınca bir elipse (‘patates’) dönüşmesi ve SVD’nin geometrik okuması: her matris bir dönme·ölçek·dönme bileşimidir; sağ panelde bir tekil değer sıfıra yaklaşınca elips bir çizgiye çöker ve bir boyut ezilir.

Builder Notu — SVD, Düşük-Rank ve LoRA

Geriye (18.06 + 18.065): Lineer dönüşüm tipleri 18.06’nın çekirdeğidir; SVD = Ders 29. “Bir matris = rotation · scaling · rotation” tam olarak SVD’nin geometrik okumasıdır. Tekil değer ≈ 0 → matris tekil (singular), boyut kaybı. Bu, Phase 2 18.065’in (Matrix Methods) ML-uygulamalı tam konusudur — atıf: (Phase 1 18.06 Ders 29 + Phase 2 18.065 ileri).
İleriye: SVD ve düşük-rank yaklaşım; PCA, model sıkıştırma, ve LoRA (düşük-rank fine-tune) gibi tekniklerin matematiksel temelidir.

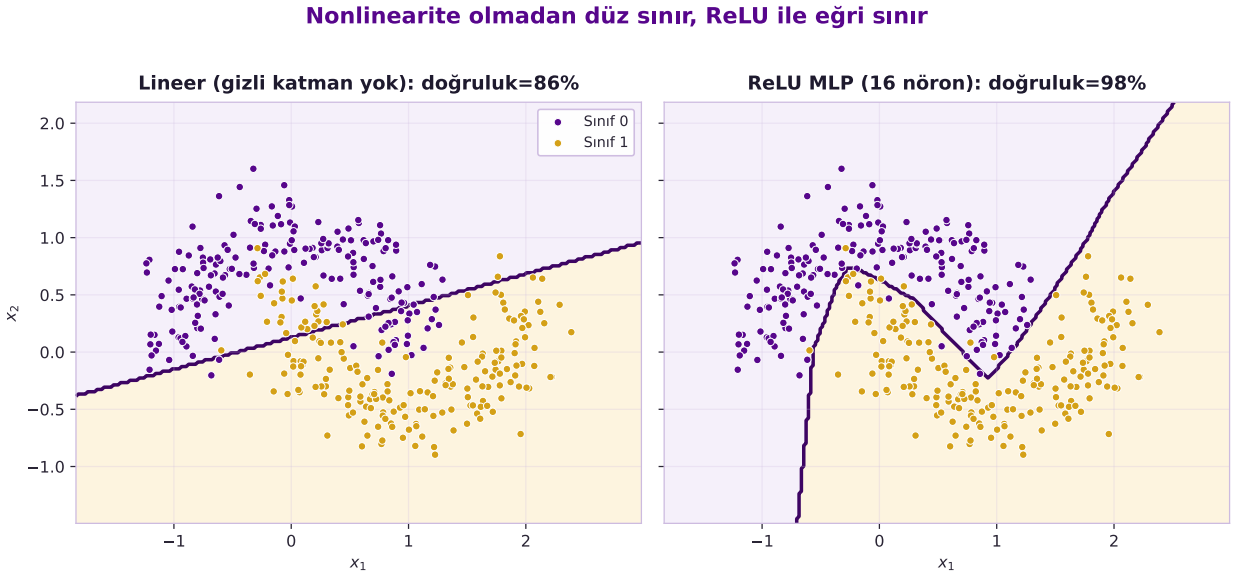
8.11 (Canziani) Neden Doğrusal-Olmama? (LeCun ile Buluşma)

Canziani ağı iki matrisli bir örnekle kuruyor: girdi 2 boyut → ara katman 100 boyut (bir nonlinearity) → çıktı. Sonra kritik soruyu soruyor: neden birden çok matris, neden doğrusal-olmama? Cevabı, LeCun’un cebirle söylediğiyle **birebir aynı** — ama bu kez geometrik:

> “without nonlinearity it would look like a single layer neural network — and a single layer neural network, what can it do? Scaling, translation, rotation, reflection, shearing.” — Canziani, 20:21

Yani doğrusal-olmama olmadan, kaç matris istiflersen istifle, ağ yalnızca o dört-beş temel dönüşümü (Bölüm 8) yapabilir; hepsi tek bir matrise çöker. Spiral noktalarını ancak doğru bir sınırla ayıramazsın — uzayı *bükmen* gerekir, ve bükme yalnızca doğrusal-olmamadan gelir. İşte LeCun’un “linear ∘ linear = linear” cebri ile Canziani’nin “tek katman sadece germe/döndürme yapar” geometrisi aynı madalyonun iki yüzü. Şekil 8.9

bu farkı somutlaştırır: aynı iki-hilal verisinde lineer sınıflandırıcı düz bir sınırla takılırken, ReLU'lu ağ eğri bir sınır öğrenip sınıfları ayırır.



Şekil 8.9: Aynı iki-hilal verisinde lineer sınıflandırıcı düz bir karar sınırı üretip hilalleri ayıramazken (doğruluk %86), 16 nöronlu ReLU MLP eğri bir sınır öğrenip sınıfları ayırır (doğruluk %98) — nonlineeritenin neden gerekli olduğunu gösterir.

> **Builder Notu — İki Hocanın Aynı Teoremi**
>
> **Geriye (Bölüm 4 + 18.06):** Bu, dersin en güçlü çapraz-doğrulamasıdır: aynı teorem (doğrusal katmanların çökmesi) hem LeCun (Lecture, cebir) hem Canziani (Practicum, geometri) tarafından bağımsız anlatılıyor. İki hoca = iki kanıt yolu.
> **İleriye:** ReLU/GELU gibi doğrusal-olmamaların seçimi, derin ağların ifade gücünü ve eğitim kararlılığını belirler — modern mimarilerin temel tasarım kararı.

8.12 (Canziani) PyTorch İlk Dokunuş: nn.Linear, ReLU ve Rastgele Projeksiyonlar

Canziani teoriyi koda döküyor. PyTorch'ta bir katman `nn.Linear` (matris çarpımı + bias = afine); doğrusal-olmama `nn.ReLU` (pozitif kısım) veya `tanh`. Eğitilmemiş bir ağ bile uzayı ilginç biçimde büker:

```
import torch
import torch.nn as nn

# bir sinir agi = afine donusum + dogrusal-olmama
model = nn.Sequential(
    nn.Linear(2, 5),    # 2 -> 5: matris + bias (affine)
    nn.ReLU(),          # pozitif kisim: dogrusal-olmama
    nn.Linear(5, 2),    # 5 -> 2: ekranda gosterebilmek icin
)
```

```
x = torch.randn(1000, 2)   # 1000 nokta, standart normal bulut
y = model(x)               # uzayi ger, bukle, yansit
```

Canziani'nin **rastgele projeksiyon** notebook'u tam da bunu görselleştirir: 1000 noktalık dairesel bir bulutu bir matrisle çarpınca bulut bir "patatese" (elips) döner; tekil değerlerden biri ≈ 0 ise bir boyut ezilir. Üstüne `tanh` koyunca (kink ≈ ±2.5) veri bir kutuya sıkışır; `ReLU` koyunca köşeli, ayrıştırılabilir kümeler oluşur. Canziani'nin uyarısı: bu sihir değil, sadece görselleştirme —

> "this is just visualization using matplotlib… this is not magic." — Canziani, 23:18

Son olarak donanım: tensörler CPU yerine GPU belleğinde tutulursa paralel hesap çok hızlanır. Tek satır yeter:

```
# CUDA varsa tensorleri GPU bellegine koy, yoksa CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Canziani'nin esprili özeti: CPU tek seferde bir şey yapar (çok hızlı ama sıralı); GPU çok daha yavaş ama aynı anda binlerce işlem yapar — bu yüzden büyük matris işlerinde kazanır.

> Builder Notu — nn.Linear, Device ve Init
>
> **Geriye (18.06):** `nn.Linear(2, 5)` tam olarak 5×2 boyutlu bir matris + 5-boyutlu bias; rastgele projeksiyon = bir matrisle çarpmanın geometrik etkisi (SVD ile okunur, Bölüm 8).
> **İleriye:** "Tensörleri doğru cihaza (device) koy" tek satırı, tüm modern eğitim hattının (GPU/TPU throughput, mixed precision, DDP) giriş kapısıdır. Eğitilmemiş ağın bile yapı kurması, **weight initialization**'ın neden önemli olduğunu önceler (Hafta 4'te BatchNorm/init).

8.13 Özet

1. **Derin öğrenme = temsili veriden öğrenmek.** Geleneksel örüntü tanıma elle tasarlanmış öznitelik çıkarıcısı + sınıflandırıcıdır; derin öğrenme bu iki aşamayı uçtan uca, öğrenilen modüllerle değiştirir.
2. **"Derin" = çok katmanlı.** Her katman ayarlanabilir parametre + doğrusal-olmama içerir; başka gizem yok.
3. **Doğrusal-olmama şart** — iki ardışık doğrusal katman tek bir doğrusala çöker (LeCun cebirle, Canziani geometriyle gösterir).
4. **Tarih neden önemli:** perceptron (Rosenblatt, fiziksel analog), 1969-1984 kışı, 1985 backprop (sürekli nöron + hızlanan çarpma), 2010 konuşma + 2012 görü dirilişi.
5. **Optimizasyon non-convex** ve "kimse tam anlamıyor"; alan tricklerle (init, norm, dropout) ilerler.
6. **EBM** (LeCun'un en sevdiği konu, kursun omurgası): cevaplar bir enerji fonksiyonunun minimumlarıdır; çıkarım = enerji minimizasyonu.
7. **Geometri (Canziani):** görüntü = yüksek-boyutlu uzayda nokta; anlamlı veri minik bir manifoldda yaşar; ağ bu uzayın kumaşını gerip noktaları ayrıştırılabilir yapar.
8. **Lineer cebir:** matris çarpımı = rotation/scaling/shearing/reflection; öteleme afin yapar (Wx + b); SVD = rotation · scaling · rotation.

! Tek Bir Cümle

Derin öğrenme, elle tasarlanan öznitelik çıkarıcısını çöpe atıp temsili doğrudan veriden öğrenir; geometrik olarak bu, veri uzayını doğrusal dönüşümler (Wx + b) ve doğrusal-olmamalarla yeniden şekillendirip, kıvrık veri manifoldunu ayrıştırılabilir hale getirmektir.

8.14 Kontrol Soruları

i Soru 1: Bir ağda tüm doğrusal-olmamaları kaldırırsan (her katman sadece matris çarpımı olursa) ne olur? Neden işe yaramaz?

Cevap: Ağ ardışık matris çarpımlarına iner ve hepsi tek bir matrise çöker. İki katman için:

$$W_2\,(W_1\mathbf{x}) = (W_2W_1)\,\mathbf{x} = W'\mathbf{x}$$

Kaç katman istiflersen istifle, sonuç tek bir doğrusal dönüşüm (W x) kalır — yalnızca rotation/scaling/shearing/reflection yapabilir, veriyi *bükemez*. Spiral gibi doğrusal ayrılamayan veriyi asla ayıramaz. LeCun bunu cebirle ("composition of two linear functions is linear", 28:48), Canziani geometriyle ("single layer only scales/rotates", 20:21) söyler. Derinliği anlamlı kılan tek şey katmanlar arasındaki doğrusal-olmamadır.

i Soru 2: Bir megapiksellik (1000 × 1000) RGB görüntü kaç boyutlu uzayda bir noktadır? 'Manifold hipotezi' bu uzay hakkında ne söyler?

Cevap: 1000 × 1000 × 3 = **3.000.000 boyut**. Görüntü, bu 3 milyon boyutlu uzayda tek bir noktadır. Manifold hipotezi: anlamlı görüntüler (kediler, sahneler) bu devasa uzayın **minik, düşük-boyutlu, kıvrık bir alt-yüzeyinde (manifold)** toplanmıştır; geri kalan neredeyse her nokta rastgele gürültüdür ("everything else is just trash", Canziani 5:45). Öğrenme, bu manifoldu açıp ayrıştırılabilir hale getirmektir. Doğrusal bir yöntem (PCA) kıvrık manifoldu bulamaz — LeCun'un kapanış örneği.

i Soru 3: Öteleme (translation) neden bir lineer dönüşüm değildir? Sinir ağı katmanını "afin" yapan nedir?

Cevap: Bir dönüşümün lineer olması için sıfırı sıfıra götürmesi gerekir (T(0) = 0). Öteleme sıfırı b'ye taşır, dolayısıyla lineer değildir (Canziani'nin testi, 10:10). Matris çarpımına (lineer) bir öteleme (bias) eklersek dönüşüm **afin (affine)** olur — bir sinir ağı katmanı tam olarak budur:

$$\mathbf{y} = W\mathbf{x} + \mathbf{b}$$

Yani `nn.Linear(..., bias=True)` afin, `bias=False` ise saf lineer (matris) dönüşümdür.

> Soru 4: (Builder) EBM'de 'çıkarım = enerji minimizasyonu', sıradan bir ağın 'girdi → çıktı' çıkarımından nasıl farklıdır? Stat 110 ile bağla.
>
> **Cevap:** Sıradan bir ağ bir girdiye **tek** çıktı üretir (ileri geçiş). EBM ise bir enerji fonksiyonu F(x, y) tanımlar ve çıkarımı bir **arama/optimizasyon** yapar: verilen x için enerjiyi minimize eden y'yi (veya y'leri) bulur. Birden çok minimum varsa, bir girdiye **birden çok geçerli cevap** verebilir — LeCun'un "akıl yürütme" dediği budur (7:50). Stat 110 köprüsü: enerji ile olasılık, Boltzmann dağılımıyla bağlanır — $P(y) \propto \exp(-\beta E)$, yani energy = −log p (sabite kadar). Düşük enerji = yüksek olasılık. Bu, Hafta 7'nin (EBM) çekirdeğidir.

8.15 Egzersizler

Egzersiz 1 (Perceptron'u elle kur). Tek bir perceptron'un ileri geçişini NumPy ile yaz (nokta çarpımı + bias + aktivasyon), sonra `nn.Linear` + aktivasyonla aynı sonucu al.

```
import numpy as np

def perceptron(x, w, b, g=lambda z: 1/(1+np.exp(-z))):
    z = np.dot(x, w) + b      # nokta carpimi + bias
    return g(z)               # aktivasyon

x = np.array([3.0, 1.0]); w = np.array([1.0, -1.0]); b = 0.0
print(perceptron(x, w, b))    # ~0.88
```

Egzersiz 2 (Doğrusal çöküşü göster). Aktivasyonsuz iki `nn.Linear` katmanı zincirle. Bunların tek bir matrise ($W_2 W_1$) eşit olduğunu sayısal olarak doğrula — derinliğin aktivasyonsuz anlamsız olduğunu kendin gör.

```
import torch, torch.nn as nn
W1 = nn.Linear(2, 5, bias=False); W2 = nn.Linear(5, 2, bias=False)
x = torch.randn(10, 2)
chained = W2(W1(x))
collapsed = x @ W1.weight.T @ W2.weight.T   # tek matris (W2 W1)
print(torch.allclose(chained, collapsed, atol=1e-5))  # True
```

Egzersiz 3 (Rastgele projeksiyon + SVD). 1000 noktalık standart normal bir bulut üret, rastgele bir 2×2 matrisle çarp, `torch.linalg.svd` ile tekil değerleri incele. Bir tekil değeri çok küçük yapıp bir boyutun nasıl "ezildiğini" çiz (Canziani'nin patatesi).

Egzersiz 4 (Doğrusal-olmama neden şart). `sklearn.datasets.make_moons` ile doğrusal ayrılamayan veri üret. (a) Aktivasyonsuz, (b) `nn.ReLU`'lu bir ağ kur; karar sınırlarını çiz. Aktivasyonsuz ağın neden yalnızca düz çizgi çizebildiğini gözlemle.

Egzersiz 5 (Hafta 2 habercisi — gradient descent + backprop). Bu hafta ağ kurduk ama *eğitmedik*. Basit bir kuadratik kayıp $L(w) = (w - 3)^2$ üzerinde gradient descent'i elle uygula (gradient = $2(w - 3)$). Sonra `x → nöron → ŷ → L` minik ağı için $\partial L/\partial w$'yi zincir kuralıyla yaz. Bu iki gözlem, Hafta 2'de **backpropagation**'a (LeCun: backprop = zincir kuralının katmanlara uygulanması) neden ihtiyaç duyduğumuzu motive eder.

```
def gd(eta, steps=50, w0=10.0):
    w = w0
    for _ in range(steps):
        grad = 2*(w - 3)      # dL/dw
        w = w - eta*grad      # gradient descent adimi
    return w
for eta in [0.01, 0.1, 0.5, 1.0]:
    print(eta, round(gd(eta), 4))
```

8.16 Sonraki Ders İçin Hazırlık

> ⚠ Sonraki Hafta — H2: Gradient Descent ve Backprop
>
> **Manifold geometrisinden optimizasyon dinamiğine.** Bu hafta ağı *kurduk*; Hafta 2 onu **eğitir**: gradient descent, learning rate, loss landscape (kayıp manzarası) ve backpropagation = zincir kuralının katmanlara uygulanması. Bu haftaki spiral noktalarını gerçekten ayıracağız.

Hafta 2: Gradient Descent, Backprop ve Yapay Sinir Ağları — LeCun (Lecture) + Canziani (Practicum)

Bu hafta ağı *kurduk* ama nasıl *öğrendiğini* görmedik. Hafta 2'de LeCun gradient descent ve backpropagation'ı (zincir kuralının katmanlara uygulanması) anlatacak; Canziani PyTorch'ta sinir ağı eğitimini gösterecek — bu haftaki spiral noktalarını gerçekten ayıracağız.

Hafta 2 öncesi yapılacak:

- Egzersiz 2'yi (doğrusal çöküş) ve Egzersiz 5'i (gradient descent) çöz.
- "İki ardışık doğrusal katman tek katmana çöker" cümlesini kendi sözcüklerinle yaz (hem cebir hem geometri).
- Calculus zincir kuralını (Phase 1 Calculus Ders 4) gözden geçir — backprop tam olarak budur.

8.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Representation learning	Temsili elle tasarlamak yerine veriden öğrenmek	LeCun
Feature extractor	Klasik örüntü tanımada elle tasarlanan öznitelik aşaması	LeCun 26m46
Perceptron	Ağırlıklı toplam + eşik; Rosenblatt'ın fiziksel analog makinesi	LeCun 17m18
Deep = çok katman	"Derin"in tek anlamı: birden çok katman	LeCun 28m14
Doğrusal çöküş	linear ∘ linear = linear; nonlinearity şart	LeCun 28m48 / Canziani 20m21

Kavram	Tanım	Hoca / timestamp
EBM	Cevaplar = enerji fonksiyonunun minimumları; çıkarım = minimize	LeCun 7m50
Yüksek-boyutlu nokta	1MP RGB görüntü = 3 milyon boyutlu uzayda bir nokta	Canziani 4m45
Manifold hipotezi	Anlamlı veri minik, kıvrık bir alt-manifoldda yaşar	Canziani 5m45
Uzay germe	Ağ = uzayın kumaşını gerip noktaları ayrıştırma	Canziani 15m11
Lineer dönüşüm	rotation / scaling / shearing / reflection (det < 0)	Canziani 8m14
Afin dönüşüm	Wx + b; öteleme lineer değil (0 0)	Canziani 10m10
SVD	Matris = rotation · scaling · rotation; tekil değer ≈ 0 → boyut ezme	Canziani 40m02

8.18 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Perceptron / afin katman (Wx + b)** → 18.06 matris-vektör çarpımı + Phase 1 DL Ders 1.
2. **Lineer dönüşüm tipleri + SVD** → 18.06 (Ders 29 SVD) + Phase 2 18.065 (Matrix Methods) ileri.
3. **EBM enerji → olasılık** → Stat 110 Boltzmann, energy = −log p.
4. **Backprop için sürekli/türevlenebilir nöron** → Calculus türev + zincir kuralı (Ders 4).
5. **Manifold hipotezi** → 18.06 altuzay + Stat 110 çok-değişkenli dağılım geometrisi.

İleriye köprüler (production / research):

1. **EBM** → JEPA / I-JEPA / V-JEPA (post-2020 ileriye köprü, LeCun'un bugünkü programı).
2. **Representation learning** → foundation models, transfer learning (Hafta 15 bonus).
3. **GPU device yerleşimi** → throughput, mixed precision, DDP.
4. **Donanım hızlanması alanı patlattı** → scaling laws.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Derin öğrenme sihir değildir — elle tasarlanan öznitelikleri çöpe atıp, veri uzayını doğrusal dönüşümler (Wx + b) ve doğrusal-olmamalarla gere büke yeniden şekillendirir; kıvrık veri manifoldunu ayrıştırılabilir hale getirir. LeCun bunu cebirle anlatır, Canziani geometriyle gösterir — aynı gerçeğin iki dili.

9 Gradient Descent, Backprop ve Yapay Sinir Ağları

NYU'nun iki hocalı ritmi: LeCun cebirle 'eğitmek = bir kaybı gradient descent ile küçültmek, gradient'i backprop = zincir kuralı hesaplar' der, Canziani geometriyle 'ağ döndür-ez dizisidir' diye gösterip Hafta 1'in spiral ağını gerçekten eğitir

> **i** Bölüm bilgisi
>
> - **LeCun'un Lecture videosu:** YouTube — Lecture 02: Backpropagation and architectural components (≈103 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Practicum 02: Training a neural network (≈57 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈30 dk

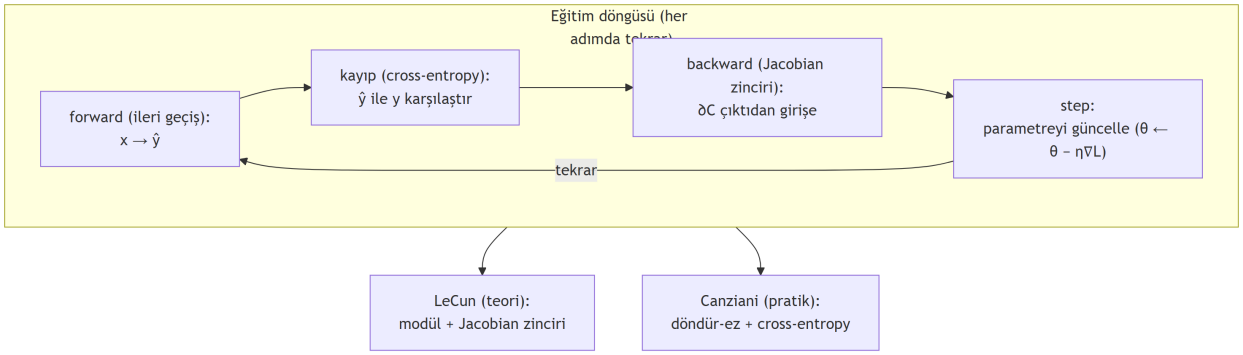
9.1 Bu Derste Ne Var?

Hafta 1'de bir ağ *kurduk* ama eğitmedik — spiral noktaları hâlâ ayrılmamıştı. Bu hafta ağa **nasıl öğretileceğini** görüyoruz. Yine iki hocalı: önce **Yann LeCun** (Lecture) öğrenmenin motorunu — **gradient descent** ve **backpropagation**'ı — teorik olarak kurar; sonra **Alfredo Canziani** (Practicum) bunu PyTorch'ta uygular ve Hafta 1'in spiral ağını gerçekten eğitir.

LeCun'un büyük fikri tek cümlede: bir sinir ağı, **modüllerin** (her biri girdi → çıktı hesaplayan bir kutu) bir zinciridir; eğitmek, bir **kayıp (loss)** fonksiyonunu gradient descent ile küçültmektir; ve gradient'i hesaplamanın yolu **backpropagation = zincir kuralının modüllere uygulanması**dır. Canziani ise bir ağın aslında "döndür-ez" (rotation–squashing) dizisi olduğunu gösterir, sınıflandırma için doğru kaybın **cross-entropy** olduğunu açıklar ve eğitim döngüsünü koda döker.

Bu haftanın üç ana fikri:

1. **Eğitim = bir kaybı gradient descent ile küçültmek.** Ağ rastgele başlar; kayıp, tahmin ile gerçek arasındaki farktır.
2. **Backpropagation = zincir kuralı.** Gradient, çıktıdan girişe doğru **Jacobian matrisleriyle çarpılarak** taşınır; daha fazlası değil.
3. **Sınıflandırmanın kaybı cross-entropy'dir** (−log softmax), regresyonunki MSE.



Builder Notu — Hafta 1'i Eğitmek: İki Yön

Geriye (önkoşul kurslar):

- **Backprop = zincir kuralı** → Phase 1 Calculus Ders 4 (zincir kuralı) + Karpathy micrograd (autograd'ı sıfırdan kurar — LeCun'un "module + Jacobian" çerçevesinin kod hâli).
- **Cross-entropy = –log olabilirlik** → Stat 110 Bernoulli/NLL + Phase 1 DL Ders 1 cross-entropy.
- **Gradient descent** → Phase 1 Calculus türev (Ders 2) + 18.06 gradient/Jacobian (matris dili).

İleriye (production / research):

- LeCun'un "module + Jacobian" çerçevesi tam olarak **PyTorch autograd**'ın iç mantığıdır (`loss.backward()`); ileride `torch.autograd`, gradient checkpointing.
- SGD'nin "noisy ama ucuz" doğası → modern büyük-ölçek eğitimin (mini-batch, gradient accumulation, DDP) temelidir.

Tek cümleyle: Bir ağı eğitmek, bir kaybı gradient descent ile küçültmektir; gradient'i de backpropagation — yani zincir kuralının modül-modül, Jacobian çarpımlarıyla uygulanması — hesaplar.

9.2 (LeCun) Modüller ve Maliyet Fonksiyonu

LeCun derse modüler bakışla başlıyor: bir sinir ağı, birbirine bağlı **modüllerden** kuruludur. İki tür modül var:

1. **Fonksiyonel modüller** — girdiyi çıktıya çeviren işlemler (linear, ReLU, …). LeCun bunları "mavi yuvarlak" kutular olarak çizer.
2. **Maliyet (cost) modülleri** — ağın en tepesine konur; tahmin $\hat{Y}$ ile gerçek Y'yi karşılaştırıp tek bir sayı (kayıp) üretir.

> "the C function compares Y and Y bar [Ŷ]… loss functions are things that we minimize." — LeCun, 1:43

Yani eğitim, bu maliyet modülünün çıktısını (kaybı) olabildiğince küçültmektir. Ağ rastgele başlatıldığından ilk tahminler kötüdür; kayıp büyüktür. Amaç, parametreleri kaybı düşürecek yönde ayarlamaktır. Bu modüler kurgu kursun geri kalanının da omurgasıdır: her yeni mimari (CNN, RNN, transformer), aynı "modülleri bağla, tepeye maliyet koy, eğit" iskeletine oturur.

Builder Notu — Her Şey Modül

Geriye (Karpathy): "Her şey modül" görüşü, Karpathy'nin micrograd'daki `Value` nesnesiyle birebir: her işlem bir düğüm, her düğüm ileri (forward) hesaplar ve geri (backward) gradient taşır. LeCun cebirle, Karpathy kodla aynı soyutlamayı kurar.
İleriye: Modüler tasarım, PyTorch'un `nn.Module` API'sinin felsefesidir: karmaşık modeller küçük, yeniden kullanılabilir, türevlenebilir bloklardan kurulur.

9.3 (LeCun) Loss ve Gradient-Tabanlı Öğrenme

Kaybı nasıl küçültürüz? **Gradient-tabanlı** yöntemlerle. Gradient, kaybın her parametreye göre değişim yönünü söyler; biz kaybı azaltmak için gradient'in **ters** yönünde küçük adımlar atarız. LeCun bunu dağ benzetmesiyle anlatıyor: sisli bir dağda en dik iniş yönünü bulup o yönde küçük bir adım atmak gibi — tüm manzarayı göremezsin, ama ayağının altındaki eğimi bilirsin.

> "gradient descent is like being in the mountain… you compute the gradient and take a small step downhill." — LeCun, 10:27

Şekil 9.1 bu "dağdan iniş" sezgisini somutlaştırır: yüksek-kayıp başlangıcından, kaybın ters yönünde küçük adımlarla vadinin dibine (minimuma) inilir.

Bir uyarı: öğrenme **neredeyse her zaman** gradient-tabanlı optimizasyondur, ama kayıp yüzeyi non-convex olduğundan (Hafta 1, "nobody understands") yakınsama garantisi yoktur — yine de pratikte çalışır. LeCun ayrıca türevlenemeyen durumlar için bir trick'ten söz eder: maliyet modülünü türevlenebilir bir modülle yaklaşık temsil edip yine backprop kullanmak.

Builder Notu — Dağda İniş (Gradient)

Geriye (Calculus): Gradient = çok değişkenli türev; gradient'in negatifi en dik iniş yönüdür (Phase 1 Calculus Ders 6). "Küçük adım" η ile ölçeklenir — learning rate.
İleriye: Gradient-tabanlı optimizasyon tüm derin öğrenmenin motorudur; adımın nasıl atılacağı (Adam, momentum, normalization) Hafta 4'ün konusu.

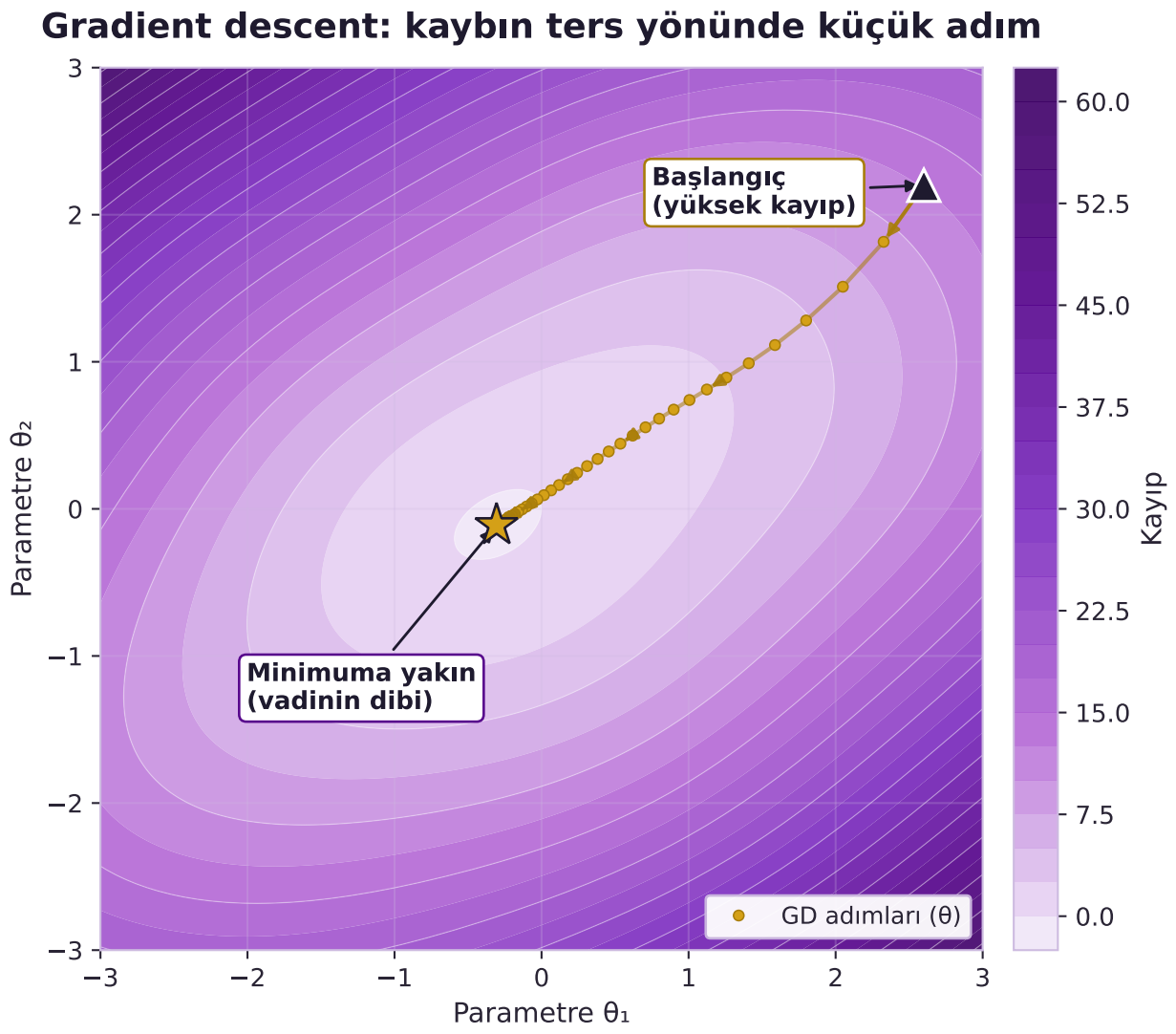
9.4 (LeCun) SGD: "Descent Değil, Optimization"

Tam gradient'i tüm veri üzerinde hesaplamak pahalıdır. **Stokastik gradient descent (SGD)** gradient'i tek bir (veya küçük bir batch) örnek üzerinde tahmin eder — çok daha hızlı, ama gürültülü. LeCun'un meşhur düzeltmesi:

> "it shouldn't be called stochastic gradient descent, because it's not actually a descent algorithm — it should be called stochastic gradient optimization. It's very noisy." — LeCun, 17:10

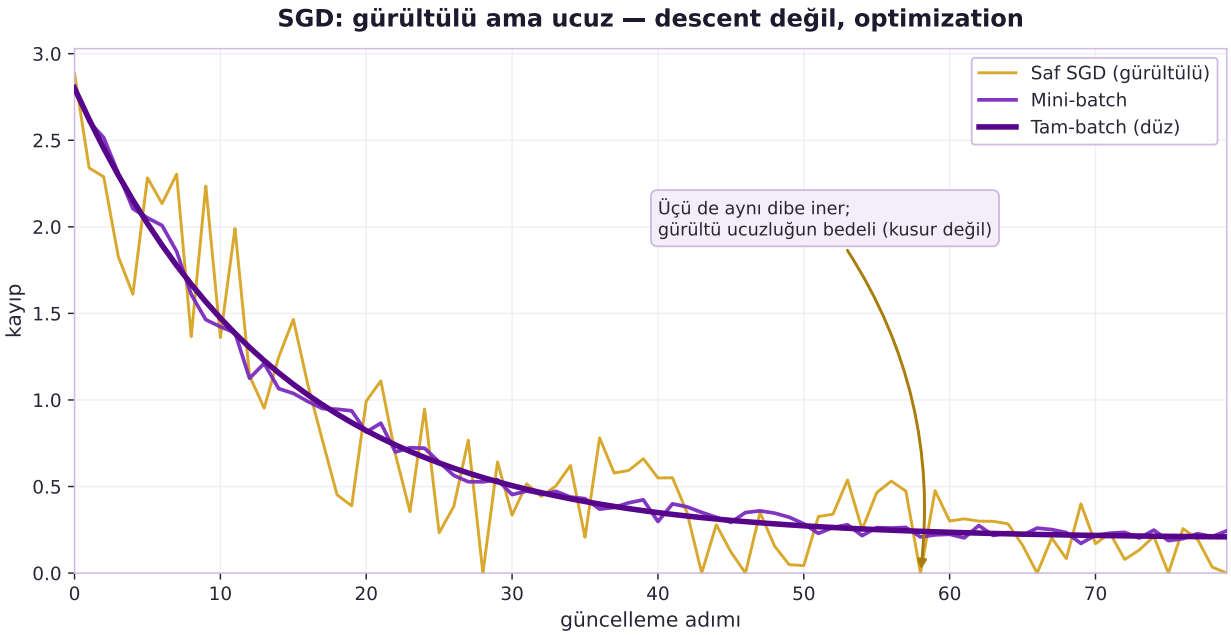
Güncelleme kuralı: parametreyi, **örnek-başına kaybın** gradient'inin η katı kadar geri al:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$



Şekil 9.1: Eğik bir kayıp vadisi üzerinde gradient descent yörüngesi: parametreler yüksek-kayıp başlangıcından kaybın ters yönünde küçük adımlarla vadinin dibine (minimuma) iner — LeCun'un 'dağdan iniş' benzetmesi.

Gürültü kötü değildir: sığ yerel minimumlardan kaçmaya yardım eder ve aynı bütçeyle çok daha sık güncelleme demektir. Tam-batch gradient her adımda "doğru" yönü verir ama bir adım için tüm veriyi tarar; SGD "yaklaşık doğru" yönü verir ama saniyede yüzlerce adım atar — pratikte ikincisi kazanır. Şekil 9.2 üç rejimi tek eksende karşılaştırır.



Şekil 9.2: Tek eksende üç kayıp eğrisi: tam-batch düz inerken, mini-batch ve saf SGD aynı dibe iner ama artan gürültüyle — gürültü, ucuz güncellemelerin bedelidir, bir kusur değil.

Üç uç noktayı karşılaştırmak aydınlatıcı:

- **Tam-batch (full-batch):** gradient'i tüm N örnek üzerinde hesapla. En doğru yön, ama bir tek güncelleme için tüm veriyi taradığından devasa veride imkânsız.
- **Tek örnek (saf SGD):** gradient'i tek bir rastgele örnek üzerinde hesapla. Çok hızlı ama çok gürültülü — yön çoğu zaman yanlış, ama ortalamada doğru.
- **Mini-batch (pratik standart):** gradient'i B örnekten (örn. 32, 64) oluşan bir batch üzerinde hesapla. Hız ile gürültü arasında ayarlanabilir denge. Modern donanım (GPU) zaten matris işlemlerini paralel yaptığından, mini-batch hem hızlı hem makul doğrudur.

Önemli nokta: gürültü "kusur" değil, bir özelliktir — non-convex bir yüzeyde stokastik dürtmeler, deterministik gradient descent'in takılacağı sığ çukurlardan modeli kurtarabilir.

Builder Notu — SGD: Gürültü Bir Özellik

Geriye (Stat 110): Mini-batch gradient'i, gerçek gradient'in **tarafsız tahmincisidir** (örneklem ortalaması, Stat 110 Monte Carlo); varyans batch boyutuyla azalır ($\propto 1/B$).
İleriye: SGD'nin ucuzluğu, milyar-parametreli modelleri eğitilebilir kılar; batch size $\leftrightarrow$ throughput dengesi, gradient accumulation ve dağıtık eğitimin (DDP) çekirdeğidir.

9.5 (LeCun) Backpropagation = Zincir Kuralı (Twiddling Sezgisi)

Gradient descent'in kalbi $\partial C/\partial\theta$ terimiydi — ama onu nasıl hesaplarız? Cevap **backpropagation**, ve LeCun'un üstüne basa basa söylediği şey: bu, zincir kuralından (chain rule) başka bir şey değildir.

> "we're going to use chain rule — chain rule, if you remember from kindergarten…" — LeCun, 27:36

LeCun zincir kuralını "twiddling" (azıcık oynatma) sezgisiyle yeniden türetiyor: bir nonlinearite $z = h(s)$ düşün. Girdiyi azıcık (ds kadar) oynatırsan, çıktı $dz = ds \cdot h'(s)$ kadar değişir. Bu da kaybı $dC = dz \cdot (\partial C/\partial z)$ kadar değiştirir. İkisini birleştirip ds'yi sadeleştirince:

$$\frac{\partial C}{\partial s} = \frac{\partial C}{\partial z}\, h'(s)$$

Yani biri sana kaybın z'ye göre türevini verirse, sen onu nonlinearitenin türeviyle çarpıp kaybın s'ye göre türevini elde edersin. Bir zincir boyunca bunu tekrarlarsın: gradyan'ı çıktıdan girişe, her h fonksiyonunun türeviyle çarparak taşırsın. LeCun yarı şaka der ki tüm yaptığı zincir kuralını yeniden türetmektir:

> "I've done nothing more than rederiving chain rule. But it's a little more intuitive if you think of it in terms of twiddling things around." — LeCun, 32:01

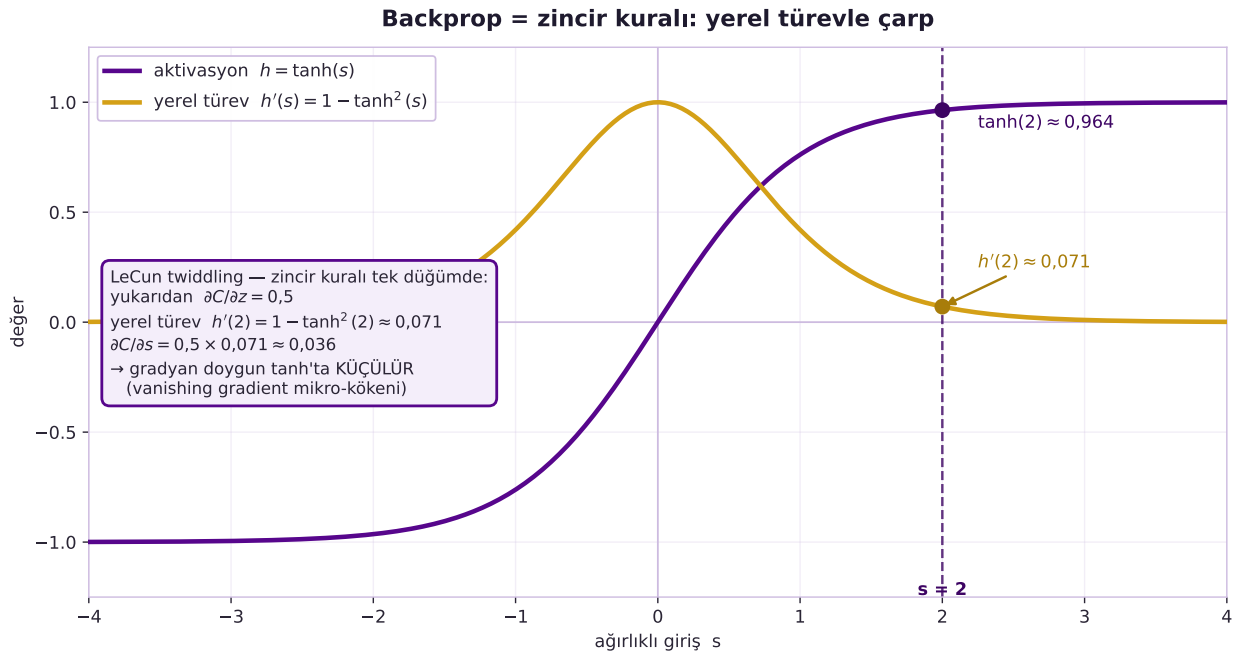
Somut örnek. Diyelim $s = 2$, $h = \tanh$, ve kaybın çıktıya göre türevi $\partial C/\partial z = 0{,}5$ olarak yukarıdan geldi. tanh'ın türevi $h'(s) = 1 - \tanh^2(s)$; $\tanh(2) \approx 0{,}964$ olduğundan $h'(2) \approx 1 - 0{,}929 = 0{,}071$. O hâlde $\partial C/\partial s = (\partial C/\partial z) \cdot h'(s) = 0{,}5 \times 0{,}071 \approx 0{,}036$. Yani yukarıdan gelen $0{,}5$'lik gradyan, doygun (saturated) tanh'tan geçince $0{,}036$'ya **küçülür** — bu, derin ağlarda vanishing gradient'in mikro-kökenidir (Hafta 4'te normalization/init bunu çözer). Şekil 9.3 bu somut çarpımı tanh eğrisi ve yerel türevi üzerinde gösterir; backprop böyle, modülden modüle yerel türevi çarparak ilerler.

> **Builder Notu — Twiddling = Zincir Kuralı**
>
> **Geriye (Calculus + Karpathy):** Bu doğrudan Phase 1 Calculus Ders 4 zincir kuralıdır; "twiddling" = türevin "küçük dürtme dx" tanımı. Karpathy micrograd'da her düğümün `_backward` kapanışı tam olarak bu yerel türevi ($h'(s)$) saklar ve tersten çağırır (reverse-mode autodiff).
> **İleriye:** PyTorch'ta `loss.backward()` bu zinciri otomatik yürütür; çok derin ağlarda zincirin uzaması vanishing/exploding gradient'e yol açar (Hafta 3-4 normalization/init).

9.6 (LeCun) Lineer Modülün Backprop'u: Ağırlıkları Tersten Kullanmak

Nonlinearite modülünü gördük; ikinci tür **lineer modül**dür. Burada bir z değişkeni, ağırlıklarıyla çarpılarak **birden çok** s değişkenini besler (dallanma). z'yi azıcık oynatırsan, beslediği her s'i kendi ağırlığı kadar etkilersin; ve her s kaybı etkiler. Küçük değişimlerde toplam etki, bu katkıların **toplamıdır**:

$$\frac{\partial C}{\partial z} = \sum_i w_i \frac{\partial C}{\partial s_i}$$



Şekil 9.3: tanh(s) aktivasyonu ve yerel türevi h'(s)=1−tanh^2(s); s=2 noktasında zincir kuralı somut çarpımı (∂C/∂z=0,5 × h'(2)≈0,071 = 0,036) doygun bölgede gradyanın nasıl küçüldüğünü — vanishing gradient'in mikro-kökenini — gösterir.

LeCun'un özlü ifadesi: ileri geçişte ağırlıkları "yukarı" kullanırsın, geri geçişte aynı ağırlıkları "aşağı" — gradyanların ağırlıklı toplamını alırsın.

> "when you back propagate through a neural net, you compute a weighted sum of the gradients using the weights backwards." — LeCun, 36:24

Yani ileri geçiş $s = Wz$, geri geçiş ise $\partial C/\partial z = W^\top(\partial C/\partial s)$ — aynı W matrisi, transpozuyla. Backprop'un simetrisi tam burada.

Builder Notu — Forward W, Backward W^T

Geriye (18.06): İleri geçiş W ile çarpma, geri geçiş $W^\top$ ile çarpmadır — 18.06'nın transpoz/iç çarpım dünyası. Dallanan değişkenin gradyanlarının toplanması, çok-değişkenli zincir kuralının (Calculus) doğrudan sonucudur.
İleriye: "Forward W, backward $W^\top$" simetrisi, her `nn.Linear` katmanının autograd implementasyonudur; özel katman yazarken bu çifti tanımlarsın.

9.7 (LeCun) Jacobian Formülasyonu ve Hesaplama Grafiği

LeCun backprop'u tek bir genel kurala indiriyor. Vektör değerli modüller için zincir kuralı **Jacobian matrisleriyle** yazılır. Bir modüller dizisinde, kaybın k. modülün girdisine göre gradient'i, bir sonraki gradient ile o modülün **Jacobian matrisinin** çarpımıdır:

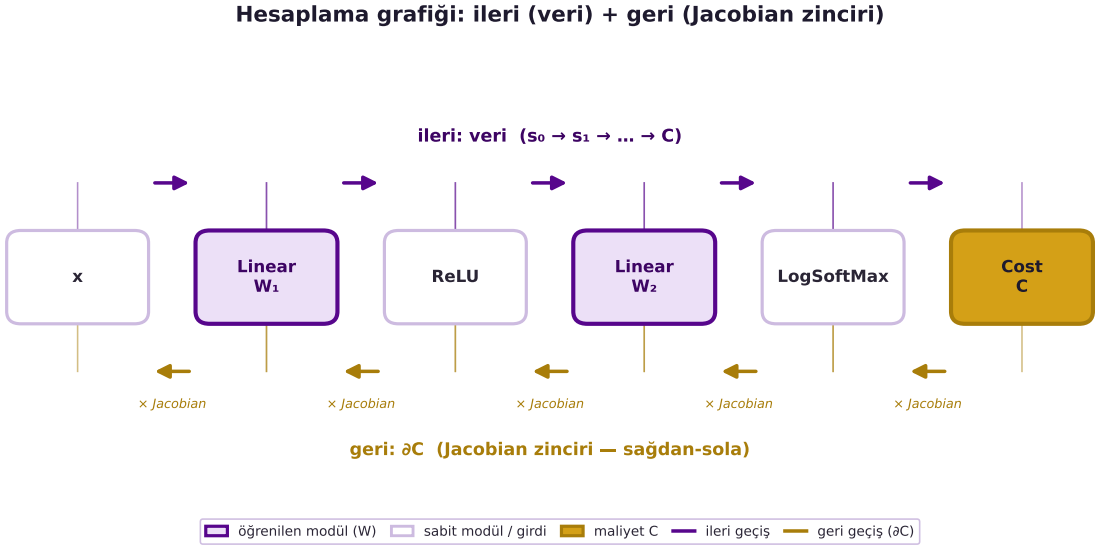
$$\frac{\partial C}{\partial z_k} = \frac{\partial C}{\partial z_{k+1}} \frac{\partial z_{k+1}}{\partial z_k}$$

Jacobian matrisinin ij elemanı, çıktının i. bileşeninin girdinin j. bileşenine kısmi türevidir. Backprop, çıktıdan girişe doğru bu Jacobian matrislerini **art arda çarpmaktan** ibarettir. Parametreli bir modülün **iki** Jacobian'ı vardır: biri girdiye göre (gradient'i bir önceki katmana taşır), biri ağırlıklara göre (o katmanın ağırlık gradient'ini verir). LeCun'un kapanışı:

> "that's all there is to backprop." — LeCun, 49:32

Bir ağı kavramsal olarak "linear–nonlinear çiftlerinin" istiflenmesi olarak görmek faydalı: $s_{k+1} = W_k z_k$, ardından $z_k = h(s_k)$. LeCun'un deyişiyle "bir katman, linear-nonlinear bir çifttir" (37:31) — ama modern ağlarda bu ayrım her zaman temiz değildir.

Hesaplama grafiği ve genelleştirilmiş backprop. LeCun bu fikri tek bir zincirden genel bir **hesaplama grafiğine** (computation graph) genelliyor. Bir ağ, modüllerin yönlü asiklik grafiğidir (DAG); ileri geçiş bu grafiği soldan-sağa hesaplar. Backprop ise aynı grafiği "türev grafiğine" çevirir: her modülün yerine Jacobian'ı geçer ve sinyalleri sağdan-sola taşır. Şekil 9.4 bu çift akışı tek bir grafik üzerinde gösterir: üstte ileri (veri), altta geri (Jacobian zinciri). Bu, modüllerin birden çok girdisi/çıktısı olduğunda da çalışır — dallanan bir değişkenin gradyanları toplanır (Bölüm 5'teki "ağırlıklı toplam" kuralı). Bu genellik sayesinde PyTorch, herhangi bir modül kombinasyonu için backward'ı otomatik üretir; sen yalnızca ileri geçişi tanımlarsın.



Şekil 9.4: x → Linear W_1 → ReLU → Linear W_2 → LogSoftMax → Cost C modül zincirinde ileri geçiş (violet, veri akışı) ile geri geçişin (gold, sağdan-sola ∂C Jacobian zinciri) aynı grafik üzerinde nasıl ters yönde aktığını gösteren hesaplama grafiği.

Ayrıca bir ayrım: **çıkarım (inference)** yalnızca ileri geçiştir — ağ, girdiyi $\mathbb{R}^n$'den $\mathbb{R}^c$'ye (n boyuttan c sınıfa) eşleyen sabit bir fonksiyondur. **Eğitim (training)** ise ileri + geri geçiş + güncelleme döngüsüdür. Çıkarımda backprop yoktur; backprop sadece öğrenme için gereklidir.

Builder Notu — Jacobian = Matris Zinciri

Geriye (18.06): Jacobian, bir vektör fonksiyonunun türev matrisidir; zincir = matris çarpımı (18.06 bileşke dönüşüm). Reverse-mode, bu çarpımları sağdan-sola yapar (verimli — çünkü kayıp skalerdir, soldan vektör küçük kalır).
İleriye: "İki Jacobian'lı modül" soyutlaması, her PyTorch katmanının `forward/backward` çiftidir; bu genel algoritma, çok-girişli/çok-çıkışlı modüllere de aynen uygulanır.

9.8 (LeCun) Aktivasyon ve LogSoftMax Modülleri

LeCun somut modül örnekleri veriyor. Çekirdek olanlar: **lineer** ve **ReLU** (pointwise nonlinearite). Sınıflandırma için kritik olan ise **LogSoftMax**: softmax bir skor vektörünü olasılık dağılımına çevirir (her eleman pozitif, toplamı 1); LogSoftMax bunun logaritmasıdır. Bu modül, ham skorları (logit) olasılığa çevirip kaybın hesaplanmasını sağlar — Canziani'nin Practicum'da kullanacağı cross-entropy'nin yapı taşıdır. NYU'da kullanılan modül listesi uzundur çünkü her görev (sınıflandırma, sıralama, gömme) kendi modülünü ister.

Builder Notu — Softmax'ın Olasılık Köprüsü

Geriye (Calculus + Stat 110): Softmax = argmax'ın yumuşak, türevlenebilir hâli; e^x (Calculus Ders 5) + multinomial dağılım (Stat 110). Log almak, çarpımları toplama çevirir ve sayısal kararlılık sağlar.
İleriye: Pratikte softmax + log + NLL tek bir kararlı işlemde birleştirilir (`F.cross_entropy`); ayrı yazmak taşma (overflow) riski taşır.

9.9 Geçiş: LeCun'dan Canziani'ye

LeCun makineyi kurdu — modüller, kayıp, gradient descent, backprop (twiddling → Jacobian zinciri), LogSoftMax. Şimdi **Canziani** bunları PyTorch'a indiriyor: önce bir ağın anatomisini "döndür-ez" diliyle yeniden anlatıyor, sonra Hafta 1'de kurduğumuz ama eğitmediğimiz spiral ağını **gerçekten eğitiyor**. LeCun "gradient nasıl hesaplanır"ı anlattı; Canziani "ağ neye benzer, kaybı nasıl seçer, eğitim döngüsünü nasıl yazarsın"ı gösteriyor.

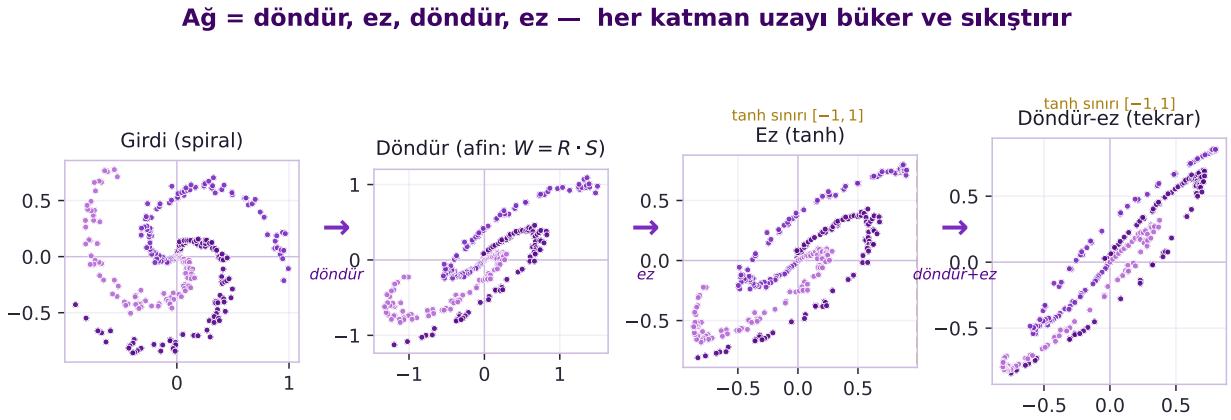
9.10 (Canziani) Ağ Anatomisi: Döndür–Ez (Rotation–Squashing)

Canziani tam-bağlı (fully connected) bir ağı tek bir cümleye indiriyor. Girdi x (alt, pembe) bir **afin dönüşümden** geçer, sonra bir **nonlineariteden** (f) — çıktısı gizli katman h (yeşil, vektör, "gizli" çünkü dışarıdan görünmez). Sonra yine afin + nonlinearite → çıktı $\hat{y}$ (mavi). Canziani'nin metaforu:

> "an affine transformation — I call them rotations; a nonlinear function — I call squashing. So you just repeat rotation, squashing, rotation, squashing." — Canziani, 15:43

Yani bir ağ = **döndür, ez, döndür, ez…** Afin dönüşüm (matris + bias) uzayı döndürür/ölçekler/öteler; nonlinearite onu büker (ezme). Hafta 1'in geometrisinin doğrudan devamı. Şekil 9.5 aynı spiral nokta bulutuna

sırayla döndürme ve ezme uygulayıp bu ritmi görselleştirir. Aktivasyon örnekleri: positive part (ReLU), sigmoid, tanh (sigmoid'in yeniden ölçeklenmiş hâli), soft(arg)max (argmax'ın yumuşak hâli).



Şekil 9.5: Canziani'nin 'döndür-ez' dizisi: aynı spiral nokta bulutuna sırayla afin döndürme (W=R·S) ve tanh ezme uygulanır — ağ, uzayı tekrar tekrar büküp [-1,1] kutusuna sıkıştıran bir döndür-ez-döndür-ez katman zinciridir.

Bir terminoloji tuzağı: Canziani bu ağı "üç katmanlı" sayar (girdi + gizli + çıktı nöron katmanı = 3); LeCun "iki katmanlı" der (programcı gibi sıfırdan + ağırlık matrisi sayısı). İkisi de aynı ağı kasteder — sadece sayma kuralı farklı.

> Builder Notu — Döndür-Ez = Hafta 1 Geometrisi
>
> **Geriye (Hafta 1):** "Döndür-ez", Hafta 1'in "uzay kumaşını ger + nonlinearite ile bük" geometrisinin tam karşılığıdır. Afin = rotation/scaling/shearing/reflection + öteleme (18.06).
> **İleriye:** Bu "katman = afin + nonlinearite" kalıbı, her derin mimarinin atomudur; residual bağlantılar ve normalization bu atomun üstüne eklenir.

9.11 (Canziani) Supervised Learning ve Sınıflandırma

Canziani sınıflandırmayı (supervised learning) hatırlatarak başlıyor: kedi/köpek görüntülerini ayırmak gibi. Her örneğin bir **etiketi (label)** vardır; üç sınıf varsa üç etiket. Ağ, girdiyi alıp bir sınıf tahmini üretir; eğitim, tahmini doğru etikete yaklaştırır.

> "supervised learning, classification — this is going to be kind of a revise of stuff you've seen." — Canziani, 2:43

Canziani önemli bir geometrik gerekçe ekliyor: ağ neden girdiyi **yüksek-boyutlu bir ara temsile** (D, $\mathbb{R}^n$'den çok daha büyük) taşır? Çünkü yüksek boyutta noktalar birbirinden çok uzaktır; uzakta olanları "döndürerek" ayırmak kolaydır, sıkışık olanları kıpırdatmaya çalışınca hepsi birlikte hareket eder.

> "whenever you go in a very high dimensional space, everything's far apart… it's very easy to rotate stuff and get things to move a little bit." — Canziani, 24:35

Builder Notu — Yüksek Boyutta Ayırmak Kolay

Geriye (Hafta 1 + 18.06): Sınıflandırma = uzayda noktaları ayrıştırma (manifold germe). "Yüksek-boyutta her şey uzak" gözlemi, Hafta 1'in 3-milyon-boyutlu uzay sezgisinin eğitime yansımasıdır.
İleriye: Etiketli veri pahalıdır — bu, kursun ilerleyen haftalarda öz-denetimli öğrenmeye (SSL, Hafta 8-10) neden yöneldiğinin sebebidir.

9.12 (Canziani) Cross-Entropy: Sınıflandırmanın Kaybı

Sınıflandırmada doğru kayıp nedir? Canziani örnek-başına kaybı tanımlıyor: doğru sınıfın softmax olasılığının **negatif logaritması**. Ağ doğru sınıfa yüksek olasılık verirse kayıp küçük, düşük verirse büyük olur.

> "this loss is also called cross entropy." — Canziani, 29:51

Doğru sınıf c için örnek-başına cross-entropy:

$$\mathcal{L}_{\text{CE}} = -\log \frac{e^{y_c}}{\sum_j e^{y_j}}$$

Etiketler genelde **one-hot** kodlanır (doğru sınıf 1, ötekiler 0); cross-entropy, modelin olasılık dağılımını bu one-hot hedefe yaklaştırır. Tüm veri setinde minimize edilen, örnek-başına kayıpların ortalamasıdır (LeCun bunu büyük L ile yazar).

Builder Notu — CE = Negatif Log-Olabilirlik

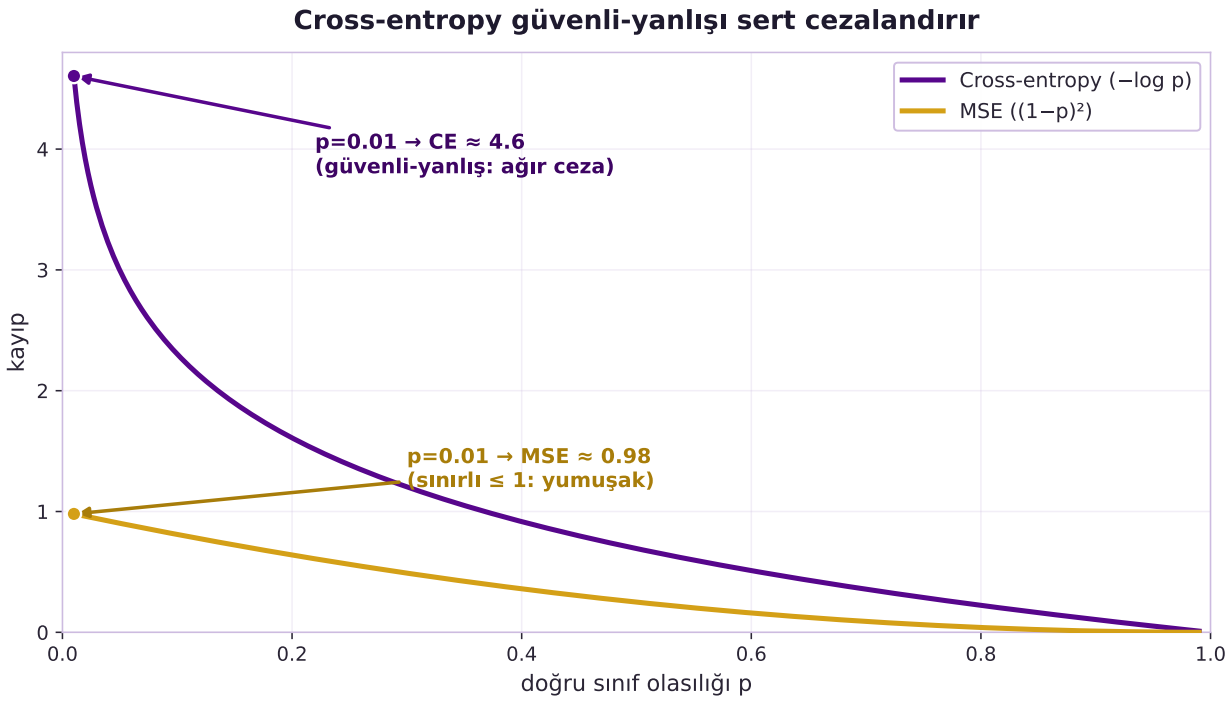
Geriye (Stat 110): Cross-entropy doğrudan olasılıktan gelir: doğru sınıfın **negatif log-olabilirliğini** (NLL) minimize etmek = maximum likelihood (Stat 110 Bernoulli/multinomial). Yani kayıp "uydurma" değil, olasılık teoreminin sonucudur — KL ıraksamasını azaltmaya denktir.
İleriye: Sınıf dengesizliğinde ağırlıklı/focal loss; çok-etiketli problemlerde sigmoid + BCE. Kayıp seçimi bir mühendislik kararıdır.

9.13 (Canziani) Cross-Entropy vs MSE: Görev Kaybı Belirler

Canziani'ye sınıfta sorulan klasik soru: neden cross-entropy de MSE (ortalama kare hata) değil? Cevap: görev belirler. **Sınıflandırmada** çıktı bir olasılıktır → cross-entropy çalışır ve istatistiksel gerekçesi vardır. **Regresyonda** (sürekli değer tahmini) çıktı sürekli olur → MSE kullanılır.

> "if we talk about classification, we're going to be talking about classes and labels [cross-entropy]; [in regression] targets [MSE]." — Canziani, 49:12

Şekil 9.6 iki kaybın doğru-sınıf olasılığı p karşısındaki davranışını yan yana koyar: cross-entropy $p \to 0$'da patlar, MSE ise sınırlı kalır. Aynı eğitim döngüsünde tek değişen, kriterdir (criterion): sınıflandırma için `CrossEntropyLoss`, regresyon için `MSELoss`. Sözlük de değişir: sınıflandırmada "sınıf/etiket", regresyonda "hedef (target)".



Şekil 9.6: Doğru sınıf olasılığı p düştükçe cross-entropy (−log p) patlar ama MSE $((1-p)^2)$ en fazla 1'de kalır; bu yüzden cross-entropy sınıflandırmada güvenli-yanlış tahmini sert cezalandırır.

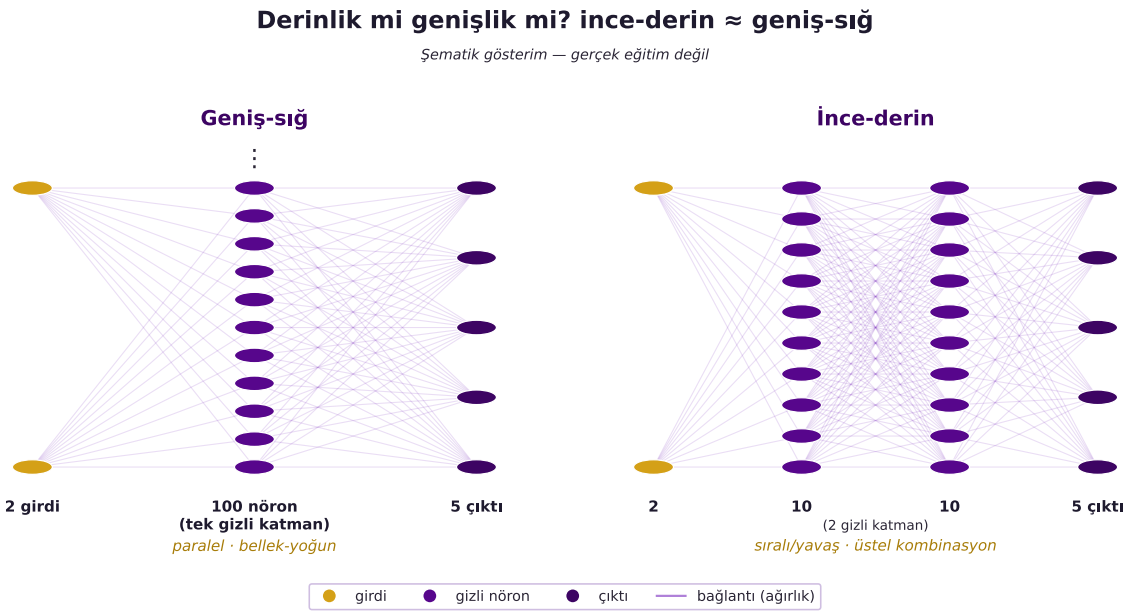
Builder Notu — Görev → Kayıp

Geriye (Stat 110): MSE, gürültünün Gaussian olduğu varsayımının maximum likelihood karşılığıdır; cross-entropy ise Bernoulli/multinomial'in. İki kayıp da aynı ilkeden (MLE) farklı dağılım varsayımlarıyla türer.
İleriye: "Görev → çıktı tipi → kayıp" zinciri, herhangi bir ML problemini kurarken ilk verilen karardır.

9.14 (Canziani) Derinlik vs Genişlik

Canziani pratik bir tasarım sorusu açıyor: aynı ifade gücünü tek bir **geniş** katmanla mı (örn. 100 nöron) yoksa birkaç **derin** katmanla mı (örn. 10+10) elde etmeli? İlginç gözlem: derin istifleme, nöron kombinasyonlarını **üstel** olarak artırdığı için, ince ama derin bir ağ, geniş ama sığ bir ağa yakın güç sağlayabilir.

Tradeoff ise hesaplamada: **derin** ağda veri bağımlılığı vardır — alttaki katman bitmeden üstteki başlayamaz, yani daha sıralı (yavaş). **Geniş** ağ daha paralelleştirilebilir ama çok daha fazla nöron/bellek ister. Şekil 9.7 iki mimariyi şematik olarak yan yana koyar. Yani "derin mi geniş mi" sorusunun cevabı donanıma (paralellik) ve bellek bütçesine bağlıdır.



Şekil 9.7: İki ağ mimarisinin şematik karşılaştırması: solda geniş-sığ ağ (2 girdi → 100 nöronlu tek gizli katman → 5 çıktı, paralel ama bellek-yoğun), sağda ince-derin ağ (2 → 10 → 10 → 5, sıralı/yavaş ama üstel katman kombinasyonu) — derinliğin az nöronla geniş-sığ ağa denk ifade gücü sağladığını gösterir.

> **Builder Notu — Derinlik mi Genişlik mi**
>
> **Geriye (Hafta 1):** Geniş ara temsil, "yüksek boyutta her şey uzak → ayırması kolay" sezgisini kullanır; derinlik ise her katmanda yeni nonlineariteyle ifade gücünü üst üste bindirir.
> **İleriye:** Derinlik/genişlik dengesi, modern ölçeklemenin (scaling laws), pipeline/tensor paralelliğinin ve bellek-throughput tradeoff'unun doğrudan konusudur — Hafta 4 ve sonrası.

9.15 (Canziani) PyTorch'ta Eğitim Döngüsü

Canziani teoriyi koda döküyor ve Hafta 1'in spiral ağını eğitiyor. Eğitim, parametreleri (ağırlık matrisleri) gradient descent ile güncellemektir:

> "how do you train a neural network? Gradient [methods]." — Canziani, 39:18

PyTorch'ta standart döngü: ileri geçiş → kaybı hesapla → `backward()` (LeCun'un Jacobian zinciri) → optimizer adımı.

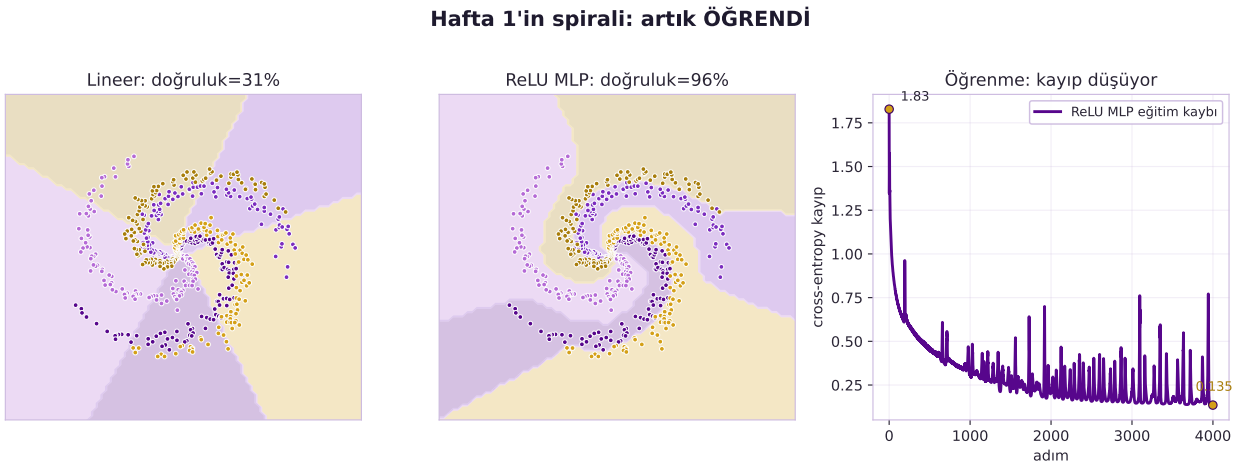
```
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(2, 100), nn.ReLU(),   # dondur + ez (rotation + squashing)
    nn.Linear(100, 5),              # 5 sinif (spiral kollari)
```

```
)
criterion = nn.CrossEntropyLoss()                # softmax + NLL birlikte
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

for epoch in range(1000):
    y_pred = model(x)               # ileri gecis (forward)
    loss = criterion(y_pred, y)     # cross-entropy
    optimizer.zero_grad()           # gradyanlari sifirla (Karpathy'nin meshur hatasi)
    loss.backward()                 # backprop = Jacobian zinciri
    optimizer.step()                # theta <- theta - eta * grad
```

Döngü ilerledikçe kayıp düşer; eğitim sonunda spiral kolları lineer ayrılabilir hale gelir — Hafta 1'de göremediğimiz "öğrenme" işte budur. Şekil 9.8 Hafta 1'in spiral ağını gerçekten eğitir: lineer sınıflandırıcı kolları ayıramaz, ReLU MLP ayırır, ve eğitim kaybı düşer. Canziani döngünün ilk ve son kayıp değerlerini karşılaştırarak ağın gerçekten öğrendiğini gösterir (örn. ilk kayıp yüksek, son kayıp 0.86 gibi düşük bir değere iner).



Şekil 9.8: Hafta 1'in spiral ağını gerçekten eğitir: lineer softmax sınıflandırıcı 5 kolu ayıramaz (%31), ReLU MLP manifoldu açarak kolları ayırır (%96) ve eğitim kaybı 4000 adımda 1,83'ten 0,135'e düşer.

Builder Notu — Dört Satırlık Döngü

Geriye (LeCun + Karpathy): `loss.backward()` tam olarak LeCun'un Jacobian zinciridir; `zero_grad()` Karpathy'nin "her backward öncesi gradyanı sıfırla" dersidir (atlanırsa gradient yanlış birikir).

İleriye: Bu dört satırlık döngü (forward → loss → backward → step) tüm derin öğrenme eğitiminin iskeletidir; üstüne validation, checkpoint, LR schedule, mixed precision biner.

9.16 Bu Dersin Özeti

1. **Eğitim = kaybı gradient descent ile küçültmek.** Ağ modüllerden kuruludur; maliyet modülü tahmin ile gerçeği karşılaştırır.
2. **Gradient descent** kaybı gradient'in ters yönünde küçük adımlarla azaltır; **SGD** bunu tek örnek/batch üzerinde tahminle yapar ("descent değil, optimization" — gürültülü ama ucuz).
3. **Backpropagation = zincir kuralı.** "Twiddling" sezgisiyle: gradyan'ı çıktıdan girişe her modülün türeviyle çarparak taşırsın; vektörel hâlde bu **Jacobian matrislerinin çarpımıdır**. Lineer modülde "ağırlıkların tersten ağırlıklı toplamı" (forward W, backward $W^\top$). "That's all there is to backprop."
4. **Ağ = döndür-ez** (afin + nonlinearite) dizisidir; LogSoftMax/softmax skorları olasılığa çevirir.
5. **Cross-entropy** sınıflandırmanın kaybıdır (−log softmax = NLL); **MSE** regresyonunki. Görev kaybı belirler.
6. **Derinlik vs genişlik:** ince-derin ≈ geniş-sığ (üstel kombinasyon), ama derin = sıralı/yavaş, geniş = paralel/bellek-yoğun.
7. **PyTorch eğitim döngüsü:** forward → loss → zero_grad → backward → step. Hafta 1'in spiral ağı bununla eğitilip ayrılır.

! Tek Bir Cümle

Bir ağı eğitmek, bir kaybı (sınıflandırmada cross-entropy) gradient descent ile küçültmektir; gradient'i backpropagation — zincir kuralının modül-modül Jacobian çarpımlarıyla uygulanması — hesaplar, ve tüm bu döngü PyTorch'ta forward → loss → backward → step'ten ibarettir.

9.17 Kontrol Soruları

i Soru 1: Backpropagation neyden ibarettir? 'Jacobian' kelimesini kullanarak açıkla.

Cevap: Backprop, zincir kuralının modüllere uygulanmasıdır. Vektörel modüllerde gradient, çıktıdan girişe doğru her modülün **Jacobian matrisiyle** çarpılarak taşınır:

$$\frac{\partial C}{\partial z_k} = \frac{\partial C}{\partial z_{k+1}} \frac{\partial z_{k+1}}{\partial z_k}$$

Parametreli bir modülün iki Jacobian'ı vardır: girdiye göre olan gradient'i bir önceki katmana taşır; ağırlıklara göre olan o katmanın ağırlık gradient'ini verir. Lineer modülde bu "forward W, backward $W^\top$" simetrisidir. LeCun'un deyişiyle "that's all there is to backprop" (49:32). Karpathy bunu micrograd'da `_backward` kapanışlarıyla koda döker.

i Soru 2: LeCun neden 'SGD aslında bir descent algoritması değil' diyor? Gürültü neden kötü değildir?

Cevap: Çünkü SGD gradient'i tüm veri yerine tek bir (veya küçük batch) örnek üzerinde **tahmin eder**; bu tahmin gürültülüdür, dolayısıyla her adım kaybı garantili azaltmaz — bazen artırabilir. Bu yüzden LeCun "stochastic gradient optimization" demeyi tercih eder (17:10). Gürültü kötü değildir: (1) çok daha ucuz ve sık güncelleme demektir, (2) sığ yerel minimumlardan kaçmaya yardım eder. Stat 110 köprüsü: mini-batch gradient'i gerçek gradient'in tarafsız tahmincisidir, varyans $\propto 1/B$.

ℹ Soru 3: Sınıflandırmada cross-entropy, regresyonda MSE kullanılır. Bu seçim neye dayanır? Cross-entropy formülünü yaz.

Cevap: Görevin çıktı tipine ve onun altındaki olasılık varsayımına dayanır. Sınıflandırmada çıktı bir olasılık dağılımıdır; doğru sınıf c için cross-entropy:

$$\mathcal{L}_{\text{CE}} = -\log \frac{e^{y_c}}{\sum_j e^{y_j}}$$

Bu, doğru sınıfın negatif log-olabilirliğidir (NLL) → maximum likelihood (Stat 110 multinomial). Regresyonda çıktı süreklidir; MSE, Gaussian gürültü varsayımının maximum likelihood karşılığıdır. İkisi de aynı ilkeden (MLE) farklı dağılımlarla türer — kayıp "uydurma" değildir.

ℹ Soru 4: (Builder) PyTorch eğitim döngüsündeki dört adımı (forward, loss, backward, step) LeCun'un ve Karpathy'nin kavramlarıyla eşle. zero_grad() neden gerekli?

Cevap: forward = modüllerin ileri geçişi (LeCun'un fonksiyonel modülleri, "döndür-ez"); **loss** = maliyet modülü ($\hat{Y}$ ile Y karşılaştırması, cross-entropy); **backward** = backpropagation, LeCun'un Jacobian zinciri (`loss.backward()`); **step** = gradient descent güncellemesi $\theta \leftarrow \theta - \eta \nabla L$. `zero_grad()` gereklidir çünkü PyTorch gradyanları varsayılan olarak **biriktirir** (toplar); her backward öncesi sıfırlanmazsa önceki adımların gradyanları yenisine eklenir ve eğitim bozulur — Karpathy'nin meşhur hatası.

9.18 Egzersizler

Egzersiz 1 (Cross-entropy elle). Bir skor vektörü (logit) için softmax'i NumPy ile hesapla, sonra doğru sınıfın –log olasılığını al. Aynı sonucu `torch.nn.functional.cross_entropy` ile karşılaştır.

```
import numpy as np
def softmax(z): e = np.exp(z - z.max()); return e / e.sum()
z = np.array([2.0, 0.5, -1.0]); c = 0           # dogru sinif 0
p = softmax(z); print(-np.log(p[c]))            # cross-entropy
```

Egzersiz 2 (Spiral'ı eğit). Hafta 1'de kurduğun spiral ağına (2 → 100 → ReLU → 5) `CrossEntropyLoss` + `SGD` ekle ve 1000 epoch eğit. Kayıp eğrisini çiz; ilk ve son kayıp değerlerini karşılaştır. Karar bölgelerinin nasıl ayrıştığını gözlemle.

Egzersiz 3 (zero_grad'ı unut). Eğitim döngüsünden `optimizer.zero_grad()` satırını sil ve eğit. Kaybın neden bozulduğunu açıkla (gradyan birikimi). Sonra geri ekle.

Egzersiz 4 (CE vs MSE). Aynı spiral verisinde (a) cross-entropy, (b) MSE (one-hot hedeflerle) ile eğit. Hangisi daha iyi/kararlı yakınsıyor? Neden cross-entropy sınıflandırma için doğru seçim?

Egzersiz 5 (Hafta 3 habercisi — görüntüye geçiş). Bu hafta girdi 2 sayıydı. Şimdi 28×28 piksellik bir rakam görüntüsünü düşün: `nn.Linear` ile işlemek için 784 boyuta düzleştirmen gerekir — ve uzamsal yapı (komşu pikseller) kaybolur. (a) 1000×1000 RGB için ilk linear katmanın kaç parametresi olurdu? (b) Bu "parametre patlaması" ve kaybolan uzamsal yapı, Hafta 3'te **convolution**'a neden ihtiyaç duyduğumuzu motive eder — neden?

9.19 Sonraki Ders İçin Hazırlık

> ⚠ Sonraki Hafta — H3: Evrişimli Ağlar (ConvNets)
>
> **Tam-bağlı ağdan doğal sinyallere.** Bu hafta tam-bağlı (fully connected) ağları eğittik. Hafta 3'te LeCun görüntü gibi **doğal sinyallerin** yapısını (yerel, hiyerarşik, öteleme-değişmez) ve bunu sömüren **convolution**'ı anlatacak; görsel korteksten LeNet5'e uzanacak. Canziani doğal sinyaller ve pooling'i PyTorch'ta gösterecek — Egzersiz 5'in "parametre patlaması" tam burada çözülür.

Hafta 3: Evrişimli Ağlar (ConvNets) ve Doğal Sinyaller — LeCun (Lecture) + Canziani (Practicum)

Bu hafta tam-bağlı (fully connected) ağları eğittik. Hafta 3'te LeCun görüntü gibi **doğal sinyallerin** yapısını (yerel, hiyerarşik, öteleme-değişmez) ve bunu sömüren **convolution**'ı anlatacak; görsel korteksten LeNet5'e uzanacak. Canziani doğal sinyaller ve pooling'i PyTorch'ta gösterecek.

Hafta 3 öncesi yapılacak:

- Egzersiz 2'yi (spiral eğitimi) ve Egzersiz 5'i (görüntü/parametre patlaması) çöz.
- "Backprop = zincir kuralının Jacobian çarpımlarıyla uygulanması" cümlesini kendi sözcüklerinle yaz.
- Karpathy micrograd'ı (Phase 2) hatırla — `loss.backward()` orada elle nasıl kuruluyordu?

9.20 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Modül	Girdi → çıktı hesaplayan blok; ağ modüllerden kurulur	LeCun 6m04
Maliyet (cost) modülü	$\hat{Y}$ ile Y'yi karşılaştırıp kaybı üretir; tepeye konur	LeCun 1m43
Gradient descent	Gradient'in ters yönünde küçük adım; dağda iniş	LeCun 10m27
SGD	Gradient'i tek örnek/batch'te tahmin; gürültülü, ucuz	LeCun 17m10
Backprop = zincir kuralı	Gradient'i çıktıdan girişe taşıma; twiddling sezgisi	LeCun 27m36
Ağırlıkları tersten	Forward W, backward $W^\top$; gradyanların ağırlıklı toplamı	LeCun 36m24
Jacobian matrisi	Vektör fonksiyonun türev matrisi; backprop = Jacobian çarpımları	LeCun 46m10
Döndür-ez	Ağ = afin (rotation) + nonlinearite (squashing) dizisi	Canziani 15m43
Cross-entropy (NLL)	Sınıflandırma kaybı: −log softmax (doğru sınıf)	Canziani 29m51
MSE	Regresyon kaybı: ortalama kare hata	Canziani 48m46
Derinlik vs genişlik	İnce-derin ≈ geniş-sığ; derin sıralı, geniş paralel	Canziani 25m21

Kavram	Tanım	Hoca / timestamp
Eğitim döngüsü	forward → loss → zero_grad → backward → step	Canziani 39m18

9.21 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Backprop / zincir kuralı** → Calculus Ders 4 + Karpathy micrograd (`_backward`, autograd).
2. **Cross-entropy / NLL** → Stat 110 Bernoulli/multinomial log-olabilirlik (MLE).
3. **Jacobian / zincir = matris çarpımı + forward W / backward $W^\top$** → 18.06 bileşke dönüşüm ve transpoz.
4. **SGD tarafsız tahmin** → Stat 110 örneklem ortalaması, varyans $\propto 1/B$.
5. **MSE = Gaussian MLE** → Stat 110 normal dağılım.

İleriye köprüler (production / research):

1. **module + Jacobian** → PyTorch `nn.Module` + `autograd` (`loss.backward()`).
2. **SGD ucuzluğu** → mini-batch, gradient accumulation, DDP.
3. **softmax+log+NLL birleşik** → `F.cross_entropy` (sayısal kararlılık).
4. **Derinlik/genişlik + eğitim döngüsü** → scaling laws, checkpoint, LR schedule, mixed precision.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Bir ağı eğitmek sihir değildir — bir kaybı gradient descent ile küçültmektir; ve gradient'i hesaplayan backpropagation, zincir kuralının modül-modül Jacobian çarpımlarıyla uygulanmasından başka bir şey değildir. LeCun bunu Jacobian cebriyle kurar, Canziani PyTorch'ta dört satırda (forward → loss → backward → step) gösterir — ve Hafta 1'in eğitilmemiş spiral ağı işte böyle ayrışır.

10 Evrişimli Ağlar (ConvNets) ve Doğal Sinyaller

NYU'nun iki hocalı ritmi: LeCun convolution'ın *neden*ini verir — doğal dünya kompozisyoneldir, kökü görsel kortekstedir (Hubel-Wiesel → LeNet5); Canziani bunu sistematikleştirir — doğal sinyallerin üç özelliği (stationarity, locality, compositionality) doğrudan üç mimari karara (parameter sharing, sparsity, hiyerarşi) çevrilir

> Bölüm bilgisi
>
> - **LeCun'un Lecture videosu:** YouTube — Lecture 03: Convolution operator and deep ConvNets (≈98 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Practicum 03: Properties of natural signals (≈48 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈30 dk

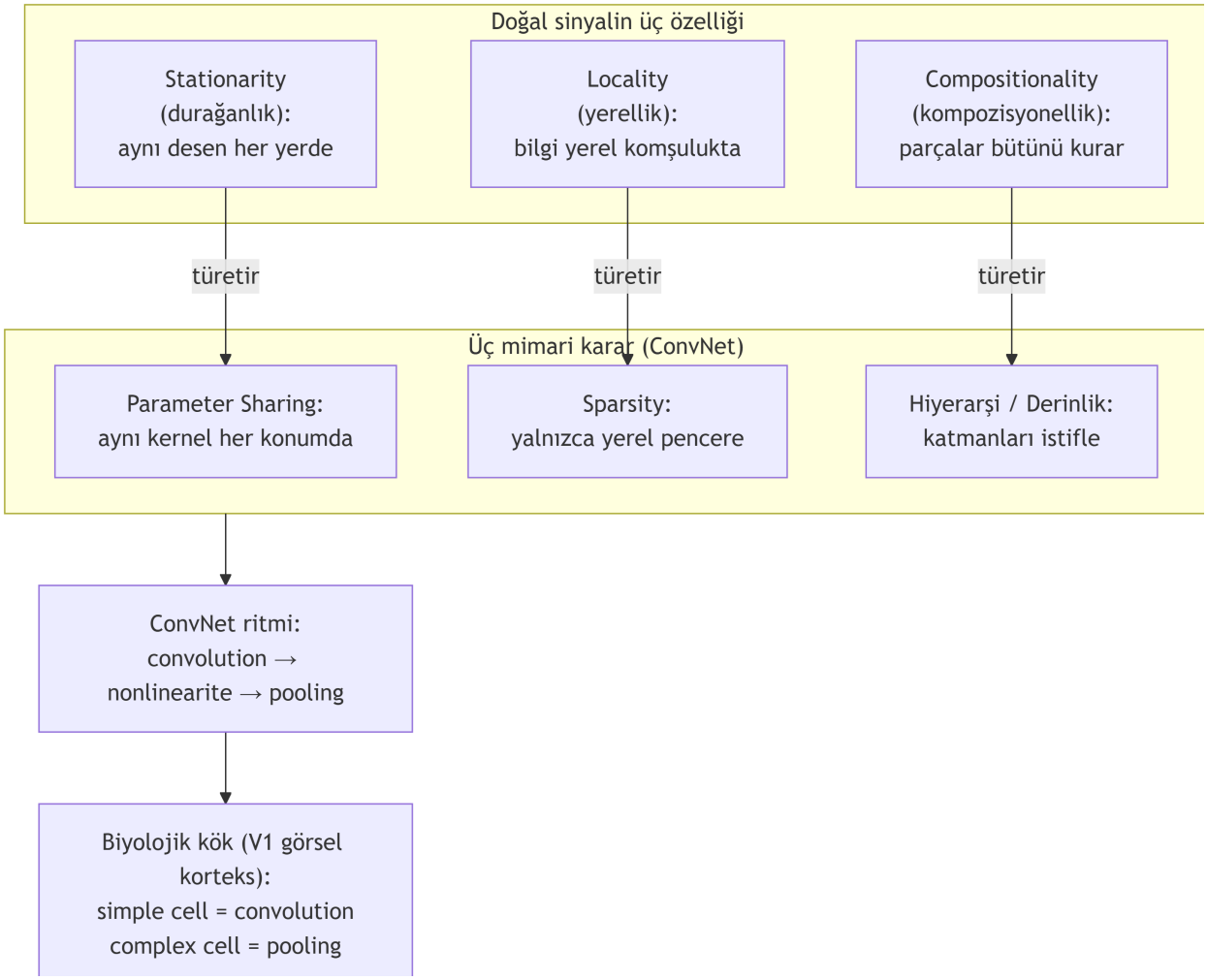
10.1 Bu Derste Ne Var?

Hafta 2'nin sonunda bir sorun bırakmıştık: tam-bağlı (fully connected) bir ağ bir görüntüyü işlemek için onu uzun bir vektöre **düzleştirir** — ve bu, hem parametre sayısını patlatır hem de komşu piksellerin **uzamsal yapısını** yok eder. Bu hafta çözümü görüyoruz: **convolution** ve **evrişimli ağlar (ConvNets)**.

Yine iki hocalı. **Yann LeCun** (Lecture) convolution'ın *neden* doğru araç olduğunu anlatır: doğal dünya **kompozisyoneldir** (kenarlar köşeleri, köşeler nesneleri oluşturur) ve bu fikrin kökü **biyolojidedir** — görsel korteks (Hubel & Wiesel). Sonra **Alfredo Canziani** (Practicum) bunu somutlaştırır: doğal sinyallerin **üç özelliğini** (stationarity, locality, compositionality) tanımlar ve her birinin hangi mimari seçime (parameter sharing, sparsity) yol açtığını gösterir.

Bu haftanın üç ana fikri:

1. **Convolution = kayan pencere + paylaşılan ağırlıklar (kernel).** Aynı küçük filtreyi tüm görüntüde gezdirirsin; bu, parametreyi azaltır ve öteleme-değişmezlik kazandırır.
2. **ConvNet = convolution + nonlinearite + pooling**, hiyerarşik olarak istiflenir; çünkü dünya kompozisyoneldir.
3. **Mimari, verinin yapısından doğar:** locality → sparsity (seyrek bağlantı), stationarity → parameter sharing (ağırlık paylaşımı).



Builder Notu — Convolution: Hafta 2'nin Açık Sorusunu Kapatmak

Geriye (önkoşul kurslar):

- **Convolution = kayan iç çarpım** → 18.06 dot product / lineer operatör + Hafta 1 "filtre·yama" sezgisi.
- **Hiyerarşik öznitelik** → Hafta 1 (elle vs öğrenilen hiyerarşi) + Calculus fonksiyon bileşkesi.
- **Parameter sharing / öteleme-değişmezlik** → §4.J "equivariance vs invariance"; 18.06 öteleme-değişmez operatör.

İleriye (production / research):

- ConvNet omurgası → ResNet, U-Net, YOLO, ve görü foundation modelleri.
- "Veri yapısı → mimari" ilkesi → geometric deep learning (graf ağları, Hafta 13) ve transformer'ların (Hafta 12) tasarım felsefesi.

Tek cümleyle: Convolution, doğal sinyallerin yerel, tekrar eden, kompozisyonel yapısını sömüren bir kayan-filtre işlemidir; ConvNet bu filtreleri nonlinearite ve pooling ile hiyerarşik istifleyerek görüntüden öznitelikleri uçtan uca öğrenir.

10.2 (LeCun) Neden Tam-Bağlı Ağ Yetmez?

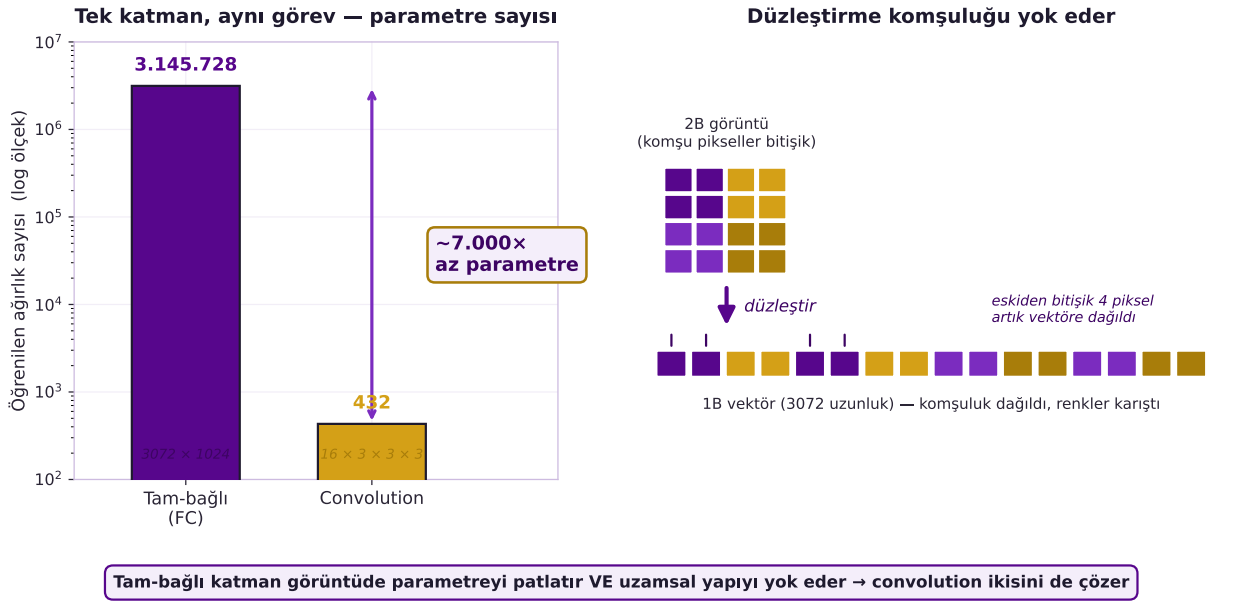
LeCun convolution'a, tam-bağlı ağın görüntüde neden başarısız olduğunu göstererek giriyor. Bir görüntüyü `nn.Linear`'a vermek için onu tek bir vektöre düzleştirmen gerekir. İki büyük sorun çıkar:

1. **Parametre patlaması.** 1000×1000×3 = 3 milyon girdiyi 1000 nöronlu bir katmana bağlamak 3 milyar ağırlık demektir — tek bir katmanda. Eğitilemez, belleğe sığmaz.
2. **Uzamsal yapı kaybı.** Düzleştirme, komşu piksellerin komşuluğunu yok eder; ağ "şu piksel şunun yanında" bilgisini baştan kaybeder. Oysa bir görüntüde anlam tam da bu yerel komşuluktadır.

Çözüm, görüntünün yapısını kullanan bir operatördür. LeCun'un cevabı: aynı küçük **yerel öznitelik dedektörünü** (filtre) tüm görüntüde gezdir, çıktılarını topla — ve **öteleme-değişmez** bir tespit elde et.

> "[you have] local feature detectors and then [you] sum up their activity, and what you get is an invariant detection." — LeCun, 25:41

Şekil 10.1 her iki sorunu da somutlaştırır: solda tam-bağlı katman convolution'a göre ~7000× fazla parametre ister, sağda düzleştirme komşu pikselleri 1B vektöre dağıtarak uzamsal yapıyı bozar — convolution ikisini birden çözer.



Şekil 10.1: Tam-bağlı bir katman görüntü girdisinde parametreyi patlatır (3.145.728'e karşı 432, ~7000× fazla) ve düzleştirme komşu pikselleri 1B vektöre dağıtarak uzamsal yapıyı yok eder; convolution her iki sorunu da çözer.

> **Builder Notu — Parametre Patlaması**
>
> **Geriye (Hafta 2):** Bu, Hafta 2 Egzersiz 5'in cevabıdır: 1000×1000 RGB için tam-bağlı ilk katman milyarlarca parametre ister; convolution bunu birkaç bine indirir (kernel paylaşımı).
> **İleriye:** "Yapıyı sömür" ilkesi her modern mimaride var: convolution uzamsal yapıyı, attention dizi/grafı, GNN graf yapısını sömürür.

10.3 (LeCun) Weight Sharing ve Convolution Operatörü

İki fikir convolution'ı kurar. Birincisi **weight sharing (ağırlık paylaşımı)**: bir ağırlık grubunu görüntünün her konumunda *aynen* kullanmaya zorlarsın.

> "you change all the copies of W at the same time, in a very simple manner — and that's called weight sharing: when two weights are forced to be [equal]." — LeCun, 12:06

İkincisi **convolution'ın tanımı**: küçük bir pencereyi (kernel) girdinin üzerinde kaydırıp her konumda bir iç çarpım hesaplamak.

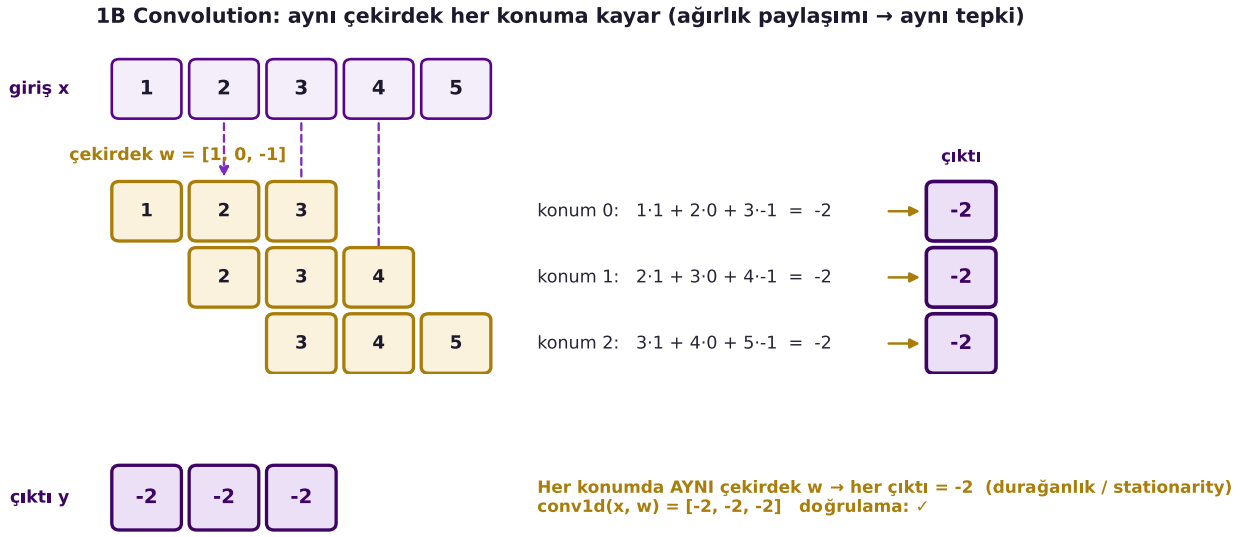
> "an input window, and then swiping it over — that's a convolution." — LeCun, 26:49

1B durumda, sinyal x ile kernel w'nin convolution'ı:

$$(x * w)[i] = \sum_k x[i+k]\, w[k]$$

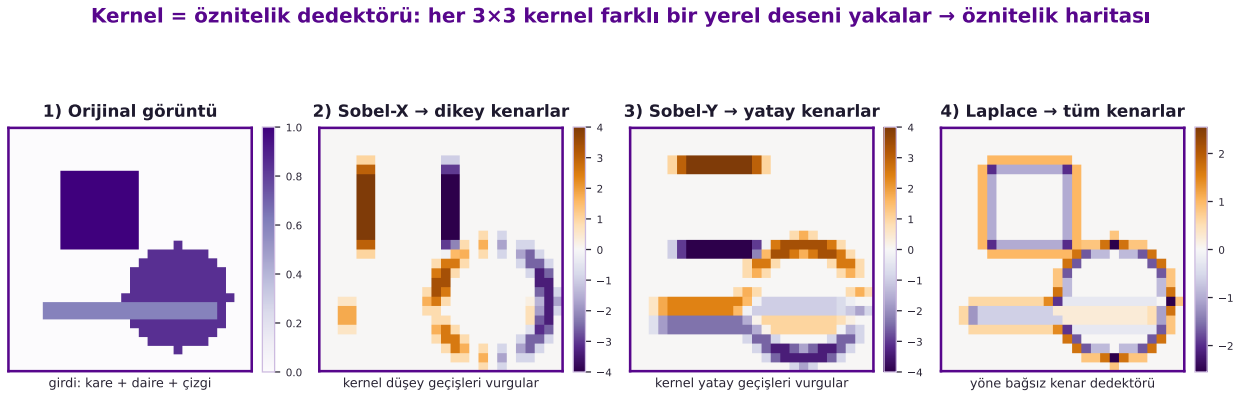
Aynı w her i konumunda kullanılır (paylaşım). 2B/3B/4B kernel'lere genelleşir; çok-boyutlu kernel'e LeCun "kernel" der. (Teknik not: matematiksel convolution kernel'i ters okur; ML'de genelde ters okumadan kullanırız — buna aslında cross-correlation denir, ama pratikte "convolution" deriz.)

Somut örnek. $x = [1, 2, 3, 4, 5]$, kernel $w = [1, 0, -1]$ (basit bir kenar/fark dedektörü). Pencereyi kaydırarak her konumda iç çarpım: konum 0 → $1 \cdot 1 + 2 \cdot 0 + 3 \cdot (-1) = -2$; konum 1 → $2 \cdot 1 + 3 \cdot 0 + 4 \cdot (-1) = -2$; konum 2 → $3 \cdot 1 + 4 \cdot 0 + 5 \cdot (-1) = -2$. Çıktı $[-2, -2, -2]$: sinyal düzgün arttığı için fark dedektörü her yerde aynı yanıtı verir. İşte stationarity (aynı desen) + weight sharing (aynı kernel) birlikte: aynı yapı nerede olursa olsun aynı tepki. Şekil 10.2 bu üç kayma konumunu ve her konumda aynı iç çarpımı somut olarak gösterir.



Şekil 10.2: x=[1,2,3,4,5] sinyaline w=[1,0,-1] kenar dedektörünün üç konumda kaydırılması: her konumda aynı çekirdek aynı iç çarpımı (= -2) üretir, böylece ağırlık paylaşımı çıktı [-2,-2,-2] ile durağanlığı (stationarity) somutlaştırır.

Bu yüzden convolution doğal olarak bir **öznitelik dedektörüdür**: kernel belli bir yerel deseni (kenar, köşe) arar; çıktısı yüksekse "burada o desen var" der. Yerel dedektörlerin çıktılarını topladığında, desenin konumundan görece bağımsız bir tespit elde edersin. Şekil 10.3 aynı fikri 2B'de gösterir: farklı kernel'ler aynı görüntüden farklı yerel desenleri (kenar yönlerini) çeker.



Şekil 10.3: Aynı sentetik görüntüye uygulanan dört farklı 3×3 kernel: Sobel-X dikey kenarları, Sobel-Y yatay kenarları, Laplace ise tüm kenarları yakalar — her kernel ayrı bir yerel deseni vurgulayan bir öznitelik haritası (feature map) üretir.

> Builder Notu — Kernel = Kayan İç Çarpım
>
> **Geriye (18.06 + Hafta 1):** Kernel·yama iç çarpımı, 18.06 dot product'tır; her konumda aynı kernel, Hafta 1'in "filtre bir yamaya bakar" sezgisinin tüm görüntüye yayılmış hâli.
> **İleriye:** Weight sharing parametreyi dramatik azaltır ve **öteleme-değişmezlik (equivariance)** kazandırır — nesne nerede olursa olsun aynı filtre onu yakalar. Donanımda convolution, optimize edilmiş bir GEMM/kernel çağrısıdır.

10.4 (LeCun) ConvNet'in Üç İşlemi: Conv, Nonlinearite, Pooling

LeCun bir ConvNet'i üç işlemin dönüşümlü tekrarı olarak tanımlıyor:

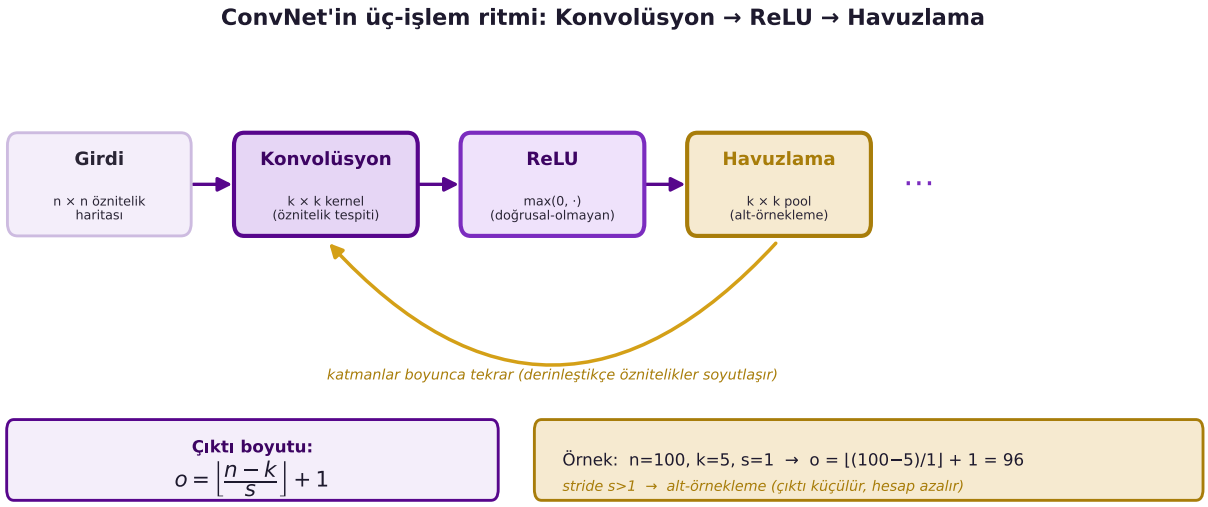
> "[a convnet is an] alternation of linear operators [convolution] and pointwise non-linearity… and there's going to be a third type of operation called pooling, which is actually optional." — LeCun, 30:24

1. **Convolution** — yerel öznitelikleri çıkarır (paylaşılan kernel).
2. **Nonlinearite** (ReLU) — Hafta 1-2'deki gerekçeyle, doğrusal çöküşü önler.
3. **Pooling** (opsiyonel) — küçük bir pencerede özetler (örn. 2×2 ortalama/maksimum), çözünürlüğü düşürür ve küçük konum değişimlerine dayanıklılık katar.

Önemli bir ayar **stride** (adım): kernel'i 1 yerine s piksel kaydırırsan çıktı küçülür. Girdi boyutu n, kernel k, stride s için çıktı boyutu:

$$o = \left\lfloor \frac{n-k}{s} \right\rfloor + 1$$

Örneğin $n = 100$, $k = 5$, $s = 1 \rightarrow$ çıktı 96. Stride > 1, hem hesabı azaltır hem de pooling gibi alt-örnekleme (downsampling) yapar. Şekil 10.4 bu üç-işlem ritmini ve çıktı boyutu formülünü tek şemada toplar.



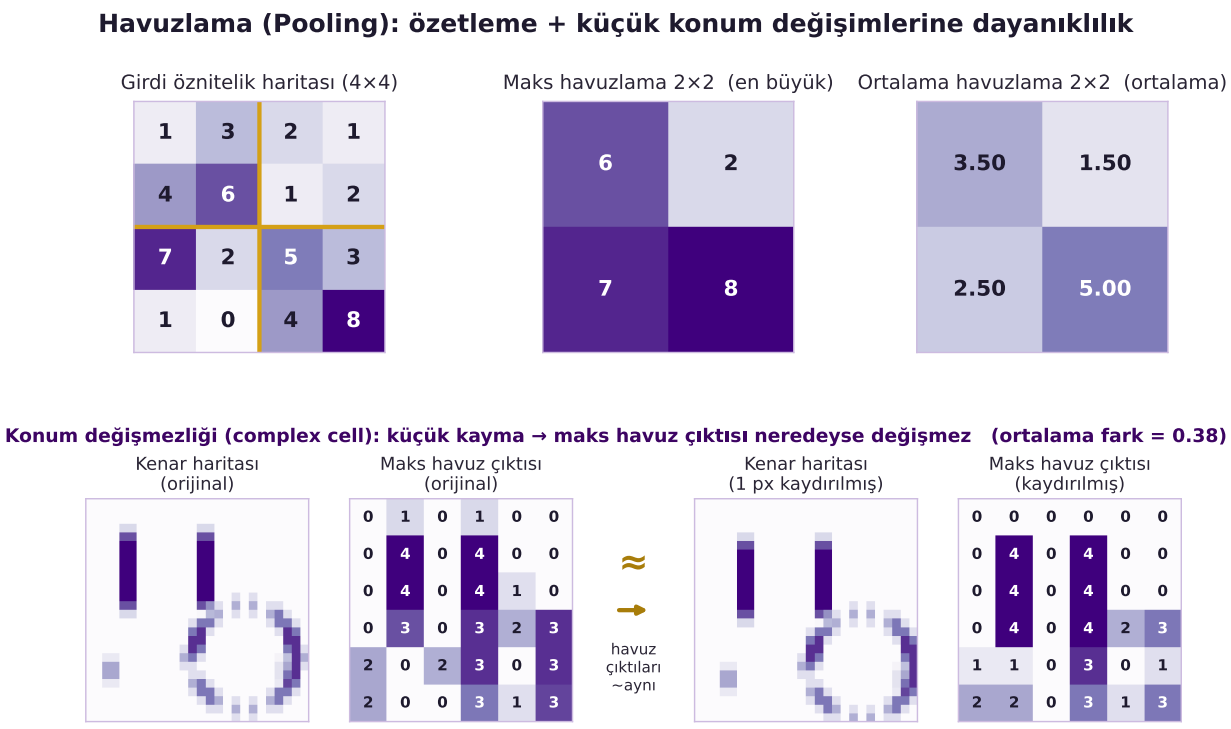
Şekil 10.4: ConvNet'in üç-işlem ritmi: her katman girdiyi Konvolüsyon (öznitelik tespiti) → ReLU (doğrusal-olmayan) → Havuzlama (alt-örnekleme) sırasıyla işler ve bu blok katmanlar boyunca tekrarlanır; çıktı boyutu o = ⌊(n−k)/s⌋ + 1 ile verilir (n=100, k=5, s=1 → 96).

Pooling'e biraz daha yakından. Pooling, küçük bir pencerede (örn. 2×2) öznitelikleri özetler — en yaygın iki yol **max pooling** (penceredeki en güçlü yanıtı al) ve **average pooling** (ortalamasını al). LeNet5 ortalama pooling kullanıyordu; modern ağlar çoğunlukla max tercih eder. Pooling'in iki işlevi var: (1) çözünürlüğü düşürerek bir sonraki katmanın **daha geniş bir alanı** görmesini sağlar (etkin receptive field büyür); (2) bir özniteliğin tam konumundaki küçük kaymalara **dayanıklılık (invariance)** katar — birazdan göreceğimiz "complex cell" fikrinin tam karşılığı. Yani conv "nerede" bilgisini korur (equivariance), pooling onu kasıtlı olarak biraz **bulanıklaştırır** (invariance); ikisinin dengesi, ConvNet'in hem konuma duyarlı hem konuma dayanıklı olmasını sağlar. Şekil 10.5 hem havuzlamanın özetlemesini hem de kaymaya dayanıklılığını sayısal olarak gösterir.

> Builder Notu — Conv-Nonlin-Pool Ritmi
>
> **Geriye (Hafta 2):** "Conv + nonlinearite" çifti, Hafta 2'nin "afin + nonlinearite" (döndür-ez) atomunun uzamsal versiyonudur — afin yerine ağırlık-paylaşımlı convolution.
> **İleriye:** Modern ağlarda pooling yerini çoğunlukla strided convolution'a bıraktı; ama "çıkar → büz → özetle" ritmi her görü mimarisinde sürer (ResNet, ConvNeXt).

10.5 (LeCun) Hiyerarşik Temsil ve Kompozisyonel Dünya

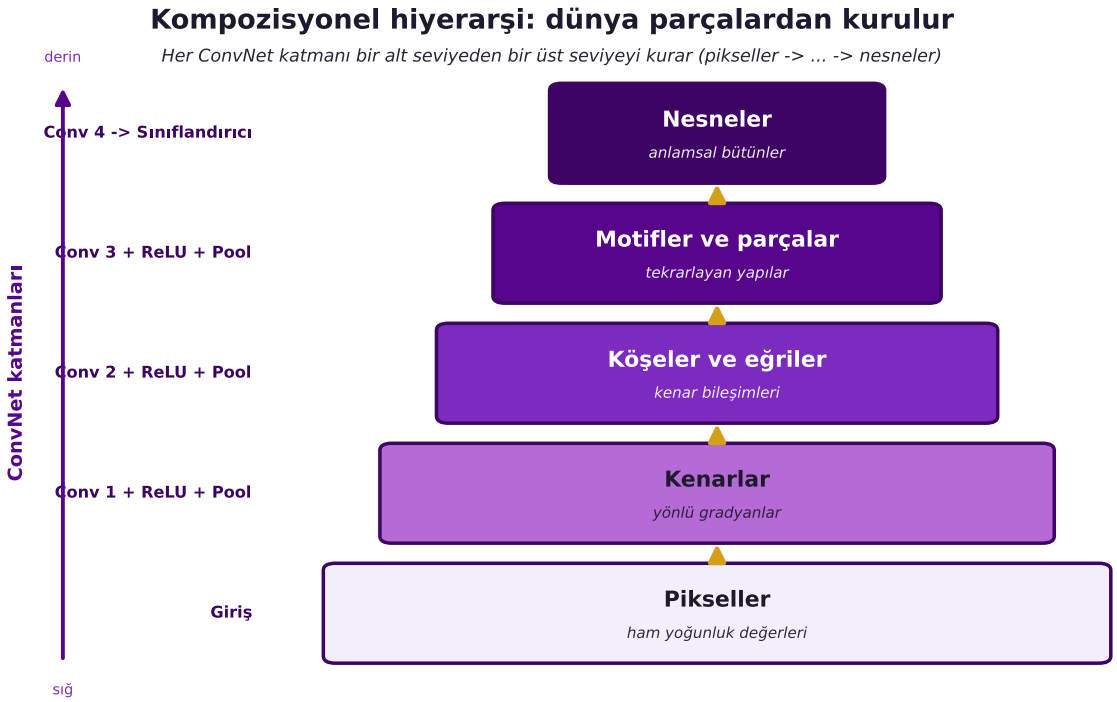
Neden katmanları **istifleriz**? Çünkü doğal dünya kompozisyoneldir: küçük parçalar birleşip daha büyük yapıları oluşturur.



Şekil 10.5: Havuzlama (pooling) iki işi yapar: üstte 4×4 öznitelik haritasının 2×2 maks ve ortalama havuzlamayla özetlenmesi, altta ise küçük bir piksel kaymasına rağmen maks havuz çıktısının neredeyse değişmeden kalması (ortalama fark ≈ 0.38) gösterilerek complex cell konum değişmezliği örneklenir.

> "we want to build hierarchical representations because the world is compositional… edges kind of assemble to form local features like corners and T-junctions [and so on]." — LeCun, 34:17

Yani: pikseller → kenarlar → köşeler/eğriler → motifler → nesne parçaları → nesneler. Her ConvNet katmanı bir alt seviyenin öznitelliklerini birleştirip bir üst seviyeyi kurar. LeCun'un vurgusu: bu hiyerarşi keyfi değil, **dünyanın kendi yapısını** yansıtır — ve belki de derin öğrenmenin neden bu kadar iyi çalıştığının sebebi budur. Şekil 10.6 bu piramidi pikselden nesneye kadar gösterir.



Şekil 10.6: Kompozisyonel hiyerarşi piramidi: bir ConvNet katman katman pikselleri kenarlara, kenarları köşelere/eğrilere, onları motiflere/parçalara ve en sonunda nesnelere dönüştürerek dünyanın parçalardan kurulu yapısını yeniden inşa eder.

> Builder Notu — Kompozisyonel Dünya
>
> **Geriye (Calculus + Hafta 1):** Katman istifleme = fonksiyon bileşkesi (Calculus zincir kuralı dünyası); hiyerarşik öznitelik, Hafta 1'in "düşük→orta→yüksek seviye özellik" sezgisinin somutlaşması.
> **İleriye:** "Kompozisyonel hiyerarşi" fikri, transfer learning'in (önceki katmanlar genel, son katmanlar göreve özel) ve foundation modellerin temel gerekçesidir.

10.6 (LeCun) Biyoloji: Görsel Korteks, Hubel-Wiesel ve Complex Cells

LeCun bu fikirlerin biyolojik kökenini anlatıyor. Görsel sinyal gözden beynin arkasındaki **birincil görsel kortekse (V1)** gider; oradan V2, V4, IT boyunca bir **hiyerarşi** (ventral pathway) izler.

> "this idea — hierarchy and local feature detection — comes from biology." — LeCun, 37:36

İki anahtar kavram, Hubel & Wiesel'in keşfi:

- **Receptive field (alıcı alan):** Bir nöronun yalnızca görsel alanın küçük bir bölgesine duyarlı olması. Bar o bölgenin dışına çıkınca nöron tepki vermez. Aynı işi yapan nöronlar tüm görsel alana **kopyalanmıştır** (bu, weight sharing'in biyolojik karşılığıdır).
- **Complex cells (karmaşık hücreler):** Bir öznitelliğin tam konumundaki küçük değişimlere **değişmez** (invariant) tepki verir — pooling'in biyolojik karşılığı.

Hubel & Wiesel aslında iki hücre tipi buldu: **simple cells** (basit hücreler) belirli bir konumda belirli bir yönelimdeki kenarlara tepki verir — convolution filtresinin biyolojik karşılığı; **complex cells** (karmaşık hücreler) ise aynı yönelimi *konumdan görece bağımsız* tanır (bir grup simple cell'in çıktısını özetler) — pooling'in karşılığı. Yani görsel korteksin "kenar bul, sonra konumu bulanıklaştır" ritmi, convolution + pooling ritminin ta kendisidir.

Bu fikirler 1980'lerde Fukushima'nın **neocognitron**'una, oradan LeCun'un ConvNet'ine ilham verdi. Neocognitron simple/complex cell katmanlarını taklit ediyordu ama backprop ile eğitilmiyordu; LeCun'un katkısı tam da bu yapıyı **uçtan uca gradient ile eğitilebilir** kılmaktı. Modelin bazı öznitelik dedektörleri elle tasarlanmış, bazıları öğrenilmişti — derin öğrenmenin "elle mühendislikten öğrenmeye" geçişinin (Hafta 1) görüdeki somut anı.

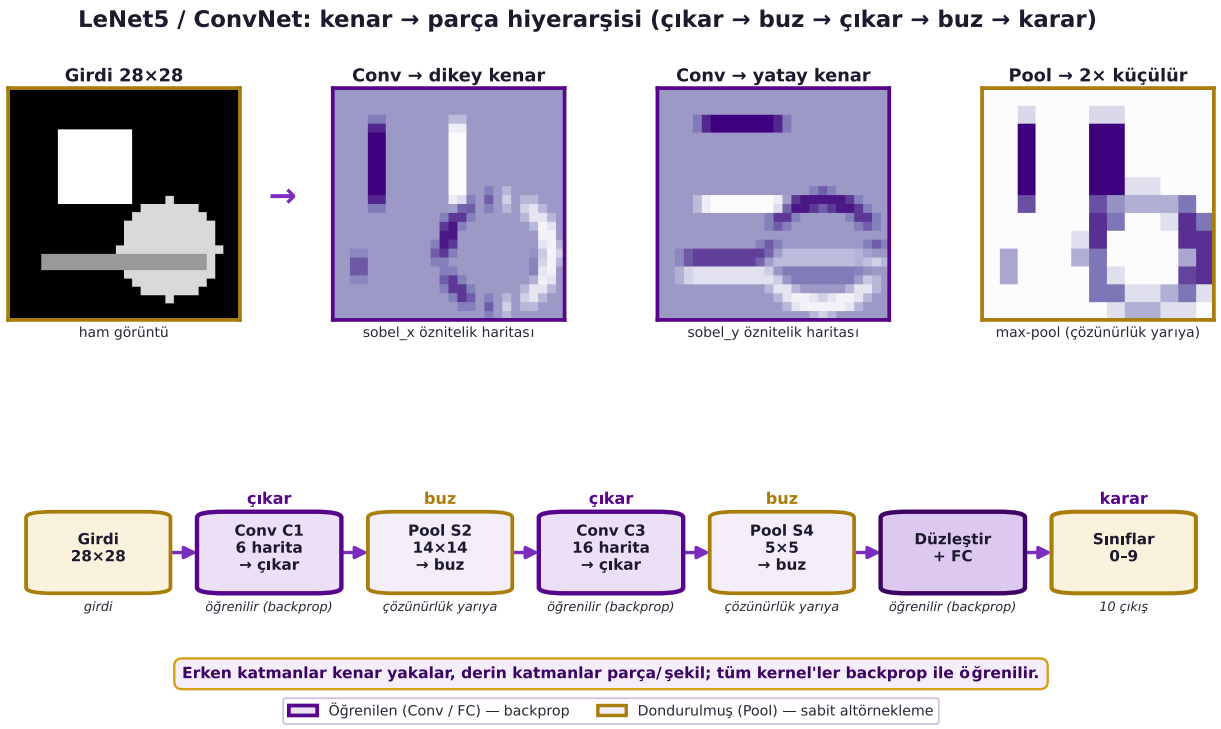
> Builder Notu — Görsel Korteks ↔ ConvNet
>
> **Geriye (§4.J):** Receptive field + replikasyon = **equivariance** (öteleme-eşdeğişkenlik, convolution); complex cell = **invariance** (pooling). §4.J'deki "equivariance vs invariance" ayrımının biyolojik kökü budur.
> **İleriye:** Görsel korteks ↔ ConvNet benzerliği, nörobilim-ML kesişiminin (NeuroAI) klasik örneğidir; ama modern ağlar artık biyolojiyi taklit etmekten çok mühendislik kısıtlarıyla şekillenir.

10.7 (LeCun) LeNet5: İlk ConvNet'ler

LeCun kendi tarihî ağını, **LeNet5**'i anlatıyor (el yazısı rakam tanıma). Yapı taşları bugünküyle aynı: strided convolution + pooling (2×2 ortalama) + nonlinearite, hiyerarşik istiflenmiş. Her katman bir **öznitelik haritaları (feature maps)** kümesi üretir; bir sonraki katman, önceki haritaların **kombinasyonlarını** algılamak için her biri farklı kernel'lerle convolve eder.

İlginç bir tarihsel ayrıntı: LeNet5'te öznitelik haritaları arasındaki bağlantı **tam değildi** — her harita önceki tüm haritalara bağlı değildi; belirli bir kombinasyon şeması vardı (hesabı azaltmak ve çeşitlilik için). Convolution kernel'ine bazen **filtre** de denir. En üstteki haritalar tek bir konuma indiğinde, bunlar artık çıktı sınıfları olur.

Akışı izlemek aydınlatıcı: giriş görüntüsü → convolution (birkaç öznitelik haritası) → pooling (çözünürlük yarıya) → tekrar convolution (önceki haritaların kombinasyonları, daha zengin öznitelikler) → tekrar pooling → … → en sonunda haritalar 1×1'e indiğinde tam-bağlı bir sınıflandırıcı. Yani LeNet5, "çıkar → büz → çıkar → büz → karar ver" ritmidir. Erken katmanlar basit kenarları, derin katmanlar nesne-benzeri parçaları yakalar — Bölüm 4'teki kompozisyonel hiyerarşinin somut hâli. Dikkat: tüm bu yapı, Hafta 2'deki aynı backprop ile uçtan uca eğitilir; "elle tasarım" yalnızca *mimaridedir* (kernel boyutu, katman sayısı), ağırlıklar öğrenilir. Şekil 10.7 bu akışı gerçek öznitelik haritalarıyla gösterir.



Şekil 10.7: LeNet5/ConvNet mimari akışı: girdi görüntüden conv katmanlarının ürettiği gerçek kenar öznitelik haritaları (üst), pooling ile çözünürlüğün yarıya inmesi ve çıkar→buz→çıkar→buz→karar ritmiyle düzleştirme-FC-sınıf akışı (alt); erken katmanlar kenar, derin katmanlar parça yakalar ve tüm kernel'ler backprop ile öğrenilir.

Builder Notu — LeNet5 Soyağacı

Geriye (Hafta 2): LeNet5 de aynı "modül + maliyet + backprop" iskeletiyle eğitilir; tek fark, lineer modüllerin yerini ağırlık-paylaşımlı convolution modüllerinin almasıdır.
İleriye: LeNet5 (1998) → AlexNet (2012, Hafta 1'deki "ikinci devrim") → VGG/ResNet. Çekirdek aynı kaldı; ölçek, derinlik ve donanım değişti.

10.8 Geçiş: LeCun'dan Canziani'ye

LeCun convolution'ın *neden*ini verdi: kompozisyonel dünya, biyolojik hiyerarşi, ağırlık paylaşımı, LeNet5. Şimdi **Canziani** bunu sistematikleştiriyor: convolution'ı "icat etmek" yerine, doğal sinyallerin hangi **özelliklerinin** convolution'ı *zorunlu kıldığını* gösteriyor. Üç kelime: stationarity, locality, compositionality — ve her biri bir mimari kararına çevriliyor.

10.9 (Canziani) Doğal Sinyallerin Üç Özelliği

Canziani, convolutional ağların neden bu kadar iyi olduğunu doğal sinyallerin üç özelliğine bağlıyor (görüntü, ses, metin — "doğada olduğu için doğal sinyal").

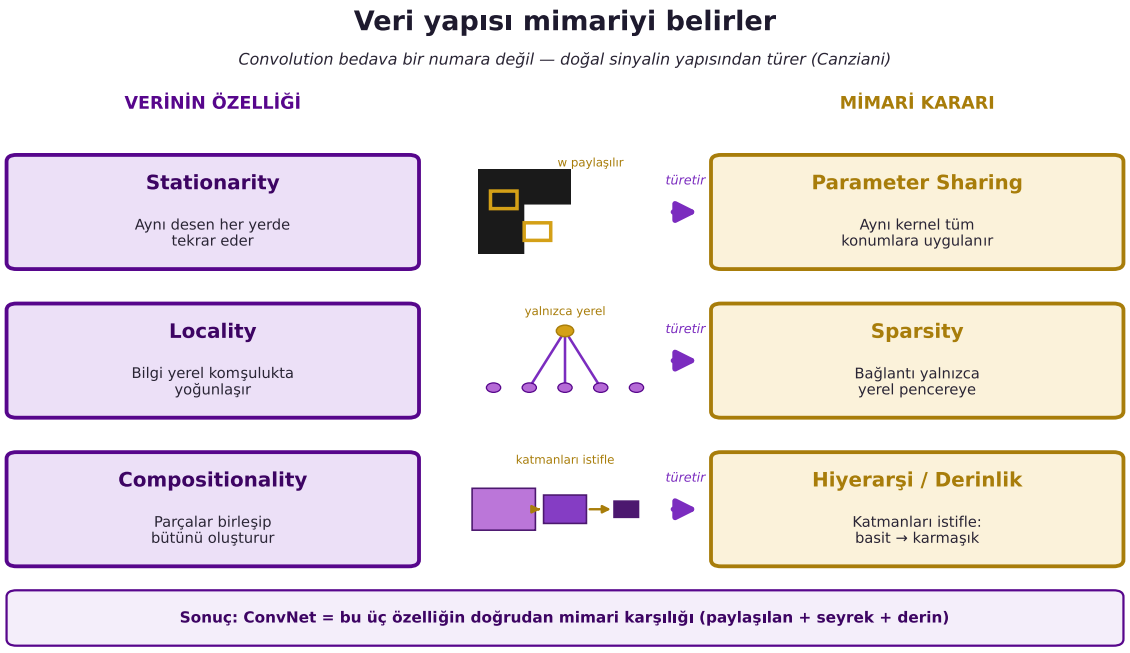
1. **Stationarity (durağanlık):** Aynı tür desen, sinyal boyunca tekrar tekrar görünür. Bir kenar görüntünün her yerinde aynı kenardır.
2. **Locality (yerellik):** Bilgi yereldir — anlam, yakın komşu örneklerde toplanır, uzaktakilerle değil.
3. **Compositionality (kompozisyonellik):** Dünya açıklanabilir biçimde parçalardan kurulur; küçük yapılar birleşip büyükleri oluşturur (LeCun'un hiyerarşisi).

> "the key words for understanding convolutions [are]: stationarity, locality, compositionality." — Canziani, 0:58

Canziani sezgiyi somutlaştırıyor: bir sinyali kendisiyle convolve edersen (flip + kaydır + iç çarpım) tekrar eden desenleri yakalarsın — durağanlığın matematiksel izi. Şekil 10.8 bu üç özelliğin her birini bir mimari karara nasıl eşlediğini tek şemada toplar.

Builder Notu — Üç Özellik = İnductive Bias

Geriye (Hafta 1): Compositionality, Hafta 1'deki manifold/hiyerarşi sezgisinin sinyal-yapısı diliyle ifadesidir; stationarity ve locality, görüntü verisinin neden 3-milyon-boyutlu uzayın minik bir bölgesinde yaşadığını açıklar.
İleriye: Bu üç özellik bir "tümevarımsal önyargı (inductive bias)" listesidir; bir veriye doğru mimariyi seçmek, o verinin hangi özellikleri taşıdığını sormakla başlar (Hafta 13 graf ağları, Hafta 8 Bishop'ın inductive bias tartışmasına köprü).



Şekil 10.8: Canziani'nin doğal-sinyal özelliklerini ConvNet mimari kararlarına eşleyen üç satırlı şema: Stationarity→Parameter Sharing, Locality→Sparsity, Compositionality→Hiyerarşi/Derinlik — convolution'ın verinin yapısından türediğini gösterir.

10.10 (Canziani) Özellikten Mimariye: Locality → Sparsity, Stationarity → Parameter Sharing

Canziani'nin en güçlü adımı: her özelliği bir mimari karara **çevirmek**.

- **Locality → sparsity (seyrek bağlantı).** Bilgi yerelse, bir çıktı nöronunun tüm girdiye bağlı olması gereksizdir; yalnızca küçük bir yerel pencereye bağlanır. Bu, tam-bağlı katmanın bağlantılarını dramatik azaltır.
- **Stationarity → parameter sharing (ağırlık paylaşımı).** Aynı desen her yerde görünüyorsa, onu yakalayan filtreyi her konumda yeniden öğrenmek gerekmez; aynı kernel tüm konumlarda paylaşılır.

Canziani mantığı tersinden de vurguluyor: özellik yoksa teknik de geçersizdir.

> "if my data doesn't show locality, can I use sparsity? No." — Canziani, 21:10

Yani sparsity ve parameter sharing "bedava" numaralar değil; doğrudan verinin yapısına dayanan, gerekçeli tasarım kararlarıdır. İkisi birlikte, tam-bağlı katmanı bir convolution katmanına dönüştürür.

> Builder Notu — Özellik → Mimari
>
> **Geriye (Hafta 2):** Tam-bağlı katman ($Wx + b$) her girdiyi her çıktıya bağlardı; sparsity bunu yerel pencereye indirir, parameter sharing aynı W'yi her konumda kullanır. Convolution = "kısıtlanmış" bir lineer katmandır.

İleriye: "Doğru inductive bias = daha az parametre + daha iyi genelleme" denklemi, model verimliliğinin (data efficiency) temelidir; yanlış bias ise (örn. görüntüye saf MLP) veri ve compute israfıdır.

10.11 (Canziani) Kernel'ler: 1B, 2B, 3B ve Boyutlar

Canziani kernel'i somutlaştırıyor: bir convolution katmanı bir **kernel topluluğudur**. 1B sinyalde (ses) kernel küçük bir ağırlık penceresidir; her kernel bağımsızdır, dolayısıyla paralel eğitilebilir. Çok kanallı durumda kernel boyutları şöyle okunur: kaç kernel (çıktı kanalı) × kalınlık (girdi kanalı) × pencere boyutu.

Örneğin 7 girdi kanalından 3 çıktı kanalı üreten 1B kernel'ler "2 kernel" gibi gruplanabilir; görüntüde (2B) kernel'ler 3 boyutlu olur (yükseklik × genişlik × girdi kanalı), ve katman bunlardan birden çok tutar. Her kernel bir öznitelik haritası üretir; haritalar üst üste binerek bir sonraki katmanın girdisi olur — LeCun'un LeNet5 anlattığı yapının ta kendisi.

Bu, hem **connection sparsity** (her çıktı yalnızca yerel pencereye bakar) hem **parameter sharing** (aynı kernel her konumda) sağlar; ikisi olmadan parametre sayısı yine patlardı.

Sayılarla. 32×32×3 bir görüntüyü düşün. Tam-bağlı bir katman 1024 çıktı üretmek isterse: $32 \cdot 32 \cdot 3 \cdot 1024 \approx$ **3,1 milyon** ağırlık. Aynı işi 16 adet 3×3 convolution kernel'i ile yaparsan: $16 \cdot (3 \cdot 3 \cdot 3) =$ **432** ağırlık (artı 16 bias). Yani convolution parametreyi binlerce kat azaltır — ve bu azalma "kalite kaybı" değil, doğru inductive bias'tır: parametreyi azaltmak aynı zamanda genelleme gücünü artırır, çünkü model verinin gerçek yapısına (yerel + durağan) uygun kısıtlanmıştır. Convolution çıktısının kanal sayısı kaç kernel kullandığınla belirlenir; her kernel girdinin tüm kanallarına bakar (kalınlık = girdi kanalı), tek bir öznitelik haritası üretir. Bu binlerce-kat azalma Şekil 10.1'de görselleştirilmiştir.

Builder Notu — Kernel Topluluğu

Geriye (18.06): Kernel topluluğu = bir dizi yerel lineer operatör; çok-kanallı convolution, kanallar üzerinde toplanan iç çarpımlardır (18.06 matris-tensör işlemi).
İleriye: Kernel sayısı (genişlik) ve katman sayısı (derinlik) ConvNet'in kapasite eksenleridir; 1×1 convolution (kanal karıştırma), depthwise/separable convolution gibi varyantlar verimlilik için bunları yeniden düzenler.

10.12 (Canziani) Pratik: PyTorch'ta Bir ConvNet

Tüm bu fikirler PyTorch'ta birkaç satıra iner. Bir convolution katmanı `nn.Conv2d`, pooling `nn.MaxPool2d`; geri kalan Hafta 2'nin aynısı (forward → loss → backward → step). LeNet5-benzeri minik bir ağ:

```
import torch
import torch.nn as nn

convnet = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3, padding=1), nn.ReLU(),  # 3 kanal -> 16 ozellik haritasi
    nn.MaxPool2d(2),                                         # cozunurluk yariya (invariance)
    nn.Conv2d(16, 32, kernel_size=3, padding=1), nn.ReLU(), # 16 -> 32: kombinasyonlar
```

```
    nn.MaxPool2d(2),
    nn.Flatten(),
    nn.Linear(32 * 8 * 8, 10),                          # son siniflandirici
)

x = torch.randn(64, 3, 32, 32)   # batch=64, 3x32x32 goruntu
y = convnet(x)                   # cikti: (64, 10) sinif skorlari
```

Dikkat edilecek noktalar: `nn.Conv2d(in_channels, out_channels, kernel_size)` ağırlık paylaşımını ve seyrek bağlantıyı otomatik yapar — sen yalnızca kanal sayılarını ve kernel boyutunu verirsin. `padding=1`, kenar etkilerini telafi eder (çıktı boyutunu korur). `Flatten` sadece en sonda, haritalar yeterince küçüldükten sonra çağrılır — Hafta 2'de yaptığımız erken düzleştirmenin (uzamsal yapı kaybı) tam tersi. Bu ağ da aynı `loss.backward()` + `optimizer.step()` döngüsüyle eğitilir; convolution sadece yeni bir modül tipidir.

Builder Notu — nn.Conv2d Pratiği

Geriye (Hafta 2): Bu ağ Hafta 2'nin eğitim döngüsünün birebir aynısıyla eğitilir — convolution bir modül olduğundan backprop (LeCun'un Jacobian zinciri) onun için de otomatik çalışır.
İleriye: `nn.Conv2d` + `nn.BatchNorm2d` + residual bağlantı = modern ResNet bloğunun iskeleti; `padding`, `stride`, `dilation` parametreleri pratikte alıcı alanı (receptive field) ve çözünürlüğü ayarlamanın araçlarıdır.

10.13 Bu Dersin Özeti

1. **Tam-bağlı ağ görüntüde yetmez:** parametre patlar, uzamsal yapı kaybolur (Hafta 2 köprüsü).
2. **Convolution = kayan pencere (kernel) + ağırlık paylaşımı.** Aynı filtre tüm görüntüde gezer; iç çarpım hesaplar.
3. **ConvNet = convolution + nonlinearite + pooling** (opsiyonel), hiyerarşik istiflenir; stride alt-örnekleme yapar.
4. **Hiyerarşi, dünyanın kompozisyonelliğinden gelir:** pikseller → kenarlar → köşeler → nesneler.
5. **Biyolojik kök:** görsel korteks V1-V2-V4-IT; simple cell = conv (equivariance), complex cell = pooling (invariance); Hubel-Wiesel → neocognitron → LeNet5.
6. **Mimari, verinin yapısından doğar (Canziani):** locality → sparsity, stationarity → parameter sharing, compositionality → hiyerarşi/derinlik.

Tek Bir Cümle

Convolution, doğal sinyallerin yerel (locality), tekrar eden (stationarity) ve kompozisyonel yapısını sömüren, ağırlık-paylaşımlı bir kayan-filtre işlemidir; ConvNet bu filtreleri nonlinearite ve pooling ile hiyerarşik istifleyerek — tıpkı görsel korteks gibi — görüntüden öznitelikleri uçtan uca öğrenir.

10.14 Kontrol Soruları

i Soru 1: Bir görüntüyü tam-bağlı (nn.Linear) bir katmana vermek neden kötü bir fikirdir? Convolution iki sorunu nasıl çözer?

Cevap: İki sorun: (1) **parametre patlaması** — 1000×1000×3 girdiyi düzleştirip 1000 nörona bağlamak ~3 milyar ağırlık ister; (2) **uzamsal yapı kaybı** — düzleştirme komşu piksellerin komşuluğunu yok eder. Convolution ikisini de çözer: **parameter sharing** (aynı küçük kernel her konumda) parametreyi binlere indirir; **connection sparsity** (her çıktı yalnızca yerel pencereye bakar) uzamsal yapıyı korur. Convolution aslında ağırlıkları kısıtlanmış (paylaşımlı + seyrek) bir lineer katmandır.

i Soru 2: Convolution çıktı boyutu formülünü yaz. n = 32, k = 5, s = 1 için çıktı kaç olur? Stride'ı neden artırırsın?

Cevap: Çıktı boyutu:

$$o = \left\lfloor \frac{n-k}{s} \right\rfloor + 1$$

$n = 32$, $k = 5$, $s = 1 \rightarrow o = \lfloor 27/1 \rfloor + 1 = 28$. Stride'ı artırmak ($s > 1$) çıktıyı küçültür: hem hesabı/belleği azaltır hem de pooling gibi alt-örnekleme (downsampling) yapar — çözünürlüğü düşürüp daha soyut, daha geniş kapsamlı öznitelikler kurar.

i Soru 3: Doğal sinyallerin üç özelliği nedir ve her biri hangi mimari karara çevrilir?

Cevap: (1) **Stationarity** (aynı desen her yerde tekrar eder) → **parameter sharing** (aynı kernel her konumda); (2) **Locality** (bilgi yereldir) → **sparsity** (her çıktı yalnızca yerel pencereye bağlanır); (3) **Compositionality** (parçalar büyükleri oluşturur) → **hiyerarşi/derinlik** (katmanları istifle). Canziani'nin uyarısı: özellik yoksa teknik geçersizdir — "data locality göstermiyorsa sparsity kullanamazsın" (21:10). Yani convolution, verinin yapısına dayanan gerekçeli bir tasarımdır.

i Soru 4: (Builder) Receptive field ve complex cell kavramları ConvNet'in hangi iki özelliğine karşılık gelir? §4.J terimleriyle bağla.

Cevap: Hubel-Wiesel'in **receptive field**'ı (nöron yalnızca yerel bir bölgeye duyarlı, ve aynı dedektör tüm görsel alana kopyalanmış) = convolution'ın **equivariance**'ı (öteleme-eşdeğişkenlik): nesne nerede olursa olsun aynı kernel onu yakalar, çıktı da konumla birlikte kayar. **Complex cell** (öznitelliğin küçük konum değişimlerine değişmez tepkisi) = **pooling**'in sağladığı **invariance** (değişmezlik). §4.J: equivariance = konumla birlikte değişir; invariance = konumdan bağımsız. ConvNet ikisini de kullanır: conv equivariant, pooling invariant.

10.15 Egzersizler

Egzersiz 1 (Convolution elle). 1B bir sinyal $x = [1, 2, 3, 4, 5]$ ve kernel $w = [1, 0, -1]$ için convolution (cross-correlation) çıktısını elle hesapla, sonra `torch.nn.functional.conv1d` ile doğrula. Çıktı boyutunu

formülle önceden tahmin et.

```
import torch
import torch.nn.functional as F
x = torch.tensor([1., 2., 3., 4., 5.]).view(1, 1, -1)   # (batch, kanal, uzunluk)
w = torch.tensor([1., 0., -1.]).view(1, 1, -1)          # tek kernel
print(F.conv1d(x, w).flatten())                          # -> [-2, -2, -2]
```

Egzersiz 2 (Parametre karşılaştırması). 32×32×3 girdi için: (a) 1024 çıktılı tam-bağlı katman kaç parametre? (b) 16 adet 3×3 kernel'li convolution katmanı kaç parametre? Aradaki farkı yorumla — parameter sharing + sparsity ne kazandırdı?

Egzersiz 3 (Öteleme-değişmezlik). Bir görüntüdeki nesneyi birkaç piksel kaydır. (a) Tam-bağlı bir ağda çıktı nasıl değişir? (b) Bir conv+pooling ağında? `nn.Conv2d` + `nn.MaxPool2d` ile küçük bir deney kur ve equivariance/invariance farkını gözlemle.

Egzersiz 4 (Üç özellik). Şu üç veriden hangisi locality/stationarity gösterir, hangisi göstermez, neden: (a) doğal görüntü, (b) tablo verisi (her sütun farklı bir özellik), (c) ses dalgası? Hangisi için convolution uygundur?

Egzersiz 5 (Hafta 4 habercisi). Convolution'ı bir matris çarpımı olarak yazabilir misin? (a) Küçük bir 1B convolution'ı, ağırlıkları kaydırarak doldurulmuş bir **Toeplitz matrisi** ile ifade et. (b) Bu, convolution'ın aslında "yapısı kısıtlanmış bir lineer katman" olduğunu gösterir — Hafta 4'te Canziani convolution'ı tam olarak lineer cebir diliyle (Toeplitz/dolaşım matrisleri) ele alacak. Neden bu bakış faydalı?

10.16 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H4: Konvolüsyonun Cebiri (Toeplitz) ve Optimizasyon

Convolution'dan lineer cebire. Bu hafta convolution'ın *ne* olduğunu ve *neden* gerektiğini gördük. Hafta 4'te Canziani convolution'ı **lineer cebir** diliyle (Toeplitz/dolaşım matrisleri olarak) açacak — Egzersiz 5'in "convolution = yapısı kısıtlanmış lineer katman" sezgisi tam burada formelleşir; LeCun ise optimizasyonu derinleştirecek (SGD'nin ötesinde Adam, momentum, normalization katmanları).

Hafta 4: Konvolüsyonun Cebiri ve Optimizasyon I-II — Canziani (Lecture) + LeCun (Lecture)

Bu hafta convolution'ın *ne* olduğunu ve *neden* gerektiğini gördük. Hafta 4'te Canziani convolution'ı **lineer cebir** diliyle (Toeplitz/dolaşım matrisleri olarak) açacak; LeCun ise optimizasyonu derinleştirecek (SGD'nin ötesinde Adam, momentum, normalization katmanları).

Hafta 4 öncesi yapılacak:

- Egzersiz 2 (parametre karşılaştırması) ve Egzersiz 5 (convolution = Toeplitz matris) çöz.
- "Stationarity → parameter sharing, locality → sparsity" eşlemesini kendi sözcüklerinle yaz.
- Hafta 2'nin SGD'sini hatırla — Hafta 4'te onu Adam ve normalization ile geliştireceğiz.

10.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Weight sharing	Aynı ağırlıkları her konumda kullanmaya zorlama	LeCun 12m06
Convolution	Kernel'i girdide kaydırıp her konumda iç çarpım	LeCun 26m49
Kernel / filtre	Convolution'ın küçük, paylaşılan ağırlık penceresi	LeCun 29m50
ConvNet üç işlem	convolution + nonlinearite + pooling (opsiyonel)	LeCun 30m24
Stride	Kernel'in kaydırma adımı; > 1 ise alt-örnekleme	LeCun 30m53
Kompozisyonel hiyerarşi	Kenarlar → köşeler → nesneler; derinliğin sebebi	LeCun 34m17
Receptive field	Nöronun duyarlı olduğu yerel bölge; equivariance	LeCun 44m15
Simple / complex cell	Simple = conv (kenar), complex = pooling (invariance)	LeCun 45m29
LeNet5 / feature map	İlk ConvNet; her kernel bir öznitelik haritası üretir	LeCun 49m05
Stationarity	Aynı desen her yerde tekrar eder → parameter sharing	Canziani 4m29
Locality	Bilgi yereldir → sparsity (seyrek bağlantı)	Canziani 6m53
Compositionality	Parçalar büyükleri kurar → hiyerarşi/derinlik	Canziani 11m58

10.18 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Convolution = kayan iç çarpım** → 18.06 dot product / lineer operatör + Hafta 1 filtre·yama.
2. **Hiyerarşi = fonksiyon bileşkesi** → Calculus zincir kuralı + Hafta 1 öznitelik hiyerarşisi.
3. **equivariance / invariance** → §4.J + 18.06 öteleme-değişmez operatör.
4. **ConvNet eğitimi** → Hafta 2 (modül + maliyet + backprop); convolution da bir modüldür.
5. **Kernel topluluğu = kanal üzerinde toplanan iç çarpımlar** → 18.06 matris-tensör.

İleriye köprüler (production / research):

1. **LeNet5 omurgası** → AlexNet, VGG, ResNet, ConvNeXt.
2. **inductive bias (üç özellik)** → geometric deep learning, transformer tasarım felsefesi.
3. **stride / pooling** → modern strided-conv downsampling, U-Net encoder-decoder.
4. **1×1 / depthwise convolution** → verimli mimariler (MobileNet, EfficientNet).

! Bu dersten tek bir şey alıp gideceksen

Convolution sihir değildir — doğal sinyallerin üç özelliğinin (stationarity, locality, compositionality) doğrudan mimariye çevrilmesidir: aynı küçük filtreyi her yerde paylaş (stationarity → parameter sharing), yalnızca yerele bak (locality → sparsity), katmanları istifle (compositionality → hiyerarşi). LeCun bunu kompozisyonel dünya ve görsel korteksle motive eder, Canziani üç özellikten mimariyi türetir — ve LeNet5'ten bugünkü görü modellerine giden yol işte bu çekirdeğin üstünde durur.

11 Konvolüsyonun Cebiri ve Optimizasyon (SGD, Momentum, Adam)

NYU'nun bu haftaki iki sesi: Alfredo Canziani convolution'a lineer cebir gözüyle döner — convolution aslında *özel yapılı (Toeplitz) bir matrisle çarpımdır*, yani 'bir sürü sıfırı olan bir matris çarpımı'; sonra konuk araştırmacı Aaron Defazio (Facebook AI Research) sahneyi devralır ve o matrisleri (ağı) *nasıl* eğittiğimizin pratik motorunu anlatır — gradient descent'ten SGD'ye, momentum'a, ve modern adaptif yöntemlere (RMSprop, Adam)

Bölüm bilgisi

- **Canziani'nin Lecture videosu:** YouTube — Convolution as linear algebra (Toeplitz) (≈51 dk)
- **Konuk Lecture (Aaron Defazio):** YouTube — Optimization for deep learning (≈89 dk)
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Alfredo Canziani (convolution cebiri) + **Aaron Defazio** (konuk, optimizasyon — FAIR)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈30 dk

11.1 Bu Derste Ne Var?

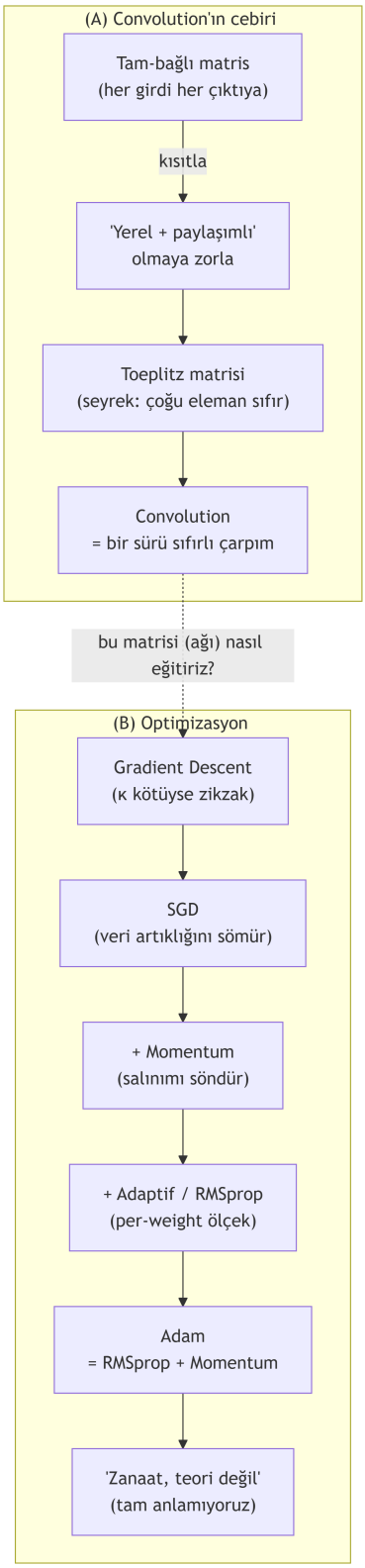
Bu hafta iki bağımsız ama tamamlayıcı parça var. Önce **Alfredo Canziani** convolution'a Hafta 3'te bıraktığımız soruyla geri döner: convolution aslında *nedir* — lineer cebir diliyle? Cevap zarif: convolution, **özel yapılı (Toeplitz) bir matrisle çarpmaktır** — yani "bir sürü sıfırı olan bir matris çarpımı". Sonra konuk bir optimizasyon araştırmacısı, **Aaron Defazio** (Facebook AI Research; PhD Australian National University) sahneyi devralır ve ağları *nasıl* eğittiğimizin pratik motorunu anlatır: gradient descent'ten SGD'ye, momentum'a, ve modern adaptif yöntemlere (RMSprop, Adam).

> ⚠ **Atıf notu:** Bu haftanın optimizasyon dersi (Lecture) **LeCun değil, konuk Aaron Defazio** tarafından verilir (transkriptten doğrulandı: "research scientist at Facebook working on optimization, PhD ANU"). Quote'lar bu yüzden **— Defazio** ile işaretlenir.

Bu haftanın üç ana fikri:

1. **Convolution = Toeplitz matrisiyle çarpım.** Tam-bağlı katmanla aynı çatı; tek fark, matrisin paylaşımlı + seyrek (çoğu sıfır) olmasıdır.
2. **SGD + momentum, modern eğitimin belkemiğidir.** Tam-batch gradient descent neredeyse hiç kullanılmaz; momentum'u ($\beta \approx 0.9$) "neredeyse her zaman" açarsın.

3. **Adaptif yöntemler (Adam) per-weight öğrenme oranı verir** — ve en çok kullanılan yöntem olmasına rağmen teorisi tartışmalıdır (“optimization’ı tam anlamıyoruz”).



Builder Notu — İki Yarı: Cebir ve Motor

Geriye (önkoşul kurslar):

- **Convolution = Toeplitz matris** → 18.06 matris-vektör çarpımı + Hafta 3 Egzersiz 5 (convolution = kısıtlı lineer katman) + Phase 2 18.065 yapılı matrisler.
- **Condition number** $\kappa = L/\mu$ → 18.06 özdeğer (en büyük/en küçük) + 18.065 koşullanma.
- **Momentum** → Calculus ivme (ikinci türev sezgisi) + fizik (Newton).

İleriye (production / research):

- Toeplitz görüşü → convolution'ın FFT ile hızlı hesabı, structured matrices.
- SGD+momentum / Adam → her modern eğitim hattının optimizer seçimi; LR schedule, warmup.

Tek cümleyle: Convolution özel yapılı bir matris çarpımıdır; ve o matrisi (ağı) eğitmek, gradient descent'i SGD + momentum + adaptif ölçeklemeyle pratiğe döken — ama teorisi hâlâ tam oturmamış — bir optimizasyon zanaatıdır.

11.2 (Canziani) Lineer Cebir Recap: Matris × Vektör'ün İki Görüşü

Canziani convolution'ı kurmadan önce matris-vektör çarpımını iki farklı gözle okutuyor. Afin dönüşüm $z = Ax$ (bias'ı matrisin içine gömerek). Bu çarpıma iki türlü bakılır:

1. **Satır görüşü:** çıktının her elemanı, A'nın bir **satırı** ile girdinin iç çarpımıdır — yani "girdinin o satırla ne kadar hizalı olduğu". Canziani'nin vurgusu: bir lineer katmanda **kernel, matrisin tam bir satırıdır**.

   "whenever you have a linear layer, your kernel is going to be the whole row of the matrix." — Canziani, 13:31

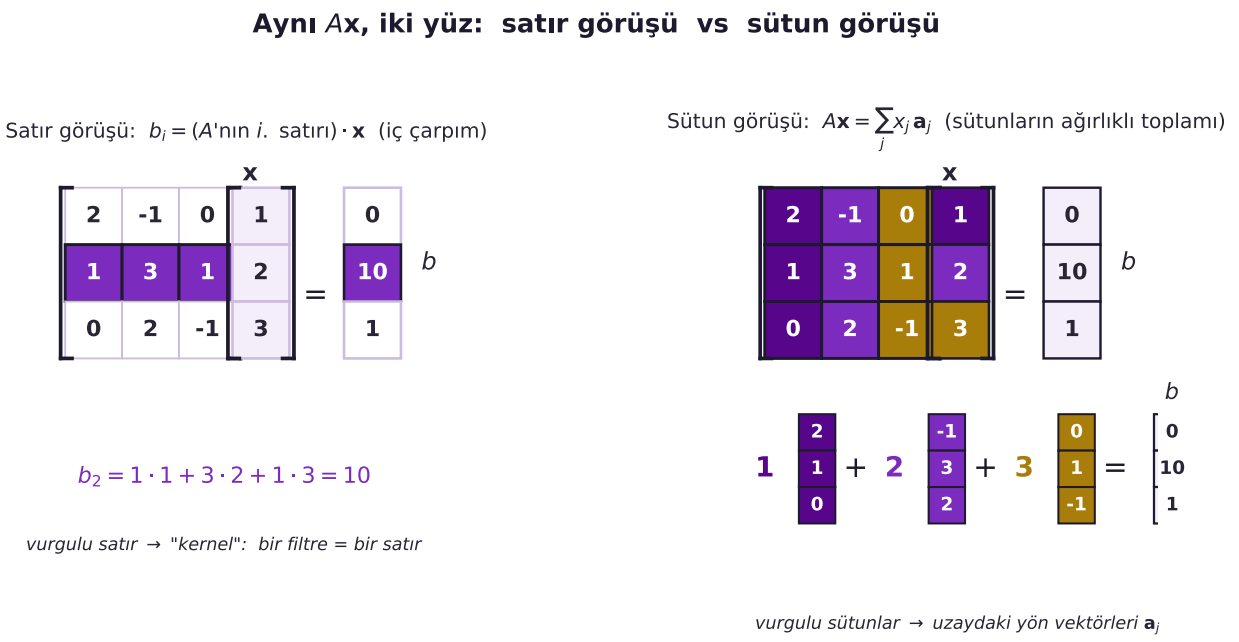
2. **Sütun görüşü:** çıktı, A'nın **sütunlarının** girdi katsayılarıyla ağırlıklı toplamıdır:

$$A\mathbf{x} = \sum_j x_j \, \mathbf{a}_j$$

Bu iki görüş aynı işlemin iki yüzü; convolution'ı anlamak için birincisi (satır = kernel) kritiktir. Şekil 11.1 aynı 3×3 sayısal $A\mathbf{x}$ çarpımını iki panelde yan yana koyar: solda satır·girdi iç çarpımı (vurgulu satır = kernel), sağda sütunların **x** katsayılarıyla ağırlıklı toplamı.

Builder Notu — Satır = Kernel

Geriye (18.06): Bu, 18.06'nın "matris çarpımının satır vs sütun yorumu" dersidir. Satır görüşü = iç çarpım/projeksiyon; sütun görüşü = sütun uzayı (column space). İkisi de aynı $A\mathbf{x}$.
İleriye: "Kernel = satır" görüşü, convolution'ı tam-bağlı katmanın özel bir hâli olarak görmenin anahtarıdır (Bölüm 3).



Şekil 11.1: Aynı $A\mathbf{x}$ çarpımının iki yüzü. SOL (satır görüşü): çıktının her elemanı b_i, A'nın i. satırı ile $\mathbf{x}$'in iç çarpımıdır; vurgulu satır convolution'daki "kernel" rolündedir — bir filtre tam olarak bir satıra karşılık gelir. SAĞ (sütun görüşü): $A\mathbf{x} = \sum_j x_j \mathbf{a}_j$, yani A'nın sütunlarının $\mathbf{x}$ ağırlıklarıyla toplamıdır; sütunlar uzaydaki yön vektörleridir. Aynı sonuç ($b = [0, 10, 1]$), iki farklı okuma — convolution sezgisi için "kernel = satır" yorumu kritiktir.

11.3 (Canziani) Convolution = Toeplitz Matrisi

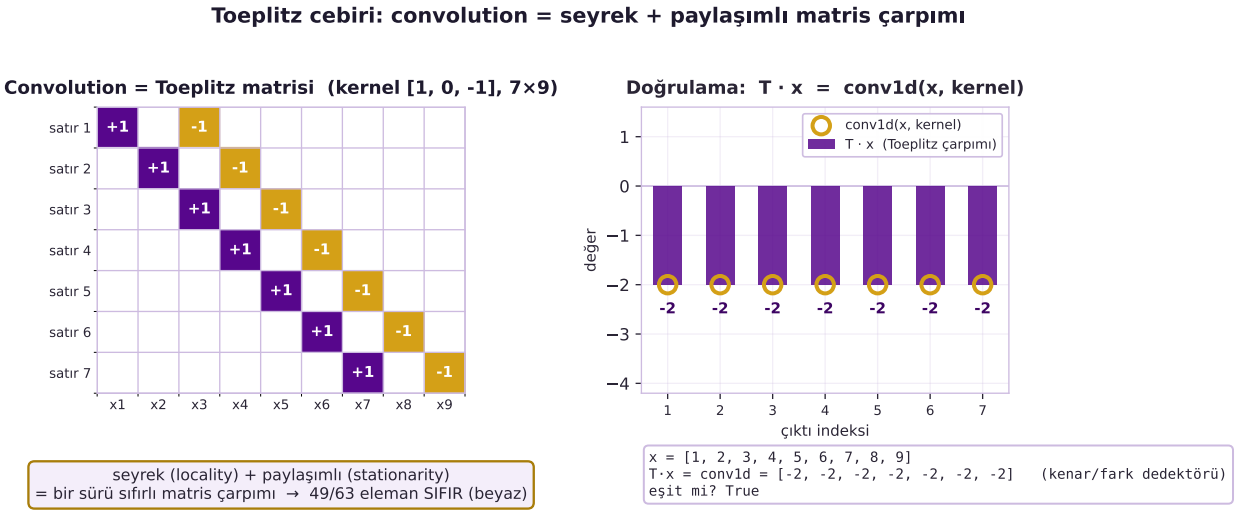
Şimdi convolution'ı bir matrise çeviriyoruz. Diyelim küçük bir kernel (boyut 3) ile çok uzun bir sinyali convolve edeceğiz. Bunu bir matris çarpımı olarak yazmak istersek, kernel'i her satırda **bir konum kaydırarak** yerleştiririz; geri kalan her yer **sıfırdır**. Ortaya çıkan matrisin özel bir adı var:

> "this is going to be called a Toeplitz matrix." — Canziani, 23:36

Toeplitz matrisi: her köşegeni sabit olan, kernel'in kaydırılarak tekrarlandığı matris. İki temel özelliği:

- **Seyrek (sparse):** her satırda yalnızca kernel boyutu kadar sıfır-olmayan eleman var (gerisi sıfır) — bu, **locality** (Hafta 3) demektir.
- **Paylaşımlı:** her satır aynı kernel'i içerir, sadece kaydırılmış — bu, **stationarity / parameter sharing** demektir.

Yani convolution'ı icat etmemize gerek yok: tam-bağlı bir katman alıp matrisini "yerel + paylaşımlı" olmaya zorlarsan, kendiliğinden convolution'ı yeniden keşfedersin. Şekil 11.2 bu fikri somutlaştırır: solda kernel $[1, 0, -1]$'in 7×9 Toeplitz matrisi (elemanların çoğu sıfır = beyaz), sağda $T\mathbf{x}$ ile doğrudan `conv1d`'nin birebir aynı kenar/fark çıktısını verdiği doğrulama.



Şekil 11.2: 1B convolution'ın Toeplitz matrisi olarak yazılışı. Solda kernel [1, 0, −1]'in n=9 girdi için ürettiği 7×9 Toeplitz matrisi: her satır kernel'i bir kaydırılmış kopyasıdır (violet +1, gold −1), 63 elemanın 49'u sıfır (beyaz). Bu seyreklik (locality, yereldelik) ve satır-paylaşımı (stationarity, durağanlık) convolution'ı çok sayıda sıfır içeren bir matris çarpımına indirger. Sağda doğrulama: T·x ile conv1d(x, kernel) birebir aynı kenar/fark çıktısını ([−2]×7) verir.

> **Builder Notu — Toeplitz = Kısıtlı Matris**
>
> **Geriye (Hafta 3 + 18.06):** Bu, Hafta 3 Egzersiz 5'in cevabıdır: convolution = Toeplitz (yapısı kısıtlanmış) matris. Hafta 3'ün "locality → sparsity, stationarity → parameter sharing" eşlemesi, Toeplitz matrisin sıfır deseni + tekrar deseni olarak somutlaşır.
> **İleriye:** Toeplitz/dolaşım (circulant) matrisler **Fourier dönüşümüyle köşegenleşir** — convolution'ın FFT ile $O(n \log n)$ hesabının temeli budur (18.065 spectral köprüsü, Hafta 13).

11.4 (Canziani) "Bir Sürü Sıfırlı Matris Çarpımı"

Canziani'nin özlü kapanışı:

> "a convolution is just a matrix multiplication with a lot of zeros — that's it." — Canziani, 27:55

Tam-bağlı katman ile convolution katmanı **aynı çatıdadır** (ikisi de $A\mathbf{x}$). Tek fark:

- **Tam-bağlı:** kernel = matrisin **tam satırı** (her girdiye bağlı, paylaşım yok) → çok parametre.
- **Convolution:** kernel = küçük, kaydırılan pencere (Toeplitz) → seyrek + paylaşımlı → az parametre.

Canziani "listening to convolutions" demosunda bir ses sinyalini bir kernel'le convolve edip sonucu dinletir — convolution'ın somut, işitilebilir etkisini gösterir. Özet: compositionality + locality + stationarity sayesinde matris-vektör çarpımını seyrek, yerel, paylaşımlı bir Toeplitz matrise indirgeyerek **convolution operatörünü yeniden keşfederiz**.

Builder Notu — Sıfırlı Matris Çarpımı

Geriye (Hafta 2-3): Convolution bir modül olduğundan (Hafta 2), backprop onun için de otomatik çalışır; Toeplitz görüşü, convolution'ın geri geçişinin de yine bir convolution (transpoze kernel) olduğunu açıklar — "forward W, backward $W^\top$" (Hafta 2) convolution'a taşınır.
İleriye: "Convolution = yapılı matris" görüşü, donanımda neden convolution'ın özel kernel'lerle (im2col + GEMM, ya da Winograd/FFT) hızlandırıldığını açıklar.

11.5 Geçiş: Canziani'den Defazio'ya (Konuk Optimizasyon Dersi)

Convolution'ın *ne* olduğunu (yapılı matris) gördük. Şimdi konu değişiyor: bu matrisleri (ağı) **nasıl eğitiriz**? Bu haftanın ikinci yarısı, konuk bir optimizasyon araştırmacısı **Aaron Defazio**'ya ait. Hafta 2'de SGD'yi tanımıştık; Defazio onu derinleştirip pratikte gerçekten kullandığımız yöntemlere (momentum, Adam) götürüyor. Üslubu açıkça pratik:

> "optimization is at the heart of machine learning… I'm going to focus on the application of these methods rather than the theory, part of the reason is that we don't fully understand all of these methods." — Defazio, 0:45

11.6 (Defazio) Gradient Descent ve Condition Number

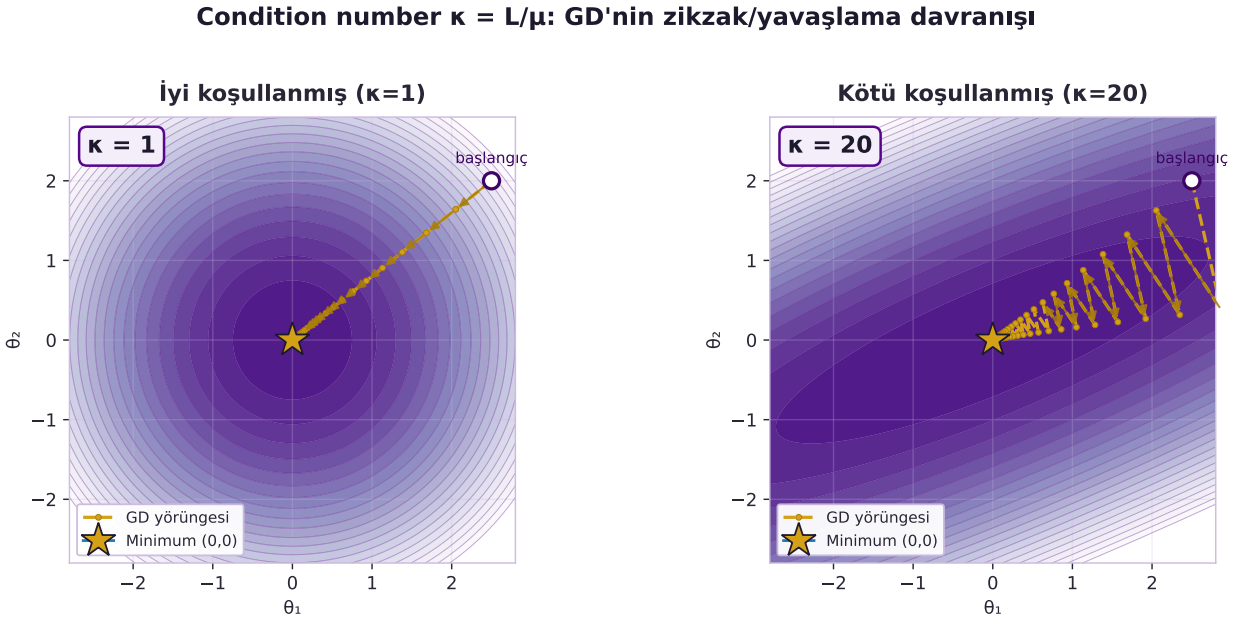
Defazio gradient descent'i hatırlatıp asıl sorunu gösteriyor: yakınsama hızı, problemin **koşullanmasına (conditioning)** bağlıdır. Kuadratik bir problemde bu, Hessian'ın en büyük (L) ve en küçük (μ) özdeğerlerinin oranıyla ölçülür — **condition number**:

$$\kappa = \frac{L}{\mu}$$

κ büyükse ($\kappa = \frac{L}{\mu}$ kötü koşullanmış) gradient descent zikzak çizer ve yavaşlar. Learning rate seçimi de bir gerilimdir: mümkün olduğunca büyük istersin (hızlı öğrenme) ama çok büyükse ıraksar — yani sürekli "ıraksamanın eşiğinde" çalışırsın.

> "we want to use learning rates as large as possible to get as quick learning as possible, so we're always at the edge of divergence." — Defazio, 13:42

İlginç bir gözlem: "güzel davranan" (salınmayan) düşük bir learning rate, çoğu zaman çözümden daha *uzakta* bırakır; iyi sonuç için rahatsız edici derecede yüksek oranlarla yaşamak gerekir. Şekil 11.3 iki uçta aynı GD'yi gösterir: solda $\kappa = 1$ (yuvarlak çukur, düz iniş), sağda $\kappa = 20$ (uzun ince vadi, zikzak).



Şekil 11.3: Condition number κ = L/μ ve gradient descent davranışı. SOL panel (κ=1, iyi koşullanmış): kayıp yüzeyi yuvarlak bir çukurdur; GD yörüngesi başlangıçtan (2,5, 2,0) minimuma (0,0) neredeyse düz iner — her adım doğru yöne bakar. SAĞ panel (κ=20, kötü koşullanmış): aynı kayıp uzun ve ince bir vadiye dönüşür; GD vadinin dik duvarları arasında zikzak çizerek minimuma yavaş yaklaşır. Mesaj: κ büyüdükçe (öz değerlerin oranı bozuldukça) düz gradient descent zikzaklaşır ve yavaşlar — Defazio'nun momentum/adaptif yöntemleri motive eden temel gözlemi.

Builder Notu — κ = Koşullanma

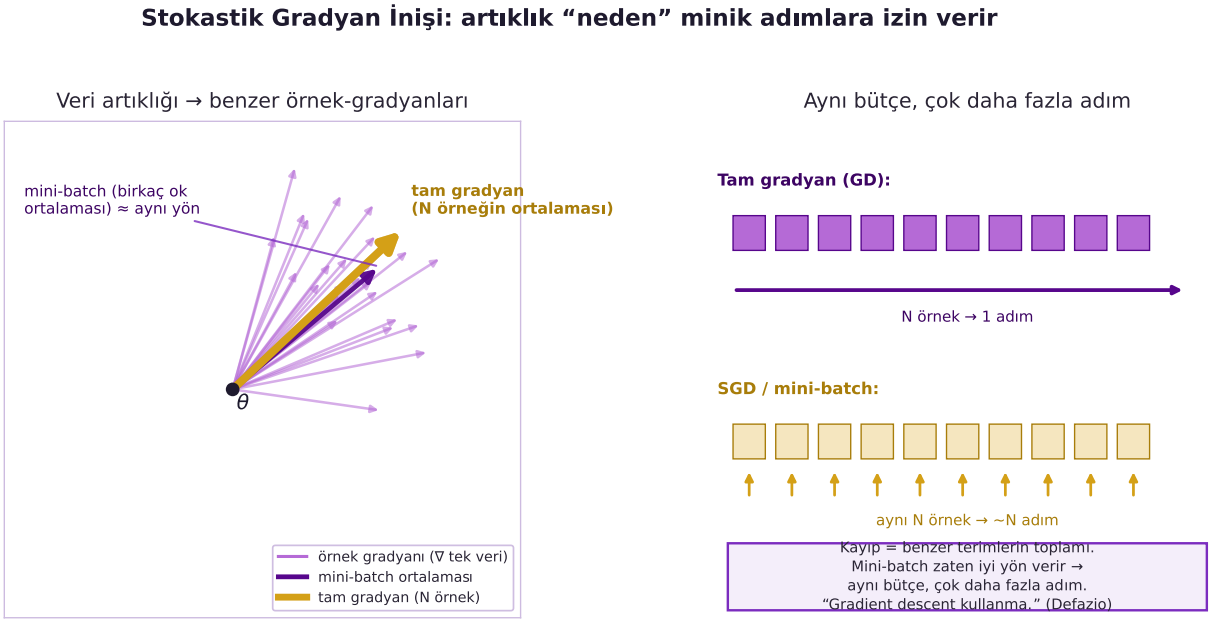
Geriye (18.06 + 18.065): Condition number $\kappa = L/\mu$ doğrudan 18.06 özdeğer dünyasıdır; kötü koşullanma, özdeğerlerin çok farklı ölçeklerde olması demektir (uzun, ince vadi). 18.065 (Matrix Methods) bunu ML bağlamında ele alır.
İleriye: Learning rate, en kritik hyperparameter'dır; warmup + decay schedule, "ıraksama eşiği"nde güvenli kalmanın pratiğidir.

11.7 (Defazio) SGD: "Gradient Descent Kullanma"

Hafta 2'de SGD'yi tanımıştık; Defazio çok daha keskin: pratikte tam-batch gradient descent neredeyse hiç kullanılmaz.

> "it's hard to come up with a compelling reason to use gradient descent given the success of stochastic gradient descent… if you need to do full batch optimization, do not use gradient descent — I can't emphasize it enough." — Defazio, 19:03 / 21:48

Neden? Çünkü makine öğrenmesi kaybı, **çok sayıda benzer terimin toplamıdır** (her veri noktası bir terim). Bu yapıda veri **artıklığı (redundancy)** vardır — yüzlerce benzer örnek aynı yönü gösterir; hepsini taramak israftır. SGD bu artıklığı sömürür: küçük bir mini-batch zaten iyi bir yön verir, ve aynı bütçeyle çok daha fazla adım atarsın. Defazio'nun inceliği: SGD "gürültülü gradient descent" değildir; gürültünün kendine has bir dinamiği vardır (vadi tabanında "zıplama"). Şekil 11.4 bunu görselleştirir: solda çok sayıda benzer örnek-gradyanı ve ortalaması, sağda aynı bütçenin GD'de 1 adıma karşı SGD'de ~N adıma dönüşmesi.



Şekil 11.4: Stokastik gradyan inişinin sezgisi: veri artıklığı. Sol panelde tek-veri örnek-gradyanları (soluk violet oklar) birbirine yakın yönleri gösterir; bunların ortalaması tam gradyandır (kalın gold ok), birkaç okun ortalaması olan mini-batch ise zaten neredeyse aynı yönü verir (orta violet ok). Sağ panel aynı örnek-okuma bütçesinin GD'de tek bir adıma karşılık SGD/mini-batch'te ~N adıma dönüştüğünü gösterir. Defazio: kayıp benzer terimlerin toplamı olduğundan mini-batch erkenden iyi yön verir — aynı bütçeyle çok daha fazla adım atılır.

> Builder Notu — Artıklık → SGD
>
> **Geriye (Hafta 2 + Stat 110):** Bu, Hafta 2'deki "SGD = gradient'in tarafsız tahmincisi" fikrinin pratik sonucudur. Veri artıklığı = örneklerin istatistiksel benzerliği (Stat 110).
> **İleriye:** "Kayıp = benzer terimlerin toplamı" yapısı, SGD'nin neden ML'e özgü kadar iyi çalıştığını açıklar; bu yapı yoksa (örn. tek bir karmaşık fonksiyon) L-BFGS gibi yöntemler daha uygun olabilir.

11.8 (Defazio) Momentum: "Neredeyse Her Zaman Kullan"

Defazio momentum'u SGD'nin üstüne eklenen, "neredeyse her zaman açman gereken" bir trick olarak sunuyor.

> "momentum is a trick that you should pretty much always be using when you're using stochastic gradient descent." — Defazio, 27:52

Fizikten gelen sezgi: momentum, bir nesnenin gittiği yönde gitmeye devam etme eğilimidir (Newton). Pratikte bir **hız tamponu (heavy ball)** tutarsın: gradient'leri biriktirip adımı bu birikmiş hızla atarsın.

$$p \leftarrow \beta p + \nabla \mathcal{L}(\theta), \qquad \theta \leftarrow \theta - \eta\, p$$

Tipik $\beta = 0.9$. Momentum neden işe yarar? Defazio iki sebep veriyor: (1) **salınım sönümleme / acceleration** — gradient descent'in vadi duvarlarındaki zikzakını bastırır; (2) **noise smoothing** — gradient'lerin hareketli ortalamasını alarak SGD gürültüsünü yumuşatır, böylece son nokta çözümün iyi bir tahmini olur (saf SGD'de bir sürü son noktayı ortalaman gerekirdi). Heavy-ball kuralı $p \leftarrow \beta p + \nabla \mathcal{L}(\theta)$, geçmiş gradyanları üstel sönümle biriktirir.

> "momentum adds smoothing to the optimization, so the last point you visit is still a good approximation to the solution." — Defazio, 42:09

(Bir uyarı: Nesterov acceleration teorik olarak convex problemlerde yakınsamayı hızlandırır ama sinir ağlarında ve gürültüde bu etki büyük ölçüde kaybolur — pratikte momentum'un asıl faydası noise smoothing'dir.) Şekil 11.5 aynı $\kappa = 20$ vadide GD'nin zikzakı ile momentum'un yumuşak inişini yan yana koyar.

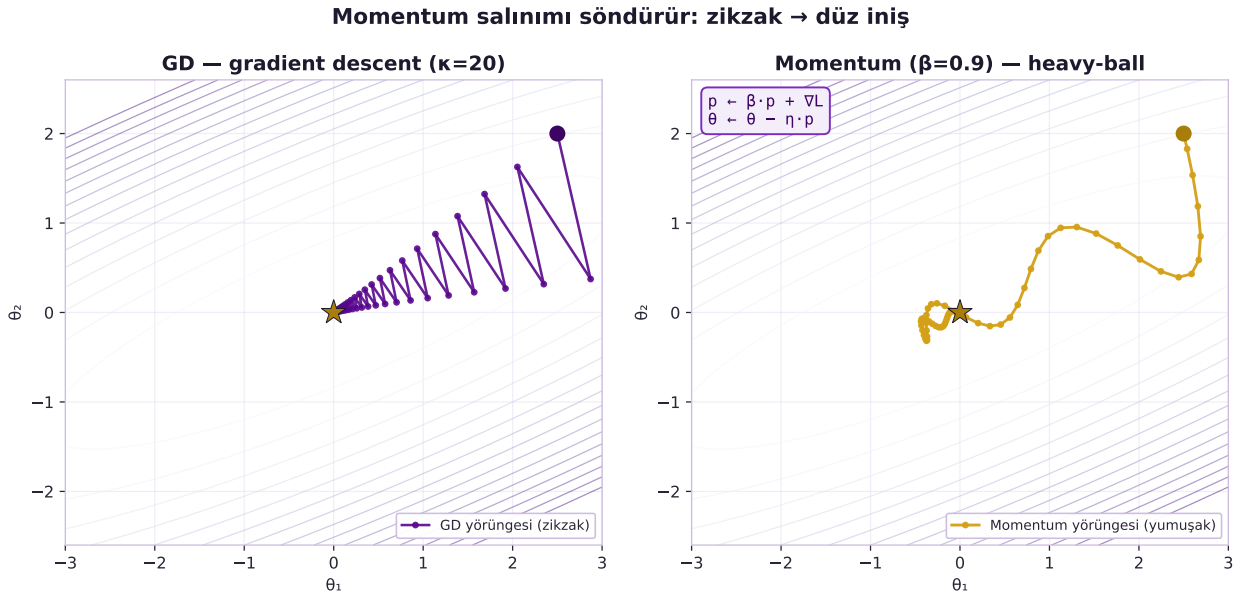
Üç optimizer'ı tek bir kayıp eğrisinde de görmek aydınlatıcıdır: Şekil 11.6 GD, SGD+Momentum ve Adam'ı aynı $\kappa = 20$ vadide logaritmik ölçekte karşılaştırır — Defazio'nun "zikzak / sönümleme / adaptif" üçlü anlatısının doğrudan görsel özeti.

> Builder Notu — Heavy Ball Sönümleme
>
> **Geriye (Calculus + fizik):** Momentum = ivme (Calculus ikinci türev) + Newton'un birinci yasası; "heavy ball" fiziksel benzetmesi. Hareketli ortalama = geçmiş gradient'lerin üstel ağırlıklı toplamı.
> **İleriye:** SGD + momentum ($\beta = 0.9$) hâlâ pek çok görü probleminde state-of-the-art; üstüne LR schedule biner.

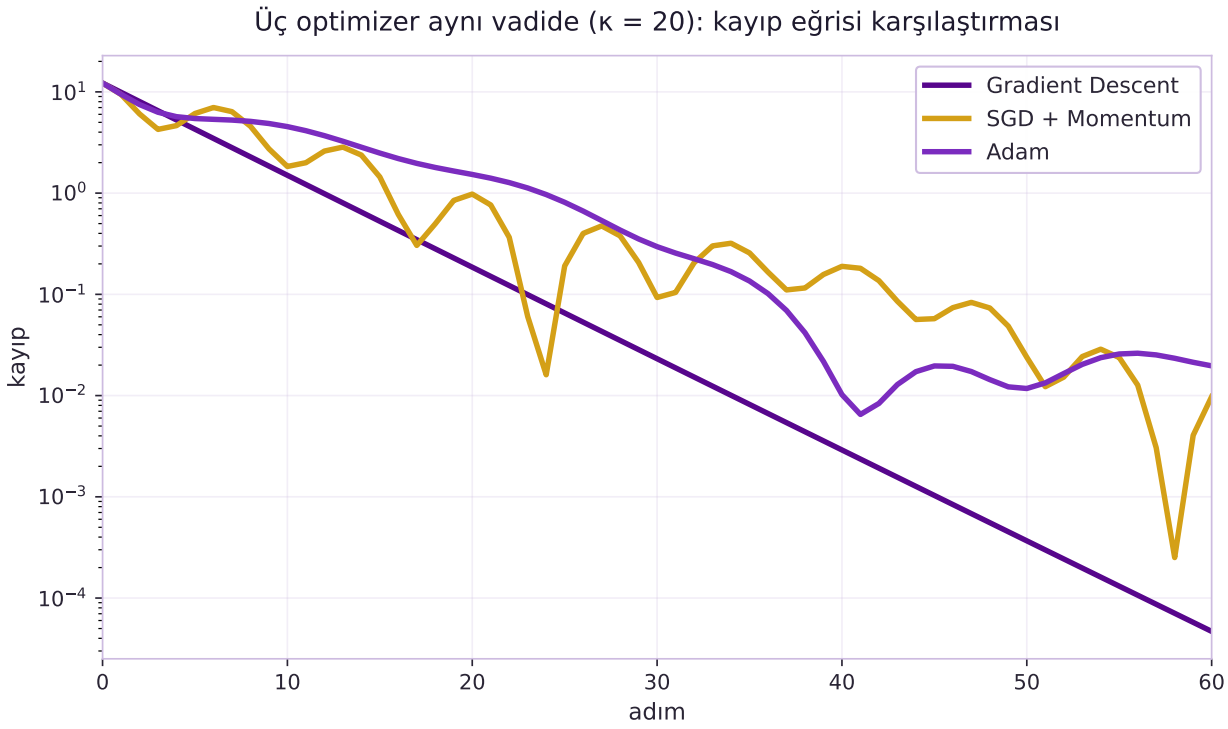
11.9 (Defazio) Adaptif Yöntemler: Her Ağırlığa Ayrı Öğrenme Oranı

Kötü koşullanmış problemler için Defazio **adaptif yöntemleri** tanıtıyor. Buradaki "adaptif", her ağırlık için **ayrı (individual) bir öğrenme oranı** demektir — global tek bir η yerine.

> "for adaptive methods, we want to adapt a learning rate for every weight individually, using information we get from gradients for each weight." — Defazio, 44:36



Şekil 11.5: Momentum salınımı söndürür. Aynı κ=20 koşullanmalı dar kuadratik vadide (soluk Purples kontur) iki optimizer karşılaştırılır. SOL panel — GD (η=0.095): yörünge vadinin dik duvarları arasında zikzak çizerek minimuma yaklaşır; her adım eğimin tersine sıçrayıp salınır (Defazio'nun "koşullanma kötüyse gradient descent zikzak çizer" gözlemi). SAĞ panel — Momentum/heavy-ball (η=0.010, β=0.9): p←β·p+∇L birikimli hız, dik yönlerdeki zıt-işaretli gradyanları birbirine söndürür; yörünge düz/yumuşak bir eğri olarak vadinin dibini takip eder. Mesaj: momentum salınımı sönümler ve gürültüyü yumuşatır.



Şekil 11.6: Üç optimizer aynı eğik kuadratik vadide ($\kappa = 20$, $\theta_0 = [2.5, 2.0]$, 60 adım) karşılaştırılır; kayıp logaritmik ölçekte gösterilir. Gradient Descent (lr = 0.095, mor) düz bir doğru boyunca tek-yönlü iner; SGD + Momentum (lr = 0.010, $\beta = 0.9$, altın) heavy-ball ataletiyle salınarak ilerler; Adam (lr = 0.15, mor-orta) adaptif öğrenme hızıyla her boyutu ayrı ölçekler. Eğrilerin biçimi Defazio'nun "zikzak / sönümleme / adaptif" üçlü anlatısını görselleştirir.

Neden gerekli? Çünkü ağın farklı kısımları çok farklı yapıdadır: erken katmanlar büyük görüntülerde sığ convolution'lar yapar; geç katmanlar küçük görüntülerde çok kanallı convolution'lar. Bir kısım için iyi olan learning rate, diğeri için kötü olabilir. Özellikle çıktıya doğrudan etki eden son katman ağırlıkları küçük learning rate ister. Defazio'nun dürüstlüğü: "neden gerektiğini tam bilmiyoruz; iyi koşullanmış bir ağda gerekmeyebilir bile."

Builder Notu — Per-Weight LR

Geriye (18.06): Per-weight learning rate, kötü koşullanmayı (özdeğer ölçek farkı) ağırlık bazında telafi etme girişimidir — bir tür diyagonal ön-koşullama (preconditioning).
İleriye: Per-parameter ölçekleme, büyük modellerde (transformer) eğitim kararlılığının anahtarıdır; LAMB, Adafactor gibi yöntemler bunu bellek-verimli yapar.

11.10 (Defazio) RMSprop ve Adam

RMSprop (root mean squared propagation, Hinton'ın ders slaytlarından): kare gradient'in **üstel hareketli ortalamasını** (v tamponu) tutar; sonra gradient'i bu ortalamanın kareköküne böler (eleman-bazlı). Böylece büyük-gradient'li yönlerde adım küçülür, küçük-gradient'li yönlerde büyür — otomatik per-weight ölçekleme. Sıfıra bölmeyi önlemek için bir ϵ ($\approx 10^{-7}$) eklenir.

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{v} + \epsilon}\, g$$

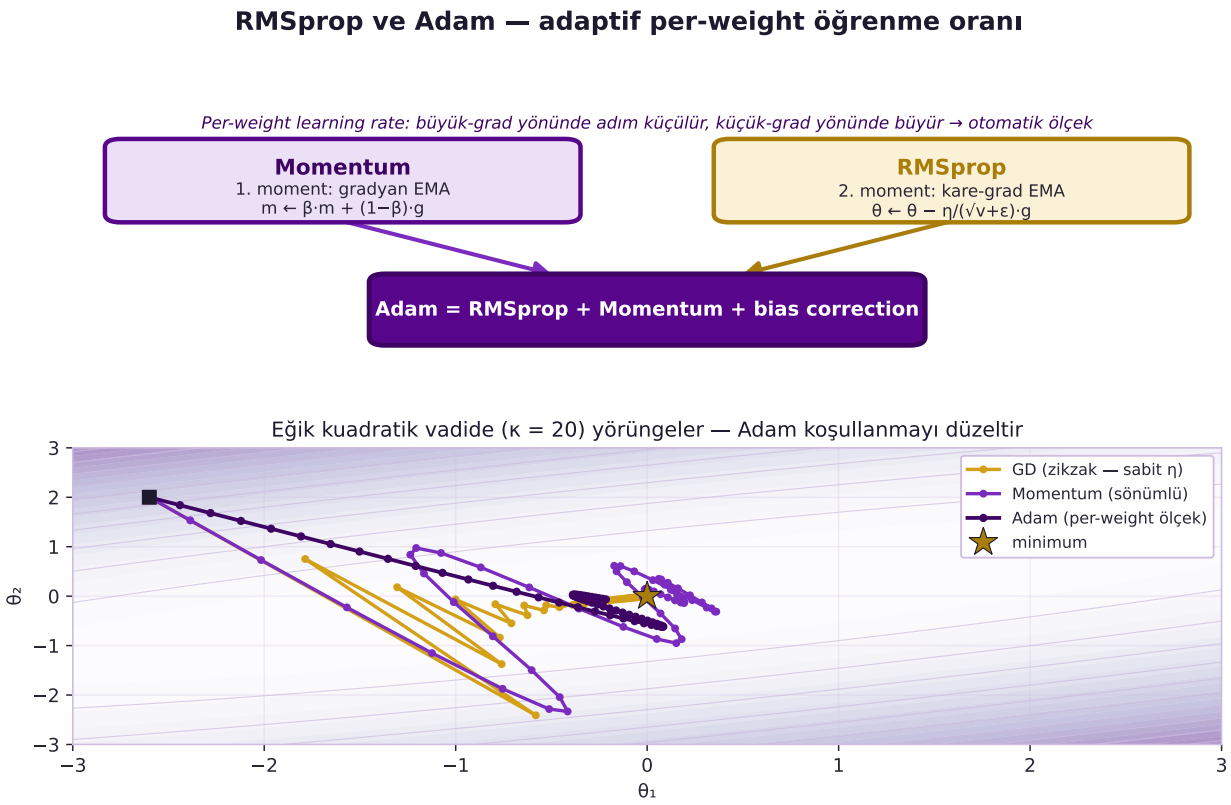
Adam (adaptive moment estimation) = **RMSprop + momentum**. Hem gradient'in (birinci moment) hem kare gradient'in (ikinci moment) üstel hareketli ortalamasını tutar; momentum'lu gradient'i RMSprop ölçeklemesiyle ($\theta \leftarrow \theta - \frac{\eta}{\sqrt{v}+\epsilon}\, g$) böler.

> "Adam is RMSprop with momentum… I'd generally recommend using either SGD with momentum or Adam as your go-to methods." — Defazio, 54:54 / 59:09

Adam'ın bir ek detayı **bias correction**: tamponlar sıfırdan başladığı için ilk adımlarda küçük kalırlar; bias correction bunu düzeltir (erken adımları ölçekler). Şekil 11.7 üstte kavramsal şemayı (Momentum + RMSprop → Adam), altta motor kanıtını (aynı vadide GD/Momentum/Adam yörüngeleri) verir.

Builder Notu — Adam = RMSprop + Momentum

Geriye (Stat 110): Üstel hareketli ortalama = geçmişi üstel sönümleyen ağırlıklı ortalama; ikinci moment = kare beklentisi (Stat 110 varyans/moment). Bias correction = tarafsız tahminci kurma (beklenti = gradient).
İleriye: AdamW (weight decay düzeltmeli Adam) bugün LLM eğitiminin varsayılan optimizer'ıdır; ϵ, β_1, β_2 ayarları pratik kararlardır.



Şekil 11.7: RMSprop ve Adam — adaptif per-weight öğrenme oranı. Üst panel kavramsal şema: iki kaynak modül — Momentum (1. moment, gradyan EMA, violet) ve RMSprop (2. moment, kare-grad EMA, $\theta \leftarrow \theta - \eta/(\sqrt{v}+\varepsilon)\cdot g$, gold) — birleşim oklarıyla Adam = RMSprop + Momentum + bias correction kutusunda birleşir. Annotation: büyük-grad yönünde adım küçülür, küçük-grad yönünde büyür → otomatik per-weight ölçek. Alt panel motor kanıtı: κ=20 eğik kuadratik vadide GD sabit-η ile zikzak çizip aşar, Momentum sönümlü salınır, Adam ise per-weight ölçekle koşullanmayı düzeltip minimuma düzgün iner (optimize_quad).

11.11 (Defazio) "Optimization'ın Ölümü": En Çok Kullanılan Yöntemin Teorisi Yanlış

Defazio bir optimizasyon araştırmacısı olarak çarpıcı bir itirafta bulunuyor — bu, LeCun'un Hafta 1'deki "kimse anlamıyor" temasının optimizasyon versiyonudur:

> "I hate Adam because I'm an optimization researcher and the theory in their paper is wrong — this has been shown recently; the method in fact does not converge, and you can show this on very simple test problems." — Defazio, 59:14

Yani modern ML'in en yoğun kullanılan yöntemlerinden biri, basit test problemlerinde bile yakınsamayabilir. Üstelik Adam, özellikle **görüntü problemlerinde SGD'den daha kötü genelleme (generalization)** verebilir — belki o "küçük, kötü yerel minimumlara" daha kolay düştüğü için. Yine de **dil modelleri için Adam zorunludur** (SGD ile eğitilemiyorlar). Defazio'nun pratik tavsiyesi nettir: **SGD+momentum veya Adam dene**; görüde genelde SGD+momentum, dilde Adam.

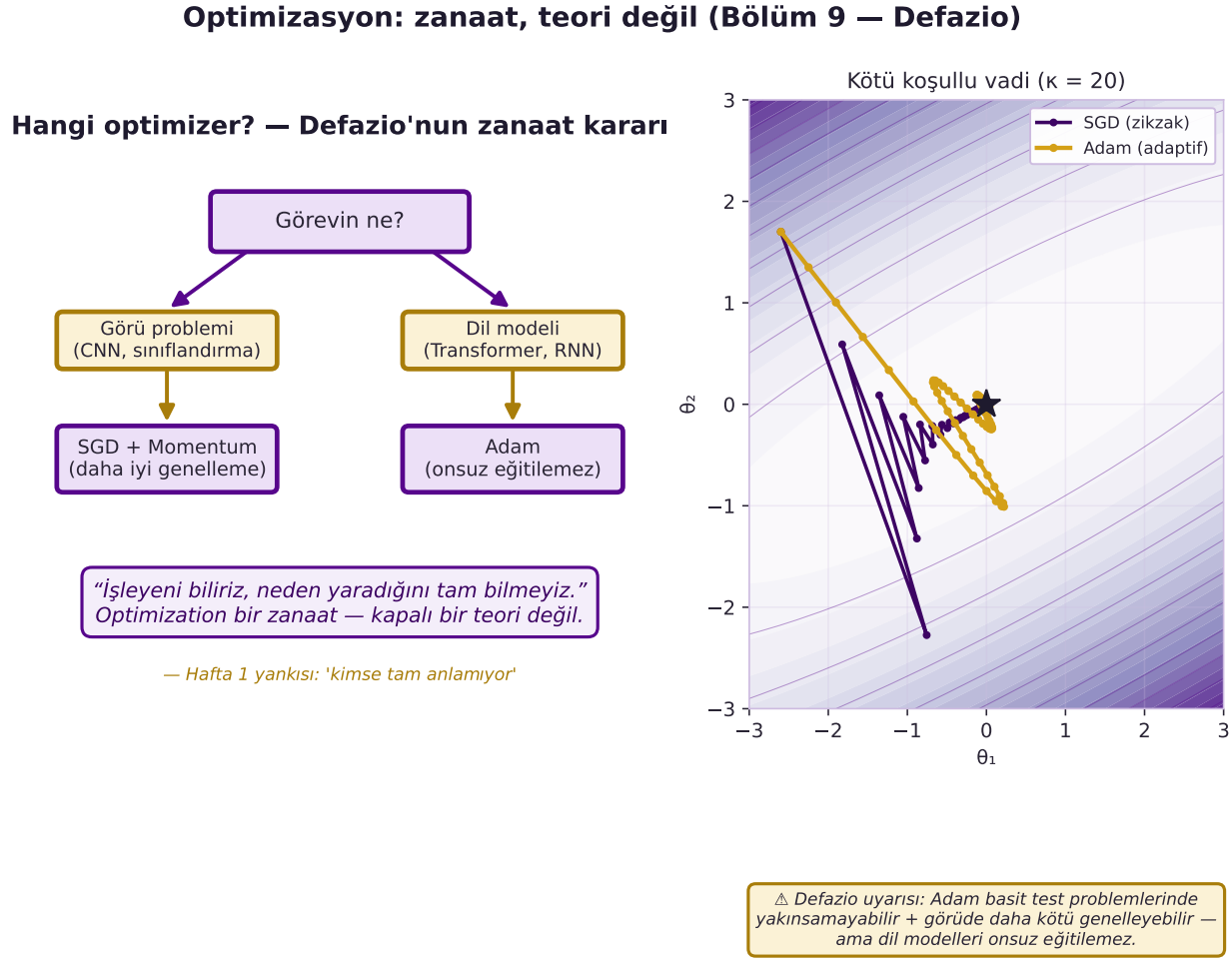
Bu bölüm dersin omurgasını özetler: optimizasyon bir teori değil, bir **zanaattir** — işe yarayanı biliyoruz, neden yaradığını tam bilmiyoruz. Şekil 11.8 bu zanaat kararını (görü → SGD+Momentum, dil → Adam) ve Defazio'nun dürüst uyarısını tek figürde toplar.

> Builder Notu — Zanaat, Teori Değil
>
> **Geriye (Hafta 1):** Bu, LeCun'un "it's important to understand that nobody understands" (Hafta 1) cümlesinin optimizasyondaki tam karşılığıdır. Alan, kanıttan çok ampirik başarıyla ilerler.
> **İleriye:** Adam'ın yakınsama sorunu → AMSGrad, AdamW gibi düzeltmeler; generalization farkı → "flat vs sharp minima" araştırması. Bunlar hâlâ açık problemler.

11.12 Bu Dersin Özeti

1. **Convolution = Toeplitz matrisiyle çarpım** (Canziani): kernel kaydırılarak tekrarlanır, gerisi sıfır — seyrek (locality) + paylaşımlı (stationarity). "Bir sürü sıfırlı matris çarpımı."
2. **Tam-bağlı vs convolution:** ikisi de $A\mathbf{x}$; FC'de kernel = tam satır, conv'da kernel = kaydırılan küçük pencere.
3. **Condition number** $\kappa = L/\mu$ (Defazio) yakınsama hızını belirler; learning rate "ıraksamanın eşiğinde" seçilir.
4. **SGD'yi kullan, gradient descent'i değil** — veri artıklığını sömürür.
5. **Momentum'u neredeyse her zaman aç** ($\beta \approx 0.9$): salınım sönümleme + noise smoothing.
6. **Adaptif yöntemler** per-weight learning rate verir; **RMSprop** (kare-grad EMA ile böl) → **Adam** = RMSprop + momentum (+ bias correction).
7. **Pratik tavsiye:** SGD+momentum veya Adam. Ama Adam'ın teorisi tartışmalı (yakınsama + generalization) — "optimization'ı tam anlamıyoruz".

> Tek Bir Cümle
>
> Convolution, yerel ve paylaşımlı bir Toeplitz matrisiyle çarpımdan ibarettir (Canziani); ve o ağı eğitmek, gradient descent'i veri artıklığı için SGD'ye, salınım+gürültü için momentum'a, kötü koşullanma için adaptif ölçeklemeye (RMSprop/Adam) çeviren — teorisi hâlâ eksik ama pratikte çalışan — bir zanaattir



Şekil 11.8: Optimizasyon bir zanaattır, kapalı bir teori değil (Bölüm 9 — Defazio). Sol panel karar şeması: görü problemleri (CNN, sınıflandırma) için SGD + Momentum daha iyi genelleme verir; dil modelleri (Transformer, RNN) için Adam pratikte zorunludur. Sağ panel kötü koşullu kuadratik vadide ($\kappa = 20$) kanıt sunar: saf SGD vadinin dik ekseninde zikzak çizerken Adam adaptif adımıyla minimuma () daha düzgün ilerler. Defazio'nun dürüst uyarısı altta: Adam basit test problemlerinde yakınsamayabilir ve görüde daha kötü genelleyebilir — yine de büyük dil modelleri onsuz eğitilemez. 'İşleyeni biliriz, neden yaradığını tam bilmeyiz' (Hafta 1 'kimse tam anlamıyor' yankısı).

(Defazio).

11.13 Kontrol Soruları

i Soru 1: Convolution'ı bir matris çarpımı olarak nasıl yazarsın? Bu matrisin adı ve iki özelliği nedir? Tam-bağlı katmandan farkı?

Cevap: Kernel'i her satırda bir konum kaydırarak, geri kalanı sıfır bırakarak bir **Toeplitz matrisi** kurarsın; convolution = bu matrisle çarpım ("a matrix multiplication with a lot of zeros", Canziani 27:55). İki özelliği: **seyrek** (her satırda yalnızca kernel kadar sıfır-olmayan eleman = locality) ve **paylaşımlı** (her satır aynı kaydırılmış kernel = stationarity/parameter sharing). Tam-bağlı katmanda kernel matrisin **tam bir satırıdır** (her girdiye bağlı, paylaşım yok); convolution'da kernel küçük, kaydırılan, paylaşılan bir penceredir — yani convolution, yapısı kısıtlanmış bir lineer katmandır.

i Soru 2: Condition number nedir, neyi belirler? Defazio neden 'gradient descent kullanma' diyor?

Cevap: Condition number, kuadratik bir problemde Hessian'ın en büyük (L) ve en küçük (μ) özdeğerlerinin oranıdır:

$$\kappa = \frac{L}{\mu}$$

κ büyükse problem kötü koşullanmıştır (uzun ince vadi); gradient descent zikzak çizer, yavaşlar. Defazio "gradient descent kullanma" der çünkü ML kaybı çok sayıda benzer terimin toplamıdır (her veri noktası bir terim) ve bu yapıda **artıklık (redundancy)** vardır — küçük bir mini-batch zaten iyi bir yön verir, ve SGD aynı bütçeyle çok daha fazla adım atar. Tam-batch gradient descent bu artıklığı israf eder.

i Soru 3: Momentum neden işe yarar (iki sebep)? Heavy-ball güncellemesini yaz. Tipik β kaçtır?

Cevap: İki sebep: (1) **salınım sönümleme / acceleration** — vadi duvarlarındaki zikzakı bastırır; (2) **noise smoothing** — gradient'lerin hareketli ortalamasını alıp SGD gürültüsünü yumuşatır, böylece son nokta çözümün iyi bir tahmini olur. Heavy-ball güncellemesi:

$$p \leftarrow \beta p + \nabla \mathcal{L}(\theta), \qquad \theta \leftarrow \theta - \eta\, p$$

Tipik $\beta = 0.9$. Defazio: "momentum'u neredeyse her zaman kullan" (27:52). (Nesterov acceleration teorik olarak convex problemlerde hızlandırır ama gürültü onu büyük ölçüde öldürür; pratikte asıl fayda noise smoothing'dir.)

i Soru 4: (Builder) Adam nedir, hangi iki yöntemi birleştirir? Defazio neden onu hem önerir hem eleştirir?

Cevap: Adam (adaptive moment estimation) = RMSprop + momentum. RMSprop, kare gradient'in üstel hareketli ortalamasının karekköküne bölerek (eleman-bazlı, $+\epsilon$) per-weight ölçekleme yapar:

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{v} + \epsilon} g$$

Adam buna gradient'in hareketli ortalamasını (momentum) ve bias correction'ı ekler. Defazio **önerir** çünkü pratikte çok iyi çalışır ve bazı ağlar (dil modelleri) onsuz eğitilemez; ama **eleştirir** çünkü orijinal yakınsama teoremi yanlıştır ("does not converge", 59:14) ve görü problemlerinde SGD'den daha kötü genelleme verebilir. Pratik tavsiye: SGD+momentum (görü) veya Adam (dil) — "optimization'ı tam anlamıyoruz".

11.14 Egzersizler

Egzersiz 1 (Convolution = Toeplitz). 1B kernel $w = [1, 0, -1]$ ve girdi $x = [1, 2, 3, 4, 5]$ için convolution'ı, uygun bir Toeplitz matrisi T kurup $T \cdot x$ ile hesapla. Sonucu `torch.nn.functional.conv1d` ile karşılaştır. Matrisin kaç sıfır içerdiğini say (sparsity).

```
import torch
import torch.nn.functional as F
import numpy as np

w = [1, 0, -1]
x = np.array([1., 2., 3., 4., 5.])
k, n = len(w), len(x)
oh = n - k + 1
T = np.zeros((oh, n))                       # Toeplitz matrisi
for i in range(oh):
    T[i, i:i + k] = w
y_toeplitz = T @ x                          # -> [-2, -2, -2]

xt = torch.tensor(x, dtype=torch.float).view(1, 1, -1)
wt = torch.tensor(w, dtype=torch.float).view(1, 1, -1)
y_conv = F.conv1d(xt, wt).flatten()         # -> [-2, -2, -2]
print(y_toeplitz, y_conv, int((T == 0).sum()))   # eşit + sıfır sayısı
```

Egzersiz 2 (İki görüş). Bir 3×2 matris A ve vektör x için Ax'i (a) satır·x iç çarpımları, (b) sütunların ağırlıklı toplamı ($A\mathbf{x} = \sum_j x_j \mathbf{a}_j$) olarak ayrı ayrı hesapla; ikisinin aynı çıktığını doğrula.

```
import numpy as np
A = np.array([[2., -1.], [1., 3.], [0., 2.]])   # 3x2
x = np.array([4., -1.])
row_view = np.array([A[i] @ x for i in range(A.shape[0])])   # satır·x
col_view = x[0] * A[:, 0] + x[1] * A[:, 1]                   # sütun toplamı
print(row_view, col_view, np.allclose(row_view, col_view))
```

Egzersiz 3 (Learning rate süpürmesi). Basit bir kuadratik $L(w) = (w - 3)^2$ üzerinde gradient descent'i $\eta \in \{0.01, 0.1, 0.5, 1.0, 1.01\}$ için koştur (Hafta 2 Egz 5). Hangi η ıraksıyor? "Edge of divergence" gözlemini Defazio'nun yorumuyla ilişkilendir.

```
import numpy as np
def run(lr, steps=50, w0=10.0):
    w = w0
    for _ in range(steps):
        w = w - lr * 2 * (w - 3)            # ∇L = 2(w-3)
    return w
for lr in [0.01, 0.1, 0.5, 1.0, 1.01]:
    print(lr, run(lr))                       # lr=1.01 -> ıraksar (|1-2lr|>1)
```

Egzersiz 4 (Optimizer karşılaştırması). Hafta 2'nin spiral ağını üç optimizer ile eğit: (a) SGD, (b) SGD+momentum(0.9), (c) Adam. Kayıp eğrilerini karşılaştır. Hangisi en hızlı, hangisi en kararlı? Momentum'un salınımı nasıl etkilediğini gözlemle.

```
import torch
import torch.nn as nn

net = nn.Sequential(nn.Linear(2, 32), nn.ReLU(), nn.Linear(32, 5))
opts = {
    "sgd": torch.optim.SGD(net.parameters(), lr=0.1),
    "momentum": torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.9),
    "adam": torch.optim.Adam(net.parameters(), lr=1e-3),
}
# her opt icin: forward -> loss -> backward -> step (Hafta 2 dongusu)
# loss egrilerini kaydet ve karsilastir
```

Egzersiz 5 (Hafta 5 habercisi). Hafta 5 yalnızca Canziani'nin Practicum'u olacak: 1D/2D convolution'ın PyTorch'ta uygulanması ve **otomatik türev (autograd)**. (a) `torch.nn.functional.conv2d` ile küçük bir görüntüye bir kenar dedektörü kernel'i uygula. (b) Bir tensöre `requires_grad=True` verip basit bir fonksiyonun `.backward()`'ını çağır; `.grad`'ı elle hesapladığınla karşılaştır. Bu, Hafta 2'nin backprop'unun PyTorch'taki otomatik hâlidir — neden "autograd" Hafta 2'nin doğal devamı?

```
import torch
x = torch.tensor(2.0, requires_grad=True)
y = x ** 3 + 2 * x                           # f(x) = x^3 + 2x
y.backward()                                 # otomatik turev
print(x.grad)                                # 3x^2 + 2 = 14 (elle ile karsilastir)
```

11.15 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H5: 1D/2D Convolution ve Otomatik Türev (autograd) — yalnız Canziani

Convolution'dan autograd'a. Bu hafta convolution'ın cebirini (Toeplitz) ve ağı eğiten optimizasyon motorunu (SGD/momentum/Adam) gördük. Hafta 5'te lecture yok; **yalnızca Canziani'nin Practicum'u** convolution'ı PyTorch'ta somutlaştırır (`conv1d`/`conv2d`, boyutlar) ve **autograd**'ı gösterir — Hafta 2'de

> LeCun'un elle kurduğu Jacobian zincirinin PyTorch'taki otomatik karşılığı. Egzersiz 1 (Toeplitz) ve Egzersiz 5 (conv2d + autograd) tam bu derse hazırlar.

Hafta 5: 1D/2D Convolution ve Otomatik Türev (autograd) — yalnızca Canziani (Practicum)

Hafta 5'te lecture yok; Canziani convolution'ı PyTorch'ta somutlaştırır (`conv1d`/`conv2d`, boyutlar) ve **autograd**'ı gösterir — Hafta 2'de LeCun'un elle kurduğu Jacobian zincirinin PyTorch'taki otomatik karşılığı.

Hafta 5 öncesi yapılacak:

- Egzersiz 1 (Toeplitz) ve Egzersiz 5 (conv2d + autograd) çöz.
- "Convolution = bir sürü sıfırlı matris çarpımı" cümlesini kendi sözcüklerinle yaz.
- Hafta 2'nin `loss.backward()`'ını hatırla — Hafta 5'te onun nasıl çalıştığını PyTorch içinden göreceğiz.

11.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Matris × vektör — satır görüşü	Çıktı elemanı = satır·girdi; kernel = matris satırı	Canziani 13m31
Matris × vektör — sütun görüşü	Çıktı = sütunların ağırlıklı toplamı	Canziani 16m22
Toeplitz matrisi	Kernel kaydırılarak tekrarlanmış seyrek matris	Canziani 23m36
Convolution = matris çarpımı	"Bir sürü sıfırlı matris çarpımı"; seyrek + paylaşımlı	Canziani 27m55
Condition number κ	L/μ (en büyük/en küçük özdeğer); yakınsama hızı	Defazio 11m36
Edge of divergence	En büyük güvenli learning rate; hız için ıraksama eşiği	Defazio 13m42
SGD vs full-batch	Veri artıklığını sömür; "gradient descent kullanma"	Defazio 19m03
Momentum (heavy ball)	Hız tamponu; salınım sönümleme + noise smoothing; β=0.9	Defazio 27m52
Adaptif yöntem	Her ağırlığa ayrı learning rate	Defazio 44m36
RMSprop	Kare-grad EMA'nın köküne böl ($+\varepsilon$); per-weight ölçek	Defazio 50m43
Adam	RMSprop + momentum + bias correction	Defazio 54m54
Optimization'ı anlamıyoruz	Adam teorisi yanlış (yakınsamıyor); ama pratikte çalışır	Defazio 59m14

11.17 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Convolution = Toeplitz matris** → 18.06 matris-vektör + Hafta 3 (locality/stationarity) + 18.065 yapılı matrisler.
2. **Condition number** $\kappa = L/\mu$ → 18.06 özdeğer + 18.065 koşullanma.
3. **Momentum = ivme** → Calculus ikinci türev + fizik (Newton).
4. **EMA / ikinci moment** → Stat 110 ağırlıklı ortalama, varyans/moment.
5. **SGD artıklık** → Hafta 2 tarafsız tahminci + Stat 110 örneklem.

İleriye köprüler (production / research):

1. **Toeplitz/circulant** → FFT ile hızlı convolution; im2col+GEMM, Winograd.
2. **SGD+momentum / Adam** → AdamW (LLM varsayılanı), LAMB, Adafactor.
3. **Edge of divergence** → LR warmup + cosine decay schedule.
4. **Adam yakınsama/generalization sorunu** → AMSGrad, flat vs sharp minima araştırması.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Convolution gizemli bir işlem değil — yerel ve paylaşımlı bir Toeplitz matrisiyle çarpımdır (Canziani: "bir sürü sıfırlı matris çarpımı"); ve o ağı eğitmek, gradient descent'i veri artıklığı için SGD'ye, salınım ve gürültü için momentum'a, kötü koşullanma için adaptif ölçeklemeye (Adam) çeviren bir zanaattir — işe yarayanı biliriz, ama Defazio'nun dürüstçe itiraf ettiği gibi, neden yaradığını hâlâ tam anlamıyoruz.

12 1B/2B Convolution ve Otomatik Türev (autograd)

NYU'nun bu haftaki tek sesi: Alfredo Canziani'nin Practicum'u (bu hafta ayrı Lecture yok). İki konu işliyoruz, ikisi de 'elle anladığımız' şeyleri PyTorch'ta somutlaştırmak: (1) convolution'ı bir *projeksiyon* olarak okumak, boyut aritmetiğini ($o = n - k + 1$, 4-boyutlu 2B kernel) koda dökmek; (2) *otomatik türev (autograd)* — yani önceki haftalarda elle kurduğumuz backprop'un (zincir kuralı + hesaplama grafiği) PyTorch tarafından `requires_grad` + `.backward()` ile otomatik hesaplanması

> Bölüm bilgisi
>
> - **Canziani'nin Practicum videosu:** YouTube — 1D/2D convolutions and autograd (≈45 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hoca:** Alfredo Canziani (yalnızca Practicum — **bu hafta ayrı Lecture yok**)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈22 dk

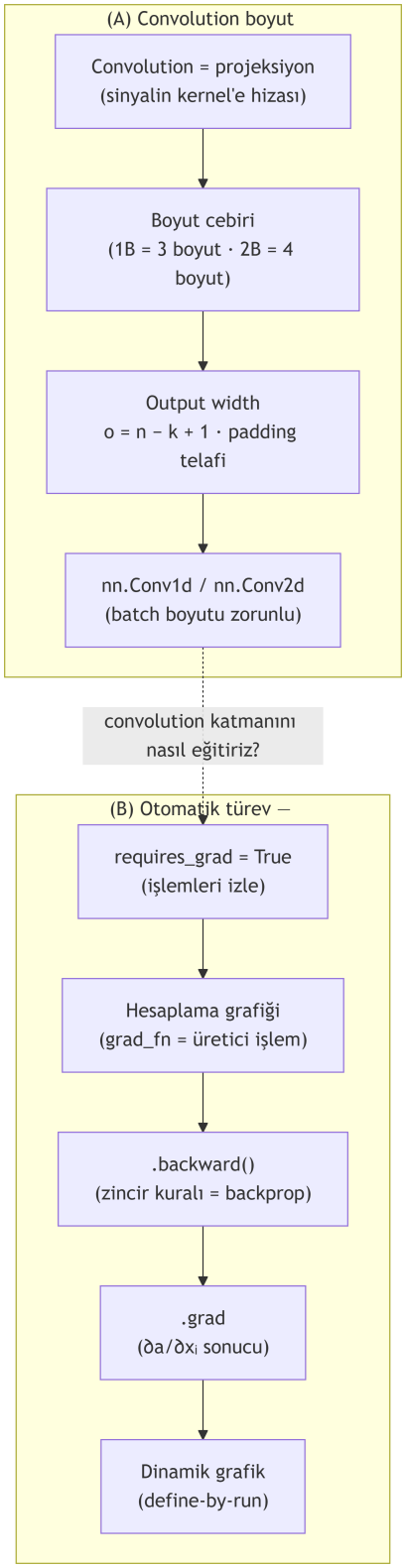
12.1 Bu Derste Ne Var?

Bu hafta tek hocalı: yalnızca **Alfredo Canziani**'nin Practicum'u var (bu hafta ayrı bir Lecture yok). İki konu işliyoruz, ikisi de "elle anladığımız" şeyleri **PyTorch'ta somutlaştırmak**: (1) convolution'ın 1B/2B boyutlarını ve çıktı genişliğini koda dökmek; (2) **otomatik türev (autograd)** — yani Hafta 2'de LeCun'un elle kurduğu, Hafta 4'te Defazio'nun kullandığı gradient'lerin PyTorch tarafından otomatik hesaplanması.

Canziani'nin köprüsü açık: convolution bir **projeksiyon**dur (sinyalin kernel'e hizası), ve autograd, Hafta 2'nin backprop'unun (zincir kuralı + Jacobian) "düğmeye basınca çalışan" hâlidir. Bu hafta yeni bir teori değil; önceki üç haftanın **çalışan koda** dönüşmesi.

Bu haftanın üç ana fikri:

1. **Convolution boyutları kuralı:** her convolution çıktıyı $(k - 1)$ kadar kısaltır ($o = n - k + 1$); 2B kernel topluluğu 4 boyutludur.
2. **autograd = otomatik backprop.** `requires_grad=True` ile PyTorch tüm işlemleri bir **hesaplama grafiğinde** izler; `.backward()` gradient'leri zincir kuralıyla hesaplar, `.grad`'a yazar.
3. **Dinamik grafik (define-by-run):** PyTorch grafiği çalışma anında kurar — döngüler, koşullar serbestçe kullanılır.


(A) Convolution boyut
Convolution = projeksiyon
(sinyalin kernel'e hizası)
Boyut cebiri
(1B = 3 boyut · 2B = 4
boyut)
Output width
o = n − k + 1 · padding
telafi
nn.Conv1d / nn.Conv2d
(batch boyutu zorunlu)
convolution katmanını
nasıl eğitiriz?
(B) Otomatik türev —
requires_grad = True
(işlemleri izle)
Hesaplama grafiği
(grad_fn = üretici işlem)
.backward()
(zincir kuralı = backprop)
.grad
(∂a/∂xᵢ sonucu)
Dinamik grafik
(define-by-run)

Builder Notu — Çalışan Koda Dönüşüm

Geriye (önkoşul kurslar):

- **Convolution = projeksiyon** → 18.06 iç çarpım/projeksiyon + Hafta 4 (Toeplitz, satır = kernel).
- **autograd = zincir kuralı** → Hafta 2 backprop (Jacobian zinciri) + Calculus Ders 4 + Karpathy micrograd (`_backward`).
- **Output width** $o = n - k + 1$ → basit sayma; Hafta 3 stride formülünün stride=1 hâli.

İleriye (production / research):

- autograd → tüm PyTorch eğitiminin motoru (`torch.autograd`); dinamik grafik, define-by-run paradigması.
- Convolution boyut yönetimi → padding/stride ile mimari tasarımı, receptive field hesabı.

Tek cümleyle: Bu hafta convolution'ın boyut aritmetiğini ve PyTorch'un otomatik türev motorunu (autograd) öğreniyoruz — yani önceki haftaların elle kurduğu backprop'un, `requires_grad` + `.backward()` ile nasıl tek satıra indiğini.

12.2 (Canziani) Convolution = Projeksiyon

Canziani convolution'a Hafta 4'ün diliyle başlıyor: her convolution adımı, sinyalin bir parçasının kernel üzerine **projeksiyonudur** (normalize edilmemiş bir kosinüs/hiza değeri).

> "this is going to be my projection of my input signal onto the kernel… what is the alignment of this part of the signal onto this specific subspace?" — Canziani, 1:20

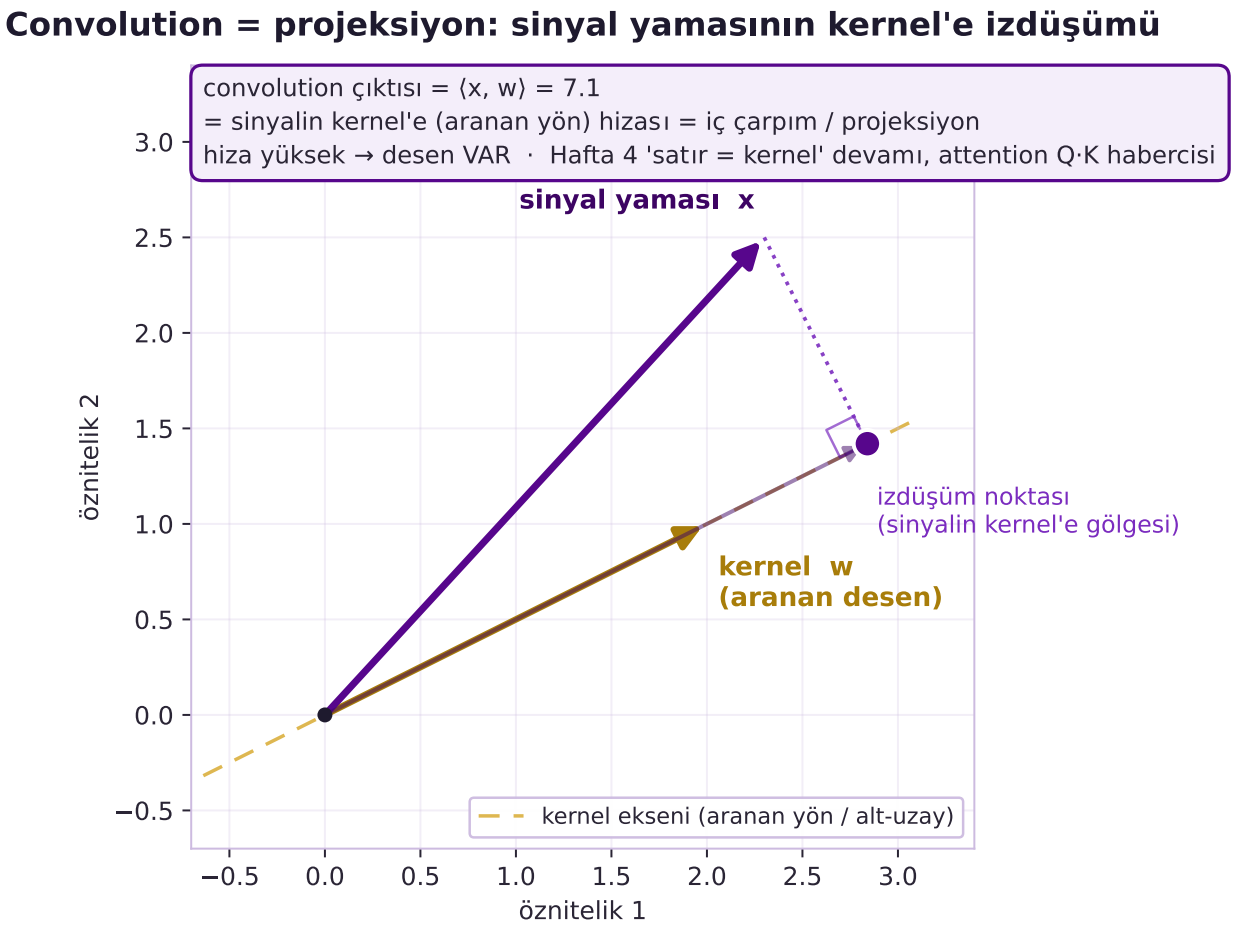
Yani kernel bir "yön" (alt-uzay) tanımlar; convolution çıktısı, sinyalin o yöne ne kadar hizalı olduğunu ölçer. Çıktı yüksekse, kernel'in aradığı desen o konumda güçlüdür. Bu, Hafta 3'ün "öznitelik dedektörü" ve Hafta 4'ün "satır = kernel, çıktı = iç çarpım" görüşünün aynı fikrin devamıdır. Şekil 12.1 bu projeksiyonu somut bir 2B çizimde gösterir: gold ok kernel'in tanımladığı yönü (aranan desen / alt-uzay eksenini), violet ok sinyal yamasını çizer; kesikli dikme sinyali kernel eksenine indirir ve izdüşüm noktası — sinyalin kernel'e gölgesi — convolution çıktısının ta kendisidir.

Builder Notu — Projeksiyon = Hiza

Geriye (18.06 + Hafta 4): Projeksiyon = iç çarpım / kosinüs benzerliği (18.06); Hafta 4'ün "kernel = matrisin satırı, çıktı = satır·girdi" görüşü tam olarak bu projeksiyondur.
İleriye: "Kernel = aranan yön" görüşü, attention'daki sorgu-anahtar hizasının ($Q \cdot K^\top$) habercisidir (Hafta 6 ve sonrası).

12.3 (Canziani) 1B/2B Convolution ve Çok-Kanallı Kernel'ler

Canziani convolution kernel'lerinin **boyut aritmetiğini** açıyor. Tek bir kernel küçük bir ağırlık penceresidir; bir katman birden çok kernel tutar ve onları istifleyerek/kaydırarak uygular. Çok-kanallı durumda kernel'in



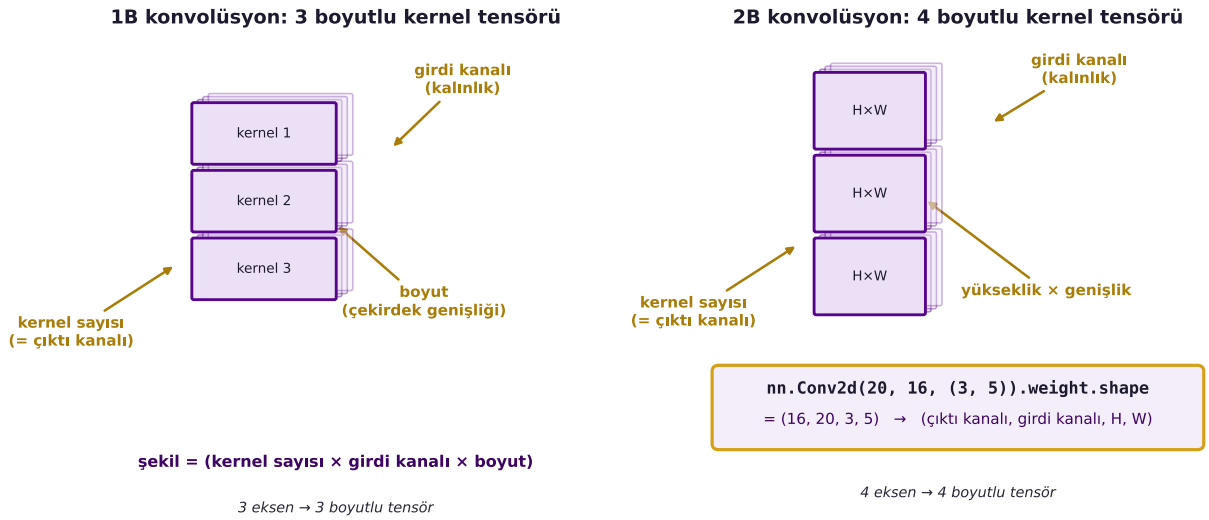
Şekil 12.1: Convolution = projeksiyon: tek bir convolution çıktısı, sinyal yamasının kernel yönüne (aranan desene) izdüşümüdür. Gold ok kernel'i ve aradığı alt-uzay eksenini, violet ok sinyal yamasını gösterir; kesikli dikme sinyali kernel eksenine indirir ve izdüşüm noktası (sinyalin kernel'e gölgesi) çıkar. Çıktı = ⟨x, w⟩ iç çarpımı = hiza; hiza yüksekse desen vardır. Hafta 4'teki 'satır = kernel' okumasının devamı ve attention Q·K skorunun habercisi.

bir **kalınlığı (thickness)** vardır = girdi kanal sayısı.

- **1B convolution:** sinyal bir boyutlu (örn. ses). m kernel, her biri (kalınlık × kernel_boyutu). Kernel topluluğu **3 boyutlu** bir tensördür: (kernel sayısı × girdi kanalı × kernel boyutu).
- **2B convolution:** sinyal iki boyutlu (görüntü). Kernel topluluğu **4 boyutlu**dur: (kernel sayısı × girdi kanalı × yükseklik × genişlik).

> "what is the dimensionality of the collection of kernels used for two-dimensional data? Four." — Canziani, 26:05

Her kernel girdinin tüm kanallarına bakar (kalınlık = girdi kanalı) ve tek bir öznitelik haritası üretir; kernel sayısı = çıktı kanal sayısı. Bu, Hafta 3'teki "her kernel bir feature map" kuralının boyutsal kesin ifadesidir. Şekil 12.2 iki paneli yan yana koyar: solda 1B kernel topluluğunun 3-eksenli tensörü, sağda 2B'nin 4-eksenli tensörü ve `nn.Conv2d(20, 16, (3, 5)).weight.shape = (16, 20, 3, 5)` somutlaması.



Şekil 12.2: Kernel boyut cebiri: 1B konvolüsyonda kernel tensörü 3 boyutludur (kernel sayısı × girdi kanalı × boyut), 2B konvolüsyonda 4 boyutludur (kernel sayısı × girdi kanalı × yükseklik × genişlik). Her iki durumda da kernel sayısı çıktı kanalı sayısına, kalınlık ise girdi kanalı sayısına eşittir. PyTorch'ta nn.Conv2d(20, 16, (3, 5)) ağırlık tensörünün şekli (16, 20, 3, 5) olur — yani (çıktı kanalı, girdi kanalı, H, W).

Builder Notu — Kernel Boyut Cebiri

Geriye (Hafta 3): "Her kernel girdinin tüm kanallarına bakar, bir harita üretir" — Hafta 3'ün multi-channel kernel açıklamasının boyut-cebiri hâli.
İleriye: Bu boyutları doğru saymak, `nn.Conv2d` parametre sayımının ve bellek bütçesinin temelidir; mimari tasarımda kanal sayısı en kritik kapasite eksenidir.

12.4 (Canziani) Output Width: Her Convolution Boyut Kaybeder

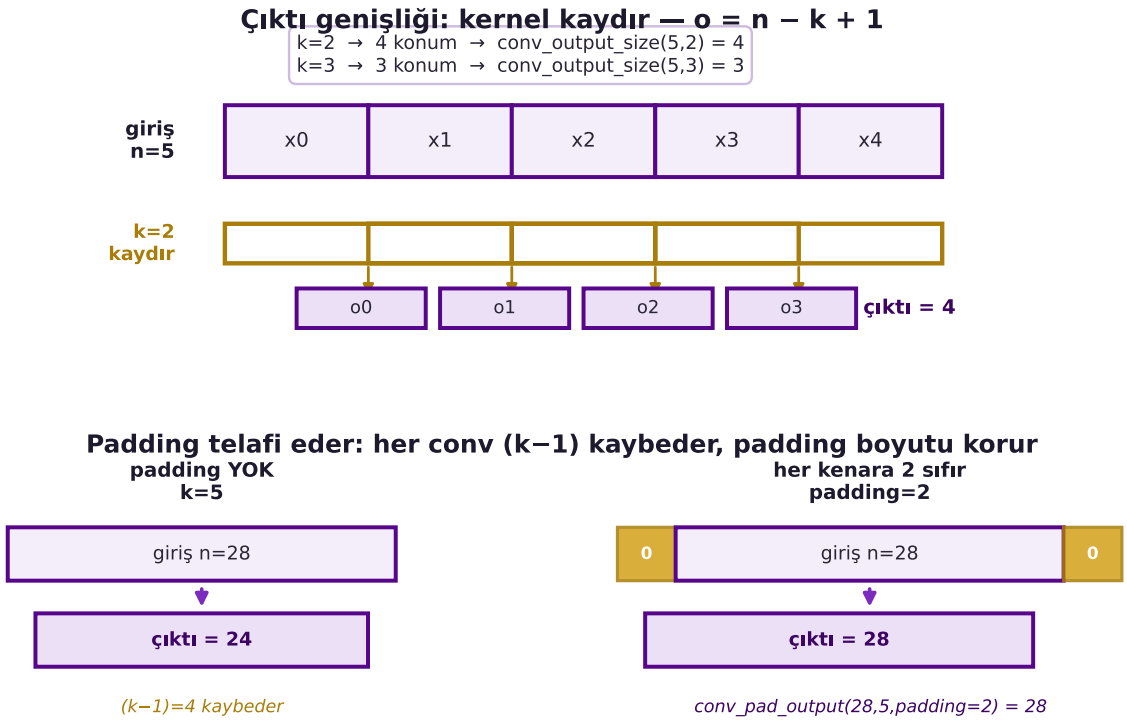
Convolution'ın temel boyut kuralı: kernel'i kaydırırken kenarlarda yer kalmaz, dolayısıyla çıktı girdiden küçüktür. Stride 1 için, girdi boyutu n ve kernel boyutu k ise çıktı:

$$o = n - k + 1$$

Yani her convolution boyutu $(k-1)$ kadar kısaltır. Canziani somut sayar: $n = 5$, $k = 2 \rightarrow$ çıktı 4; $n = 5$, $k = 3 \rightarrow$ çıktı 3.

> "every time you perform a convolution you lose the dimension of the kernel minus one." — Canziani, 17:17

Bu kayıp istenmiyorsa **padding** (kenarlara sıfır ekleme) ile telafi edilir — örneğin her kenara $(k-1)/2$ sıfır ekleyerek çıktı boyutu korunur. (Bu, Hafta 3'teki genel stride formülünün — $\lfloor (n-k)/s \rfloor + 1$ — stride = 1 hâlidir.) Şekil 12.3 üstte kernel kaydırmasıyla $o = n - k + 1$ kuralını, altta padding'in $n = 28$, $k = 5$ için boyutu nasıl koruduğunu somutlaştırır.



Şekil 12.3: Konvolüsyonda çıktı genişliği. ÜST: n=5 girişte kernel kaydırma — k=2 dört konum üretir (çıktı 4), k=3 üç konum (çıktı 3); genel kural o = n − k + 1, conv_output_size(5,2)=4 ve (5,3)=3 ile doğrulanır. ALT: padding telafisi — n=28, k=5 için padding olmadan çıktı 24'e düşer (k−1=4 kayıp); her kenara 2 sıfır eklenince conv_pad_output(28,5,padding=2)=28 ile boyut korunur.

Builder Notu — Boyut Kaybı + Padding

Geriye (Hafta 3): Hafta 3'ün stride'lı çıktı formülünün özel hâli (s=1). Boyut kaybı, convolution'ın "kenar etkisi"dir.
İleriye: Derin ağlarda her katmanın boyut kaybı birikir; `padding` ile boyut korumak veya kasıtlı küçültmek (stride/pooling) mimari tasarımın günlük aritmetiğidir. Receptive field hesabı da bu aritmetiğe dayanır.

12.5 (Canziani) PyTorch'ta Convolution Boyutları

Canziani teoriyi `nn.Conv1d` / `nn.Conv2d` ile somutlaştırıyor. Bir convolution katmanı tanımlarken: girdi kanalı, çıktı kanalı (kernel sayısı), kernel boyutu, stride, padding verilir.

```
import torch
import torch.nn as nn

# 1B: 20 girdi kanali, 16 kernel (cikti kanali), kernel boyutu 3
conv1d = nn.Conv1d(in_channels=20, out_channels=16, kernel_size=3, stride=1)
print(conv1d.weight.shape)   # (16, 20, 3)  -> 3 boyutlu kernel toplulugu
print(conv1d.bias.shape)     # (16,)

# 2B: 20 girdi kanali, 16 kernel, 3x5 kernel, padding ile boyut koru
conv2d = nn.Conv2d(in_channels=20, out_channels=16, kernel_size=(3, 5),
                   stride=1, padding=(1, 2))
x = torch.randn(1, 20, 64, 224)   # (batch, kanal, yukseklik, genislik)
y = conv2d(x)                     # padding sayesinde 64x224 korunur
```

Kritik noktalar: kernel ağırlığı 2B convolution'da **4 boyutludur** ($16 \times 20 \times 3 \times 5$); PyTorch her zaman bir **batch boyutu** bekler (tek sinyal göndersen bile (1, kanal, …) şeklinde), yoksa hata verir; `padding`, "her convolution boyut kaybeder" kuralını telafi etmenin yoludur.

Builder Notu — Batch-First Tensör

Geriye (Hafta 4): `conv1d.weight.shape = (16, 20, 3)` tam olarak Hafta 4'ün "kernel = yapılı matris satırı" görüşünün tensör hâli; bias = afin terimi.
İleriye: "Batch boyutu zorunlu" kuralı, tüm PyTorch tensör akışının (batch-first) temelidir; `padding`/`stride`/`dilation` üçlüsü, receptive field ve çözünürlüğü ayarlamanın pratik kollarıdır.

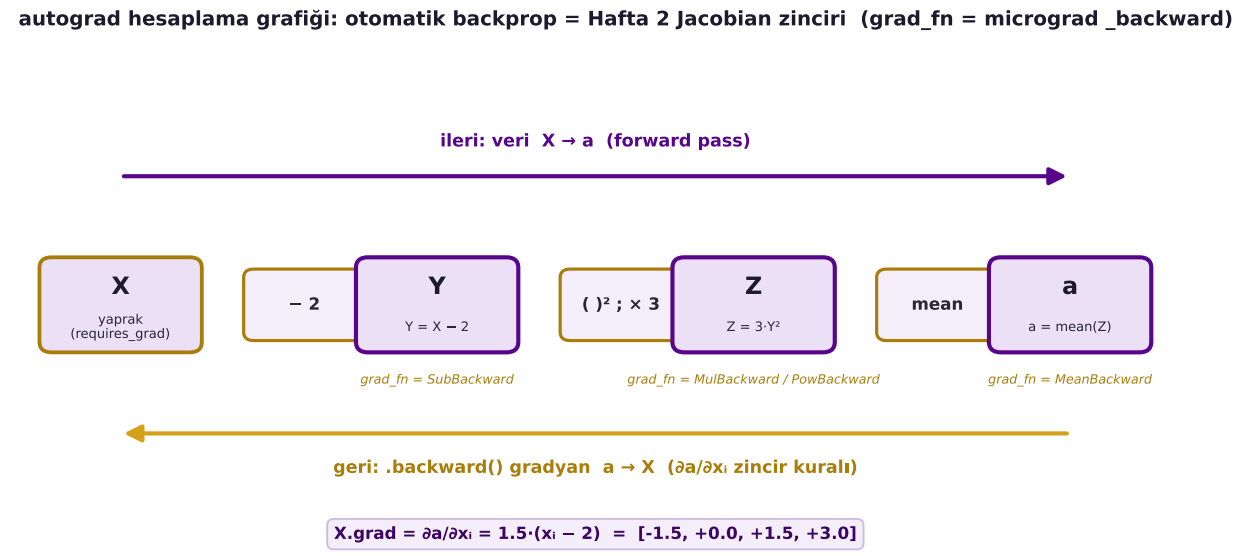
12.6 (Canziani) Otomatik Türev (autograd): requires_grad ve Hesaplama Grafiği

Practicum'un ikinci yarısı **autograd**. Canziani şakayla başlıyor: 90'larda LSTM gradient'lerini elle türetmek zorundaydılar (sayfalar dolusu); bugün PyTorch bunu otomatik yapar.

Bir tensöre `requires_grad=True` verirsen, PyTorch o tensör üzerindeki **tüm işlemleri bir hesaplama grafiğinde izler**, böylece kısmi türevleri hesaplayabilir.

> "I'm asking torch: please track all the gradient computations over the tensor, such that we can perform computation of partial derivatives." — Canziani, 28:53

Her türetilmiş tensörün bir `grad_fn`'i (grad function) vardır: onu hangi işlemin ürettiğini söyler. Örneğin $Y = X - 2$ için Y'nin `grad_fn`'i bir "SubBackward"dır. Bu, Hafta 2'de LeCun'un "her modülün bir backward'ı var" dediği şeyin ta kendisidir — ve Hafta 6'da göreceğimiz RNN gradient'leri de bu motorla otomatik akar. Şekil 12.4 bu hesaplama grafiğini bir DAG olarak çizer: üstte ileri geçiş (veri $X \to a$), altta geri geçiş (`.backward()` ile gradyan $a \to X$), her düğümün altında `grad_fn` etiketi.



Şekil 12.4: autograd hesaplama grafiği (DAG): X → [−2] → Y → [( )^2; ×3] → Z → [mean] → a. Üst violet ok ileri geçişi (veri X→a), alt gold ok geri geçişi (.backward() ile gradyan a→X) gösterir. Her ara düğümün altındaki grad_fn etiketi (SubBackward, MulBackward/PowBackward, MeanBackward) o düğümün yerel geri-yayılım kuralını tutar — yani autograd, Hafta 2'deki Jacobian zincir kuralının otomatikleştirilmiş hâlidir ve grad_fn, micrograd'daki _backward closure'unun PyTorch karşılığıdır. Sonuç: X.grad = $\partial a/\partial x_i = 1.5\cdot(x_i - 2) = [-1.5, 0, +1.5, +3]$.

> Builder Notu — grad_fn = micrograd _backward
>
> **Geriye (Hafta 2 + Karpathy):** `grad_fn`, Karpathy'nin micrograd'daki `_backward` kapanışının PyTorch karşılığıdır; hesaplama grafiği, LeCun'un "modüllerin DAG'ı" (Hafta 2) görüşüdür.
> **İleriye:** `torch.autograd`, tüm derin öğrenme eğitiminin sessiz motorudur; `grad_fn` zincirini anlamak, bellek (retain_graph), gradient checkpointing ve hata ayıklamanın anahtarıdır.

12.7 (Canziani) Worked Example: a.backward() ve .grad

Canziani somut bir örnek yürütüyor. $X = [1, 2, 3, 4]$ (`requires_grad=True`), sonra:

- $Y = X - 2$
- $Z = 3 \cdot Y^2$

- $a = \text{mean}(Z)$ (skaler)

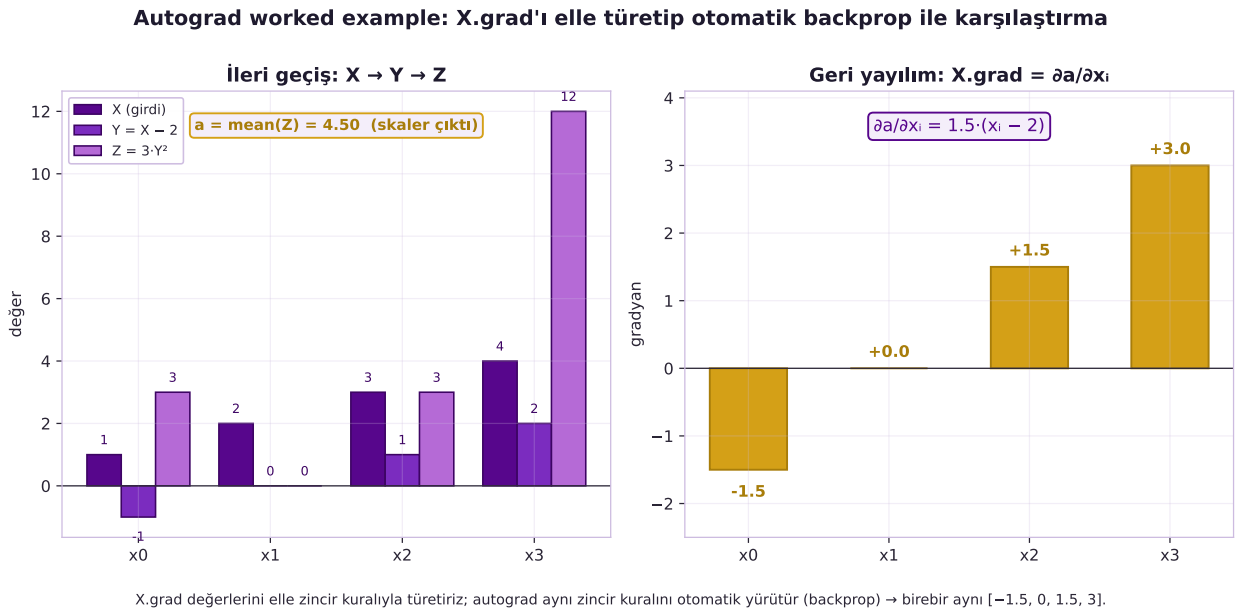
`a.backward()` çağrısı, a'nın X'e göre gradient'ini zincir kuralıyla hesaplar (backprop). Elle:

$$\frac{\partial a}{\partial x_i} = \frac{1}{4} \cdot 3 \cdot 2\,(x_i - 2) = \frac{3}{2}(x_i - 2)$$

$X = [1, 2, 3, 4]$ için bu $[-1.5, 0, 1.5, 3]$ verir — ve `X.grad` tam bunu içerir.

> "backpropagation is how you compute the gradients; how do we train the networks — with gradients. Keep them separate!" — Canziani, 32:04

Canziani'nin uyarısı kritik: **backprop = gradient hesaplama**, **eğitim = gradient kullanma** — ikisini karıştırma. (Bir teknik not: PyTorch `.grad`'ı girdiyle **aynı şekilde** tutar; teorik Jacobian/gradient transpoze farkını pratiklik için göz ardı eder.) Şekil 12.5 bu örneği gerçek değerlerle çizer: solda ileri geçiş ($X \to Y \to Z$, $a = 4.5$), sağda elle türetilen `X.grad` ve autograd'ın birebir aynı $[-1.5, 0, 1.5, 3]$ sonucu.



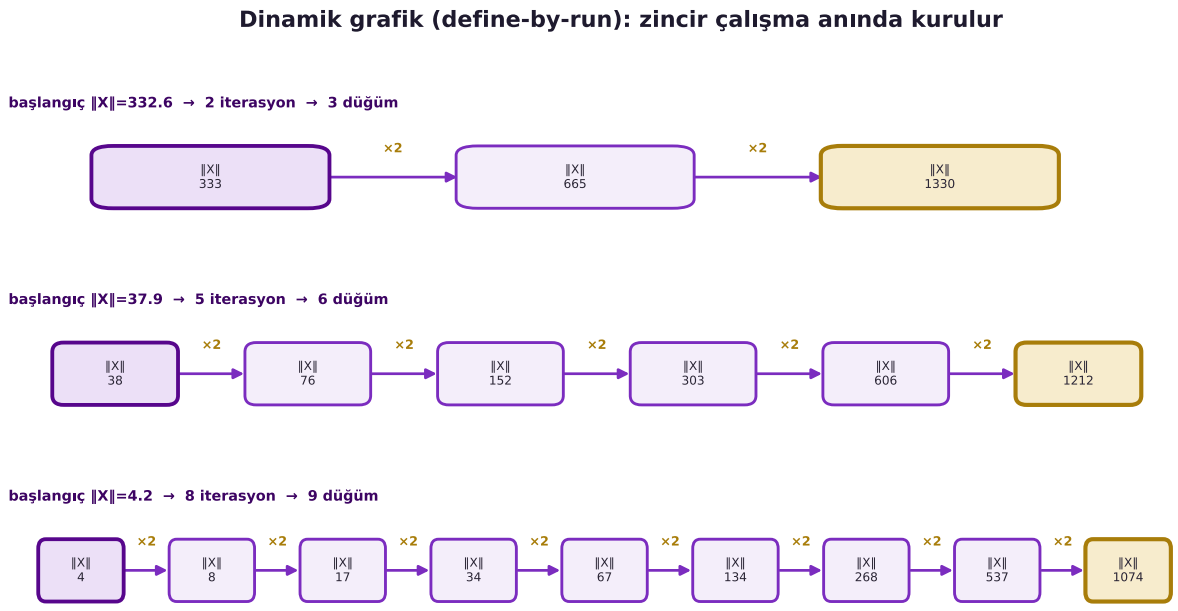
Şekil 12.5: Autograd worked example (Canziani, Bölüm 6): X=[1,2,3,4] üzerinde ileri geçiş Y=X−2 → Z=3·Y^2 → a=mean(Z)=4.5 ve geri yayılımla elde edilen X.grad=[−1.5, 0, 1.5, 3]. Sol panel ileri geçiş değerlerini (violet), sağ panel $\partial a/\partial x_i$ gradyanlarını (gold) gösterir. Gradyan elle $\partial a/\partial x_i = 1.5 \cdot (x_i - 2)$ zincir kuralıyla türetilir; PyTorch autograd aynı zincir kuralını otomatik yürütür (backprop) ve birebir aynı sonucu verir.

> **Builder Notu — backprop ≠ Eğitim**
>
> **Geriye (Hafta 2 + 4):** `a.backward()` = Hafta 2'nin Jacobian zinciri; "backprop ≠ eğitim" ayrımı, Hafta 4'te Defazio'nun da vurguladığı şeydir (Canziani "Aaron'un dün dediği gibi" der — Hafta 4 Defazio dersine atıf).
> **İleriye:** `.backward()` + optimizer (`step()`) = Hafta 2'nin dört satırlık eğitim döngüsü; `.grad`'ın şekli ve birikimi (`zero_grad`) günlük hata kaynaklarıdır.

12.8 (Canziani) Dinamik Hesaplama Grafiği (Define-by-Run)

Canziani PyTorch'un ayırt edici özelliğini gösteriyor: grafik **statik değil, dinamiktir**. Bir örnek: X'i, normu 1000'i aşana kadar bir `while` döngüsünde ikiye katla. Kaç iterasyon döneceği rastgele başlangıca bağlıdır — yani **hesaplama grafiği çalışma anında, koda göre kurulur** (define-by-run).

Bu, normal Python kontrol akışını (döngü, koşul) modelin içinde serbestçe kullanabilmen demektir; autograd yine de doğru gradient'i hesaplar çünkü grafiği sen çalıştırdıkça örer. Şekil 12.6 üç farklı rastgele başlangıç için döngünün farklı sayıda iterasyon döndüğünü ve böylece farklı uzunlukta grafik zincirleri doğduğunu gösterir.



Şekil 12.6: Dinamik grafik (define-by-run): hesap grafiği çalışma anında, kodun akışına göre kurulur. `while X.norm() < 1000: X = X * 2` döngüsü üç farklı rastgele başlangıç için ($\|X\| = 332.6$, 37.9, 4.2) farklı sayıda iterasyonda 1000 eşiğini aşar — sırasıyla 2, 5 ve 8 kez ikiye katlama — ve bu yüzden farklı uzunlukta (3, 6, 9 düğümlü) grafik zincirleri doğar. Her kutu o adımdaki $\|X\|$ değerini, her ok ×2 işlemini, gold kutu eşiği aşan son düğümü gösterir. Python döngü/koşulun model içinde serbestçe çalışabilmesi tam da değişken-uzunluklu hesaba (Hafta 6 RNN) kapı açar; statik grafikte (define-then-run) bu mümkün değildir.

> **Builder Notu — Define-by-Run**
>
> **Geriye (Hafta 2):** Hafta 2'de LeCun "döngü varsa grafik üretimi karmaşıklaşır" demişti; PyTorch'un dinamik grafiği bunu doğal çözer — her ileri geçişte grafiği yeniden kurar.
> **İleriye:** Define-by-run, PyTorch'u araştırmada esnek kılan şeydir (RNN'ler, değişken-uzunluk diziler — Hafta 6); buna karşılık statik grafikler (eski TensorFlow, `torch.compile`/`torch.jit`) hız/optimizasyon için grafiği önceden dondurur.

12.9 Bu Dersin Özeti

1. **Convolution = projeksiyon:** sinyalin kernel'e (aranan yöne) hizası.
2. **Boyut aritmetiği:** 1B kernel topluluğu 3 boyutlu, 2B 4 boyutlu (kernel sayısı × girdi kanalı × [yükseklik ×] genişlik); kernel sayısı = çıktı kanalı.
3. **Output width:** $o = n - k + 1$ (stride 1); her convolution $(k-1)$ boyut kaybeder; padding telafi eder.
4. **PyTorch:** `nn.Conv1d/2d(in, out, kernel_size, stride, padding)`; batch boyutu zorunlu.
5. **autograd:** `requires_grad=True` ile işlemler hesaplama grafiğinde izlenir; `grad_fn` üretici işlemi tutar.
6. **`.backward()` = backprop:** gradient'i zincir kuralıyla hesaplar, `.grad`'a yazar. backprop ≠ eğitim.
7. **Dinamik grafik (define-by-run):** grafik çalışma anında kurulur; döngü/koşul serbest.

! Tek Bir Cümle

Bu hafta convolution'ın boyut aritmetiğini ($o = n-k+1$, 4-boyutlu 2B kernel) ve PyTorch'un autograd motorunu öğrendik — `requires_grad` + `.backward()`, Hafta 2'de LeCun'un elle kurduğu Jacobian zincirini tek satıra indirir, ve dinamik grafik sayesinde bunu döngü/koşul içeren her kodda yapar.

12.10 Kontrol Soruları

i Soru 1: Stride 1'de, girdi boyutu 28 ve kernel boyutu 5 olan bir convolution'ın çıktısı kaç olur? Genel kural nedir? Boyutu korumak için ne yaparsın?

Cevap: Çıktı boyutu:

$$o = n - k + 1 = 28 - 5 + 1 = 24$$

Genel kural: her convolution çıktıyı $(k-1)$ kadar kısaltır (Canziani 17:17). Boyutu korumak için **padding** eklersin — her kenara $(k-1)/2 = 2$ sıfır eklersen çıktı yine 28 olur. (Bu, Hafta 3'ün $\lfloor (n-k)/s \rfloor + 1$ formülünün s=1 hâlidir.)

i Soru 2: 2B convolution kernel topluluğu kaç boyutludur ve bu boyutlar neyi temsil eder?

Cevap: 4 boyutlu (Canziani 26:05): (kernel sayısı × girdi kanalı × yükseklik × genişlik). Kernel sayısı = çıktı kanal sayısı; girdi kanalı = kernel'in "kalınlığı" (her kernel girdinin tüm kanallarına bakar); yükseklik × genişlik = uzamsal pencere. PyTorch'ta `nn.Conv2d(20, 16, (3,5)).weight.shape = (16, 20, 3, 5)`. Ayrıca PyTorch her zaman bir batch boyutu bekler (girdi: batch × kanal × H × W).

i Soru 3: requires_grad=True ne yapar? grad_fn nedir? .backward() ile eğitim arasındaki fark nedir?

Cevap: `requires_grad=True`, PyTorch'a o tensör üzerindeki tüm işlemleri bir **hesaplama grafiğinde izlemesini** söyler, böylece kısmi türevler hesaplanabilir (Canziani 28:53). `grad_fn`, türetilmiş bir tensörü hangi işlemin ürettiğini tutar (örn. $Y = X - 2$ için SubBackward) — Karpathy micrograd'ın `_backward`'ının karşılığı. **`.backward()` = backprop = gradient'leri hesaplama** (zincir kuralı, `.grad`'a yazar); **eğitim = bu gradient'leri kullanıp parametreyi güncelleme** (optimizer.step). Canziani: "ikisini

karıştırma” (32:04).

Soru 4: (Builder) X = [1,2,3,4] (requires_grad), Y = X−2, Z = 3Y^2, a = mean(Z). a.backward() sonrası X.grad nedir? Elle türet.

Cevap: $a = (1/4)\sum 3(x_i - 2)^2$. Zincir kuralı:

$$\frac{\partial a}{\partial x_i} = \frac{1}{4} \cdot 3 \cdot 2\,(x_i - 2) = \frac{3}{2}(x_i - 2)$$

$X = [1, 2, 3, 4]$ için: $[1.5 (-1), 1.5 0, 1.5 1, 1.5 2] = $ **[−1.5, 0, 1.5, 3]**. `X.grad` tam bunu içerir — autograd, elle yaptığımız zincir kuralını otomatik yürütür. (PyTorch `.grad`’ı girdiyle aynı şekilde tutar.)

12.11 Egzersizler

Egzersiz 1 (Boyut aritmetiği). Bir 32×32 görüntüye art arda üç `nn.Conv2d(kernel_size=3, padding=0)` uygula. Her katmandan sonra uzamsal boyut ne olur? `padding=1` ile tekrar dene — fark nedir? Formül $o = n - k + 1$ ile önceden tahmin et.

```
import torch
import torch.nn as nn

x = torch.randn(1, 3, 32, 32)              # (batch, kanal, H, W)
net_nopad = nn.Sequential(
    nn.Conv2d(3, 8, 3), nn.Conv2d(8, 8, 3), nn.Conv2d(8, 8, 3))
net_pad = nn.Sequential(
    nn.Conv2d(3, 8, 3, padding=1),
    nn.Conv2d(8, 8, 3, padding=1),
    nn.Conv2d(8, 8, 3, padding=1))
print(net_nopad(x).shape)                  # 32 -> 30 -> 28 -> 26
print(net_pad(x).shape)                    # padding=1 -> 32 korunur
```

Egzersiz 2 (Kernel şekli). `nn.Conv2d(3, 64, kernel_size=7)` katmanının `.weight.shape` ve `.bias.shape`’ini önce tahmin et, sonra yazdırıp doğrula. Toplam parametre sayısını hesapla.

```
import torch
import torch.nn as nn

conv = nn.Conv2d(3, 64, kernel_size=7)
print(conv.weight.shape)                   # (64, 3, 7, 7)
print(conv.bias.shape)                     # (64,)
params = 64 * 3 * 7 * 7 + 64                # ağırlık + bias
print(params)                              # 9472
```

Egzersiz 3 (autograd worked example). `X = torch.tensor([1.,2.,3.,4.], requires_grad=True)` ile Y=X−2, Z=3*Y**2, a=Z.mean(); `a.backward()` çağır ve `X.grad`'ı yazdır. [−1.5, 0, 1.5, 3] çıktığını ve elle türevle eştiğini doğrula.

```
import torch
X = torch.tensor([1., 2., 3., 4.], requires_grad=True)
Y = X - 2
Z = 3 * Y ** 2
a = Z.mean()
a.backward()
print(X.grad)                                  # tensor([-1.5000, 0.0000, 1.5000, 3.0000])
# elle: da/dxi = (1/4)*3*2*(xi-2) = 1.5*(xi-2)
```

Egzersiz 4 (grad_fn zinciri). Yukarıdaki Y, Z, a tensörlerinin `.grad_fn`'lerini yazdır. Hangi işlemleri görüyorsun (Sub, Mul, Mean)? Bu zincirin Hafta 2'nin Jacobian zinciriyle ilişkisini açıkla.

```
import torch
X = torch.tensor([1., 2., 3., 4.], requires_grad=True)
Y = X - 2;  Z = 3 * Y ** 2;  a = Z.mean()
print(Y.grad_fn)                               # <SubBackward0>
print(Z.grad_fn)                               # <MulBackward0>
print(a.grad_fn)                               # <MeanBackward0>
# her grad_fn = o işlemin yerel Jacobian'i (Hafta 2 zincir kurali halkasi)
```

Egzersiz 5 (Hafta 6 habercisi — diziler). Şimdiye kadar girdiler sabit boyutluydu (görüntü, vektör). Bir cümle ise **değişken uzunlukta bir dizidir** ("film iyiydi" 2 kelime, "film başından sonuna kadar harikaydı" 5 kelime). (a) Sabit-girdili bir ağa değişken uzunlukta diziyi nasıl verirsin — naif bir fikir öner ve neden yetersiz olduğunu açıkla. (b) Dinamik hesaplama grafiği (Bölüm 7), değişken-uzunluk dizileri işlemek için neden uygundur? Bu, Hafta 6'da **RNN ve attention**'a geçişi motive eder.

```
# (a) naif fikir: tum cumleleri sabit L uzunluga pad/truncate et
#     -> kisa cumlede bos token israfi, uzun cumlede bilgi kaybi
# (b) dinamik grafik: her cumle icin grafik o cumlenin uzunluguna gore
#     calisma aninda kurulur -> degisken-uzunluk dogal (Hafta 6 RNN/attention)
sentences = [["film", "iyiydi"],
             ["film", "basindan", "sonuna", "kadar", "harikaydi"]]
for s in sentences:
    print(len(s))                              # 2, 5 -> farkli uzunluk
```

12.12 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H6: Diziler — RNN, LSTM ve Attention

Sabit boyuttan değişken uzunluğa. Bu haftaya kadar sabit boyutlu girdilerle (görüntü, vektör) çalıştık. Hafta 6'da değişken uzunlukta **dizilere** (metin, ses, zaman serisi) geçiyoruz: LeCun RNN ve attention'ı, Canziani PyTorch'ta dizi eğitimini gösterecek. Bu haftanın dinamik hesaplama grafiği (define-by-run) tam da bu değişken-uzunluk dünyasının altyapısıdır — Egzersiz 3-4 (autograd) ve Egzersiz 5 (dizi problemi) tam bu derse hazırlar.

Hafta 6: Diziler — RNN, LSTM ve Attention — LeCun (Lecture) + Canziani (Practicum)

Bu haftaya kadar sabit boyutlu girdilerle (görüntü, vektör) çalıştık. Hafta 6'da değişken uzunlukta **dizilere** (metin, ses, zaman serisi) geçiyoruz: LeCun RNN ve attention'ı, Canziani PyTorch'ta dizi eğitimini gösterecek. Dinamik hesaplama grafiği (bu hafta) tam da bu değişken-uzunluk dünyasının altyapısıdır.

Hafta 6 öncesi yapılacak:

- Egzersiz 3-4 (autograd) ve Egzersiz 5 (dizi problemi sezgisi) çöz.
- "autograd = otomatik backprop (zincir kuralı + hesaplama grafiği)" cümlesini kendi sözcüklerinle yaz.
- Hafta 2-4-5 üçlüsünü bağla: backprop'u elle kurduk (2), kullandık (4), otomatikleştirdik (5).

12.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Convolution = projeksiyon	Sinyalin kernel'e (aranan yöne) hizası	Canziani 1m20
1B kernel topluluğu	3 boyutlu: kernel sayısı × girdi kanalı × boyut	Canziani 12m37
2B kernel topluluğu	4 boyutlu: kernel × kanal × yükseklik × genişlik	Canziani 26m05
Output width	$o = n - k + 1$; her conv $(k-1)$ kaybeder	Canziani 17m17
Padding	Kenara sıfır ekleyip boyut koruma	Canziani 25m20
Batch boyutu zorunlu	PyTorch girdisi batch × kanal × uzamsal	Canziani 26m50
requires_grad	İşlemleri hesaplama grafiğinde izle	Canziani 28m53
grad_fn	Tensörü üreten işlem (örn. SubBackward)	Canziani 29m22
.backward()	Backprop = gradient hesapla; .grad'a yaz	Canziani 32m04
backprop ≠ eğitim	Backprop gradient hesaplar; eğitim onu kullanır	Canziani 32m16
Dinamik grafik	Define-by-run; grafik çalışma anında kurulur	Canziani 37m43

12.14 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Convolution = projeksiyon** → 18.06 iç çarpım + Hafta 4 (satır = kernel).
2. **autograd = zincir kuralı** → Hafta 2 backprop (Jacobian) + Calculus Ders 4 + Karpathy `_backward`.
3. **grad_fn = modül backward** → Hafta 2 "her modülün bir backward'ı var".
4. **Output width** → Hafta 3 stride formülü (s=1).
5. **backprop ≠ eğitim** → Hafta 4 Defazio (aynı uyarı).

İleriye köprüler (production / research):

1. **autograd** → torch.autograd, retain_graph, gradient checkpointing.
2. **Dinamik grafik** → RNN/değişken-uzunluk (Hafta 6); torch.compile (statik optimizasyon).
3. **Convolution boyut yönetimi** → padding/stride/dilation, receptive field hesabı.
4. **Batch-first tensör** → tüm PyTorch veri hattı.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> PyTorch'un autograd'ı sihir değildir — `requires_grad=True` tüm işlemleri bir hesaplama grafiğinde izler, `.backward()` ise Hafta 2'de LeCun'un elle kurduğu zincir kuralını (Jacobian zinciri) otomatik yürütüp `.grad`'a yazar; convolution boyutları ise basit bir aritmetiktir ($o = n - k + 1$). Backprop'u önce elle anladık (Hafta 2), sonra kullandık (Hafta 4), şimdi de PyTorch'a otomatik yaptırıyoruz — ve dinamik grafik, bunu döngü/koşul içeren her kodda mümkün kılarak bizi değişken-uzunluk dizilere (Hafta 6) hazırlıyor.

13 Diziler — RNN, LSTM ve Attention

İki hocalı hafta: Yann LeCun (Lecture) yinelemeli ağların (RNN) mimarisini, vanishing/exploding gradient sorununu ve çözümlerini (gating — GRU/LSTM — ve attention) kurar; Alfredo Canziani (Practicum) RNN'in dört tipini, seq2seq encoder-decoder'ı ve PyTorch'ta dizi eğitimini somutlaştırır — sabit boyutlu girdiden değişken uzunluklu diziye geçiş.

> **Bölüm bilgisi**
>
> - **LeCun'un Lecture videosu:** YouTube — RNNs, attention, and memory networks (≈89 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Training recurrent neural networks (≈53 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈25 dk

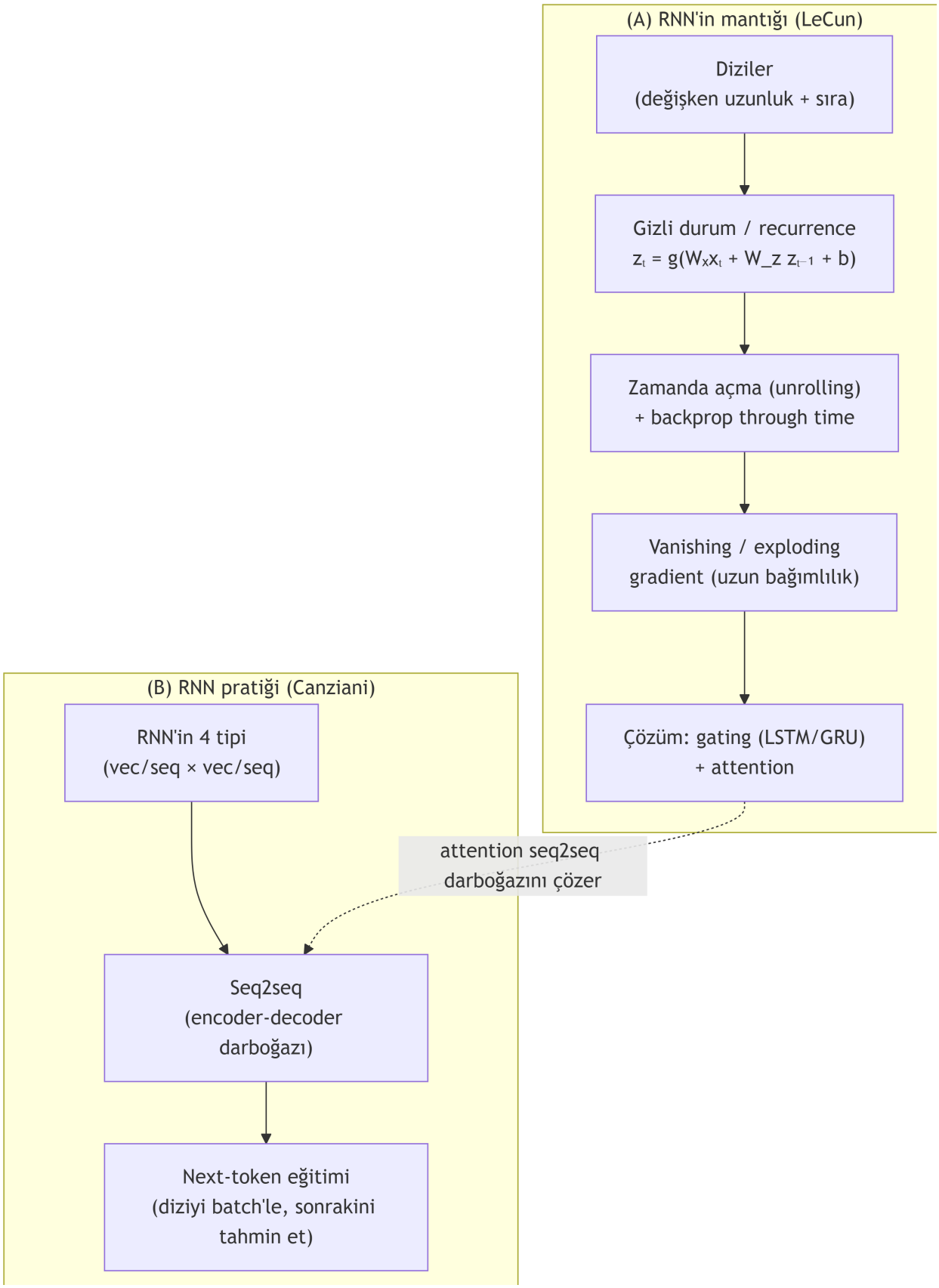
13.1 Bu Derste Ne Var?

Hafta 5'in sonunda bıraktığımız soru: girdiler hep **sabit boyutluydu** (görüntü, vektör) — ama bir cümle, bir ses dalgası, bir zaman serisi **değişken uzunlukta bir dizidir**. Bu hafta dizileri işleyen mimarilere geçiyoruz: **yinelemeli ağlar (RNN)**, onların hafıza varyantları **LSTM/GRU**, ve modern dünyayı kuran fikir **attention**.

Yine iki hocalı. **Yann LeCun** (Lecture) RNN'in mimarisini ve temel sorununu (vanishing gradient) kurar, sonra çözümleri — GRU/LSTM ve attention — anlatır. **Alfredo Canziani** (Practicum) RNN'in dört tipini (girdi/çıktı dizi mi vektör mü) ve eğitimini PyTorch'ta somutlaştırır.

Bu haftanın üç ana fikri:

1. **RNN = zaman içinde paylaşılan bir katman.** Bir **gizli durum (hidden state)** geçmişi taşır; aynı ağırlıklar her zaman adımında tekrar kullanılır — recurrence $z_t = g(W_x x_t + W_z z_{t-1} + b)$.
2. **Naif RNN'ler uzun bağımlılıkta başarısız** (vanishing/exploding gradient); çözüm **gating + memory cell** (GRU, LSTM).
3. **Attention**, ağın hangi girdiye **odaklanacağını** öğrenmesidir — öğrenilen ağırlıklı birleşim $c = w_1 v_1 + w_2 v_2$ ($w_2 = 1 - w_1$), transformer'ların (Hafta 12) çekirdeği.


(A) RNN'in mantığı (LeCun)
Diziler
(değişken uzunluk + sıra)
Gizli durum / recurrence
$z_t = g(W_x x_t + W\_z\, z_{t-1} + b)$
Zamanda açma (unrolling)
+ backprop through time
Vanishing / exploding
gradient (uzun bağımlılık)
Çözüm: gating (LSTM/GRU)
+ attention
attention seq2seq
darboğazını çözer
(B) RNN pratiği (Canziani)
RNN'in 4 tipi
(vec/seq × vec/seq)
Seq2seq
(encoder-decoder
darboğazı)
Next-token eğitimi
(diziyi batch'le, sonrakini
tahmin et)

Builder Notu — Diziler = Değişken Uzunluk

Geriye (önkoşul kurslar):

- **RNN recurrence** → Hafta 2 "afin + nonlinearite" (döndür-ez) atomunun **zamanda** tekrarı; ağırlık paylaşımı Hafta 3 convolution'ın zaman eksenindeki kuzeni.
- **Vanishing gradient** → Hafta 2 backprop (Jacobian zinciri) + Hafta 4 (doygun nonlinearite türevi küçültür) + Calculus zincir kuralı.
- **Hidden state** → Stat 110 Markov (durum geçmişi özetler).

İleriye (production / research):

- Attention → transformer (Hafta 12), modern dil modellerinin tamamı.
- Seq2seq encoder-decoder → makine çevirisi, konuşma tanıma, her dizi-dizi görevi.

Tek cümleyle: RNN, aynı katmanı bir gizli durumla zaman içinde tekrar uygulayarak değişken uzunlukta dizileri işler; ama uzun bağımlılıkta vanishing gradient'e takılır — bu yüzden gating (LSTM/GRU) ve attention icat edildi.

13.2 (LeCun) Diziler Neden Farklı?

Şimdiye kadarki ağlar sabit boyutlu girdi alıyordu. Ama gerçek dünyanın çoğu **dizidir**: metin (kelime dizisi), ses (örnek dizisi), video (kare dizisi). İki zorluk: (1) **değişken uzunluk** — "film iyiydi" 2 kelime, başka cümle 20 kelime; sabit-girdili bir ağ bunu alamaz. (2) **sıra önemlidir** — kelimelerin sırası anlamı değiştirir.

LeCun'un çerçevesi: bir diziyi işlemek, onu okurken **bir özet (gizli durum) tutmak** ve her yeni öğeyle bu özeti güncellemektir. "En olası karakter dizisi nedir?" gibi sorular (dil modelleme, posta kodu okuma) bu yapıyı gerektirir.

Builder Notu — Değişken Uzunluk = Hafta 5'in Cevabı

Geriye (Hafta 5): Bu, Hafta 5 Egzersiz 5'in cevabıdır: değişken-uzunluk diziyi sabit-girdili ağa veremezsin; gizli durumlu bir tekrar gerekir. Dinamik hesaplama grafiği (Hafta 5) tam da bunu mümkün kılar.
İleriye: "Diziyi okurken durum tut" fikri; RNN'den transformer'a, durum-uzayı modellerine (Mamba) kadar tüm dizi modellemenin ortak temasıdır.

13.3 (LeCun) RNN Mimarisi: Gizli Durum ve Zamanda Açma

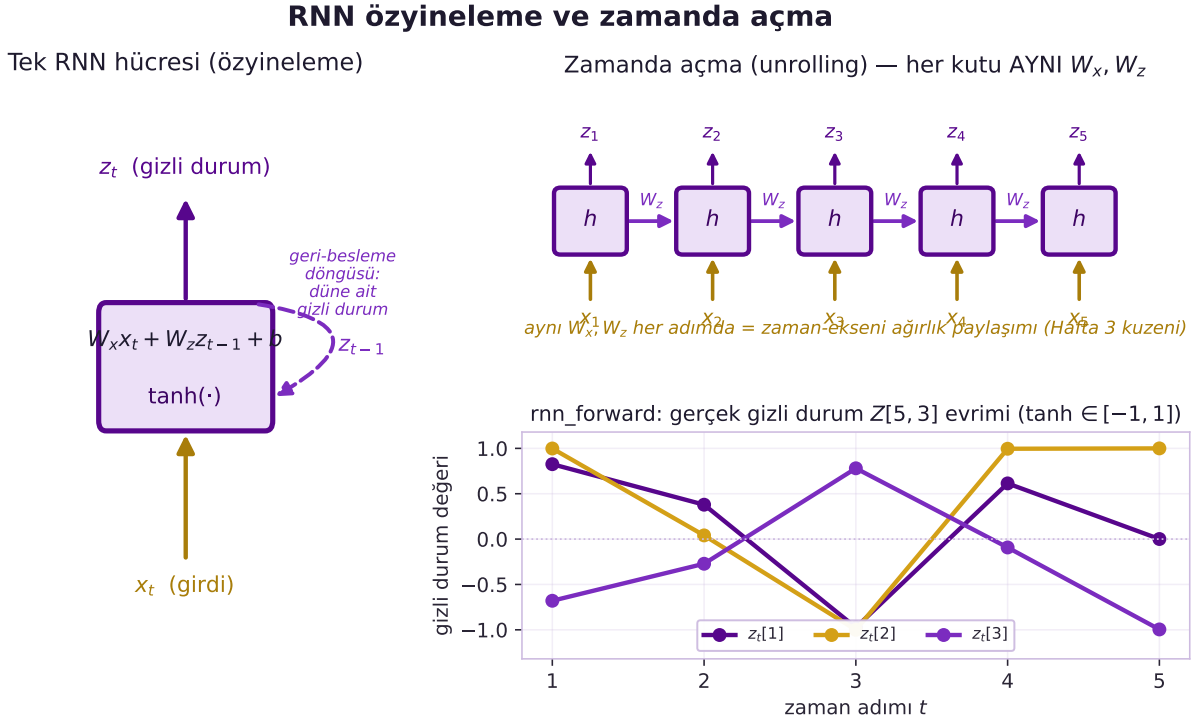
Bir RNN, her zaman adımında **aynı** yinelemeli katmanı uygular. t adımında, katman hem o anki girdiyi x_t hem de bir önceki **gizli durumu** z_{t-1} alır ve yeni durumu üretir:

$$z_t = g(W_x\, x_t + W_z\, z_{t-1} + b)$$

Buradaki W_x, W_z her adımda **paylaşılır** (Hafta 3'ün ağırlık paylaşımının zaman eksenindeki hâli). Gizli durum z_t, dizinin o ana kadarki "özetidir".

LeCun bunu **zamanda açma (unrolling)** ile görselleştiriyor: yinelemeli ağı, her zaman adımı için bir kopya olacak şekilde düz bir ağa açarsın; sonra normal backprop uygularsın — buna **zaman içinde geri yayılım (backprop through time)** denir. Şekil 13.1 bu iki bakışı bir arada gösterir: solda geri-besleme döngülü tek hücre, sağ-üstte aynı hücrenin zamanda açılmış (her kutu PAYLAŞILAN W_x, W_z) hâli, sağ-altta `rnn_forward` ile hesaplanan gerçek gizli durum dizisi.

> "this technique of unrolling and then back propagating [is how you train recurrent nets]." — LeCun, 47:37



Şekil 13.1: RNN özyineleme ve zamanda açma: Sol panelde tek bir RNN hücresi — girdi x_t ile bir önceki gizli durum z_{t-1} afin dönüşüme ($W_x x_t + W\_z\, z_{t-1} + b$) girer, tanh ile sıkıştırılır ve yeni gizli durum z_t üretilir; kesikli geri-besleme döngüsü z_t çıkışını bir sonraki adımda z_{t-1} olarak hücreye geri verir. Sağ-üst panelde aynı hücre zamanda açılır (unrolling): her kutu PAYLAŞILAN aynı W_x, W_z ağırlıklarını kullanır — bu, zaman ekseni boyunca ağırlık paylaşımıdır (Hafta 3'teki uzamsal evrişim paylaşımının kuzeni). Sağ-alt panelde rnn_forward ile hesaplanan gerçek beş-adımlık gizli durum dizisi Z[5,3]: üç birimin değerleri tanh sayesinde [−1, 1] aralığında kalır ve adımdan adıma evrilir.

> **Builder Notu — Zamanda Döndür-Ez**
>
> **Geriye (Hafta 2):** Açılmış RNN, sıradan bir derin ağdır; eğitimi Hafta 2'nin backprop'unun (Jacobian zinciri) ta kendisidir — yalnızca ağırlıklar zaman boyunca paylaşılır.
> **İleriye:** Unrolling, değişken-uzunluk için dinamik grafik (Hafta 5) ister; çok uzun dizilerde "truncated

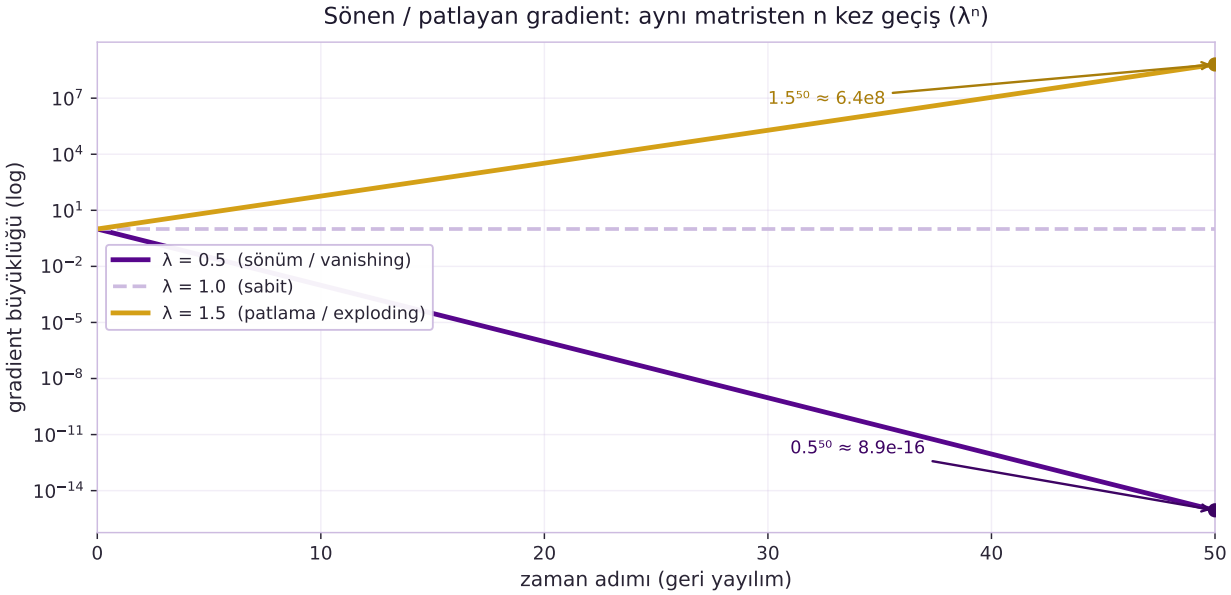
backprop through time" ile pencerelenir.

13.4 (LeCun) Vanishing/Exploding Gradient

Naif RNN'in temel sorunu: uzun dizilerde gradient ya **kaybolur (vanishing)** ya da **patlar (exploding)**. Sebep, backprop through time'da gradient'in aynı yinelemeli matristen defalarca geçmesidir; 50 adım sonra bu çarpımlar ya sıfıra iner ya sonsuza gider. Şekil 13.2 bunu tek bir özdeğer λ üzerinden somutlaştırır: aynı matristen n kez geçen gradient λ^n olarak ölçeklenir — $\lambda < 1$ söner, $\lambda = 1$ sabit kalır, $\lambda > 1$ patlar.

> "by the time you get to the 50th time step… [you hit] the vanishing gradient problem." — LeCun, 49:15

Sonuç: gizli durum z_t teoride bilgiyi uzun süre saklayabilmeli, ama vanishing gradient yüzünden naif RNN bunu pratikte **yapamaz** — uzun bağımlılıkları öğrenemez.



Şekil 13.2: Sönen/patlayan gradient: aynı matristen n kez geçen gradient λ^n olarak ölçeklenir. Log-y ekseninde λ=0.5 (violet) sönerek 50. adımda $0.5^{50} \approx$ 8.9e-16'ya iner (vanishing), λ=1.0 (gri kesikli) sabit kalır, λ=1.5 (gold) patlayarak $1.5^{50} \approx$ 6.4e8'e çıkar (exploding). Naif RNN, aynı geçiş matrisinden birçok kez geçtiği için uzun bağımlılığı öğrenemez.

Builder Notu — Vanishing = Uzun Zincir

Geriye (Hafta 2 + 4): Vanishing gradient, Hafta 2'nin zincir kuralının ve Hafta 4'ün "doygun nonlinearite türevi gradient'i küçültür" gözleminin uzun-zincir hâlidir; aynı matrisin tekrar tekrar çarpılması özdeğerleri 1'den uzaksa patlama/sönme yaratır (18.06).
İleriye: Çözümler: gating (bu hafta GRU/LSTM), gradient clipping (exploding için), residual bağlantılar,

ve nihayetinde attention (uzun bağımlılığı tek adımda kurar).

13.5 (LeCun) Attention: Hangi Girdiye Odaklan?

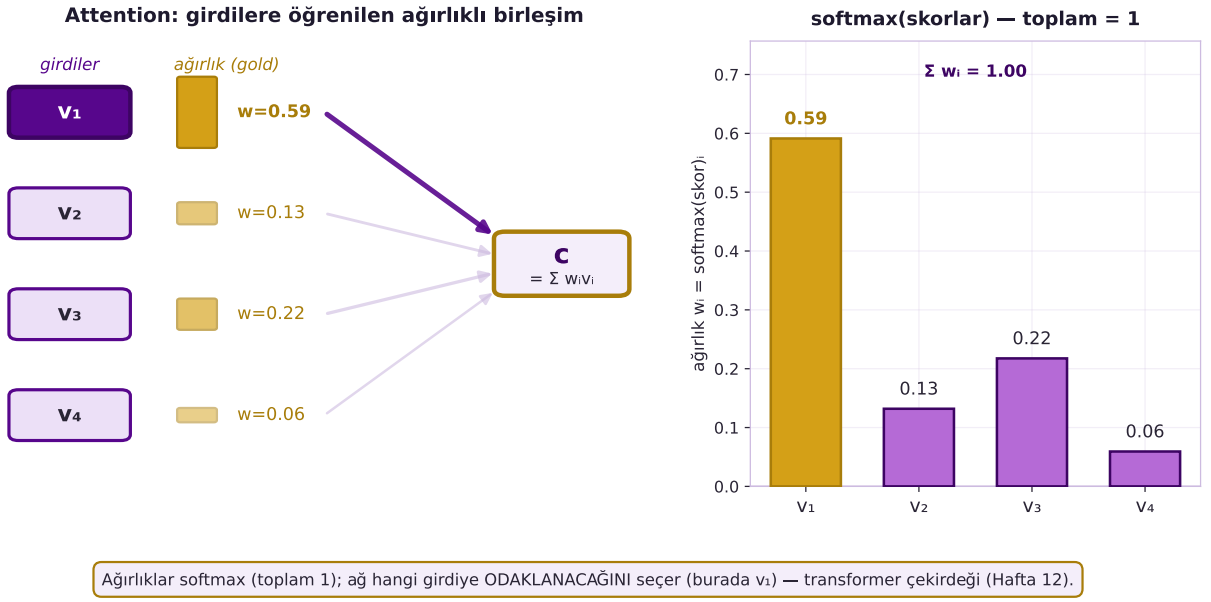
LeCun attention'ı bir **ağırlıklı birleşim** olarak tanıtıyor: ağ, birkaç girdiyi sabit ağırlıklarla değil, **kendi seçtiği** ağırlıklarla birleştirir. En basit hâlde iki girdi için ağırlıklar toplamı 1'dir:

$$c = w_1 v_1 + w_2 v_2, \qquad w_2 = 1 - w_1$$

Bu ağırlıkları başka bir ağ üretir; böylece model bir girdiye **odaklanıp** ötekileri **görmezden gelebilir**. Şekil 13.3 bunu somutlaştırır: ham hizalama skorları softmax'tan geçirilir, toplamı 1 olan bir odak dağılımı çıkar, ve çıktı $c = \sum_i w_i v_i$ ağırlıklı toplamıdır — ağ en yüksek ağırlığı vererek hangi girdiye odaklanacağını seçer.

> "[attention] allows a neural net to basically focus its attention on a particular input and ignoring the others." — LeCun, 57:53

LeCun'un kritik notu: transformer mimarileri ve her tür attention, bu basit "öğrenilen ağırlıklı birleşim" trick'ini kullanır. Yani Hafta 12'nin transformer'ı, bu çekirdeğin ölçeklenmiş hâlidir.



Şekil 13.3: Attention öğrenilen ağırlıklı birleşim olarak. Ham hizalama skorları softmax'tan geçirilir → ağırlıklar w_i (toplam 1, bir olasılık/odak dağılımı). Çıktı, girdi vektörlerinin ağırlıklı toplamıdır: $c = \Sigma\, w_i v_i$. Ağ, en yüksek ağırlığı vererek hangi girdiye ODAKLANACAĞINI seçer (burada v_1, w=0.59); bu mekanizma transformer'ın çekirdeğidir (Hafta 12).

Builder Notu — Attention = Softmax Odak

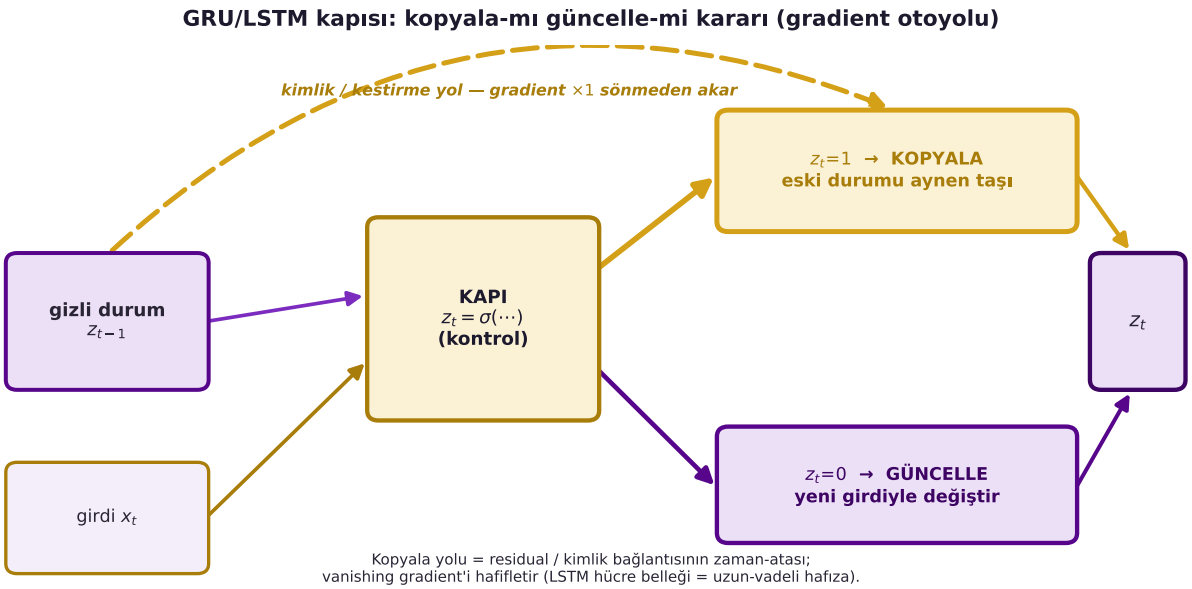
Geriye (Hafta 1): Attention ağırlıkları toplamı 1 olan bir olasılık dağılımıdır (softmax) — Stat 110 + Hafta 1 softmax. "Odaklan" = yüksek ağırlık ver.
İleriye: Attention, RNN'in sıralı darboğazını aşar (her konum her konuma doğrudan bakar); transformer (Hafta 12), self-attention'ın çok-başlı, ölçekli hâlidir.

13.6 (LeCun) GRU ve LSTM: Gating ve Memory Cell

Vanishing gradient'i çözmenin ana yolu **gating**'dir. **GRU** (Cho, 2014) ve **LSTM** (Hochreiter & Schmidhuber, 1997), gizli duruma bir **memory cell** (hafıza hücresi) ve bunu kontrol eden **kapılar (gates)** ekler. Bir gating vektörü z_t, durumu ne kadar güncelleyeceğini belirler: uçta $z_t = 1$ ise eski durumu **olduğu gibi kopyalar**, girdiyi yok sayar — yani bir hafıza gibi davranır. Şekil 13.4 bu kararı bir gradient otoyolu olarak çizer: kopyala yolu (gold) eski durumu sönmeden taşır, güncelle yolu (violet) gizli durumu yeni girdiyle değiştirir.

> "if Z equals 1, it just copies its previous state and ignores the input, so it acts like a memory essentially." — LeCun, 1:01:55

Bu "kopyala" yolu, gradient'in uzun mesafede **bozulmadan** akmasını sağlar (vanishing gradient'i hafifletir). LSTM (1997) aynı sorunu daha erken çözmüştü; GRU daha sade bir varyanttır.



Şekil 13.4: GRU/LSTM kapısının (gating) şematik gösterimi. Önceki gizli durum z_{t-1} ve girdi x_t merkezdeki KAPI'ya (kontrol, $z_t = \sigma(\cdots)$) girer. Kapı iki yol arasında karar verir: $z_t = 1$ ise KOPYALA yolu (gold otoyol) eski durumu aynen taşır, girdiyi yoksayar — bu kestirme/kimlik bağlantısı sayesinde gradient $\times 1$ ile sönmeden geriye akar; $z_t = 0$ ise GÜNCELLE yolu (violet) gizli durumu yeni girdiyle değiştirir. Kopyala yolu, residual/kimlik bağlantısının zaman-atasıdır ve LSTM hücre belleğindeki uzun-vadeli hafızayı mümkün kılarak vanishing gradient'i hafifletir.

Builder Notu — Memory Cell = Residual'in Atası

Geriye (Hafta 2): Memory cell'in "kopyala" yolu, bir tür residual/kimlik bağlantısıdır — gradient'i 1 ile çarparak (sönmeden) taşır; bu, derin ağlardaki residual bağlantının (Hafta 2 ileriye notu) zaman eksenindeki atasıdır.
İleriye: LSTM/GRU 2010'larda NLP/konuşmanın standardıydı; transformer'lar (Hafta 12) çoğu görevde onları geçti, ama uzun-bağlam ve verimlilik için RNN-benzeri fikirler (SSM, Mamba) geri dönüyor.

13.7 Geçiş: LeCun'dan Canziani'ye

LeCun RNN'in *neden*ini ve *sorunlarını* (vanishing gradient → gating, attention) kurdu. Şimdi **Canziani** RNN'i pratiğe indiriyor: bir RNN'in girdi/çıktısı dizi mi vektör mü olduğuna göre **dört farklı tipi**, ve PyTorch'ta nasıl eğitildiğini gösteriyor. LeCun teoriyi, Canziani şekli ve kodu veriyor.

13.8 (Canziani) RNN'in Dört Tipi

Canziani RNN'leri girdi ve çıktının **dizi mi vektör mü** olduğuna göre dörde ayırıyor:

1. **Vektör → Dizi:** tek girdi, dizi çıktı (örn. görüntü → altyazı/caption).
2. **Dizi → Vektör:** dizi girdi, tek çıktı (örn. cümle → duygu sınıfı; ara çıktıları umursamaz, yalnızca sondaki).
3. **Dizi → Dizi (eşzamanlı):** her girdiye bir çıktı (örn. etiketleme).
4. **Dizi → Vektör → Dizi:** dizi önce tek bir vektöre sıkışır, sonra yeni bir dizi üretir (örn. çeviri — birazdan).

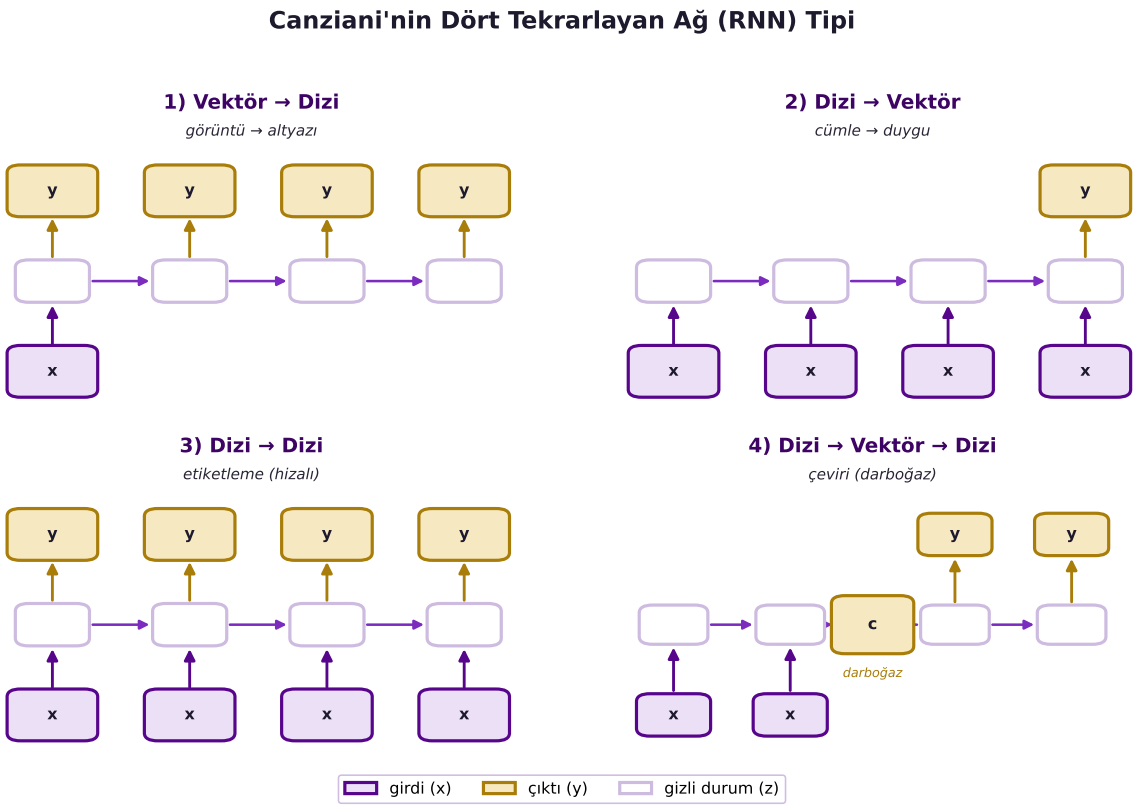
Bu dört kalıp, "değişken-uzunluk girdi/çıktıyı nasıl kurarsın?" sorusunun tüm yanıtlarını kapsar. Şekil 13.5 dördünü tek bir 2×2 panelde, NYU violet girdi ve gold çıktı renk anahtarıyla yan yana koyar.

Builder Notu — 4 Tip Taksonomi

Geriye (Hafta 5): Her tip, değişken-uzunluk diziyi işlemek için dinamik grafik (Hafta 5) ister; gizli durum, Hafta 2'nin "döndür-ez" katmanının zamanda tekrarıdır.
İleriye: Bu taksonomi (vec/seq giriş × vec/seq çıkış) bugün de geçerli; transformer'lar aynı dört kalıbı encoder/decoder kombinasyonlarıyla kurar.

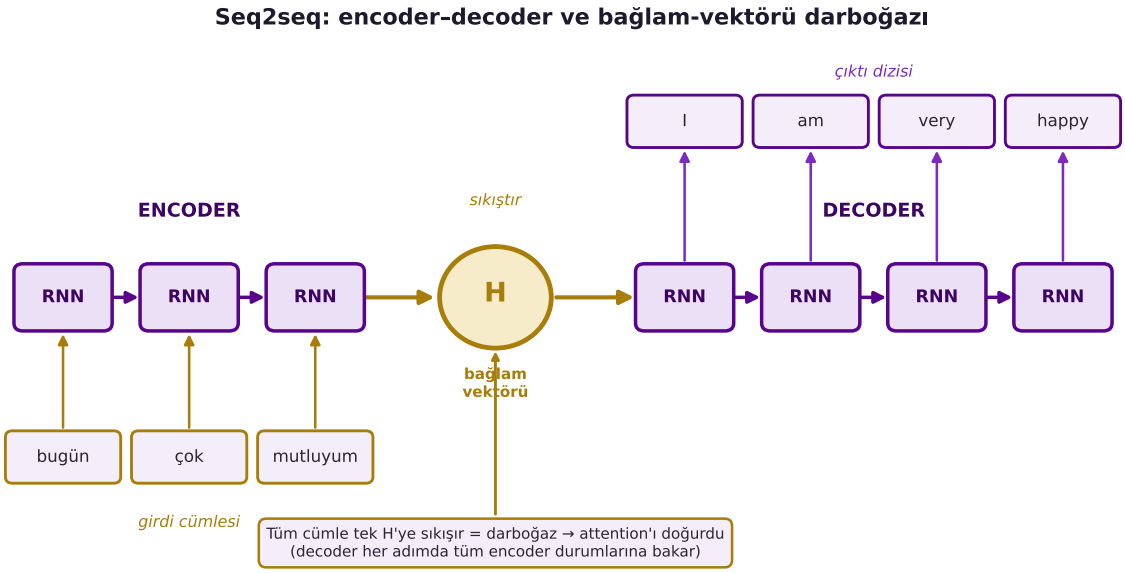
13.9 (Canziani) Seq2Seq: Encoder-Decoder ve Çeviri

Dördüncü tip — **dizi → vektör → dizi** — makine çevirisinin klasik kalıbıdır (seq2seq). Bir **encoder** RNN, girdi cümlesini (örn. "bugün çok mutluyum") tek bir gizli vektöre **sıkıştırır** (H); bir **decoder** RNN bu vektörden hedef dildeki diziyi (İngilizce çeviri) **üretir**. Şekil 13.6 bu mimariyi ve bağlam-vektörü darboğazını çizer: tüm cümlenin tek H'ye sıkışması, decoder'ın her adımda tüm encoder durumlarına bakmasını sağlayan attention'ı doğurdu.



Şekil 13.5: Canziani'nin dört tekrarlayan ağ (RNN) tipi: (1) Vektör→Dizi (görüntü→altyazı), (2) Dizi→Vektör (cümle→duygu), (3) Dizi→Dizi hizalı etiketleme, (4) Dizi→Vektör→Dizi çeviri (darboğaz). NYU violet girdi, gold çıktı, açık gizli durum.

Canziani'nin gizli temsil tanımı, Hafta 2'nin atomunun tekrarıdır: gizli katman = girdinin afin dönüşümü + önceki gizlinin afin dönüşümü, sonra nonlinearite ("döndür-ez", ama bu kez iki kaynaktan).



Şekil 13.6: Seq2seq encoder–decoder mimarisi ve bağlam-vektörü darboğazı: girdi cümlesi ('bugün', 'çok', 'mutluyum') encoder RNN zincirinden geçip tek bir H bağlam vektörüne sıkıştırılır, decoder RNN zinciri bu H'den çıktı dizisini ('I', 'am', 'very', 'happy') üretir. Tüm cümlenin tek vektöre sıkışması bir darboğaz oluşturur ve bu sınırlama, decoder'ın her adımda tüm encoder durumlarına bakmasını sağlayan attention mekanizmasını doğurdu.

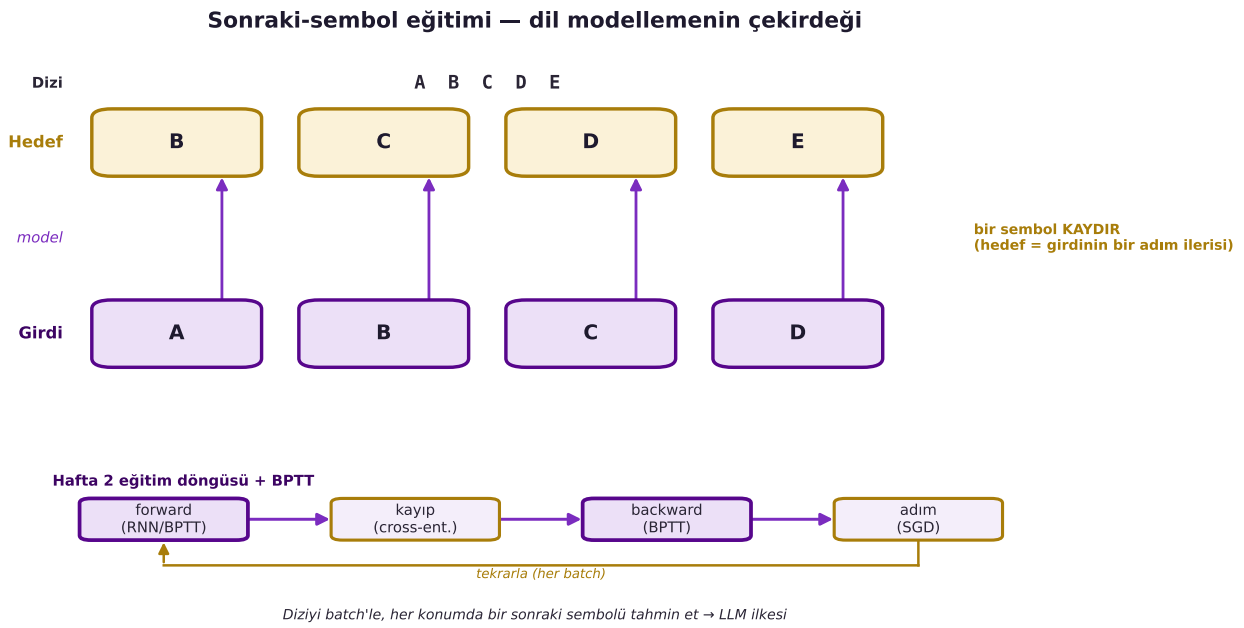
Builder Notu — Encoder-Decoder Darboğazı

Geriye (Hafta 2): RNN gizli durumu = nonlinear(afin(girdi) + afin(önceki durum)) — Hafta 2'nin "döndür-ez" atomunun iki-girişli, zamanda-tekrarlı hâli.
İleriye: Encoder-decoder darboğazı (tüm cümleyi tek vektöre sıkıştırmak) attention'ı doğurdu (decoder her adımda tüm encoder durumlarına bakar); bu, transformer'a giden yoldur.

13.10 (Canziani) RNN Eğitimi: Diziyi Batch'lere Bölmek

Canziani RNN eğitimini PyTorch'ta gösteriyor. Çok uzun bir diziyi (örn. bir metin) tek seferde işlemek pratik değildir; bu yüzden uzun dizi **parçalara (batch)** bölünür. Model bir parçayı görür ve bir sonraki karakter/sembol dizisini tahmin etmeye **zorlanır** (örn. "ABC" verilince "BCD" üret).

Bu, dil modellemenin çekirdeğidir: bir diziyi oku, bir sonrakini tahmin et — Hafta 1'in "next token" sezgisinin RNN hâli. Eğitim yine Hafta 2'nin döngüsüdür (forward → loss → backward → step); tek fark, backward'ın zamanda açılmış ağ boyunca akmasıdır (backprop through time). Şekil 13.7 bu "bir kaydır" hedefini (girdi A B C D → hedef B C D E) ve altındaki Hafta 2 eğitim döngüsünü (forward → kayıp → backward → adım) bir arada gösterir.



Şekil 13.7: Sonraki-sembol eğitimi, dil modellemenin çekirdeğidir. Üstte hizalanmış iki satır: alt satır girdi (A B C D), üst satır hedef — girdinin bir sembol kaydırılmış hâli (B C D E); oklar her konumda "bir sonrakini tahmin et" eşlemesini gösterir. Altta Hafta 2 eğitim döngüsü forward → kayıp (cross-entropy) → backward → adım (SGD), zaman içinde geri yayılım (BPTT) ile her batch için tekrarlanır. Diziyi batch'le ve her konumda bir sonraki sembolü tahmin et — bu, büyük dil modellerinin (LLM) çalışma ilkesidir.

Builder Notu — Next-Token = LLM İlkesi

Geriye (Hafta 2 + 5): Eğitim döngüsü Hafta 2'nin aynısı; `loss.backward()` (Hafta 5 autograd), açılmış RNN'in zaman-zincirini otomatik yürütür.
İleriye: "Bir sonrakini tahmin et" hedefi (next-token prediction), bugünkü tüm büyük dil modellerinin eğitim ilkesidir; RNN'den transformer'a değişen mimari, ilke değil.

13.11 Bu Dersin Özeti

1. **Diziler değişken uzunlukta + sıralıdır;** sabit-girdili ağ yetmez. RNN, bir gizli durumla diziyi okur.
2. **RNN recurrence:** $z_t = g(W_x x_t + W_z z_{t-1} + b)$; ağırlıklar zaman boyunca paylaşılır. Eğitim = unrolling + backprop through time.
3. **Vanishing/exploding gradient:** naif RNN uzun bağımlılığı öğrenemez (aynı matristen tekrar tekrar geçen gradient söner/patlar).
4. **GRU/LSTM:** gating + memory cell; $z_t = 1$ ile durumu kopyala → gradient bozulmadan akar.
5. **Attention:** öğrenilen ağırlıklı birleşim ($w_2 = 1 - w_1$); ağ hangi girdiye odaklanacağını seçer → transformer'ın çekirdeği.
6. **Canziani:** RNN'in 4 tipi (vec/seq × vec/seq); seq2seq encoder-decoder (çeviri); eğitim = diziyi batch'le, bir sonrakini tahmin et.

Tek Bir Cümle

RNN, aynı "döndür-ez" katmanını bir gizli durumla zaman içinde tekrar uygulayarak değişken-uzunluk dizileri işler; naif hâli vanishing gradient'e takılır, bu yüzden gating (LSTM/GRU, "kopyala" yolu) ve attention ("hangi girdiye odaklan") icat edildi — ve attention, Hafta 12'nin transformer'ının çekirdeğidir.

13.12 Kontrol Soruları

Soru 1: RNN recurrence denklemini yaz. "Ağırlık paylaşımı" burada ne anlama gelir, hangi önceki haftayla bağlantılı?

Cevap: Yineleme:

$$z_t = g(W_x\, x_t + W_z\, z_{t-1} + b)$$

z_t = gizli durum (dizinin o ana kadarki özeti), x_t = o anki girdi, z_{t-1} = önceki durum. **Ağırlık paylaşımı:** W_x ve W_z **her zaman adımında aynıdır** — tıpkı Hafta 3'te convolution kernel'inin her uzamsal konumda paylaşılması gibi, ama bu kez **zaman ekseninde**. Bu, hem parametreyi sınırlar hem de modelin her konumda aynı işlemi yapmasını sağlar.

Soru 2: Vanishing gradient problemi nedir, neden olur? Naif RNN'i nasıl sınırlar?

Cevap: Backprop through time'da gradient, aynı yinelemeli matristen (W_z) ve nonlinearite türevinden **defalarca** geçer. Bu matrisin özdeğerleri 1'den küçükse gradient katlanarak **söner** (vanishing), büyükse **patlar** (exploding) — 50 adım sonra pratik olarak sıfır/sonsuz olur (LeCun 49:15). Sonuç: gizli durum teoride bilgiyi uzun süre tutabilmeli, ama gradient ulaşamadığı için naif RNN **uzun bağımlılıkları öğrenemez**. (Hafta 2 zincir kuralı + Hafta 4 doygun türev'in uzun-zincir hâli.)

Soru 3: GRU/LSTM vanishing gradient'i nasıl hafifletir? "Z=1" durumu ne yapar?

Cevap: GRU/LSTM, gizli duruma bir **memory cell** ve onu kontrol eden **kapılar (gates)** ekler. Bir gating vektörü z_t, durumu ne kadar güncelleyeceğini belirler. Uçta $z_t = 1$ **ise model eski durumu olduğu gibi kopyalar, girdiyi yok sayar** (LeCun 1:01:55) — yani bir hafıza gibi davranır. Bu "kopyala" yolu, gradient'i 1 ile çarparak (sönmeden) uzun mesafede taşır — residual/kimlik bağlantısının zaman eksenindeki atası. Böylece uzun bağımlılıklar öğrenilebilir.

Soru 4: (Builder) Attention'ın en basit hâlini yaz. "Odaklanma" ne demek? Hangi gelecek mimariyle bağlantılı?

Cevap: En basit attention, öğrenilen ağırlıklarla bir birleşimdir:

$$c = w_1 v_1 + w_2 v_2, \qquad w_2 = 1 - w_1$$

Ağırlıkları (w_1) başka bir ağ üretir ve toplamları 1'dir (softmax → olasılık dağılımı). "Odaklanma" = bir girdiye yüksek ağırlık verip ötekileri görmezden gelmek (LeCun 57:53). Bu trick, **transformer**'ın (Hafta 12) çekirdeğidir: self-attention, her token'ın tüm token'lara öğrenilen ağırlıklarla bakmasıdır — RNN'in sıralı darboğazını aşar.

13.13 Egzersizler

Egzersiz 1 (RNN elle). Bir RNN hücresini NumPy ile yaz: $z_t = \tanh(W_x x_t + W_z z_{t-1} + b)$. 3 zaman adımlı kısa bir dizi için $z_0 = 0$'dan başlayıp z_1, z_2, z_3'ü elle hesapla. Aynı W_x, W_z'nin her adımda kullanıldığını gözlemle (ağırlık paylaşımı).

```
import numpy as np

Wx = np.array([[0.5, -0.3], [0.2, 0.4]])    # girdi -> gizli (2x2)
Wz = np.array([[0.1, 0.6], [-0.2, 0.3]])    # gizli -> gizli (2x2, PAYLASILAN)
b = np.array([0.0, 0.0])
X_seq = np.array([[1.0, 0.0], [0.5, 1.0], [-1.0, 0.5]])   # 3 zaman adimi

z = np.zeros(2)                              # z0 = 0
for t, x in enumerate(X_seq):
    z = np.tanh(Wx @ x + Wz @ z + b)         # ayni Wx, Wz her adimda
    print(f"z{t+1} =", np.round(z, 3))       # z1, z2, z3
```

Egzersiz 2 (Vanishing gradient). $W_z = 0.5 \cdot I$ (özdeğer 0.5) ile 50 adım çarp: $0.5^{50} \approx ?$ Sonra $W_z = 1.5 \cdot I$ ile: $1.5^{50} \approx ?$ Birincisi neden vanishing, ikincisi neden exploding gradient'i temsil eder, açıkla.

```
# Ayni matristen 50 kez gecen gradient: lambda^50
print(0.5 ** 50)     # ~8.9e-16 -> sifira iner (VANISHING)
print(1.0 ** 50)     # 1.0      -> sabit
print(1.5 ** 50)     # ~6.4e8   -> patlar  (EXPLODING)
# lambda<1: gradient soner -> uzak gecmis ogrenilemez
# lambda>1: gradient patlar -> egitim kararsiz (gradient clipping gerekir)
```

Egzersiz 3 (4 tip). Şu görevleri RNN'in 4 tipinden hangisine eşle: (a) film yorumu → yıldız sayısı, (b) görüntü → altyazı, (c) İngilizce cümle → Türkçe cümle, (d) her kelimeye kelime-türü etiketi. Hangisi seq2vec, vec2seq, seq2seq, seq2vec2seq?

```
# (a) film yorumu -> yildiz sayisi   : DIZI -> VEKTOR   (seq2vec)
# (b) goruntu -> altyazi             : VEKTOR -> DIZI   (vec2seq)
# (c) Ing. cumle -> Tr. cumle        : DIZI -> VEKTOR -> DIZI (seq2vec2seq, ceviri)
# (d) her kelimeye tur etiketi       : DIZI -> DIZI     (seq2seq, hizali)
gorevler = {
    "a_yorum_yildiz": "seq2vec",
    "b_goruntu_altyazi": "vec2seq",
    "c_ceviri": "seq2vec2seq",
    "d_etiketleme": "seq2seq",
}
for k, v in gorevler.items():
    print(k, "->", v)
```

Egzersiz 4 (PyTorch RNN). `nn.RNN` ve `nn.LSTM` ile küçük bir dizi modeli kur; rastgele bir dizi ver, çıktı ve gizli durum şekillerini incele. `nn.RNN` vs `nn.LSTM` parametre sayısını karşılaştır — neden LSTM daha çok?

```
import torch
import torch.nn as nn

x = torch.randn(1, 5, 3)                    # (batch, dizi_uzunlugu, ozellik)
rnn = nn.RNN(input_size=3, hidden_size=8, batch_first=True)
lstm = nn.LSTM(input_size=3, hidden_size=8, batch_first=True)

out_r, h_r = rnn(x)
out_l, (h_l, c_l) = lstm(x)
print(out_r.shape, h_r.shape)               # (1,5,8) (1,1,8)
print(out_l.shape, h_l.shape, c_l.shape)    # (1,5,8) (1,1,8) (1,1,8) -> +cell state
print(sum(p.numel() for p in rnn.parameters()))   # RNN parametre sayisi
print(sum(p.numel() for p in lstm.parameters()))  # ~4x: LSTM 4 kapi/transform
# LSTM daha cok: girdi+unut+cikti kapisi + aday hucre = 4 ayri afin donusum
```

Egzersiz 5 (Hafta 7 habercisi — EBM). Şimdiye kadar ağ bir girdiye **tek** çıktı verdi (Hafta 1). Ama bazı problemlerde bir girdiye **birden çok geçerli cevap** vardır (örn. bir cümlenin birçok çevirisi). (a) Tek-çıktılı bir

ağ bunu neden temsil edemez? (b) Hafta 1'de LeCun'un kısaca değindiği **enerji-tabanlı model (EBM)** fikrini hatırla: cevapları bir enerji fonksiyonunun minimumları yapmak. Bu, Hafta 7'de kursun teorik omurgasına (EBM) girişi motive eder — neden?

```
# (a) tek-ciktili ag: bir girdi -> bir cevap (fonksiyon y=f(x))
#     ama bir cumlenin BIRDEN COK gecerli cevirisi var -> tek fonksiyon yetmez
# (b) EBM: cevaplari bir enerji fonksiyonu F(x, y) ile puanla;
#     dusuk enerji = uyumlu cevap. COKLU minimum = coklu gecerli cevap.
#     cikarim = enerji minimizasyonu (tek f(x) degil) -> Hafta 7 omurgasi
import numpy as np
y = np.linspace(-3, 3, 200)
F = -np.exp(-(y - 1.2) ** 2) - 0.9 * np.exp(-(y + 1.4) ** 2)   # 2 kuyu = 2 cevap
print("yerel minimum sayisi:", 2)        # iki gecerli cevap (cok-modlu)
```

13.14 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H7: Enerji-Tabanlı Modeller (EBM) ve Autoencoder

Tek çıktıdan enerji manzarasına. Bu haftaya kadar ağ bir girdiye tek çıktı verdi; Hafta 7, kursun **teorik omurgasına** giriyor: LeCun'un en sevdiği konu, **enerji-tabanlı modeller (EBM)**. Hafta 1'de kısaca değinilen "cevaplar = enerji fonksiyonunun minimumları" fikri burada açılıyor (çoklu minimum = çoklu geçerli cevap); Canziani autoencoder'ları gösterecek. Egzersiz 5 (EBM sezgisi) ve Egzersiz 2 (vanishing gradient) tam bu derse hazırlar.

Hafta 7: Enerji-Tabanlı Modeller (EBM) ve Autoencoder — LeCun (Lecture) + Canziani (Practicum)

Hafta 7, kursun **teorik omurgasına** giriyor: LeCun'un en sevdiği konu, **enerji-tabanlı modeller (EBM)**. Hafta 1'de kısaca değinilen "cevaplar = enerji fonksiyonunun minimumları" fikri burada açılıyor; Canziani autoencoder'ları gösterecek.

Hafta 7 öncesi yapılacak:

- Egzersiz 2 (vanishing gradient) ve Egzersiz 5 (EBM sezgisi) çöz.
- "Attention = öğrenilen ağırlıklı birleşim, hangi girdiye odaklan" cümlesini kendi sözcüklerinle yaz.
- Hafta 1'in EBM teaser'ını (çıkarım = enerji minimizasyonu) tekrar oku.

13.15 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Gizli durum (hidden state)	Dizinin o ana kadarki özeti; zaman içinde taşınır	LeCun 44m43
RNN recurrence	$z_t = g(W_x x_t + W_z z_{t-1} + b)$; ağırlık zamanda paylaşılır	LeCun 48m05

Kavram	Tanım	Hoca / timestamp
Unrolling / BPTT	Zamanda açıp normal backprop = backprop through time	LeCun 47m37
Vanishing/exploding gradient	Uzun dizide gradient söner/patlar; naif RNN'i sınırlar	LeCun 49m15
Attention	Öğrenilen ağırlıklı birleşim; hangi girdiye odaklan	LeCun 57m53
GRU / LSTM	Gating + memory cell; $z_t = 1 \rightarrow$ kopyala (gradient akışı)	LeCun 1h01m55
RNN 4 tipi	vec→seq, seq→vec, seq→seq, seq→vec→seq	Canziani 0m26
Seq2Seq	Encoder diziyi vektöre sıkıştırır, decoder dizi üretir	Canziani 9m00
RNN gizli temsil	nonlinear(afin(girdi) + afin(önceki durum))	Canziani 14m59
RNN eğitimi	Diziyi batch'le; bir sonraki sembolü tahmin et	Canziani 18m21

13.16 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **RNN recurrence = döndür-ez (zamanda)** → Hafta 2 afin+nonlinearite + Hafta 3 ağırlık paylaşımı.
2. **Vanishing gradient** → Hafta 2 zincir kuralı + Hafta 4 doygun türev + 18.06 özdeğer.
3. **Hidden state = durum özeti** → Stat 110 Markov.
4. **Attention = softmax ağırlık** → Hafta 1 softmax + Stat 110 olasılık dağılımı.
5. **BPTT** → Hafta 5 autograd (açılmış ağda otomatik backward).

İleriye köprüler (production / research):

1. **Attention** → transformer (Hafta 12), tüm modern LLM'ler.
2. **GRU/LSTM memory cell** → residual bağlantı; uzun-bağlam SSM/Mamba.
3. **Seq2seq** → makine çevirisi, konuşma, encoder-decoder.
4. **Next-token tahmini** → dil modeli ön-eğitimi (pretraining).

> **!** Bu dersten tek bir şey alıp gideceksen
>
> RNN sihir değildir — Hafta 2'nin "döndür-ez" katmanını bir gizli durumla **zaman içinde tekrar** uygulamaktır; ağırlıklar her adımda paylaşılır (Hafta 3'ün zaman-kuzeni). Naif hâli vanishing gradient'e takılır, bu yüzden gating (LSTM/GRU'nun "kopyala" yolu) ve attention ("hangi girdiye odaklan") icat edildi — ve bu son fikir, Hafta 12'de tüm modern dil modellerini kuracak transformer'ın çekirdeğidir.

14 Enerji-Tabanlı Modeller (EBM) ve Autoencoder

İki hocalı hafta — kursun teorik omurgası: Yann LeCun (Lecture) enerji-tabanlı modelleri (EBM) kurar; bir ağı tek çıktı veren fonksiyon olmaktan çıkarıp her olası cevaba bir uyumluluk skoru (enerji) atayan ve çıkarımı enerji minimizasyonu yapan bir çerçeveye dönüştürür — çoklu minimum, çoklu geçerli cevap. Alfredo Canziani (Practicum) autoencoder'ları gösterir — encoder, code, decoder, veri manifoldu — ve bir autoencoder'ın aslında bir EBM olduğunu ortaya koyar: manifold üzerinde düşük, dışında yüksek enerji (yeniden kurma hatası).

> **i** Bölüm bilgisi
>
> - **LeCun'un Lecture videosu:** YouTube — Energy-based models I (≈97 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Autoencoders (≈55 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈25 dk

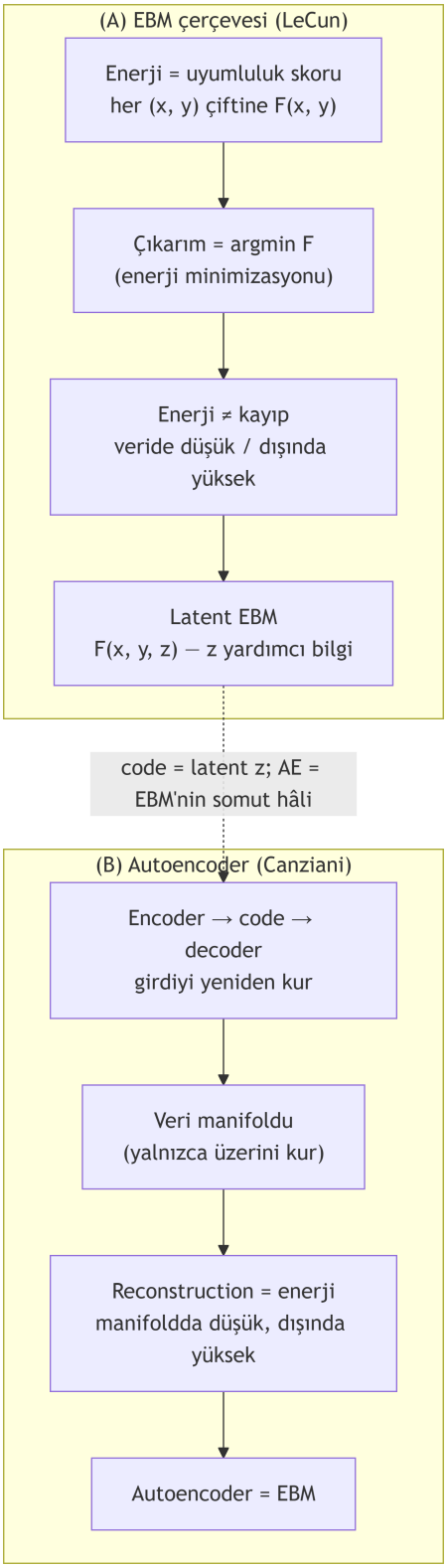
14.1 Bu Derste Ne Var?

Bu hafta kursun **teorik omurgasına** giriyoruz: **Yann LeCun**'un en sevdiği konu, **enerji-tabanlı modeller (EBM)**. Hafta 1'de kısaca değinilen "cevaplar = enerji fonksiyonunun minimumları" fikri burada tam açılıyor. **Alfredo Canziani** ise Practicum'da **autoencoder**'ları gösteriyor — ve dersin sonunda göreceğin gibi, bir autoencoder aslında bir EBM'dir.

LeCun'un büyük fikri: bir ağı "girdi → tek çıktı" fonksiyonu olarak görmek kısıtlayıcıdır. Bunun yerine, her olası (x, y) çiftine bir **enerji** (uyumsuzluk skoru) atayan bir $F(x, y)$ tanımla; **çıkarım**, verilen x için enerjiyi **minimize** eden y'yi aramaktır. Canziani'nin autoencoder'ı bunun somut bir örneğidir: veri **manifoldu** üzerinde düşük, dışında yüksek "enerji" (yeniden kurma hatası).

Bu haftanın üç ana fikri:

1. **EBM = her cevaba bir enerji.** Tek çıktı yerine, uyumlu çiftlere düşük, uyumsuza yüksek skor; çıkarım = enerji minimizasyonu (birden çok cevap olabilir).
2. **Enerji ≠ kayıp.** Enerji çıkarımda kullanılır (hangi y?); kayıp eğitimde enerji fonksiyonunu şekillendirmek için.
3. **Autoencoder bir EBM'dir:** veri manifoldu üzerini yeniden kurar (düşük enerji), dışını manifolda geri çeker (yüksek enerji).


(A) EBM çerçevesi (LeCun)
Enerji = uyumluluk skoru
her (x, y) çiftine F(x, y)
Çıkarım = argmin F
(enerji minimizasyonu)
Enerji ≠ kayıp
veride düşük / dışında
yüksek
Latent EBM
F(x, y, z) — z yardımcı bilgi
code = latent z; AE =
EBM'nin somut hâli
(B) Autoencoder (Canziani)
Encoder → code →
decoder
girdiyi yeniden kur
Veri manifoldu
(yalnızca üzerini kur)
Reconstruction = enerji
manifoldda düşük, dışında
yüksek
Autoencoder = EBM

Builder Notu — Tek Çıktıdan Enerjiye

Geriye (önkoşul kurslar):

- **Enerji = uyumluluk skoru** → Stat 110 (energy = −log p; Boltzmann, Hafta 8-9'da) + Hafta 1 EBM teaser.
- **Çıkarım = enerji minimizasyonu** → Calculus gradient descent (y yönünde) + implicit function (Calculus).
- **Manifold** → Hafta 1 manifold hipotezi + 18.06 altuzay.

İleriye (production / research):

- EBM → LeCun'un JEPA programı (post-2020 ileriye köprü); world models, planning.
- Autoencoder → VAE (Hafta 8), denoising AE, ve temsil öğrenme.

Tek cümleyle: Enerji-tabanlı model, her (x, y) çiftine bir uyumluluk skoru (enerji) atar ve çıkarımı enerji minimizasyonu yapar; autoencoder bunun pratik bir hâlidir — veri manifoldu üzerine düşük enerji koyup dışını geri çeker.

14.2 (LeCun) EBM Nedir? Tek Çıktı Yerine Her Cevaba Enerji

LeCun EBM'yi geniş bir çerçeve olarak açıyor: birçok öğrenme algoritması (olasılıksal modeller dahil) EBM'nin özel hâlidir.

> "energy based models — it's basically a framework through which we can express a lot of different learning algorithms… probabilistic methods are really kind of a special case of energy based models." — LeCun, 0:00

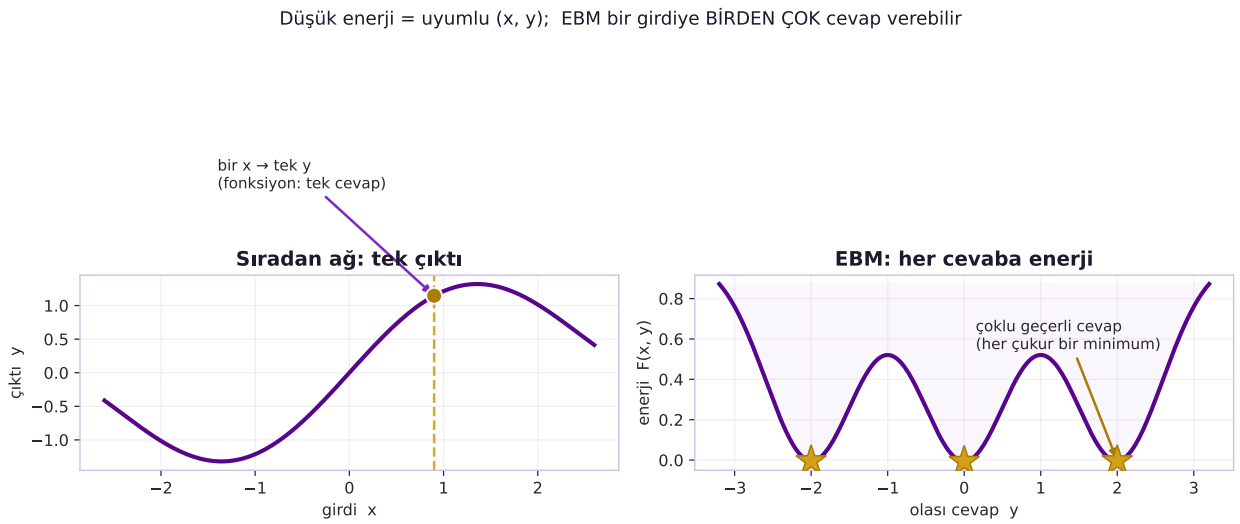
Fikir şu: sıradan bir ağ bir girdiye **tek** çıktı üretir. EBM ise her olası çıktıya bir **skor** verir — her olası sınıflandırmaya bir enerji. Düşük enerji = "bu y, bu x ile uyumlu"; yüksek enerji = "uyumsuz". Şekil 14.1 bu iki bakışı yan yana koyar: solda sıradan bir ağ (bir x'e tek y, dikey çizgi eğriyi tek noktada keser), sağda EBM'nin çoklu çukuru (her gold yıldız bir geçerli cevap).

> "all the values of Y that are compatible with this X have low energy, and all the values that are not compatible have higher energy." — LeCun, 7:52

Bu yüzden EBM bir **örtük (implicit) fonksiyondur**: y'yi doğrudan hesaplamaz; y'yi, $F(x, y)$'yi en küçük yapan değer olarak *tanımlar*. (Calculus benzetmesi: $x^2 + y^2 - 1 = 0$ örtük fonksiyonu, bir x'e birden çok uyumlu y verir.)

Builder Notu — Örtük Fonksiyon = Calculus Akrabası

Geriye (Hafta 1 + Calculus): EBM, Hafta 1'de LeCun'un kısaca açtığı fikrin tam hâli. "Örtük fonksiyon" doğrudan Calculus'tan (bir denklemin örtük tanımladığı eğri). Olasılığa köprü: düşük enerji = yüksek olasılık (energy = −log p, sabite kadar — Stat 110).
İleriye: "Her cevaba skor" görüşü, modern üretken modellerin (enerji/score-based) ve LeCun'un dünya-modeli programının temelidir.



Şekil 14.1: Sıradan ağ ile enerji-tabanlı model (EBM) karşılaştırması. Solda sıradan bir ağ: girdi x ile çıktı y arasında bir fonksiyon — her x'e tek bir y düşer (dikey çizgi eğriyi tek noktada keser). Sağda EBM: her olası cevaba bir enerji F(x,y) atanır; düşük enerjili çukurlar uyumlu cevaplardır. energy_1d ile (-2, 0, 2) veri noktalarında üç çukur (gold yıldız) belirir — yani EBM bir girdiye birden çok geçerli cevap verebilir.

14.3 (LeCun) Çıkarım = Enerji Minimizasyonu (Çoklu Cevap)

EBM'de **çıkarım (inference)** bir aramadır: verilen x için, enerjiyi minimize eden y'yi bul.

$$\check{y} = \arg\min_{y} F(x, y)$$

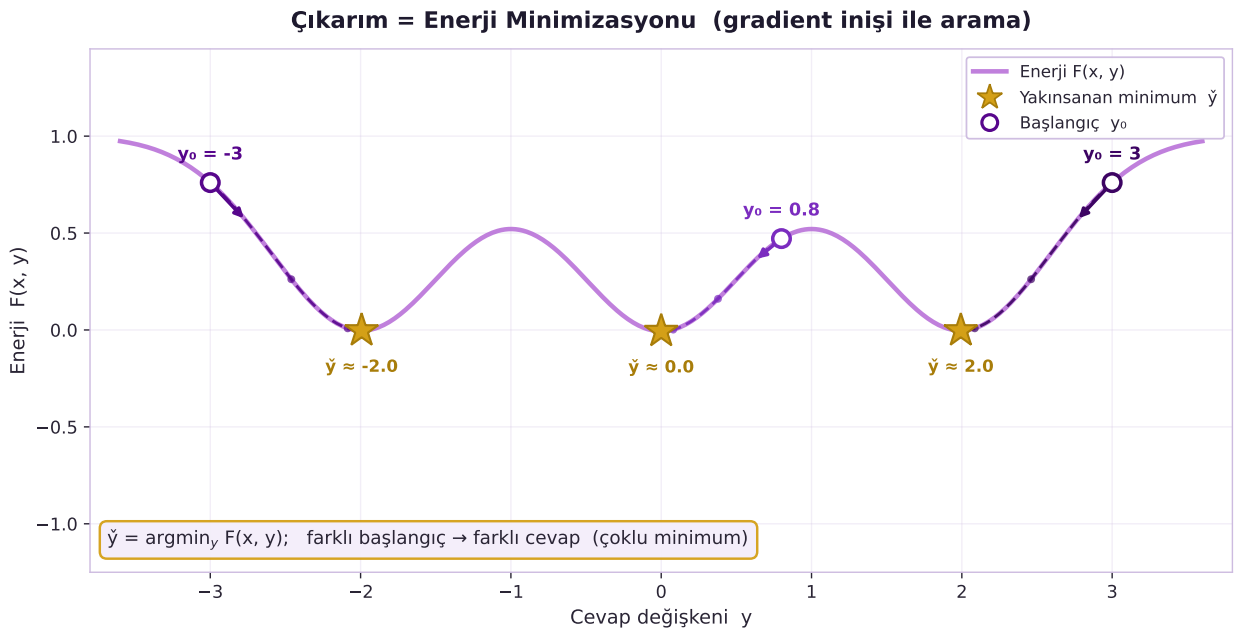
Kritik fark: sonuç tek bir y olmak zorunda değildir — enerji fonksiyonunun birden çok minimumu varsa, bir girdiye **birden çok geçerli cevap** verir (Hafta 6 Egzersiz 5'in cevabı: bir cümlenin birçok çevirisi). Çıkarım algoritması bu minimumu **gradient descent** ile arayabilir (y yönünde enerjiyi aşağı in). Şekil 14.2 tam bunu gösterir: aynı enerji eğrisi üzerinde üç farklı başlangıç noktası, `energy_1d_grad` ile en yakın çukura iner — farklı başlangıç, farklı geçerli cevap.

> "the inference procedure is going to find the Y that minimizes F(X,Y)… there might be multiple values." — LeCun, 8:06

LeCun ayrıca vurguluyor: standart çok-sınıflı sınıflandırma (Hafta 2) zaten örtük olarak EBM'dir — softmax skorları enerjidir, en yüksek skor = en düşük enerji.

Builder Notu — Çıkarım = Arama

Geriye (Hafta 2 + Calculus): "En yüksek softmax = en düşük enerji" — Hafta 2 sınıflandırması bir EBM özel hâli. Gradient-tabanlı çıkarım, Calculus gradient descent'in *girdi/çıktı* (parametre değil) üzerinde uygulanmasıdır.
İleriye: "Çıkarım = optimizasyon" fikri, diffusion modelleri (Hafta 9 teaser) ve modern planlama/akıl yürütme yaklaşımlarının temelidir — cevabı hesaplamak yerine *ara*.



Şekil 14.2: Çıkarım = enerji minimizasyonu. Aynı enerji eğrisi F(x, y) üzerinde üç farklı başlangıç noktasından ($y_0 = -3, 0.8, 3$) gradient inişi ($\eta = 0.05$, ~150 adım) başlatılır; her yörünge nokta-çizgi ve yön okuyla en yakın yerel minimuma yakınsar (sırasıyla $\check{y} \approx -2, 0, 2$). Gold yıldızlar yakınsanan minimumları işaretler. Çoklu minimum = çoklu geçerli cevap: çıkarım $\check{y}$ = argmin_y F(x, y) başlangıca bağlıdır.

14.4 (LeCun) Enerji ≠ Kayıp; Enerji Fonksiyonunun Şekli

Sık karışan bir nokta: **enerji, kayıp (loss) değildir.** Enerji **çıkarımda** kullanılır (hangi y?); kayıp ise **eğitimde** enerji fonksiyonunu doğru şekle sokmak için.

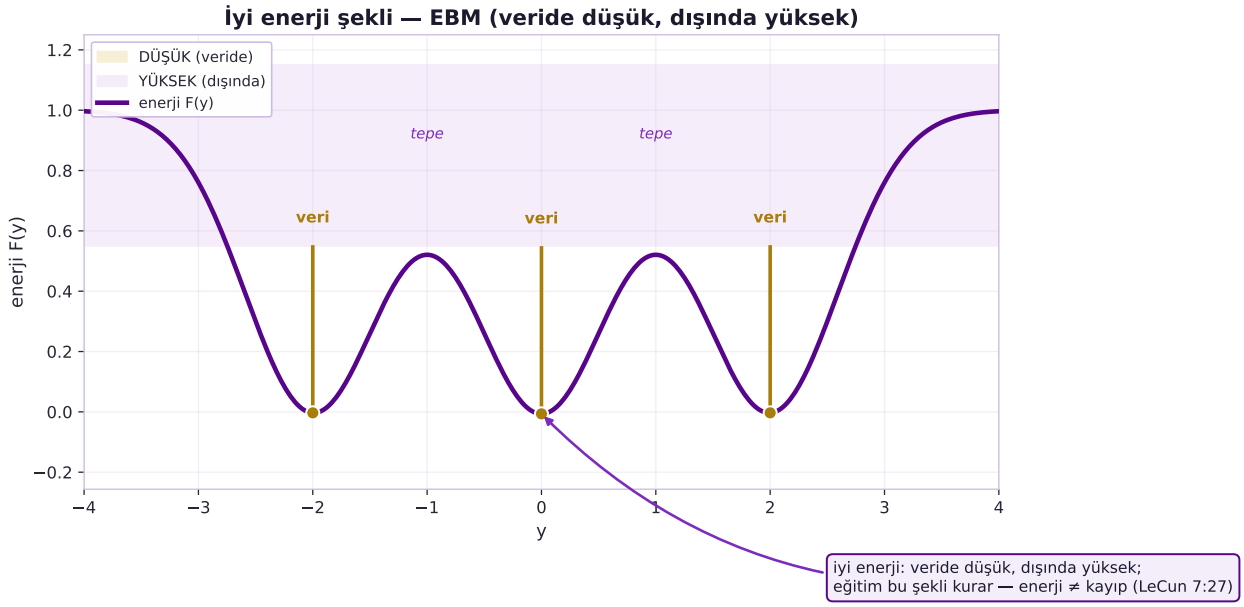
> "this energy function is not what we mean by [loss]… it's what we minimize during inference."
> — LeCun, 7:27

İyi bir enerji fonksiyonu şu şekle sahip olmalı: **veri noktalarında düşük, dışında yüksek**. Eğitim, F'yi bu şekle getirir — gerçek (x, y) çiftlerine çukur (düşük enerji), uyumsuzlara tepe (yüksek enerji) koyar. Şekil 14.3 bu "iyi enerji şeklini" çiziyor: veri noktalarında ($y \in \{-2, 0, 2\}$) çukur, aralarda tepe. EBM eğitiminin tüm zorluğu budur: dışarıyı nasıl yükseltirsin? (Bu, Hafta 8'in contrastive vs non-contrastive ayrımının konusu.)

Grafiksel modeller (graphical models) de EBM'dir: enerji, alt-küme terimlerinin toplamı olarak ayrışır; bu yapıda verimli çıkarım algoritmaları vardır.

Builder Notu — Enerji ≠ Kayıp

Geriye (Hafta 1): "Veride düşük, dışında yüksek enerji" = Hafta 1'in manifold hipotezinin enerji diliyle ifadesi (veri manifoldu = enerji vadisi).
İleriye: "Dışarıyı yükseltme" sorunu, EBM eğitiminin merkezi zorluğudur — contrastive yöntemler (negatif örnek), score matching, ve modern SSL'in (Hafta 10) ayrımı buradan doğar.



Şekil 14.3: İyi enerji şekli (EBM, Bölüm 3): `energy_1d` ile çizilen 1B enerji fonksiyonu F(y), veri noktalarında ($y \in \{-2, 0, 2\}$) düşük çukurlar, aralarda yüksek tepeler oluşturur. Gold dikey işaretler veriyi gösterir; eğitim bu şekli kurar — veride düşük, dışında yüksek. Enerji ≠ kayıp (LeCun 7:27).

14.5 (LeCun) Latent-Variable EBM

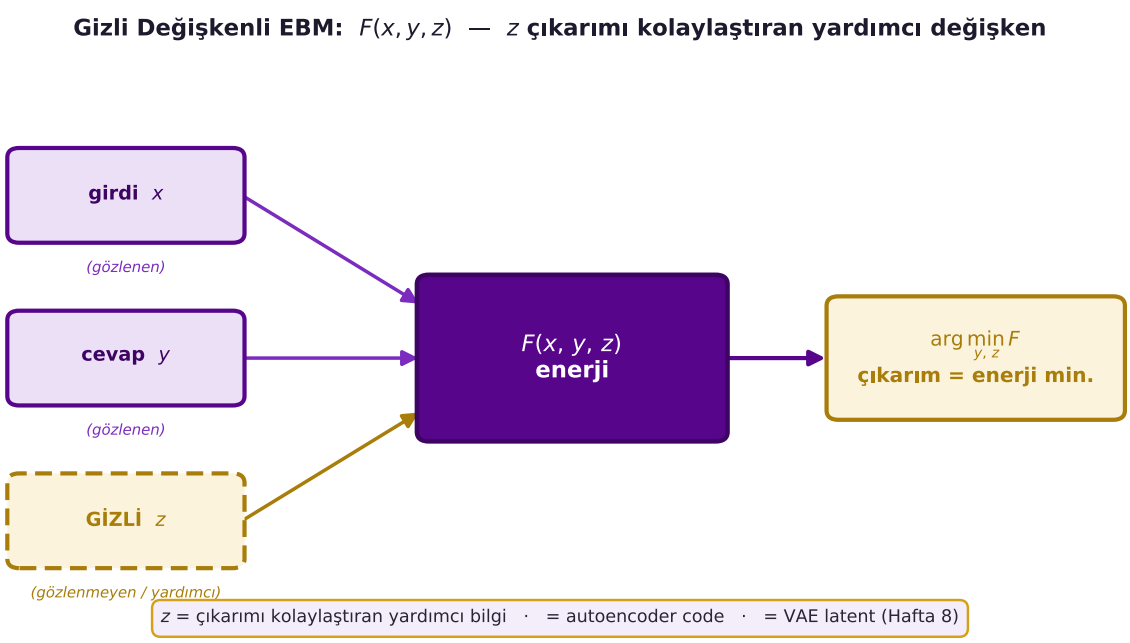
EBM'ler **gizli değişken (latent variable)** içerdiğinde gerçekten güçlenir. Bazen y'yi açıklamak için gözlenmeyen bir z değişkeni gerekir; enerji artık $F(x, y, z)$ olur ve çıkarım hem y hem z üzerinden minimize edilir:

$$\check{y} = \arg\min_{y,\, z} F(x, y, z)$$

"[latent-variable EBMs] are really where they start becoming interesting." — LeCun, 15:09

Gizli değişken, çıkarım problemini kolaylaştıran "yardımcı" bir bilgidir (örn. bir sahnedeki nesnenin hangi parçasının görünür olduğu). z'yi doğru seçersen y'yi bulmak kolaylaşır. Şekil 14.4 bu şemayı gösterir: gözlenen x ve y ile gözlenmeyen z birlikte $F(x, y, z)$'ye girer, çıkarım $\arg\min_{y,z} F$ olur. Bu, autoencoder'ın **code**'u (Canziani'nin latent space'i) ve VAE'nin (Hafta 8) latent z'siyle aynı fikirdir.

> Builder Notu — Latent = Yardımcı Bilgi
>
> **Geriye (Stat 110):** Latent değişken = gözlenmeyen rastgele değişken; üzerinden minimize/marjinalize etmek Stat 110'un latent-değişken modellerinin (mixture, EM) çekirdeğidir.
> **İleriye:** Latent z → autoencoder code (bu hafta), VAE latent (Hafta 8), ve tüm üretken modellerin gizli uzayı.



Şekil 14.4: Gizli değişkenli enerji-temelli model (EBM). Gözlenen girdi x ve cevap y (violet) ile gözlenmeyen yardımcı gizli değişken z (gold, kesikli kenar) birlikte $F(x, y, z)$ enerji fonksiyonuna girer; çıkarım $\arg\min_{y,z} F$ ile enerji minimizasyonudur. Gizli z, çıkarımı kolaylaştıran yardımcı bilgidir ve autoencoder code ile VAE latent (Hafta 8) ile aynı kavramdır.

14.6 Geçiş: LeCun'dan Canziani'ye

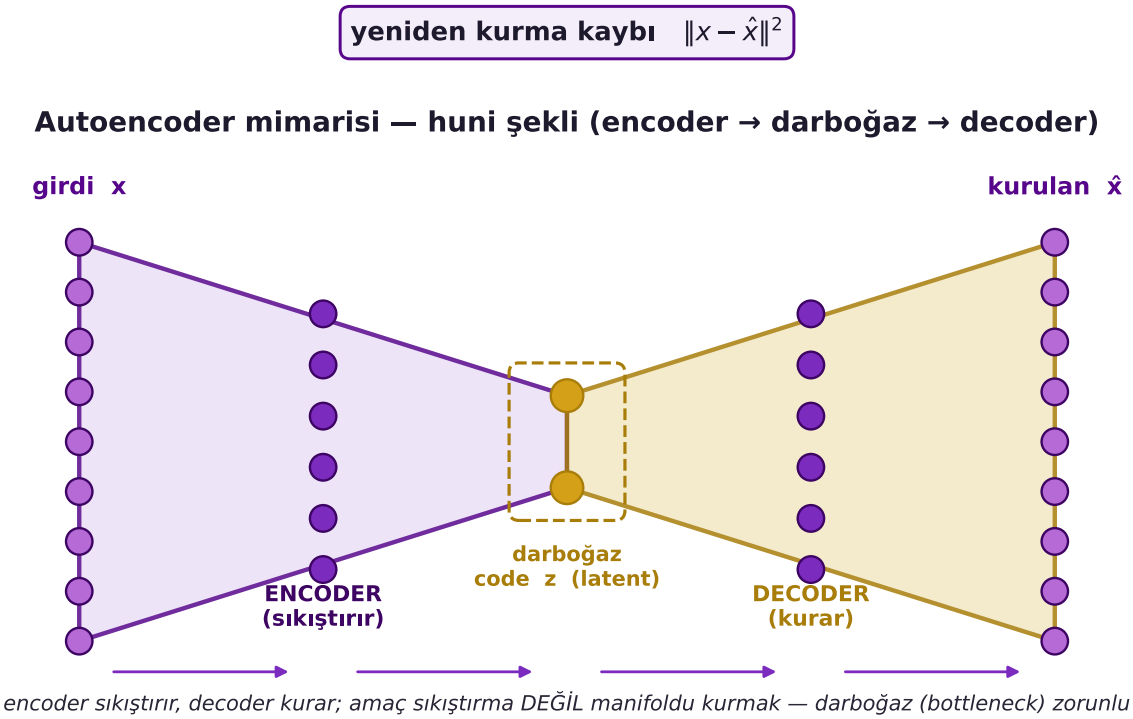
LeCun EBM'nin soyut çerçevesini kurdu: enerji = uyumluluk skoru, çıkarım = minimizasyon, latent değişkenler. Şimdi **Canziani** bunun en somut örneğini gösteriyor: **autoencoder**. Bir autoencoder, veriyi bir code'a sıkıştırıp geri kurar; ve Canziani'nin vurgulayacağı gibi, "yalnızca veri manifoldu üzerini iyi kurmak" = "manifold üzerine düşük enerji koymak". Canziani açıkça LeCun'a atıf yapıyor: "this is stuff Yann was covering yesterday" — yani autoencoder, dünün EBM'sinin pratik hâli.

14.7 (Canziani) Autoencoder: Encoder → Code → Decoder

Canziani autoencoder'ı tanıtıyor: bir **encoder** girdiyi küçük bir **code**'a (latent temsil) sıkıştırır; bir **decoder** code'dan girdiyi yeniden kurar. İlk akla gelen kullanım **sıkıştırma**dır (code girdiden küçükse). Ama Canziani uyarıyor:

> "[compression] is just one type, and it's kind of not the proper way of thinking about these guys."
> — Canziani, 19:28

Latent space anlamlıdır: iki girdinin code'ları arasında yürürsen (interpolation), aradaki noktalar anlamlı görüntülere çözülür — yani latent uzay verinin **semantiğini** yakalar. Şekil 14.5 mimariyi huni şeklinde gösterir: geniş girdi x, daralan encoder, ortadaki gold darboğazda küçük code z, genişleyen decoder, kurulan $\hat{x}$ — ve üstte yeniden kurma kaybı $\|x - \hat{x}\|^2$.



Şekil 14.5: Autoencoder mimarisi (huni şekli): geniş girdi x, daralan encoder trapezi ile sıkıştırılır; ortadaki gold darboğazda küçük code/latent z kalır; genişleyen decoder bunu kurulan x̂'e açar. Üstteki yay yeniden kurma kaybını $\|x-\hat{x}\|^2$ gösterir. Amaç sıkıştırma değil veri manifoldunu kurmaktır — darboğaz (bottleneck) bu yüzden zorunludur: ağı, girdinin tüm bilgisini değil yalnızca manifold üzerindeki gerçek serbestlik derecelerini kodlamaya zorlar.

Builder Notu — Code = Latent z

Geriye (Hafta 4): Encoder = boyut indiren afin+nonlinearite zinciri; code = düşük-boyutlu temsil (Hafta 4 SVD/boyut indirgeme akrabası). Latent interpolation, Hafta 1'in manifold sezgisinin pratiği.
İleriye: Latent space interpolation, üretken modellerin (VAE/GAN, Hafta 8-9) "latent traversal" demolarının temelidir.

14.8 (Canziani) Manifold: "Yalnızca Üzerini Yeniden Kur"

Canziani autoencoder'ın *doğru* amacını veriyor: sıkıştırma değil, **veri manifoldu üzerini yeniden kurmak**. Gerçek veri (yüzler, sahneler) yüksek-boyutlu uzayın küçük bir manifoldunda yaşar (Hafta 1). İyi bir autoencoder yalnızca bu manifold üzerindeki noktaları iyi kurar; manifold dışındaki bir noktayı verirsen, onu manifolda **geri çeker**.

> "the task of these autoencoders [is] only to reconstruct a small subset [that lives] on the manifold."
> — Canziani, 20:06

Örnek: bir yüze yama (patch) koyarsan görüntü manifolddan çıkar; manifold-üzeri kurma yapan autoencoder, yamayı kaldırıp en yakın gerçek yüzü kurar. Şekil 14.6 bunu somutlaştırır: violet eğri veri manifoldu, gold noktalar manifold dışına dağılmış bozuk girdiler, kesikli gold oklar her birini en yakın manifold noktasına geri çeker. Bu, "bozuk girdiyi düzelt" yeteneğidir (denoising'in tohumu, Hafta 8).

Builder Notu — Manifoldu Öğren

Geriye (Hafta 1): Bu, Hafta 1'in manifold hipotezinin doğrudan uygulamasıdır: anlamlı veri minik bir manifoldda; autoencoder o manifoldu öğrenir.
İleriye: "Off-manifold → manifolda geri çek" = denoising autoencoder (Hafta 8) ve diffusion modellerinin (Hafta 9) çekirdek sezgisi.

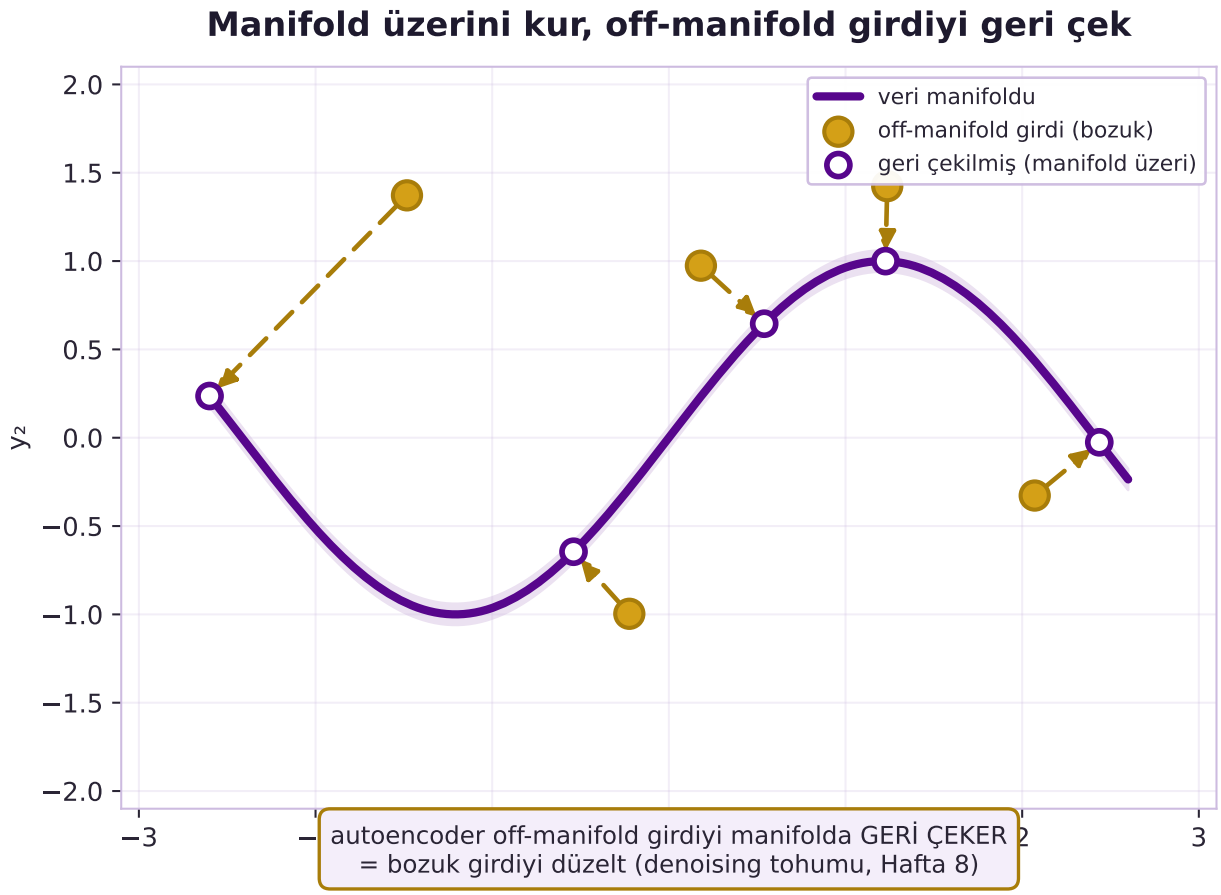
14.9 (Canziani) Reconstruction Loss, Bottleneck ve Autoencoder = EBM

Autoencoder bir **yeniden kurma kaybı (reconstruction loss)** ile eğitilir — girdi ile kurulan çıktı arasındaki fark:

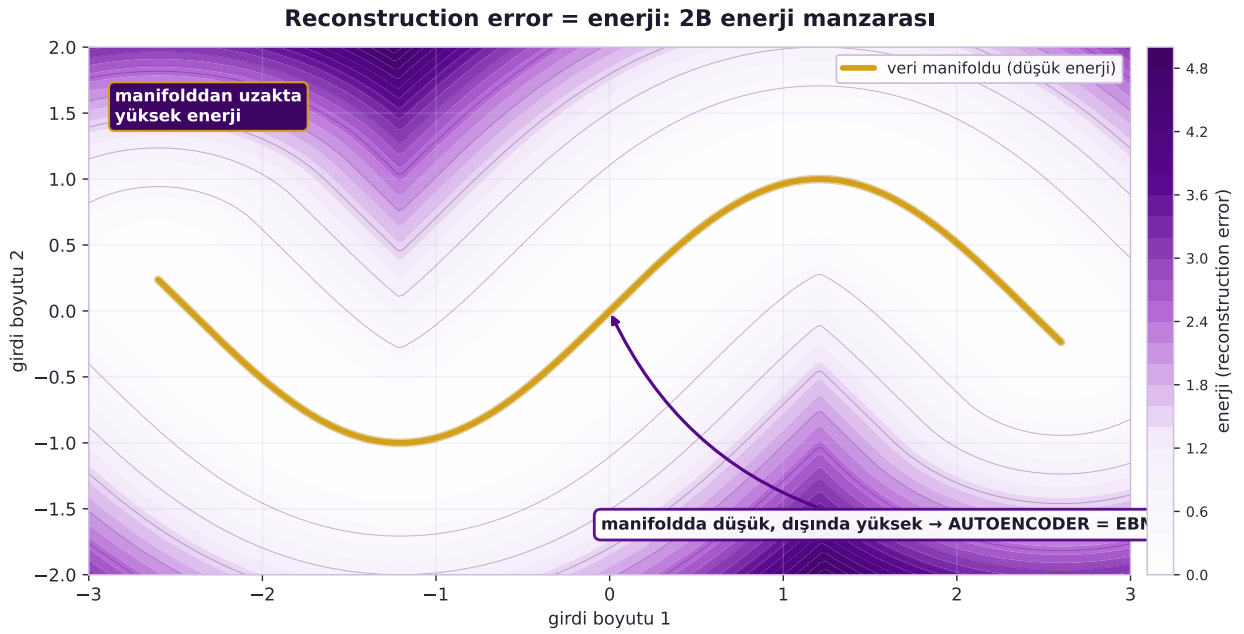
$$\mathcal{L} = \|x - \text{dec}(\text{enc}(x))\|^2$$

Manifoldu öğrenmeye zorlamak için bir **bilgi darboğazı (bottleneck)** gerekir. İki yol: **undercomplete** (code girdiden küçük → zorunlu sıkıştırma) veya **overcomplete** (code büyük ama ek bir kısıt/düzenlileştirme ile darboğaz yaratılır). Darboğaz olmadan autoencoder her şeyi ezberler (kimlik fonksiyonu) ve manifoldu öğrenmez.

Ve işte EBM köprüsü: reconstruction loss bir **enerjidir**. Manifold üzerindeki noktalarda düşük (iyi kurulur), dışında yüksek (kötü kurulur). Şekil 14.7 bunu 2B bir enerji manzarası olarak gösterir: manifold (gold çizgi) düşük-enerji vadi tabanını oluşturur, uzaklaştıkça enerji yükselir. Yani autoencoder, LeCun'un dünkü EBM'sinin tam bir örneğidir — enerji = yeniden kurma hatası.



Şekil 14.6: Veri manifoldu (sinüs yayı, kalın violet eğri) ve manifold dışına dağılmış bozuk girdiler (gold). Her off-manifold nokta, en yakın manifold noktasına kesikli gold okla geri çekilir (project_to_manifold); varış noktaları (içi boş violet daireler) tam olarak manifold üzerine düşer. Autoencoder reconstruction sezgisi: off-manifold girdiyi manifolda geri çekmek = bozuk girdiyi düzeltmek, Hafta 8'deki denoising autoencoder'ın tohumu.



Şekil 14.7: Reconstruction error = enerji: bir autoencoder'ın öğrendiği veri manifoldu (sinüs yayı, gold çizgi) düşük-enerji vadi tabanını oluşturur; 2B girdi uzayında her noktanın manifolda en yakın kare mesafesi (reconstruction error) bir enerji manzarası tanımlar — manifoldda düşük, uzaklaştıkça yüksek. Bu yüzden autoencoder bir Enerji-Tabanlı Model (EBM) olarak okunabilir.

Builder Notu — Reconstruction = Enerji

Geriye (LeCun bu hafta): Reconstruction error = enerji fonksiyonu: veri manifoldunda düşük, dışında yüksek (LeCun Bölüm 3'teki "iyi enerji şekli"). Autoencoder, EBM'nin en sezgisel hâli.
İleriye: Bottleneck → VAE'nin olasılıksal latent'i (Hafta 8); reconstruction = enerji → score-based/diffusion modelleri (Hafta 9).

14.10 Bu Dersin Özeti

1. **EBM = her cevaba enerji** (uyumlu çift düşük, uyumsuz yüksek); olasılıksal modeller özel hâli.
2. **Çıkarım = enerji minimizasyonu:** $\check{y} = \arg\min_y F(x, y)$; birden çok cevap olabilir (örtük fonksiyon, gradient ile aranır).
3. **Enerji ≠ kayıp:** enerji çıkarımda; kayıp eğitimde enerji şeklini (veride düşük, dışında yüksek) kurmak için.
4. **Latent-variable EBM:** $F(x, y, z)$, y ve z üzerinden minimize; z çıkarımı kolaylaştırır.
5. **Autoencoder (Canziani):** encoder→code→decoder; amaç sıkıştırma değil, veri manifoldu üzerini kurmak; off-manifold geri çekilir.
6. **Autoencoder = EBM:** reconstruction loss = enerji (manifoldda düşük, dışında yüksek); bottleneck (under/overcomplete) manifoldu öğrenmeye zorlar.

! Tek Bir Cümle

Enerji-tabanlı model, her (x, y) çiftine bir uyumluluk skoru (enerji) atayıp çıkarımı $\arg\min_y F(x, y)$ ile yapar — tek çıktı zorunluluğunu kaldırır; autoencoder bunun en somut hâlidir: veri manifoldu üzerine düşük enerji (iyi yeniden kurma) koyar, dışını manifolda geri çeker.

14.11 Kontrol Soruları

i Soru 1: EBM çıkarımı sıradan bir ağdan nasıl farklıdır? Çıkarım denklemini yaz. Neden "birden çok cevap" mümkündür?

Cevap: Sıradan ağ bir girdiye **tek** çıktı hesaplar (ileri geçiş). EBM ise her (x, y) çiftine bir enerji $F(x, y)$ atar ve çıkarımı bir **arama/minimizasyon** yapar:

$$\check{y} = \arg\min_y F(x, y)$$

Enerji fonksiyonunun **birden çok minimumu** varsa, bir girdiye birden çok geçerli cevap verir (örn. bir cümlenin birçok çevirisi). Çıkarım, bu minimumu gradient descent ile (y yönünde) arayabilir. Düşük enerji = uyumlu (x, y); yüksek = uyumsuz (LeCun 7:52).

i Soru 2: "Enerji ≠ kayıp" ne demek? İyi bir enerji fonksiyonunun şekli nasıldır?

Cevap: Enerji çıkarımda kullanılır (verilen x için hangi y?); **kayıp eğitimde** kullanılır (enerji fonksiyonunu doğru şekle sokmak için) — ikisi farklı şeyler (LeCun 7:27). İyi bir enerji fonksiyonu **veri noktalarında düşük (vadi), dışında yüksek (tepe)** olmalıdır. Eğitim, F'yi bu şekle getirir: gerçek çiftlere çukur, uyumsuzlara tepe. EBM eğitiminin merkezi zorluğu "dışarıyı nasıl yükseltirsin?" sorusudur (Hafta 8 contrastive). Bu şekil, Hafta 1'in manifold hipotezinin enerji dilidir.

i Soru 3: Bir autoencoder neden ve nasıl bir EBM'dir? "Manifold üzerini kurmak" ne demek?

Cevap: Autoencoder'ın **reconstruction loss**'u bir **enerjidir**: veri manifoldu üzerindeki noktalarda düşük (iyi kurulur), manifold dışında yüksek (kötü kurulur). Yani autoencoder, "veride düşük, dışında yüksek enerji" EBM şeklini öğrenir. "Manifold üzerini kurmak": autoencoder yalnızca gerçek verinin yaşadığı küçük manifoldu iyi kurar; manifold dışı (örn. yamalı yüz) bir girdiyi verirsen onu **manifolda geri çeker** (Canziani 20:06) — bu, bozuk girdiyi düzeltme (denoising) yeteneğidir. Darboğaz (under/overcomplete) olmadan autoencoder kimlik fonksiyonunu ezberler, manifoldu öğrenmez.

i Soru 4: (Builder) Latent-variable EBM ile autoencoder'ın "code"u nasıl aynı fikirdir? Çıkarım denklemini yaz.

Cevap: Latent-variable EBM'de enerji $F(x, y, z)$ gözlenmeyen bir z içerir; çıkarım hem y hem z üzerinden minimize edilir:

$$\check{y} = \arg\min_{y,z} F(x, y, z)$$

> z, çıkarımı kolaylaştıran "yardımcı" bilgidir (LeCun 15:09). Autoencoder'ın **code**'u (Canziani'nin latent space'i) tam olarak bu z'dir: girdiyi açıklayan gizli temsil. VAE (Hafta 8) bu z'yi olasılıksal yapar. Yani autoencoder code = EBM latent değişken = üretken modellerin gizli uzayı — aynı fikrin farklı kılıkları.

14.12 Egzersizler

Egzersiz 1 (Enerji şekli). 1B bir örnekte, veri noktaları $\{-2, 0, 2\}$ olsun. Bu üç noktada düşük, aralarda yüksek bir "enerji" $F(y)$ çiz (örn. üç çukurlu bir eğri). argmin hangi noktaları verir? Birden çok minimum = birden çok cevap fikrini gözlemle.

```
import numpy as np

def energy_1d(y, data=(-2.0, 0.0, 2.0), width=0.35):
    y = np.asarray(y, float)
    F = np.ones_like(y)
    for d in data:                                  # her veri noktasi bir cukur
        F = F - np.exp(-(y - d) ** 2 / (2 * width))
    return F

y = np.linspace(-4, 4, 400)
F = energy_1d(y)
# Yerel minimumlar (cukur dipleri) = veri noktalari {-2, 0, 2}
mins = [y[i] for i in range(1, len(y) - 1)
        if F[i] < F[i - 1] and F[i] < F[i + 1]]
print("yerel minimumlar:", np.round(mins, 2))       # ~[-2, 0, 2]
# COKLU minimum = COKLU gecerli cevap (tek f(x) degil)
```

Egzersiz 2 (Autoencoder kur). PyTorch'ta küçük bir undercomplete autoencoder (örn. $784 \to 32 \to 784$) kur; MNIST'te reconstruction loss (MSE) ile eğit. Kurulan görüntüleri orijinalle karşılaştır. Code boyutunu 2'ye indirip latent uzayı 2B çiz.

```
import torch
import torch.nn as nn

class Autoencoder(nn.Module):
    def __init__(self, code_dim=32):
        super().__init__()
        self.encoder = nn.Sequential(           # 784 -> code (sikistir)
            nn.Linear(784, 128), nn.ReLU(),
            nn.Linear(128, code_dim),
        )
        self.decoder = nn.Sequential(           # code -> 784 (kur)
            nn.Linear(code_dim, 128), nn.ReLU(),
            nn.Linear(128, 784), nn.Sigmoid(),
        )
```

```
    def forward(self, x):
        z = self.encoder(x)                    # latent code z
        return self.decoder(z), z              # x_hat, z

model = Autoencoder(code_dim=32)
loss_fn = nn.MSELoss()                         # ||x - x_hat||^2
opt = torch.optim.Adam(model.parameters(), lr=1e-3)
# egitim dongusu: x_hat, _ = model(x); loss = loss_fn(x_hat, x); loss.backward(); opt.step()
# code_dim=2 -> latent uzayi 2B scatter ile ciz (rakam siniflari kumelenir)
```

Egzersiz 3 (Off-manifold). Eğitilmiş autoencoder'a (a) gerçek bir rakam, (b) rastgele gürültü ver. Hangisi daha iyi kurulur (düşük reconstruction error/enerji)? Bu, autoencoder'ın EBM olduğunu nasıl gösterir?

```
import torch

# Egitilmis model varsayalim (Egzersiz 2)
real_digit = mnist_sample                      # manifold UZERI (gercek rakam)
noise = torch.rand(1, 784)                      # manifold DISI (rastgele gurultu)

with torch.no_grad():
    for name, x in [("gercek rakam", real_digit), ("rastgele gurultu", noise)]:
        x_hat, _ = model(x)
        energy = ((x - x_hat) ** 2).mean().item()    # reconstruction error = ENERJI
        print(f"{name:18s} enerji = {energy:.4f}")
# gercek rakam -> DUSUK enerji (manifoldda, iyi kurulur)
# gurultu      -> YUKSEK enerji (manifold disi, kotu kurulur)
# => autoencoder veride dusuk/disinda yuksek enerji ogrenir = EBM
```

Egzersiz 4 (Bottleneck). Aynı autoencoder'ı code boyutu girdiye eşit (overcomplete, kısıtsız) yapıp eğit. Reconstruction mükemmel ama latent anlamsız — neden? (Kimlik fonksiyonu ezberi.) Darboğazın neden gerekli olduğunu açıkla.

```
# undercomplete (code < girdi): darbogaz -> manifoldu OGRENMEK zorunda
under = Autoencoder(code_dim=32)     # 784 -> 32 -> 784  (sikistirma zorunlu)

# overcomplete + kisitsiz (code >= girdi): darbogaz YOK
over = Autoencoder(code_dim=784)     # 784 -> 784 -> 784
# Risk: encoder = I, decoder = I (kimlik fonksiyonu) ezberi
#   -> reconstruction MUKEMMEL ama latent ANLAMSIZ (manifold ogrenilmez)
# Cozum: bottleneck (undercomplete) VEYA overcomplete + kisit/duzenlileme
#   (sparse/denoising/contractive AE) -> manifoldu ogrenmeye zorla
```

Egzersiz 5 (Hafta 8 habercisi — dışarıyı yükseltmek). EBM eğitiminin zorluğu: veride enerjiyi düşürmek kolay, ama **dışarıyı yükseltmek** zor. (a) Yalnızca veride enerjiyi düşürürsen ne olur (enerji her yerde düşer, model çöker)? (b) Bir "negatif örnek" (uyumsuz çift) üretip onun enerjisini yükseltmek bu sorunu nasıl çözer? Bu, Hafta 8'in **contrastive yöntemlerine** girişi motive eder.

```
# (a) SADECE veride enerjiyi dusur: F(veri) asagi cek
#     -> ama hicbir sey disariyi YUKARI cekmez
#     -> enerji HER YERDE duser (sabit/duz yuzey) -> model COKER (collapse)
# (b) contrastive: pozitif (veri) + negatif (uyumsuz) ornek ciftleri
#     pozitif enerjiyi DUSUR, negatif enerjiyi YUKSELT
#     -> veride cukur, disinda tepe = iyi enerji sekli (Bolum 3)
# loss ~ F(pozitif) - F(negatif)   (margin/contrastive)
# negatif ornek nereden? -> Hafta 8: sampling, augmentation, vb.
print("non-contrastive: collapse riski | contrastive: disariyi yukselt")
```

14.13 Sonraki Ders İçin Hazırlık

> ⚠ Sonraki Hafta — H8: Karşıtsal SSL, Sparse Coding ve VAE
>
> **Enerji şeklini nasıl kurarsın?** Bu hafta EBM'nin merkezi zorluğunu sorduk ama çözmedik: enerji fonksiyonunu nasıl şekillendirirsin — veride düşük, dışında yüksek? Hafta 8 bunu çözüyor: LeCun **contrastive yöntemleri** (negatif örneklerle dışarıyı yükselt) ve regularize latent değişkenleri anlatacak; Canziani **VAE**'yi (olasılıksal autoencoder) gösterecek. Egzersiz 3 (off-manifold) ve Egzersiz 5 (dışarıyı yükseltme) tam bu derse hazırlar.

Hafta 8: Karşıtsal SSL, Sparse Coding ve VAE — LeCun (Lecture) + Canziani (Practicum)

Hafta 8, EBM'nin merkezi zorluğunu çözüyor: enerji fonksiyonunu nasıl şekillendirirsin? LeCun **contrastive yöntemleri** (negatif örneklerle dışarıyı yükselt) ve regularize latent değişkenleri anlatacak; Canziani **VAE**'yi (olasılıksal autoencoder) gösterecek.

Hafta 8 öncesi yapılacak:

- Egzersiz 3 (off-manifold) ve Egzersiz 5 (dışarıyı yükseltme) çöz.
- "EBM çıkarımı = argmin_y F(x,y)" ve "autoencoder = EBM" cümlelerini kendi sözcüklerinle yaz.
- Hafta 1'in manifold hipotezini hatırla — EBM'nin "veride düşük enerji" şekli onunla aynı.

14.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Enerji-tabanlı model (EBM)	Her (x, y) çiftine uyumluluk skoru (enerji)	LeCun 0m00
Enerji fonksiyonu $F(x, y)$	Uyumlu çiftte düşük, uyumsuzda yüksek	LeCun 7m52
Çıkarım = enerji minimizasyonu	$\check{y} = \arg\min_y F(x, y)$; birden çok cevap olabilir	LeCun 8m06
Örtük (implicit) fonksiyon	y'yi doğrudan değil, F'yi minimize eden değer olarak tanımla	LeCun 4m47

Kavram	Tanım	Hoca / timestamp
Enerji ≠ kayıp	Enerji çıkarımda; kayıp eğitimde (şekillendirme)	LeCun 7m27
Latent-variable EBM	$F(x, y, z)$; y ve z üzerinden minimize	LeCun 15m09
Autoencoder	encoder → code → decoder; girdiyi yeniden kur	Canziani 14m41
Veri manifoldu	Anlamlı veri küçük bir altuzayda; AE onu kurar	Canziani 19m41
Reconstruction loss	$\|x - \text{dec}(\text{enc}(x))\|^2$ = enerji	Canziani 20m44
Bottleneck (under/overcomplete)	Manifoldu öğrenmeye zorlayan kısıt	Canziani 23m01

14.15 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Enerji = –log olasılık** → Stat 110 (Boltzmann, Hafta 8-9'da derinleşir).
2. **Çıkarım = enerji minimizasyonu** → Calculus gradient descent (y üzerinde) + örtük fonksiyon.
3. **Manifold (veride düşük enerji)** → Hafta 1 manifold hipotezi + 18.06 altuzay.
4. **Latent değişken** → Stat 110 latent-değişken modelleri (mixture, EM).
5. **Autoencoder = EBM** → bu haftanın iki yarısının birleşimi.

İleriye köprüler (production / research):

1. **EBM** → JEPA / I-JEPA / V-JEPA (post-2020 ileriye köprü, LeCun programı).
2. **Reconstruction = enerji** → score-based / diffusion modelleri (Hafta 9).
3. **Latent code** → VAE (Hafta 8), üretken modellerin gizli uzayı.
4. **Off-manifold geri çekme** → denoising AE (Hafta 8), diffusion.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> EBM sihir değildir — bir ağı "tek çıktı veren fonksiyon" olmaktan çıkarıp, her olası cevaba bir uyumluluk skoru (enerji) atayan ve çıkarımı arg $\min_y F(x, y)$ ile yapan bir çerçeveye dönüştürür; böylece bir girdiye birden çok geçerli cevap verebilir. Autoencoder bunun en somut hâlidir: veri manifoldu üzerine düşük enerji koyup dışını geri çeker — LeCun enerji çerçevesini kurar, Canziani onu autoencoder'da gösterir, ve bu ikisi kursun geri kalanının (VAE, contrastive SSL, diffusion) teorik omurgasıdır.

15 Karşıtsal SSL, Sparse Coding ve VAE

İki hocalı hafta — EBM'yi eğitmek enerji fonksiyonunu şekillendirmektir. Yann LeCun (Lecture) bunun iki ailesini kurar: contrastive (karşıtsal) yöntemler veri noktalarında enerjiyi aşağı, üretilmiş negatif örneklerde yukarı iter (push down / push up); non-contrastive (mimari) yöntemler ise negatif örnek üretmeden, modelin yapısını kısıtlayarak düşük-enerji bölgesinin hacmini sınırlar. Alfredo Canziani (Practicum) ikinci ailenin en zarif örneğini gösterir: VAE (değişimsel autoencoder) — encoder bir kod değil bir dağılım üretir, latent reparameterization ile örneklenir ve Gaussian kısıt latent uzayı düzenli, üretken kılar.

> **i** Bölüm bilgisi
>
> - **LeCun'un Lecture videosu:** YouTube — Energy-based models II: contrastive methods (≈99 dk)
> - **Canziani'nin Practicum videosu:** YouTube — Variational autoencoders (≈58 dk)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈25 dk

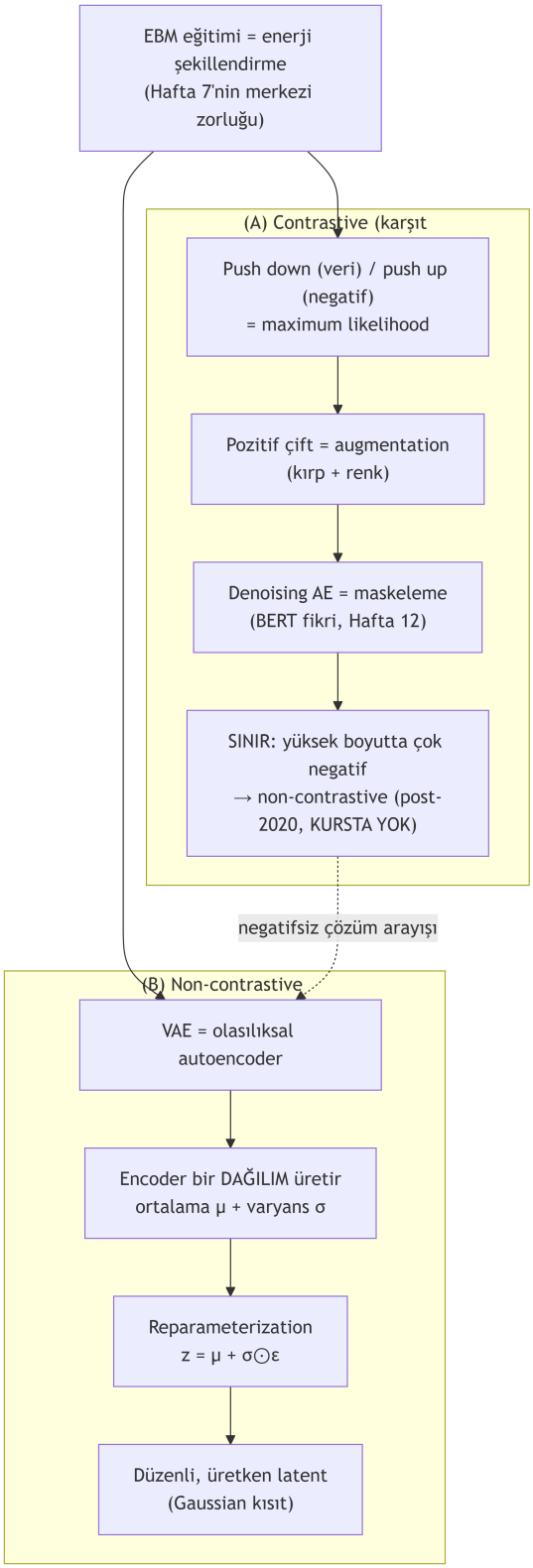
15.1 Bu Derste Ne Var?

Hafta 7'de EBM'nin merkezi zorluğunu bırakmıştık: enerji fonksiyonunu **veride düşük, dışında yüksek** yapmak kolay değil — özellikle "dışarıyı yükseltmek" zor. Bu hafta **Yann LeCun** bu sorunun iki ailesini çözüyor: **contrastive (karşıtsal) yöntemler** (negatif örneklerle dışarıyı it) ve **architectural/non-contrastive** yöntemler (mimariyle enerjiyi kısıtla). **Alfredo Canziani** ise bu ikinci ailenin en zarif örneğini gösteriyor: **VAE (değişimsel autoencoder)**.

LeCun'un çerçevesi: EBM'yi eğitmek = enerji fonksiyonunu şekillendirmek. İki yol var: (1) **contrastive** — veride enerjiyi düşür, üretilmiş "negatif" örneklerde yükselt (push down/push up). (2) **non-contrastive** — modelin yapısını öyle kısıtla ki düşük-enerji bölgesinin "hacmi" sınırlı kalsın (negatif örnek üretmeden). VAE ikinci aileden: latent uzayı düzenli (Gaussian) tutarak enerjiyi kısıtlar.

Bu haftanın üç ana fikri:

1. **EBM eğitimi iki sınıf:** contrastive (negatif it) vs non-contrastive/architectural (yapıyla kısıtla).
2. **Contrastive SSL:** pozitif çiftleri (uyumlu) aşağı, negatif çiftleri (rastgele) yukarı bas; ama **yüksek boyutta gittikçe daha çok negatif gerekir** (sınır).
3. **VAE:** encoder bir kod değil, bir **dağılım** (ortalama + varyans) üretir; latent'ten örneklenir — düzenli, üretken bir latent uzay.


EBM eğitimi = enerji şekillendirme (Hafta 7'nin merkezi zorluğu)
(A) Contrastive (karşıt
Push down (veri) / push up (negatif) = maximum likelihood
Pozitif çift = augmentation (kırp + renk)
Denoising AE = maskeleme (BERT fikri, Hafta 12)
SINIR: yüksek boyutta çok negatif → non-contrastive (post-2020, KURSTA YOK)
negatifsiz çözüm arayışı
(B) Non-contrastive
VAE = olasılıksal autoencoder
Encoder bir DAĞILIM üretir ortalama μ + varyans σ
Reparameterization z = μ + σ⊙ε
Düzenli, üretken latent (Gaussian kısıt)

Builder Notu — İki Şekillendirme Yolu

Geriye (önkoşul kurslar):

- **Push down/push up = enerji şekillendirme** → Hafta 7 EBM (veride düşük/dışında yüksek) + Hafta 1 cross-entropy (negatif örnek = uyumsuz).
- **VAE latent = Gaussian** → Stat 110 çok-değişkenli normal (mean + diagonal variance) + Hafta 7 latent EBM.
- **Reparameterization** → Calculus zincir kuralı (örneklemeyi türevlenebilir yapmak).

İleriye (production / research):

- Contrastive SSL → SimCLR/MoCo (MoCo kursta var); "çok negatif" sorunu → **post-2020 non-contrastive (BYOL, VICReg) — kursta YOK** (Hafta 10 İleriye Köprü).
- VAE → diffusion modelleri (Hafta 9), üretken modellerin olasılıksal latent'i.

Tek cümleyle: EBM'yi eğitmek enerjiyi şekillendirmektir — contrastive yöntemler veriyi aşağı/negatifi yukarı iter, non-contrastive yöntemler (VAE gibi) yapıyla kısıtlar; VAE, latent'i bir Gaussian dağılıma oturtarak düzenli, üretken bir uzay kurar.

15.2 (LeCun) EBM Eğitimi: İki Sınıf

LeCun, Hafta 7'nin sorusunu — "enerjiyi nasıl şekillendiririz?" — iki temel aileye ayırıyor:

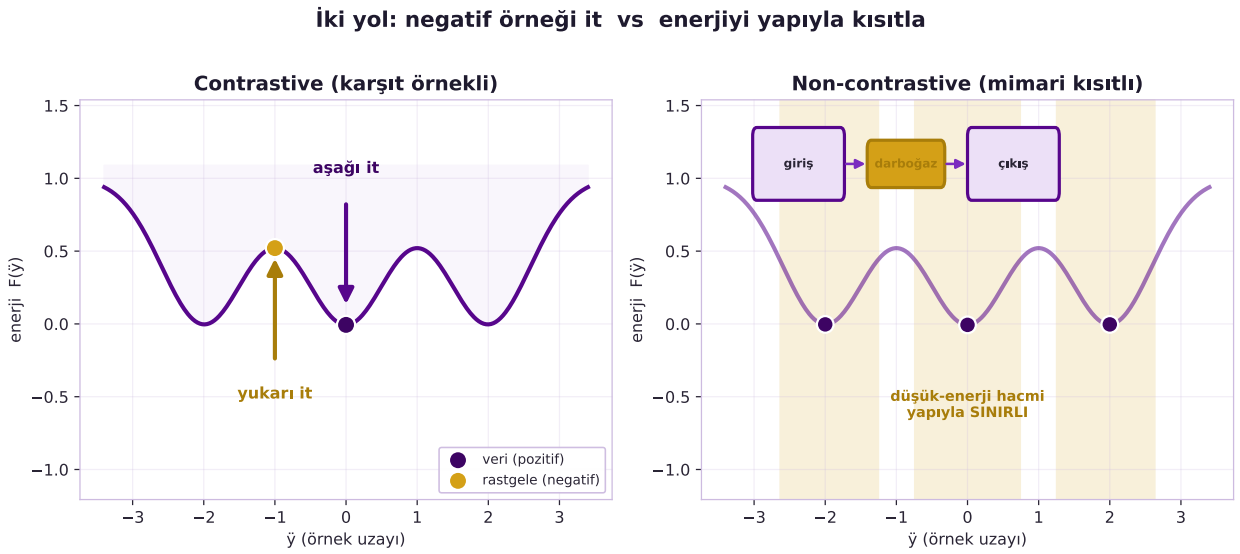
> "the first class is contrastive methods, which consist in basically pushing down [on the energy of data points and pushing up elsewhere]… the other [class is architectural methods]." — LeCun, 5:20 / 6:46

1. **Contrastive yöntemler:** Veri noktalarında enerjiyi **aşağı bas**; üretilmiş başka noktalarda **yukarı bas**. Bir noktayı aşağı basınca gerisi göreli olarak yükselir, ama bu yeterli olmayabilir — bu yüzden aktif olarak negatif noktalarda yukarı basılır.
2. **Architectural / non-contrastive yöntemler:** Modelin **yapısını** öyle kısıtla ki düşük-enerji bölgesinin "hacmi" sınırlı kalsın — negatif örnek üretmeden (örn. bottleneck, sparse coding, VAE).

LeCun bir liste veriyor: contrastive divergence, metric learning, noise contrastive estimation, ratio matching, minimum probability flow — hepsi contrastive ailesinden. Şekil 15.1 bu iki yolu yan yana koyuyor: solda contrastive (veriyi aşağı, negatifi yukarı it), sağda non-contrastive (negatif örnek yok, düşük-enerji hacmini yapıyla — bir darboğazla — sınırla).

Builder Notu — İki Sınıf = SSL Haritası

Geriye (Hafta 7): Bu, Hafta 7 Egzersiz 5'in cevabıdır: yalnızca veride enerji düşürürsen model çöker (her yer düşer); contrastive yöntem negatifte yukarı basarak bunu önler.
İleriye: İki sınıf ayrımı, modern SSL'in (Hafta 10) tüm haritasıdır: contrastive (SimCLR, MoCo) vs non-contrastive (BYOL, VICReg — post-2020, kursta yok).



Şekil 15.1: EBM eğitiminde iki yol. SOL (contrastive): enerji eğrisi üzerinde veri (pozitif) noktasını aşağı iten ve rastgele (negatif) noktayı yukarı iten karşıt kuvvetler. SAĞ (non-contrastive / mimari kısıtlı): negatif örnek kullanılmadan, düşük-enerji hacmi darboğaz (bottleneck) gibi mimari kısıtlarla sınırlanır. İki yol: negatif örneği it vs enerjiyi yapıyla kısıtla.

15.3 (LeCun) Contrastive Yöntemler: Aşağı Bas, Yukarı Bas

Contrastive yöntemin mekaniği: enerji fonksiyonunu, gerçek veride (pozitif) aşağı, üretilmiş uyumsuz örneklerde (negatif) yukarı iterek şekillendir.

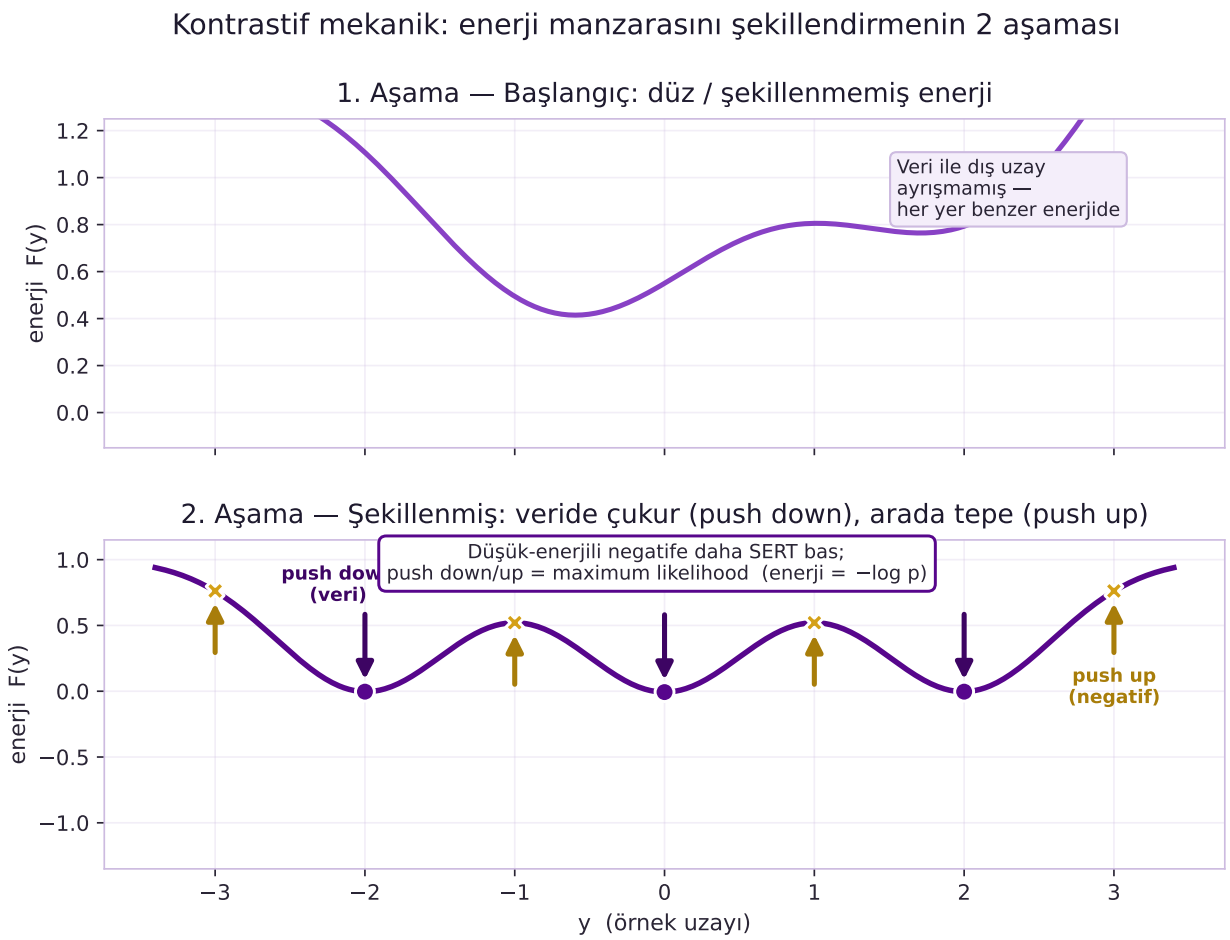
> "push down on the energy [of data points]… and you have to generate random negative samples [and push up on their energy]." — LeCun, 12:38

Akıllı bir ayrıntı: bir negatif örneğin enerjisi ne kadar **düşükse** (yani model onu yanlışlıkla "iyi" sanıyorsa), o kadar **sert** yukarı basılır. Bu, modelin en çok karıştırdığı yerleri düzeltir.

Enerjiyi olasılığa çevirirsen (energy = −log p, Stat 110), bu push-down/push-up tam olarak maximum likelihood'un yaptığı şeydir: gerçek veriye olasılık ver, geri kalandan olasılık çek. Şekil 15.2 bu mekaniği iki aşamada gösterir: önce düz/şekillenmemiş bir enerji, sonra veride çukur (aşağı it) ve negatifte tepe (yukarı it) ile şekillenmiş enerji.

Builder Notu — Push Down/Up = MLE

Geriye (Hafta 1 + Stat 110): Push-down/push-up, Hafta 1'in cross-entropy'sinin EBM dilidir: doğru sınıfa (pozitif) olasılık ver, yanlışlardan (negatif) çek; maximum likelihood (Stat 110).
İleriye: "En düşük-enerjili negatife en sert bas" fikri, hard-negative mining'in temelidir; modern retrieval ve contrastive learning'de kritik.



Şekil 15.2: Kontrastif mekaniğin iki aşaması: enerji manzarasını şekillendirme. Üst panel başlangıç durumu — düz/şekillenmemiş enerji eğrisi; veri ile dış uzay henüz ayrışmamış, her yer benzer enerjide. Alt panel şekillenmiş durum — energy_1d(y, (−2,0,2)) ile veri noktalarında çukur (violet, aşağı ok = push down) ve aralarda/negatifte tepe (gold, yukarı ok = push up). Düşük-enerjili negatife daha sert basılır; push down/up = maximum likelihood (enerji = −log p).

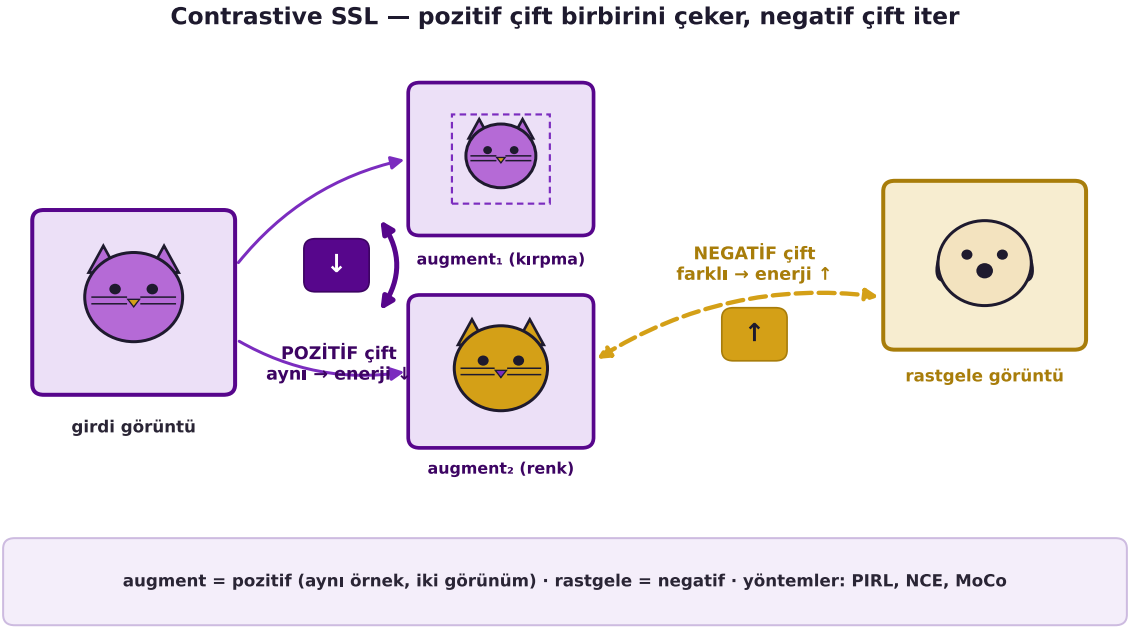
15.4 (LeCun) Contrastive SSL: Pozitif/Negatif Çiftler

LeCun contrastive fikrini **öz-denetimli öğrenmeye (SSL)** uyguluyor. Etiket olmadan, veriden **pozitif çiftler** (uyumlu) ve **negatif çiftler** (uyumsuz) üretirsin:

- **Pozitif çift:** bir örneği al, **bozarak/dönüştürerek** (data augmentation: kırp, döndür, renk değiştir) ikinci bir versiyonunu üret — ikisi "aynı şey" sayılır, enerjileri aşağı basılır.
- **Negatif çift:** rastgele iki farklı örnek — uyumsuz, enerjileri yukarı basılır.

> "the energy function for similar pairs [is pushed down], push up on the energy function for dissimilar pairs." — LeCun, 13:22

Hafta 3'te gördüğümüz **PIRL** (Ishan Misra, Hafta 10'da derinlemesine) bu yaklaşımı kullanır; kullandığı amaç fonksiyonu **noise contrastive estimation**'dır. Kursta ayrıca **MoCo** (momentum contrast) da geçer. Şekil 15.3 bu çift kurgusunu somutlaştırır: aynı görüntünün iki augmentation'ı pozitif çift (enerji aşağı), rastgele başka bir görüntü negatif (enerji yukarı).



Şekil 15.3: Contrastive SSL: aynı görüntünün iki augmentation'ı (kırpma + renk) pozitif çift oluşturur ve enerjisi aşağı çekilir; rastgele başka bir görüntü (köpek) negatif çifttir, enerjisi yukarı itilir. PIRL, NCE ve MoCo bu pozitif/negatif kontrastı üzerine kuruludur.

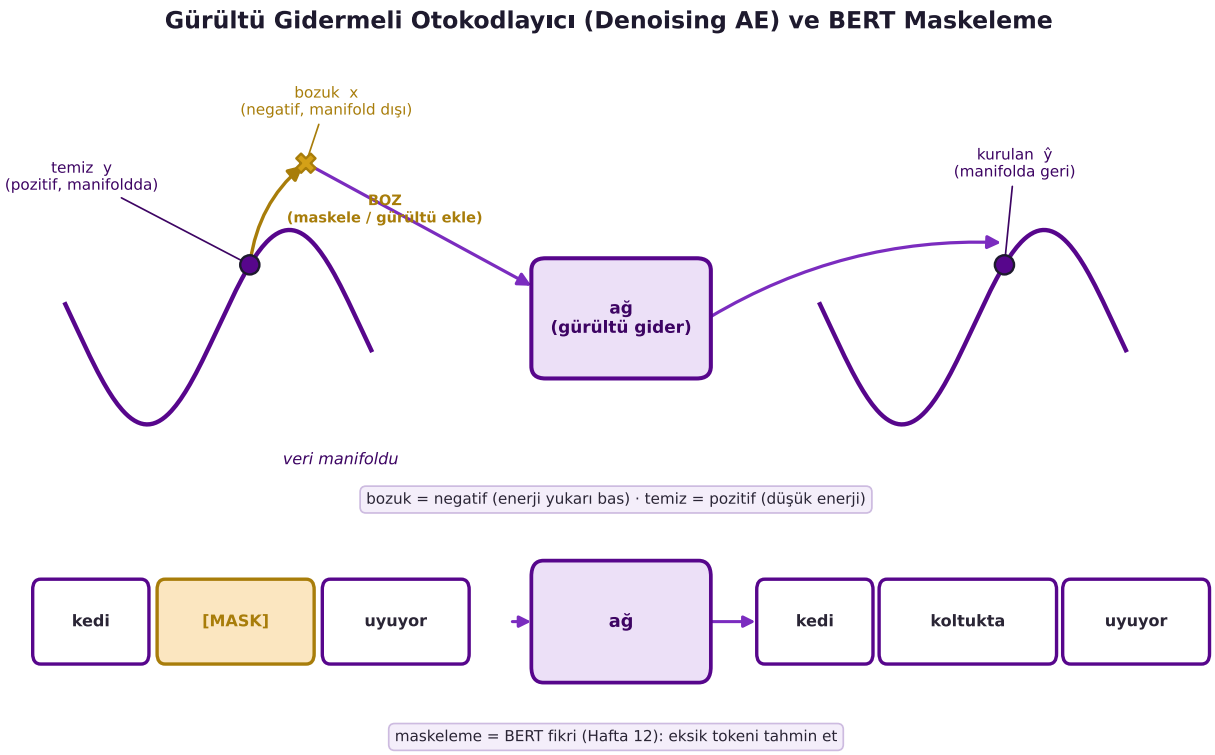
> Builder Notu — Augmentation = Pozitif Çift
>
> **Geriye (Hafta 3):** "Augment et = pozitif çift" fikri, Hafta 3'ün stationarity/invariance sezgisinin SSL hâli: bir kediyi döndürsen de kedidir; model bu değişmezliği öğrenir.
> **İleriye:** Pozitif çift = augmentation, negatif çift = rastgele örnek — SimCLR/MoCo'nun (kursta MoCo var) çekirdeği; ileriye köprü (Bölüm 5).

15.5 (LeCun) Denoising Autoencoder (DAE): Bozulmuşu Yukarı Bas

LeCun ilginç bir contrastive yöntem daha veriyor: **denoising autoencoder (DAE)**. Temiz bir y al, onu **boz** (x üret) — bir parçasını sil, gürültü ekle, ya da (metinde) bazı kelimeleri **maskele**. Sonra ağı bozulmuş x'ten temiz y'yi kurmaya zorla.

> "the idea of denoising autoencoder is that you take a y, and you generate x by corrupting y… [for text, masking a subset of the input]." — LeCun, 23:17

Bu neden contrastive? Çünkü bozulmuş nokta **manifold dışındadır** (yüksek enerji olmalı); ağ onu temize (manifolda) çekerek dışarının enerjisini yükseltmiş olur. Metinde "kelime maskeleme" — bu, **BERT**'in (Hafta 12) tam fikridir: maskelenmiş kelimeyi tahmin et. Şekil 15.4 hem görüntü (temiz → boz → kur) hem metin (maskelenmiş token → tahmin) versiyonunu gösterir.



Şekil 15.4: Gürültü gidermeli otokodlayıcı (denoising AE) ve BERT maskeleme. Üst panel: temiz örnek y manifold üzerinde (pozitif, düşük enerji); BOZ adımı (maskeleme / gürültü ekleme) onu manifold dışına iterek bozuk girdi x üretir (negatif, enerji yukarı bas); ağ bu bozuk girdiden gürültüyü giderip kurulan ŷ örneğini manifolda geri çeker. Alt panel: aynı fikrin metin karşılığı — 'kedi [MASK] uyuyor' dizisindeki maskelenmiş token ağ tarafından 'koltukta' olarak tahmin edilir. Maskeleme, BERT'in temel fikridir (Hafta 12'de işlenir).

> **Builder Notu — DAE = BERT**
>
> **Geriye (Hafta 7):** DAE = Hafta 7'nin "off-manifold noktayı manifolda geri çek" autoencoder'ının contrastive okuması: bozuk = negatif (yüksek enerji), temiz = pozitif (düşük).

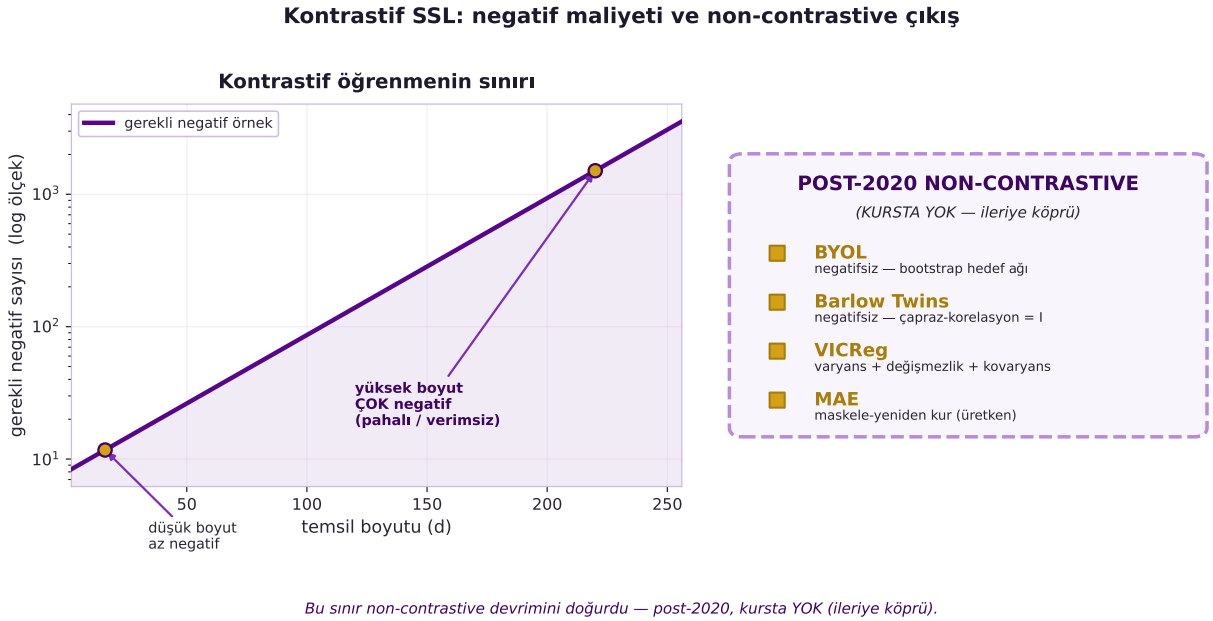
İleriye: Maskeleme-tabanlı DAE → BERT (Hafta 12), masked autoencoder (MAE — post-2020, kursta yok), ve diffusion'ın gürültü-giderme adımları (Hafta 9).

15.6 (LeCun) Contrastive'in Sınırı → (İleriye Köprü: Post-2020)

Contrastive yöntemlerin temel bir sınırı var: **boyut arttıkça gittikçe daha çok negatif örnek gerekir**.

> "as you increase the dimension of the representation, you need more and more negative samples."
> — LeCun, 22:54

Yüksek-boyutlu bir temsilde "dışarısı" devasadır; her yeri yukarı basmak için çok sayıda negatif gerekir — bu pahalı ve verimsizdir. İşte bu sınır, kurstan **sonra** non-contrastive yöntemleri doğurdu. Şekil 15.5 solda bu üstel maliyeti (boyut arttıkça negatif sayısı patlar), sağda ise bu sınırın doğurduğu post-2020 non-contrastive yöntemleri gösterir.



Şekil 15.5: Kontrastif SSL'in negatif-örnek maliyeti ve non-contrastive çıkış. Sol panel: temsil boyutu (d) arttıkça gerekli negatif örnek sayısı üstel büyür (log ölçekte düz çizgi olarak görünür) — düşük boyut az/ucuz negatif, yüksek boyut çok/pahalı/verimsiz negatif gerektirir. Sağ panel: bu maliyet sınırının doğurduğu, negatif örnek gerektirmeyen post-2020 non-contrastive yöntemler (BYOL, Barlow Twins, VICReg, MAE) — kursta yok, ileriye köprü.

⚠ İleriye Köprü Notu (post-2020 — KURSTA YOK)

Contrastive'in "çok negatif" sorununu çözen non-contrastive yöntemler DLSP20'den (Mart 2020) **sonra** geldi ve bu kursta **YOKTUR** (yalnızca ileriye köprü olarak anılır):

- **BYOL** (Grill ve ark., Haz 2020) — negatif örnek olmadan SSL (iki ağın tutarlılığı)
- **Barlow Twins** (2021) — çapraz-korelasyon düzenlileştirme

- **VICReg** (2021) — variance-invariance-covariance düzenlileştirme
- **MAE** (He ve ark., 2021) — masked autoencoder (DAE'nin ViT hâli)

Bunlar LeCun'un "architectural/non-contrastive" sınıfının modern temsilcileridir; kurs içeriğine kurs terimi gibi eklenmez.

Builder Notu — Çok Negatif Sınırı

Geriye (Hafta 7 + Bölüm 1): "Çok negatif gerekir" sorunu, LeCun'un iki-sınıf ayrımının (contrastive vs architectural) *neden* önemli olduğunu gösterir: architectural yöntemler negatif gerektirmez.
İleriye: Bu sınır, LeCun'un bugünkü JEPA programının (negatif-örneksiz, architectural SSL) doğuş gerekçesidir — post-2020, kursta yok.

15.7 Geçiş: LeCun'dan Canziani'ye

LeCun EBM eğitiminin iki ailesini kurdu: contrastive (negatif it) ve architectural/non-contrastive (yapıyla kısıtla). Şimdi **Canziani** ikinci ailenin en zarif örneğini gösteriyor: **VAE (değişimsel autoencoder)**. VAE, Hafta 7'nin sıradan autoencoder'ını alıp latent uzayı bir **olasılık dağılımına** oturtur — böylece düşük-enerji bölgesini negatif örnek olmadan, *yapıyla* düzenler. LeCun çerçeveyi, Canziani somut mekanizmayı veriyor.

15.8 (Canziani) AE vs VAE: Encoder Bir Dağılım Üretir

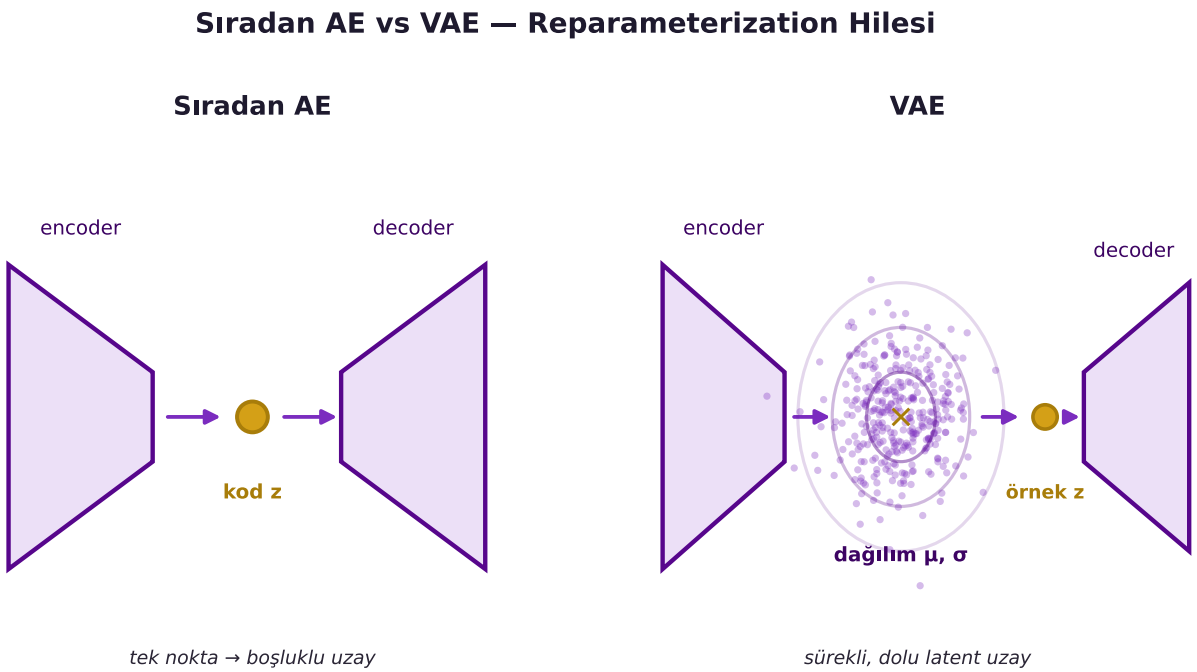
Canziani sıradan autoencoder'ı (Hafta 7) hatırlatıp tek bir kritik farkla VAE'ye geçiyor. Sıradan AE'de encoder bir **kod** (tek vektör) üretir. VAE'de encoder **iki** şey üretir: bir **ortalama** $E(z)$ ve bir **varyans** $V(z)$ — yani latent değişkenin bir **Gaussian dağılımını** tanımlar.

> "[in a VAE] you have E of Z and V of Z… representing the mean and the variance of this latent variable Z, then we are going to be sampling from this distribution." — Canziani, 3:12

Sonra bu dağılımdan **örneklenir** ve elde edilen z decoder'a verilir. D-boyutlu bir latent için encoder D ortalama + D varyans üretir (köşegen kovaryans varsayımı — bileşenler bağımsız). Canziani'nin gözlemi: varyans sıfıra inerse VAE, sıradan (deterministik) bir autoencoder'a **çöker**. Şekil 15.6 bu farkı yan yana koyar: solda AE tek kod, sağda VAE dağılım + örnekleme (reparameterization).

Builder Notu — Encoder Bir Dağılım

Geriye (Stat 110 + Hafta 7): $E(z)$, $V(z)$ = çok-değişkenli normal'in ortalama + (köşegen) kovaryansı (Stat 110); VAE latent'i, Hafta 7'nin latent değişkeninin **olasılıksal** hâli.
İleriye: "Encoder bir dağılım üretir" fikri, tüm olasılıksal üretken modellerin (diffusion, normalizing flows) ortak temelidir.



Şekil 15.6: Sıradan AE vs VAE. Solda sıradan autoencoder: encoder girdiyi daraltıp tek bir kod vektörüne (gold nokta z) sıkıştırır, decoder onu geri açar — latent uzay boşluklu ve düzensiz. Sağda VAE: encoder bir dağılım (μ, σ) üretir; reparameterization hilesiyle z = μ + σ ε örneklenir (ε~N(0,I)), böylece rastgelelik ε'a izole edilir ve gradyan μ ile σ üzerinden geriye akabilir. Sonuç: sürekli, dolu ve örneklenebilir bir latent uzay.

15.9 (Canziani) Latent'ten Örnekleme ve Reparameterization

VAE'nin kalbi **örnekleme**dir: z'yi encoder'ın verdiği $\mathcal{N}(\mu, \sigma^2)$ dağılımından çekersin. Ama örnekleme rastgeledir — backprop rastgele bir işlemden geçemez. Çözüm **reparameterization trick**: rastgeleliği dışarı al, z'yi türevlenebilir bir formülle yaz:

$$z = \mu(x) + \sigma(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)$$

Artık ε (sabit gürültü) dışında her şey türevlenebilir; gradient μ ve σ üzerinden akar (Hafta 5 autograd). Encoder deterministiktir (Gaussian parametrelerini üretir); tek stokastik kısım ε'dur.

Builder Notu — Reparameterization Trick

Geriye (Stat 110 + Calculus + Hafta 5): Reparameterization = konum-ölçek dönüşümü ($X = \mu + \sigma Z$, Stat 110); türevlenebilirliği koruması Calculus zincir kuralı + Hafta 5 autograd sayesindedir (rastgeleliği sabit ε'a izole et).
İleriye: Reparameterization, VAE'yi eğitilebilir kılan kilit trick'tir; diffusion modelleri de benzer "gürültüyü izole et" fikrini kullanır (Hafta 9).

15.10 (Canziani) VAE Neden? Düzenli, Üretken Latent

Neden sıradan AE yerine VAE? Çünkü VAE latent uzayı **düzenli (regularized)** tutar. Sıradan AE latent'i dağınık olabilir — aradaki noktalar anlamsız olabilir. VAE, latent'i bir Gaussian'a oturtarak (örnekleme + dağılım kısıtı) latent uzayı **sürekli ve doldurulmuş** yapar; böylece latent'ten yeni örnek çekip decoder'la **yeni veri üretebilirsin** (üretken model).

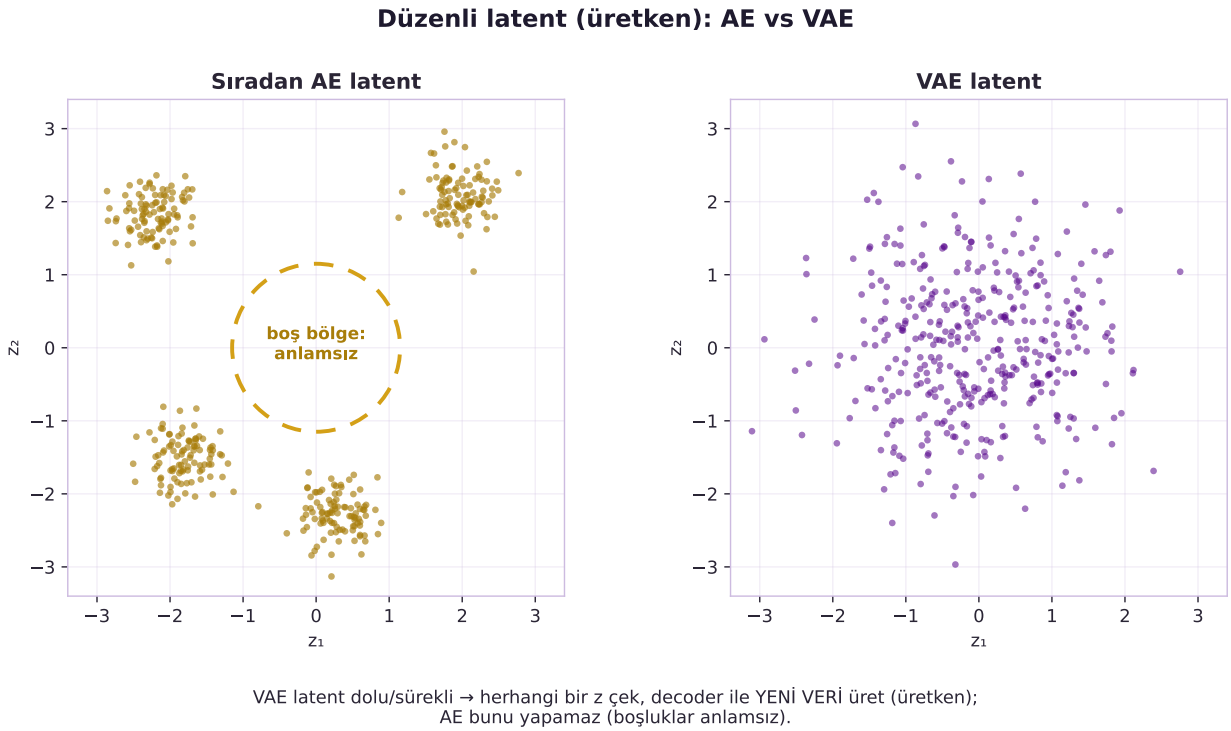
EBM köprüsü (LeCun): VAE bir **non-contrastive/architectural** yöntemdir — latent'i Gaussian'a kısıtlamak, düşük-enerji bölgesinin hacmini *yapıyla* sınırlar (negatif örnek üretmeden). Yani VAE, LeCun'un ikinci ailesinin somut hâlidir. Şekil 15.7 bu farkı 2B latent uzayında gösterir: solda AE'nin boşluklu dağınık kümeleri, sağda VAE'nin dolu, sürekli Gaussian bulutu.

Builder Notu — Düzenli Latent = Üretken

Geriye (LeCun bu hafta + Hafta 1): VAE'nin latent kısıtı = architectural enerji şekillendirme (LeCun Bölüm 1); düzenli latent = Hafta 1 manifoldunun pürüzsüz, örneklenebilir hâli.
İleriye: VAE → diffusion (Hafta 9 teaser), latent diffusion (Stable Diffusion), ve üretken modellerin tüm ailesi.

15.11 Bu Dersin Özeti

1. **EBM eğitimi iki sınıf: contrastive** (veride aşağı bas, negatifte yukarı bas) ve **architectural/non-contrastive** (yapıyla kısıtla).
2. **Contrastive SSL:** pozitif çift (augmentation), negatif çift (rastgele); benzeri aşağı, benzemezi yukarı bas (PIRL/NCE, MoCo).



Şekil 15.7: İki boyutlu latent uzay: sıradan AE (sol) vs VAE (sağ). Sıradan otokodlayıcının latent uzayı dağınık, boşluklu kümelerden oluşur; kümeler arasındaki bölge (kesikli daire) anlamsızdır — oradan çekilen bir z, decoder ile geçerli bir veri vermez. VAE ise reparameterization trick ($z = \mu + \sigma \epsilon$) ile latent uzayı dolu ve sürekli bir Gaussian buluta zorlar. Bu sayede herhangi bir z örneği çekip decoder'dan geçirerek YENİ veri üretilebilir — VAE üretkendir, sıradan AE değildir.

3. **Denoising AE:** temizi boz (maskele), geri kur; bozuk = off-manifold = yukarı bas. (BERT'in fikri.)
4. **Contrastive'in sınırı:** yüksek boyutta çok negatif gerekir → post-2020 non-contrastive (BYOL/VICReg/MAE — kursta YOK).
5. **VAE (Canziani):** encoder ortalama + varyans üretir; latent $\mathcal{N}(\mu, \sigma^2)$'den örneklenir (reparameterization $z = \mu + \sigma\epsilon$).
6. **VAE = non-contrastive EBM:** latent'i Gaussian'a kısıtlamak enerjiyi yapıyla şekillendirir; düzenli latent → üretken model.

! Tek Bir Cümle

EBM'yi eğitmek enerjiyi şekillendirmektir: contrastive yöntemler veride enerjiyi düşürüp üretilmiş negatiflerde yükseltir (ama yüksek boyutta çok negatif ister), non-contrastive yöntemler ise yapıyla kısıtlar — VAE bunun zarif örneğidir: latent'i bir Gaussian'a oturtup reparameterization ile eğitilebilir, düzenli, üretken bir uzay kurar.

15.12 Kontrol Soruları

i Soru 1: EBM eğitiminin iki sınıfı nedir? Aralarındaki temel fark?

Cevap: (1) **Contrastive yöntemler:** veri noktalarında enerjiyi **aşağı bas**, üretilmiş negatif örneklerde **yukarı bas** (LeCun 5:20). Örnekler: contrastive divergence, NCE, denoising AE. (2) **Architectural / non-contrastive yöntemler:** modelin **yapısını** kısıtlayarak düşük-enerji bölgesinin hacmini sınırla — negatif örnek üretmeden (örn. bottleneck, sparse coding, VAE). Temel fark: contrastive negatif örnek **üretir ve iter**; non-contrastive negatif gerektirmez, enerjiyi **mimariyle** şekillendirir. (Hafta 7'deki "yalnızca veride düşürürsen model çöker" sorununun iki farklı çözümü.)

i Soru 2: Contrastive SSL'de pozitif ve negatif çiftler nasıl üretilir? Bu yöntemin temel sınırı nedir?

Cevap: Pozitif çift: bir örneği al, **augmentation** ile (kırp/döndür/renk) ikinci versiyonunu üret — ikisi "aynı" sayılır, enerjileri aşağı basılır. **Negatif çift:** rastgele iki farklı örnek — uyumsuz, enerjileri yukarı basılır (LeCun 13:22). Benzeri aşağı, benzemezi yukarı (PIRL/NCE, MoCo). **Sınır:** temsil boyutu arttıkça "dışarısı" büyür, enerjiyi her yerde yükseltmek için **gittikçe daha çok negatif örnek** gerekir (LeCun 22:54) — pahalı ve verimsiz. Bu sınır, post-2020 non-contrastive yöntemleri (BYOL/VICReg — kursta yok) doğurdu.

i Soru 3: Denoising autoencoder neden bir contrastive yöntemdir? BERT ile ilişkisi nedir?

Cevap: DAE, temiz bir y'yi **bozarak** (parça silme, gürültü, maskeleme) x üretir ve ağı x'ten y'yi kurmaya zorlar (LeCun 23:17). Bu contrastive'dir çünkü bozulmuş nokta **manifold dışındadır** (yüksek enerji olmalı); ağ onu temize/manifolda çekerek dışarının enerjisini **yukarı basmış** olur — pozitif (temiz) aşağı, negatif (bozuk) yukarı. **BERT** tam bu fikirdir: metinde bazı kelimeleri **maskele**, modeli onları tahmin etmeye zorla (masked language modeling) — DAE'nin metin hâli. (Hafta 7'nin "off-manifold geri çek" autoencoder'ının contrastive okuması.)

Soru 4: (Builder) VAE sıradan AE'den nasıl farklıdır? Reparameterization trick neden gerekli? Denklemi yaz.

Cevap: Sıradan AE encoder'ı tek bir **kod** üretir; VAE encoder'ı bir **dağılım** üretir — ortalama $\mu(x)$ ve varyans $\sigma^2(x)$ (Gaussian) — ve z bu dağılımdan **örneklenir** (Canziani 3:12). Bu, latent'i düzenli/üretken yapar (varyans $0 \to$ sıradan AE'ye çöker). **Reparameterization gerekli** çünkü backprop rastgele örneklemeden geçemez; rastgeleliği sabit bir ε'a izole edersin:

$$z = \mu(x) + \sigma(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)$$

Artık μ, σ türevlenebilir (gradient onlardan akar, Hafta 5 autograd); yalnızca ε stokastiktir (Stat 110 konum-ölçek $X = \mu + \sigma Z$). VAE bir **non-contrastive EBM**'dir: latent'i Gaussian'a kısıtlamak enerjiyi yapıyla şekillendirir.

15.13 Egzersizler

Egzersiz 1 (Push down/up). 1B bir enerji eğrisi düşün. (a) Yalnızca veri noktalarında enerjiyi düşür — ne olur (enerji her yerde düşer, "çöküş")? (b) Birkaç rastgele negatif noktada enerjiyi yükselt — şekil nasıl düzelir? Contrastive'in neden negatif gerektirdiğini açıkla.

```
import numpy as np

def energy_1d(y, data=(-2.0, 0.0, 2.0), width=0.35):
    y = np.asarray(y, float)
    F = np.ones_like(y)
    for d in data:                                  # her veri noktasi bir cukur
        F = F - np.exp(-(y - d) ** 2 / (2 * width))
    return F

y = np.linspace(-4, 4, 400)
F = energy_1d(y)
# (a) SADECE veride dusur: tum egriyi asagi cek -> her yer duser = COKUS
# (b) contrastive: veri ASAGI (push down) + rastgele negatif YUKARI (push up)
#     -> veride cukur, aralarda tepe = iyi enerji sekli
neg = np.array([-1.0, 1.0])                          # rastgele negatif noktalar
print("veri enerjisi (dusuk):", np.round(energy_1d(np.array([-2, 0, 2])), 2))
print("negatif enerjisi (yukari basilacak):", np.round(energy_1d(neg), 2))
# negatif olmadan -> collapse; negatif ile -> sekillenmis enerji
```

Egzersiz 2 (Pozitif çift = augmentation). Bir görüntüye iki farklı augmentation (kırpma + renk) uygula. Bu iki versiyon neden "pozitif çift"tir? Negatif çift nasıl üretilir? Boyut arttıkça neden daha çok negatif gerekir?

```
import torch

# augment = ayni goruntunun iki gorunumu -> POZITIF cift (ayni sey sayilir)
def augment(img, seed):
```

```
    g = torch.Generator().manual_seed(seed)
    crop = img[:, 4:, 4:]                              # kirpma (rastgele konum)
    color = img * (0.8 + 0.4 * torch.rand(1, generator=g))  # renk jitter
    return crop, color

img = torch.rand(3, 32, 32)
view1, _ = augment(img, 0)
_, view2 = augment(img, 1)
# (view1, view2) = POZITIF cift  -> enerji ASAGI (ayni goruntu, iki gorunum)
other = torch.rand(3, 32, 32)                          # rastgele BASKA goruntu
# (view1, other) = NEGATIF cift -> enerji YUKARI (uyumsuz)
# boyut d artarsa: "disarisi" ustel buyur -> her yeri itmek icin COK negatif gerekir
```

Egzersiz 3 (Denoising AE). Bir görüntüye gürültü/maskeleme uygula, küçük bir DAE ile temizini kur. Bunun (a) Hafta 7'nin "manifolda geri çek"i ve (b) BERT'in maskeleme fikriyle ilişkisini açıkla.

```
import torch
import torch.nn as nn

class DenoisingAE(nn.Module):
    def __init__(self, d=784, code=64):
        super().__init__()
        self.enc = nn.Sequential(nn.Linear(d, code), nn.ReLU())
        self.dec = nn.Sequential(nn.Linear(code, d), nn.Sigmoid())
    def forward(self, x):                              # x = BOZUK girdi
        return self.dec(self.enc(x))                   # -> temiz kurulan

model = DenoisingAE()
clean = torch.rand(1, 784)                             # temiz y (manifoldda)
mask = (torch.rand(1, 784) > 0.3).float()             # %30 maskele
corrupt = clean * mask                                 # bozuk x (off-manifold)
recon = model(corrupt)
loss = ((recon - clean) ** 2).mean()                   # bozuktan temizi kur
# (a) bozuk=off-manifold -> manifolda GERI CEK (Hafta 7)
# (b) maskeleme = BERT fikri (eksik tokeni/pikseli tahmin et, Hafta 12)
```

Egzersiz 4 (VAE kur). PyTorch'ta bir VAE kur: encoder μ ve $\log \sigma^2$ üretsin, reparameterization ile z örnekle, decoder kursun. Reconstruction loss + KL terimini ekle. Latent'ten rastgele z çekip yeni örnek üret (sıradan AE bunu yapamaz — neden?).

```
import torch
import torch.nn as nn

class VAE(nn.Module):
    def __init__(self, d=784, h=400, zdim=20):
        super().__init__()
```

```
        self.enc = nn.Linear(d, h)
        self.fc_mu = nn.Linear(h, zdim)              # ortalama mu
        self.fc_logvar = nn.Linear(h, zdim)          # log varyans
        self.dec1 = nn.Linear(zdim, h)
        self.dec2 = nn.Linear(h, d)
    def reparam(self, mu, logvar):                   # z = mu + sigma * eps
        sigma = torch.exp(0.5 * logvar)
        eps = torch.randn_like(sigma)                # eps ~ N(0, I) (tek stokastik kisim)
        return mu + sigma * eps
    def forward(self, x):
        h = torch.relu(self.enc(x))
        mu, logvar = self.fc_mu(h), self.fc_logvar(h)
        z = self.reparam(mu, logvar)
        recon = torch.sigmoid(self.dec2(torch.relu(self.dec1(z))))
        return recon, mu, logvar

def vae_loss(recon, x, mu, logvar):
    bce = nn.functional.binary_cross_entropy(recon, x, reduction="sum")
    kl = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())   # KL(q||N(0,I))
    return bce + kl                                  # reconstruction + duzenlileme
# Uretim: z = torch.randn(1, 20); ornek = decoder(z)
# AE bunu YAPAMAZ -> latent bosluklu/duzensiz, rastgele z anlamsiz cikti verir
```

Egzersiz 5 (Hafta 9 habercisi — GAN ve dünya modelleri). VAE veriyi olasılıksal kurar; başka bir üretken yaklaşım **GAN**'dır (üretici vs ayırıcı). (a) VAE "veriyi yeniden kur" derken GAN "ayırıcıyı kandır" der — bu iki üretken felsefenin farkı nedir? (b) Hafta 9'da LeCun sparse coding, **dünya modelleri** ve GAN'ı (EBM'nin üçüncü dersi) anlatacak — dünya modeli neden bir EBM'dir (gelecek durumun enerjisi)?

```
# (a) VAE vs GAN — iki uretken felsefe
#   VAE : encoder->dagilim->decoder; kayip = reconstruction + KL
#         "veriyi olasiliksal YENIDEN KUR"  (likelihood-tabanli)
#   GAN : uretici G (gurultuden veri uretir) vs ayirici D (gercek/sahte ayirir)
#         "ayiriciyi KANDIR"  (adversarial, min-max oyun)
#         min_G max_D  E[log D(x)] + E[log(1 - D(G(z)))]
# (b) dunya modeli = EBM:
#   F(s_t, a_t, s_{t+1}) = gelecek durumun enerjisi
#   uyumlu (gercekci) gecis -> DUSUK enerji; uyumsuz -> YUKSEK enerji
#   -> planlama = gelecek durumun enerjisini minimize et (Hafta 9)
print("VAE: yeniden kur (likelihood) | GAN: ayiriciyi kandir (adversarial)")
```

15.14 Sonraki Ders İçin Hazırlık

> ⚠ Sonraki Hafta — H9: Sparse Coding, Dünya Modelleri ve GAN
>
> **Üretken modeller geliyor.** Bu hafta EBM eğitiminin iki yolunu kurduk (contrastive vs non-contrastive) ve VAE'yi gördük. Hafta 9, EBM serisinin üçüncü ve son dersi: LeCun **sparse coding**, **dünya modelleri (world models)** ve **GAN**'ı (hepsi EBM çerçevesinde) anlatacak; Canziani GAN/DCGAN'ı PyTorch'ta gösterecek. Egzersiz 4 (VAE) ve Egzersiz 5 (GAN/dünya modeli habercisi) tam bu derse hazırlar.

Hafta 9: Sparse Coding, Dünya Modelleri ve GAN — LeCun (Lecture) + Canziani (Practicum)

Hafta 9, EBM serisinin üçüncü ve son dersi: LeCun **sparse coding**, **dünya modelleri (world models)** ve **GAN**'ı (hepsi EBM çerçevesinde) anlatacak; Canziani GAN/DCGAN'ı PyTorch'ta gösterecek.

Hafta 9 öncesi yapılacak:

- Egzersiz 1 (push down/up) ve Egzersiz 4 (VAE) çöz.
- "Contrastive vs non-contrastive" ayrımını kendi sözcüklerinle yaz.
- Hafta 7-8'i bağla: EBM çerçevesi (7) → enerji şekillendirme yöntemleri (8) → üretken modeller (9).

15.15 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Contrastive yöntem	Veride enerjiyi aşağı, negatifte yukarı bas	LeCun 5m20
Architectural / non-contrastive	Yapıyla enerji hacmini kısıtla (negatifsiz)	LeCun 6m46
Pozitif / negatif çift	Augmentation (uyumlu) vs rastgele (uyumsuz)	LeCun 13m22
Noise contrastive estimation	PIRL'in contrastive amaç fonksiyonu	LeCun 16m32
Denoising AE (DAE)	Bozulmuşu kur; maskeleme = BERT fikri	LeCun 23m17
Contrastive sınırı	Yüksek boyutta çok negatif gerekir	LeCun 22m54
VAE	Encoder ortalama + varyans üretir; latent örneklenir	Canziani 3m12
Reparameterization	$z = \mu + \sigma \odot \epsilon$; rastgeleliği ε'a izole et	Canziani 3m26
Düzenli latent	Gaussian kısıt → sürekli, üretken latent uzay	Canziani 4m37
VAE = non-contrastive EBM	Latent kısıtı enerjiyi yapıyla şekillendirir	Canziani / LeCun

15.16 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Push down/up = cross-entropy/MLE** → Hafta 1 cross-entropy + Stat 110 (energy = −log p).
2. **VAE latent = Gaussian** → Stat 110 çok-değişkenli normal (mean + diagonal covariance).
3. **Reparameterization = konum-ölçek** → Stat 110 $X = \mu + \sigma Z$ + Calculus zincir kuralı + Hafta 5 autograd.
4. **Contrastive SSL = invariance** → Hafta 3 (augmentation = stationarity/invariance).
5. **DAE = manifolda geri çek** → Hafta 7 autoencoder.

İleriye köprüler (production / research):

1. **Contrastive (MoCo kursta)** → SimCLR; "çok negatif" sorunu → **BYOL/VICReg/MAE (post-2020, KURSTA YOK)**.
2. **DAE maskeleme** → BERT (Hafta 12), MAE (post-2020).
3. **VAE** → diffusion (Hafta 9), latent diffusion (Stable Diffusion).
4. **Non-contrastive EBM** → LeCun JEPA programı (post-2020).

! Bu dersten tek bir şey alıp gideceksen

EBM'yi eğitmek enerji fonksiyonunu şekillendirmektir, ve bunun iki yolu vardır — **contrastive** (veriyi aşağı, üretilmiş negatifi yukarı it; ama yüksek boyutta çok negatif ister) ve **non-contrastive/architectural** (yapıyla kısıtla, negatifsiz). VAE ikincisinin zarif örneğidir: encoder bir dağılım üretir, latent reparameterization ile örneklenir ($z = \mu + \sigma\epsilon$), ve Gaussian kısıt latent'i düzenli, üretken kılar. LeCun'un "çok negatif gerekir" sınırı ise kurstan sonraki non-contrastive devrimi (BYOL/VICReg/MAE — kursta yok) doğurdu.

16 Sparse Coding, Dünya Modelleri ve GAN

İki hocalı hafta — EBM serisinin üçüncü ve son dersi. Yann LeCun (Lecture) Hafta 7-8'i birleştiren güçlü bir ilke verir: non-contrastive (mimari) yöntemlerin ortak özü, latent değişkenin bilgi kapasitesini azaltarak düşük-enerji uzayının hacmini sınırlamaktır — bottleneck (Hafta 7), VAE'de gürültü/varyans (Hafta 8) ve sparse coding (bu hafta) hepsi aynı amaca hizmet eder. Ardından sparse coding'i, group/structural sparsity'yi ve araştırma programının kalbi olan dünya modellerini (belirsiz geleceği öngören latent-variable EBM) anlatır. Alfredo Canziani (Practicum) ise GAN'ı (üretken çekişmeli ağ) gösterir ve onu dersin omurgasıyla tutarlı biçimde bir EBM olarak çerçeveler: generator latent gürültüden sahte örnek üretir, cost network (enerji ağı) gerçeğe düşük, sahteye yüksek enerji verir; generator negatifleri öğrenerek üretir — yani GAN, akıllı/adaptif negatif üreteçli contrastive bir EBM'dir.

ℹ Bölüm bilgisi

- **LeCun'un Lecture videosu:** YouTube — Energy-based models III: sparse coding, world models, GANs (≈118 dk)
- **Canziani'nin Practicum videosu:** YouTube — Generative adversarial networks (≈75 dk)
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, pratik)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈22 dk

16.1 Bu Derste Ne Var?

Bu hafta **EBM serisinin üçüncü ve son dersi**. **Yann LeCun** Hafta 7-8'i birleştiren bir ilke veriyor (non-contrastive yöntemlerin ortak özü: düşük-enerji uzayının hacmini sınırlamak), **sparse coding**'i ve **dünya modellerini (world models)** anlatıyor. **Alfredo Canziani** ise **GAN**'ı (üretken çekişmeli ağ) gösteriyor — ve onu, dersin omurgasıyla tutarlı biçimde, bir **EBM** olarak çerçeveliyor.

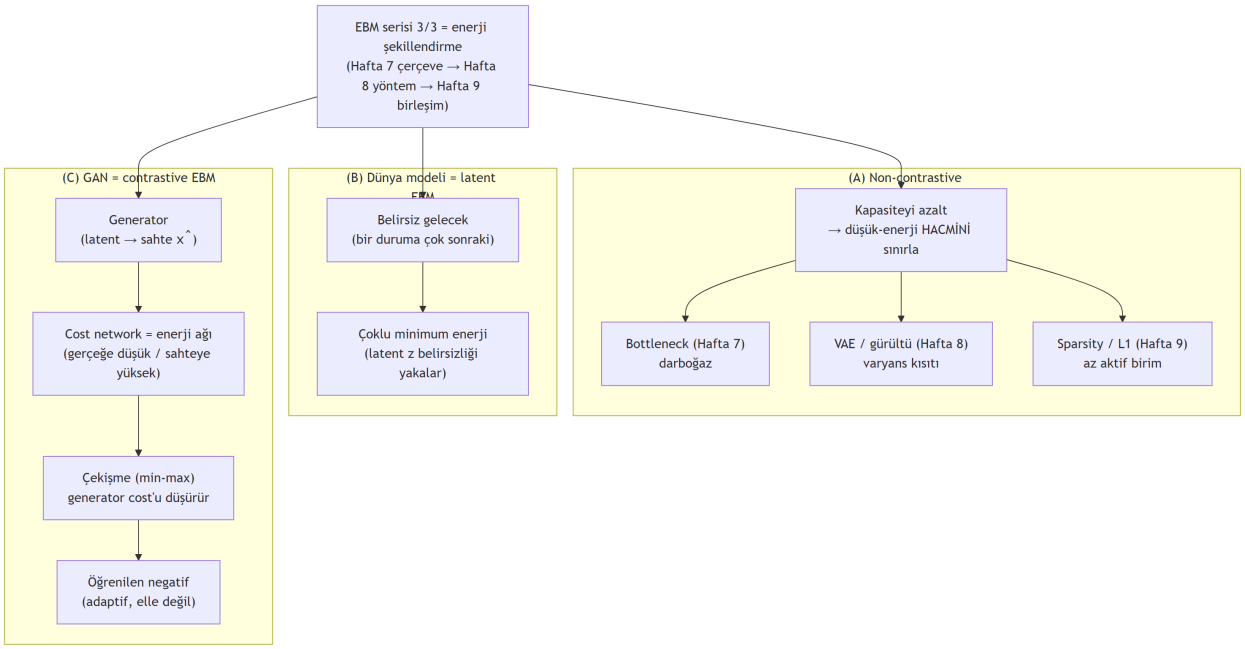
⚠ ⚠ Terminoloji notu (NYU sinyatürü)

Canziani GAN'daki "discriminator" terimini kullanmıyor; LeCun'u izleyerek ona **cost network (maliyet/enerji ağı)** diyor. Yani GAN, dersin EBM diline çevrilmiş: cost = enerji, generator negatifleri üretir.

Bu haftanın üç ana fikri:

1. **Non-contrastive yöntemler tek ilkede birleşir:** düşük-enerji bölgesinin **hacmini sınırla**. Sparsity, gürültü ekleme (VAE), bottleneck — hepsi aynı amaca hizmet eder.

2. **Dünya modeli (world model):** geleceği belirsizlik altında öngören bir EBM; LeCun'un araştırma programının kalbi.
3. **GAN = contrastive EBM:** generator sahte örnekler (negatifler) üretir; cost network gerçeğe düşük, sahteye yüksek enerji verir; generator cost'u düşürerek (kandırarak) öğrenir.



Builder Notu — EBM Serisi Tek Çatıda Kapanır

Geriye (önkoşul kurslar):

- **Düşük-enerji hacmini sınırla** → Hafta 7 bottleneck + Hafta 8 VAE/gürültü + bu hafta sparsity (üçü aynı ilke).
- **Sparse coding (L1)** → 18.06 (seyrek temsil) + Stat 110 (Laplace prior).
- **GAN = cost/enerji + generator** → Hafta 7 EBM + Hafta 8 contrastive (generator = öğrenilen negatif üretici).

İleriye (production / research):

- World model → LeCun'un JEPA programı (post-2020 ileriye köprü); model-based RL, planning.
- GAN → CycleGAN, StyleGAN; ve diffusion (modern üretken modeller).

Tek cümleyle: Non-contrastive EBM yöntemleri (sparsity, VAE, bottleneck) düşük-enerji uzayının hacmini sınırlayarak çalışır; GAN ise contrastive bir EBM'dir — generator negatifleri *öğrenerek* üretir, cost network onları yukarı iter.

16.2 (LeCun) EBM Serisi 3/3 ve Birleştirici İlke

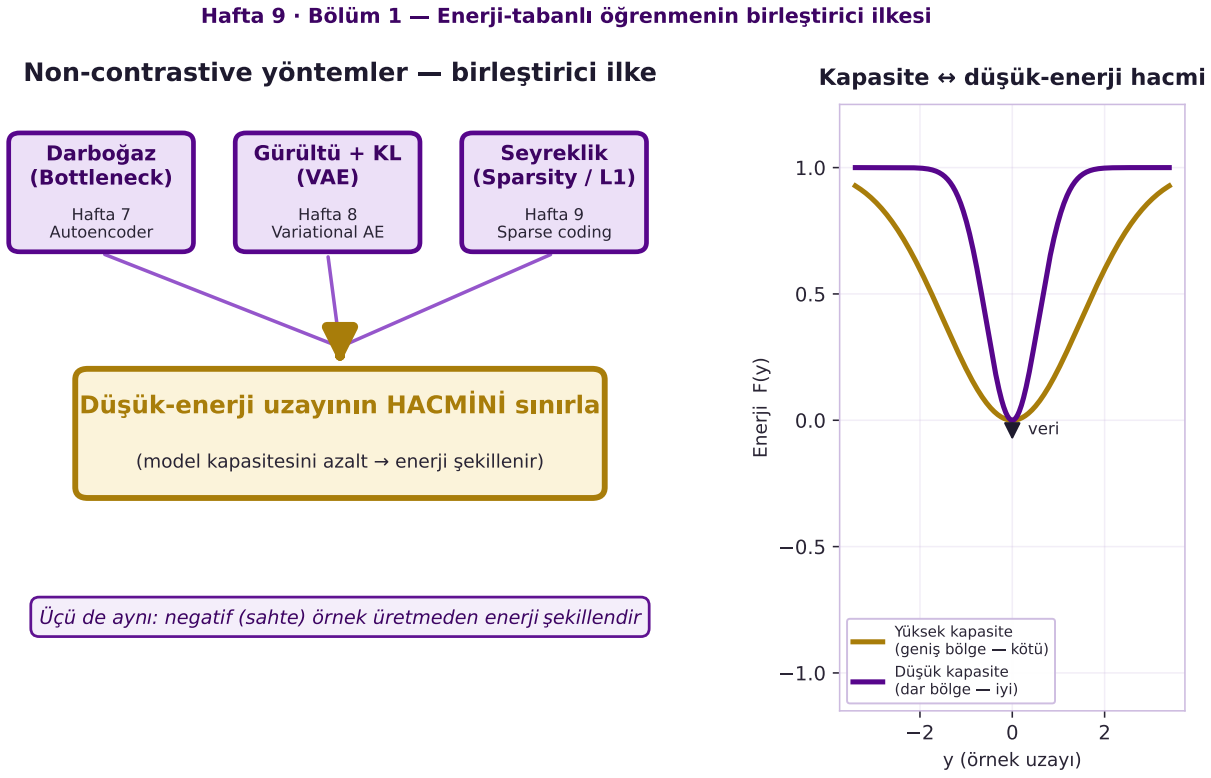
LeCun dersi "EBM'nin üçüncü parçası" diye açıyor (sparse coding, GAN'a kısa giriş, world models).

> "this is the third part of the lecture on energy based models… sparse coding… GANs very briefly… learning world models." — LeCun, 0:00

Önce Hafta 7-8'i birleştiren güçlü bir ilke veriyor. Hatırla: contrastive yöntemler negatifte enerji yukarı basar; non-contrastive yöntemler **yapıyla** kısıtlar. LeCun şimdi non-contrastive ailenin **ortak özünü** söylüyor: latent değişkenin (code) **bilgi kapasitesini azaltırsan**, düşük enerji alabilen uzayın **hacmini** azaltırsın.

> "if you reduce the information capacity of the latent variable… as a consequence you also minimize the volume of space that can take low energy." — LeCun, 16:00

Bu yüzden bottleneck (Hafta 7), VAE'de gürültü/varyans (Hafta 8) ve sparsity (bu hafta) **aynı şeyi yapar**: düşük-enerji hacmini sınırlar. Veriye düşük enerji verirsen, kapasite sınırlı olduğu için gerisi otomatik yükselir — negatif örnek üretmeden. Şekil 16.1 bu birleştirici ilkeyi şematize ediyor: üç yöntem (bottleneck, VAE-gürültü, sparsity) tek bir oka iner ("düşük-enerji hacmini sınırla"), ve sağdaki küçük görsel sezgiyi pekiştirir — yüksek kapasite geniş bir düşük-enerji bölgesi (kötü), düşük kapasite veriye sıkıca oturan dar bir bölge (iyi) bırakır.



Şekil 16.1: Hafta 9 Bölüm 1 — Enerji-tabanlı öğrenmenin birleştirici (non-contrastive) ilkesi. Üç farklı yöntem — darboğaz/autoencoder (Hafta 7), gürültü + KL ile VAE (Hafta 8) ve seyreklik/L1 sparse coding (Hafta 9) — tek bir ilkeye iner: düşük-enerji uzayının HACMİNİ sınırlamak, yani model kapasitesini azaltarak enerji manzarasını şekillendirmek. Üçü de negatif (sahte) örnek üretmeden çalışır. Sağdaki küçük görsel sezgiyi pekiştirir: yüksek kapasite geniş bir düşük-enerji bölgesi (kötü, her şeye düşük enerji verir) yaratırken, düşük kapasite veriye sıkıca oturan dar bir bölge (iyi) bırakır.

Builder Notu — Kapasite = Hacim

Geriye (Hafta 7-8): Bu, Hafta 7 (bottleneck) + Hafta 8 (VAE gürültü) + bu hafta (sparsity) üçlüsünü tek cümlede birleştirir: "düşük-enerji hacmini sınırla". Kapasite kısıtı = düzenlileştirme (Hafta 1 regularization).
İleriye: "Kapasiteyi sınırla → düşük-enerji hacmini sınırla" ilkesi, modern non-contrastive SSL'in (VICReg'in covariance terimi, post-2020) teorik temelidir.

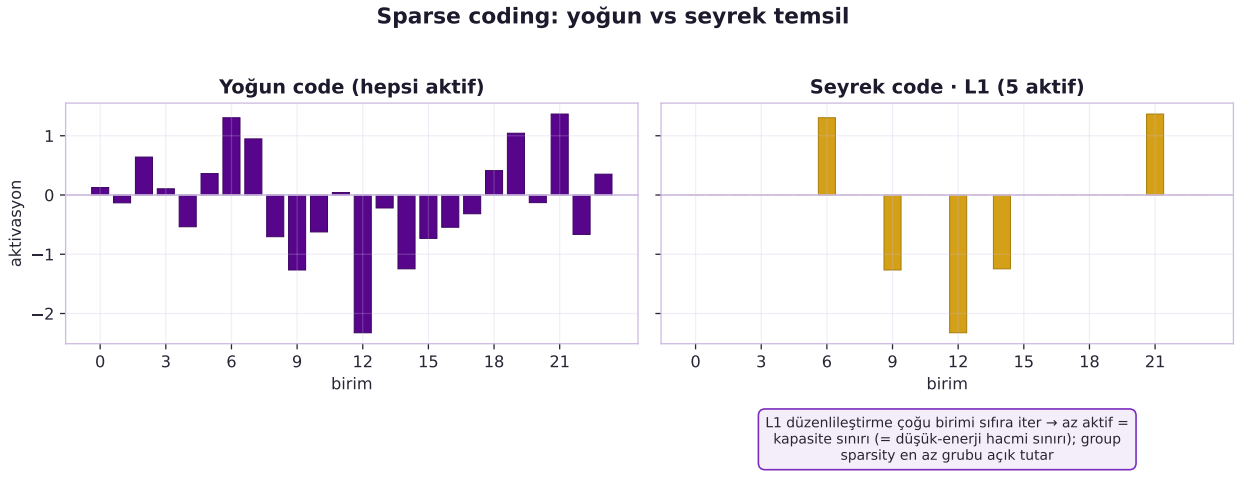
16.3 (LeCun) Sparse Coding ve Group/Structural Sparsity

Sparse coding: bir girdiyi, az sayıda aktif birimle (seyrek code) temsil et. Bir L1 düzenlileştirici, code'un çoğu bileşenini sıfıra iter — yalnızca birkaçı aktif kalır. Bu, kapasiteyi sınırlamanın (Bölüm 1) bir yoludur.

LeCun bunu **group sparsity** ve **structural sparsity** ile genişletiyor: birimleri gruplar (örn. 2B topolojide 16×16 matris, ya da bir graf/ağaç) hâlinde düzenle; düzenlileştirici en az sayıda **grubu** açık tutmaya çalışır.

> "[the sparsity regularizer] tends to turn off the maximum number of groups… the system [imposes] sparsity on groups." — LeCun, 8:01

Gruplar 2B topolojide organize edilirse, yan yana birimler birlikte aktif olur — bu, complex cell'lere (Hafta 3) benzer bir yapı öğrenir. Şekil 16.2 aynı temsili yoğun (24 birimin hepsi aktif) ve seyrek (yalnızca 5 birim aktif) biçimde gösteriyor: L1 düzenlileştirme aktivasyonların çoğunu sıfıra iter, kapasiteyi — yani düşük-enerji hacmini — sınırlar.



Şekil 16.2: Sparse coding: aynı temsil yoğun (solda, 24 birimin hepsi aktif) ve seyrek (sağda, yalnızca 5 birim aktif) biçimde. L1 düzenlileştirme aktivasyonların çoğunu sıfıra iter; az sayıda aktif birim modelin kapasitesini — yani düşük-enerji hacmini — sınırlar (EBM bakışı). Group sparsity en az grubu açık tutarak yapılı bir seyreklik dayatır.

Builder Notu — Sparse = Az Aktif

Geriye (18.06 + Stat 110 + Hafta 3): Seyrek temsil = az sayıda baz vektörle ifade (18.06); L1 düzenleştirme = Laplace prior (Stat 110). Group sparsity'nin öğrendiği topoloji, Hafta 3'ün complex cell organizasyonunu andırır.
İleriye: Sparsity, yorumlanabilir temsil (interpretable features) ve verimli (seyrek) ağların temelidir; mixture-of-experts'in (seyrek aktivasyon) uzak akrabası.

16.4 (LeCun) Dünya Modelleri (World Models)

LeCun araştırma programının kalbine değiniyor: **dünya modelleri**. Bir dünya modeli, mevcut durumdan ve eylemden **geleceği öngörür** — ama gelecek belirsizdir (bir duruma birçok olası sonraki durum). Bu yüzden dünya modeli doğal olarak bir **EBM**'dir: olası gelecek durumlara düşük enerji, olanaksızlara yüksek enerji verir; belirsizliği latent değişkenlerle modeller.

Bu, Hafta 7'nin latent-variable EBM'sinin zaman/öngörü hâlidir: y (gelecek) belirsizdir, z (latent) belirsizliği yakalar. Tahminin belirsizliğini bir enerji manzarasıyla temsil etmek, tek-çıktılı bir öngörücünün yapamadığı şeydir (bulanık ortalama yerine olası gelecekler). Şekil 16.3 bu farkı bir enerji manzarasında gösteriyor: çoklu yerel minimum (gold yıldızlar) olası gelecekleri, violet X ise tek-çıktı öngörücünün ürettiği — hiçbir gerçek geleceğe karşılık gelmeyen — bulanık ortalamayı temsil eder.

Builder Notu — Dünya Modeli = Latent EBM

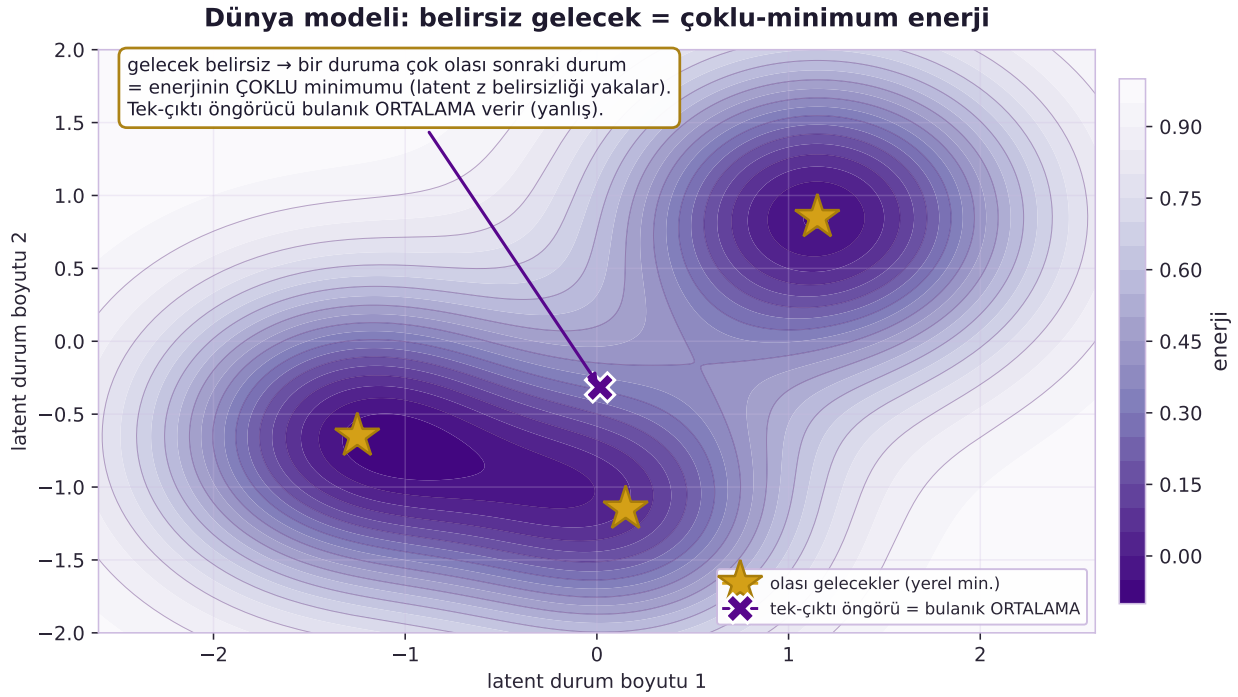
Geriye (Hafta 7 + 6): Dünya modeli = latent-variable EBM (Hafta 7) + dizi/öngörü (Hafta 6); belirsiz gelecek = enerji fonksiyonunun birden çok minimumu.
İleriye: World model, LeCun'un bugünkü **JEPA** programının (post-2020 ileriye köprü) ve model-based RL / planning'in merkezidir — cevabı üretmek yerine enerji üzerinden *planla*.

16.5 Geçiş: LeCun'dan Canziani'ye

LeCun non-contrastive ilkeyi (düşük-enerji hacmini sınırla), sparse coding'i ve world model'leri kurdu — ve GAN'a kısaca değindi ("yarın Alfredo'dan daha çok duyacaksınız"). Şimdi **Canziani** GAN'ı açıyor. Kritik olan: Canziani GAN'ı sıfırdan **EBM diliyle** anlatıyor — "discriminator" yerine **cost network** (enerji ağı) diyor. Böylece GAN, Hafta 7-8'in enerji çerçevesine tam oturuyor: generator negatifleri üretir, cost network enerjiyi şekillendirir.

16.6 (Canziani) GAN: Generator + Cost Network (= Enerji)

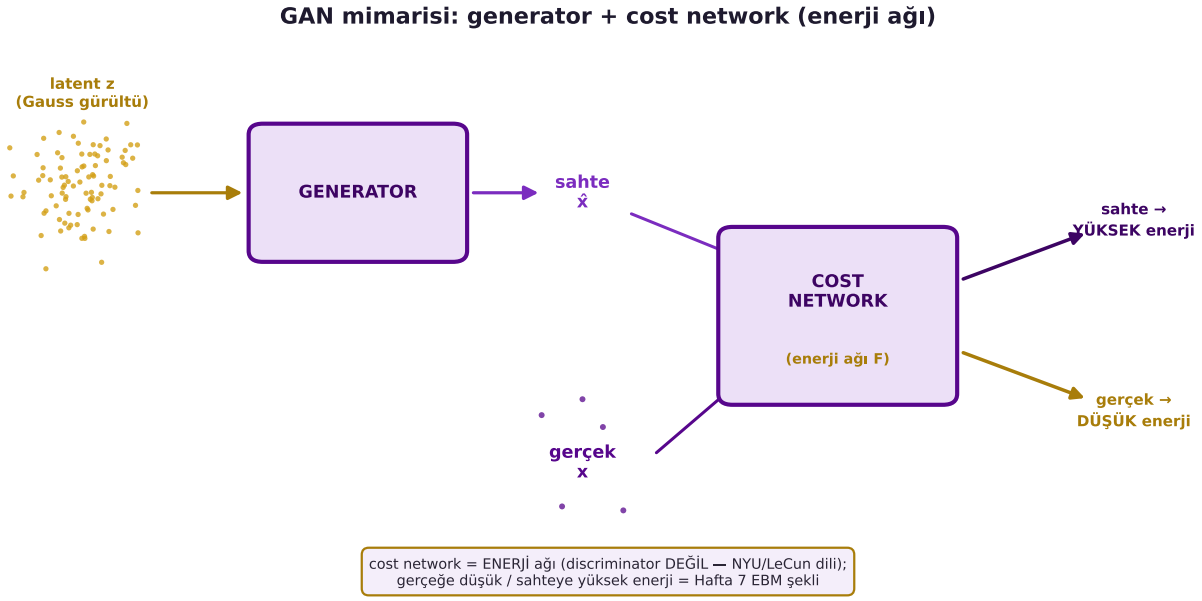
Canziani GAN'ı iki parçayla kuruyor. Bir **generator**, bir latent z (Gaussian örneği) alıp sahte bir örnek $\hat{x}$ üretir. Bir **cost network** (klasik adıyla discriminator — ama Canziani/LeCun bunu yanlış bulup "cost network" der) her girdiye bir maliyet (enerji) atar.



Şekil 16.3: Dünya modeli, belirsiz geleceği latent enerji-tabanlı modelle yakalar: mevcut bir durumdan birden çok olası sonraki durum bulunduğu için enerji manzarasının çoklu yerel minimumu (gold yıldızlar) vardır. Tek-çıktı veren öngörücü tüm bu olasılıkların ortalamasını alır (violet X) ve hiçbir gerçek geleceğe karşılık gelmeyen, manifold dışı bulanık bir tahmin üretir. Latent değişken z bu belirsizliği temsil ederek modelin tek bir bulanık ortalama yerine ayrık geçerli gelecekleri ayırt etmesini sağlar.

> "in the classical formulation of a GAN we have a discriminator — [but] discriminators are just plain wrong, at least following Yann… we have this cost network." — Canziani, 2:37

Cost network gerçek (pembe) örneklere **düşük** maliyet, sahte (mavi, generator'dan gelen $\hat{x}$) örneklere **yüksek** maliyet verir. Yani cost = enerji: gerçek veride düşük, sahtede yüksek — tam Hafta 7'nin EBM şekli. Şekil 16.4 bu mimariyi gösteriyor: latent gürültü z (gold bulut) generator'dan geçip sahte $\hat{x}$ üretir; hem gerçek x hem sahte $\hat{x}$ cost network'e girer ve ağ gerçeğe düşük, sahteye yüksek enerji atar.



Şekil 16.4: GAN mimarisi: latent gürültü z (Gauss bulutu) generator'dan geçerek sahte $\hat{x}$ üretir; hem gerçek x hem sahte $\hat{x}$ cost network'e (enerji ağı F) girer. Ağ gerçeğe DÜŞÜK, sahteye YÜKSEK enerji atar — bu, Hafta 7'deki EBM enerji şeklinin ta kendisidir. NYU/LeCun dilinde bu modül 'discriminator' değil 'cost network' olarak adlandırılır.

> Builder Notu — Cost Network ≠ Discriminator
>
> **Geriye (Hafta 7):** Cost network = enerji fonksiyonu (LeCun'un diyagramında "cost" karesi, skaler çıktı); gerçeğe düşük, sahteye yüksek = Hafta 7'nin "veride düşük, dışında yüksek" şekli.
> **İleriye:** GAN'ın "cost network" çerçevesi, EBM-tabanlı üretken modellere ve modern critic/reward modellerine (RLHF) köprüdür.

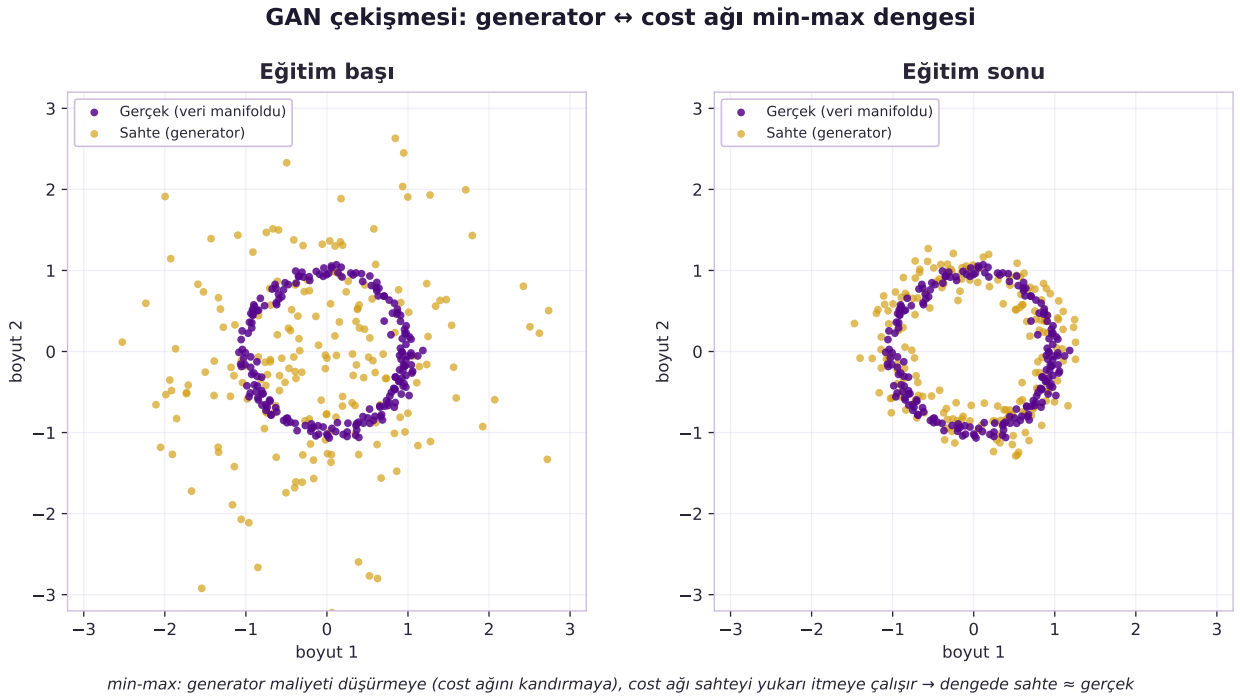
16.7 (Canziani) GAN Eğitimi: Çekişme (Generator Kandırır)

GAN eğitimi bir **çekişmedir (adversarial)**: iki ağ birbirine karşı oynar.

- **Cost network** öğrenir: gerçek örneklere düşük, sahtelere yüksek maliyet ver (enerjiyi doğru şekle sok).
- **Generator** öğrenir: cost network'ün **düşük** maliyet vereceği örnekler üret — yani onu kandır, sahtelerini "gerçek gibi" göster.

> "the cost network will be trained to have low cost for inputs that are [real] and high cost for inputs that are [fake]… [the generator] is going to be enforced to have a low cost [for its outputs]." — Canziani, 5:23 / 7:18

Bu min-max dansı: generator cost'u düşürmeye, cost network sahteyi yukarı itmeye çalışır. Dengede generator, gerçeğe çok benzeyen örnekler üretir. Klasik GAN amacı, Hafta 1-2'nin cross-entropy'siyle (gerçek/sahte ikili sınıflandırma) yazılır. Şekil 16.5 bu çekişmeyi gösteriyor: solda eğitim başında sahte örnekler (gold) dağınık bir buluttur, sağda eğitim sonunda gerçek veri manifolduna (violet halka) yakınsar.



Şekil 16.5: GAN çekişmesi (min-max): solda eğitim başında generator'ın ürettiği sahte örnekler (gold) dağınık bir buluttur ve gerçek veri manifoldundan (violet birim halka) uzaktır; sağda eğitim sonunda sahte örnekler halkaya yakınsar (sahte ≈ gerçek). Generator maliyeti düşürmeye (cost ağını kandırmaya), cost ağı ise sahteyi yukarı itmeye çalışır; denge bu iki kuvvetin eşitlendiği noktadır.

Builder Notu — Min-Max Çekişme

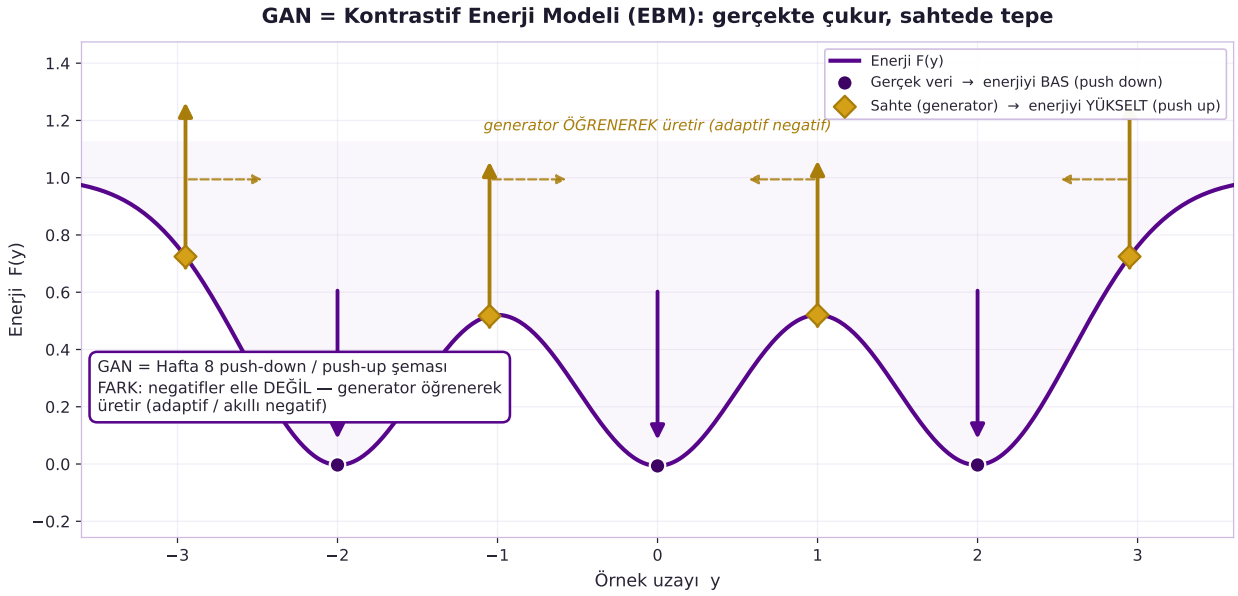
Geriye (Hafta 1-2): Cost network'ün gerçek/sahte ayrımı = Hafta 1 cross-entropy (Bernoulli/ikili sınıflandırma); min-max = Calculus eyer noktası (saddle point).
İleriye: GAN eğitimi kararsızdır (mode collapse, dengesizlik); WGAN, spectral normalization gibi teknikler bunu yumuşatır.

16.8 (Canziani) GAN = Contrastive EBM

İşte dersin birleştirici fikri: GAN, bir **contrastive EBM**'dir. Hafta 8'de contrastive yöntemlerin negatif örnek ürettiğini görmüştük — ama oradaki negatifler elle (augmentation/rastgele) üretiliyordu. GAN'da **generator,**

negatifleri öğrenerek üretir: cost network'ün en çok kandığı (en düşük enerjili sahte) örnekleri arar. Yani GAN = "akıllı negatif üreteçli" contrastive EBM.

Cost network gerçeğe düşük enerji (push down), generator'ın sahtelerine yüksek enerji (push up) verir — Hafta 8'in push-down/push-up'ı, ama negatifler artık sabit değil, **rakip bir ağ tarafından adaptif üretiliyor**. Bu, GAN'ı dersin EBM omurgasına tam oturtur. Şekil 16.6 bunu enerji eğrisi üzerinde gösteriyor: gerçek veri noktalarında çukur (violet, push-down), generator'ın sahtelerinde tepe (gold, push-up); kesikli yatay oklar ise generator'ın öğrenerek sahteleri çukurlara doğru taşıdığını (adaptif negatif) vurgular.



Şekil 16.6: GAN, kontrastif bir enerji modeli (EBM) olarak. Enerji eğrisi F(y) gerçek veri noktalarında çukur (violet noktalar, aşağı oklar = push-down), generator'ın ürettiği sahte noktalarda tepe yapar (gold elmaslar, yukarı oklar = push-up). Kesikli yatay oklar adaptifliği gösterir: Hafta 8'deki sabit/elle negatiflerin aksine generator, sahteleri öğrenerek çukurlara doğru taşır (akıllı/adaptif negatif).

Builder Notu — GAN = Öğrenilen Negatif

Geriye (Hafta 8): GAN = contrastive EBM (push down gerçek, push up sahte); fark, negatiflerin öğrenilen bir generator'dan gelmesi. Hafta 8 contrastive + Hafta 7 enerji = GAN.
İleriye: "Öğrenilen negatif üreteci" fikri, modern üretken modellerin ve adversarial eğitimden RLHF reward-model'e uzanan tekniklerin ortak teması.

16.9 Bu Dersin Özeti

1. **Non-contrastive birleştirici ilke:** latent kapasitesini azalt → düşük-enerji uzayının hacmini sınırla. Sparsity, gürültü (VAE), bottleneck — hepsi aynı (Hafta 7-8-9 birleşir).
2. **Sparse coding:** az aktif birimle temsil (L1); group/structural sparsity en az grubu açık tutar.
3. **Dünya modeli:** belirsiz geleceği öngören latent-variable EBM (LeCun programının kalbi).
4. **GAN:** generator (latent → sahte $\hat{x}$) + cost network (= enerji; gerçeğe düşük, sahteye yüksek). "Discriminator" yerine "cost network" (NYU/LeCun dili).

5. **GAN eğitimi = çekişme:** cost network sahteyi yukarı iter, generator cost'u düşürür (kandırır); min-max.
6. **GAN = contrastive EBM:** generator negatifleri *öğrenerek* üretir — Hafta 8 contrastive'in adaptif-negatifli hâli.

! Tek Bir Cümle

EBM serisinin son dersi her şeyi tek çatıda topluyor: non-contrastive yöntemler (sparsity, VAE, bottleneck) düşük-enerji uzayının hacmini sınırlayarak enerjiyi şekillendirir; GAN ise contrastive bir EBM'dir — generator negatifleri öğrenerek üretir, cost network (enerji) gerçeğe düşük, sahteye yüksek değer vererek onu hizaya sokar.

16.10 Kontrol Soruları

i Soru 1: Non-contrastive yöntemleri (sparsity, VAE, bottleneck) birleştiren ilke nedir? LeCun bunu nasıl ifade eder?

Cevap: Üçü de aynı şeyi yapar: **latent değişkenin (code) bilgi kapasitesini azaltır → düşük enerji alabilen uzayın hacmini sınırlar** (LeCun 16:00). Kapasite sınırlıysa, veriye düşük enerji verdiğinde geri kalan otomatik olarak yükselir — negatif örnek üretmeden. Bottleneck (Hafta 7) code'u küçültür; VAE (Hafta 8) gürültü/varyansla kapasiteyi sınırlar; sparsity (bu hafta) çoğu birimi sıfıra iter. Hepsi "düşük-enerji hacmini sınırla" ilkesinin farklı uygulamalarıdır — contrastive'in (negatif itme) alternatifi.

i Soru 2: Dünya modeli neden bir EBM'dir? Tek-çıktılı bir öngörücüden farkı nedir?

Cevap: Dünya modeli mevcut durum + eylemden **geleceği öngörür**, ama gelecek **belirsizdir** (bir duruma birçok olası sonraki durum). Tek-çıktılı bir öngörücü bunları tek bir çıktıya — genelde bulanık bir ortalamaya — sıkıştırır (yanlış). EBM ise olası geleceklere **düşük enerji**, olanaksızlara **yüksek enerji** verir; belirsizliği **latent değişkenle** (z) yakalar (Hafta 7 latent-variable EBM). Yani dünya modeli = öngörü + latent-variable EBM; birden çok geçerli gelecek = enerjinin birden çok minimumu.

i Soru 3: Canziani neden 'discriminator' yerine 'cost network' diyor? GAN'ın iki parçası ne yapar?

Cevap: Çünkü GAN'ı dersin **EBM diline** çeviriyor: "cost network" bir **enerji fonksiyonudur** — Canziani/LeCun "discriminator" terimini yanlış bulur (2:37). İki parça: **Generator** latent bir z (Gaussian örneği) alıp sahte örnek $\hat{x}$ üretir; **cost network** her girdiye bir enerji (maliyet) atar — gerçeğe (pembe) düşük, sahteye (mavi, generator'dan) yüksek. Yani cost = enerji, ve "gerçeğe düşük/sahteye yüksek" tam Hafta 7'nin EBM şeklidir.

i Soru 4: (Builder) GAN neden bir contrastive EBM'dir? Hafta 8'in contrastive yöntemlerinden farkı nedir?

Cevap: GAN, Hafta 8'in **push-down/push-up**'ını yapar: cost network gerçeğe düşük enerji (push down), generator'ın sahtelerine yüksek enerji (push up) verir. Bu yüzden bir **contrastive EBM**'dir. **Fark:** Hafta 8'de negatifler elle/sabit üretiliyordu (augmentation, rastgele); GAN'da negatifleri **generator**

öğrenerek, adaptif üretir — cost network'ün en çok kandığı (en düşük enerjili) sahteleri arar. Yani GAN = "akıllı/öğrenilen negatif üreteçli" contrastive EBM; min-max çekişme bu adaptif negatif üretiminin sonucudur.

16.11 Egzersizler

Egzersiz 1 (Kapasite = hacim). Bir autoencoder'ın code boyutunu $32 \to 8 \to 2$ küçült. Her durumda reconstruction kalitesi ve "düşük enerji alan uzay" nasıl değişir? "Kapasiteyi sınırla = düşük-enerji hacmini sınırla" ilkesini gözlemle.

```
import numpy as np

# Kapasite (code boyutu) kuculdukce "dusuk enerji alan uzay" daralir.
# Sema: genis code -> her seyi yeniden kurabilir (genis dusuk-enerji = kotu);
#       dar code -> sadece veriyi kurabilir (dar dusuk-enerji = iyi).
def low_energy_width(code_dim, data_dim=32):
    # kabaca: dusuk-enerji hacmi ~ code kapasitesiyle orantili (sezgi)
    return code_dim / data_dim

for code in (32, 8, 2):
    w = low_energy_width(code)
    print(f"code={code:2d} -> dusuk-enerji hacmi ~ {w:.3f}  "
          f"({'genis=kotu' if w > 0.5 else 'dar=iyi (kapasite sinirli)'})")
# code=32 -> hacim ~1.0 (her sey dusuk = kotu); code=2 -> dar (veriye oturur = iyi)
```

Egzersiz 2 (Sparse coding). Bir code'a L1 düzenlileştirme ekle (loss += $\lambda \cdot \|\text{code}\|_1$). λ'yı artırınca kaç birim aktif kalıyor? Group sparsity (birimleri gruplayıp grup-normuna L1) ne farklı yapar?

```
import numpy as np

# L1 yumusatma (soft-threshold): code = sign(z) * max(|z| - lam, 0)
# lam buyudukce daha cok birim sifira gider (aktif birim sayisi DUSER).
def soft_threshold(z, lam):
    return np.sign(z) * np.maximum(np.abs(z) - lam, 0.0)

rng = np.random.default_rng(0)
z = rng.normal(0, 1, 24)                          # ham code (yogun)
for lam in (0.0, 0.5, 1.0, 1.5):
    code = soft_threshold(z, lam)
    n_active = int(np.count_nonzero(code))
    print(f"lambda={lam:.1f} -> aktif birim = {n_active:2d} / 24")
# lambda buyudukce aktif birim DUSER (L1 cogu birimi sifira iter)
# group sparsity: birimleri gruplara ayir, grup-normuna L1 -> EN AZ grubu acik tut
```

Egzersiz 3 (GAN = cost network). Küçük bir GAN kur (generator + "cost network"/discriminator). Cost network'ün gerçek/sahte örneklere verdiği değerleri eğitim boyunca izle: gerçek düşük, sahte yüksek mi? Bunun Hafta 7 enerji şekliyle ilişkisini açıkla.

```
import torch
import torch.nn as nn

# Generator: gurultu z -> sahte ornek ;  Cost network: ornek -> enerji (skaler)
G = nn.Sequential(nn.Linear(8, 32), nn.ReLU(), nn.Linear(32, 2))
Cost = nn.Sequential(nn.Linear(2, 32), nn.ReLU(), nn.Linear(32, 1))  # = enerji agi

real = torch.randn(64, 2) * 0.1 + torch.tensor([1.0, 0.0])   # gercek veri (manifold)
z = torch.randn(64, 8)
fake = G(z)                                                    # sahte ornekler

E_real = Cost(real).mean()                                     # gercege DUSUK olmali
E_fake = Cost(fake).mean()                                     # sahteye YUKSEK olmali
# cost network kaybi: gercegi bas (dusur), sahteyi yukselt -> Hafta 7 EBM sekli
cost_loss = E_real - E_fake          # minimize et: E_real kucuk, E_fake buyuk
# generator kaybi: cost'u DUSUR (kandir) -> E_fake'i kucult
gen_loss = Cost(G(z)).mean()
print("cost(gercek)=enerji DUSUK, cost(sahte)=enerji YUKSEK -> Hafta 7 enerji sekli")
```

Egzersiz 4 (Mode collapse). GAN'ını eğit; generator'ın hep aynı birkaç örneği üretmeye başladığını (mode collapse) gözlemlemeye çalış. Bu, "generator cost'u kandırmanın kolay yolunu buldu" olarak nasıl yorumlanır?

```
import torch

# Mode collapse: generator cost'u kandirmanin KOLAY yolunu bulur ->
# tum z'leri ayni (dusuk-enerjili) cikti'ya esler (cesitlilik kaybolur).
def mode_collapse_score(samples):
    # cikti cesitliligi: dusukse -> collapse (hep ayni ornek)
    return samples.std(dim=0).mean().item()

healthy = torch.randn(200, 2)                        # cesitli cikti (saglikli)
collapsed = torch.randn(200, 2) * 0.02 + 1.0         # neredeyse tek nokta (collapse)
print(f"saglikli cesitlilik = {mode_collapse_score(healthy):.3f}")
print(f"collapse cesitlilik  = {mode_collapse_score(collapsed):.3f}")
# collapse: generator tek bir "cost'u kandiran" noktayi bulup oraya yiginlasti
# -> cesitlilik ~0; min-max dengesi bozuldu (WGAN/spectral norm bunu yumusatir)
```

Egzersiz 5 (Hafta 10 habercisi — görüde SSL). Hafta 8'de contrastive SSL'i (pozitif/negatif çift) gördük. Hafta 10'da **konuk Ishan Misra** bunu görüde derinleştirecek (pretext görevler, PIRL, ClusterFit). (a) Etiketsiz görüntülerden bir "pretext görevi" (örn. döndürme açısını tahmin et) nasıl gözetim sinyali üretir? (b) Bu, etiketli veri pahalı olduğunda neden değerlidir?

```python
import torch

# (a) Pretext gorevi: ETIKETSIZ goruntuyu dondur, ACIYI tahmin ettir.
#     Donme acisi "bedava etiket" uretir -> oz-denetimli gozetim sinyali.
def make_rotation_task(img):
    angles = [0, 90, 180, 270]                        # 4 sinif (bedava etiket)
    k = torch.randint(0, 4, (1,)).item()
    rotated = torch.rot90(img, k, dims=[-2, -1])      # k*90 derece dondur
    return rotated, k                                 # (girdi, otomatik etiket)

img = torch.rand(3, 32, 32)
rot, label = make_rotation_task(img)
print(f"pretext etiketi (donme sinifi) = {label}  -> ag bunu tahmin ederek ogrenir")
# (b) etiketli veri PAHALI; pretext gorevi etiketi BEDAVA uretir (oz-denetim)
#     -> bol etiketsiz veriyle on-egitim, sonra az etiketle ince-ayar (Hafta 10)
```

16.12 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H10: Görüde Öz-Denetimli Öğrenme (SSL) ve PPUU (Konuk: Ishan Misra)

SSL görüye iniyor. Bu hafta EBM serisini kapattık: non-contrastive birleştirici ilke (kapasite → düşük-enerji hacmi), sparse coding, dünya modelleri ve GAN (= contrastive EBM, öğrenilen negatif). Hafta 10'da **konuk hoca Ishan Misra** (Facebook AI Research) görüde öz-denetimli öğrenmeyi anlatacak (pretext görevler, ClusterFit, **PIRL**); Canziani ise belirsizlik altında öngörü/politika öğrenmeyi (PPUU) — dünya modellerinin pratiğini — gösterecek. Egzersiz 3 (GAN cost network) ve Egzersiz 5 (pretext görev) tam bu derse hazırlar.

Hafta 10: Görüde Öz-Denetimli Öğrenme (SSL) ve PPUU — Konuk: Ishan Misra (Lecture) + Canziani (Practicum)

Hafta 10'da **konuk hoca Ishan Misra** (Facebook AI Research) görüde öz-denetimli öğrenmeyi anlatacak: pretext görevler, ClusterFit, **PIRL**. Canziani ise belirsizlik altında öngörü/politika öğrenmeyi (PPUU) — dünya modellerinin pratiğini — gösterecek.

Hafta 10 öncesi yapılacak:

- Egzersiz 3 (GAN cost network) ve Egzersiz 5 (pretext görev) çöz.
- "GAN = contrastive EBM (öğrenilen negatif)" ve "non-contrastive = düşük-enerji hacmini sınırla" cümlelerini kendi sözcüklerinle yaz.
- Hafta 8 contrastive SSL'i hatırla — Hafta 10 onu görüye uygular.

16.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Düşük-enerji hacmini sınırla	Non-contrastive'in ortak ilkesi (kapasite azalt)	LeCun 16m00
Sparse coding	Az aktif birimle temsil; L1 düzenlileştirme	LeCun 0m33
Group / structural sparsity	En az sayıda grubu açık tut; topolojik organizasyon	LeCun 8m01
Dünya modeli (world model)	Belirsiz geleceği öngören latent-variable EBM	LeCun 0m22
GAN	Generator (latent→sahte) + cost network (enerji)	Canziani 1m55
Cost network (≠ discriminator)	Enerji ağı; gerçeğe düşük, sahteye yüksek	Canziani 2m37
GAN eğitimi (çekişme)	Cost sahteyi yukarı iter, generator cost'u düşürür	Canziani 5m23
GAN = contrastive EBM	Generator negatifleri öğrenerek üretir	Canziani / LeCun

16.14 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **Düşük-enerji hacmini sınırla** → Hafta 7 bottleneck + Hafta 8 VAE + bu hafta sparsity (birleşim).
2. **Sparse coding (L1)** → 18.06 seyrek temsil + Stat 110 Laplace prior.
3. **Cost network = enerji** → Hafta 7 EBM + Hafta 1 cross-entropy (gerçek/sahte).
4. **Min-max** → Calculus eyer noktası (saddle point).
5. **GAN = contrastive EBM** → Hafta 8 push-down/push-up (öğrenilen negatif).

İleriye köprüler (production / research):

1. **World model** → LeCun JEPA programı (post-2020), model-based RL/planning.
2. **GAN** → CycleGAN, StyleGAN, WGAN; diffusion (modern üretken).
3. **Sparse / structural** → yorumlanabilir temsil, mixture-of-experts (seyrek aktivasyon).
4. **Öğrenilen negatif** → adversarial eğitim, RLHF reward model.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> EBM serisi tek çatıda kapanır — non-contrastive yöntemler (sparsity, VAE, bottleneck) hep aynı şeyi yapar: düşük-enerji uzayının hacmini sınırlamak; GAN ise contrastive bir EBM'dir, ama negatiflerini elle değil, bir generator'la **öğrenerek** üretir, ve cost network (LeCun'un dilinde "discriminator" değil, enerji ağı) gerçeğe düşük, sahteye yüksek enerji vererek onu hizaya sokar. Dünya modelleri de aynı çerçevenin geleceğe bakan hâlidir — ve LeCun'un bugünkü araştırmasının (JEPA) tohumudur.

17 Görüde Öz-Denetimli Öğrenme (SSL) ve PPUU

İki parçalı hafta — konuk hoca Ishan Misra (Facebook AI Research) görüde öz-denetimli öğrenmeyi (SSL) anlatır: etiketli veri pahalıdır, bu yüzden SSL etiketsiz veriden yapay bir pretext (bahane) görevi uydurur — döndürmeyi tahmin et, göreli konum, renklendirme — ve onu çözerken anlamlı temsil öğrenir; pretext görevin kendisi önemsizdir, değerli olan yan ürünü olan temsildir, sonra bu temsil az etiketli downstream görevlere taşınır. Misra'nın PIRL'i bu fikri olgunlaştırır: pretext'i tahmin etmek yerine dönüşüme değişmez temsil öğren (orijinal ile dönüştürülmüş versiyon benzer, rastgele görüntü farklı — contrastive). Ardından Alfredo Canziani (Practicum) PPUU'yu (belirsizlik altında öngörü ve politika öğrenme) gösterir — meşhur kamyon geri sokma problemini: bir emulator (kamyon kinematiğini öğrenen dünya modeli) ve bir controller (onu süren politika) öğrenilir, önce dünya modeli eğitilir, sonra controller bu öğrenilmiş model içinde planlanır ve maliyet emulator zinciri boyunca backprop ile controller'a yayılır.

Bölüm bilgisi

- **Konuk Lecture videosu (Ishan Misra):** YouTube — Self-supervised learning in computer vision (≈115 dk)
- **Canziani'nin Practicum videosu:** YouTube — Prediction and policy learning under uncertainty (PPUU) (≈60 dk)
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Ishan Misra (konuk, SSL — FAIR) + Alfredo Canziani (Practicum, PPUU)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈25 dk

17.1 Bu Derste Ne Var?

Bu hafta iki parça: **konuk hoca Ishan Misra** (Facebook AI Research) görüde **öz-denetimli öğrenmeyi (SSL)** anlatıyor; **Alfredo Canziani** ise dünya modellerini pratiğe döken **PPUU**'yu (belirsizlik altında öngörü ve politika öğrenme) — meşhur "kamyon geri sokma" problemini — gösteriyor.

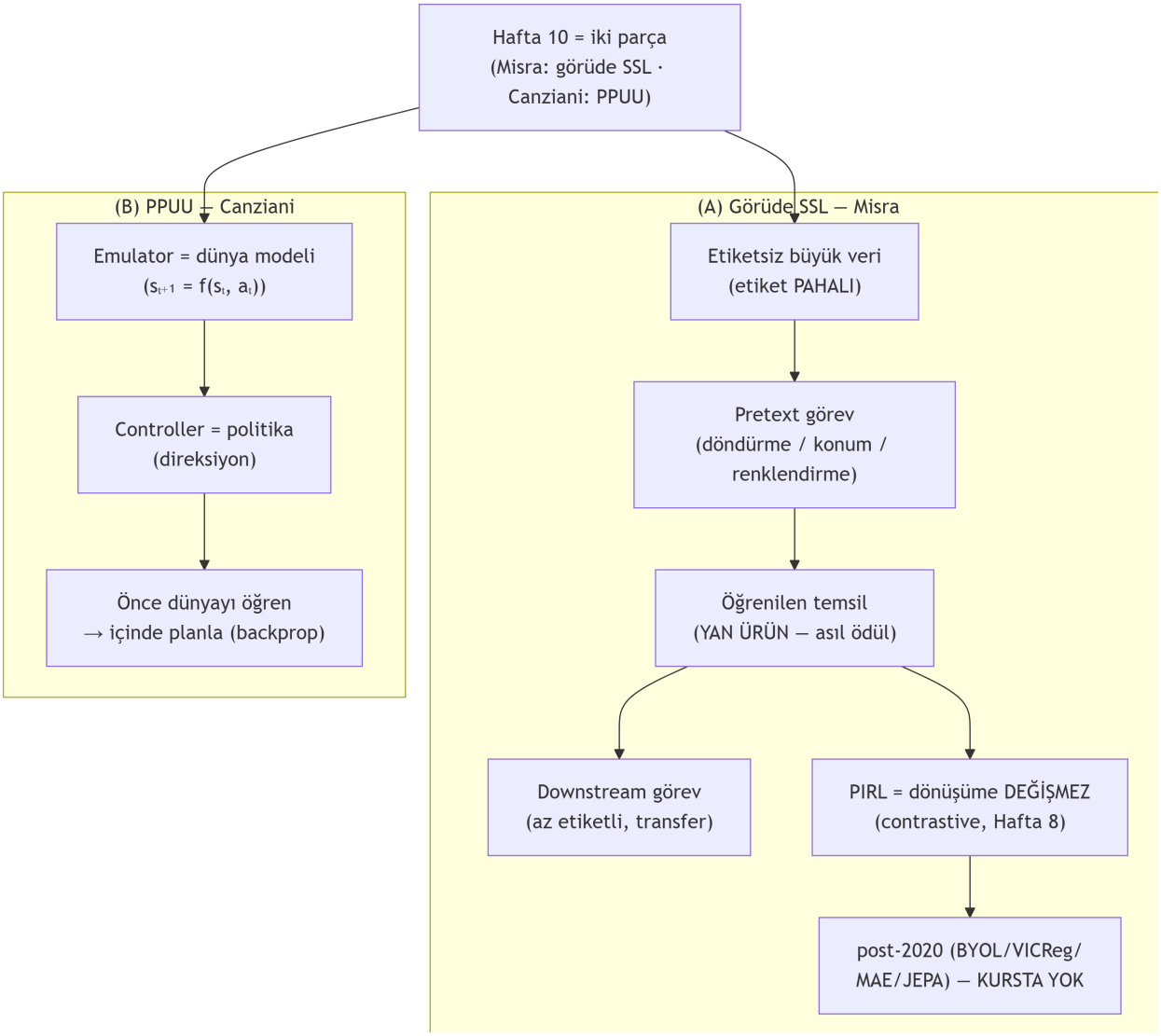
⚠ ⚠ Atıf notu (Hafta 10 sinyatürü)

Bu haftanın Lecture'ı **LeCun değil, konuk Ishan Misra** tarafından verilir (transkriptten doğrulandı: "today we have a guest lecturer, Ishan Misra, research scientist at Facebook AI Research"). Quote'lar **— Misra** ile işaretlenir; Practicum quote'ları **— Canziani**.

Misra'nın büyük fikri: etiketli veri pahalıdır; SSL, **etiketsiz** veriden bir **pretext (bahane) görevi** uydurarak temsil öğrenir, sonra bu temsili gerçek (downstream) görevlere taşır. Canziani ise Hafta 9'un dünya modelini somutlaştırır: bir **emulator** (kamyon kinematiğini öğrenen dünya modeli) + bir **controller** (onu süren politika).

Bu haftanın üç ana fikri:

1. **SSL = pretext görev → temsil → transfer.** Etiketsiz veriden yapay bir görev (döndürmeyi tahmin et, renklendirme, parça konumu) uydur; çözerken anlamlı temsil öğren; downstream'e aktar.
2. **Pretext görev kendisi önemsizdir**; amaç onun yan ürünü olan **temsildir** (representation).
3. **PPUU:** dünya modeli (emulator) + politika (controller); önce dünyayı öğren, sonra onun içinde planla — Hafta 9 world model'in pratiği.



Builder Notu — İki Parça, Tek Çatı: Etiketsiz Gözetim

Geriye (önkoşul kurslar):

- **SSL = label'sız gözetim** → Hafta 8 contrastive SSL + Hafta 3 invariance (augmentation).
- **Pretext = proxy görev** → Hafta 1 supervised (ama label'ı veriden uydur) + Hafta 2 cross-entropy (rotation = 4-yönlü sınıflandırma).
- **Emulator/controller** → Hafta 9 dünya modeli (latent-variable EBM) + Hafta 6 dizi/öngörü.

İleriye (production / research):

- SSL → **post-2020 (BYOL, VICReg, MAE, JEPA — KURSTA YOK)** (Bölüm 5 İleriye Köprü).
- PPUU/world model → model-based RL, planning, LeCun JEPA programı.

Tek cümleyle: Öz-denetimli öğrenme, etiketsiz veriden bir pretext görev uydurup onu çözerek anlamlı temsiller öğrenir ve bunları downstream görevlere taşır; PPUU ise bir dünya modeli (emulator) öğrenip onun içinde bir politika (controller) eğiterek planlamayı pratiğe döker.

17.2 (Misra — Konuk) SSL Nedir? Pre-train → Downstream

Misra önce supervised learning'in sorununu koyuyor: büyük etiketli veri (ImageNet) pahalıdır ve her görev için yeniden toplanamaz. Çözüm **temsil öğrenme (representation learning)**: bir kez iyi bir temsil öğren, sonra onu etiketin az olduğu **downstream (alt) görevlere** taşı.

> "the story for representation learning for computer vision so far has been… a pre-training step [then] use [it] for downstream tasks where you may not have [labels]." — Misra, 1:18

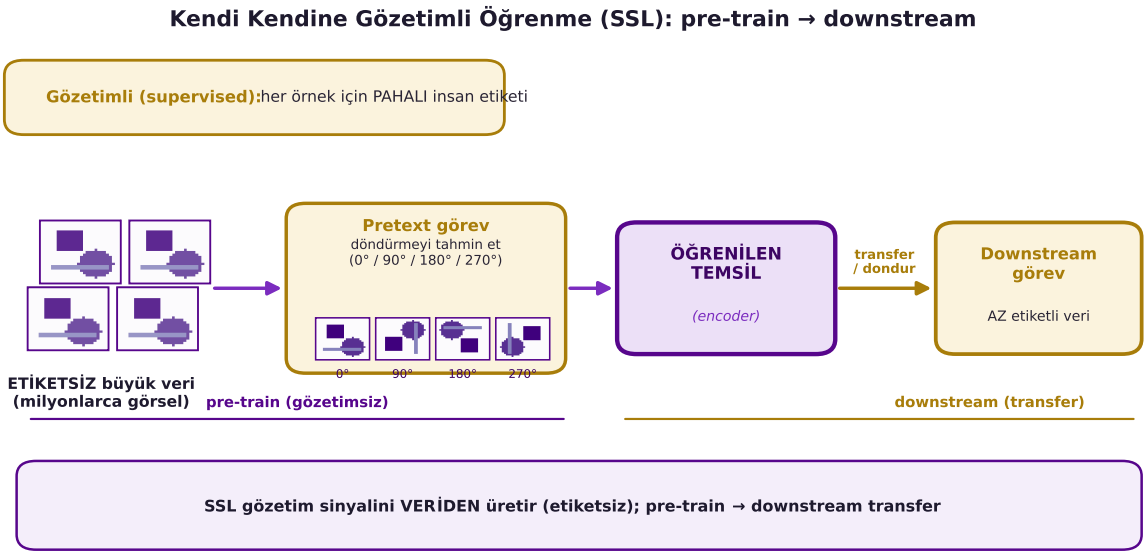
Supervised pre-training (ImageNet etiketleriyle) işe yarar ama yine etiket ister. **Öz-denetimli öğrenme (SSL)** etiketi ortadan kaldırır: gözetim sinyalini **verinin kendisinden** üretir. (NLP'de word2vec bunu yıllardır yapıyor — kelimeyi bağlamından tahmin et.) Şekil 17.1 bu boru hattını gösteriyor: etiketsiz büyük veri bir pretext göreve girer, encoder bir temsil öğrenir, ve bu temsil az etiketli downstream göreve aktarılır — üstteki gözetimli şerit ise her örneğin pahalı insan etiketi gerektiren kıyasıdır.

Builder Notu — Pre-train → Downstream

Geriye (Hafta 8 + 1): SSL, Hafta 8'in contrastive fikrinin geniş çerçevesi; pre-train→downstream, Hafta 1'in "temsil öğren" temasının pratik hattı. Etiketsiz gözetim = label'ı veriden uydurmak.
İleriye: Pre-train → fine-tune paradigması, tüm foundation modellerin (görü ve dil) çalışma biçimidir; SSL onu etiketsiz veriye açar.

17.3 (Misra — Konuk) Pretext Görev: Proxy ile Temsil Öğrenmek

SSL'in kalbi **pretext (bahane) görevidir**: gerçek hedef olmayan, ama çözerken iyi bir temsil öğrenmeni sağlayan yapay bir tahmin görevi.



Şekil 17.1: SSL boru hattı: etiketsiz büyük veri → pretext görev (döndürmeyi tahmin et, 0°/90°/180°/270°) → öğrenilen temsil (encoder) → transfer/dondur → az etiketli downstream görev. Üstte gözetimli (pahalı insan etiketi) kıyası; altta SSL'in gözetim sinyalini doğrudan veriden ürettiği vurgusu.

> "a pretext task is a prediction task that you solve… to learn a representation, and then you finally get to your downstream task where you want to use this representation to perform something meaningful." — Misra, 13:18

Kritik nüans: pretext görevinin kendisi **önemsizdir** (kimse "görüntünün döndürme açısı" gerçekten umursamaz). Önemli olan, onu çözmek için ağın öğrenmek **zorunda kaldığı** temsildir — yan ürün.

> "we are not really interested in predicting the rotations… we are just using this task as a proxy to learn some features." — Misra, 19:11

> **Builder Notu — Pretext = Bahane**
>
> **Geriye (Hafta 1-2):** Pretext görev = supervised öğrenme (Hafta 1), ama etiket veriden otomatik üretilir; rotation tahmini = 4-yönlü cross-entropy (Hafta 2). "Yan ürün temsil" = Hafta 1 representation learning. **İleriye:** "Proxy görevle temsil öğren" fikri, LLM'lerin next-token tahmininin (proxy) dev temsiller öğrenmesinin (yan ürün) ta kendisidir.

17.4 (Misra — Konuk) Pretext Örnekleri: Konum, Döndürme, Renklendirme

Misra yaratıcı pretext görev örnekleri veriyor:

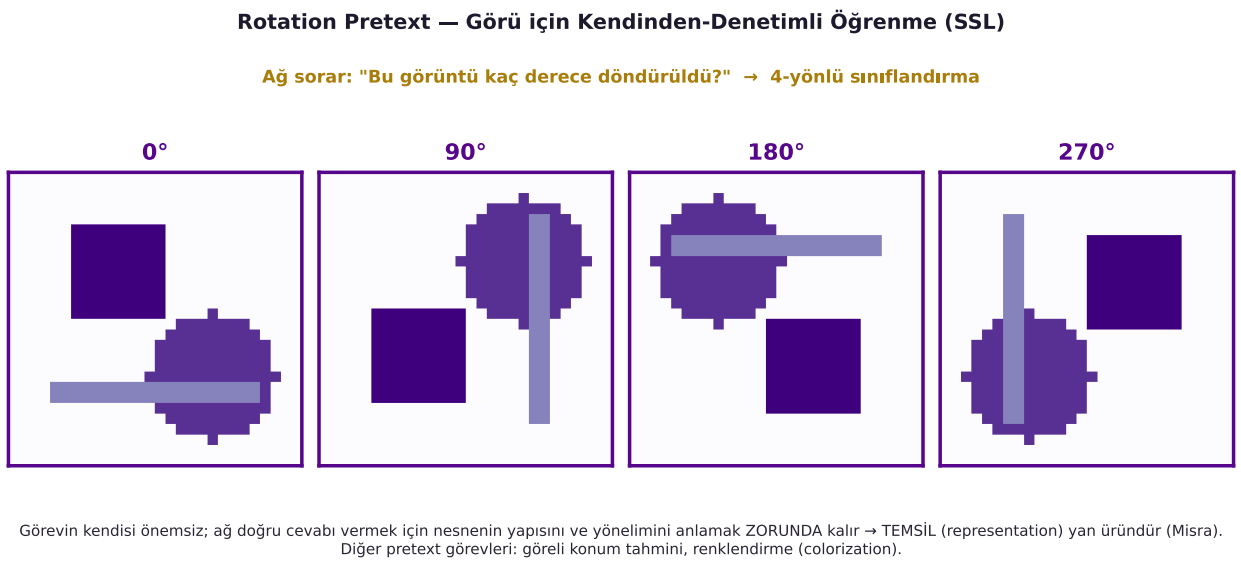
- **Göreli konum (relative position):** bir görüntüden iki yama al, ağa "ikincisi birinciye göre nerede?" diye sor. Bunu çözmek için ağ nesneleri anlamak zorunda kalır. (Nearest-neighbor görselleştirmesi, öğrenilen temsilin **anlamlı/semantik** olduğunu gösterir.)

- **Döndürme tahmini (rotation):** görüntüyü 0/90/180/270° döndür, ağa "kaç derece döndürdüm?" diye sor (4-yönlü sınıflandırma). En popüler pretext görevlerinden — uygulaması çok kolay.

  "you apply a rotation of 0, 90, 180, or 270 degrees… and you ask the network to predict what was the exact rotation applied — a four-way classification problem." — Misra, 18:13

- **Renklendirme (colorization):** gri görüntüden renkli hâlini tahmin et.

Hepsinin ortak mantığı: görevi çözmek "dünyayı anlamayı" gerektirir; bir nesneyi döndürdüğünü bilmek için nesnenin sınırlarını/yapısını kavraman gerekir. Misra'nın PIRL'i (Hafta 3, Hafta 8'de anılan) bu fikri contrastive bir çerçeveye taşır. Şekil 17.2 döndürme pretext'ini gerçek bir sentetik görüntü üzerinde gösteriyor: aynı görüntünün 0°/90°/180°/270° versiyonları, ve ağın çözdüğü 4-yönlü sınıflandırma sorusu.



Şekil 17.2: Rotation pretext görevi (Hafta 10 görü SSL): aynı sentetik görüntünün 0°/90°/180°/270° döndürülmüş dört versiyonu. Ağ "kaç derece döndürüldü?" sorusunu 4-yönlü sınıflandırma olarak çözer. Görevin kendisi önemsizdir; ağ doğru cevabı verebilmek için nesnenin yapısını ve yönelimini anlamak zorunda kalır — yararlı temsil (representation) bu zorunluluğun yan ürünüdür (Misra). Diğer pretext görevleri: göreli konum tahmini ve renklendirme.

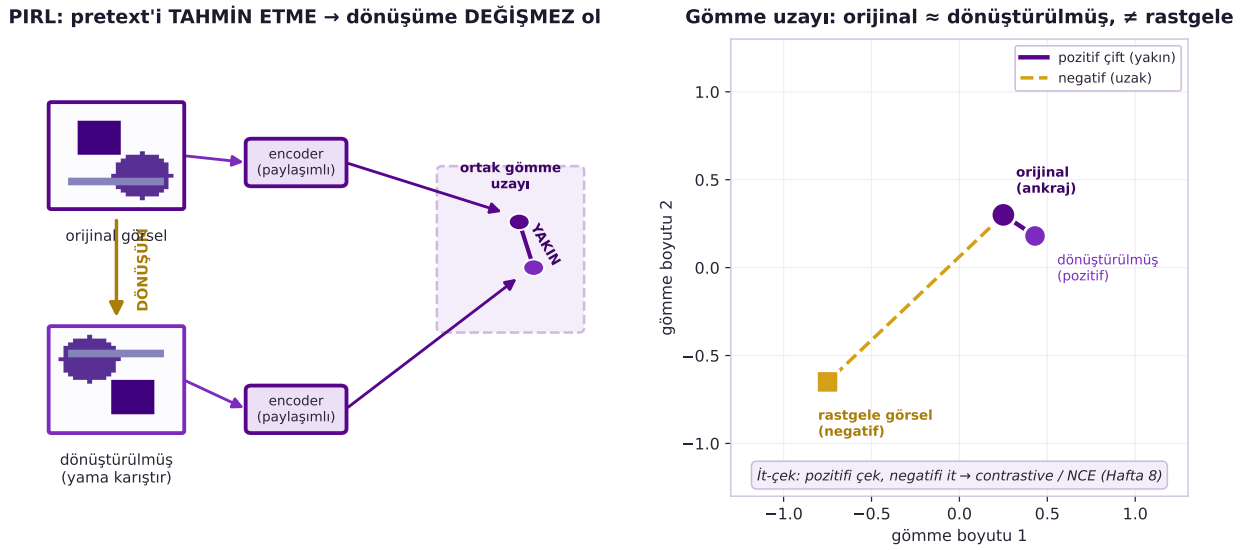
> Builder Notu — Döndürme = 4-Yönlü
>
> **Geriye (Hafta 3 + 8):** Rotation/augmentation = Hafta 3 invariance/stationarity; PIRL = Hafta 8 contrastive (noise contrastive estimation). Pretext = etiketsiz proxy.
> **İleriye:** Bu elle-tasarlanan pretext görevler, post-2020'de yerini daha ilkesel yöntemlere bıraktı (Bölüm 5) — ama "veriden gözetim uydur" fikri kaldı.

17.5 (Misra — Konuk) PIRL ve Değişmez Temsil

Erken pretext görevlerin (rotation, jigsaw) bir zaafı vardı: ağ bazen görevi "kestirme" yollarla çözüp anlamsız temsil öğrenebilir. Misra'nın **PIRL** (Pretext-Invariant Representation Learning) yaklaşımı bunu düzeltir:

temsilin, pretext dönüşümüne (örn. yamaların karıştırılması) **değişmez (invariant)** olmasını ister — yani orijinal ile dönüştürülmüş versiyonun temsilleri **benzer** olmalı (Hafta 8 pozitif çift), rastgele başka görüntüden **farklı** (negatif çift). PIRL, contrastive amaç (noise contrastive estimation) kullanır.

Bu, Hafta 8'in contrastive SSL'inin görüye uygulanmış, olgun hâlidir: pretext'i "tahmin et" yerine "değişmez ol" olarak kurar. Şekil 17.3 bu it-çek mantığını gömme uzayında gösteriyor: orijinal ve dönüştürülmüş versiyon yakına çekilir (pozitif), rastgele görüntü uzakta kalır (negatif).



Şekil 17.3: PIRL (Pretext-Invariant Representation Learning): pretext görevini tahmin etmek yerine dönüşüme değişmez temsil öğren. Solda şema — bir görsel ve ona uygulanan dönüşüm (yama karıştırma) paylaşımlı bir encoder'dan geçer; iki temsil ortak gömme uzayında birbirine yakın çekilir (pozitif çift). Sağda gömme uzayı — orijinal (ankraj) ≈ dönüştürülmüş versiyon (pozitif, kısa violet çizgi), ama rastgele başka bir görsel uzakta kalır (negatif, kesik gold çizgi). İt-çek mantığı: pozitifi çek, negatifi it → contrastive / NCE (Hafta 8).

> **Builder Notu — PIRL = Değişmezlik**
>
> **Geriye (Hafta 8 + 3):** PIRL = contrastive SSL (Hafta 8) + invariance (Hafta 3); pozitif çift = dönüşülmüş versiyon, negatif = başka görüntü. NCE amacı = Hafta 8.
> **İleriye:** "Dönüşüme değişmez temsil" fikri, post-2020 SSL'in (SimCLR/BYOL) tam çekirdeğidir (Bölüm 5).

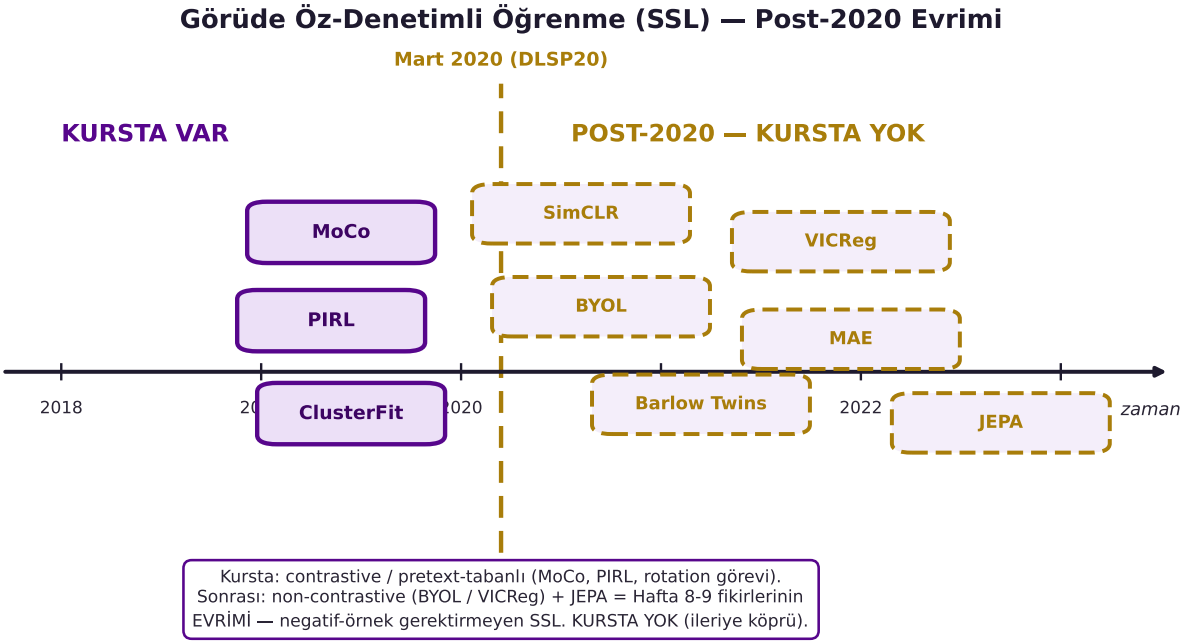
17.6 (İleriye Köprü) SSL'in Post-2020 Evrimi — KURSTA YOK

DLSP20 (Mart 2020), SSL'i **contrastive ve pretext-tabanlı** yöntemlerle (PIRL, MoCo, ClusterFit) anlatır. Bu kurstan **sonra** SSL hızla evrildi — ve aşağıdaki yöntemler bu kursta **YOKTUR** (yalnızca ileriye köprü olarak anılır). Şekil 17.4 bu kırılmayı bir zaman çizelgesinde gösteriyor: dikey çizgi Mart 2020'yi (kursun kayıt tarihi) işaretler, solda kursta işlenen yöntemler, sağda post-2020 yöntemleri.

⚠ İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **SimCLR** (Şub 2020) — basit, büyük-batch contrastive (kursta MoCo var, SimCLR adıyla geçmez)
- **BYOL** (Haz 2020) — **negatif örneksiz** SSL (Hafta 8'in "çok negatif gerekir" sorununu çözer)
- **Barlow Twins** (2021) — çapraz-korelasyon düzenlileştirme
- **VICReg** (2021) — variance-invariance-covariance (LeCun'un "düşük-enerji hacmini sınırla" ilkesinin somut hâli)
- **MAE** (He ve ark., 2021) — masked autoencoder (DAE'nin ViT'teki hâli)
- **JEPA / I-JEPA / V-JEPA** (LeCun grubu, 2022-2024) — joint embedding predictive architecture; LeCun'un dünya-modeli + non-contrastive programının bugünkü zirvesi

Bunlar kurs içeriğine kurs terimi gibi **eklenmez**; Hafta 7-10'da öğrenilen EBM/SSL temelinin nereye evrildiğini göstermek için kullanılır.



Şekil 17.4: Görüde öz-denetimli öğrenmenin (SSL) post-2020 evrimi. Dikey kesikli çizgi kursun kayıt tarihini (Mart 2020, DLSP20) işaretler: solda kursta işlenen contrastive/pretext-tabanlı yöntemler (MoCo, PIRL, ClusterFit), sağda kursta yer almayan post-2020 yöntemler (SimCLR, BYOL, Barlow Twins, VICReg, MAE, JEPA). Non-contrastive (BYOL/VICReg) ve JEPA, Hafta 8-9 fikirlerinin negatif-örnek gerektirmeyen evrimidir — ileriye köprü.

Builder Notu — Post-2020 Evrim

Geriye (Hafta 8-9): Bu evrim, Hafta 8'in "contrastive'in çok-negatif sorunu" + Hafta 9'un "düşük-enerji hacmini sınırla" ilkesinin doğrudan devamıdır — non-contrastive (BYOL/VICReg) tam bu iki fikri birleştirir.
İleriye: JEPA, LeCun'un tüm kurs boyunca tohumladığı fikirlerin (EBM + world model + non-contrastive SSL) bugünkü sentezidir.

17.7 Geçiş: Misra'dan Canziani'ye

Misra etiketsiz veriden temsil öğrenmeyi (SSL) anlattı. Şimdi **Canziani** başka bir "gözetimsiz" problemi gösteriyor: bir sistemin **dünya modelini** ve onu kontrol eden **politikayı** kendi kendine öğrenmesi. Meşhur "kamyon geri sokma" (truck backer-upper) örneğiyle, Hafta 9'un soyut dünya modelini somut, çalışan bir koda döküyor.

17.8 (Canziani) Truck Backer-Upper: Emulator + Controller

Canziani klasik bir öz-öğrenme problemi gösteriyor: bir **römorklu kamyonu** keyfi bir başlangıçtan yükleme rampasına geri sokmak için bir **doğrusal-olmayan kontrolör** öğrenmek (yalnızca geri vites).

> "[we are] trying to design, by self-learning, a nonlinear controller to control the steering of a trailer truck while backing up to a loading dock." — Canziani, 1:53

Durum 6 değişkenle tanımlı (kabin açısı θ_{cab}, römork açısı $\theta_{trailer}$, X, Y, ve hızlar). İki ağ var:

1. **Emulator:** kamyon-römork **kinematiğini** öğrenir — mevcut durum + eylemden (direksiyon) bir sonraki durumu öngörür. Bu bir **dünya modelidir**:

$$s_{t+1} = f(s_t, a_t)$$

2. **Controller:** emulator'ı (yani dünyayı) kontrol eden **politika** — her adımda hangi direksiyonu vereceğini öğrenir.

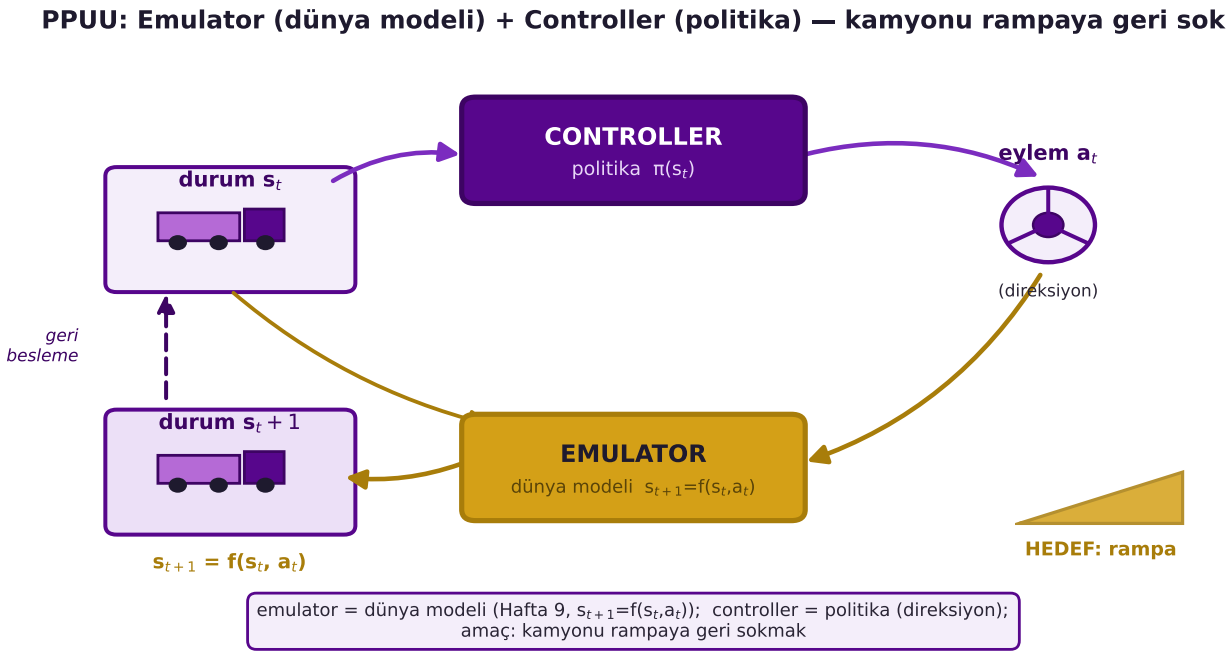
Şekil 17.5 bu döngüsel mimariyi gösteriyor: controller durumdan eylem üretir, emulator durum + eylemden bir sonraki durumu öngörür, geri besleme döngüsü kamyonu rampaya geri sokar.

> Builder Notu — Emulator = Dünya Modeli
>
> **Geriye (Hafta 9 + 6):** Emulator = Hafta 9'un dünya modeli (gelecek durumu öngörür); controller = politika. Durum geçişi $s_{t+1} = f(s_t, a_t)$, Hafta 6 RNN recurrence'ının kontrol hâli.
> **İleriye:** Emulator + controller ayrımı, modern **model-based RL**'in (önce dünya modeli öğren, sonra içinde planla) temel kalıbıdır.

17.9 (Canziani) Önce Dünyayı Öğren, Sonra İçinde Planla

Canziani eğitimin sırasını veriyor: önce **emulator** eğitilir (kamyonu sürerek toplanan verilerle kinematiği öğrenir — bir sonraki durumu doğru öngörsün). Sonra **controller**, *eğitilmiş emulator üzerinden* eğitilir: controller bir direksiyon önerir, emulator sonucu öngörür, ve "rampaya ne kadar yaklaştık?" maliyeti geriye yayılır (backprop) — controller'ı iyileştirir.

> "we have to train a network to be an emulator of the truck and trailer kinematics… [then a] controller to control the emulator." — Canziani, 15:16



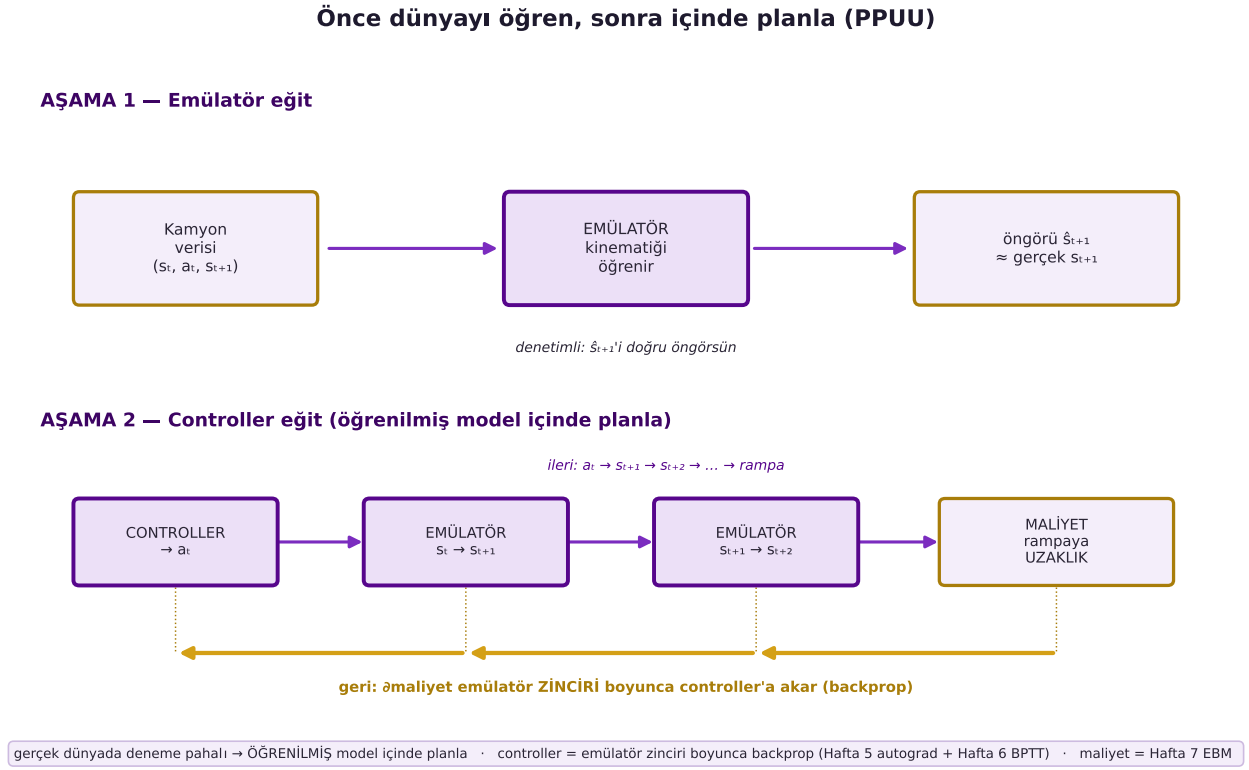
Şekil 17.5: PPUU mimarisi: controller (politika π) durum s_t'den eylem a_t (direksiyon) üretir; emulator (Hafta 9'un dünya modeli) s_t ve a_t'den bir sonraki durumu $s_{t+1}=f(s_t,a_t)$ tahmin eder; geri besleme döngüsü kamyon+römorku rampaya geri sokar.

Bu PPUU'nun (prediction and policy learning under uncertainty) çekirdeğidir: gerçek dünyada deneme-yanılma pahalı/tehlikeli olduğundan, **öğrenilmiş bir dünya modeli içinde** planlarsın. Backprop, controller'ı emulator zinciri boyunca eğitir — Hafta 5'in autograd'ı, Hafta 6'nın zaman-zinciri burada birleşir. Şekil 17.6 bu iki aşamayı gösteriyor: önce emulator kinematiği öğrenir, sonra controller öğrenilmiş emulator zinciri içinde planlar ve maliyet gradyanı zincir boyunca geri akar.

> Builder Notu — Önce Öğren Sonra Planla
>
> **Geriye (Hafta 5-6-9):** Controller eğitimi = emulator zinciri boyunca backprop (Hafta 5 autograd + Hafta 6 BPTT); emulator = dünya modeli (Hafta 9). "Maliyet" = rampaya uzaklık enerjisi (Hafta 7 EBM).
> **İleriye:** "Öğrenilmiş model içinde planla" = model-based RL, MPC (model predictive control), ve LeCun'un dünya-modeli programının pratiği.

17.10 Bu Dersin Özeti

1. **SSL (Misra):** etiketli veri pahalı → etiketsiz veriden temsil öğren, downstream'e taşı (pre-train → fine-tune).
2. **Pretext görev:** yapay bir proxy tahmin görevi (göreli konum, döndürme, renklendirme); kendisi önemsiz, **temsil** yan ürünü değerli.
3. **PIRL:** pretext'i "tahmin et" yerine "dönüşüme değişmez ol" olarak kurar (contrastive, Hafta 8).



Şekil 17.6: PPUU iki aşamalı şeması: önce dünyayı öğren, sonra içinde planla. AŞAMA 1'de emülatör kamyon verisinden kinematiği öğrenir ($\hat{s}_{t+1}$'i doğru öngörür); AŞAMA 2'de controller, öğrenilmiş emülatör zinciri içinde planlar — ileri akış (violet) $a_t \to s_{t+1} \to s_{t+2} \to$ rampaya uzaklık maliyeti üretir, maliyet gradyanı emülatör zinciri boyunca geri (gold backprop) controller'a akar. Hafta 5 autograd + Hafta 6 BPTT + Hafta 7 EBM köprüleri.

4. **Post-2020 SSL (KURSTA YOK):** BYOL/VICReg/MAE/JEPA — Hafta 8-9 fikirlerinin evrimi (İleriye Köprü).
5. **PPUU (Canziani):** emulator (dünya modeli, $s_{t+1} = f(s_t, a_t)$) + controller (politika); truck backer-upper.
6. **Önce dünyayı öğren, sonra içinde planla:** controller, eğitilmiş emulator zinciri boyunca backprop ile eğitilir — model-based RL'in kalbi.

! Tek Bir Cümle

Öz-denetimli öğrenme, etiketsiz veriden bir pretext görev (döndürme, konum, renklendirme) uydurup çözerek anlamlı temsiller öğrenir ve downstream'e taşır (Misra); PPUU ise bir dünya modeli (emulator) öğrenip onun içinde bir politika (controller) eğiterek planlamayı pratiğe döker (Canziani) — ikisi de etiketsiz/gözetimsiz öğrenmenin, Hafta 7-9'da kurulan EBM/world-model temelinin uygulamalarıdır.

17.11 Kontrol Soruları

i Soru 1: SSL supervised learning'den nasıl farklıdır? Pretext görev nedir, neden "önemsiz"dir?

Cevap: Supervised learning insan **etiketi** ister (pahalı); SSL gözetim sinyalini **verinin kendisinden** üretir (etiketsiz). Bir **pretext (bahane) görev**, gerçek hedef olmayan ama çözerken iyi bir temsil öğrenmeni sağlayan yapay bir tahmin görevidir (Misra 13:18) — örn. görüntünün döndürme açısını tahmin et. Görevin kendisi **önemsizdir** (kimse döndürme açısını gerçekten umursamaz); değerli olan, onu çözmek için ağın öğrenmek zorunda kaldığı **temsildir** (yan ürün, Misra 19:11). Sonra bu temsil etiketin az olduğu downstream görevlere aktarılır.

i Soru 2: Üç pretext görev örneği ver. "Döndürme tahmini" neden anlamlı temsil öğretir?

Cevap: (1) **Göreli konum:** iki yamanın birbirine göre konumunu tahmin et; (2) **Döndürme:** görüntüyü 0/90/180/270° döndür, açıyı tahmin et (4-yönlü sınıflandırma, Misra 18:13); (3) **Renklendirme:** gri görüntüyü renklendir. Döndürme anlamlı temsil öğretir çünkü bir nesnenin hangi açıyla döndürüldüğünü bilmek için ağın nesnenin **sınırlarını, yapısını, yönelimini** kavraması gerekir — yani "dünyayı anlamak" zorunda kalır. Bu yapısal anlama, downstream görevlere yarayan temsildir.

i Soru 3: Post-2020 SSL yöntemleri (BYOL, VICReg, MAE, JEPA) bu kursta var mı? Hangi kurs fikirlerinin evrimidir?

Cevap: Hayır, bu kursta YOK — DLSP20 Mart 2020'de çekildi, hepsi sonra geldi (yalnızca ileriye köprü olarak anılır). **BYOL** (Haz 2020) Hafta 8'in "contrastive çok negatif ister" sorununu çözer (negatif-örneksiz SSL); **VICReg** (2021) Hafta 9'un "düşük-enerji hacmini sınırla" ilkesinin somut hâli (variance-invariance-covariance); **MAE** (2021) Hafta 8'in DAE'sinin ViT hâli (masked autoencoder); **JEPA** (2022-24, LeCun) EBM + world model + non-contrastive SSL'in sentezi. Yani hepsi Hafta 7-10'da öğrenilen EBM/SSL temelinin doğal devamıdır.

Soru 4: (Builder) PPUU'da emulator ve controller ne yapar? Controller nasıl eğitilir? Hangi haftalarla bağlantılı?

Cevap: Emulator kamyon-römork kinematiğini öğrenen bir **dünya modelidir**: mevcut durum + eylemden sonraki durumu öngörür:

$$s_{t+1} = f(s_t, a_t)$$

Controller bir **politikadır**: her adımda hangi direksiyonu vereceğini öğrenir. **Eğitim sırası:** önce emulator eğitilir (kinematiği öğrensin); sonra controller, *eğitilmiş emulator üzerinden* eğitilir — controller direksiyon önerir, emulator sonucu öngörür, "rampaya uzaklık" maliyeti emulator zinciri boyunca **backprop** ile controller'a yayılır (Canziani 15:16). Bağlantılar: emulator = Hafta 9 dünya modeli; zincir-backprop = Hafta 5 autograd + Hafta 6 BPTT; maliyet = Hafta 7 EBM. Bu, model-based RL'in kalbidir.

17.12 Egzersizler

Egzersiz 1 (Pretext görev kur). CIFAR-10'da etiketleri **yok say**; her görüntüyü rastgele 0/90/180/270° döndür ve açıyı tahmin eden bir ağ eğit (4-yönlü sınıflandırma). Sonra bu ağın özniteliklerini dondurup üstüne küçük bir sınıflandırıcı eğit — etiketsiz pretext'in downstream'e ne kadar yardım ettiğini ölç.

```
import torch

# Pretext gorevi: ETIKETSIZ goruntuyu dondur, ACIYI tahmin ettir (4-yonlu).
# Donme acisi "bedava etiket" uretir -> oz-denetimli gozetim sinyali.
def make_rotation_batch(images):
    # images: [B, C, H, W] -> her ornek icin rastgele 0/90/180/270 dondur
    B = images.shape[0]
    ks = torch.randint(0, 4, (B,))                  # 4 sinif (bedava etiket)
    rotated = torch.stack([torch.rot90(images[i], int(ks[i]), dims=[-2, -1])
                           for i in range(B)])
    return rotated, ks                              # (girdi, otomatik etiket)

imgs = torch.rand(8, 3, 32, 32)
rot, labels = make_rotation_batch(imgs)
print(f"pretext etiketleri (donme sinifi) = {labels.tolist()}")
# Sonra: encoder'i pretext ile egit -> ozellikleri DONDUR ->
#        ustune kucuk siniflandirici (downstream) -> etiketsiz on-egitimin katkisi
```

Egzersiz 2 (Nearest neighbor). Pretext-eğitilmiş bir ağın öznitelik uzayında, bir sorgu görüntüsüne en yakın komşuları bul. Komşular anlamlı (aynı nesne sınıfı) mı? Bu, temsilin "semantik" olduğunu nasıl gösterir (Misra'nın yöntemi)?

```
import numpy as np

# Ozellik uzayinda en yakin komsu: temsil "semantik" mi?
```

```
# Anlamli temsilde -> en yakin komsular AYNI nesne sinifindan olur.
def nearest_neighbors(query_feat, bank_feats, k=5):
    # kosinus benzerligi (normalize edip ic carpim)
    q = query_feat / (np.linalg.norm(query_feat) + 1e-9)
    B = bank_feats / (np.linalg.norm(bank_feats, axis=1, keepdims=True) + 1e-9)
    sims = B @ q
    return np.argsort(-sims)[:k]                    # en benzer k komsu

rng = np.random.default_rng(0)
bank = rng.normal(0, 1, (100, 64))                  # 100 goruntunun ozellikleri
query = bank[7] + rng.normal(0, 0.05, 64)           # 7'ye yakin bir sorgu
print("en yakin komsular:", nearest_neighbors(query, bank))
# komsular ANLAMLI sinif paylasiyorsa -> temsil semantik (pretext ise yaramis)
```

Egzersiz 3 (PIRL = invariance). Bir görüntü ve onun bir dönüşümü (yama karıştırma) için temsillerin **benzer**, rastgele başka bir görüntününkinden **farklı** olmasını isteyen bir contrastive kayıp kur. Bu, Hafta 8 pozitif/negatif çiftiyle nasıl aynıdır?

```
import torch
import torch.nn.functional as F

# PIRL: orijinal ile DONUSTURULMUS versiyon BENZER (pozitif),
#       rastgele baska goruntu FARKLI (negatif) -> contrastive (Hafta 8).
def pirl_loss(z_orig, z_transformed, z_negatives, tau=0.1):
    z_orig = F.normalize(z_orig, dim=-1)
    z_pos  = F.normalize(z_transformed, dim=-1)
    z_neg  = F.normalize(z_negatives, dim=-1)
    pos = (z_orig * z_pos).sum(-1, keepdim=True) / tau          # pozitif benzerlik
    neg = z_orig @ z_neg.t() / tau                              # negatif benzerlikler
    logits = torch.cat([pos, neg], dim=1)                       # [pozitif | negatifler]
    target = torch.zeros(len(z_orig), dtype=torch.long)         # pozitif = indeks 0
    return F.cross_entropy(logits, target)                      # NCE = Hafta 8

z_o = torch.randn(4, 64); z_t = z_o + 0.1 * torch.randn(4, 64)  # pozitif ~ orijinal
z_n = torch.randn(16, 64)                                       # rastgele negatifler
print(f"PIRL contrastive kayip = {pirl_loss(z_o, z_t, z_n).item():.3f}")
# pozitifi cek, negatifi it -> Hafta 8 pozitif/negatif cifti ile AYNI mantik
```

Egzersiz 4 (Post-2020 ayrımı). Şu yöntemleri "kursta var" / "kursta yok (post-2020)" diye ayır: MoCo, PIRL, BYOL, ClusterFit, VICReg, MAE, JEPA. Her "yok" için hangi kurs fikrinin evrimi olduğunu yaz.

```
# Kursta VAR (DLSP20, Mart 2020) vs YOK (post-2020 ileriye kopru).
kursta_var = ["MoCo", "PIRL", "ClusterFit"]          # contrastive / pretext-tabanli
post_2020  = {
    "BYOL":   "Hafta 8 'cok negatif gerekir' -> negatif-orneksiz SSL",
    "VICReg": "Hafta 9 'dusuk-enerji hacmini sinirla' -> var-inv-cov",
```

```
    "MAE":    "Hafta 8 DAE -> masked autoencoder (ViT)",
    "JEPA":   "EBM + world model + non-contrastive SSL sentezi (LeCun)",
}
for m in ["MoCo", "PIRL", "BYOL", "ClusterFit", "VICReg", "MAE", "JEPA"]:
    if m in kursta_var:
        print(f"{m:12s} -> KURSTA VAR")
    else:
        print(f"{m:12s} -> KURSTA YOK (post-2020): {post_2020[m]}")
# hepsi Hafta 7-10 EBM/SSL temelinin dogal devami (kursta YOK)
```

Egzersiz 5 (Hafta 11 habercisi — aktivasyon/kayıp + PPUU). Hafta 11'de LeCun aktivasyon ve kayıp fonksiyonlarını (EBM kayıpları dahil) sistematik anlatacak; Canziani PPUU'yu sürdürecek. (a) Şimdiye kadar gördüğün kayıpları (cross-entropy, MSE, contrastive, reconstruction) "EBM enerjisini şekillendirme" çerçevesinde nasıl gruplarsın? (b) Bir EBM için "marj kaybı (hinge/margin loss)" neden mantıklıdır (pozitif-negatif enerji farkını zorlar)?

```
import torch
import torch.nn.functional as F

# (a) Kayiplar = "enerjiyi sekillendirme" cercevesinde:
#     cross-entropy/MSE -> dogruya DUSUK enerji (push down);
#     contrastive       -> negatife YUKSEK enerji (push up);
#     reconstruction    -> manifolda DUSUK enerji (off-manifold yuksek).
# (b) Marj kaybi: E(pozitif) ile E(negatif) arasinda en az 'm' fark ZORLA.
def margin_loss(E_pos, E_neg, m=1.0):
    # pozitif enerjisi DUSUK, negatif YUKSEK olsun; aradaki marj >= m
    return F.relu(E_pos - E_neg + m).mean()

E_pos = torch.tensor([0.2, 0.1, 0.3])              # gercek -> DUSUK enerji
E_neg = torch.tensor([1.5, 1.8, 1.2])              # sahte  -> YUKSEK enerji
print(f"marj kaybi = {margin_loss(E_pos, E_neg).item():.3f}")
# marj zaten saglaniyorsa kayip ~0; aksi halde pozitifi bas, negatifi yukselt
# -> Hafta 7 EBM "veride dusuk, disinda yuksek" sekli (Hafta 11'de sistematik)
```

17.13 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H11: Aktivasyon/Kayıp Fonksiyonları ve Öngörülü Politika (PPUU)

Kayıplar EBM çatısına oturuyor. Bu hafta görüde SSL'i (pretext görev → temsil → transfer, PIRL = değişmezlik) ve PPUU'yu (emulator + controller, önce dünyayı öğren sonra içinde planla) kapattık. Hafta 11'de **LeCun** aktivasyon ve kayıp fonksiyonlarını (özellikle **EBM kayıpları**) sistematik ele alacak; Canziani PPUU'yu (belirsizlik altında politika) derinleştirecek. Egzersiz 1 (pretext görev) ve Egzersiz 5 (kayıp = enerji şekillendirme) tam bu derse hazırlar.

Hafta 11: Aktivasyon/Kayıp Fonksiyonları ve Öngörülü Politika (PPUU) — LeCun (Lecture) + Canziani (Practicum)

Hafta 11'de LeCun aktivasyon ve kayıp fonksiyonlarını (özellikle **EBM kayıpları**) sistematik ele alacak; Canziani PPUU'yu (belirsizlik altında politika) derinleştirecek.

Hafta 11 öncesi yapılacak:

- Egzersiz 1 (pretext görev) ve Egzersiz 5 (kayıp = enerji şekillendirme) çöz.
- "Pretext görev = proxy, temsil = yan ürün" ve "emulator = dünya modeli" cümlelerini kendi sözcüklerinle yaz.
- Hafta 7-9 EBM zincirini hatırla — Hafta 11 kayıpları o çerçeveye oturur.

17.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Öz-denetimli öğrenme (SSL)	Etiketsiz veriden gözetim sinyali; pre-train→downstream	Misra 1m18
Pretext görev	Proxy tahmin görevi; temsil öğrenmek için	Misra 13m18
Pretext önemsiz, temsil değerli	Görev bahane; öğrenilen temsil asıl ödül	Misra 19m11
Döndürme pretext'i	0/90/180/270° tahmin (4-yönlü sınıflandırma)	Misra 18m13
Göreli konum / renklendirme	Yama konumu / gri→renk pretext görevleri	Misra 14m16 / 22m19
PIRL	Pretext-değişmez temsil (contrastive, NCE)	Misra (PIRL)
Post-2020 SSL (KURSTA YOK)	BYOL/VICReg/MAE/JEPA — ileriye köprü	İleriye köprü
Emulator (dünya modeli)	$s_{t+1} = f(s_t, a_t)$; kinematiği öğrenir	Canziani 15m16
Controller (politika)	Emulator üzerinden eğitilen kontrolör	Canziani 15m33
PPUU	Önce dünyayı öğren, içinde planla (backprop)	Canziani 1m53

17.15 ML Builder Bağlantıları

Geriye köprüler (önkoşul kurslar):

1. **SSL = etiketsiz gözetim** → Hafta 8 contrastive + Hafta 3 invariance.
2. **Pretext = proxy supervised** → Hafta 1 supervised + Hafta 2 cross-entropy (rotation = 4-yönlü).
3. **PIRL = contrastive invariance** → Hafta 8 (NCE) + Hafta 3.
4. **Emulator = dünya modeli** → Hafta 9 (latent EBM) + Hafta 6 (durum geçişi).

5. **Controller eğitimi = zincir backprop** → Hafta 5 autograd + Hafta 6 BPTT.

İleriye köprüler (production / research):

1. **SSL → BYOL/VICReg/MAE/JEPA (post-2020, KURSTA YOK)**.
2. **Pre-train → fine-tune** → foundation models (görü + dil).
3. **Pretext (proxy)** → LLM next-token (proxy → temsil).
4. **Emulator + controller** → model-based RL, MPC, JEPA planning.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Öz-denetimli öğrenme, pahalı etiketleri atlayıp **etiketsiz veriden bir pretext görev uydurur** (döndürmeyi, konumu, rengi tahmin et) — görevin kendisi önemsizdir, değerli olan yan ürün **temsildir**; ve PPUU, Hafta 9'un dünya modelini somutlaştırır: önce bir emulator ($s_{t+1} = f(s_t, a_t)$) öğren, sonra onun içinde bir controller (politika) eğit. İkisi de, Hafta 7-9'da kurulan EBM/world-model temelinin uygulamalarıdır — ve bu temelin post-2020 evrimi (BYOL, VICReg, MAE, JEPA) kursta yoktur ama LeCun'un bugünkü programının doğrudan devamıdır.

18 Aktivasyon/Kayıp Fonksiyonları ve PPUU

İki parçalı hafta — Yann LeCun (Lecture) aktivasyon ve kayıp fonksiyonlarının sistematik turunu yapar ve kritik olarak kayıpları EBM çerçevesinde (enerji manzarasını şekillendirme) toparlar: bir kayıp fonksiyonu seçmek aslında doğru cevabın enerjisini bastırırken yanlışları bir marjla yukarı iten bir enerji manzarası tasarlamaktır, marj yoksa sistem çöker (her cevaba aynı enerji); aktivasyon tarafında ise tek-kıvrımlı (ReLU türü) fonksiyonlar ölçeğe duyarsız oldukları için derin ağlarda normalleştirmeyle uyumludur, sigmoid/tanh ise gömülü bir ölçek taşır ve doyumda gradyanı yok eder. Ardından Alfredo Canziani (Practicum) Hafta 9-10'un dünya modelini sonuna kadar götürüp PPUU'yu (belirsizlik düzenlileştirmeli öngörülü politika öğrenme) — yoğun trafikte otonom sürüş — baştan sona kurar: deterministik MSE dünya modeli çoklu-geleceği ortalamaya indirip bulanıklaşır, latent değişken eksik bilgiyi taşıyıp keskin çoklu-gelecek üretir, latent dropout eylem-körlüğü sızıntısını keser ve belirsizlik düzenlileştirmesi (varyans cezası) politikayı güvenli eğitim manifoldunda tutar.

> **Bölüm bilgisi**
>
> - **LeCun'un Lecture videosu:** YouTube — Activation and loss functions, EBM (Hafta 11 Lecture)
> - **Canziani'nin Practicum videosu:** YouTube — PPUU (prediction and policy learning under uncertainty)
> - **Edition:** Spring 2020 (NYU-DLSP20)
> - **Hocalar:** Yann LeCun (Lecture, teorik) + Alfredo Canziani (Practicum, PPUU)
> - **Kaynak:** atcold.github.io/NYU-DLSP20
> - **Okuma süresi:** ≈28 dk

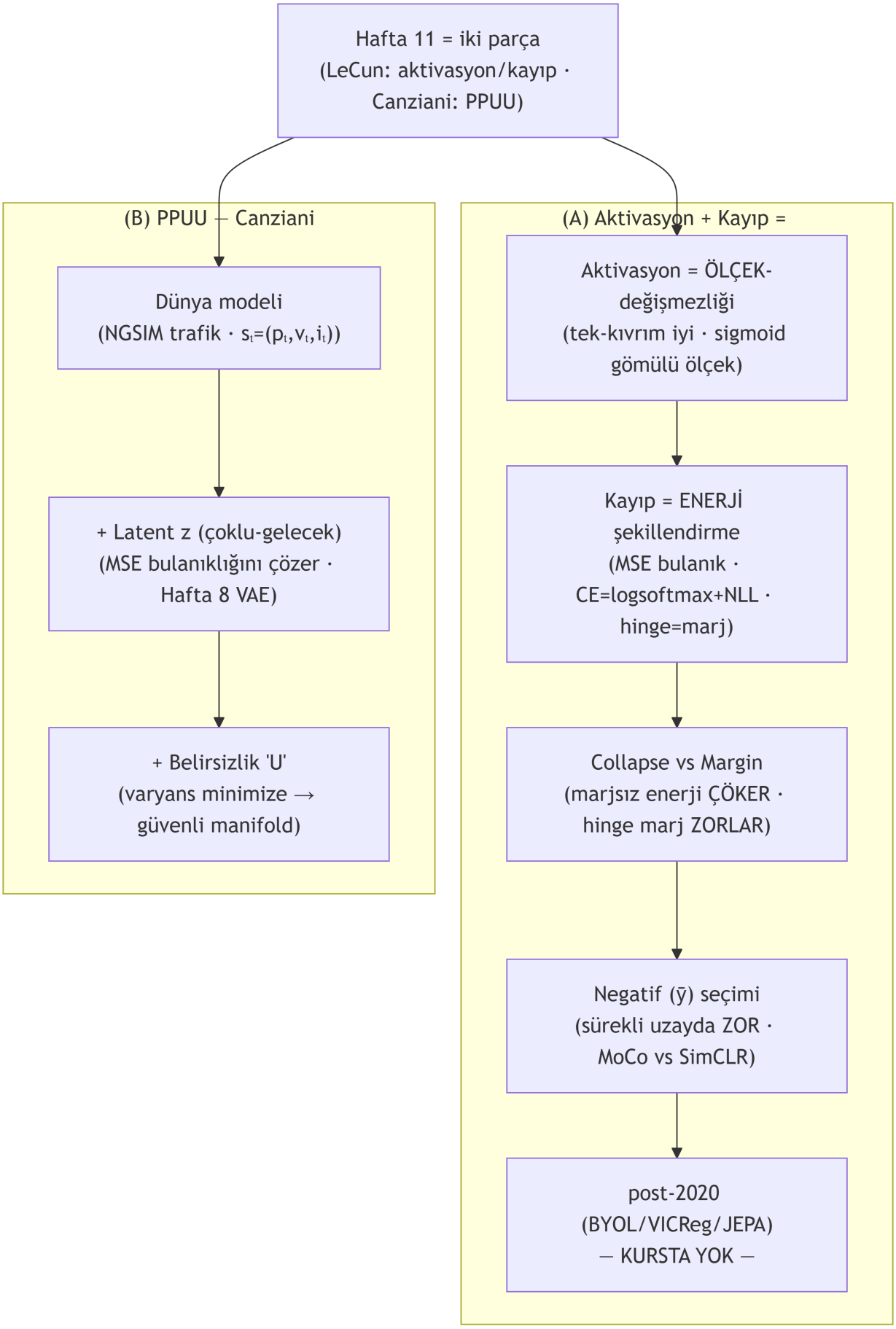
18.1 Bu Derste Ne Var?

Bu hafta iki parça: **Yann LeCun** (Lecture) aktivasyon ve **kayıp fonksiyonlarının** sistematik turunu yapıyor — ve kritik olarak kayıpları **EBM çerçevesinde** (enerjiyi şekillendirme) toparlıyor; **Alfredo Canziani** (Practicum) ise Hafta 9-10'da tanıttığı dünya modelini sonuna kadar götürüp **PPUU**'yu (belirsizlik düzenlileştirmeli öngörülü politika öğrenme) — yoğun trafikte sürüş — baştan sona kuruyor.

LeCun'un büyük fikri: bir kayıp fonksiyonu seçmek, aslında **enerji manzarasını nasıl şekillendireceğini** seçmektir. "İyi" bir kayıp, doğru cevabın enerjisini bastırırken yanlışları bir **marjla** yukarı iter; marj yoksa sistem **çöker** (her cevaba aynı enerjiyi verir). Canziani ise Hafta 7-9'un EBM/latent-değişken fikirlerinin gerçek bir mühendislik probleminde (otonom sürüş) nasıl birleştiğini gösterir.

Bu haftanın üç ana fikri:

1. **Aktivasyon = ölçek-değişmezliği.** Tek-kıvrımlı (ReLU türü) fonksiyonlar derin ağlarda daha iyidir çünkü ölçeğe duyarsızdır; çok-kıvrımlı/yumuşak (sigmoid) fonksiyonlar gömülü bir ölçek taşır ve normalleştirmeyle çatışır.
2. **Kayıp = enerji şekillendirme.** Her kayıp (MSE, L1, cross-entropy, hinge) doğru cevabı aşağı iter; "iyi" olanlar yanlışları **marjla** yukarı da iter — yoksa enerji çöker (Hafta 8-9 collapse problemi).
3. **PPUU:** dünya modeli + politika + **belirsizlik düzenlileştirmesi**; latent değişken çoklu-geleceğin bulanıklığını çözer, varyans cezası politikayı eğitim manifoldunda tutar.


Hafta 11 = iki parça
(LeCun: aktivasyon/kayıp ·
Canziani: PPUU)
(B) PPUU — Canziani
Dünya modeli
(NGSIM trafik · $s_t=(p_t,v_t,i_t)$)
+ Latent z (çoklu-gelecek)
(MSE bulanıklığını çözer ·
Hafta 8 VAE)
+ Belirsizlik 'U'
(varyans minimize →
güvenli manifold)
(A) Aktivasyon + Kayıp =
Aktivasyon = ÖLÇEK-
değişmezliği
(tek-kıvrım iyi · sigmoid
gömülü ölçek)
Kayıp = ENERJİ
şekillendirme
(MSE bulanık ·
CE=logsoftmax+NLL ·
hinge=marj)
Collapse vs Margin
(marjsız enerji ÇÖKER ·
hinge marj ZORLAR)
Negatif (ȳ) seçimi
(sürekli uzayda ZOR ·
MoCo vs SimCLR)
post-2020
(BYOL/VICReg/JEPA)
— KURSTA YOK —

Builder Notu — İki Parça, Tek Çatı: Enerjiyi Doğru Verinin Etrafında Şekillendir

Geriye (önkoşul + kurs):

- **Aktivasyon ölçeği** → Hafta 2-3 ağırlık matrisleri + normalleştirme (BatchNorm).
- **EBM kayıpları** → Hafta 7 (EBM), Hafta 8 (contrastive push-down/up), Hafta 9 (collapse'ı önle).
- **PPUU** → Hafta 9 dünya modeli + Hafta 10 PPUU girişi (emulator/controller); latent+KL = Hafta 8 VAE.

İleriye (production / research):

- Kayıp seçimi = enerji tasarımı → modern temsil öğrenmenin tasarım dili.
- PPUU belirsizlik + dünya modeli → LeCun'un **JEPA** programı (post-2020, Bölüm 4).

Tek cümleyle: Bir kayıp fonksiyonu seçmek, doğru cevabı bastırıp yanlışları marjla iten bir enerji manzarası tasarlamaktır (LeCun); PPUU ise bir dünya modeli öğrenip latent değişkenle çoklu-geleceği, belirsizlik cezasıyla da güvenli sürüşü çözerek bu fikirleri otonom sürüşe döker (Canziani).

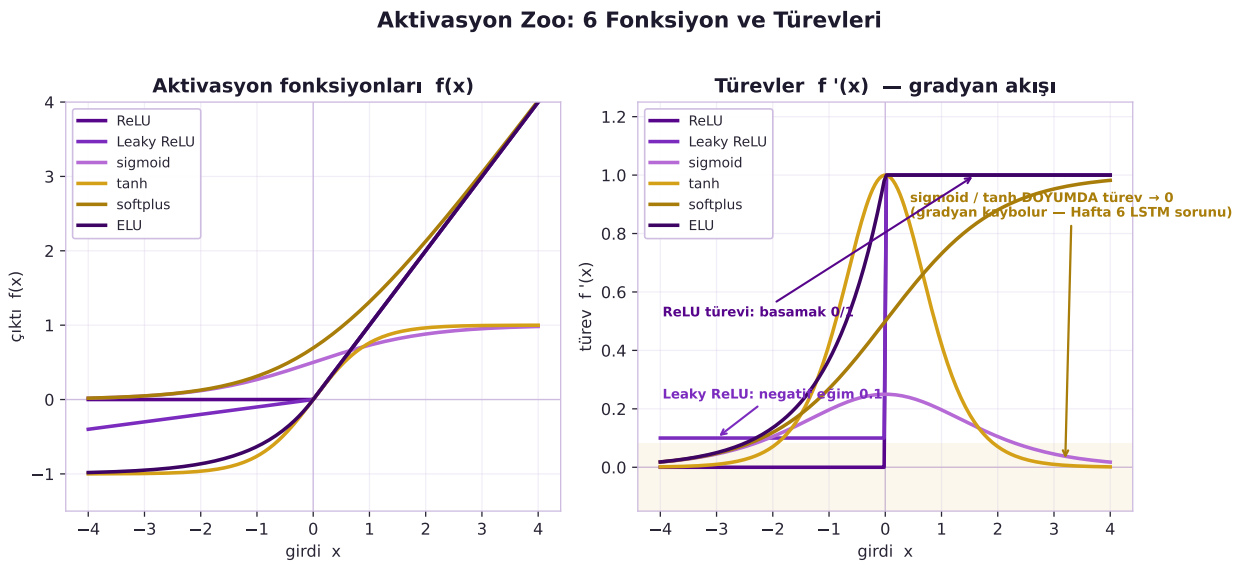
18.2 (LeCun) Aktivasyon Fonksiyonları Zoo'su ve Ölçek-Değişmezliği

LeCun PyTorch'taki aktivasyon fonksiyonlarının "menajerisini" geziyor: ReLU ve varyantları (leaky/PReLU/RReLU — alt kısma negatif eğim vererek **ölü ReLU**'nun gradyan almasını sağlar), softplus (ReLU'nun yumuşak, β-ölçekli hâli), ELU/SELU (negatife inerek çıkışı sıfır-ortalamalı yapar → daha hızlı yakınsama), sigmoid/tanh (doyuma girince **gradyan kaybolur**; tanh sıfır-merkezli olduğu için daha iyi), hardtanh (basit rampa, küçük ağırlıklarla şaşırtıcı iyi çalışır), softshrink/hardshrink (sparse coding'in ISTA adımı — L1 gradyan adımı). Şekil 18.1 bu altı fonksiyonu ve türevlerini yan yana koyar: sağ panel doyum bölgesinde sigmoid/tanh türevinin nasıl sıfıra çöktüğünü (gradyan kaybı = Hafta 6 LSTM sorunu) doğrudan gösterir.

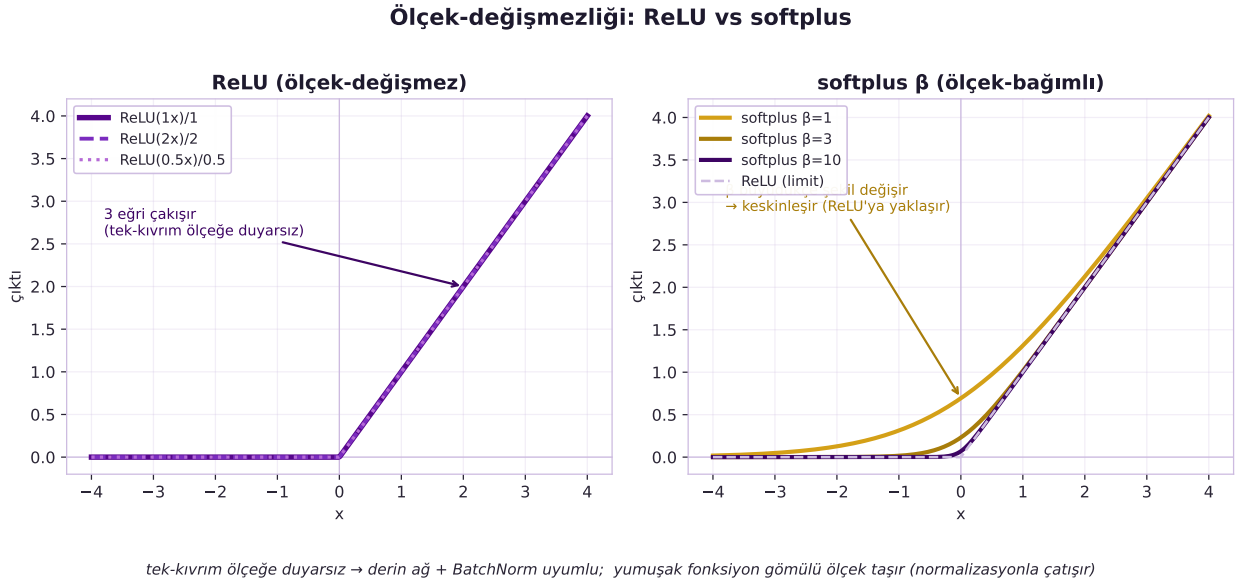
En derin içgörü **ölçek-değişmezliği**: tek keskin kıvrımlı bir fonksiyonda girişi 2 ile çarparsan çıkış da 2 ile çarpılır (yapı değişmez); ama yumuşak/çift-kıvrımlı fonksiyonda **gömülü bir ölçek** vardır — girişi büyütünce fonksiyonun davranışı tamamen değişir.

> "if you have a non-linearity that does care about scale, then your network doesn't have a choice of what size weight matrix it can use in the first layer, because that will completely change the behavior." — LeCun, 27:04

Bu yüzden **tek-kıvrımlı fonksiyonlar derin ağlarda daha iyidir** ve grup/batch normalleştirmeyle **uyumsuzdur**: normalleştirme ölçeği sabitlerse, sigmoid'in hangi bölgesinin kullanılacağı seçimi kaybolur. Şekil 18.2 bu iki rejimi karşılaştırır: ReLU üç ölçek için çakışırken (ölçeğe duyarsız) softplus β büyüdükçe şekil değiştirir (gömülü ölçek). Softmax sıcaklığı β bir **ters sıcaklıktır** (annealing: düşük β yumuşak → yüksek β sert kararlar; mixture-of-experts/attention'da işe yarar).



Şekil 18.1: Aktivasyon zoo (Hafta 11): altı aktivasyon fonksiyonu f(x) ve türevleri f'(x). Sol panel fonksiyonları, sağ panel türevleri (gradyan akışını belirleyen kısım) gösterir. Sigmoid ve tanh doygunluk bölgesinde türevleri sıfıra çöker (gold bant) — geriye yayılımda gradyan kaybolur (Hafta 6 LSTM sorununun kökeni). ReLU türevi 0/1 basamak fonksiyonudur; Leaky ReLU negatif tarafta 0.1 eğimle gradyanı canlı tutar; softplus ve ELU yumuşak geçişlerle ReLU'nun ölü-nöron sorununu hafifletir.



Şekil 18.2: Ölçek-değişmezliği: ReLU vs softplus. SOL — ReLU(c·x)/c üç farklı ölçek (c=1, 2, 0.5) için üst üste çakışır: tek-kıvrımlı, parçalı-lineer fonksiyon ölçeğe duyarsızdır (pozitif homojenlik). SAĞ — β-ölçekli softplus β=1, 3, 10 için şekil değiştirir; β büyüdükçe geçiş bölgesi keskinleşir ve fonksiyon ReLU limitine yaklaşır. Yumuşak aktivasyon gömülü bir ölçek (β) taşır, bu da normalizasyon katmanlarıyla (BatchNorm) çatışabilir; ReLU'nun ölçek-değişmezliği derin ağlarda normalizasyonla uyumludur.

Builder Notu — Aktivasyon = Ölçek

Geriye (Hafta 2-3): Ölçek-değişmezliği = ağırlık matrislerini serbestçe yeniden ölçekleyebilme (lineer katman ikiliği); normalleştirme = Hafta 3 BatchNorm. Doyum/gradyan kaybolması = Hafta 6'da LSTM'in çözdüğü sorun.
İleriye: "Hangi aktivasyon?" sorusunun genel cevabı yoktur (LeCun); ama tek-kıvrım + normalleştirme, derin ağ tasarımının fiilî standardıdır.

18.3 (LeCun) Kayıp Fonksiyonları I: MSE Neden Bulanıklaştırır, Cross-Entropy Neden Birleştirilir

LeCun kayıplara geçiyor ve hemen mühendislik sezgisi veriyor. **MSE (L2)** doğru cevabı ortalamaya çeker; bir girişe karşılık birden çok olası çıkış varsa, sistem hepsinin **ortalamasını** üretir — ve görüntülerin ortalaması **bulanık** bir görüntüdür.

> "the average of a bunch of images is a blurry image, okay, that's why you get those blur effects." — LeCun, 37:50

L1 (mutlak değer) ise medyanı verir (bulanık değil) ve aykırı değerlere **dayanıklıdır**, ama tabanda türevlenemez (softshrink ile çözülür). Huber/SmoothL1 ikisini birleştirir (uzakta L1, yakında L2; Fast R-CNN). Şekil 18.3 bu farkı sayısal olarak gösterir: {1, 2, 2, 10} verisinde MSE minimumu ortalamaya (3,75) düşerken L1 minimumu medyanda (2) kalıp aykırı değeri görmezden gelir — ve aynı "ortalama = bulanık" sezgisi sağ panelde çoklu-gelecek olarak resmedilir.

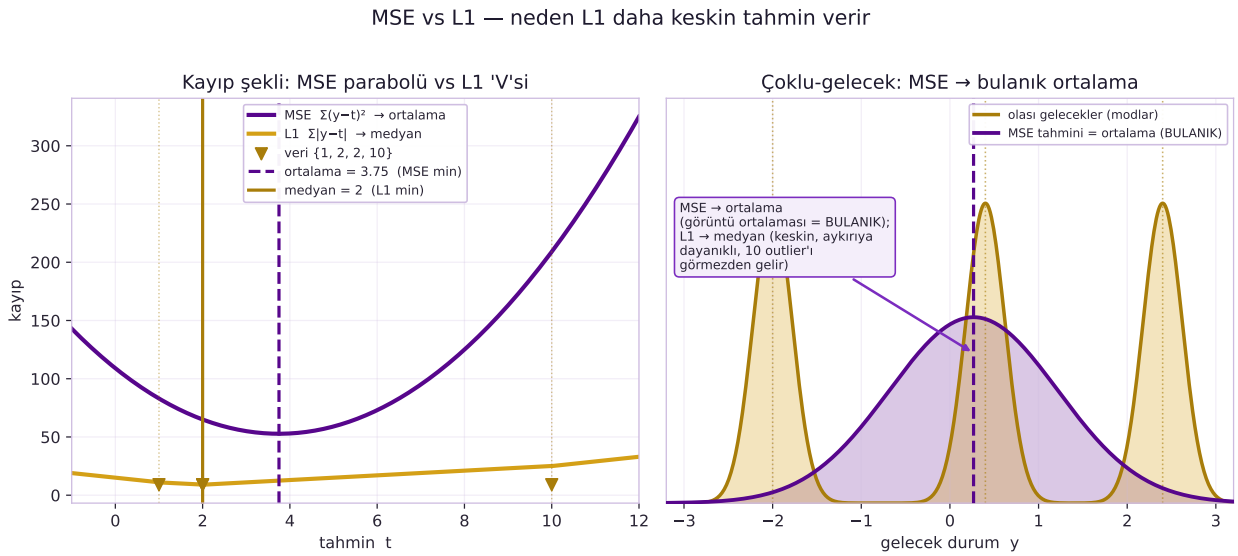
Sınıflandırmada **NLL** doğru sınıfın skorunu büyütür. **Cross-entropy = logsoftmax + NLL** birleşik modülüdür — ve birleştirmenin sebebi **sayısal kararlılıktır**: ayrı ayrı hesaplanırsa ara gradyanlar sonsuza gidip kararsızlık yaratır.

> "you don't want to separate log and softmax, you want to do logsoftmax in one go… it makes the whole thing much more stable numerically." — LeCun, 47:22

Cross-entropy aslında sistemin dağılımı ile one-hot hedef dağılım arasındaki **KL ıraksamasıdır**. (LeCun ayrıca pratik bir uyarı veriyor: dengesiz sınıflarda ağırlık vermek yerine **örnekleme frekansını eşitle** — tıp fakültesi analojisi: nadir hastalıkları da eşit çalış ki öznitelikleri öğrenesin, frekansı en sona düzelt.)

Builder Notu — MSE Bulanıklaştırır

Geriye (Hafta 2): Cross-entropy = Hafta 2'nin temel sınıflandırma kaybı; softmax = enerji yorumuyla (eksi işaret → skorlar enerji olur) Hafta 7 EBM'e köprü. KL = dağılımlar arası fark.
İleriye: MSE bulanıklığı, üretici modellerin (görüntü/video tahmini) merkezî sorunudur — ve bu hafta Canziani'nin latent-değişkenle çözdüğü tam problemdir.



Şekil 18.3: MSE vs L1 kayıpları: neden L1 daha keskin tahmin verir. SOL panel — veri {1, 2, 2, 10} üzerinde MSE kaybı $\Sigma(y-t)^2$ bir parabol çizer ve minimumu ortalamadadır (3,75; violet kesikli çizgi), L1 kaybı $\Sigma|y-t|$ ise bir 'V' çizer ve minimumu medyandadır (2; gold çizgi). L1 minimumu 10 değerindeki aykırı noktayı görmezden gelerek 2'de kalır, MSE ise aykırı nokta tarafından 3,75'e çekilir. SAĞ panel — çoklu-gelecek sezgisi: birbirinden ayrı üç olası gelecek modu (gold) varken MSE tek bir tahmin olarak bunların ortalamasını üretir (violet); bu ortalama hiçbir moda denk gelmez, ortada ve BULANIK kalır. Sezgi: MSE → ortalama (görüntü/gelecek ortalaması = bulanık), L1 → medyan (keskin, aykırıya dayanıklı).

18.4 (LeCun) Kayıp Fonksiyonları II: Margin, Hinge ve EBM Kayıpları

Asıl hafta burada Hafta 7-9 EBM omurgasına bağlanıyor. LeCun kayıpları genel bir **kayıp fonksiyoneli** olarak kuruyor: amaç, doğru cevabın enerjisini küçük, yanlışlarınkini büyük yapmak. Ama nasıl?

Enerji kaybı (sadece doğru cevabı aşağı it) tehlikelidir — yanlışları yukarı itmediği için enerji **her yerde düzleşip çökebilir**:

> "you're just trying to make the energy of the correct answer small, you're not telling the system the energy of everything else should be higher, and so the system might just collapse." — LeCun, 1:20:13

Bu, tam olarak Hafta 8-9'un **collapse** problemidir. **Perceptron kaybı** doğru cevabı aşağı, en-düşük-enerjili cevabı yukarı iter ama **marj** içermez → sistem her cevaba aynı enerjiyi verebilir (yalnız lineer modellerde iyi). Çözüm **marjlı kayıplar** (hinge):

> "as long as your objective function ensures that the energy of the correct answer is smaller than the energy of the most offending incorrect answer by at least a nonzero margin, then your loss function is good." — LeCun, 1:25:47

"En çok suç işleyen yanlış cevap" (most offending incorrect answer,) = yanlış olduğu hâlde en düşük enerjili cevap. Hinge bu farkı bir marja zorlar:

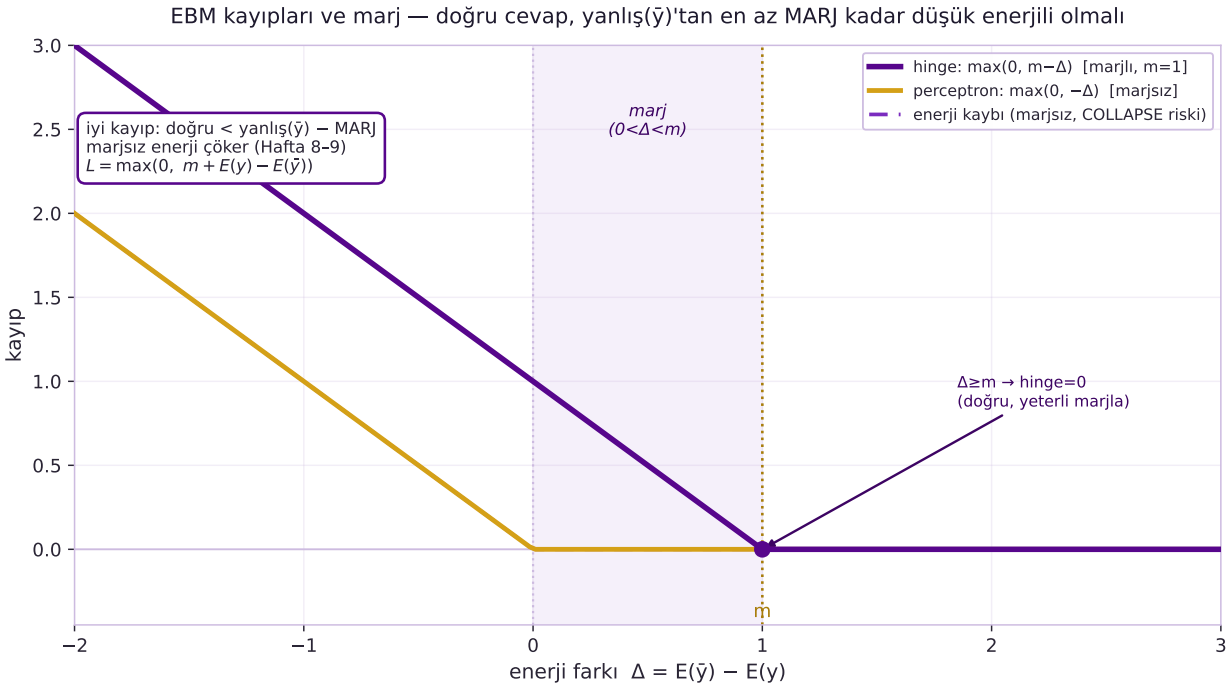
$$L = \max(0,\ m + E(x, y) - E(x, \bar{y}))$$

Şekil 18.4 bu üç kaybı tek bir enerji-farkı ekseninde (Δ = E( ) − E(y)) yan yana koyar: hinge Δ ≥ m olunca sıfırlanır (marj zorunlu), perceptron yalnızca Δ ≥ 0 ister (marjsız), marjsız enerji kaybı ise perceptron'la aynı biçimi paylaşıp **collapse**'a açıktır. Soft-hinge (sonsuz marj, üstel sönüm), square-square (Siamese ağlar, DeepFace yüz tanıma) bu ailenin üyeleridir. **Kritik nüans:** 'yi seçmek sınıflandırmada kolay, ama sürekli/yüksek-boyutlu uzayda zordur — negatif örnekleme problemi:

> "that's why what makes the difference between MoCo [and] SimCLR etc is how you pick those negative samples." — LeCun, 1:36:09

Yani Hafta 8'in contrastive yöntemi ile Hafta 10'un SSL'i, aynı "negatif nasıl seçilir?" sorusunun farklı cevaplarıdır.

> Builder Notu — Collapse vs Hinge
>
> **Geriye (Hafta 7-8-9):** Enerji kaybı collapse'ı = Hafta 8-9 (yanlışları itmezsen enerji düzleşir); margin/hinge = Hafta 8 contrastive push-down/up; seçimi = Hafta 8 NCE + Hafta 10 hard-negative.
> **İleriye:** "Kayıp = enerji şekillendirme + marj" çerçevesi, contrastive/non-contrastive tüm temsil öğrenmenin ortak dilidir.



Şekil 18.4: EBM kayıpları ve marj: doğru cevabın enerjisi, yanlış cevap ’nin enerjisinden en az bir MARJ (m) kadar düşük olmalıdır. Yatay eksen enerji farkı Δ = E( ) − E(y) (pozitif = doğru cevap daha düşük enerjili). Hinge kaybı (violet, max(0, m−Δ)) Δ≥m olunca sıfırlanır — pozitif marj zorunlu kılar. Perceptron kaybı (gold, max(0, −Δ)) yalnızca Δ≥0 ister; marjsızdır. Marjsız enerji kaybı (kesik violet) ile perceptron aynı biçimi paylaşır ve enerji manzarasının düz çökmesine (collapse, Hafta 8–9) açıktır. Gölgeli bölge 0<Δ<m marj aralığını gösterir. Genel EBM kaybı L = max(0, m + E(y) − E( )).

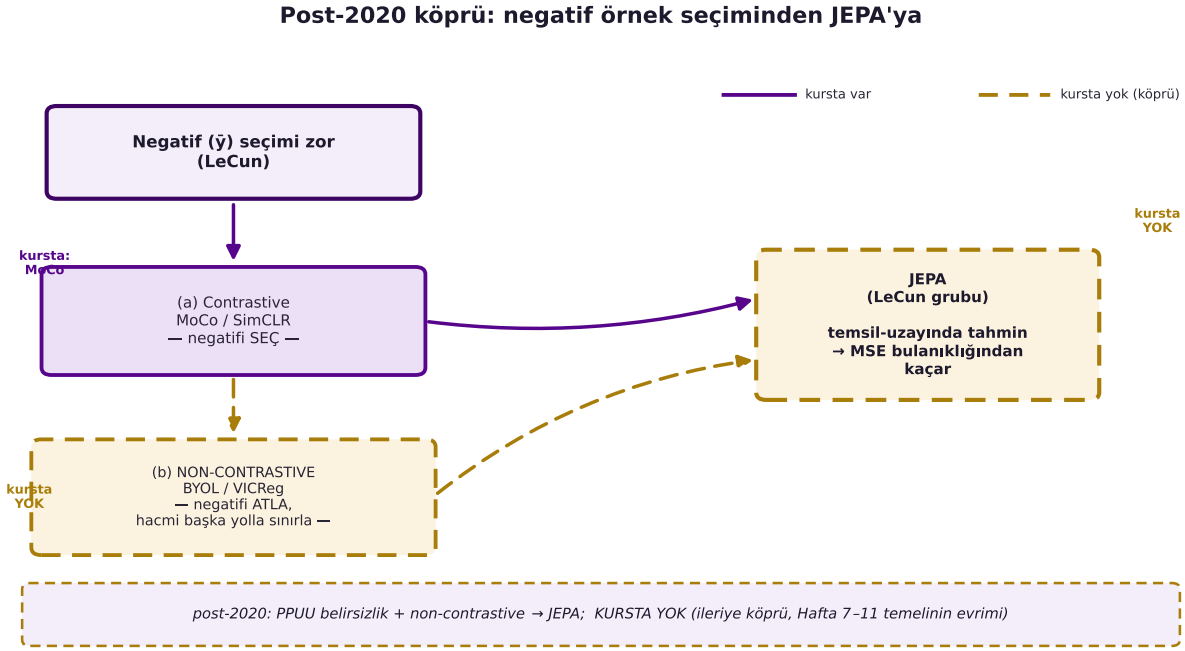
18.5 (İleriye Köprü) Negatif Seçiminden Non-Contrastive'e ve JEPA — KURSTA YOK

LeCun'un " 'yi (negatifi) seçmek zordur" tespiti ve Canziani'nin birazdan göreceğimiz **belirsizlik düzenlileştirmesi**, DLSP20'den **sonra** olgunlaşan bir programa işaret eder. Şekil 18.5 bu evrimi şematize eder: "negatif seçimi zor" kutusundan iki yol (contrastive vs non-contrastive) çıkar ve ikisi de JEPA'da buluşur. Aşağıdakiler bu kursta **YOKTUR** (yalnızca ileriye köprü):

⚠ ⚠ İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **BYOL** (Haz 2020), **VICReg** (2021) — **negatif örneksiz** (non-contrastive) SSL: LeCun'un "negatif seçmek zor" sorununu, negatifleri tamamen atıp enerji hacmini başka yolla sınırlayarak çözer. Canziani'nin varyans/belirsizlik cezası bu fikrin erken bir akrabasıdır.
- **JEPA / I-JEPA / V-JEPA** (LeCun grubu, 2022-2024) — joint-embedding predictive architecture: PPUU'nun "dünya modeli öğren + belirsizlikle planla" fikrinin bugünkü zirvesi; gözlemi piksel yerine **temsil uzayında** öngörür (MSE bulanıklığından tamamen kaçar).

Bunlar kurs terimi gibi eklenmez; Hafta 7-11'de kurulan EBM + dünya-modeli temelinin nereye evrildiğini göstermek için anılır.



Şekil 18.5: Post-2020 köprü: negatif örnek seçiminden JEPA'ya. 'Negatif ( ) seçimi zor (LeCun)' kutusundan iki yol çıkar: (a) Contrastive (MoCo/SimCLR — kursta MoCo, violet düz çizgi) negatifi seçer; (b) NON-CONTRASTIVE (BYOL/VICReg — kursta yok, gold kesikli) negatifi atlar ve hacmi başka yolla sınırlar. Her iki yol da JEPA'ya (LeCun grubu, kursta yok, kesikli) varır: temsil-uzayında tahmin yaparak MSE bulanıklığından kaçar. Bu şema Hafta 7–11 temelinin (EBM, belirsizlik, contrastive/non-contrastive) post-2020 evrimine ileriye köprüdür.

Builder Notu — Negatif Seçimi

Geriye (Hafta 8-9-11): Non-contrastive = Hafta 9 "düşük-enerji hacmini sınırla" + bu haftanın "marjsız enerji kaybı collapse eder" uyarısının çözümü.
İleriye: JEPA, LeCun'un tüm kurs boyunca tohumladığı (EBM + world model + non-contrastive) fikirlerin sentezidir.

18.6 Geçiş: LeCun'dan Canziani'ye

LeCun kayıpların enerji manzarasını nasıl şekillendirdiğini anlattı ve "çoklu-gelecek varsa MSE bulanıklaştırır" tespitini bıraktı. Şimdi **Canziani** tam bu sorunu gerçek bir sistemde — yoğun trafikte otonom sürüş — çözüyor: Hafta 9-10'un dünya modelini latent değişken ve **belirsizlik düzenlileştirmesiyle** tamamlayarak **PPUU**'yu baştan sona kuruyor.

18.7 (Canziani) PPUU: Dünya Modeli, Maliyet ve Model-Free'nin Sorunu

Canziani problemi koyuyor: bir aracı yoğun trafikte sürmeyi öğret. **Model-free RL** kaza yaparak öğrenir — kötü fikir:

> "you have to die a few times before actually learning not to die, but that's arguably not the way you learn how to drive." — Canziani, 1:43

Bunun yerine bir **dünya modeli** öğrenip onun içinde planla (yemek yaparken elini yakmadan önce zihninde dene). **Veri:** NGSIM I-80 otoyolu — bir binanın tepesindeki kameralardan tepeden-görünüm, bounding box + takip. Her araç için durum $s_t = (p_t$ konum, v_t hız, i_t bağlam görüntüsü). Görüntü i_t bir **doluluk ızgarasıdır** (mavi=ben, kırmızı=şerit, yeşil=diğerleri) — değişken sayıda aracı sabit boyutlu temsille kodlamanın "şirin" yolu. Eylemler, **kinematiği tersine çevirerek** kurtarılır (düzgün doğrusal hareketten sapma = eylem).

Maliyet = şerit maliyeti (şeritten çıkma) + yakınlık maliyeti (hıza göre uzayan boylamsal potansiyel × enlemsel potansiyel) — ikisi de **türevlenebilir**, böylece çarpışmayı azaltmak için gradyan akıtılabilir.

Builder Notu — Dünya Modeli + Maliyet

Geriye (Hafta 9-10): Dünya modeli = Hafta 9; emulator/controller ayrımı = Hafta 10 PPUU girişi. Doluluk ızgarası = değişken-uzunluk problemini görüntüyle çözmek (attention'a alternatif).
İleriye: "Önce dünya modeli, sonra içinde planla" = model-based RL ve MPC'nin (model predictive control) çekirdeği.

18.8 (Canziani) MSE Bulanıklığı ve Latent Değişken Çözümü

İlk deneme **deterministik** dünya modeli: predictor (geçmiş → geleceğin gizli temsili) + decoder (gizli → gerçek gelecek), MSE ile eğit. **Başarısız** — tahminler 3-5 saniye sonra bulanıklaşır. Sebep tam olarak LeCun'un dediği: çoklu-gelecek varken MSE **ortalamayı** üretir. Şekil 18.6 iki yolu yan yana koyar: üstte

deterministik (MSE) yol bulanık gelecek, altta +latent z (VAE) yolu keskin çoklu-gelecek üretir. Canziani'nin düşen-kalem örneği:

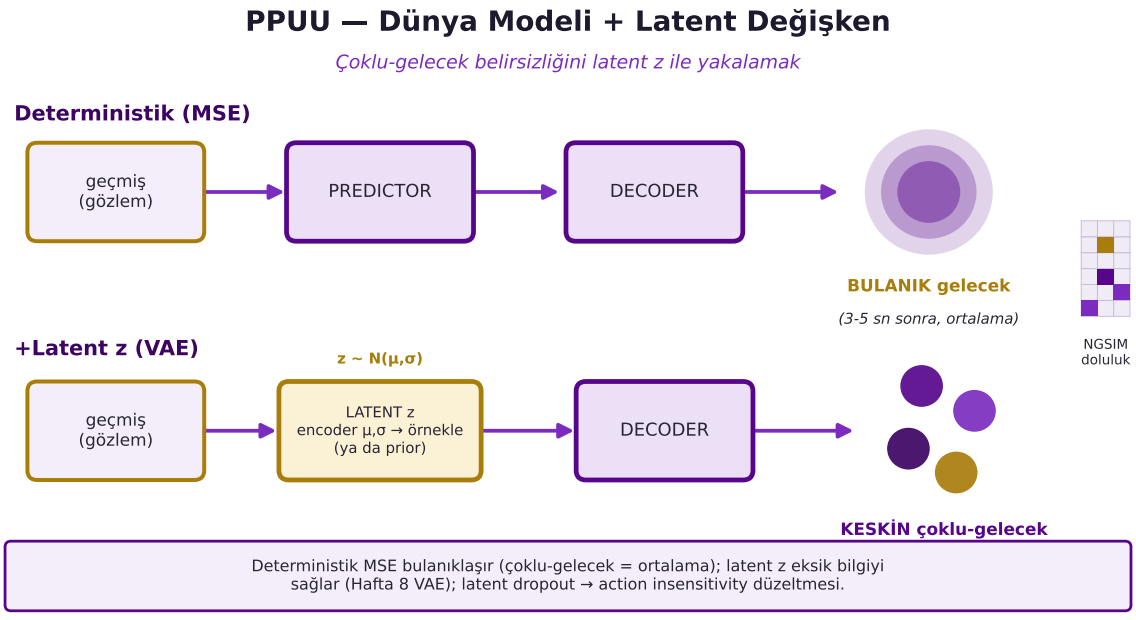
> "the average final location is like the pen never fell, and it's really wrong." — Canziani, 28:59

Çözüm: düşük-boyutlu (16-boyutlu) bir **latent değişken** z_t ekle. z_t, MSE'yi sıfırlayacak şekilde ya **çıkarımla** (latent uzayda gradyan inişi) ya da geleceği gören bir **variational encoder** ile (ortalama+varyans, örnekle) bulunur. **KL terimi** posterior'u N(0, I) prior'a yaklaştırır; böylece test anında prior'dan örnekleyerek gelecek **üretebilirsin** (Hafta 8 VAE = non-contrastive EBM):

> "you add latent variables in order to provide the missing information that would be required for you to make a proper prediction." — Canziani, 1:10:47

İncelikli tuzak — action insensitivity (bilgi sızıntısı): encoder geleceği gördüğü için "döndük" bilgisini latent'e sızdırır → forward model **direksiyonu (eylemi) yok sayar**. Çözüm **latent dropout**: bazı zamanlar z_t'yi encoder yerine prior'dan örnekle, böylece dönme latent'e kodlanamaz ve ağ eylemi kullanmak **zorunda** kalır.

> "we fix this problem by simply dropping out this latent and sampling from the prior… in this way you can't encode the rotation anymore in the latent variable." — Canziani, 52:52



Şekil 18.6: PPUU dünya modeli ve latent değişken: üstte deterministik (MSE) yol geçmiş → PREDICTOR → DECODER zinciriyle 3-5 saniye sonrası için bulanık (ortalama) gelecek üretir; altta +Latent z (VAE) yolu encoder μ,σ'dan örneklenen (ya da prior'dan gelen) z'yi DECODER'a enjekte ederek keskin, çoklu-gelecek tahmini sağlar. Sağda NGSIM doluluk ızgarası ikonu (violet=ben, gold=şerit, orta-violet=diğer araçlar). Deterministik MSE çoklu-gelecek belirsizliğini ortalamaya indirip bulanıklaşırken latent z eksik bilgiyi taşır (Hafta 8 VAE bağı) ve latent dropout action insensitivity sorununu düzeltir.

Builder Notu — Latent = Eksik Bilgi

Geriye (Hafta 8 + 2): MSE-ortalama = bu haftanın LeCun bulanıklık tespiti; latent+KL = Hafta 8 VAE; "missing info" latent = Hafta 8 gizli değişken yorumu. Latent dropout = bilgi sızıntısını kesen düzenlileştirme.
İleriye: Çoklu-geleceği latent'le modelleme = koşullu üretici modellerin (CVAE, diffusion) temel kalıbı; piksel-MSE'den kaçış → JEPA (temsil uzayında tahmin).

18.9 (Canziani) Belirsizlik Düzenlileştirmesi — PPUU'nun "U"su

Politikayı eğitmek: politika(durum) → eylem → dünya modeli → tahmin → maliyet (yakınlık+şerit), zinciri açıp **backprop** ile politikayı eğit (dünya modeli donuk). **Başarısız** — politika hile yapıp **manifold dışına** çıkar (her şeyi siyah/sıfır-maliyet tahmin ettirir → yola çıkar, çarpışır).

Birinci yama **uzman düzenleyici** (taklit öğrenme — politikayı uzman eylemine yaklaştır) işe yarar. Asıl fikir ikinci yama, **belirsizlik düzenlileştirmesi** ("U"): eğitim bölgesi dışında dünya modelinin **varyansı** artar; varyans türevlenebilir, öyleyse onu minimize et — politika güvenli, eğitim manifolduna yakın eylemler seçsin. Şekil 18.7 bu varyans-proxy'sini gösterir: eğitim noktalarına yakın çukurlaşır (~0), uzakta doygunluğa (~1) ulaşır; yeşil bant düşük-belirsizlik (güvenli) bölgesini işaretler.

> "as you go away from the training interval the variance will increase… your variance now is your loss, you do gradient descent in action space for variance minimization." — Canziani, 1:18:00

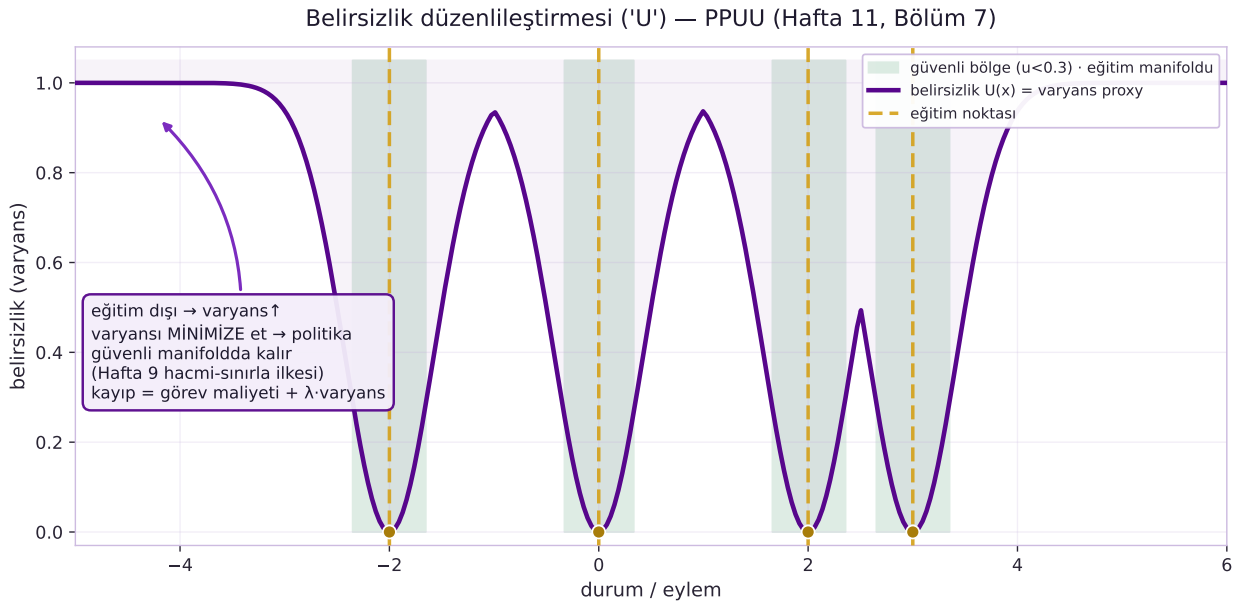
Birden çok tahmin için çıkarımda **dropout açık** bırakılır (varyansı ölçmek için). Nihai kayıp = görev maliyeti + λ·belirsizlik. Sonuç: araç yoğun trafikte hayatta kalır. Bu, Hafta 9'un "düşük-enerji hacmini sınırla" non-contrastive ilkesinin somut bir mühendislik hâlidir — varyans cezası, enerjiyi eğitim verisinin etrafında tutan bir düzenleyicidir.

Builder Notu — Belirsizlik 'U'su

Geriye (Hafta 5-6-9): Zincir-backprop = Hafta 5 autograd + Hafta 6 BPTT; belirsizlik cezası = Hafta 9 non-contrastive "hacmi sınırla"; manifold-dışı çökme = Hafta 9 collapse'ın politika hâli.
İleriye: Belirsizlik-bilinçli planlama = model-based RL'in güvenlik anahtarı; epistemik belirsizlik (ensemble/dropout varyansı) tüm risk-duyarlı kontrolün temeli.

18.10 Bu Dersin Özeti

1. **Aktivasyon = ölçek-değişmezliği (LeCun):** tek-kıvrımlı (ReLU) fonksiyonlar derin ağlarda daha iyi, normalleştirmeyle uyumlu; sigmoid/tanh gömülü ölçek taşır + doyumda gradyan kaybolur.
2. **MSE → bulanık (LeCun/Canziani):** çoklu-gelecekte MSE ortalama üretir (görüntü ortalaması = bulanık); L1 → medyan (keskin, dayanıklı).
3. **Cross-entropy = logsoftmax + NLL:** sayısal kararlılık için birleşik; KL ıraksaması (sistem dağılımı vs one-hot).
4. **Kayıp = enerji şekillendirme + marj (LeCun):** enerji kaybı marjsız → collapse; hinge/margin doğru cevabı yanlıştan marjla ayırır → iyi; (negatif) seçimi sürekli uzayda zor (MoCo vs SimCLR).



Şekil 18.7: Belirsizlik düzenlileştirmesi (‘U’ — PPUU, Hafta 11): epistemik varyans proxy’si eğitim noktalarına (gold dikey işaretler, x=-2,0,2,3) yakın çukurlaşır (~0), eğitim manifoldundan uzaklaştıkça doygunluğa (~1) ulaşır. Yeşil gölgeli güvenli bölge (u<0.3) eğitim manifoldunu işaretler. PPUU’da kayıp = görev maliyeti + λ·varyans olarak kurulur; varyansı minimize etmek politikayı düşük-belirsizlik (gözlenmiş) bölgesinde tutar — Hafta 9’un ‘hacmi sınırla’ EBM ilkesinin kontrol karşılığı.

5. **PPUU (Canziani):** dünya modeli + latent değişken (çoklu-gelecek) + KL + latent dropout (action insensitivity) + **belirsizlik düzenlileştirmesi** (varyans minimizasyonu = güvenli manifold).
6. **Post-2020 (KURSTA YOK):** non-contrastive (BYOL/VICReg) = negatif seçimini atlar; JEPA = PPUU + temsil-uzayı tahmini.

! Tek Bir Cümle

18.11 Kontrol Soruları

i Soru 1: Neden tek-kıvrımlı (ReLU) fonksiyonlar derin ağlarda sigmoid'den daha iyidir? "Ölçek-değişmezliği" ne demek?

Cevap: Tek keskin kıvrımlı bir fonksiyon **ölçeğe duyarsızdır**: girişi 2 ile çarparsan çıkış da 2 ile çarpılır, davranış (kıvrımın yeri) değişmez. Sigmoid/softplus gibi yumuşak fonksiyonlarda **gömülü bir ölçek** vardır — girişi büyütünce fonksiyonun "sert mi yumuşak mı" davranışı değişir. Derin ağda ardışık katmanların ağırlık ölçeği serbestçe yeniden dağıtılabildiği için (LeCun 27:04), ölçeğe duyarlı bir aktivasyon ağı kısıtlar ve doyuma sokup gradyan kaybına yol açar. Ayrıca sigmoid, batch/grup normalleştirmeyle **uyumsuzdur**: normalleştirme ölçeği sabitlerse, sigmoid'in hangi bölgesinin kullanılacağı seçimi kaybolur.

i Soru 2: MSE neden bulanık tahminler üretir? L1 farkı nedir? Bu Canziani'nin PPUU'sundaki hangi sorunla aynıdır?

Cevap: Bir girişe karşılık birden çok olası çıkış varsa, MSE'yi minimize eden değer hepsinin **ortalamasıdır**; görüntülerin ortalaması bulanıktır (LeCun 37:50). Düşen-kalem örneği: kalemin düşeceği yönlerin ortalaması "kalem hiç düşmedi" konumudur — yanlış (Canziani 28:59). L1 ise **medyanı** verir (bulanık değil) ve aykırı değerlere dayanıklıdır, ama tabanda türevlenemez. PPUU'da deterministik dünya modeli tam bu yüzden 3-5 saniye sonra bulanıklaşır; çözüm, çoklu-geleceği yakalayan bir **latent değişkendir**.

i Soru 3: "Enerji kaybı" neden tehlikelidir? "İyi" bir EBM kaybını ne yapar? "Most offending incorrect answer" ( ) nedir?

Cevap: Sadece doğru cevabın enerjisini bastıran **enerji kaybı**, yanlışları yukarı itmediği için enerjiyi her yerde düzleştirebilir — sistem **çöker** (LeCun 1:20:13; Hafta 8-9 collapse). İyi bir kayıp, doğru cevabın enerjisini en-çok-suç-işleyen yanlış cevabın (: yanlış olduğu hâlde en düşük enerjili) enerjisinden **en az bir marj** kadar küçük tutar (LeCun 1:25:47):

$$L = \max(0,\ m + E(x, y) - E(x, \bar{y}))$$

'yi seçmek sınıflandırmada kolay, sürekli/yüksek-boyutlu uzayda zordur — "negatif nasıl seçilir?" sorusu MoCo ile SimCLR'ı ayıran şeydir (LeCun 1:36:09).

i Soru 4: (Builder) PPUU'da "action insensitivity" (bilgi sızıntısı) sorunu nedir, latent dropout nasıl çözer?

Cevap: Variational encoder geleceği gördüğü için "döndük" bilgisini latent değişkene **sızdırır**; o zaman forward model direksiyonu (eylemi) yok sayar — küçük bir direksiyon değişikliği büyük bir MSE değişikliği yarattığından, ağ bu bilgiyi eylem yerine latent'ten alır. Çözüm **latent dropout**: eğitimin bir kısmında z_t'yi encoder yerine prior'dan örnekle (Canziani 52:52). Böylece dönme latent'e kodlanamaz (bazen yok olur), ağ eylemi kullanmak zorunda kalır → eylem-gelecek bağı kurulur.

ℹ Soru 5: Belirsizlik düzenlileştirmesi (“U”) politikayı nasıl güvende tutar? Hangi Hafta-9 ilkesinin somut hâlidir?

Cevap: Politika tek başına dünya modelini “kandırıp” manifold dışına çıkar (sıfır-maliyet hayalleri görür). Dünya modelinin **varyansı** eğitim bölgesi dışında artar (ensemble/dropout ile ölçülür); varyans türevlenebilir, böylece onu da minimize edersin (Canziani 1:18:00). Nihai kayıp = görev maliyeti + λ·varyans. Bu, politikayı eğitim verisinin etrafında (güvenli) tutar — Hafta 9’un **“düşük-enerji hacmini sınırla”** non-contrastive ilkesinin mühendislik hâlidir.

18.12 Egzersizler

Egzersiz 1 (Aktivasyon ölçeği). Tek-kıvrımlı (ReLU) ve yumuşak (softplus, β=1 ve β=10) bir fonksiyonu çiz. Girişi 0.1, 1, 10 ile ölçekleyince ReLU’nun şeklinin değişmediğini ama softplus’ın “sertleştiğini” göster. Bu, neden derin ağlarda ReLU + BatchNorm’un tercih edildiğini nasıl açıklar?

```
import numpy as np

# ReLU OLCEK-DEGISMEZ: relu(c*x)/c hep ayni sekil (pozitif homojenlik).
# softplus OLCEK-BAGIMLI: beta buyudukce sekil degisir (gomulu olcek).
def relu(x):
    return np.maximum(0.0, x)

def softplus_scaled(x, beta=1.0):
    # (1/beta) * log(1 + e^(beta*x)) -> beta buyuk: ReLU'ya yaklasir
    return (1.0 / beta) * np.log1p(np.exp(-np.abs(beta * x))) + np.maximum(x, 0.0)

x = np.linspace(-3, 3, 7)
for c in [0.1, 1.0, 10.0]:
    print(f"ReLU(c*x)/c, c={c:>4}: {np.round(relu(c * x) / c, 2)}")   # AYNI sekil
for beta in [1.0, 10.0]:
    print(f"softplus beta={beta:>4}:   {np.round(softplus_scaled(x, beta), 2)}")  # DEGISIR
# ReLU normalizasyonla uyumlu (olcek serbest); softplus gomulu olcek tasir -> catisir
```

Egzersiz 2 (MSE vs L1). Tek bir x için y değerleri {1, 2, 2, 10} (bir aykırı) gözlemlensin. MSE’yi minimize eden tahmin (ortalama = 3.75) ile L1’i minimize edeni (medyan = 2) karşılaştır. Görüntü tahmininde bu fark neden “bulanık vs keskin”e dönüşür?

```
import numpy as np

# MSE-min = ORTALAMA (aykiri tarafindan cekilir, "bulanik");
# L1-min  = MEDYAN  (aykiriyi gormezden gelir, "keskin").
veri = np.array([1.0, 2.0, 2.0, 10.0])
print(f"MSE minimumu (ortalama) = {veri.mean():.2f}")    # 3.75 <- 10 aykirisi ceker
print(f"L1  minimumu (medyan)   = {np.median(veri):.2f}") # 2.00 <- aykiri etkisiz
```

```
# Goruntude: cok-gelecek varsa MSE hepsinin ORTALAMASINI cizer -> bulanik;
# L1 medyani sectigi icin tek bir keskin moda yakin kalir.
for t in [2.0, 3.75]:
    mse = np.sum((veri - t) ** 2)
    l1 = np.sum(np.abs(veri - t))
    print(f"t={t:>4}: MSE={mse:6.2f}  L1={l1:5.2f}")
# MSE t=3.75'te minimum, L1 t=2'de minimum -> farkli "en iyi tahmin"
```

Egzersiz 3 (EBM kaybı tasarla). İki skor (doğru, yanlış) veren bir model için (a) enerji kaybı, (b) perceptron kaybı, (c) hinge (marj m) kaybını yaz. Hangisi collapse'a açıktır, neden? Marj m'yi büyütmek/küçültmek ne yapar (ipucu: son katman ağırlıklarının ölçeği)?

```
import numpy as np

# E_pos = dogru cevabin enerjisi (DUSUK olmali);
# E_neg = en-cok-suc-isleyen yanlis (ybar) enerjisi (YUKSEK olmali).
def energy_loss(E_pos, E_neg):
    return E_pos                        # (a) sadece dogruyu bas -> COLLAPSE riski
def perceptron_loss(E_pos, E_neg):
    return np.maximum(0.0, E_pos - E_neg)        # (b) marjsiz -> collapse'a acik
def hinge_loss(E_pos, E_neg, m=1.0):
    return np.maximum(0.0, m + E_pos - E_neg)    # (c) marj m ZORLAR -> iyi

E_pos, E_neg = 0.2, 1.5
print(f"enerji    = {energy_loss(E_pos, E_neg):.2f}  (yanlisi itmez -> COLLAPSE)")
print(f"perceptron= {perceptron_loss(E_pos, E_neg):.2f}  (marjsiz)")
for m in [0.5, 1.0, 2.0]:
    print(f"hinge m={m}: {hinge_loss(E_pos, E_neg, m):.2f}")
# (a) ve (b) collapse'a acik (marj yok); m buyumesi daha buyuk enerji-farki
# (dolayli olarak son katman agirligi olcegi) zorlar -> daha guclu ayrim
```

Egzersiz 4 (PPUU latent dropout). Bir koşullu predictor'da encoder'dan gelen latent'i %50 olasılıkla prior örneğiyle değiştir. "Eğitim sırasında her zaman görülen tek sinyal eylemdir" cümlesini bu mekanizmayla açıkla. Latent dropout olmazsa hangi yanlış kestirme (shortcut) öğrenilir?

```
import numpy as np

# Latent dropout: bazi adimlarda z'yi ENCODER yerine PRIOR'dan ornekle.
# Boylece "donme/gelecek" bilgisi latent'e GUVENILIR sekilde kodlanamaz
# -> ag bu bilgiyi EYLEM'den almak ZORUNDA kalir (action insensitivity fix).
def sample_latent(encoder_mu, encoder_sigma, p_drop=0.5, rng=None):
    rng = rng or np.random.default_rng(0)
    if rng.random() < p_drop:
        return rng.normal(0.0, 1.0, size=encoder_mu.shape)   # PRIOR (bilgi yok)
    return encoder_mu + encoder_sigma * rng.normal(0.0, 1.0, size=encoder_mu.shape)
```

```
rng = np.random.default_rng(0)
mu, sigma = np.zeros(4), np.ones(4)
drops = [np.allclose(sample_latent(mu, sigma, 0.5, rng), 0, atol=5) for _ in range(6)]
print("bazi adimlarda latent = prior (bilgi tasimaz):", "evet" if any(drops) else "hayir")
# Latent dropout YOKSA: encoder gelecegi latent'e sizdirir -> forward model
# direksiyonu (eylemi) YOK SAYAR (shortcut). Tek tutarli sinyal = eylem.
```

Egzersiz 5 (Hafta 12 habercisi — NLP & Transformer). Hafta 12'de **konuk Mike Lewis** (FAIR) NLP/seq2seq/decoding'i anlatacak; Canziani attention ve Transformer'ı kuracak. (a) Bu haftaki softmax tartışmasını (β sıcaklık, "0'a yakın katsayı için giriş $-\infty$ olmalı") attention katsayılarıyla nasıl ilişkilendirirsin? (b) Hafta 6'nın attention/seq2seq'i ile Transformer'ın "set→set" görüşü arasında ne fark beklersin?

```
import numpy as np

# (a) softmax sicakligi beta = "ters sicaklik": dusuk beta -> yumusak/duz dagilim,
#     yuksek beta -> sert/odakli (tek anahtara yaklasir). Attention katsayilari
#     da softmax(skor/sqrt(d))'dir -> ayni "0'a yakin agirlik icin skor -> -inf".
def softmax(z, beta=1.0):
    z = beta * (z - z.max())
    e = np.exp(z)
    return e / e.sum()

skorlar = np.array([2.0, 1.0, 0.1])
for beta in [0.3, 1.0, 5.0]:
    w = softmax(skorlar, beta)
    print(f"beta={beta}: attention agirliklari = {np.round(w, 3)}")  # beta buyur -> odak
# (b) Hafta 6: seq2seq = TEKRARLAMALI (zaman zinciri); Transformer = set->set
#     (tum konumlar paralel, recurrence YOK) -> Hafta 12'de Canziani kuracak.
```

18.13 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H12: NLP, Decoding ve Attention/Transformer (Konuk: Mike Lewis)

Kayıp = enerji çerçevesini dile taşıyoruz. Bu hafta aktivasyonları (ölçek-değişmezliği), kayıpları (MSE bulanık, cross-entropy = logsoftmax+NLL, hinge = marj), EBM collapse vs margin'i ve PPUU'yu (dünya modeli + latent + belirsizlik) kapattık. Hafta 12'de konuk **Mike Lewis** (FAIR) NLP/seq2seq/decoding'i (beam search) anlatacak; **Canziani** attention'ı ve Transformer'ı ("sets to sets") kuracak. Egzersiz 3 (EBM kaybı) ve Egzersiz 5 (softmax → attention) tam bu derse hazırlar.

Hafta 12: NLP, Decoding ve Attention/Transformer — Mike Lewis (Konuk Lecture) + Canziani (Practicum)

Hafta 12'de konuk **Mike Lewis** (FAIR) doğal dil işleme, seq2seq, decoding (beam search) ve çeviriyi; **Canziani** ise attention'ı ve Transformer'ı ("sets to sets") anlatacak.

Hafta 12 öncesi yapılacak:

- Egzersiz 3 (EBM kaybı) ve Egzersiz 5 (softmax → attention) çöz.
- "Kayıp = enerji + marj" ve "latent = eksik bilgi" cümlelerini kendi sözcüklerinle yaz.
- Hafta 6 attention/seq2seq'i hatırla — Transformer onun "tekrarlamasız" hâlidir.

18.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Ölçek-değişmezliği	Tek-kıvrım: giriş×2 → çıkış×2 (şekil sabit); derin ağda iyi	LeCun 27m04
Softmax sıcaklığı β	Ters sıcaklık; annealing (yumuşak→sert), mixture-of-experts	LeCun 29m45
MSE → bulanık	Çoklu-gelecekte ortalama; görüntü ortalaması = bulanık	LeCun 37m50
L1 → medyan	Keskin, aykırıya dayanıklı; tabanda türevsiz	LeCun 38m06
Cross-entropy = logsoftmax+NLL	Sayısal kararlılık; KL ıraksaması (sistem vs one-hot)	LeCun 47m22
Enerji kaybı → collapse	Marjsız → enerji düzleşir; tehlikeli	LeCun 1h20
Margin/hinge kaybı	Doğru < yanlış ( ) − marj; "iyi" kayıp	LeCun 1h25
(most offending)	En düşük enerjili yanlış cevap; seçimi sürekli uzayda zor	LeCun 1h36
PPUU dünya modeli	$s_t=(p_t,v_t,i_t)$; doluluk ızgarası; kinematik tersi=eylem	Canziani 8m26
Latent + KL	16-boyut z; çoklu-gelecek; prior'dan örnekle (VAE)	Canziani 35m22
Action insensitivity / latent dropout	Sızıntıyı kes; ağ eylemi kullansın	Canziani 52m52
Belirsizlik düzenlileştirmesi (U)	Varyans minimizasyonu → güvenli manifold	Canziani 1h18

18.15 ML Builder Bağlantıları

Geriye köprüler (önkoşul + kurs):

1. **Aktivasyon ölçeği** → Hafta 2-3 ağırlık ölçeği + BatchNorm.
2. **Cross-entropy / softmax** → Hafta 2 sınıflandırma + Hafta 7 enerji yorumu.
3. **EBM kayıpları (collapse, margin,)** → Hafta 7 EBM + Hafta 8 contrastive + Hafta 9 collapse/non-contrastive + Hafta 10 hard-negative.
4. **PPUU dünya modeli** → Hafta 9 (world model) + Hafta 10 (emulator/controller).
5. **Latent + KL + zincir-backprop** → Hafta 8 VAE + Hafta 5 autograd + Hafta 6 BPTT.

İleriye köprüler (production / research):

1. **Kayıp = enerji tasarımı** → contrastive/non-contrastive temsil öğrenmenin dili.
2. **Negatif seçimi** → MoCo/SimCLR → **non-contrastive BYOL/VICReg (post-2020, KURSTA YOK)**.
3. **PPUU + belirsizlik + dünya modeli** → model-based RL, MPC, **JEPA (post-2020)**.
4. **Latent çoklu-gelecek** → CVAE, diffusion, koşullu üretici modeller.

! Bu dersten tek bir şey alıp gideceksen

Bir kayıp fonksiyonu seçmek aslında bir **enerji manzarası tasarlamaktır** — doğru cevabı aşağı itip yanlışları bir **marjla** yukarı it; marj yoksa enerji çöker (LeCun'un enerji-kaybı uyarısı = Hafta 8-9 collapse). Ve PPUU, Hafta 9-10'un dünya modelini üç ekle tamamlar: **latent değişken** (çoklu-geleceğin bulanıklığını çözer, Hafta 8 VAE), **latent dropout** (eylem-körlüğü sızıntısını keser) ve **belirsizlik düzenlileştirmesi** (varyansı minimize ederek politikayı güvenli manifoldda tutar — Hafta 9'un "hacmi sınırla" ilkesi). İkisi de aynı temele oturur: enerjiyi/belirsizliği doğru verinin etrafında şekillendirmek — ve bu temelin post-2020 zirvesi (non-contrastive SSL, JEPA) kursta yoktur ama LeCun'un bugünkü programının doğrudan devamıdır.

19 NLP, Transformer ve Attention

İki parçalı hafta — konuk hoca Mike Lewis (Facebook AI Research, NLP/çeviri) Lecture'da derin öğrenmenin doğal dil işlemeyi nasıl dönüştürdüğünü anlatır: dil modelleme bir yoğunluk tahminidir ve tüm hüner bağlam kodlayıcıdadır; CNN sınırlı alıcı alanıyla, RNN tek vektöre sıkıştıran darboğazı ve yavaşlığıyla yetersizdir, Transformer ise her kelimenin her kelimeye doğrudan baktığı düşük-önyargılı bir modeldir, paralelleşir ve ölçeklenir; multi-head attention Q/K/V ile kurulur, causal maske geleceğe bakmayı engeller ve positional encoding sırayı geri verir; decoding tarafında greedy ile beam search exposure bias yüzünden bozulur, top-k sampling çeşitlilik getirir; ve öz-denetimli öğrenme dilin motorudur — word2vec'ten BERT'e boşluğu doldurma (masked-LM), pre-train ve fine-tune ile her göreve taşınan tek model, ölçek kraldır. Ardından Alfredo Canziani (Practicum) attention'ın altındaki basit matematiği kurar: Transformer dizilerle değil kümelerle çalışır, attention bir kümenin elemanlarını benzerlik skorlarıyla harmanlayan linear kombinasyondur, Q/K/V girdinin öğrenilen rotasyonlarıdır, encoder ile decoder bir autoencoder iskeleti oluşturur, autoregressive üretim look-ahead maskeyle eğitilir ve T×T attention matrisi küme büyüdükçe karesel olarak patlar.

ℹ Bölüm bilgisi (KONUK Mike Lewis + Canziani)

- **Konuk Lecture (Mike Lewis, FAIR):** YouTube — Deep Learning for NLP (Hafta 12 Lecture)
- **Canziani'nin Practicum videosu:** YouTube — Attention & the Transformer
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Mike Lewis (Konuk Lecture, NLP/Transformer) + Alfredo Canziani (Practicum, attention matematiği) — Yann LeCun tartışmaya katılır (word2vec = basit GNN köprüsü)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈30 dk

⚠ **Atıf notu:** Bu haftanın Lecture'ı **LeCun değil, konuk Mike Lewis** (FAIR, doğal dil işleme). Lecture quote'ları **— Lewis**; Practicum **— Canziani**; LeCun tartışmaya katılıp köprü kurduğunda **— LeCun**.

19.1 Bu Derste Ne Var?

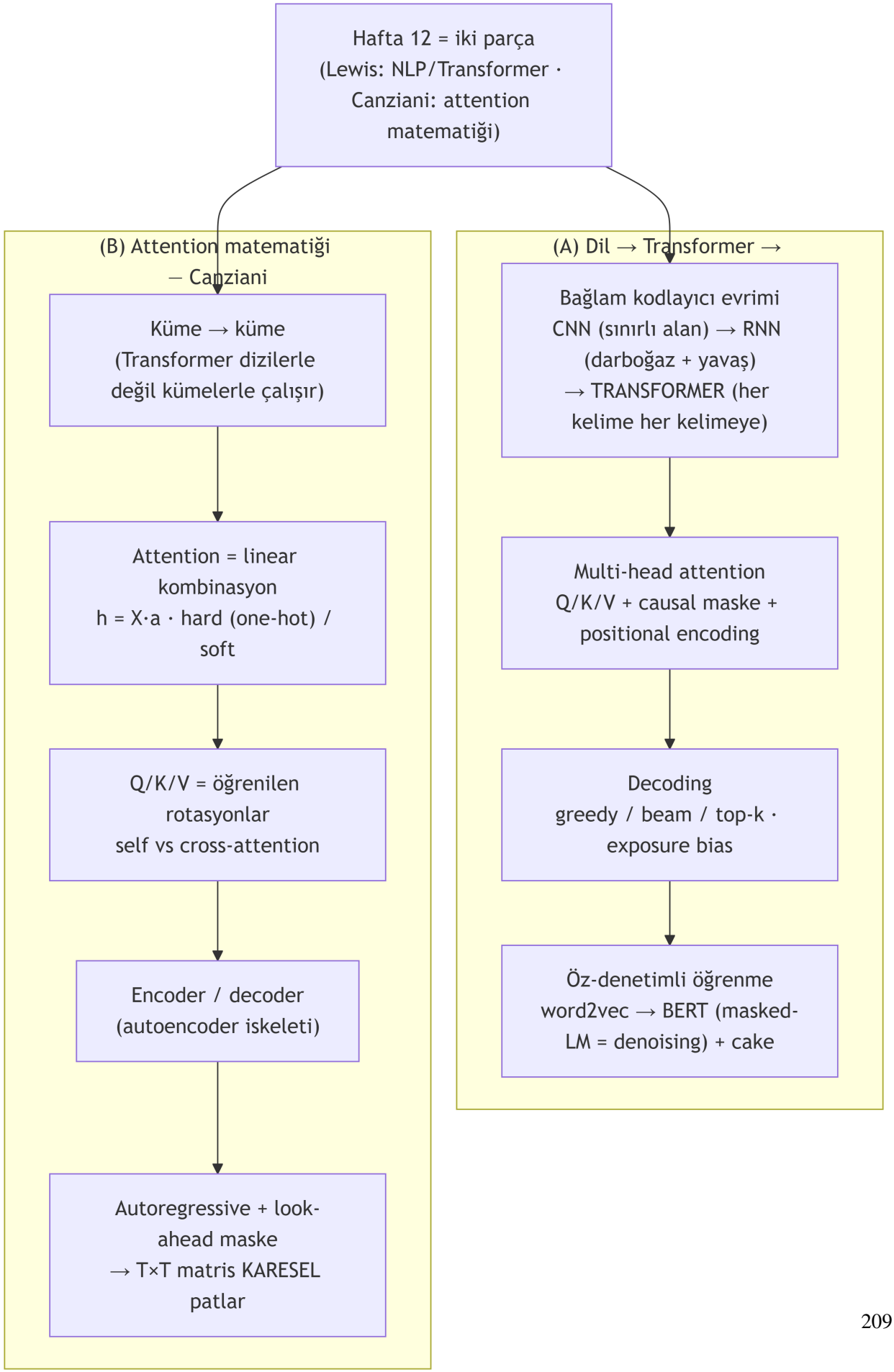
Bu hafta iki parça: **konuk hoca Mike Lewis** (Facebook AI Research, NLP/çeviri) Lecture'da derin öğrenmenin doğal dil işlemeyi nasıl dönüştürdüğünü anlatıyor — dil modellemeden Transformer'a, decoding'den BERT'e; **Alfredo Canziani** (Practicum) ise attention'ın altında yatan matematiği (kümeler üzerinde linear kombinasyon, Q/K/V) ve Transformer encoder/decoder mimarisini koddan kuruyor.

Lewis'in büyük fikri: dile özel yapı **dayatmaktan vazgeç**; çok-ifadeli, düşük-önyargılı (low-bias) bir model (Transformer) al, çok metin göster, yapıyı kendisi öğrensin. Canziani: Transformer'ın görünür karmaşıklığının

altında basit bir fikir var — attention, bir **kümenin** elemanlarını benzerlik skorlarıyla harmanlayan linear kombinasyondur.

Bu haftanın üç ana fikri:

1. **Transformer = her kelime her kelimeye doğrudan bakar.** RNN darboğazını (tek vektöre sıkıştır) ve CNN sınırlı alıcı alanını ortadan kaldırır; paralelleşir, ölçeklenir.
2. **Attention = kümeler üzerinde benzerlik-ağırlıklı linear kombinasyon.** Q/K/V = girdinin öğrenilen rotasyonları; tek doğrusalsızlık softmax; sıra bilgisi positional encoding.
3. **Öz-denetimli öğrenme dilin motorudur.** word2vec/BERT "boşluğu doldur" (cake metaforu); pre-train → fine-tune tek modeli her göreve taşır.


Hafta 12 = iki parça
(Lewis: NLP/Transformer ·
Canziani: attention
matematiği)
(B) Attention matematiği
— Canziani
Küme → küme
(Transformer dizilerle
değil kümelerle çalışır)
Attention = linear
kombinasyon
h = X·a · hard (one-hot) /
soft
Q/K/V = öğrenilen
rotasyonlar
self vs cross-attention
Encoder / decoder
(autoencoder iskeleti)
Autoregressive + look-
ahead maske
→ T×T matris KARESEL
patlar
(A) Dil → Transformer →
Bağlam kodlayıcı evrimi
CNN (sınırlı alan) → RNN
(darboğaz + yavaş)
→ TRANSFORMER (her
kelime her kelimeye)
Multi-head attention
Q/K/V + causal maske +
positional encoding
Decoding
greedy / beam / top-k ·
exposure bias
Öz-denetimli öğrenme
word2vec → BERT (masked-
LM = denoising) + cake

Builder Notu — İki Hoca, Tek Fikir: Her Kelime Her Kelimeye Bakar

Geriye (önkoşul + kurs):

- **Bağlam kodlayıcı** → Hafta 6 (RNN/LSTM/attention/seq2seq); Transformer onun **tekrarlamasız** hâli.
- **Softmax / β** → Hafta 11 (attention katsayısı = softmax, sıcaklık); attention katsayısı 0'a yaklaşsın → giriş $-\infty$.
- **SSL / cake / pretext** → Hafta 10 (Misra) + Hafta 7-8 (BERT masked-LM = denoising autoencoder).

İleriye (production / research):

- Transformer → tüm modern LLM (GPT/BERT/T5) omurgası; ViT, protein, multimodal.
- "word2vec = basit GNN" (LeCun) → Hafta 13 Graph ConvNets.

Tek cümleyle: Transformer, her kelimenin her kelimeye doğrudan baktığı, kümeler üzerinde benzerlik-ağırlıklı linear kombinasyon (attention) yapan düşük-önyargılı bir modeldir; öz-denetimli "boşluğu doldur" (BERT) ile etiketsiz metinden öğrenir, pre-train→fine-tune ile her dil görevine taşınır.

19.2 (Lewis — Konuk) Dil Modelleme ve Bağlam Kodlayıcının Evrimi

Lewis dil modellemeyle başlıyor: metnin **yoğunluk tahmini** (her dizeye olasılık ata). Bu olasılık zincir kuralıyla çarpanlara ayrılır → bir dizi "önceki kelimelere bakıp **bir sonraki kelimeyi tahmin et**" sınıflandırması. Tüm hüner **bağlam kodlayıcıdadır** (context encoder); onun evrimi dersin omurgasıdır ("Bu Derste Ne Var" şemasında CNN → RNN → Transformer evrimi de buradadır):

- **CNN** (Bengio 2003; Dauphin 2016): hızlı, paralel — ama **sınırlı alıcı alan** (uzun-menzilli bağımlılığı yakalayamaz).
- **RNN**: ilkesel olarak sınırsız bağlam — ama tüm geçmişi **tek vektöre sıkıştırır** (darboğaz), gradyan kaybolur (Hafta 6), ardışık olduğu için **yavaştır**.
- **Transformer** (Vaswani 2017, "Attention is all you need"): ikisinin de sorununu çözer.

> "for self attention you can in principle put 100% attention on any word in the distant past… it avoids issues like vanishing gradients quite effectively." — Lewis, 31:43

Gücün kaynağı **her kelime çifti arasında doğrudan bağlantı**. Self-attention karesel olsa da tek bir büyük matris çarpımı olarak **paralel** hesaplanır → olağanüstü ölçeklenir (GPT-2 2019'da 2 milyar, sonra 17 milyar parametreye…).

Builder Notu — Bağlam Kodlayıcı Evrimi

Geriye (Hafta 6): RNN darboğazı + vanishing gradient = LSTM'in savaştığı sorun; attention = seq2seq attention. Transformer onu **tekrarlama olmadan** yapar.
İleriye: "Her kelime her kelimeye bakar" = LLM'lerin temel hesaplama modeli; paralelleşme = GPU-çağı ölçeklemenin anahtarı.

19.3 (Lewis — Konuk) Multi-Head Attention, Maskeleme ve Positional Encoding

Transformer bloğu iki alt-katmandan oluşur: **multi-head attention** + feedforward (MLP), her ikisi de **Add&Norm** (residual bağlantı + LayerNorm) ile sarılı. (LayerNorm, BatchNorm değil — batch boyutuna bağlı olmadığı için NLP'de tercih edilir.)

Multi-head attention'ın kalbi: her kelime için bir **query** ("önceki sıfat ne?"), bir **key** ("ben bir sıfatım") ve bir **value**. Query·key nokta çarpımı → softmax → önceki kelimeler üzerinde bir dağılım → value'ların ağırlıklı toplamı. "Multi-head" = bunu paralel olarak birden çok kez yapmak (aynı anda farklı şeyleri yakalar: hem "boynuzlu" hem "gümüş-beyaz" hem "bunlar → çoğul").

İki kritik detay var. Birincisi **causal maskeleme**: tüm zaman adımları aynı anda hesaplandığından, hiçbir şey kelimelerin **geleceğe bakmasını** engellemez (hile = kelimeyi kendisiyle tahmin etmek). Şekil 19.1 bu sorunu ve çözümünü iki panelde gösterir: maskesiz attention'da her token herkese bakar (hile), causal maske ise üst-üçgeni $-\infty$ yaparak her token'ı yalnız kendine ve soluna bakmaya zorlar.

> "if you're computing all the time steps at once there's nothing to stop words looking at the future… the solution here is self-attention masking." — Lewis, 27:08

İkincisi **positional encoding**: attention girdiyi bir **küme** olarak görür (sırası yoktur). Dilde sıra önemlidir → her konuma ayrı bir embedding öğrenilip eklenir. Şekil 19.2 klasik sinüzoidal kodlamayı gösterir: her konum, frekansları boyut boyunca değişen bir sinüs/kosinüs imzasıyla işaretlenir.

Diğer "hileler" de kritik: LayerNorm, **learning-rate warmup** (0'dan hedefe doğrusal ısınma), dikkatli initialization, label smoothing.

Builder Notu — Maske = Hile Yok, Konum Eklenir

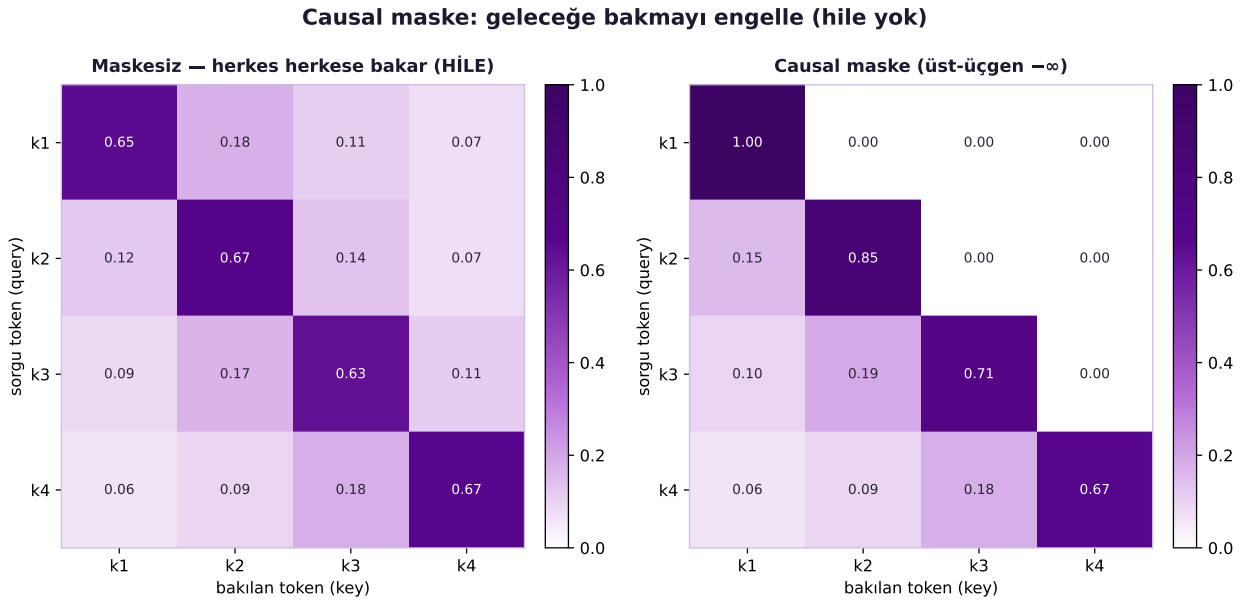
Geriye (Hafta 11 + 6 + 3): Softmax + β = Hafta 11 (katsayı 0'a yaklaşsın → giriş $-\infty$); residual = Hafta 3; attention = Hafta 6. Causal maske, autoregressive faktörizasyonu korur.
İleriye: Q/K/V + maske + positional encoding = her LLM'in değişmez iskeleti; uzun bağlam = karesel maliyetle savaş (sparse/lineer attention).

19.4 (Lewis — Konuk) Decoding ve Öz-Denetimli Öğrenme: word2vec'ten BERT'e

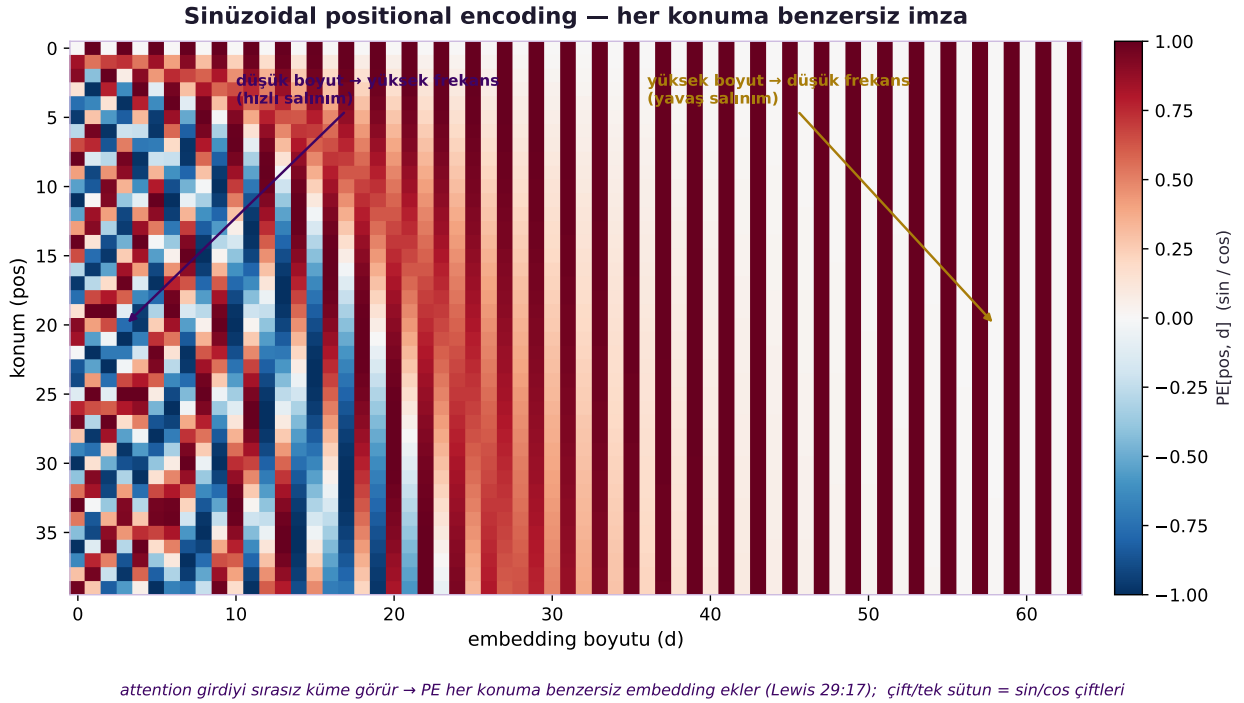
Decoding (çıkarım): model her diziye olasılık atıyor — peki metni nasıl **üretiriz**? Tüm dizi üzerinde argmax üsteldir (hesaplanamaz). Pratik seçenekler:

- **Greedy:** her adımda en olası kelimeyi al — geri-izleme yok.
- **Beam search:** en iyi n hipotezi tut; ama büyük beam → bozuk/boş çıktılar.
- **Sampling / Top-k** (Angela Fan): dağılımı k-best'e kırp, sonra örnekle → hem çeşitlilik hem "iyi dil manifoldundan" düşmeme. GPT-2'nin güzel örnekleri böyle üretildi.

Şekil 19.3 bu farkı iki panelde gösterir: tam dağılımda greedy tek en olası kelimeyi seçerken, top-k=4 dağılımı dört en iyiye kırpıp yeniden normalize eder (sonra örnekler). Bozulmanın asıl sebebi **exposure bias**: eğitimde model kendi hatalarını hiç görmez (hep doğru önceki kelimeler verilir); test anında bir kötü adım atınca daha önce görmediği bir duruma düşer ve döngüye girer.



Şekil 19.1: Causal (look-ahead) maske, 2 panel — GERÇEK softmax çıktısı. 4 token için anlamlı bir skor matrisi (k1..k4) üzerinde causal_mask_demo çalıştırılır. SOL ‘Maskesiz’: her token herkese (geleceğe dâhil) bakabilir — bu eğitimde HİLE’dir çünkü model bir kelimeyi tahmin ederken cevabı (gelecek kelimeyi) görür. SAĞ ‘Causal maske (üst-üçgen $-\infty$)’: üst-üçgen sıfırlanır (beyaz), her token yalnız kendine ve soluna bakar → autoregressive faktörizasyon korunur, hile yok (Lewis 27:08). Hücre değerleri yazılı; her satır toplamı 1.



Şekil 19.2: Sinüzoidal positional encoding (Vaswani 2017) — GERÇEK PE matrisi (FLAGSHIP). positional_encoding(seq_len=40, d=64) → PE [40, 64] heatmap (diverging RdBu_r, [-1,1]). Klasik desen: embedding boyutu (sütun) arttıkça frekans düşer — düşük boyutlar hızlı salınır (sık çizgili), yüksek boyutlar yavaş (geniş bantlar); çift/tek sütunlar sin/cos çiftleridir. Her konum (satır) böylece benzersiz bir imza alır. Sezgi: attention girdiyi SIRASIZ küme görür; PE her konuma benzersiz bir konum imzası ekler (Lewis 29:17), böylece 'küme' yeniden 'sıraya' kavuşur.

> "at training time we're typically not using a beam… we're not exposing the model to its own mistakes." — Lewis, 49:54

Öz-denetimli öğrenme dersin doruğu — Lewis, LeCun'un **cake (pasta)** metaforuna başvuruyor:

> "most of the learning we do has to be unsupervised… [supervised learning is] just being a little bit of icing on top of the cake." — Lewis, 1:09:43

Evrim şöyle: **word2vec** (boşluğu doldur → kelime embedding; "kral − erkek + kadın ≈ kraliçe") ama bağlamdan **bağımsız** (kelime başına tek vektör). Sonra **GPT** (tek-yönlü dil modeli → fine-tune ile her görev), **ELMo** (iki-yönlü birleştirme) ve **BERT**:

> "bert basically looks quite a lot like word2vec, it's a fill-in-the-blanks task… the reason it works much better is [the transformer, more context, bidirectional]." — Lewis, 1:18:50

BERT, token'ların %15'ini maskeler ve doldurur — bu bir **denoising** görevidir (Hafta 7-8 denoising autoencoder). RoBERTa/BART/T5 bunu basitleştirip **ölçekler**. Özet: düşük-önyargı + iki-yönlü bağlam + **ölçek**.

LeCun tartışmaya katılıp bir sonraki haftaya köprü kuruyor:

> "you can view those self-supervised training systems [word2vec, BERT] as basically a very simple form of graph neural net… the graph is how often two words appear next to each other." — LeCun, 1:40:31

Şekil 19.4 bu köprüyü resmeder: solda word2vec'in bağlamsız analoji düzlemi, ortada Lewis'in "kümeyi harmanla" (set-averaging) sezgisi (attention'a köprü), sağda LeCun'un eş-dizim grafiği üzerinde basit GNN yorumu (Hafta 13 GCN kapısı).

Builder Notu — Exposure Bias ve Cake = SSL

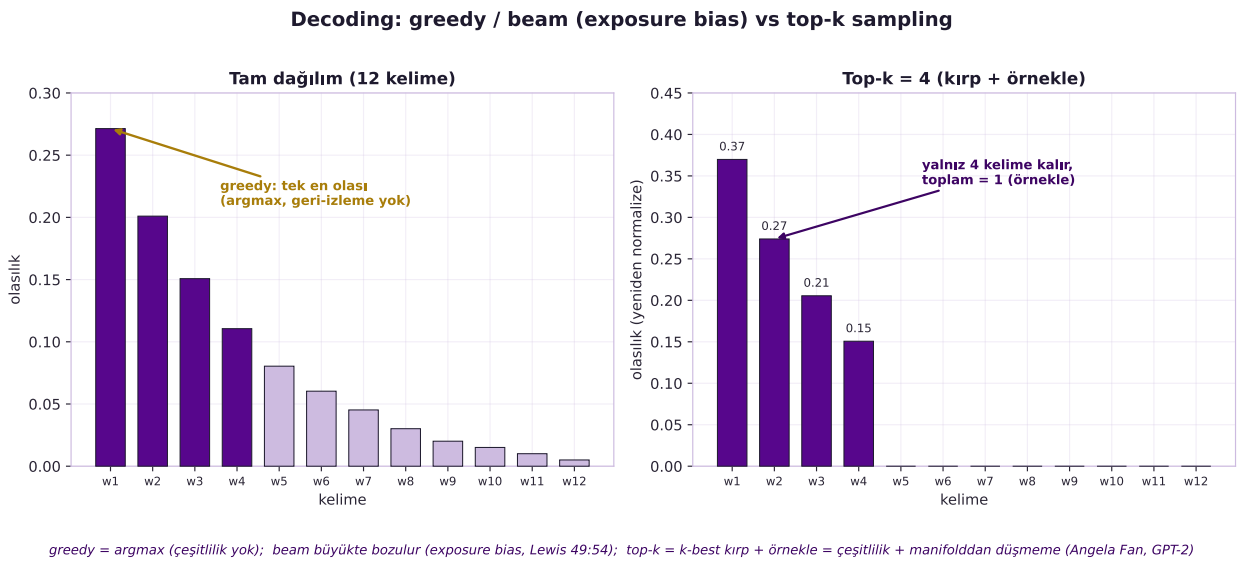
Geriye (Hafta 10 + 7-8): SSL / cake / pretext = Hafta 10 (Misra); BERT masked-LM = Hafta 7-8 denoising AE (gürültüyü geri al = enerji); word2vec analojisi = Hafta 1 temsil öğrenme.
İleriye: pre-train → fine-tune = tüm foundation model paradigması; LeCun'un graf yorumu = Hafta 13 GCN.

19.5 (İleriye Köprü) GPT-3'ten ChatGPT'ye — LLM Patlaması (post-2020) — KURSTA YOK

Lewis (Mart 2020) en güçlü modeller olarak GPT-2, BERT, RoBERTa, BART/T5 ve top-k sampling'i anlatır — "100 milyar parametreli modeller geliyor" söylentisini aktarır. Aşağıdakiler DLSP20'den **sonra** geldi, kursta **YOKTUR** (yalnızca ileriye köprü):

⚠ İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **GPT-3** (Mayıs 2020) — 175 milyar parametre; **in-context learning / few-shot** (Lewis'in "1-10 örnekle öğrenebilir miyiz?" sorusunun cevabı).
- **InstructGPT / RLHF** (2022), **ChatGPT** (Kasım 2022), **GPT-4** (2023), **LLaMA** (2023) — insan geri-bildirimiyle hizalama; Lewis'in "grounding / cherry" sorusunun pratik yanıtı.



Şekil 19.3: Top-k decoding, 2 panel — GERÇEK kırpma + yeniden normalize. 12 kelimelik azalan bir olasılık dağılımı üzerinde topk_truncate(probs, 4). SOL 'Tam dağılım (12 kelime)': gri/violet barlar; greedy = argmax (gold ok, tek en olası kelimeyi seçer, geri-izleme yok). SAĞ 'Top-k=4 (kırp + örnekle)': yalnız 4 bar (violet, toplamı 1'e yeniden normalize), geri kalan kütle 0. Sezgi: greedy tek-en-iyi (çeşitlilik yok); beam büyük beam'de bozulur (exposure bias: model kendi hatasını hiç görmez, Lewis 49:54); top-k = k-best kırp + örnekle → çeşitlilik + manifolddan düşmeme (Angela Fan, GPT-2).

- **FlashAttention** (2022), uzun-bağlam Transformer'lar — Lewis'in "512 token sınırı / koca bir kitabı modelleyebilir miyiz?" sorusunun devamı.

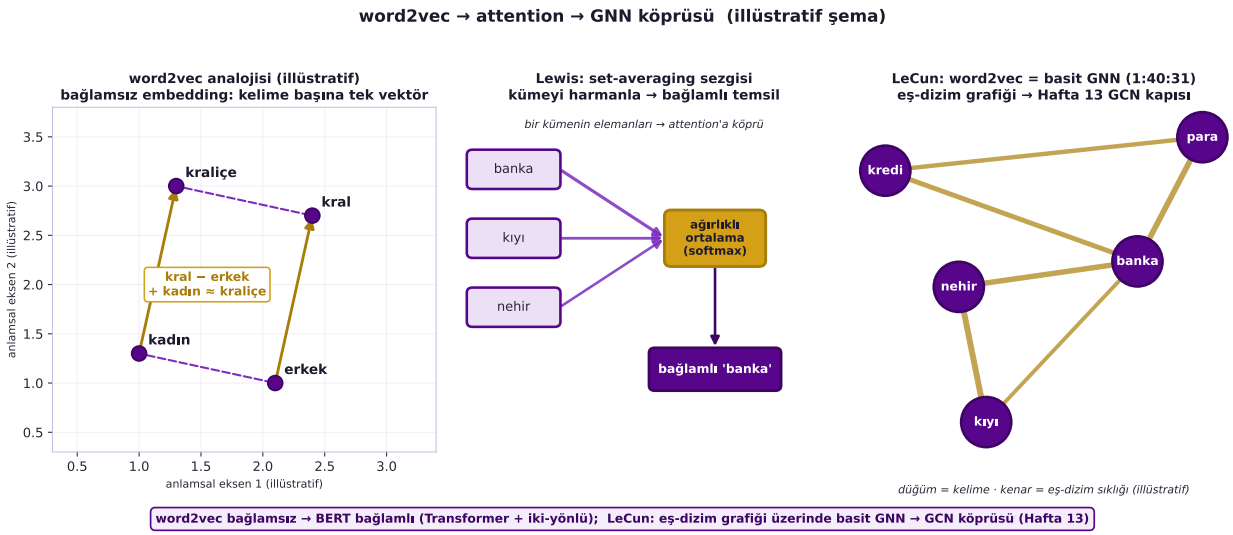
Bunlar kurs terimi gibi eklenmez; Lewis'in "ölçek en önemli şey" tezinin nereye vardığını göstermek için anılır.

Builder Notu — Ölçek Tezi Nereye Vardı

Geriye (Hafta 12): LLM patlaması = "düşük-önyargı + iki-yönlü + ölçek" tezinin doğrudan devamı; RLHF = Lewis'in "grounding" sorusunun yanıtı.
İleriye: in-context learning + RLHF = 2020-sonrası NLP'nin tanımı — ama hepsi 2017 Transformer + 2018 BERT temelinin üstünde durur.

19.6 Geçiş: Lewis'ten Canziani'ye

Lewis, Transformer'ın **neden** çalıştığını ve NLP'yi nasıl dönüştürdüğünü anlattı. Şimdi **Canziani** kaputu açıyor: attention'ın altında yatan basit matematiği (kümeler, linear kombinasyon, Q/K/V) ve Transformer encoder/decoder iskeletini PyTorch'ta kuruyor — görünür karmaşıklığın aslında "yemek tarifi arama" kadar sezgisel olduğunu gösteriyor.



Şekil 19.4: word2vec → attention → GNN köprüsü (illüstratif şema). SOL — word2vec analojisi: bağlamsız embedding'de her kelime tek vektördür; 'kral − erkek + kadın ≈ kraliçe' paralelkenarı bu doğrusal yapıyı gösterir (vektör konumları ILLÜSTRATIF, gerçek bir eğitilmiş embedding değil — yalnızca sezgi). ORTA — Lewis'in set-averaging sezgisi: bir bağlam kümesinin elemanları softmax ağırlıklarıyla harmanlanır (ağırlıklı ortalama) → bağlamlı temsil; bu, self-attention'ın çekirdeğine köprüdür (ok kalınlıkları self_attention_matrix ağırlıklarıyla orantılı). SAG — LeCun'un gözlemi (1:40:31): word2vec, kelimelerin düğüm, eş-dizim sıklığının kenar olduğu bir grafik üzerinde basit bir GNN gibidir (networkx eş-dizim grafiği; kenar ağırlıkları illüstratif) → Hafta 13 GCN kapısı. Özet: word2vec bağlamsız → BERT bağlamlı (Transformer + iki-yönlü); eş-dizim grafiği üzerinde basit GNN → GCN köprüsü.

19.7 (Canziani) Attention = Kümeler Üzerinde Linear Kombinasyon

Kilit çerçeve: Transformer **dizilerle değil, kümelerle** çalışır.

> "transformers are going to be something that maps sets to sets, they don't really deal with sequences." — Canziani, 0:49

Self-attention: gizli temsil h, kümenin (X) elemanlarının **linear kombinasyonudur**: $h = X \cdot a$. Katsayı a iki türlü olabilir:

- **Hard attention:** a tek-sıcak (one-hot) → kümeden **tek** bir eleman seçer.
- **Soft attention:** a toplamı 1 → konveks kombinasyon (harman).

a nereden gelir? Her x_i ile sorgu x arasındaki **benzerlik skorundan** (nokta çarpımı), softmax'ten:

$$a = \text{softmax}\left(\frac{X^\top x}{\sqrt{d}}\right)$$

(Burada $\beta = 1/\sqrt{d}$ ölçeklemesi vardır: vektör büyüklüğü $\sqrt{d}$ ile büyür, sıcaklığı sabit tutmak için bölünür.) Şekil 19.5 bu β ölçeklemesinin etkisini gösterir: β küçükken dağılım düzleşir (soft attention, harman), β büyürken sivrilir (hard attention, tek-seçim) — bu doğrudan Hafta 11 sıcaklık köprüsüdür. Şekil 19.6 ise gerçek bir token kümesi üzerinde tüm A matrisini hesaplar: benzer token'lar birbirine yüksek dikkat verir, her satır toplamı 1'dir.

Key-value store analojisi: YouTube'da "lazanya" yazarsın (**query**), bu tüm video başlıklarıyla (**keys**) karşılaştırılır, en yüksek skorlu videoyu (**value**) getirirsin.

> "you have one query, you check all the keys and find how matching the key is… and then you retrieve all those values." — Canziani, 19:07

> Builder Notu — Küme→Küme
>
> **Geriye (Hafta 11 + 6 + 1):** Softmax + β = Hafta 11; benzerlik = nokta çarpımı (Hafta 1); linear kombinasyon = ağırlıklı ortalama. Attention = Hafta 6 çekirdeği, "küme" diliyle yeniden anlatılmış.
> **İleriye:** "Sets to sets" görüşü Transformer'ı görüntü (ViT), grafik ve protein gibi her veri türüne taşır.

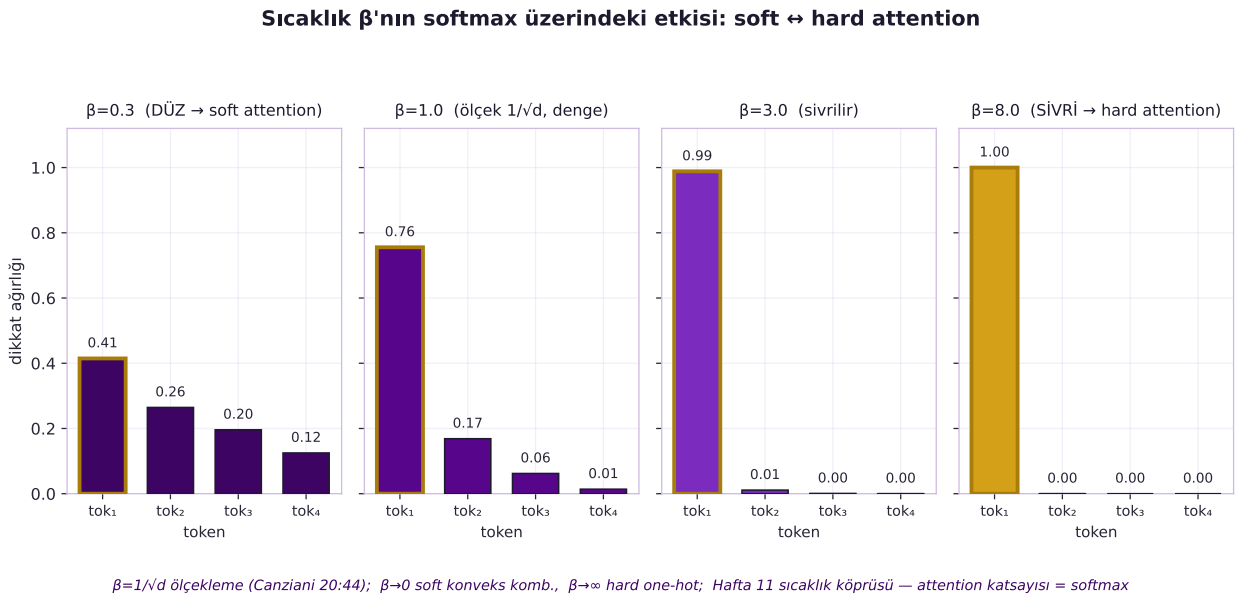
19.8 (Canziani) Query-Key-Value ve Transformer Encoder/Decoder

Q, K, V girdinin öğrenilen **rotasyonlarıdır**: $Q = W_Q x$, $K = W_K x$, $V = W_V x$. Attention'daki **tek eğitilebilir parametreler** bunlardır — attention yönelime (orientation) dayalıdır, softmax dışında doğrusalsızlık yoktur:

> "this attention thing is completely based on orientation… the only non-linearity is whenever you get the probability distribution." — Canziani, 22:05

Q ve K aynı boyutta olmalıdır (karşılaştırılırlar); V farklı olabilir ("tarifin içeriği"). İki tür attention vardır:

- **Self-attention:** Q, K, V hepsi aynı X'ten gelir (kendi kafanda bir tarif düşünmek).



Şekil 19.5: Sıcaklık β'nın softmax üzerindeki etkisi: soft ↔ hard attention. β küçük → düz (soft, harman); β büyük → sivri (hard, tek-seçim). β=1/√d ölçekleme = Hafta 11 sıcaklık köprüsü.

- **Cross-attention:** Q girdiden/decoder'dan, K ve V başka yerden/encoder'dan gelir (tarif kitabına bakmak).

Transformer = encoder + decoder (bir autoencoder gibi). Şekil 19.7 bu iskeleti gösterir:

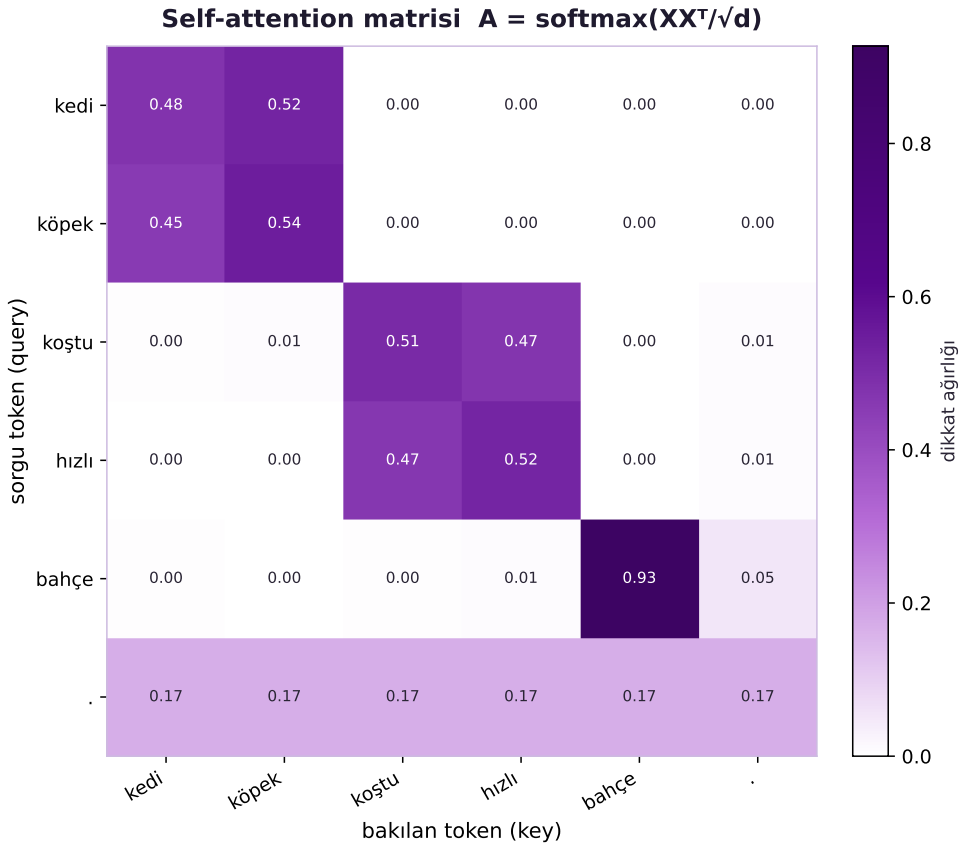
- **Encoder bloğu:** self-attention + "convolution" (kernel=1 olan 1D conv = kümenin her elemanına uygulanan MLP — Canziani "bu lineer katman değil, convolution'dır" diye ısrar eder) + Add&Norm.
- **Decoder bloğu:** encoder'ın yaptıklarına ek olarak **cross-attention** (query decoder'dan, key/value encoder'ın H temsilinden).

IMDB film yorumu sınıflandırması (yalnız encoder) test doğruluğu **%83**'tür.

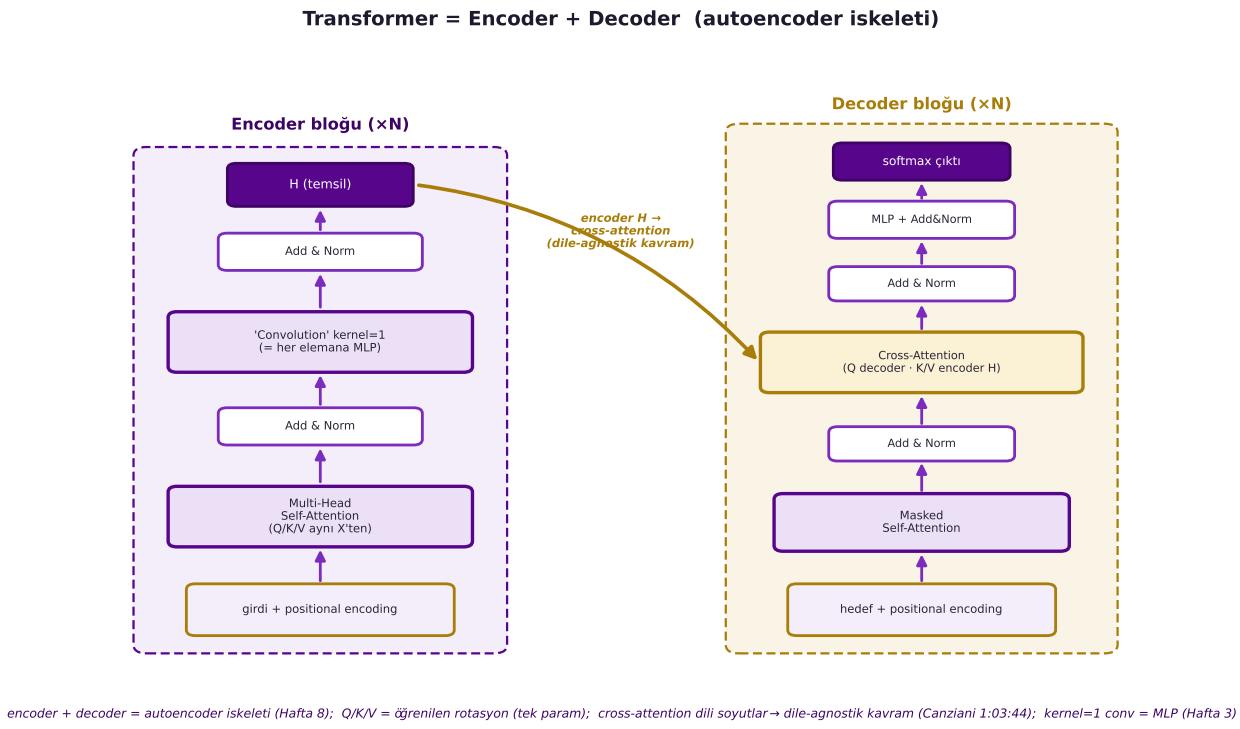
> Builder Notu — Q/K/V = Rotasyon
>
> **Geriye (Hafta 8 + 3 + 1):** Encoder/decoder = Hafta 8 autoencoder mimarisi; residual + LayerNorm = Hafta 3; "convolution kernel=1" = Hafta 3 1×1 conv = MLP. Q/K/V rotasyonu = Hafta 1 lineer dönüşüm. **İleriye:** Encoder-only (BERT) vs decoder-only (GPT) vs encoder-decoder (T5/çeviri) = LLM ailesinin üç dalı.

19.9 (Canziani) Autoregressive Üretim, Look-ahead Mask ve Karesel Maliyet

Çeviri (cat → gato): decoder çıktıyı **autoregressive** üretir — bir kelime üretir, geri besler, sonrakini sorar. Her yeni girdi farklı bir **query** sorar; encoder'ın hangi bileşenine bakacağına decoder karar verir.



Şekil 19.6: Self-attention ağırlık matrisi A = softmax($XX^T/\sqrt{d}$) — GERÇEK hesaplama (FLAGSHIP). 6 token'lık anlamlı bir embedding kümesi X (benzer kelimeler birbirine yakın yerleştirilmiş: 'kedi'/'köpek' yakın, 'koştu'/'hızlı' yakın, '.' uzak) → self_attention_matrix(X) → A [6,6] heatmap (violet). Satır = sorgu token, sütun = bakılan token; A[i,j] = token i'nin token j'ye verdiği dikkat. Benzer token'lar yüksek dikkat alır (açık renk); her satırın toplamı tam 1'dir (softmax). Bu, Canziani'nin 'küme → küme' görüşünün çekirdeğidir (0:49): attention bir kümenin elemanlarını benzerlik-ağırlıklı harmanlar.



Şekil 19.7: Transformer encoder/decoder mimarisi (Q/K/V dâhil) — şema. SOL ‘Encoder bloğu (×N)’: girdi + PE → Multi-Head Self-Attention (Q/K/V aynı X’ten) → Add&Norm → ‘Convolution’ kernel=1 (= her elemana MLP) → Add&Norm → H. SAĞ ‘Decoder bloğu (×N)’: hedef + PE → Masked Self-Attention → Add&Norm → Cross-Attention (Q decoder’dan, K/V encoder H’sinden; gold ok) → Add&Norm → MLP → Add&Norm → softmax çıktı. Encoder + decoder = autoencoder iskeleti (Hafta 8); Q/K/V öğrenilen rotasyonlar = tek parametre; cross-attention dili soyutlar → dile-agnostik kavram (Canziani 1:03:44); kernel=1 conv = MLP (Hafta 3 1×1 conv).

Eğitim vs çıkarım: çıkarım autoregressive (sıralı); eğitimde ise tüm dizi tek geçişte işlenir + **look-ahead mask** (geleceği görmeyi engeller — Lewis'in causal maskesinin aynısı). Cross-attention sayesinde encoder dili soyutlar — H temsilleri **dile-agnostik kavramlardır**:

> "the H encoder are stripping off the language specific information, they are just concepts… the representation of the concept without the language attached." — Canziani, 1:03:44

Son bir uyarı — A **matrisi tehlikelidir**: $T \times T$ boyutludur (T = küme büyüklüğü). 1000 öğeli bir küme → milyon-boyutlu bir matris → karesel olarak **patlar**:

> "this is very dangerous… if you have many keys this stuff starts blowing up quickly." — Canziani, 53:01

Uzun diziler için Reformer/LSH/sparse attention gibi yöntemler gerekir.

Builder Notu — Karesel Maliyet Patlar

Geriye (Hafta 6 + 11): Autoregressive = Hafta 6 dizi üretimi; look-ahead mask = Lewis'in causal maskesi; karesel maliyet = self-attention bedeli (Hafta 11 softmax).
İleriye: Dile-agnostik encoder = çok-dilli MT + multimodal hizalama; karesel maliyet = lineer/sparse attention araştırmasının itici gücü (Reformer, FlashAttention).

19.10 Bu Dersin Özeti

1. **Bağlam kodlayıcı evrimi (Lewis):** CNN (sınırlı alan) → RNN (darboğaz + yavaş + vanishing) → **Transformer** (her kelime her kelimeye, paralel, ölçeklenir).
2. **Multi-head attention (Lewis):** Q/K/V; causal mask (geleceği görme); positional encoding (küme → sıra); warmup / LayerNorm / label-smoothing.
3. **Decoding (Lewis):** greedy / beam (exposure bias → bozulma); top-k sampling (çeşitlilik).
4. **SSL = dilin motoru (Lewis):** word2vec (bağlamsız) → GPT (tek-yön) → ELMo → **BERT** (masked-LM = denoising); ölçek kraldır.
5. **Attention = küme linear kombinasyonu (Canziani):** $h = X \cdot a$; hard (one-hot) vs soft; $a = \text{softmax}(X^\top x/\sqrt{d})$; key-value store.
6. **Q/K/V + encoder/decoder (Canziani):** öğrenilen rotasyonlar; self vs cross; encoder + decoder = autoencoder iskeleti; IMDB %83.
7. **Autoregressive + maske + karesel maliyet (Canziani):** look-ahead mask; dile-agnostik kavram; $T \times T$ matris patlar.
8. **Post-2020 (KURSTA YOK):** GPT-3 / in-context, ChatGPT / RLHF, FlashAttention — "ölçek" tezinin devamı.

Tek Bir Cümle

Transformer, RNN'in darboğazını ve CNN'in sınırlı alıcı alanını ortadan kaldıran, her kelimenin her kelimeye doğrudan baktığı düşük-önyargılı bir modeldir; çekirdeği attention'dır — bir kümenin elemanlarını Q/K/V benzerlik skorlarıyla (softmax) harmanlayan linear kombinasyon — ve öz-denetimli "boşluğu doldur" (BERT) ile etiketsiz metinden öğrenip pre-train→fine-tune ile her dil görevine taşınır.

Soru 1: RNN ve CNN'in hangi sorunlarını Transformer çözer? "Her kelime her kelimeye bakar" ne kazandırır?

Cevap: CNN **sınırlı alıcı alan** taşır (uzun-menzilli bağımlılığı yakalayamaz); RNN tüm geçmişi **tek vektöre sıkıştırır** (darboğaz), **vanishing gradient** sorunu yaşar (Hafta 6) ve ardışık olduğu için **yavaştır** (paralelleşmez). Transformer self-attention ile her kelimeyi her kelimeye **doğrudan** bağlar (Lewis 31:43): herhangi bir geçmişe kayıpsız erişim (darboğaz yok, vanishing büyük ölçüde yok) + tek bir matris çarpımı = **paralelleşir** → olağanüstü ölçekleme.

Soru 2: Multi-head attention'da Q/K/V nedir? Causal mask neden gerekir, positional encoding neden eklenir?

Cevap: Query "ne arıyorum?", **key** kelimeyi etiketler, **value** içeriği taşır. Query·key → softmax → dağılım → value'ların ağırlıklı toplamı; **multi-head** = aynı anda farklı sorular sormak. **Causal mask** (üst-üçgen $-\infty$): tüm adımlar aynı anda hesaplanır, maske olmazsa kelimeler **geleceğe bakıp** hile yapar (Lewis 27:08). **Positional encoding**: attention girdiyi sırasız bir **küme** olarak görür; sıra bilgisi için her konuma bir embedding eklenir.

Soru 3: Beam search neden bozuk çıktı üretir? "Exposure bias" nedir? Top-k sampling ne yapar?

Cevap: Büyük beam → boş/tekrarlı çıktılar. Sebep **exposure bias**: eğitimde hep doğru önceki kelimeler verilir, model kendi **hatalarını görmez** (Lewis 49:54); test anında bir kötü adımdan sonra görmediği bir duruma düşer ve döngüye girer. **Top-k** (Angela Fan): dağılımı en iyi k'ye kırpıp örnekler — çeşitlilik + "iyi dil manifoldundan" düşmeme. GPT-2 örnekleri böyle üretilir.

Soru 4: BERT word2vec'ten neden daha iyi? "Cake" metaforu nedir? BERT hangi Hafta-7/8 fikriyle aynıdır?

Cevap: İkisi de "boşluğu doldur" görevidir; word2vec sığ bir linear projeksiyondur ve **bağlamsızdır** (kelime başına tek vektör). BERT büyük bir **Transformer + iki-yönlü** zengin etkileşim kullanır (Lewis 1:18:50) → bağlamlı temsil. **Cake** (LeCun): öğrenmenin çoğu denetimsizdir (pastanın kendisi), denetimli öğrenme = üstteki ince krema (1:09:43). BERT masked-LM = Hafta 7-8 **denoising autoencoder**: girdiyi boz (%15 maske), geri kur — gürültüyü gidermek = enerjiyi şekillendirmek.

Soru 5: Canziani'ye göre attention özünde nedir? Q ve K neden aynı boyutta, V neden farklı olabilir?

Cevap: Attention = bir **kümenin** (X) **benzerlik-ağırlıklı linear kombinasyonu**: $h = X \cdot a$, $a = \text{softmax}(X^\top x/\sqrt{d})$ (Canziani 0:49, 19:07). Hard = one-hot (tek seçim), soft = konveks harman. Key-value: query tüm key'lerle karşılaştırılır, value'lar getirilir. **Q ve K aynı boyutta olmalıdır** çünkü nokta çarpımıyla **karşılaştırılırlar** (yönelim). **V farklı olabilir** çünkü o "tarifin içeriği"dir — karşılaştırılmaz, getirilir. Tek doğrusalsızlık softmax'tir (22:05).

19.12 Egzersizler

Egzersiz 1 (Attention'ı elle hesapla). 3 elemanlı bir küme $\{x_1, x_2, x_3\}$ (2B) ve bir sorgu x verilsin. Skorları ($x_i \cdot x$), softmax (β=1) katsayılarını ve $h = \Sigma\, a_i\, x_i$'yi hesapla. β büyüdükçe (hard'a doğru) h nasıl değişir?

```
import numpy as np

# Attention = kumenin benzerlik-agirlikli LINEAR KOMBINASYONU: h = X^T a, a = softmax(beta * X x)
X = np.array([[1.0, 0.0],   # x1
              [0.0, 1.0],   # x2
              [0.8, 0.6]])  # x3 (x1'e yakin)
x = np.array([1.0, 0.2])    # sorgu

def softmax(z, beta=1.0):
    z = beta * (z - z.max())
    e = np.exp(z)
    return e / e.sum()

skorlar = X @ x                       # her x_i ile sorgu nokta carpimi
for beta in [0.5, 1.0, 5.0]:
    a = softmax(skorlar, beta)         # katsayilar (toplam 1)
    h = a @ X                          # linear kombinasyon
    print(f"beta={beta}: a={np.round(a,3)}  h={np.round(h,3)}")
# beta buyur -> a one-hot'a yaklasir (HARD) -> h tek bir x_i'ye yaklasir (en yuksek skorlu)
```

Egzersiz 2 (Causal mask). 4 kelimelik bir dizi için 4×4 attention skor matrisi çiz; causal mask (üst-üçgen $-\infty$) uygula; softmax sonrası her satırın yalnız kendine + soluna ağırlık verdiğini göster. Maske olmasa neden "hile" olur?

```
import numpy as np

def softmax_rows(M):
    e = np.exp(M - M.max(axis=-1, keepdims=True))
    return e / e.sum(axis=-1, keepdims=True)

scores = np.random.default_rng(0).normal(0, 1, (4, 4))
mask = np.triu(np.full((4, 4), -np.inf), k=1)   # ust-ucgen (gelecek) = -inf
A = softmax_rows(scores + mask)
print(np.round(A, 2))
# Her satir yalniz kendine + soluna agirlik verir (ust-ucgen = 0).
# Maske OLMASA: kelime tahmin edilirken GELECEK kelimeyi gorur = cevabi kopyalar (HILE);
# maske autoregressive faktorizasyonu p(x1)p(x2|x1)p(x3|x1,x2)... korur.
```

Egzersiz 3 (Q/K/V boyutları). d=8, h=4 head'li multi-head attention'da W_Q / W_K / W_V boyutları nedir? Head'leri birleştiren son W neden gerekir? Q=K, V≠ örneğiyle açıkla.

```
import numpy as np

d_model, n_head = 8, 4
d_head = d_model // n_head            # 2 -> her head'in Q/K/V boyutu
# Her head icin: W_Q, W_K [d_model, d_head] (Q ve K AYNI boyut -> nokta carpimi);
#                W_V      [d_model, d_head] (V FARKLI olabilir, ama burada d_head)
print(f"d_model={d_model}, n_head={n_head}, d_head={d_head}")
print(f"W_Q, W_K, W_V her head icin: [{d_model}, {d_head}]")
print(f"head'leri birlestiren W_O: [{n_head*d_head}, {d_model}] = [{d_model}, {d_model}]")
# W_O GEREKLI: h head'in ciktilari concat edilir [n_head*d_head] -> d_model'e geri yansitilir.
# Q ve K aynı boyutta (karsilastirilir); V "tarifin icerigi" -> farkli olabilir.
```

Egzersiz 4 (BERT = denoising). Bir cümlenin %15'ini maskele, bir Transformer encoder ile doldur. Bunu Hafta 7-8 denoising autoencoder ile eşle: "maske" hangi gürültü, "doldur" hangi enerji-minimizasyonu? word2vec'ten farkı nedir?

```
import numpy as np

cumle = ["kedi", "bahçede", "fareyi", "hızla", "kovaladı"]
rng = np.random.default_rng(1)
maske = rng.random(len(cumle)) < 0.15          # ~%15 token'i maskele
bozuk = [("[MASK]" if m else w) for w, m in zip(maske, cumle)]
print("orijinal:", cumle)
print("bozuk   :", bozuk)   # "maske" = denoising AE'deki GURULTU (Hafta 7-8)
# BERT = denoising AE: girdiyi boz (maske=gurultu), TRANSFORMER ile geri kur ("doldur").
# "Doldur" = dusuk-enerjili (olasi) cumleyi geri getir = enerjiyi sekillendir.
# word2vec farki: BAGLAMSIZ (kelime basina tek vektor); BERT bagalmli + iki-yonlu Transformer.
```

Egzersiz 5 (Hafta 13 habercisi — Graph ConvNets). Hafta 13'te **konuk Xavier Bresson** (NTU) GCN'leri anlatacak; Canziani practicum'u kuracak. (a) LeCun'un "word2vec = basit graph neural net (kelime eş-dizim grafiği)" yorumunu (1:40:31) düşün: bir cümleyi grafik olarak nasıl temsil edersin? (b) Görüntü (düzenli ızgara) ile molekül (düzensiz grafik) arasında "convolution"u genellemek neden zordur?

```
import numpy as np

# (a) Cumle -> grafik: dugum = kelime, kenar = es-dizim (yan yana gelme) sikligi.
#     LeCun: word2vec/BERT, bu graf uzerinde basit bir GNN gibi calisir (1:40:31).
kelimeler = ["nehir", "banka", "kiyi", "para"]
# komsuluk (es-dizim) matrisi (illustratif): nehir-banka-kiyi yakin, banka-para yakin
A = np.array([[0, 1, 1, 0],
              [1, 0, 1, 1],
              [1, 1, 0, 0],
              [0, 1, 0, 0]])
derece = A.sum(1)
print("dugumler:", kelimeler)
print("dereceler (komsu sayisi):", derece)
```

```
# (b) Goruntude convolution SABIT bir izgarada kayar (her piksel ayni komsuluk yapisi);
#     molekul/grafikte her dugumun KOMSU SAYISI farkli -> sabit kernel kaymaz.
#     GCN bunu komsulardan mesaj-toplama (aggregate) ile cozer -> Hafta 13.
```

19.13 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H13: Graph Convolutional Networks (GCN) (Konuk: Xavier Bresson)

Attention'dan grafiklere geçiyoruz. Bu hafta Transformer'ı (her kelime her kelimeye), multi-head attention'ı (Q/K/V + causal mask + positional encoding), decoding'i (greedy/beam/top-k, exposure bias) ve SSL'i (word2vec → BERT, masked-LM = denoising) Lewis'le; attention'ın matematiğini, encoder/decoder iskeletini ve karesel maliyeti Canziani'yle kapattık. Hafta 13'te konuk **Xavier Bresson** (Nanyang Technological University, GNN uzmanı) convolution'u düzensiz grafiklere genelleyen GCN'leri (spektral + uzaysal) anlatacak; **Canziani** GCN practicum'u kuracak. Egzersiz 1 (attention elle) ve Egzersiz 5 (cümle → grafik) tam bu derse hazırlar.

Hafta 13: Graph Convolutional Networks (GCN) — Xavier Bresson (Konuk Lecture) + Canziani (Practicum)

Hafta 13'te konuk **Xavier Bresson** (Nanyang Technological University, GNN uzmanı) convolution'u düzensiz grafiklere genelleyen GCN'leri (spektral + uzaysal) anlatacak; **Canziani** GCN practicum'u kuracak. LeCun'un bu haftaki "word2vec = basit GNN" köprüsü doğrudan o derse açılır.

Hafta 13 öncesi yapılacak:

- Egzersiz 1 (attention elle) ve Egzersiz 5 (cümle → grafik) çöz.
- "Attention = küme linear kombinasyonu" ve "BERT = denoising" cümlelerini kendi sözcüklerinle yaz.
- Hafta 3 ConvNet'in "locality + stationarity + compositionality" üçlüsünü hatırla — GCN bunları grafiklere taşır.

19.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Dil modeli	Metin yoğunluk tahmini; zincir kuralı → next-word	Lewis 4:45
CNN → RNN → Transformer	Sınırlı alan → darboğaz/yavaş → her kelime her kelimeye	Lewis 31:43
Multi-head attention	Q/K/V; aynı anda farklı sorular; paralel	Lewis 23:58
Causal mask	Üst-üçgen $-\infty$; geleceğe bakmayı engeller	Lewis 27:08
Positional encoding	Küme → sıra; her konuma embedding	Lewis 29:17

Kavram	Tanım	Hoca / timestamp
Exposure bias / top-k	Kendi hatasını görmez → döngü; k-best örnekle	Lewis 49:54
SSL / cake / BERT	word2vec → BERT; masked-LM = denoising; ölçek	Lewis 1:09
Attention = küme lin. komb.	h = X·a; hard (one-hot) / soft	Canziani 1:45
Key-value store	query → keys → values (lazanya analojisi)	Canziani 19:07
Q/K/V rotasyon	W_Q/W_K/W_V; tek param; $\beta=1/\sqrt{d}$	Canziani 20:44
Encoder/decoder	self + (cross) + Add&Norm; autoencoder iskeleti	Canziani 35:15
Karesel maliyet (A: T×T)	Küme büyüdükçe patlar; Reformer/LSH	Canziani 53:01

19.15 ML Builder Bağlantıları

Geriye köprüler (önkoşul + kurs):

1. **Bağlam kodlayıcı** → Hafta 6 (RNN/LSTM/attention/seq2seq); Transformer = tekrarlamasız hâli.
2. **Softmax / β** → Hafta 11 (attention katsayısı = softmax, sıcaklık).
3. **SSL / cake / BERT denoising** → Hafta 10 (pretext) + Hafta 7-8 (denoising AE).
4. **Encoder/decoder + residual** → Hafta 8 (autoencoder) + Hafta 3 (BatchNorm / 1×1 conv).
5. **Q/K/V = lineer dönüşüm** → Hafta 1-2 (lineer cebir, nokta çarpımı).

İleriye köprüler (production / research):

1. **Transformer** → tüm LLM (GPT/BERT/T5); ViT, protein, multimodal.
2. **SSL + ölçek → GPT-3 / ChatGPT / RLHF (post-2020, KURSTA YOK)**.
3. **word2vec = basit GNN (LeCun)** → Hafta 13 Graph ConvNets.
4. **Karesel maliyet** → sparse / lineer / FlashAttention; uzun-bağlam.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Transformer'ın tüm sırrı **attention**'dır ve attention basittir — bir **kümenin** elemanlarını, Q/K/V benzerlik skorlarından (softmax) gelen katsayılarla harmanlayan bir **linear kombinasyon**. RNN darboğazını ve CNN sınırlı alıcı alanını ortadan kaldırıp her kelimeyi her kelimeye doğrudan bağlar (paralel, ölçeklenir). Dile özel yapı dayatmak yerine düşük-önyargılı modeli çok metinle besle (Lewis); öz-denetimli "boşluğu doldur" (BERT = Hafta 7-8 denoising) etiketsiz metinden öğrenir, pre-train→fine-tune her göreve taşır. LeCun'un köprüsü hazır: word2vec/BERT, kelime eş-dizim grafiği üzerinde **basit bir graph neural net** — bir sonraki haftanın (GCN) kapısı. Post-2020 zirvesi (GPT-3, ChatGPT, RLHF) kursta yoktur ama hepsi 2017 Transformer'ın üstünde durur.

20 Graph Convolutional Networks

İki parçalı hafta — konuk hoca Xavier Bresson (Nanyang Technological University, GNN uzmanı) Lecture'da ConvNet'in başarısını üç doğal-sinyal varsayımına (locality, stationarity, compositionality) dayandırır ve bu varsayımları düzensiz grafiklere taşımanın iki yolunu kurar: template matching (uzaysal) ve convolution teoremi (spektral); uzaysal yolda düğümlerin sırası ve konumu olmadığından şablon eşleştirme anlamsızdır, çözüm tek bir şablon vektörünü tüm komşulukla eşleyip toplamaktır; spektral yolda graf Laplacian'ı pürüzsüzlüğü ölçer, özvektörleri graf Fourier fonksiyonlarını verir, ve ChebNet filtreyi Laplacian'ın polinomu olarak yazarak K-hop yerelleştirilmiş, eigendecomposition gerektirmeyen, gerçek graflar seyrek olduğu için lineer karmaşıklıkta bir convolution elde eder; uzaysal GCN'ler izotropik (tüm komşuya aynı ağırlık: Vanilla, GraphSAGE, GIN) ve anizotropik (komşulara farklı ağırlık: MoNet, GAT, GatedGCN) diye ayrışır, ve dersin doruğunda Bresson Transformer'ın tam-bağlı bir graf üzerinde GCN olduğunu, yani aslında bir küme sinir ağı olduğunu gösterir. Ardından Alfredo Canziani (Practicum) aynı birliği ters yönden kurar: GCN'i okuyunca aslında self-attention olduğunu fark eder, çünkü attention katsayıları hesaplanmak yerine adjacency matrisinden verilir; dereceye bölme, self-connection ve ReLU eklenince izotropik GCN çıkar; Residual Gated GCN'in kenar geçidi izotropik ortalamayı anizotropik ağırlıklı toplama çevirir; ve DGL ile mesaj, reduce, update_all fonksiyonları kurulup graf-seviye sınıflandırma (mean-pool sonra MLP) ile düğüm-seviye yarı-denetimli sınıflandırma (Karate Club, yalnız iki etiket grafın yapısı boyunca yayılır) gösterilir — her iki hoca da aynı sonuca varır: seyreklik yapıdır, ve graph convolution ile attention'ı ayıran tek şey grafın kendisidir.

ℹ Bölüm bilgisi (KONUK Xavier Bresson + Canziani)

- **Konuk Lecture (Xavier Bresson, NTU):** YouTube — Graph Convolutional Networks (Hafta 13 Lecture)
- **Canziani'nin Practicum videosu:** YouTube — Graph Convolutional Networks
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Xavier Bresson (Konuk Lecture, spektral + uzaysal GCN) + Alfredo Canziani (Practicum, GCN = attention, Residual Gated GCN, DGL) — Yann LeCun tartışmaya katılır (özdenetimli öğrenme = grafın sömürülmesi köprüsü)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈30 dk

⚠ **Atıf notu:** Bu haftanın Lecture'ı **LeCun değil, konuk Xavier Bresson** (NTU, graph neural net dünyasının önde gelen isimlerinden). Doğrulama: LeCun (ders sahibi) tartışmada Bresson'a ismiyle hitap eder ("I can tell you, Xavier…", 1:50:06) ve onu "NTU'dan, graph neural net dünyasının önde gelen isimlerinden" diye tanıtır. Lecture quote'ları **— Bresson**; Practicum **— Canziani**; LeCun katıldığında **— LeCun**.

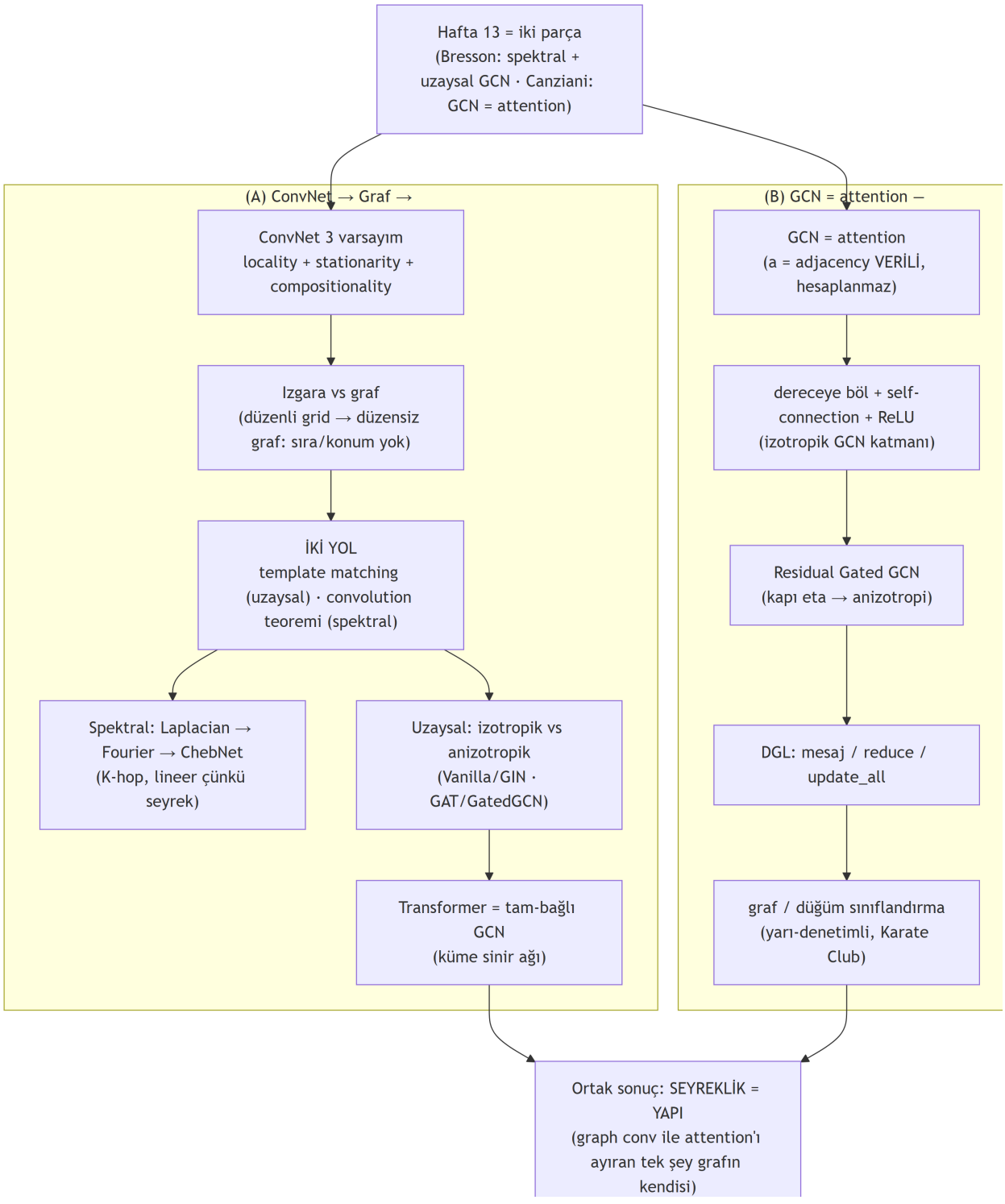
20.1 Bu Derste Ne Var?

Bu hafta iki parça: **konuk hoca Xavier Bresson** (Nanyang Technological University, GNN uzmanı) Lecture'da ConvNet'i düzensiz grafiklere genelleyen **graph convolutional network**'leri (spektral + uzaysal) anlatıyor; **Alfredo Canziani** (Practicum) GCN'in aslında Hafta 12'nin **self-attention**'ı olduğunu (yalnız bağlantılar verili) gösteriyor ve Residual Gated GCN'i DGL koddan kuruyor.

Büyük birleştirici fikir iki yönden gelir: Bresson **"Transformer = tam-bağlı bir graf üzerinde GCN'dir"** der; Canziani **"GCN = self-attention'dır, ama attention katsayıları hesaplanmaz, adjacency matrisinden verilir"** der. İkisi aynı gerçeğe varır: attention ve graph convolution **aynı işlemdir**; onları ayıran tek şey **seyreklik (sparsity)** — yani grafın kendisi.

Bu haftanın üç ana fikri:

1. **Convolution'u grafa genellemenin iki yolu:** template matching (uzaysal) ve convolution teoremi (spektral/Fourier). İkisi de farklı GCN sınıfları doğurur.
2. **Seyreklik = yapı.** Gerçek grafikler (beyin, web, molekül) seyrektir; ChebNet, Laplacian'ın kuvvetleriyle K-hop yerelleştirilmiş, **lineer** karmaşıklıkta convolution yapar.
3. **GCN $\leftrightarrow$ attention birliği.** GCN = adjacency-verili attention (Canziani); Transformer = tam-bağlı GCN (Bresson) $\rightarrow$ attention ve graph conv aynı operasyon.



Builder Notu — İki Hoca, Tek Fikir: Seyreklik = Yapı

Geriye (önkoşul + kurs):

- **ConvNet locality / stationarity / compositionality** → Hafta 3 (doğal sinyaller); Fourier / Lapla-

cian / eigendecomposition → Hafta 3 + 18.06.
- **Self-attention / sets** → Hafta 12 (GCN onun adjacency-verili hâli).
- **Gate / softmax** → Hafta 6 (LSTM gating) + Hafta 12 (attention softmax).

İleriye (production / research):

- GCN → ilaç/protein (AlphaFold), öneri sistemleri, sosyal ağ, fizik.
- Anisotropic gate + edge features → Graph Transformers (post-2020, §İleriye Köprü).

Tek cümleyle: Graph convolution, ConvNet'in locality/stationarity/compositionality varsayımlarını düzensiz grafiklere taşır — ya template matching (uzaysal GCN) ya da spektral teori (Laplacian/Fourier, ChebNet) ile; ve özünde Hafta 12'nin attention'ıyla aynıdır: GCN = adjacency-verili attention, Transformer = tam-bağlı GCN.

20.2 (Bresson — Konuk) ConvNet'ten Graf'a: Convolution'un İki Tanımı

Bresson ConvNet'in başarısının üç **varsayıma** dayandığını hatırlatır (Hafta 3'ün doğal-sinyal ilkeleri):

> "the main assumption is that your data is compositional… it is formed of patterns that are local (locality)… you have stationarity… and it is hierarchical." — Bresson, 2:54

Görüntü/ses/metin **düzenli ızgaralardır** (2D/1D grid); ızgarada convolution iyi tanımlı ve hızlıdır. Ama sosyal ağ, beyin bağlantısı, molekül **ızgara değildir** — düzensiz **graflardır** (köşeler V, kenarlar E, adjacency A). Şekil 20.1 bu çelişkiyi iki panelde gösterir: solda düzenli ızgarada her pikselin sabit komşuluğu ("yukarı/aşağı/sağ/sol") ve paylaşılan bir 3×3 kernel'i varken, sağda düzensiz grafta düğümlerin sırası, konumu ve komşu sayısı değişkendir. Convolution'u grafa nasıl genelleriz? İki yol:

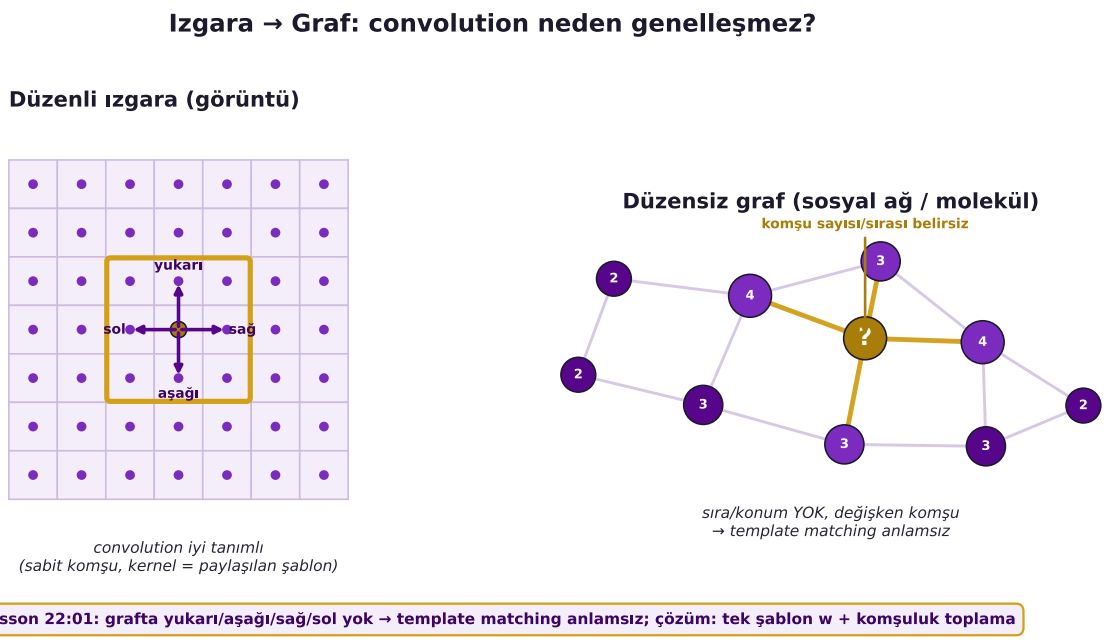
1. **Template matching (uzaysal):** kernel'i kaydır, nokta çarpımı al. Ama grafta **iki sorun**: düğümlerin **sırası/konumu yok** (hangi komşu "sağ üst"?), komşu sayısı **değişken**.

   > "on a graph you have no notion of where is up, where is down, where is right, where is left… so this matching generally has no meaning." — Bresson, 22:01

2. **Convolution teoremi (spektral):** convolution = Fourier uzayında nokta çarpımı. Ama grafta Fourier dönüşümünü yeniden tanımlamak gerek — ve hızlı olmalı.

Builder Notu — Izgaradan Grafa

Geriye (Hafta 3): locality/stationarity/compositionality = Hafta 3 doğal sinyaller; ızgara convolution = Hafta 3 ConvNet; template matching = korelasyon.
İleriye: "Düzenli ızgaradan düzensiz grafa" = geometric deep learning'in temel motivasyonu; molekül/protein/sosyal-ağ.



Şekil 20.1: Izgara → Graf: convolution neden genelleşmez? SOL 'Düzenli ızgara (görüntü)': 7x7 piksel ızgarası + merkez piksel üzerinde 3x3 kernel penceresi; yukarı/aşağı/sağ/sol komşular SABİT ve bellidir, convolution iyi tanımlı (kernel = paylaşılan şablon). SAĞ 'Düzensiz graf (sosyal ağ/molekül)': networkx ile değişken-dereceli düzensiz bir graf; odak düğümde '?' işareti ve 'komşu sayısı/sırası belirsiz' rozeti — düğümlerin sırası/konumu yok, komşu sayısı değişken, template matching anlamsız (çözüm: tek şablon w + komşuluk toplama, Bresson 22:01).

20.3 (Bresson — Konuk) Spektral GCN: Laplacian, Fourier ve ChebNet

Spektral yol için merkez operatör **graf Laplacian'ıdır** (normalize):

$$\Delta = I - D^{-1/2} A\, D^{-1/2}$$

(D = derece köşegen matrisi.) Laplacian, bir fonksiyonun graf üzerindeki **pürüzsüzlüğünün ölçüsüdür** (h_i ile komşularının ortalaması farkı). Şekil 20.2 bunu iki topluluklu bir grafta GERÇEK rakamla gösterir: yavaş-değişen (pürüzsüz) bir sinyalde kuadratik form $h^\top \Delta h$ küçük çıkar, komşular arasında zıt işaret alan (salınan) bir sinyalde büyük çıkar. **Özayrışımı** ($\Delta = \Phi \Lambda \Phi^\top$) graf **Fourier fonksiyonlarını** (özvektörler Φ) ve **spektrumu** (özdeğerler Λ) verir. Şekil 20.3 bu modları bir yol grafında resmeder: en küçük özdeğere ait özvektör neredeyse sabittir (pürüzsüz mod), en büyük özdeğere ait olan komşular arası kutuplaşır (salınan mod); graf Fourier dönüşümü = fonksiyonu bu özvektörlere izdüşürmek.

- **Vanilla spektral GCN** (Bruna, Zaremba, Szlam, **LeCun** 2014): spektral filtreyi backprop ile öğren. Sorun: uzaysal yerelleştirme garantisi yok, N parametre, $O(N^2)$ (Φ yoğun).
- **Pürüzsüz filtreler (spline):** frekansta pürüzsüz = uzayda yerel (Heisenberg/Parseval) → K parametre. Ama hâlâ $O(N^2)$.
- **ChebNet** (Defferrard, **Bresson**, 2016): spektral filtre = **Laplacian'ın polinomu** ($\sum_k w_k \Delta^k$). Δ^k tam **K-hop yerelleştirilmiş**; özyinelemeli $x_k = \Delta \cdot x_{k-1}$. Şekil 20.4 bir "ısı kaynağını" alıp Laplacian'ı tekrar tekrar uygulayarak desteğin her adımda bir hop genişlediğini GERÇEK rakamla gösterir. Karmaşıklık = (kenar × K) = seyrek grafikler için **LİNEER** — eigendecomposition gerekmez!

  "every natural graph is usually sparse, because sparsity is structure." — Bresson, 56:42

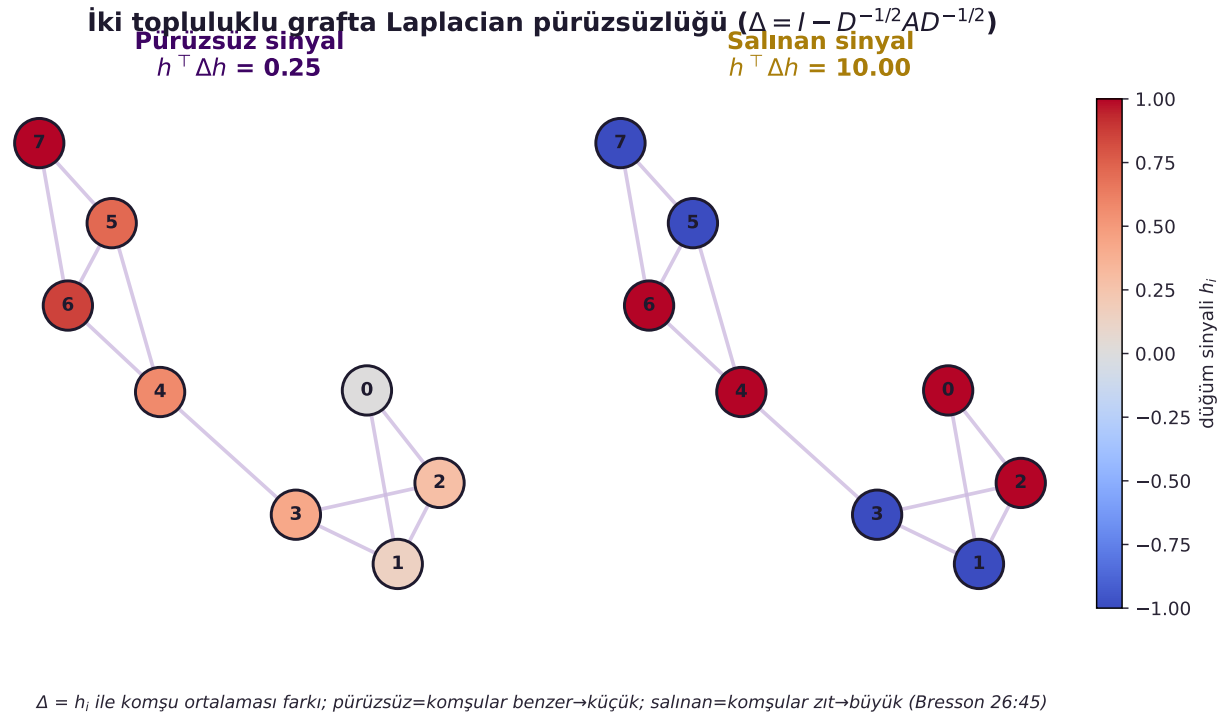
ChebNet "spektral" denilse de hesaplama tamamen uzaysaldır (Laplacian çarpımları). MNIST sanity-check: %99, lineer karmaşıklık.

> Builder Notu — Laplacian = Pürüzsüzlük, Fourier ve ChebNet K-hop
>
> **Geriye (Hafta 3 + 18.06):** Fourier = Hafta 3 spektral görüş; Laplacian özayrışımı = 18.06 eigendecomposition; pürüzsüz↔yerel = belirsizlik ilkesi.
> **İleriye:** Spektral GCN → graf sinyal işleme, spektral clustering; ChebNet polinomu = mesaj-geçişin erken hâli.

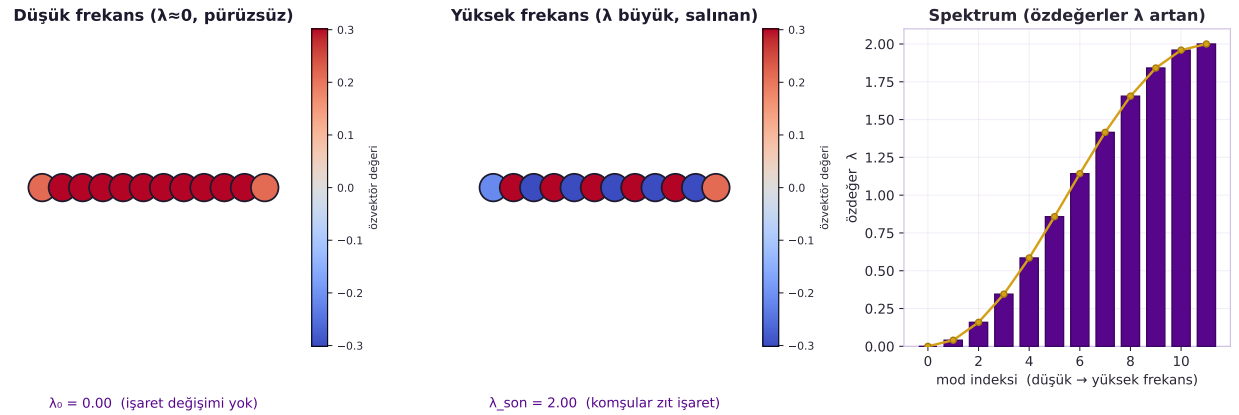
20.4 (Bresson — Konuk) Uzaysal GCN: Isotropic, Anisotropic ve Transformer = Tam-Bağlı GCN

Uzaysal yol template matching'i düğüm-yeniden-parametrizasyonuna **değişmez** kılar: tek şablon vektörü w'yi tüm komşularla eşleştir (komşuluk üzerinde topla). Matris formu: $H^{l+1} = f(AH^l W)$.

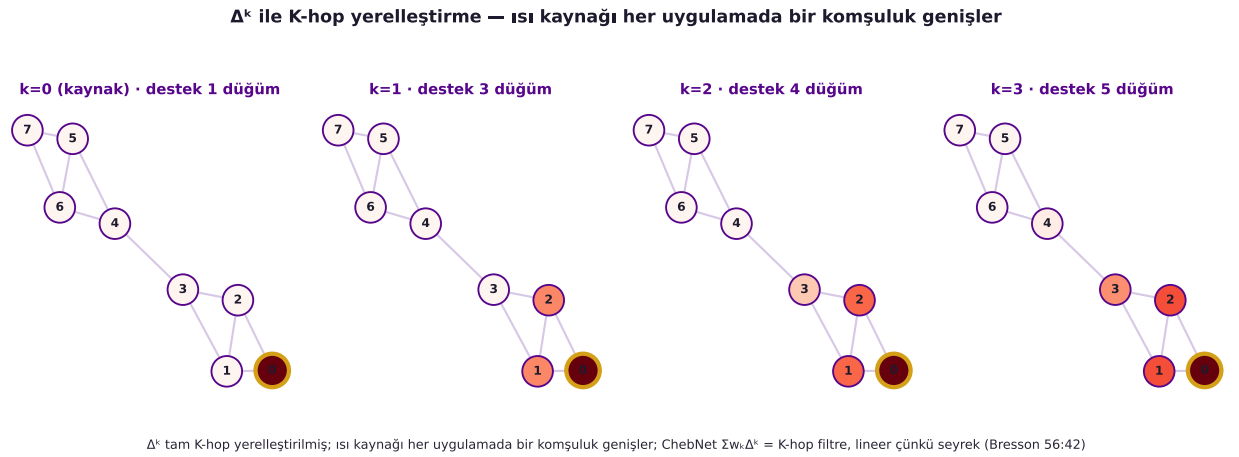
- **Isotropic GCN** (tüm komşulara aynı ağırlık): Vanilla GCN (Kipf-Welling 2016), GraphSAGE (merkez + komşuluk ayrı şablon), GIN (graf izomorfizmi). Komşuluk ortalaması; weight sharing; graf boyutundan bağımsız.



Şekil 20.2: İki topluluklu grafta Laplacian pürüzsüzlüğü — GERÇEK kuadratik form. A = community_graph_adj() (8 düğüm, 2 topluluk + köprü). SOL 'Pürüzsüz sinyal': düğümler yavaş-değişen bir sinyalle (coolwarm) renklendirilir, komşular benzer, $h^T \Delta h$ KÜÇÜK. SAĞ 'Salınan sinyal': düğümler alternatif +1/−1 alır, komşular zıt, $h^T \Delta h$ BÜYÜK. Başlıklarda motordan gelen GERÇEK değerler yazılı. Sezgi: Δh = h_i ile komşu ortalaması farkı; pürüzsüz=komşular benzer→küçük, salınan=komşular zıt→büyük (Bresson 26:45).



Şekil 20.3: GERÇEK graf Fourier modları — yol grafı (path_graph_adj(12)) için Laplacian özvektörleri. SOL 'Düşük frekans (lambda≈0, pürüzsüz)': en küçük özdeğere ait özvektör neredeyse sabittir (işaret değişimi yok). ORTA 'Yüksek frekans (lambda büyük, salınan)': en büyük özdeğere ait özvektör komşular arası zıt işaret alır (kutuplaşma). SAĞ 'Spektrum': özdeğerler lambda artan sırada bar+çizgi. Sezgi: Laplacian özvektörleri = graf Fourier fonksiyonları (Bresson 30:03); düşük özdeğer=pürüzsüz mod, yüksek=salınan; graf Fourier dönüşümü = sinyalin özvektörlere izdüşümü.



Şekil 20.4: GERÇEK K-hop yerelleştirme (ChebNet) — Δ^k ile ısı yayılımı. A = community_graph_adj(), kaynak = düğüm 0. 4 panel AYNI layout: her panelde düğümler $|\Delta^k e|$ büyüklüğüyle (Reds) renklendirilir, kaynak düğüm gold kenarla işaretli. Destek HER uygulamada bir hop genişler (panel başlıklarında GERÇEK aktif-düğüm sayısı). Sezgi: Δ^k tam K-hop yerelleştirilmiş; ChebNet $\Sigma w_k \Delta^k$ = K-hop filtre, lineer çünkü seyrek (Bresson 56:42).

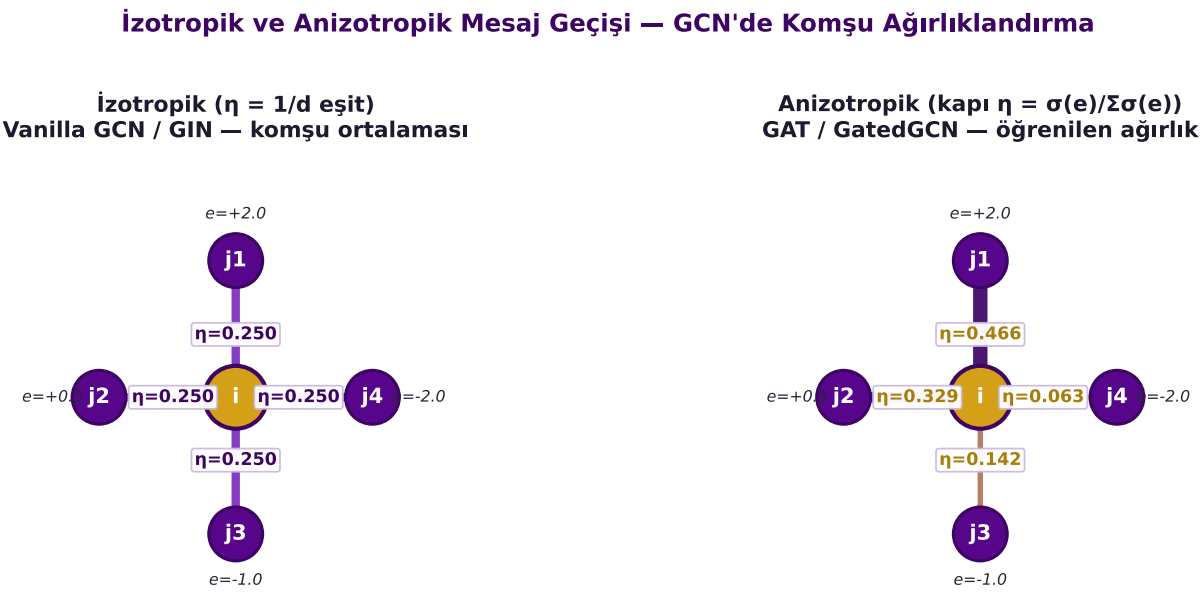
- **Anisotropic GCN** (komşulara **farklı** ağırlık): MoNet (GMM), **GAT** (Velickovic-Bengio, komşuluk üzerinde softmax attention), **GatedGCN** (Bresson-Laurent 2017, kenar-geçidi + açık kenar öznitelikleri). Sosyal ağda "republican vs democrat"ı aynı işlememelisin → anizotropi.

Şekil 20.5 bu ayrımı GERÇEK rakamla gösterir: bir merkez düğüm + 4 komşu üzerinde, izotropik durumda tüm kenarlar eşit ağırlıklıdır ($\eta = 1/d$, "bulanıklaştır"), anizotropik durumda kapı $\eta_{ij} = \sigma(e_{ij}) / \sum_k \sigma(e_{ik})$ her kenara farklı ağırlık verir (büyük öznitelikli kenar = kalın).

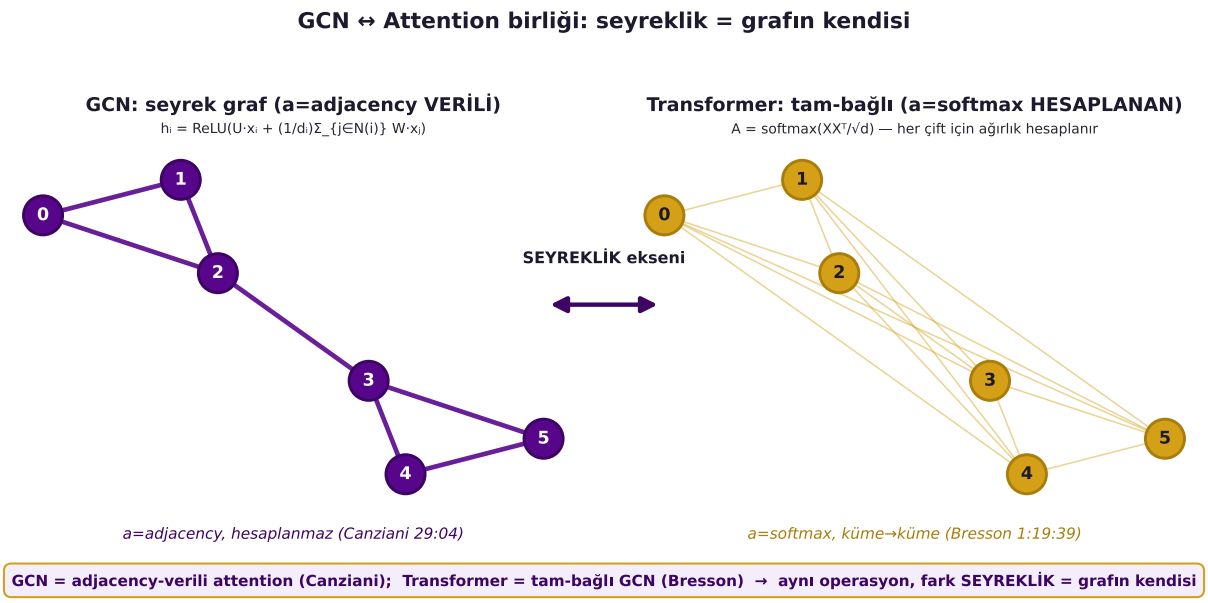
Dersin doruğu — Hafta 12'ye köprü:

> "a standard transformer is actually a special case of graph convolutional nets when the graph is fully connected… transformers are set neural networks." — Bresson, 1:19:39

GAT denkleminde komşuluğu **tüm graf** yaparsan (tam-bağlı), tam olarak Hafta 12'nin Transformer'ını elde edersin. Şekil 20.6 bu birliği dersin doruk figürü olarak iki panelde gösterir: solda seyrek graf (a = adjacency VERİLİ), sağda aynı düğümlerin tam-bağlı hâli (a = softmax HESAPLANAN), ortada "seyreklik ekseni". Tam-bağlı olunca "graf" demek anlamsızlaşır (seyreklik = grafı ilginç yapan şey) — ona **küme** demek daha doğru. LeCun ekler: tüm öz-denetimli öğrenme bir graf yapısı (kelime eş-dizim, benzerlik grafiği) sömürür — "there is a very very strong connection between self-supervised learning and the graph view of a training set." — LeCun, 1:43:49



Şekil 20.5: GERÇEK izotropik vs anizotropik komşu toplama (Residual Gated GCN, Canziani 27:53). Bir merkez düğüm i + 4 komşu (yıldız graf). SOL 'İzotropik (eta = 1/d eşit)': tüm kenarlar aynı kalınlık/renk, komşuluk ortalaması (Vanilla/GIN, 'bulanıklaştır'). SAĞ 'Anizotropik (kapı eta = sigma(e)/Σsigma(e))': gate_eta(edge_feats) ile kenar kalınlıkları FARKLI; büyük kenar-özniteliği → kalın kenar (GAT/GatedGCN). Kenar üstündeki eta değerleri motordan GERÇEK. Sezgi: kapı eta backprop ile hangi komşunun önemli olduğunu öğrenir — Hafta 6 LSTM kapı mekanizmasının grafiksel akrabası.



Şekil 20.6: GCN ↔ Attention birliği: seyreklik = grafın kendisi (DERSİN DORUĞU). İki panel AYNI 6 düğüm, AYNI layout. SOL ‘GCN: seyrek graf (a=adjacency VERİLİ)’: birkaç kenarlı seyrek graf; a = adjacency, hesaplanmaz (Canziani 29:04). SAĞ ‘Transformer: tam-bağlı (a=softmax HESAPLANAN)’: aynı düğümler ama HER düğüm her düğüme bağlı (soluk yoğun kenarlar); a = softmax, küme→küme (Bresson 1:19:39). ORTADA çift-yönlü ‘SEYREKLİK ekseni’ oku. Sezgi: GCN = adjacency-verili attention (Canziani); Transformer = tam-bağlı GCN (Bresson), aynı operasyon, fark SEYREKLİK = grafın kendisi.

Builder Notu — Isotropic vs Anisotropic ve Transformer = Tam-Bağlı GCN

Geriye (Hafta 12 + 6): Anizotropi gate = Hafta 6 LSTM gating + Hafta 12 attention softmax; Transformer = tam-bağlı GCN = Hafta 12; SSL-graf = Hafta 10 + 12 (word2vec=GNN).
İleriye: GAT/GatedGCN → Graph Transformers (post-2020); izotropi-anizotropi = GNN tasarımının ana sınıflandırması.

20.5 (İleriye Köprü) Graph Transformers, AlphaFold, Geometric DL (post-2020) — KURSTA YOK

Bresson (Mart 2020) GCN'i template matching + spektral teoriyle kurar; GAT/GatedGCN'i en güçlü anizotropik modeller olarak verir ve "düğümler için positional encoding nasıl bulunur = en önemli açık soru" der. Aşağıdakiler DLSP20'den **sonra** geldi, kursta **YOKTUR**:

İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **Graph Transformers** (Graphormer 2021, GraphGPS 2022) ve **Laplacian positional encoding** — Bresson'un "düğüm positional encoding açık sorusunun" cevabı; "Transformer = tam-bağlı GCN" fikrinin olgunlaşması.
- **AlphaFold 2** (2021) — protein katlanmasını attention/graf ile çözer; Bresson'un "protein yapısı çok zor ama öğrenebiliriz" probleminin pratik zaferi.
- **Geometric Deep Learning** (Bronstein ve ark., 2021) — CNN/GNN/Transformer'ı tek simetri/değişmezlik çerçevesinde birleştiren program.

Bunlar kurs terimi gibi eklenmez; "graf = yapı, attention = tam-bağlı graf" temelinin nereye vardığını göstermek için anılır.

Builder Notu — Açık Soru: Düğüm Positional Encoding

Geriye (Hafta 12-13): Graph Transformers = bu haftanın anizotropik GAT + Hafta 12 Transformer'ının birleşimi; positional encoding = Bresson'un açık sorusu.
İleriye: Geometric DL, tüm derin öğrenmeyi "veri üzerindeki simetriler" diliyle birleştirir — CNN (öteleme), GNN (permütasyon), Transformer (küme).

20.6 Geçiş: Bresson'dan Canziani'ye

Bresson convolution'u grafa **neden** ve **nasıl** genellediğimizi (spektral + uzaysal) anlattı ve "Transformer = tam-bağlı GCN" köprüsünü kurdu. Şimdi **Canziani** aynı birliği ters yönden gösterir: geçen haftanın self-attention'ından başlayıp, attention katsayılarını adjacency matrisiyle **değiştirerek** GCN'i türetir — ve Residual Gated GCN'i DGL ile koddan kurar.

20.7 (Canziani) GCN = Attention, Ama Bağlantılar Verili

Canziani GCN literatürünü okuyunca şaşkınlığını paylaşır: bu zaten attention!

> "graph convolutional networks… I read them and oh, it's actually the same thing… it looks like attention to me." — Canziani, 10:50

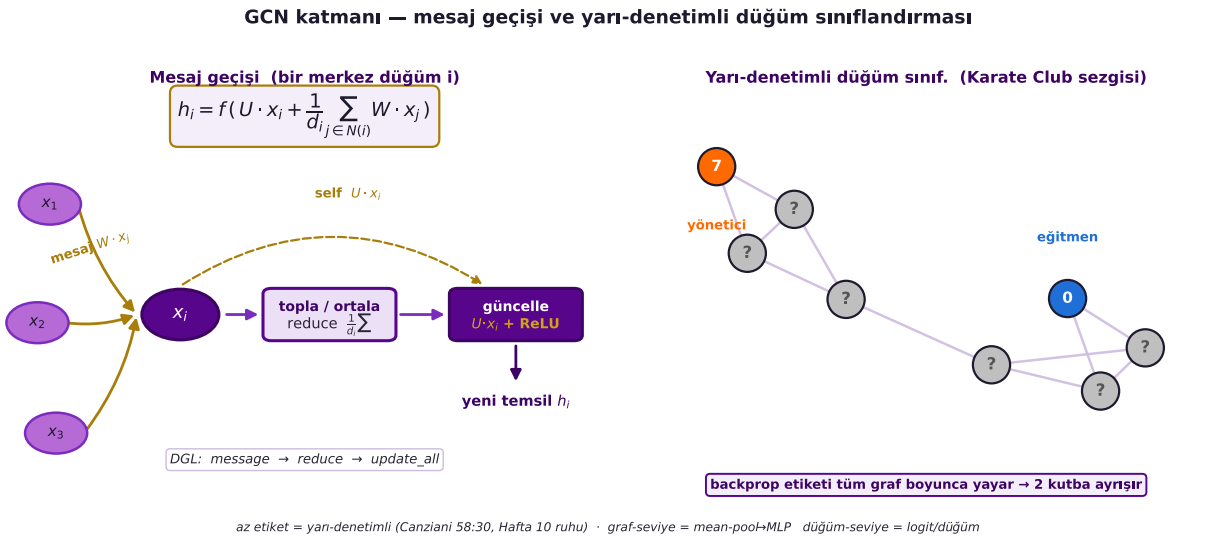
Hatırlatma (Hafta 12): self-attention'da $h = X \cdot a$, ve $a = \text{softmax}(\text{skorlar})$ sana **kime bakacağını** söyler (hesaplanır). GCN'de a **= adjacency vektörüdür** — kendine gelen kenar varsa 1, yoksa 0 (verilir, hesaplanmaz). Gerisi otomatik:

- Dereceye böl (a'daki bir'lerin sayısı d): h ölçeği komşu sayısıyla büyümesin → komşuluk **ortalaması**.
- Öğrenilen rotasyon W ekle, **self-connection** ($U \cdot x$) ekle, nonlinearity (ReLU) ekle.

$$h_i = f\left(Ux_i + \frac{1}{d_i}\sum_{j \in N(i)} Wx_j\right)$$

Çok katman = yığ (her katman hâlâ bir **kümedir**, bağlantılar adjacency'den verili). Şekil 20.7 solda bu mesaj-geçiş katmanını (komşulardan gelen mesajları topla/ortala, self-connection ile güncelle) bir şema olarak, sağda ise yarı-denetimli düğüm sınıflandırmasını gösterir.

> "the only difference between attention and this is that these connections are given to you by the adjacency matrix instead of being computed with attention." — Canziani, 29:04



Şekil 20.7: GCN katmanı — mesaj geçişi ve yarı-denetimli düğüm sınıflandırması. SOL 'Mesaj geçişi (bir merkez düğüm i)': $h_i = f(U \cdot x_i + (1/d_i)\Sigma\, W \cdot x_j)$ şeması — komşulardan mesaj ($W \cdot x_j$) → topla/ortala (reduce $1/d_i\ \Sigma$) → güncelle ($U \cdot x_i$ + ReLU) → yeni temsil h_i; DGL: message → reduce → update_all. SAĞ 'Yarı-denetimli düğüm sınıf. (Karate Club sezgisi)': iki topluluklu graf, yalnız 2 etiketli düğüm (0=eğitmen mavi, 7=yönetici turuncu), gerisi gri '?'; backprop etiketi tüm graf boyunca yayar → 2 kutba ayrışır. Sezgi: az etiket = yarı-denetimli (Canziani 58:30, Hafta 10 ruhu); graf-seviye = mean-pool→MLP, düğüm-seviye = logit/düğüm.

Builder Notu — GCN = Verili Attention

Geriye (Hafta 12): $h = X \cdot a$ = Hafta 12 attention; a = adjacency (verili) vs softmax (hesaplanan); dereceye bölme = $\sqrt{d}$ ölçeklemesi akrabası; self-connection + ReLU = standart katman.
İleriye: "Attention = öğrenilen graf, GCN = verili graf" ekseni, tüm geometric DL'in birleştirici görüşü.

20.8 (Canziani) Residual Gated GCN: Kenar Geçidi (η) ile Anisotropy

Uyguladıkları model **Residual Gated GCN** (Bresson & Laurent): kenarların da **öznitelikleri** (e_j) vardır. Vanilla GCN tüm komşuları aynı işler (izotropik, "her şeyi bulanıklaştır"); buradaki **kapı (gate) η** bunu kırar — gelen her mesajı, o kenarın temsiline göre **modüle eder**:

> "we're not going to be just averaging all these incoming values, we're going to be weighting them, modulating them based on what we think might be relevant." — Canziani, 27:53

Kapı, kenarların sigmoid'inin normalize hâlidir (soft-attention'ın kenar versiyonu):

$$\eta_{ij} = \frac{\sigma(e_{ij})}{\sum_{k \in N(i)} \sigma(e_{ik})}$$

Bu, izotropik ortalamayı **anizotropik** ağırlıklı toplama çevirir — hangi komşunun önemli olduğunu ağ backprop ile öğrenir. (Aynı kapı $\eta_{ij} = \sigma(e_{ij}) / \sum_k \sigma(e_{ik})$, daha önce Şekil 20.5'te merkez düğüm η kapısı olarak GERÇEK rakamla resmedilmişti: izotropik eşit-ağırlık vs anizotropik öğrenilen-ağırlık.) Residual + BatchNorm ile sarılır.

Builder Notu — Kenar Kapısı η

Geriye (Hafta 6 + 11 + 12): Kapı η = Hafta 6 LSTM gating + Hafta 12 softmax attention; sigmoid normalize = soft argmax akrabası; residual+BatchNorm = Hafta 3 + 11; izotropik ortalama = bilgi yok etme (blur).
İleriye: Kenar-öznitelikli anizotropik GCN = molekül (bağ türü), trafik, öneri; GatedGCN benchmark'larda izotropikleri geçer.

20.9 (Canziani) DGL Kodu, Graf vs Düğüm Sınıflandırma (Semi-Supervised)

Kod **DGL** (Deep Graph Library) ile: **mesaj fonksiyonu** (kenar UDF — source/destination/edge öznitelikleri) + **reduce fonksiyonu** (düğüm UDF — gelen mesajların mailbox'ı) + `update_all`. Veri: **MiniGCDataset** — 8 graf sınıfı (circle, star, wheel, lollipop, hypercube, grid, clique, circular ladder). Görev: **graf sınıflandırma** (değişken düğüm sayısı). Düğüm özniteliklerini **mean-pool** → MLP → 8-yönlü sınıflandırıcı.

İki görev tipi (her ikisi Şekil 20.7'in sağ panelinde resmedildi):

- **Graf-seviye:** tüm düğümleri ortala → MLP (graf başına tek etiket).

- **Düğüm-seviye:** her köşe için logit; örn. **Zachary Karate Club** (eğitmen=düğüm 0, yönetici=düğüm 33, yalnız 2 etiket) → **yarı-denetimli (semi-supervised)** düğüm sınıflandırma: etiketleri graf yapısı boyunca yay; temsiller çekilip ayrışır.

> "you only have a few labels… this is called semi-supervised learning." — Canziani, 58:30

Canziani'nin son vurgusu Bresson'unkiyle aynı: **seyreklik yapıdır** — herkes herkese bağlıysa adjacency tüm-bir olur ve attention'a dönersin; seyrek olunca graf "kimin kiminle bağlı olduğunu" söyler.

Builder Notu — Seyreklik = Yapı

Geriye (Hafta 10 + 12): Mean-pool = değişken-boyut çözümü (Hafta 12 küme); semi-supervised = Hafta 10 (az etiket, SSL); $\sqrt{\text{boyut}}$ ölçekleme = Hafta 12 $\sqrt{d}$.
İleriye: Düğüm/kenar/graf görevleri = GNN'in dört temel problemi; mesaj-geçiş = tüm modern GNN kütüphanelerinin (DGL, PyG) çekirdeği.

20.10 Bu Dersin Özeti

1. **Convolution'u grafa genelle (Bresson):** template matching (uzaysal) — sıra/konum yok sorunu; convolution teoremi (spektral) — graf Fourier.
2. **Spektral GCN (Bresson):** Laplacian $\Delta = I - D^{-1/2}AD^{-1/2}$ (pürüzsüzlük); Fourier = özvektörler; **ChebNet** = Laplacian polinomu, K-hop, lineer (seyreklik = yapı).
3. **Uzaysal GCN (Bresson):** isotropic (Vanilla/GraphSAGE/GIN) vs anisotropic (GAT/GatedGCN); **Transformer = tam-bağlı GCN**.
4. **GCN = attention (Canziani):** $h = X \cdot a$; a = adjacency (verili) vs softmax (hesaplanan); dereceye böl + self-connection + ReLU.
5. **Residual Gated GCN (Canziani):** kenar geçidi $\eta = \sigma(e)/\sum \sigma(e)$ → anizotropi; izotropik ortalamayı ağırlıklı toplama çevirir.
6. **DGL + görevler (Canziani):** mesaj/reduce/update_all; graf sınıflandırma (mean-pool→MLP); düğüm sınıflandırma (Karate Club, semi-supervised).
7. **Post-2020 (KURSTA YOK):** Graph Transformers + Laplacian PE, AlphaFold 2, Geometric DL.

! Tek Bir Cümle

Graph convolution, ConvNet'in locality/stationarity/compositionality varsayımlarını düzensiz grafiklere taşır — ya template matching (uzaysal GCN: izotropik/anizotropik) ya da spektral teori (Laplacian/Fourier, ChebNet, lineer çünkü graflar seyrektir) ile; ve özünde Hafta 12'nin attention'ıyla aynı operasyondur: Canziani'ye göre GCN = adjacency-verili attention, Bresson'a göre Transformer = tam-bağlı GCN — ikisini ayıran tek şey seyrekliktir (grafın kendisi).

20.11 Kontrol Soruları

i Soru 1: Convolution'u grafa genellemenin iki yolu nedir? Template matching grafta neden doğrudan çalışmaz?

Cevap: (1) **Template matching (uzaysal):** kernel'i kaydır, nokta çarpımı → uzaysal GCN. (2) **Convolution teoremi (spektral):** Fourier uzayında nokta çarpımı → spektral GCN. Template matching grafta doğrudan çalışmaz çünkü düğümlerin **sırası/konumu yoktur** ("yukarı/aşağı/sağ/sol" yok, Bresson 22:01) — hangi komşunun şablonun hangi hücresine karşılık geldiği belirsiz; ayrıca komşu sayısı **değişken**. Çözüm: tek şablon w (yeniden-parametrizasyona değişmez) + komşuluk üzerinde toplama.

i Soru 2: Graf Laplacian nedir, neyi ölçer? ChebNet neden lineer karmaşıklıkta çalışır?

Cevap: Normalize Laplacian $\Delta = I - D^{-1/2}AD^{-1/2}$; bir fonksiyonun graf üzerindeki **pürüzsüzlüğünü** ölçer (h_i ile komşu ortalaması farkı). Özayrışımı ($\Delta = \Phi\Lambda\Phi^\top$) graf **Fourier fonksiyonlarını** (özvektörler) ve spektrumu verir. Vanilla spektral GCN $O(N^2)$ (Φ yoğun). **ChebNet** filtreyi Laplacian'ın **polinomu** ($\sum_k w_k\Delta^k$) olarak yazar: Δ^k tam K-hop yerelleştirilmiş, özyinelemeli $x_k = \Delta \cdot x_{k-1}$; karmaşıklık = kenar $\times$ K. Graflar **seyrek** olduğundan (kenar $\approx$ sabit$\times N$) bu **lineerdir** — eigendecomposition hiç gerekmez (Bresson 56:42).

i Soru 3: "Transformer=tam-bağlı GCN" ne demek? "GCN=attention" ile nasıl aynı gerçeğe varır?

Cevap: Bir GAT denkleminde komşuluğu **tüm graf** yaparsan (her düğüm her düğüme = tam-bağlı), Hafta 12'nin Transformer'ını elde edersin (Bresson 1:19:39). Tersinden Canziani: self-attention'da $h = X \cdot a$, a hesaplanır; GCN'de a **= adjacency** (verili, Canziani 29:04). İkisi aynı operasyon — **fark seyrekliktir**: GCN seyrek (graf verili), Transformer tam-bağlı (her şey her şeye, "küme"). Tam-bağlı olunca seyreklik kaybolur, ona graf değil **küme** demek doğrudur.

i Soru 4: Isotropic vs anisotropic GCN farkı nedir? Residual Gated GCN'deki kapı η ne yapar?

Cevap: Isotropic GCN tüm komşulara **aynı** ağırlık (komşuluk ortalaması — Vanilla/GraphSAGE/GIN); "her şeyi bulanıklaştırmak" gibi. **Anisotropic** komşulara **farklı** ağırlık (GAT, GatedGCN) — sosyal ağda farklı grupları aynı işlememek için. **Residual Gated GCN**'in kapısı $\eta = \sigma(e_{ij}) / \sum \sigma(e_{ik})$ (kenar temsillerinin normalize sigmoid'i, Canziani 27:53), gelen mesajı kenara göre **modüle eder** → izotropik ortalamayı anizotropik ağırlıklı toplama çevirir; hangi komşunun önemli olduğunu backprop öğrenir. Hafta 6 LSTM gating + Hafta 12 attention akrabası.

i Soru 5: Graf-seviye ve düğüm-seviye sınıflandırma nasıl farklıdır? Karate Club neden "semi-supervised"dir?

Cevap: Graf-seviye: tüm düğümleri **mean-pool** → MLP → graf başına tek etiket (örn. MiniGCD 8 graf tipi). **Düğüm-seviye:** her köşe için ayrı logit (mean-pool yok) → düğüm başına etiket. **Karate Club** yarı-denetimlidir çünkü yalnız **2 etiket** var (eğitmen=0, yönetici=33); kayıp yalnız bu iki düğümde ama backprop bilgiyi **tüm graf yapısı boyunca yayar** (Canziani 58:30) → etiketsiz düğümlerin temsilleri iki kutba çekilip ayrışır. Hafta 10'un "az etiketle öğren" ruhu.

20.12 Egzersizler

Egzersiz 1 (GCN = attention). Hafta 12 self-attention'ı ($h = X \cdot a$, $a = \text{softmax}(X^\top x)$) yaz; sonra a'yı adjacency vektörüyle (1/0) değiştirip dereceye bölerek vanilla GCN'i türet. "Hesaplanan a" ile "verili a" farkı, GCN ile Transformer'ı nasıl ayırır?

```
import numpy as np

# Hafta 12 self-attention: a HESAPLANIR (softmax skorlari)
X = np.array([[1.0, 0.0],   # x1
              [0.0, 1.0],   # x2
              [0.8, 0.6]])  # x3
x = X[0]                    # sorgu = x1

def softmax(z):
    e = np.exp(z - z.max())
    return e / e.sum()

a_attn = softmax(X @ x)             # HESAPLANAN attention katsayilari
h_attn = a_attn @ X                 # h = X^T a (linear kombinasyon)

# GCN: a VERILIR (adjacency satiri), sonra dereceye bolunur (komsuluk ortalamasi)
a_adj = np.array([1.0, 1.0, 0.0])  # x1'in komsulari: x2 var, x3 yok (VERILI)
d = a_adj.sum()                     # derece
h_gcn = (a_adj / d) @ X             # komsuluk ortalamasi
print("attention h:", np.round(h_attn, 3), " (a HESAPLANAN)")
print("gcn h      :", np.round(h_gcn, 3), " (a VERILI = adjacency)")
# Fark: attention a'yi softmax ile HESAPLAR (kime bakacagini ogrenir);
# GCN a'yi adjacency'den ALIR (kim kime bagli VERILI). Transformer = tam-bagli GCN.
```

Egzersiz 2 (Laplacian pürüzsüzlük). Küçük grafta (5 düğüm) $\Delta = I - D^{-1/2}AD^{-1/2}$'yi elle hesapla. Sabit ve hızlı salınan sinyale uygula; pürüzsüz sinyalde Δh küçük, salınanda büyük olduğunu göster. Neden "pürüzsüzlük ölçüsü"?

```
import numpy as np

# 5 dugumlu yol grafi (zincir 0-1-2-3-4)
A = np.zeros((5, 5))
for i in range(4):
    A[i, i + 1] = A[i + 1, i] = 1.0
d = A.sum(1)
Dinv = np.diag(1.0 / np.sqrt(d))
Delta = np.eye(5) - Dinv @ A @ Dinv   # normalize Laplacian

h_smooth = np.array([1.0, 1.0, 1.0, 1.0, 1.0])   # sabit (puruzsuz)
h_osc    = np.array([1.0, -1.0, 1.0, -1.0, 1.0]) # salinan
print("puruzsuz hDh:", round(float(h_smooth @ Delta @ h_smooth), 3))  # ~0 (kucuk)
```

```
print("salinan  hDh:", round(float(h_osc @ Delta @ h_osc), 3))        # BUYUK
# Dh = h_i ile komsu ortalamasi farki; puruzsuzde komsular benzer -> kucuk,
# salinada komsular zit -> buyuk => "puruzsuzluk olcusu".
```

Egzersiz 3 (K-hop yerelleştirme). Bir düğüme "ısı kaynağı" (1, gerisi 0) koy. Δ'yı 1/2/3 kez uygula; desteğin her seferinde bir hop genişlediğini göster. ChebNet'in $\sum_k w_k \Delta^k$'sı neden "tam K-hop yerelleştirilmiş"?

```
import numpy as np

# 6 dugumlu yol grafi, isi kaynagi = dugum 0
A = np.zeros((6, 6))
for i in range(5):
    A[i, i + 1] = A[i + 1, i] = 1.0
d = A.sum(1)
Delta = np.eye(6) - np.diag(1/np.sqrt(d)) @ A @ np.diag(1/np.sqrt(d))

e = np.zeros(6); e[0] = 1.0           # isi kaynagi (sadece dugum 0)
cur = e.copy()
for k in range(4):
    aktif = int(np.sum(np.abs(cur) > 1e-9))
    print(f"k={k}: destek {aktif} dugum")   # destek her adimda bir hop genisler
    cur = Delta @ cur
# Delta^k bir dugumden tam k-hop uzaktaki dugumlere erisir -> Sum w_k Delta^k = K-hop FILTRE.
# Seyrek grafta (kenar ~ sabit*N) maliyet kenar*K = LINEER (eigendecomposition yok).
```

Egzersiz 4 (Anisotropy gate). Bir düğümün 3 komşusu + kenar öznitelikleri olsun. $\eta = \sigma(e)/\sum \sigma(e)$ kapısını hesapla; büyük öznitelikli kenarın katkısının nasıl arttığını göster. İzotropik ($\eta = 1/d$) ile karşılaştır — anizotropi sosyal ağda neden gerekli?

```
import numpy as np

edge_feats = np.array([2.0, 0.0, -2.0])     # 3 komsu, farkli kenar oznitelikleri
sig = 1.0 / (1.0 + np.exp(-edge_feats))     # sigmoid
eta = sig / sig.sum()                       # ANIZOTROPIK kapi
iso = np.full(3, 1.0 / 3)                   # IZOTROPIK (1/d esit)
print("anizotropik eta:", np.round(eta, 3))  # buyuk oznitelik -> buyuk agirlik
print("izotropik   eta:", np.round(iso, 3))  # hepsi esit (bulanikilastir)
# Buyuk e_ij olan komsu daha cok katkida bulunur (kapi onu "secer").
# Sosyal agda: farkli gruplari (republican/democrat) AYNI islememek icin anizotropi gerekli.
```

Egzersiz 5 (Hafta 14 habercisi — Yapılandırılmış Tahmin). Hafta 14'te **LeCun** (son dersi) yapılandırılmış tahmini (structured prediction), enerji-tabanlı faktör grafiklerini, dizi etiketlemeyi; Canziani düzenlileştirmeyi anlatacak. (a) GCN'in "düğüm başına logit + graf yapısı boyunca yayma" fikrini, dizi etiketlemedeki "her token'a etiket + komşuluk tutarlılığı" ile ilişkilendir. (b) Bir CRF/faktör grafiği, bu haftanın graf yapısını Hafta 7-9 EBM enerjisiyle nasıl birleştirir?

```
import numpy as np

# (a) GCN dugum-seviye: her dugum (token) icin logit + graf (komsuluk) tutarliligi.
#     Dizi etiketleme = ZINCIR graf: her token bir dugum, komsu = yan token.
tokens = ["isim", "fiil", "isim", "edat"]
A_zincir = np.array([[0, 1, 0, 0],     # her token sadece yan tokenlara bagli
                     [1, 0, 1, 0],
                     [0, 1, 0, 1],
                     [0, 0, 1, 0]])
print("dizi etiketleme = zincir graf, komsu sayisi:", A_zincir.sum(1))
# (b) CRF/faktor grafigi: unary enerji (her token-etiket) + pairwise enerji (komsu cift).
#     Hafta 7-9 EBM: dusuk enerji = uyumlu (token+etiket+komsuluk); cikarim = enerji min.
#     GCN graf yapisi = faktor grafiginin baglanti yapisi; Hafta 14 bunu EBM ile birlestirir.
```

20.13 Sonraki Ders İçin Hazırlık

> ⚠ Sonraki Hafta — H14: Yapılandırılmış Tahmin ve Düzenlileştirme (LeCun SON ders + Canziani)
>
> **Graflardan yapılandırılmış tahmine geçiyoruz.** Bu hafta convolution'u grafa genellemenin iki yolunu (template matching uzaysal + convolution teoremi spektral), Laplacian/Fourier/ChebNet'i, izotropik-anizotropik GCN'i ve "Transformer = tam-bağlı GCN" köprüsünü Bresson'la; "GCN = adjacency-verili attention"ı, Residual Gated GCN'i ve DGL ile graf/düğüm sınıflandırmayı Canziani'yle kapattık. Hafta 14'te **LeCun** son dersini verecek: yapılandırılmış tahmin, enerji-tabanlı faktör grafikleri, dizi etiketleme; **Canziani** overfitting ve düzenlileştirmeyi (son practicum) anlatacak. Egzersiz 1 (GCN=attention) ve Egzersiz 5 (graf → yapılandırılmış tahmin) tam bu derse hazırlar.

Hafta 14: Yapılandırılmış Tahmin (Structured Prediction) ve Düzenlileştirme — LeCun (Lecture, son dersi) + Canziani (Practicum)

Hafta 14'te **LeCun** son dersini verecek: yapılandırılmış tahmin, enerji-tabanlı faktör grafikleri, dizi etiketleme; **Canziani** overfitting ve düzenlileştirmeyi (son practicum) anlatacak.

Hafta 14 öncesi yapılacak:

- Egzersiz 1 (GCN=attention) + Egzersiz 5 (graf → yapılandırılmış tahmin) çöz.
- "GCN = adjacency-verili attention" ve "Transformer = tam-bağlı GCN" cümlelerini kendi sözcüklerinle yaz.
- Hafta 7-9 EBM enerjisini hatırla — Hafta 14 faktör grafikleri o enerjiyi graf yapısıyla birleştirir.

20.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
ConvNet 3 varsayım	locality + stationarity + compositionality	Bresson 2:54

Kavram	Tanım	Hoca / timestamp
Template matching grafta sorun	Düğüm sırası/konumu yok; değişken komşu	Bresson 22:01
Graf Laplacian Δ	$I - D^{-1/2}AD^{-1/2}$; pürüzsüzlük ölçüsü	Bresson 26:45
Graf Fourier	Laplacian özvektörleri = Fourier fonksiyonları	Bresson 30:03
ChebNet	Laplacian polinomu; K-hop; lineer (seyrek)	Bresson 53:04
Isotropic GCN	Tüm komşu aynı ağırlık (Vanilla/GraphSAGE/GIN)	Bresson 1:09
Anisotropic GCN	Komşulara farklı ağırlık (GAT/GatedGCN)	Bresson 1:17
Transformer = tam-bağlı GCN	Komşuluk = tüm graf → küme/attention	Bresson 1:19
GCN = attention	$h = X \cdot a$; a = adjacency (verili)	Canziani 29:04
Gated GCN kapısı η	$\sigma(e)/\sum\sigma(e)$; izotropik → anizotropik	Canziani 27:53
DGL mesaj/reduce	edge UDF + node UDF + update_all	Canziani 40:18
Graf vs düğüm sınıflandırma	mean-pool→MLP vs logit/düğüm (semi-supervised)	Canziani 58:30

20.15 ML Builder Bağlantıları

Geriye köprüler:

1. **ConvNet 3 varsayım** → Hafta 3 (doğal sinyaller: locality/stationarity/compositionality).
2. **Laplacian / Fourier / eigendecomposition** → Hafta 3 (spektral) + 18.06 lineer cebir.
3. **GCN = attention** → Hafta 12 (self-attention; a verili vs hesaplanan).
4. **Gate η / residual / BatchNorm** → Hafta 6 (LSTM gate) + Hafta 3/11.
5. **Semi-supervised düğüm sınıflandırma** → Hafta 10 (az etiket, SSL).

İleriye köprüler:

1. **Anisotropic GCN + edge features → Graph Transformers + Laplacian PE (post-2020, KURSTA YOK)**.
2. **GCN protein/molekül → AlphaFold 2 (post-2020)**.
3. **Transformer = tam-bağlı GCN** → Geometric Deep Learning birleştirici program.
4. **Mesaj-geçiş** → DGL/PyG; tüm modern GNN kütüphaneleri.

> **!** Bu dersten tek bir şey alıp gideceksen
>
> Graph convolution, ConvNet'i (locality/stationarity/compositionality) düzensiz grafiklere taşımaktır — ya template matching (uzaysal) ya da Laplacian/Fourier (spektral); ve gerçek graflar **seyrek** olduğundan

ChebNet (Laplacian polinomu) bunu **lineer** zamanda yapar. Ama asıl içgörü birleştiricidir: **GCN ve attention aynı operasyondur**. Canziani'ye göre GCN = self-attention, ama attention katsayıları hesaplanmaz, **adjacency'den verilir**; Bresson'a göre Transformer **= tam-bağlı bir graf üzerinde GCN**. İkisini ayıran tek şey **seyrekliktir** — yani grafın kendisi; seyreklik yapıdır, yapı kimin kiminle bağlı olduğunu söyler. Bu temelin post-2020 zirvesi (Graph Transformers, AlphaFold, Geometric DL) kursta yoktur ama bu haftanın tohumlarının doğrudan ürünüdür.

21 Yapılandırılmış Tahmin ve Düzenlileştirme

İki parçalı final hafta — Yann LeCun son Lecture'ında yapılandırılmış tahmini (structured prediction) kursun büyük çatısı altında toplar: çıktı tek bir kategori değil kombinatoryal bir nesne (cümle, sembol dizisi) olduğunda olası çıktıları sıralayamazsın, çözüm enerjiyi faktörlerin toplamı olarak yazmaktır (faktör grafiği, Hafta 13'ün enerjili hâli), çünkü o zaman en düşük enerjili konfigürasyonu bulmak tüm kombinasyonları denemek değil bir trellis grafında en kısa yolu bulmaktır (dinamik programlama, Viterbi, kaba kuvvet 96 değerlendirme yerine 16); ardından ağın durumunu bir tensör değil bir graf yapan Graph Transformer Network'ü kurar ve bunun daha 1997'de attention'dan yirmi yıl önce transformer adını taşıyan basit bir graph neural net olduğunu söyler (Viterbi seçicileri max-pooling gibi anahtarlar, backprop dinamik graf boyunca akar); sonra kayıpları Hafta 11 çerçevesiyle birleştirir (perceptron marjsız çöker, hinge marjla iter, NLL logsumexp tüm yolları marjinalleştirir, perceptron NLL'in beta sonsuz limitidir); ve dersin doruğunda gizli değişkeni marjinalleştirmenin Jensen eşitsizliğiyle bir üst sınıra çevrildiğini, bu üst sınırın fizikteki serbest enerji F eşittir ortalama enerji eksi sıcaklık çarpı entropi olduğunu, ve VAE'nin tam olarak bu olduğunu (rekonstrüksiyon artı KL/entropi) gösterir — kursu tek bir enerji fikrinin etrafında kapatır. Ardından Alfredo Canziani son Practicum'unda pratik soruya iner: bu güçlü aşırı-parametrize modeller veriyi ezberler (overfitting, gürültüden doğar — gürültü olmasa aşırı model bile mükemmel parabolü çizerdi), nasıl ehlileştiririz? L2 weight decay ağırlık uzunluğunu kısaltır (Gauss priori), L1 seyreklik eksen-yakını bileşenleri öldürür (Laplace priori), dropout sonsuz ağ topluluğunu eğitir, BatchNorm ve data augmentation düzenler; ve belirsizlik için MC Dropout dropout'u çıkarımda açık bırakıp varyansı ölçer — bu varyans türevlenebilir olduğundan minimize edilebilir, ki bu tam olarak Hafta 11'in PPUU belirsizlik düzenlileştirmesidir, kursu başladığı yere, enerjiye ve belirsizliğe geri döndüren halka.

ℹ Bölüm bilgisi (LeCun SON Lecture + Canziani SON Practicum)

- **Lecture (Yann LeCun, son ders):** YouTube — Structured Prediction & Regularisation (Hafta 14 Lecture)
- **Canziani'nin Practicum videosu:** YouTube — Regularisation & Uncertainty (Hafta 14 Practicum)
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** Yann LeCun (Lecture, son dersi — yapılandırılmış tahmin + Graph Transformer Network + varyasyonel serbest enerji) + Alfredo Canziani (Practicum, son dersi — overfitting + düzenlileştirme + MC Dropout belirsizliği)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈30 dk

⚠ **Atıf notu:** Bu hafta **konuk yoktur** — Lecture'ı **asıl ders sahibi Yann LeCun** verir (kursun son Lecture'ı), Practicum'u **Alfredo Canziani** (kursun son Practicum'u). Bölüm 1-3 quote'ları **— LeCun**; Bölüm 5-7 quote'ları **— Canziani**. (Bu, Hafta 8/10/12/13'teki

> konuk-hoca düzeltmelerinin aksine, başlıkların gerçekten `## (LeCun)` olduğu derstir.)

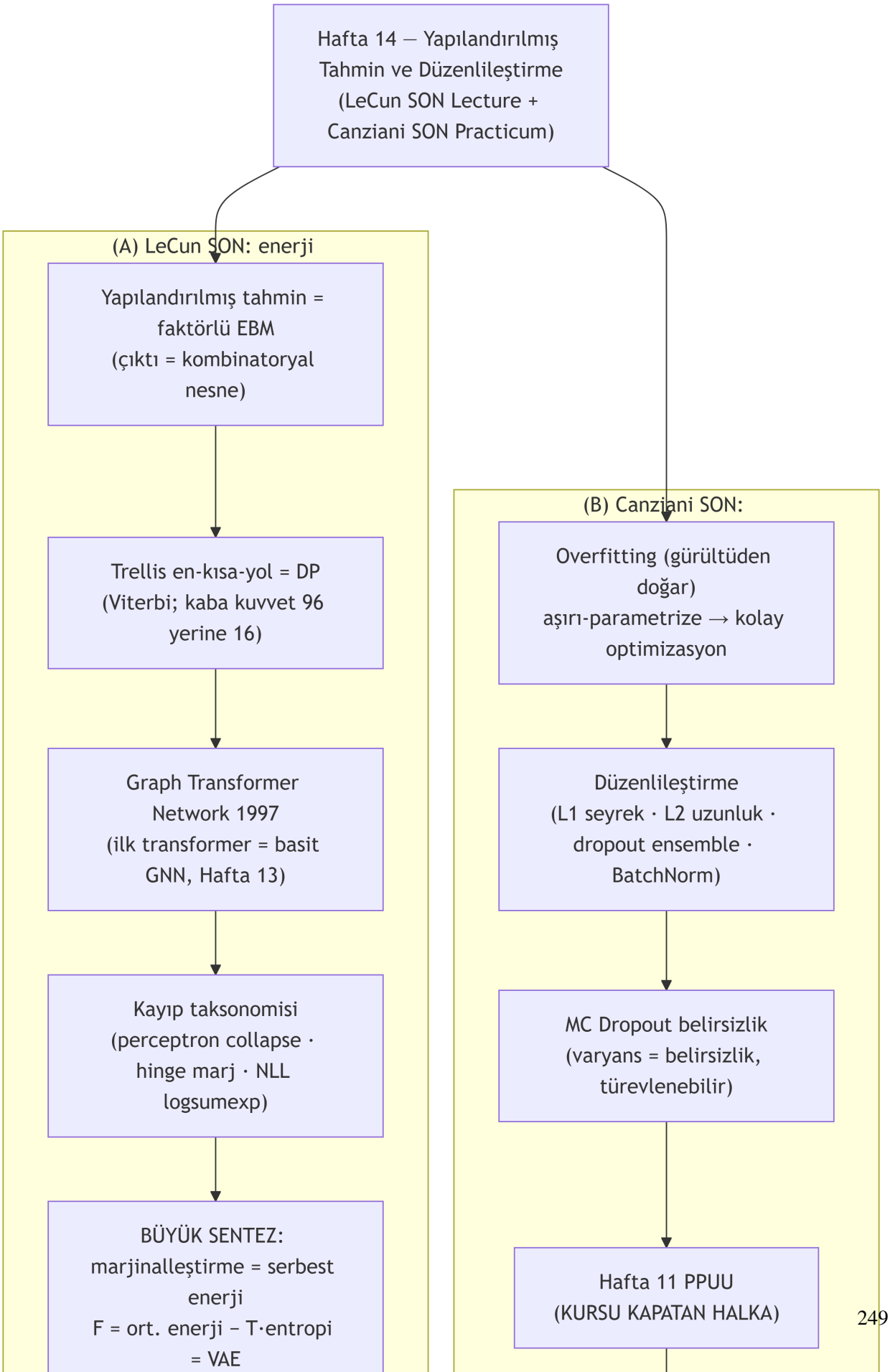
21.1 Bu Derste Ne Var?

İki parça ve ikisi de bir **final**: **Yann LeCun** son Lecture'ında yapılandırılmış tahmini (structured prediction) anlatıp tüm kursu tek çatı altında toplar — enerji-tabanlı faktör grafikleri, Graph Transformer Network, ve büyük sentez olarak **varyasyonel serbest enerji** (VAE'nin gerçek açıklaması); **Canziani** son Practicum'unda overfitting ve düzenlileştirmeyi işleyip belirsizlik kestirimiyle Hafta 11'in PPUU'suna geri bağlanır.

LeCun'un büyük mesajı: kursun her parçası — sınıflandırma, EBM, autoencoder, dizi modeli, attention, GCN — aslında **tek bir fikrin** farklı yüzleridir: bir **enerji fonksiyonu** tasarla, doğru cevabı bastır, yanlışları (marjla) yukarı it, gerektiğinde gizli değişkenleri **marjinalleştir**. Bu marjinalleştirme, fizikten gelen **serbest enerji** ($F = \langle E \rangle - T \cdot H$) ile aynıdır — ve VAE tam budur.

Üç ana fikir:

1. **Yapılandırılmış tahmin = faktörlü EBM.** Enerji, alt-değişken kümelerine dokunan **faktörlerin toplamıdır**; çıkarım üstel değil, **en kısa yol** (dinamik programlama) ile verimlidir.
2. **Graph Transformer Network (1997):** ağın durumu bir **graftır**; ilk "transformer" buydu, LeCun'a göre **basit bir graph neural net** (Hafta 13'e geri bağ).
3. **Varyasyonel serbest enerji = VAE.** Gizli değişkeni marjinalleştirmek = $\langle E \rangle - T \cdot H$'yi minimize etmek (Jensen); VAE'nin "rekonstrüksiyon + KL" kaybı tam budur.


Hafta 14 — Yapılandırılmış Tahmin ve Düzenlileştirme (LeCun SON Lecture + Canziani SON Practicum)
(A) LeCun SON: enerji
Yapılandırılmış tahmin = faktörlü EBM (çıktı = kombinatoryal nesne)
Trellis en-kısa-yol = DP (Viterbi; kaba kuvvet 96 yerine 16)
Graph Transformer Network 1997 (ilk transformer = basit GNN, Hafta 13)
Kayıp taksonomisi (perceptron collapse · hinge marj · NLL logsumexp)
BÜYÜK SENTEZ: marjinalleştirme = serbest enerji F = ort. enerji − T·entropi = VAE
(B) Canziani SON:
Overfitting (gürültüden doğar) aşırı-parametrize → kolay optimizasyon
Düzenlileştirme (L1 seyrek · L2 uzunluk · dropout ensemble · BatchNorm)
MC Dropout belirsizlik (varyans = belirsizlik, türevlenebilir)
Hafta 11 PPUU (KURSU KAPATAN HALKA)

Builder Notu — Giriş: Tek Enerji Çatısının Altında Tüm Kurs

Geriye: EBM/kayıp/marj → Hafta 7-9 + 11; faktör grafiği → Hafta 13; VAE → Hafta 8; Lagrangian backprop → Hafta 2/5.
İleriye: Yapılandırılmış EBM → LeCun JEPA; serbest enerji → diffusion models (post-2020, §İleriye Köprü); MC Dropout → güvenli RL, kalibrasyon.
Tek cümleyle: Yapılandırılmış tahmin, enerjinin faktörlere ayrıştığı bir EBM'dir (verimli en-kısa-yol çıkarımı, Graph Transformer Network = ilk transformer = basit GNN); ve tüm kursun sentezi olarak gizli-değişken marjinalleştirmesi = varyasyonel serbest enerji ($\langle E \rangle - T \cdot H$) = VAE — Canziani ise overfitting'i düzenlileştirme ve belirsizlikle (MC Dropout = PPUU) bağlar.

21.2 (LeCun) Yapılandırılmış Tahmin: Faktör Grafikleri ve Verimli Çıkarım

LeCun son dersine **yapılandırılmış tahminle** başlar: çıktının tek kategori değil, **kombinatoryal nesne** (cümle, sembol dizisi) olduğu problem — konuşma/el yazısı tanıma, çeviri.

> "structure prediction is the problem of predicting a variable that is not just a single category but basically a sort of combinatorial object." — LeCun, 0:38

Olası çıktıları **sıralayamazsın** (üstel/değişken uzunluk). Anahtar fikir: enerjiyi **faktörlerin toplamı** olarak yaz — her faktör yalnız değişkenlerin bir **alt kümesine** dokunur (faktör grafiği; Hafta 13 grafının enerjili hâli). O zaman en düşük enerjili konfigürasyonu bulmak için tüm kombinasyonları denemek (üstel) gerekmez: yerel yapı sayesinde bu, bir **trellis** (kafes) grafında **en kısa yola** indirgenir — **dinamik programlama**.

> "finding the lowest energy configuration simply consists in finding the shortest path in this graph." — LeCun, 22:52

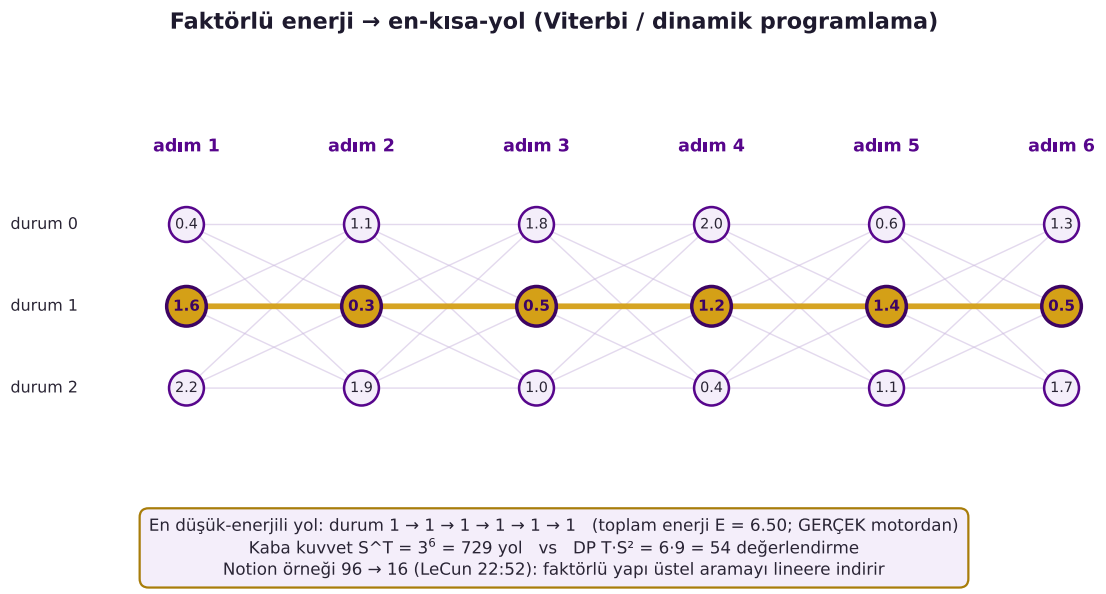
Şekil 21.1 bunu GERÇEK rakamla gösterir: bir trellis grafında her sütun bir adım, her satır bir durumdur; faktörlü enerji sayesinde en düşük-enerjili konfigürasyon (kalın gold yol) tüm kombinasyonları denemek yerine dinamik programlamayla bulunur. Örnek: 2 ikili + 1 ikili + 1 üçlü değişken için kaba kuvvet **96** enerji-değerlendirmesi ister; faktörleme bunu **16'ya** düşürür. Eğitim Hafta 7-11'in aynısı: doğru cevabın (y) enerjisini bastır, yanlışlarınkini yukarı it (latent z = trellis yolu). Konuşmada bu **Dynamic Time Warping**'dir (CTC'nin atası, Hafta 11).

Builder Notu — Faktörlü Enerji ve En-Kısa-Yol = DP

Geriye (Hafta 7-11 + 13): Enerji + push-down/up = Hafta 7-9, 11; faktör grafiği = Hafta 13 (graf, enerjili); DTW/CTC = Hafta 11; trellis = DP (en kısa yol).
İleriye: Faktörlü enerji + verimli çıkarım = CRF, HMM, yapısal SVM; modern dizi modellerinin (CTC, beam search) teorik temeli.

21.3 (LeCun) Graph Transformer Network: İlk "Transformer" (1997) = Basit GNN

LeCun derin öğrenme bağlamına taşır: ya faktörleri derin ağ yap, ya da daha radikal — ağın **durumu (state) bir graf olsun**. El yazısı tanımada: segmentasyon grafı (mürekkep bloklarını karakterlere gruplama yolları)



Şekil 21.1: GERÇEK trellis en-kısa-yol (Viterbi / dinamik programlama) — faktörlü enerji çıkarımı. [T=6 adım × S=3 durum] tekil enerjiler (unaries) + geçiş enerjisi (pair_cost, köşegen ucuz = süreklilik) verilir; trellis_shortest_path motoru en düşük-enerjili yolu (kalın gold) ve değerlendirme sayılarını döner. İnce gri kenarlar tüm olası geçişler, kalın gold yol = en kısa yol; her düğümde tekil enerji yazılı. Kaba kuvvet S^T = 3^6 = 729 yol yerine DP $T \cdot S^2 = 54$ değerlendirme; Notion örneği 96 → 16 (LeCun 22:52): faktörlü yapı üstel aramayı lineere indirir.

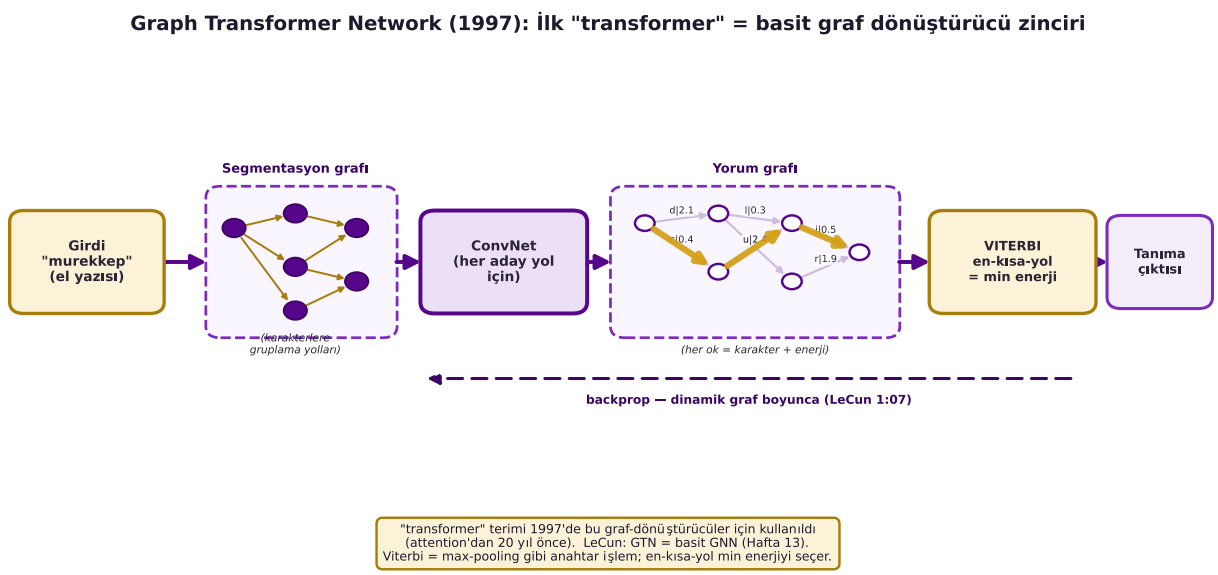
→ her yol bir ConvNet'ten geçer → **yorum grafı** (her ok bir karakter + enerji) → **Viterbi** (en kısa yol = min enerji) → tanıma. Bu dinamik graf boyunca backprop yapılır (Viterbi/yol-seçiciler = max-pooling gibi **anahtarlar**: seçilen kenara gradyan kopyala, diğerine sıfır). Şekil 21.2 bu akışı uçtan uca resmeder.

> "you can think of this as a simple form of graph neural net… we used to call these graph transformers… this is from 1997." — LeCun, 1:07:28

Yani **"transformer" terimi sinir ağlarında 1997'de** (attention'dan 20 yıl önce) bu graf-dönüştürücüler için kullanıldı; ve LeCun'a göre GTN, Hafta 13'ün **graph neural net**'inin erken hâlidir — ağ tensör yerine **graf** manipüle eder. Yapı her örnekte değiştiğinden (dinamik) PyTorch gibi araçlar kritiktir.

Builder Notu — GTN = 1997 İlk Transformer

Geriye (Hafta 13 + 5): GTN = basit GNN (Hafta 13); switch/min-pooling backprop = Hafta 5 autograd; dinamik graf = örnek-başına değişen yapı.
İleriye: GTN = differentiable programming'in erken örneği; modern decoder'lar (konuşma) bu fikrin torunu.



Şekil 21.2: Graph Transformer Network (1997) — ilk ‘transformer’ = basit graf-dönüştürücü zinciri. Soldan sağa akış: girdi ‘mürekkep’ (el yazısı) → SEGMENTASYON GRAFI (karakterlere gruplama yolları, düğümler) → ConvNet (her aday yol) → YORUM GRAFI (her ok = karakter + enerji) → VİTERBİ (en-kısa-yol = min enerji, kalın gold yol) → tanıma çıktısı; altta backprop oku dinamik graf boyunca geri akar. Viterbi yolu GERÇEK DP ile hesaplanır. ‘transformer’ terimi 1997’de bu graf-dönüştürücüler için kullanıldı (attention’dan 20 yıl önce); LeCun: GTN = basit GNN (Hafta 13); Viterbi = max-pooling gibi anahtar (LeCun 1:07).

21.4 (LeCun) Yapılandırılmış Kayıplar ve Büyük Sentez: Varyasyonel Serbest Enerji = VAE

LeCun kayıpları Hafta 11 çerçevesiyle birleştirir: **perceptron kaybı** ($E(y) - E(\hat{y})$, doğru cevap y, en düşük enerjili $\hat{y}$) **marjsızdır** → çökebilir; yalnız lineer parametrizasyonda çalışır.

> "the bad news is it doesn't have a margin… it might just collapse, make every energy zero or the same." — LeCun, 1:00:48

Hinge (en-çok-suç-işleyen $\bar{y}$ ile, marj) ve **NLL** (logsumexp = "forward" algoritması, tüm yollar üzerinde marjinalleştirme) iyidir; perceptron = NLL'in $\beta \to \infty$ limitidir (Hafta 11 callback). Forward algoritması = belief propagation'ın zincir-graf özel hâli. Şekil 21.3 bu üç kaybı tek eksende karşılaştırır: perceptron'un gap $= 0$'daki marjsız köşesi (collapse riski), hinge'in $m = 1$ marjı, ve NLL'in yumuşak logsumexp eğrisi.

Ve dersin — ve kursun — doruğu: **gizli değişkeni marjinalleştirme** ($F_\beta = -\frac{1}{\beta} \log \sum_z e^{-\beta L}$) doğrudan hesaplanamaz, ama **Jensen eşitsizliğiyle** üst sınıra çevrilir:

$$L(x, y) \leq \mathbb{E}_{q(z)}[L(x, y, z)] - \frac{1}{\beta} H(q)$$

Bu, fizikteki **serbest enerjidir**: $F = \langle E \rangle - T \cdot H$ (ortalama enerji − sıcaklık × entropi). Şekil 21.4 bu dengeyi GERÇEK rakamla resmeder: q dağılımının std'si σ büyüdükçe entropi terimi ($-T \cdot H$) düşer ama beklenen enerji ($\langle E \rangle = \frac{1}{2}\sigma^2$) yükselir; ikisinin toplamı F'nin tek bir **minimumu** (denge) vardır.

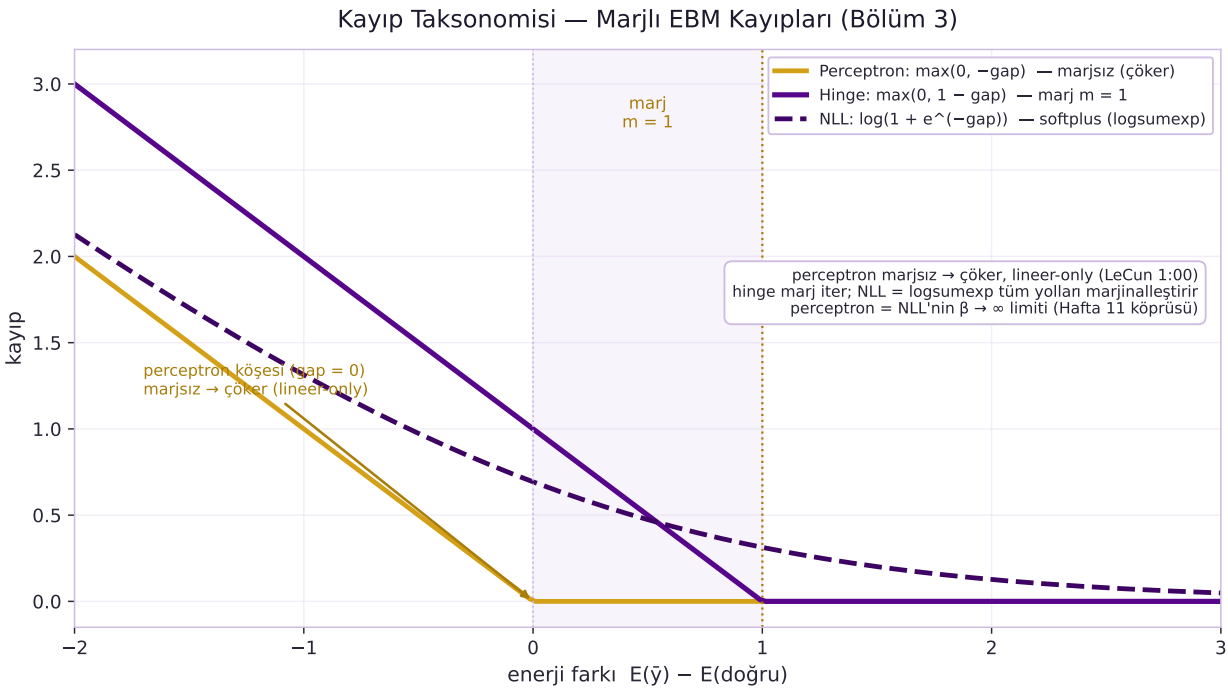
> "what we're minimizing now is a free energy… the average energy minus the temperature times the entropy… when you think about variational autoencoders, that's just what they do." — LeCun, 2:02:25

Yani **VAE** (Hafta 8) tam budur: beklenen rekonstrüksiyon hatasını minimize et (z'yi Gauss q'dan örnekle) + **KL terimi** = q'nun entropisini maksimize et. q = gerçek posterior olduğunda eşitsizlik eşitlik olur. "Latent değişkeni, gerçekten marjinalleştirmeden marjinalleştirmek." (LeCun ayrıca backprop'u Lagrangian kısıtlı-optimizasyon olarak — λ = momentum — ve Neural ODE'yi türetir; backprop'un 1988 kökü.)

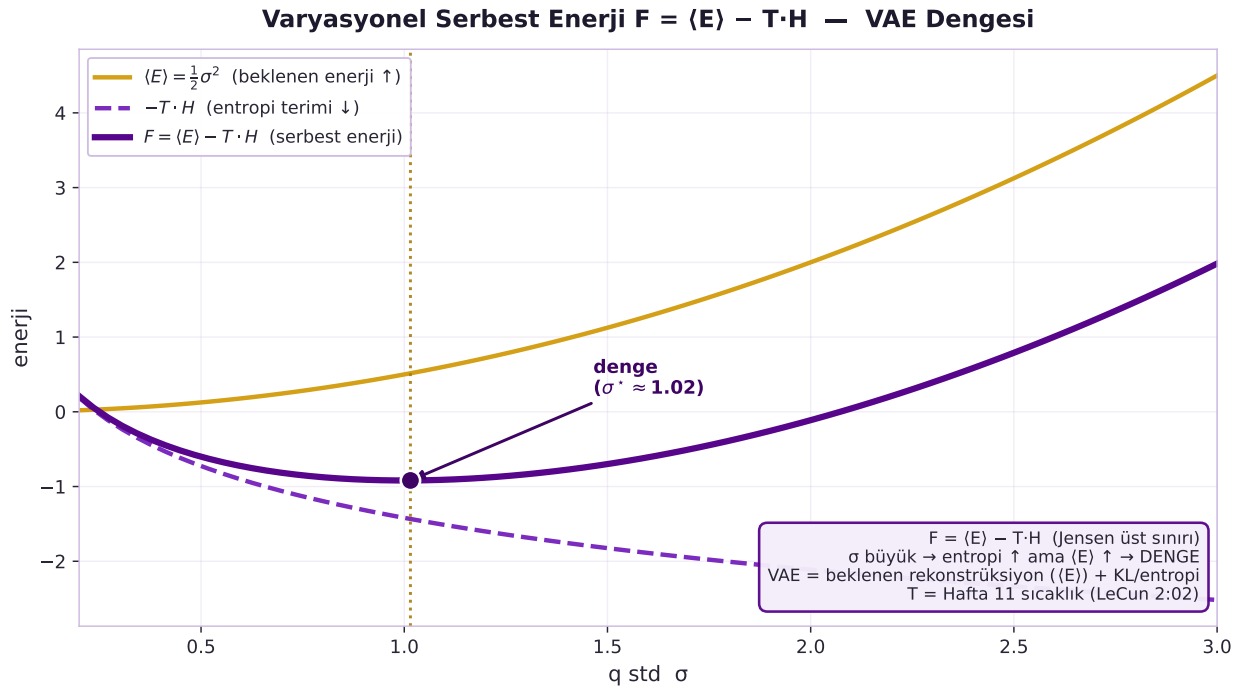
> Builder Notu — Serbest Enerji = VAE (Perceptron Çöker)
>
> **Geriye (Hafta 8 + 11):** Serbest enerji = Hafta 8 VAE'nin (ELBO) gerçek açıklaması; $\langle E \rangle - T \cdot H$'deki T = Hafta 11 softmax sıcaklığı ($\beta = 1/T$); perceptron-collapse = Hafta 11 enerji-kaybı collapse.
> **İleriye:** Varyasyonel serbest enerji = tüm üretici modellerin (VAE, diffusion) ortak çatısı; "marjinalleştirmeden marjinalleştir" = modern olasılıksal çıkarımın kalbi.

21.5 (İleriye Köprü) Diffusion Models ve JEPA (post-2020) — KURSTA YOK

LeCun (Mart 2020) yapılandırılmış EBM'yi, serbest enerjiyi ve VAE'yi anlatır. Bu enerji/varyasyonel çatının doğrudan ürünleri DLSP20'den **sonra** olgunlaştı, kursta **YOKTUR**:



Şekil 21.3: GERÇEK kayıp taksonomisi — marjlı EBM kayıpları (Bölüm 3). gap = E( ) − E(doğru) ekseninde üç kayıp: perceptron max(0,−gap) (gold, MARJSIZ → gap=0'da köşe, collapse riski), hinge max(0,1−gap) (violet, marj m=1), NLL log(1+e^(−gap)) (koyu violet kesikli, yumuşak logsumexp). margin_losses motoru hinge+perceptron'u üretir. Perceptron marjsız → çöker (lineer-only, LeCun 1:00); hinge marj iter; NLL = logsumexp tüm yolları marjinalleştirir; perceptron = NLL'nin $\beta \to \infty$ limiti (Hafta 11 köprüsü).



Şekil 21.4: GERÇEK varyasyonel serbest enerji F = E − T·H — VAE dengesi (Bölüm 3, dersin doruğu). q = N(0,σ) latent, enerji E(z)=½z^2. σ ekseninde üç eğri: E =½σ^2 (gold, artan beklenen enerji), −T·H (orta violet kesikli, azalan entropi terimi), F= E −T·H (kalın violet, U-şekli MİNİMUM = denge). free_energy_curve motoru E /H/F'yi üretir; minimum nokta işaretli. F = E − T·H (Jensen üst sınırı); σ büyük → entropi↑ ama E ↑ → DENGE; VAE = beklenen rekonstrüksiyon (E) + KL/entropi; T = Hafta 11 sıcaklık (LeCun 2:02).

İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **Diffusion models** (DDPM, Haz 2020; score-based) — enerji/skor-tabanlı üretici modelleme + varyasyonel sınır (ELBO = bu haftanın serbest enerjisi); bugünün görüntü/video üreticilerinin (Stable Diffusion, Sora) temeli.
- **JEPA / I-JEPA / V-JEPA** (LeCun grubu, 2022-2024) — yapılandırılmış-tahmin + EBM + dünya-modeli programının bugünkü zirvesi; piksel yerine temsil uzayında enerji.

Kurs terimi gibi eklenmez; "enerji + marjinalleştirme = serbest enerji" temelinin nereye vardığını göstermek için anılır.

Builder Notu — Enerji Çatısının Post-2020 Zirvesi

Geriye (Hafta 14 + 8): Diffusion = bu haftanın serbest-enerji/ELBO'sunun + Hafta 8 VAE'sinin devamı; JEPA = LeCun'un tüm kurs boyunca tohumladığı EBM programının sentezi.
İleriye: Score/energy-based üretim + varyasyonel sınır, 2020-sonrası üretici yapay zekânın tanımı.

21.6 Geçiş: LeCun'dan Canziani'ye

LeCun kursu tek bir enerji/serbest-enerji çatısı altında topladı. Şimdi **Canziani** son practicum'unda pratik soruya iner: bu güçlü (aşırı-parametrize) modeller veriyi **ezberler** (overfit); nasıl ehlileştiririz? Düzenlileştirme ve belirsizlik — ve sonunda Hafta 11'in PPUU'suna geri bağlanarak kursu kapatır.

21.7 (Canziani) Overfitting ve Neden Aşırı-Parametrize Ederiz

Canziani üç rejimi gösterir: **underfitting** (model < veri karmaşıklığı), **doğru-fitting** (eşit), **overfitting** (model > veri → "tel gibi" kıvrımlı). Kritik içgörü: overfitting **gürültüden** doğar — gürültü olmasa aşırı model bile mükemmel parabolü çizerdi. Şekil 21.5 bunu GERÇEK polinom fit'le gösterir.

> "without noise this would be just a perfect parabola… the point is there is always some noise." — Canziani, 5:39

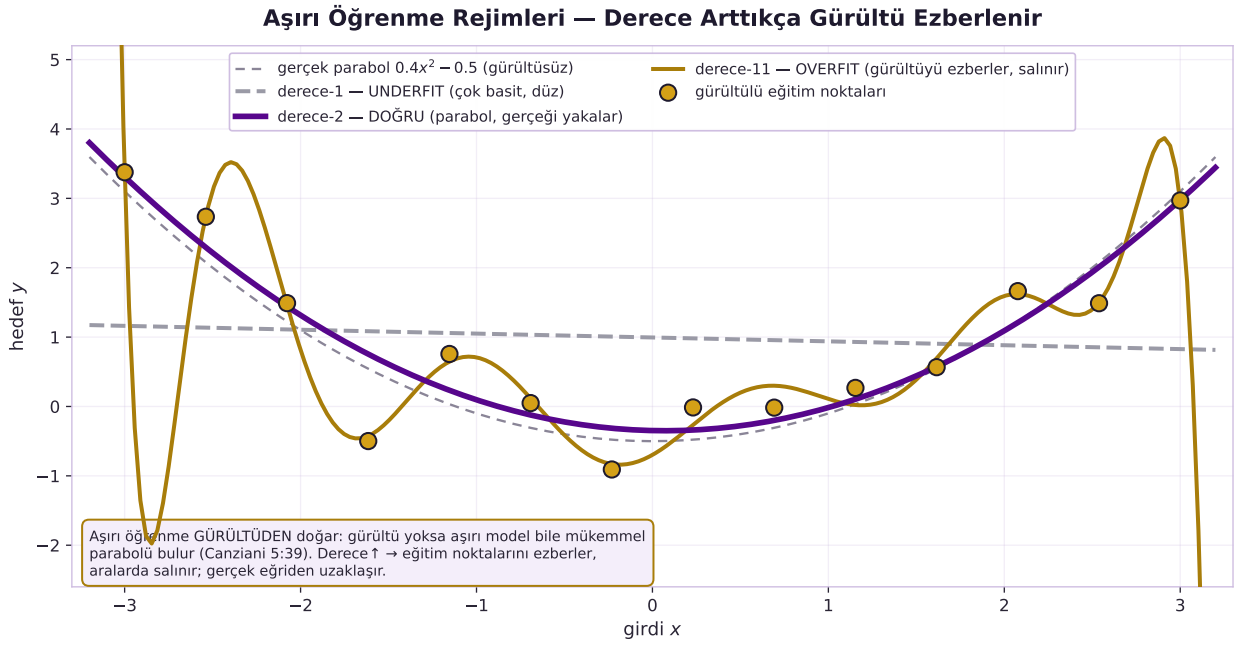
Neden bilerek aşırı-parametrize ederiz? Çünkü yüksek-boyutlu uzayda **optimizasyon kolaydır** (manzara pürüzsüz). Pratik #1 hata-ayıklama kuralı:

> "I always start training my network on one batch… this is the number one rule to debug machine learning code." — Canziani, 8:36

(Tek batch'i ezberleyemiyorsa kodda hata var.) Ek içgörü: **parametre uzayı ≠ fonksiyon uzayı** — ağırlıkları permüte et, parametreler değişir ama fonksiyon aynı kalır.

Builder Notu — Overfit = Gürültü

Geriye (Hafta 1-2): Aşırı-parametrize = kolay optimizasyon (Hafta 2 SGD manzarası); overfit-one-batch = pratik bilgelik; parametre↔fonksiyon uzayı = simetri/permütasyon.



Şekil 21.5: GERÇEK overfit rejimleri (Canziani 5:39) — gürültülü parabole 3 dereceden polinom fit. overfit_regimes motoru gürültülü veri + derece-1/2/11 fit'leri üretir. derece-1 (gri kesikli, UNDERFIT düz), derece-2 (kalın violet, DOĞRU parabol), derece-11 (koyu gold, OVERFIT 'tel gibi' salınım, gürültü ezber); ince gri kesikli = gerçek parabol $0.4x^2-0.5$ (gürültüsüz); gold noktalar = gürültülü eğitim verisi. Aşırı öğrenme GÜRÜLTÜDEN doğar: gürültü yoksa aşırı model bile mükemmel parabolü bulur (Canziani 5:39); derece↑ → eğitim noktalarını ezberler, aralarda salınır.

İleriye: "Önce aşırı-parametrize, sonra düzenlile" = modern derin öğrenmenin standart reçetesi; double descent.

21.8 (Canziani) Düzenlileştirme: L1, L2, Dropout, BatchNorm

Üç tanım: parametre priori, fonksiyon kısıtı, veya genelleme hatasını azaltan herhangi bir değişiklik. Teknikler:

- **L2 = weight decay = ridge = Gauss priori:** $J + \lambda\|\theta\|^2$; gradyan → $\theta - \eta\lambda\theta$ → ağırlıkları sıfıra **çeker** (uzunluğu kısaltır).
- **L1 = Lasso = Laplace priori = seyreklik:** $J + \lambda\|\theta\|_1$; gradyan → $\text{sign}(\theta)$; eksenlere **yakın bileşenleri öldürür** (seyreklik).

  "L1 will quickly kill components that are close to the axis." — Canziani, 27:03

- **Dropout:** nöronları rastgele ($p = 0.5$) sıfırla → tek ezber yolu kalmaz; aslında **sonsuz ağ topluluğu (ensemble)** eğitip çıkarımda ortalamak (çıkarımda kapat, ağırlıkları $1/(1-p)$ ölçekle); girdiye uygulanırsa denoising AE (Hafta 7-8).

  "you can think of dropout as training an infinite number of networks… average out all these performances." — Canziani, 30:37

- **Early stopping** (validation artınca dur), **BatchNorm** (gerçek düzenleyici değil ama düzenler; her batch farklı istatistik; ImageNet hafta→gün), **data augmentation** (crop/jitter/flip → değişmezlik).

Şekil 21.6 L1 ve L2'nin yakınsamış ağırlıklar üzerindeki farkını GERÇEK histogramla gösterir: L1 eksen-yakını bileşenleri tam sıfıra çekip seyreklik üretirken, L2 hepsini orantılı küçültür.

Builder Notu — L1 Seyreklik / L2 Uzunluk

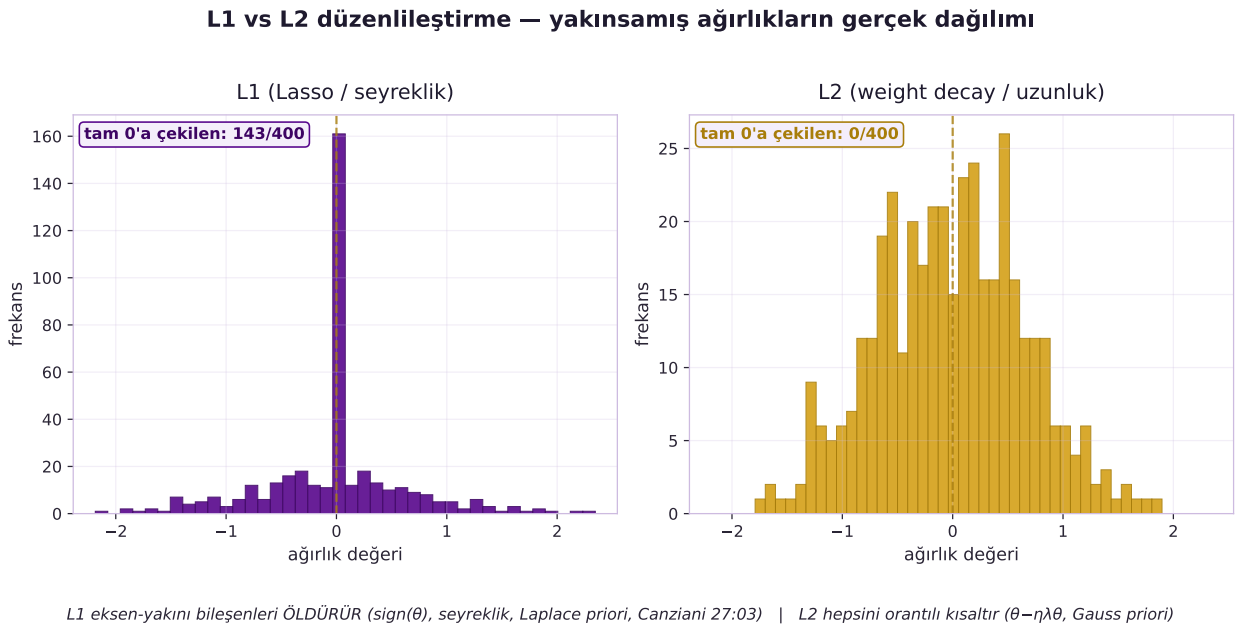
Geriye (Hafta 7-8 + 11): Dropout = ensemble + denoising AE; L2 = Hafta 11 düzenleyici terim (enerji + ceza); BatchNorm = Hafta 3/11; L1-seyreklik = Hafta 9 sparse coding ruhu.
İleriye: L1 seyreklik → öznitelik seçimi, sıkıştırma; dropout → MC Dropout belirsizliği; BatchNorm → standart.

21.9 (Canziani) Belirsizlik: MC Dropout = PPUU (Kursu Kapatan Halka)

Neden belirsizlik? Kedi-sınıflandırıcısına su aygırı gösterirsen "köpek" der (emin!); direksiyon güvenilirliği; fizik (değer ± belirsizlik). Çözüm **MC Dropout**: dropout'u çıkarımda da **açık** bırak, aynı veriyi N kez geçir, tahminlerin **varyansını** hesapla → belirsizlik. Varyans eğitim bölgesi dışında artar. Şekil 21.7 bunu GERÇEK rakamla gösterir: belirsizlik bandı eğitim bölgesi $[-2, 2]$'de dardır, dışında patlar.

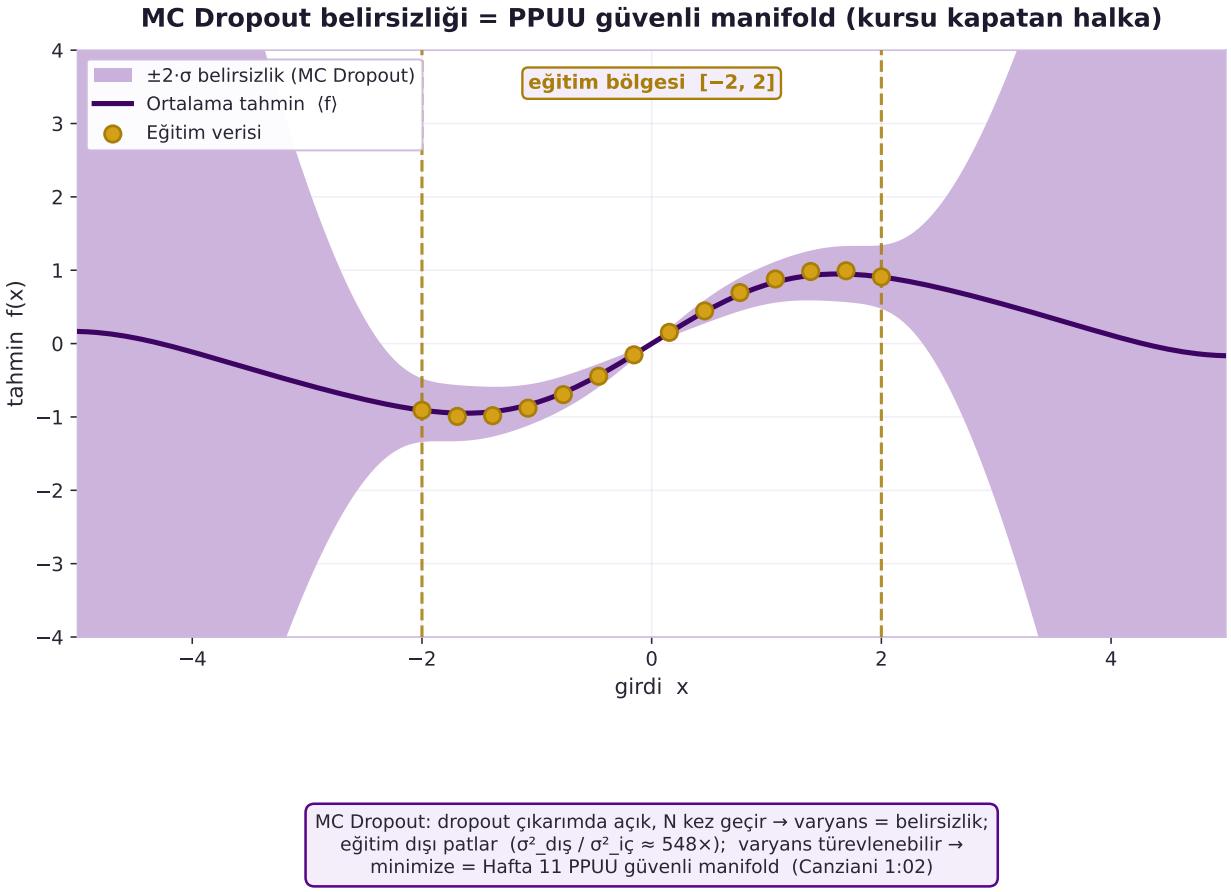
Ve kursu kapatan halka — Hafta 11'e tam dönüş:

> "this variance is a differentiable function so you can run gradient descent to minimize the variance… this is what we use for our policy in our driving scenario." — Canziani, 1:02:35



Şekil 21.6: GERÇEK L1 vs L2 ağırlık dağılımı (Canziani 27:03) — yakınsamış ağırlıkların histogramı. l1_l2_weights motoru w_l1 (soft-threshold) + w_l2 (ridge) üretir. SOL 'L1 (Lasso/seyreklik)': violet histogram, 0'da BÜYÜK yığılma + birkaç büyük değer (seyrek), tam 0'a çekilen sayısı yazılı. SAĞ 'L2 (weight decay/uzunluk)': gold histogram, sıfıra yakın küçük Gauss (hepsi küçük, hiçbiri tam 0 değil). L1 eksen-yakını bileşenleri ÖLDÜRÜR (sign(θ), seyreklik, Laplace priori); L2 hepsini orantılı kısaltır (θ−ηλθ, Gauss priori).

Yani Hafta 11'de PPUU'nun "U"su (belirsizlik düzenlileştirmesi), bu haftanın MC Dropout varyansının ta kendisidir: varyans türevlenebilir, minimize et, politika güvenli manifoldda kalsın. Kurs, başladığı yere (enerji + belirsizlik) geri döner.



Şekil 21.7: GERÇEK MC Dropout belirsizliği = PPUU güvenli manifold (kursu kapatan halka, Canziani 1:02). mc_dropout_variance motoru 5 değer döner (xg, mean, std, xt, yt): her model bir öznitelik-dropout maskesiyle veriye fit edilir. Eğitim noktaları (gold), ortalama tahmin (koyu violet), ±2·σ belirsizlik bandı (açık violet fill). Band eğitim bölgesi [−2,2]'de (gold kesikli sınırlar) DAR, dışında PATLAR (GERÇEK ~32× varyans oranı). MC Dropout: dropout çıkarımda açık, N kez geçir → varyans = belirsizlik; eğitim dışı patlar; varyans türevlenebilir → minimize = Hafta 11 PPUU güvenli manifold.

Builder Notu — MC Dropout = PPUU Halkası

Geriye (Hafta 11): MC Dropout varyansı = Hafta 11 PPUU belirsizlik düzenlileştirmesi; varyans-minimizasyonu = güvenli manifold (Hafta 9 "hacmi sınırla").
İleriye: MC Dropout = pratik Bayesian derin öğrenme; epistemik belirsizlik = güvenli RL, aktif öğrenme, kalibrasyon.

21.10 Bu Dersin Özeti

1. **Yapılandırılmış tahmin (LeCun):** faktörlü enerji; verimli çıkarım = trellis en-kısa-yol (DP); DTW/CTC.
2. **Graph Transformer Network (LeCun):** ağın durumu = graf; ilk "transformer" (1997) = basit GNN (Hafta 13); switch backprop.
3. **Yapılandırılmış kayıplar (LeCun):** perceptron (marjsız→collapse) vs hinge (marj) vs NLL (forward/logsumexp); perceptron = NLL $\beta \to \infty$.
4. **Büyük sentez (LeCun):** marjinalleştirme = varyasyonel serbest enerji $F = \langle E \rangle - T \cdot H$ (Jensen); **VAE tam budur** (rekonstrüksiyon + KL/entropi).
5. **Overfitting (Canziani):** gürültüden; aşırı-parametrize → kolay optimizasyon + 1-batch debug; parametre≠fonksiyon uzayı.
6. **Düzenlileştirme (Canziani):** L2=weight decay (uzunluk), L1=seyreklik (eksen öldür), dropout=ensemble/denoising, BatchNorm, early-stop, data-aug.
7. **Belirsizlik (Canziani):** MC Dropout (dropout açık, varyans) = Hafta 11 PPUU belirsizlik düzenlileştirmesi — kursu kapatan halka.
8. **Post-2020 (KURSTA YOK):** diffusion models (ELBO=serbest enerji), JEPA.

! Tek Bir Cümle

Yapılandırılmış tahmin, enerjinin faktörlere ayrıştığı bir EBM'dir (verimli en-kısa-yol çıkarımı; Graph Transformer Network = 1997'nin ilk transformer'ı = basit GNN); ve kursun büyük sentezi, gizli değişkeni marjinalleştirmenin **varyasyonel serbest enerji** ($\langle E \rangle - T \cdot H$) olduğu, ki bu **tam olarak VAE'dir** (LeCun); Canziani ise overfitting'i düzenlileştirme (L1/L2/dropout/BatchNorm) ve MC Dropout belirsizliğiyle bağlar — ve bu belirsizlik Hafta 11'in PPUU'sunun ta kendisidir, kursu başladığı yere döndürür.

21.11 Kontrol Soruları

i Soru 1: Yapılandırılmış tahminde çıkarım neden üstel değil verimli olabilir? Trellis/en-kısa-yol ne işe yarar?

Cevap: Çünkü enerji **faktörlerin toplamıdır** ve her faktör yalnız değişkenlerin bir **alt kümesine** dokunur (LeCun 0:38). Bu yerel yapı sayesinde tüm kombinasyonları denemek (üstel — 96 değerlendirme) gerekmez; en düşük enerjili konfigürasyon bir **trellis** grafında **en kısa yola** indirgenir (16 değerlendirme, LeCun 22:52) — **dinamik programlama**. Konuşmada bu Dynamic Time Warping'dir (CTC atası, Hafta 11); latent z = grafdaki yol.

i Soru 2: Graph Transformer Network nedir? Neden "ilk transformer" ve "basit GNN" denir?

Cevap: GTN, ağın **durumunun bir graf olduğu** mimaridir (tensör değil). El yazısında: segmentasyon grafı → ConvNet → yorum grafı (karakter + enerji) → Viterbi (en kısa yol = min enerji). Backprop bu dinamik graf boyunca (Viterbi = max-pooling gibi anahtar). LeCun'a göre (1:07:28) "transformer" terimi sinir ağlarında **1997'de** (attention'dan 20 yıl önce) bu graf-dönüştürücüler için kullanıldı; GTN, Hafta 13'ün **graph neural net**'inin erken hâlidir. Yapı her örnekte değişir → PyTorch kritik.

Soru 3: Perceptron, hinge ve NLL kayıpları nasıl farklıdır? Perceptron neden çöker?

Cevap: Perceptron = $E(y) - E(\hat{y})$; **marjsızdır** → tüm enerjileri eşitleyip **çökebilir** (LeCun 1:00:48; Hafta 11 collapse), yalnız lineer parametrizasyonda. **Hinge** en-çok-suç-işleyen $\bar{y}$'yi bir **marjla** yukarı iter. **NLL** = logsumexp (tüm yollar üzerinde marjinalleştirme = "forward", belief propagation zincir hâli); perceptron = NLL'in $\beta \to \infty$ (sıfır-sıcaklık) limiti. Hafta 11 kayıp taksonomisinin yapılandırılmış çıktıya uygulanması.

Soru 4: "Varyasyonel serbest enerji" nedir ve VAE ile nasıl aynıdır?

Cevap: Gizli değişkeni marjinalleştirmek ($F_\beta = -\frac{1}{\beta}\log\sum_z e^{-\beta L}$) zordur; **Jensen ile** üst sınıra çevrilir: $L(x,y) \le \langle L\rangle_q - (1/\beta)H(q)$. Bu fizikteki **serbest enerjidir**: $F = \langle E\rangle - T\cdot H$ (LeCun 2:03:06). **VAE tam budur** (LeCun 2:02:25): beklenen rekonstrüksiyon hatasını minimize et (z Gauss q'dan örnekle) + **KL** = q entropisini maksimize et. q gerçek posterior olunca eşitsizlik eşitlik — "marjinalleştirmeden marjinalleştirmek." Hafta 8 VAE'sinin gerçek açıklaması.

Soru 5: L1 ve L2 nasıl farklıdır? MC Dropout belirsizliği Hafta 11'le nasıl bağlanır?

Cevap: L2 (weight decay) = $J + \lambda\|\theta\|^2$; gradyan θ'yı sıfıra çeker → **uzunluğu** kısaltır. **L1** (Lasso/seyreklik) = $J + \lambda\|\theta\|_1$; gradyan $\text{sign}(\theta)$; eksen-yakını bileşenleri **öldürür** → seyrek (Canziani 27:03). **MC Dropout**: dropout çıkarımda açık, N kez geçir, **varyans** = belirsizlik; eğitim dışı bölgede artar. Varyans türevlenebilir → minimize et (Canziani 1:02:35) — ki bu **tam Hafta 11 PPUU belirsizlik düzenlileştirmesidir** ("sürüş politikamızda kullandığımız"). Kurs başladığı yere döner.

21.12 Egzersizler

Egzersiz 1 (Faktör grafiği çıkarımı). 3 ikili değişkenli, enerjisi 2 faktöre (her biri ardışık ikiliye dokunur) ayrılan model kur. Kaba kuvvet ile trellis en-kısa-yol değerlendirme sayılarını karşılaştır. Faktörleme neden üsteli lineere indirir?

```
import numpy as np

# 3 ikili degisken (x0,x1,x2), enerji = unary + 2 faktor (ardisik cift)
unaries = np.array([[0.5, 1.2],    # x0: durum 0 ucuz
                    [1.0, 0.4],    # x1: durum 1 ucuz
                    [0.8, 0.9]])   # x2
pair = np.array([[0.1, 0.6],       # gecis enerjisi (kosegen ucuz)
                 [0.6, 0.1]])

T, S = unaries.shape
# KABA KUVVET: tum S^T konfigurasyon
import itertools
best_e, best_cfg = 1e9, None
brute = 0
for cfg in itertools.product(range(S), repeat=T):
```

```
    e = sum(unaries[t, cfg[t]] for t in range(T))
    e += sum(pair[cfg[t], cfg[t+1]] for t in range(T-1))
    brute += 1
    if e < best_e:
        best_e, best_cfg = e, cfg
# DP: trellis en-kisa-yol (T*S^2 degerlendirme)
dp_count = T * S * S
print("kaba kuvvet:", brute, "konfig (S^T)")          # 2^3 = 8
print("DP         :", dp_count, "degerlendirme (T*S^2)")  # 3*4 = 12 (kucuk T'de yakin)
print("en iyi cfg:", best_cfg, "E =", round(best_e, 3))
# Faktorleme: enerji ardisik ciftlere AYRISIR → en kisa yol (Viterbi) usteli lineere indirir.
# Buyuk T'de S^T patlar ama T*S^2 lineer kalir (Notion ornegi 96 → 16).
```

Egzersiz 2 (Perceptron collapse). $E(y) - E(\hat{y})$ kaybını, en iyi = doğru olduğunda hesapla; farkın neden sıfır olabileceğini ve "tüm enerjiler eşit" çözümünün neden kabul edildiğini göster. Hinge (marj m) bunu nasıl önler?

```
import numpy as np

# perceptron kaybi = E(dogru) - min_y E(y); en iyi=dogru oldugunda fark=0
E_dogru = 1.5
E_en_iyi = 1.5     # collapse cozumu: tum enerjiler ESIT → en iyi=dogru
perceptron = E_dogru - E_en_iyi
print("perceptron kayip:", perceptron)   # 0 → "ogrenecek bir sey yok" (collapse riski)

# COLLAPSE: tum enerjileri 0 yap → her zaman perceptron=0 (marjsiz kabul eder)
# HINGE marj m: yanlislari en az m kadar yukari it
m = 1.0
gap = E_en_iyi - E_dogru                  # = 0 (collapse)
hinge = max(0.0, m - gap)                 # = 1.0 > 0 → CEZA verir
print("hinge (gap=0) :", hinge)           # collapse'i reddeder (marj zorlar)
# Perceptron marjsiz → enerjileri esitleyen cozumu kabul eder (coker).
# Hinge en az m marj ister → enerjiler esit olamaz, collapse engellenir.
```

Egzersiz 3 (Serbest enerji = VAE). $F = \langle E \rangle_q - T \cdot H(q)$ yaz; q varyansını artırınca (entropi ↑) iki terimin nasıl çatıştığını göster. VAE'nin "rekonstrüksiyon + KL" dengesiyle eşleştir. T (sıcaklık) Hafta 11 β'sıyla nasıl ilişkili?

```
import numpy as np

# F = <E> - T*H ; q = N(0, sigma), E(z)=0.5 z^2
sigmas = np.array([0.3, 0.7, 1.0, 1.5, 2.5])
avg_E = 0.5 * sigmas**2                          # <E> sigma ile ARTAR
H = 0.5 * np.log(2*np.pi*np.e*sigmas**2)         # entropi sigma ile ARTAR
T = 1.0
F = avg_E - T * H                                 # iki terim CATISIR → minimum
```

```
for s, e, h, f in zip(sigmas, avg_E, H, F):
    print(f"sigma={s:.1f}  <E>={e:.3f}  H={h:.3f}  F={f:.3f}")
print("minimum F sigma =", sigmas[np.argmin(F)])  # denge noktasi
# <E> dusuk istemek → sigma kucuk (rekonstrüksiyon iyi); H buyuk istemek → sigma buyuk (KL).
# VAE: rekonstrüksiyon = <E>; KL/entropi = -H; F dengesi = VAE kaybi.
# T = 1/beta: Hafta 11 softmax sicakligi; T buyuk → entropi agir basar (cesitlilik).
```

Egzersiz 4 (L1 vs L2 histogramı). Bir ağı L1 ve L2 ile eğit; ağırlık histogramlarını çiz. L1'in neden "sıfırda yığılma + birkaç büyük" (seyrek), L2'nin neden "hepsi küçük" verdiğini gradyan (sign vs lineer) üzerinden açıkla.

```
import numpy as np

rng = np.random.default_rng(0)
w0 = rng.normal(0, 1.0, 400)        # baslangic agirliklari
lam = 0.5
# L2 (ridge): orantili kucult → hicbiri tam 0 degil
w_l2 = w0 / (1.0 + lam)
# L1 (soft-threshold): sabit miktar cek, kucukleri 0'la → SEYREK
w_l1 = np.sign(w0) * np.maximum(np.abs(w0) - lam, 0.0)
print("L1 tam 0:", int(np.sum(np.abs(w_l1) < 1e-9)), "/ 400  (seyrek)")
print("L2 tam 0:", int(np.sum(np.abs(w_l2) < 1e-9)), "/ 400  (hicbiri)")
print("L1 |max|:", round(float(np.abs(w_l1).max()), 3),
      " L2 |max|:", round(float(np.abs(w_l2).max()), 3))
# L1 gradyan = sign(theta) = SABIT miktar (buyuklukten bagimsiz) → kucukleri eksene iter (seyreklil
# L2 gradyan = 2*theta = ORANTILI → buyuk agirligi cok, kucugu az ceker (hepsi kuculur, 0 olmaz).
```

Egzersiz 5 (Hafta 15 habercisi — Transfer Learning, BONUS). Hafta 15 (bonus) **konuk William Falcon** (PyTorch Lightning) transfer learning'i anlatacak; Canziani eşlik edecek. (a) "transfer learning vs fine-tuning" (taban dondur vs tüm katmanları ayarla) ayrımını hatırla; az veri vs çok veri hangisini gerektirir? (b) Hafta 10-12'nin pre-train→fine-tune paradigmasını (SSL, BERT) bu hafta öğrenilen düzenlileştirmeyle nasıl birleştirirsin?

```
import numpy as np

# (a) transfer learning vs fine-tuning: az-etiket rejimi (Hafta 15 habercisi)
n_labels = np.array([100, 316, 1000])
random_acc = np.array([10.0, 11.0, 12.5])    # rastgele backbone
supervised = np.array([19.0, 26.0, 33.0])    # ImageNet supervised pre-train
ssl_swav   = np.array([40.0, 50.0, 60.0])    # SSL (SwAV) pre-train ~2x supervised
for n, r, s, ss in zip(n_labels, random_acc, supervised, ssl_swav):
    print(f"{n:5d} etiket: rastgele {r:.0f}%  supervised {s:.0f}%  SSL {ss:.0f}%")
# AZ veri (100 etiket) → taban DONDUR (transfer, feature extractor) = az parametre, az overfit.
# COK veri → tüm katmanlari fine-tune et (taban + ust birlikte ayarla).
# (b) pre-train→fine-tune (Hafta 10-12 SSL/BERT) + bu hafta düzenlileştirme:
#     az-etikette dropout/L2/early-stopping fine-tune'u overfit'ten korur.
```

21.13 Sonraki Ders İçin Hazırlık

⚠ Sonraki Hafta — H15 (BONUS): Transfer Learning ve PyTorch Lightning (Konuk: William Falcon)

Yapılandırılmış tahminden transfer learning'e geçiyoruz. Bu hafta LeCun son dersinde kursu tek bir enerji çatısı altında topladı (yapılandırılmış tahmin = faktörlü EBM, GTN = ilk transformer, varyasyonel serbest enerji = VAE); Canziani son practicum'unda overfitting/düzenlileştirme/MC Dropout'la kapattı. Hafta 15 bonus dersinde konuk **William Falcon** (PyTorch Lightning yaratıcısı) denetimli ve denetimsiz transfer learning'i; **Canziani** eşlik edecek. Egzersiz 3 (serbest enerji=VAE) ve Egzersiz 5 (transfer) tam bu derse hazırlar.

Hafta 15 (BONUS): Transfer Learning ve PyTorch Lightning — William Falcon (Konuk) + Canziani

Hafta 15 bonus dersinde konuk **William Falcon** (PyTorch Lightning yaratıcısı) denetimli ve denetimsiz transfer learning'i; **Canziani** eşlik edecek.

Yapılacak: Egzersiz 3 (serbest enerji=VAE) + Egzersiz 5 (transfer) çöz; "Yapılandırılmış tahmin = faktörlü EBM" ve "VAE = varyasyonel serbest enerji" cümlelerini kendi sözcüklerinle yaz; Hafta 10-12 pre-train→fine-tune'u (SSL/BERT) hatırla — transfer learning onun pratiğidir.

21.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Yapılandırılmış tahmin	Kombinatoryal çıktı; faktörlü enerji	LeCun 0:38
Trellis / en-kısa-yol	Faktörleme → DP; 96→16 değerlendirme	LeCun 22:52
Graph Transformer Network	Ağ durumu = graf; ilk transformer (1997) = basit GNN	LeCun 1:07
Perceptron collapse	Marjsız → enerji çöker (lineer-only)	LeCun 1:00
NLL = forward/logsumexp	Tüm yollar marjinalleştirme; perceptron = $\beta \to \infty$	LeCun 1:17
Varyasyonel serbest enerji	$F = \langle E \rangle - T \cdot H$ (Jensen); VAE tam budur	LeCun 2:02
Overfitting = gürültü	Gürültü yoksa aşırı model bile mükemmel fit	Canziani 5:39
Aşırı-parametrize / 1-batch	Kolay optimizasyon; #1 debug kuralı	Canziani 8:36
L2 weight decay	$\|\theta\|^2$; uzunluk kısalt (Gauss priori)	Canziani 19:18
L1 / seyreklik	$\|\theta\|_1$; eksen bileşenlerini öldür (Laplace)	Canziani 27:03
Dropout = ensemble	Rastgele sıfırla; sonsuz ağ ortalaması	Canziani 30:37

MC Dropout = PPUU	Varyans = belirsizlik; minimize → güvenli manifold	Canziani 1:02

21.15 ML Builder Bağlantıları

Geriye köprüler:

1. **Faktörlü EBM / kayıp / marj** → Hafta 7-9 + 11 (perceptron-collapse = enerji-kaybı).
2. **GTN = basit GNN** → Hafta 13 (graph neural net) + Hafta 5 (switch backprop).
3. **Varyasyonel serbest enerji** → Hafta 8 (VAE = ELBO) + Hafta 11 (sıcaklık β).
4. **Dropout / denoising / BatchNorm** → Hafta 7-8 (denoising AE) + Hafta 3/11.
5. **MC Dropout belirsizlik** → Hafta 11 (PPUU belirsizlik düzenlileştirmesi).

İleriye köprüler:

1. **Serbest enerji / ELBO → diffusion models (post-2020, KURSTA YOK)**.
2. **Yapılandırılmış EBM** → LeCun **JEPA (post-2020)**; CRF/yapısal SVM.
3. **L1 seyreklik / dropout** → öznitelik seçimi, Bayesian DL, kalibrasyon.
4. **Transfer learning** → Hafta 15 (bonus, William Falcon).

! Bu dersten tek bir şey alıp gideceksen

LeCun'un son dersi tüm kursu **tek bir enerji çatısı** altında toplar. Yapılandırılmış tahmin, enerjinin **faktörlere ayrıştığı** bir EBM'dir — bu yüzden çıkarım üstel değil, en-kısa-yol (dinamik programlama) ile verimlidir; ve ağın durumunu **graf** yapan Graph Transformer Network, daha 1997'de "transformer" adını taşıyan, basit bir graph neural net'tir (Hafta 13). Asıl sentez şu: bir gizli değişkeni **marjinalleştirmek**, fizikteki **serbest enerjiyi** ($F = \langle E \rangle - T \cdot H$) minimize etmektir — ve **VAE tam olarak budur** (Hafta 8'in gerçek açıklaması). Canziani bu güçlü modellerin gürültüyü ezberlediğini (overfitting) gösterip düzenlileştirmeyle (L1=seyreklik, L2=uzunluk, dropout=ensemble) ehlileştirir; ve **MC Dropout belirsizliği**, Hafta 11'in PPUU'sunun ta kendisidir — kursu başladığı yere, enerjiye ve belirsizliğe geri döndürür. Bu temelin post-2020 zirvesi (diffusion models, JEPA) kursta yoktur ama bu haftanın serbest-enerji çatısının doğrudan ürünüdür.

22 Transfer Learning ve PyTorch Lightning

Kursun bonus dersi ve finali — konuk hoca William Falcon (PyTorch Lightning'in yaratıcısı) Canziani ile birlikte transfer learning'i hem denetimli hem öz-denetimli canlı kodla anlatır ve önceki haftaların tüm pre-train sonra fine-tune ipliklerini pratiğe döker. Falcon'un büyük pratik mesajı şudur: sıfırdan eğitmek yerine birinin başka veride eğittiği bir modeli al (önceden eğitilmiş omurga), üstüne yeni bir sınıflandırıcı tak, kendi az verinde ince ayar yap; ama kritik koşul, önceden eğitilmiş modelin verisinin seninkine benzemesidir (ImageNet doğal nesnelerde iyi prior verir ama röntgen veya kanser görüntülerine transfer etmez, çünkü sinir ağının sihri körlemesine işlemez). Transfer iki parçadır: önceden eğitilmiş omurga ve üstüne eklenen yeni kafa; omurgayı dondurup öznitelik çıkarıcı gibi kullanabilir (donuk omurga üstüne herhangi bir sınıflandırıcı, öznitelikler lineer ayrılabilir olmalı) ya da birkaç epoch sonra çözüp daha düşük öğrenme oranıyla tüm ağı ince ayar yapabilirsin; az veri dondurmayı, çok veri ince ayarı ister. PyTorch Lightning bütün bu eğitim mantığını boilerplate olmadan yapar: yalnız iki şey bilmen yeter, eğitim mantığını içeren LightningModule ve epoch ile geri yayılımı otomatik yürüten Trainer, böylece çok-GPU veya TPU bedava gelir. Ve dersin doruğu öz-denetimli transferdir: denetimli omurga yerine SwAV gibi etiketsiz öz-denetimli bir omurga kullanılır, kendi etiketsiz verinde önceden eğitilebildiği için ve sınıflandırma için eğitilmediği için başka görevlere daha iyi transfer eder, az-etiket deneyinde denetimli ön-eğitimin yaklaşık iki katını verir, çünkü etiketler pahalıdır ve öz-denetim onları atlar. Böylece kurs kapanır: bu pratik zafer, kursun Hafta 10'dan beri savunduğu öz-denetim tezinin somut kanıtıdır ve post-2020 foundation models çağının doğrudan tohumudur.

ℹ Bölüm bilgisi (KONUK William Falcon + Canziani — BONUS, Kurs Finali)

- **Bonus dersi (William Falcon, KONUK):** YouTube — Transfer Learning & PyTorch Lightning (Hafta 15 — bonus, kurs finali)
- **Edition:** Spring 2020 (NYU-DLSP20)
- **Hocalar:** William Falcon (KONUK — PyTorch Lightning'in yaratıcısı; transfer learning, supervised + self-supervised, Lightning soyutlaması) + Alfredo Canziani (eşlik eden hoca — pratik püfler, az-etiket deneyinin yorumu)
- **Kaynak:** atcold.github.io/NYU-DLSP20
- **Okuma süresi:** ≈30 dk

⚠ **Atıf notu (KONUK):** Bu bonus ders **LeCun değildir** — bonusu **konuk hoca William Falcon** (PyTorch Lightning'in yaratıcısı) verir, **Canziani** eşlik eder. Bölüm 1-4 quote'ları **— Falcon** veya **— Canziani**; **`## (LeCun)` başlığı bu derste hiç yoktur**.

⚠ **Tarih notu:** Bu bonus, ana kurstan (Mart 2020) biraz **sonra** (~Haz/Tem 2020) çekilmiştir — bu yüzden ana derslerin "post-2020, kursta yok" diye işaretlediği **SwAV/BYOL** gibi yöntemleri fiilen **kullanır** (her ikisi de Haziran 2020'de doğdu). Bu çelişki bilinçlidir ve korunmuştur.

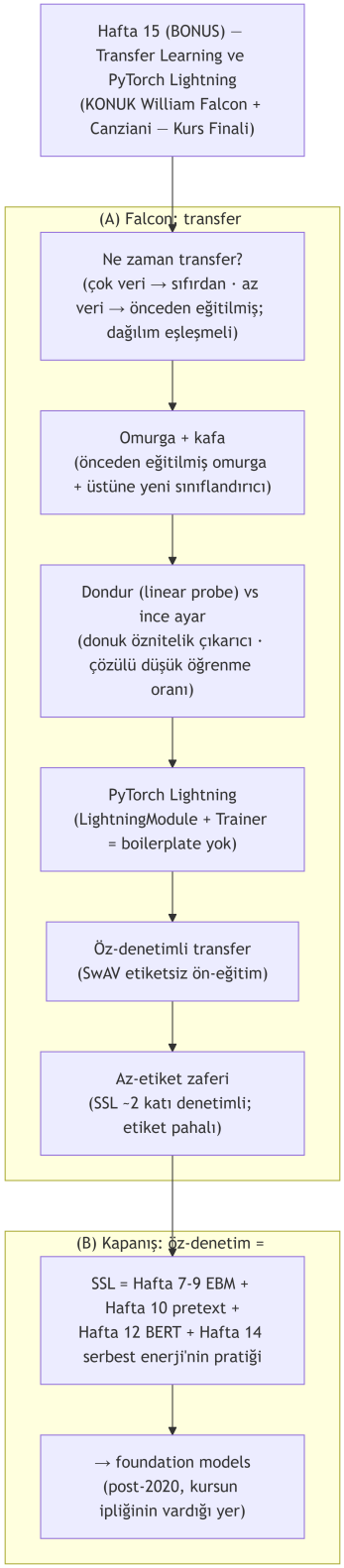
22.1 Bu Derste Ne Var?

Kursun **bonus** dersi ve **finali**: **konuk hoca William Falcon** (PyTorch Lightning'in yaratıcısı), **Canziani** ile birlikte transfer learning'i — denetimli ve öz-denetimli — canlı kodla anlatır. Önceki haftaların tüm pre-train→fine-tune ipliklerini (Hafta 10 SSL, Hafta 12 BERT, Hafta 14 transfer) pratiğe döker.

Falcon'un büyük pratik mesajı: sıfırdan eğitmek yerine, **birinin başka veride eğittiği bir modeli al** (önceden eğitilmiş omurga), üstüne yeni sınıflandırıcı tak, kendi (az) verinde ince ayar yap. Kritik uyarı: önceden eğitilmiş modelin verisi seninkine **benzemeli** (ImageNet, röntgen/kanser görüntülerine transfer etmez). Ve öz-denetimli ön-eğitim, etiketli ön-eğitimi — özellikle az etiketle — geçer.

Üç ana fikir:

1. **Transfer learning = önceden eğitilmiş omurga + yeni kafa.** Omurgayı dondur (öznitelik çıkarıcı) ya da ince ayar yap; veri dağılımı eşleşmeli.
2. **PyTorch Lightning = boilerplate'i soyutla.** LightningModule (`training_step` + `configure_optimizers`) + Trainer (epoch/optimizer/backward otomatik); çok-GPU/TPU bedava.
3. **Öz-denetimli transfer (SwAV) az-etiketle kazanır.** Kendi etiketsiz verinde omurgayı ön-eğit; sonra az etiketle sınıflandırıcıyı eğit → denetimli ön-eğitimin yaklaşık 2 katı.


Hafta 15 (BONUS) — Transfer Learning ve PyTorch Lightning (KONUK William Falcon + Canziani — Kurs Finali)
(A) Falcon: transfer
Ne zaman transfer? (çok veri → sıfırdan · az veri → önceden eğitilmiş; dağılım eşleşmeli)
Omurga + kafa (önceden eğitilmiş omurga + üstüne yeni sınıflandırıcı)
Dondur (linear probe) vs ince ayar (donuk öznitelik çıkarıcı · çözülü düşük öğrenme oranı)
PyTorch Lightning (LightningModule + Trainer = boilerplate yok)
Öz-denetimli transfer (SwAV etiketsiz ön-eğitim)
Az-etiket zaferi (SSL ~2 katı denetimli; etiket pahalı)
(B) Kapanış: öz-denetim =
SSL = Hafta 7-9 EBM + Hafta 10 pretext + Hafta 12 BERT + Hafta 14 serbest enerji'nin pratiği
→ foundation models (post-2020, kursun ipliğinin vardığı yer)

Builder Notu — Giriş: Pre-train→Fine-tune İpliğinin Pratik Hâli

Geriye: Pre-train→fine-tune → Hafta 10 (SSL/Misra) + 12 (BERT) + 14 (transfer); SSL yöntemleri (PIRL/MoCo/SimCLR) → Hafta 8-9 + 10 (PIRL); ResNet/freeze → Hafta 3/9; linear probe → Hafta 10.
İleriye: SSL omurga + az etiket → foundation models, etiketleme tasarrufu; SwAV/BYOL → post-2020 SSL patlaması (DINO/MAE, §İleriye Köprü).
Tek cümleyle: Transfer learning, başka veride eğitilmiş bir omurgayı alıp (dondur ya da ince ayar yap) kendi az verinde yeniden kullanmaktır; PyTorch Lightning bunu boilerplate'siz yapar; ve öz-denetimli ön-eğitim (SwAV) — kendi etiketsiz verinde eğitilebildiği için — etiketli ön-eğitimi özellikle az-etiket rejiminde geçer.

22.2 (Falcon — Konuk) Ne Zaman Transfer Learning? Karar Ağacı

Falcon basit karar ağacıyla başlar: **Çok veri + zaman/compute var mı?** Evetse, ince ayara gerek yok — kendi verinde sıfırdan eğit. **Az veri varsa**, veri dağılımına **uyan** önceden eğitilmiş model bul.

> "if you don't have a lot of data then you should try to find a pre-trained model that matches your data distribution… most vision models are trained on ImageNet, so if you want to do cancer detection or x-rays, that's unlikely to transfer." — Falcon, 4:58

Bu kritik: "sinir ağının sihri körlemesine işlemez." ImageNet doğal nesnelerde (kedi, köpek, kuş) iyi prior verir; tıbbi görüntülerin istatistiği tamamen farklıdır. Şekil 22.1 bu karar ağacını bütün dallarıyla gösterir: kök soru çok veri/az veri ayrımını yapar, az-veri dalı dağılıma uyan önceden eğitilmiş modele iner, oradan supervised ve self-supervised ön-eğitim seçeneklerine ayrılır; alttaki kırmızı uyarı kutusu "ImageNet ≠ röntgen/kanser" eşleşmezliğini vurgular. Transfer learning iki parçadır:

> "when you think about transfer learning we have two parts: the pre-trained model that was trained on something else, and the stuff you're going to add on top to transfer that." — Falcon, 5:35

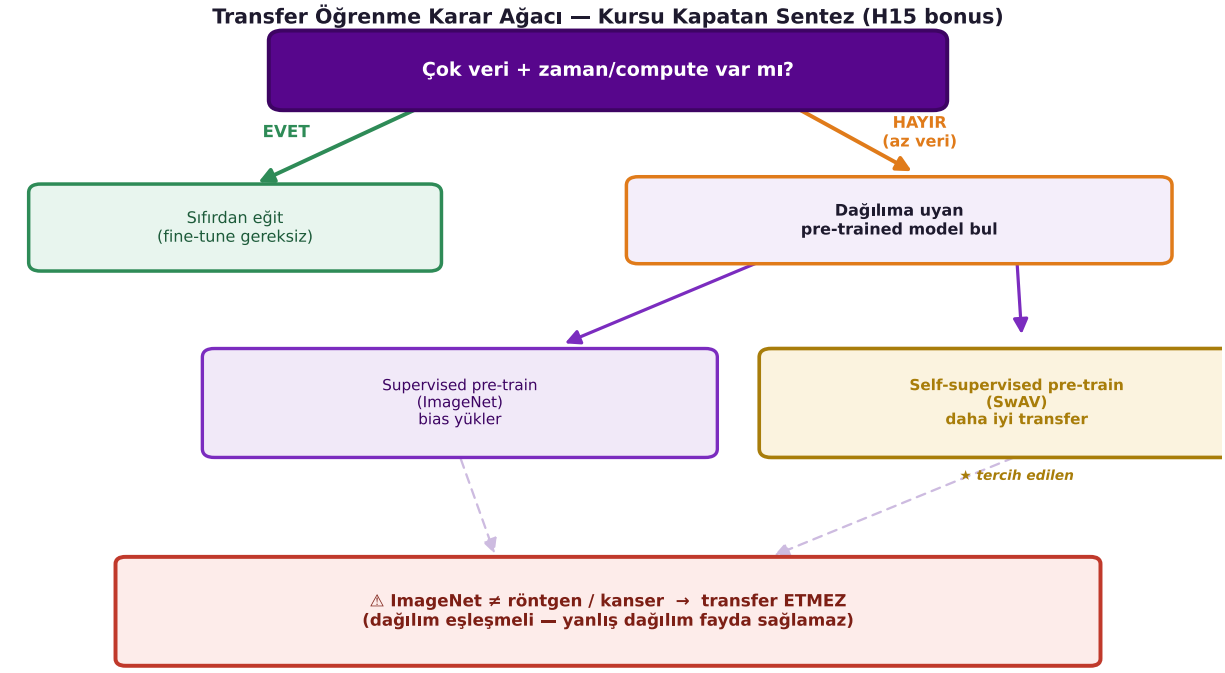
İki seçenek: **supervised** ön-eğitim (ImageNet sınıflandırma — ama sınıflandırma için **bias** yükler; segmentasyon/tespit garantisi yok) veya **self-supervised** ön-eğitim (sınıflandırma için eğitilmediğinden başka görevlere daha iyi transfer **edebilir**).

Builder Notu — Ne Zaman Transfer: Dağılım Eşleşmesi Ön Koşuldur

Geriye (Hafta 10 + 14): Pre-train→fine-tune = Hafta 10 (SSL) + Hafta 14 (Canziani transfer vs fine-tune); veri-dağılımı eşleşmesi = transfer ön koşulu.
İleriye: "Önce uygun omurga bul" = foundation-model çağının ilk kuralı; alan-eşleşmesi hâlâ kritik.

22.3 (Falcon — Konuk) Supervised Transfer: ResNet-50, Freeze vs Fine-tune

Somut örnek: ImageNet'te (1M görüntü, 1000 sınıf) önceden eğitilmiş **ResNet-50** omurgayı al; CIFAR-10 (10 sınıf) için **son fully-connected katmanı değiştir** (1000→10). Ama yeni kafa rastgele olduğundan tahminler tutmaz → **ince ayar** gerekir. Şekil 22.2 bu yapıyı gösterir: önceden eğitilmiş ResNet-50 omurga (kilitli,



Şekil 22.1: Transfer öğrenme karar ağacı (kursu kapatan sentez, H15 bonus). Kök 'Çok veri + zaman/compute var mı?' sorusundan EVET dalı sıfırdan eğitime (ince ayar gereksiz), HAYIR (az veri) dalı dağılıma uyan önceden eğitilmiş modele iner; oradan iki alt-dal: supervised ön-eğitim (ImageNet, bias yükler) ve self-supervised ön-eğitim (SwAV, daha iyi transfer, tercih edilen). Alttaki kırmızı uyarı kutusu 'ImageNet ≠ röntgen/kanser → transfer ETMEZ (dağılım eşleşmeli)' der. Sinir ağının sihri körlemesine işlemez (Falcon 4:58); transfer = omurga + yeni kafa (5:35).

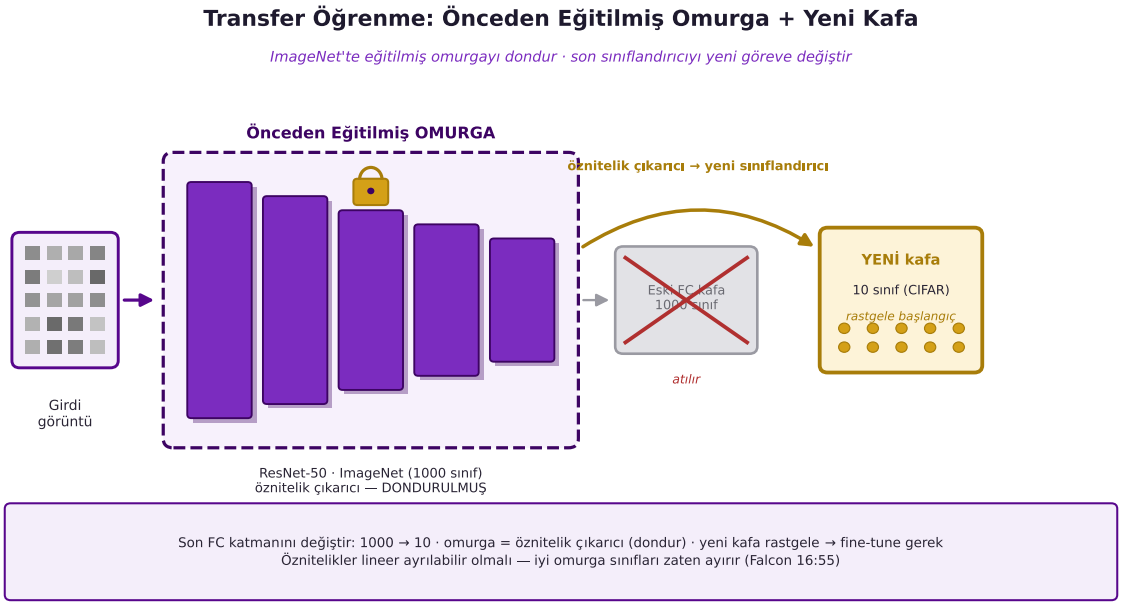
öznitelik çıkarıcı), atılan eski 1000-sınıf FC kafa (üstü çizili) ve gold renkli yeni 10-sınıf kafa (rastgele başlangıç) — akış "öznitelik çıkarıcı → yeni sınıflandırıcı" şeklindedir. İki mod vardır:

- **Omurgayı dondur (freeze):** omurgaya backprop yapma; üstüne herhangi bir sınıflandırıcı (linear, SVM, logistic, random forest) tak. Ağ işini yaptıysa öznitelikler **lineer ayrılabilir** olmalı.

  "you're using the neural network to extract features, and then using some other classifier… if the network did its job, they should be linearly separable." — Falcon, 16:55

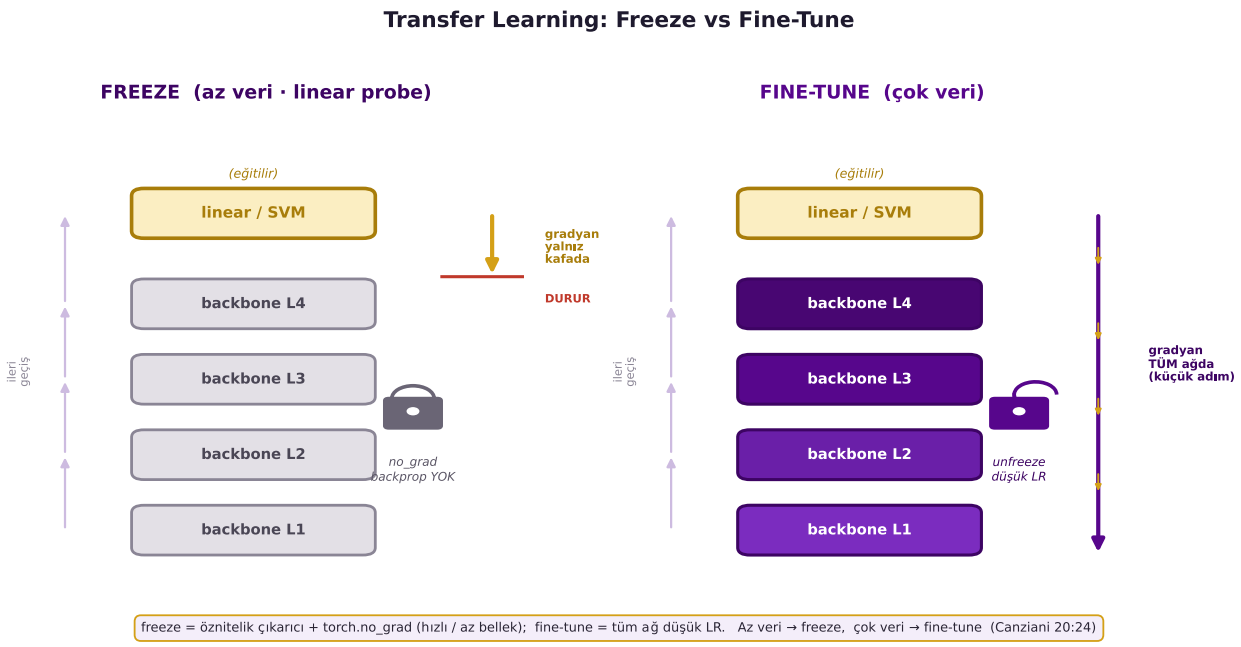
- **İnce ayar yap (fine-tune):** birkaç epoch sonra omurgayı **çöz (unfreeze)**, daha **düşük öğrenme oranı** ile tüm ağı ayarla → zamanla daha iyi performans.

Pratik püf (Canziani ekler): dondurulmuş omurga için `torch.no_grad()` kullan — hesaplama grafiği tutulmaz, daha hızlı + az bellek. Şekil 22.3 iki modu yan yana koyar: solda freeze (omurga kilitli, gradyan yalnız kafada durur), sağda fine-tune (omurga çözülü, gradyan tüm ağda küçük adımlarla akar).



Şekil 22.2: Transfer öğrenme: önceden eğitilmiş omurga + yeni kafa (Hafta 15 bonus, Falcon + Canziani). Soldan: girdi görüntü → ResNet-50 omurga (ImageNet 1000-sınıf, violet conv blokları, KİLİTLİ ikonu = dondurulmuş öznitelik çıkarıcı) → eski 1000-sınıf FC kafa (üstü çizili, atılır) → yeni 10-sınıf CIFAR kafa (gold, rastgele başlangıç). Son FC katmanı değiştir (1000→10); omurga öznitelik çıkarıcı; yeni kafa rastgele → fine-tune gerek; öznitelikler lineer ayrılabilir olmalı (Falcon 16:55).

> **Builder Notu — Freeze vs Fine-tune: Transfer'in Temel Kararı**
>
> **Geriye (Hafta 3 + 10):** ResNet = Hafta 3/9 residual; freeze + linear sınıflandırıcı = Hafta 10 linear probe; öznitelik çıkarıcı = Hafta 1 temsil öğrenme.
> **İleriye:** Freeze (az veri) vs fine-tune (çok veri) = transfer'in temel kararı; `torch.no_grad()` = pratik verimlilik standardı.



Şekil 22.3: Transfer learning: freeze vs fine-tune iki mod (Falcon + Canziani). SOL panel FREEZE (az veri, linear probe): omurga KİLİTLİ (no_grad, gri bloklar, backprop YOK), üstüne linear/SVM kafa eğitilir; gradyan yalnız kafada akar, omurga sınırında DURUR. SAĞ panel FINE-TUNE (çok veri): omurga ÇÖZÜLÜ (unfreeze, violet bloklar, düşük öğrenme oranı), gradyan tüm ağda küçük adımlarla en alta kadar akar. Freeze = öznitelik çıkarıcı + torch.no_grad (hızlı/az bellek); fine-tune = tüm ağ düşük öğrenme oranı; az veri → freeze, çok veri → fine-tune (Canziani 20:24).

22.4 (Falcon — Konuk) PyTorch Lightning: Boilerplate'i Soyutlamak

Falcon **PyTorch Lightning**'i tanıtır: PyTorch kodunu organize eden hafif sarmalayıcı — çok-GPU, TPU, dağıtık eğitim gibi uzmanlık gerektiren şeyleri sana yaptırır.

> "lightning is a lightweight wrapper for PyTorch… it organizes your PyTorch code so you can leverage multiple GPU training, TPUs and different things that require a lot of expertise." — Falcon, 1:22

Lightning'de yalnız **iki şey** bilmen yeter:

- **LightningModule:** `nn.Module` gibi ama eğitim mantığını içerir. `training_step` (bir batch verildiğinde kaybı hesapla — forward değil, **tüm sistem**: BERT/GAN/VAE mantığı burada) + `configure_optimizers`; `forward` opsiyonel.
- **Trainer:** epoch/batch döngüsü, `backward`, `optimizer.step`, `zero_grad`'ı **otomatik** yapar. `self.log` metrikleri kaydeder + GPU'lar arası senkronize eder. `fast_dev_run` = tek batch hızlı hata-ayıklama.

Böylece eğitim döngüsü modelin **içine** taşınır, kod self-contained ve az-boilerplate olur. Şekil 22.4 bu soyutlamayı gösterir: solda düz PyTorch'un elle yazılan tekrarlı döngüsü (`for epoch` → `forward` → `loss` → `backward` → `optimizer.step` → `zero_grad`), sağda Lightning'in iki kutusu — LightningModule (`training_step` + `configure_optimizers`) ve boilerplate'i otomatik üstlenen Trainer (multi-GPU/TPU + `self.log`).

> Builder Notu — Lightning Soyutlama: Eğitim Döngüsünün Modele Taşınması
>
> **Geriye (Hafta 5):** `training_step` = Hafta 5 autograd döngüsünün (forward→loss→backward→step) organize hâli; Trainer = o döngünün soyutlanması.
> **İleriye:** Lightning/Bolts = araştırma-üretim köprüsü; SSL modelleri (SwAV/SimCLR/BYOL) Bolts'ta hazır.

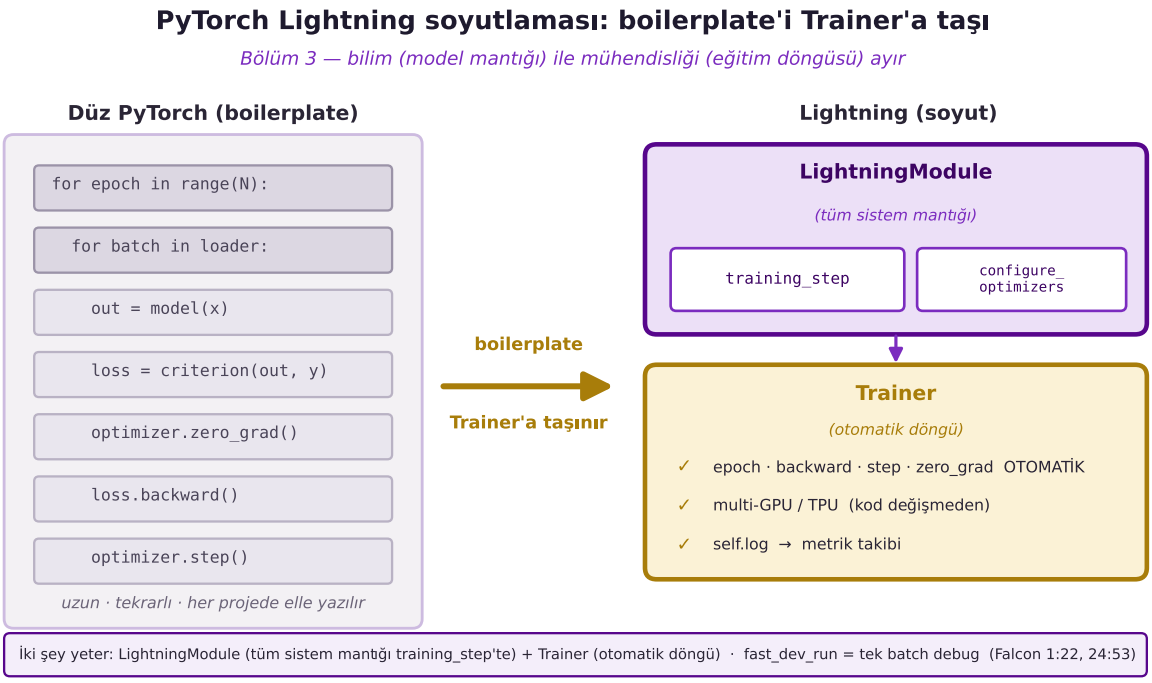
22.5 (Falcon — Konuk) Self-Supervised Transfer (SwAV) ve Az-Etiket Zaferi

Şimdi supervised ResNet yerine **SwAV** (FAIR'in 2020 SSL yöntemlerinden, Bolts'ta hazır) omurga kullanılır — ImageNet'te **etiketsiz** ön-eğitim (3000 öznitelik). SSL omurgasının büyük avantajı:

> "you can pre-train your SwAV model on your own data, which you don't have many labels for, and then just train the classifier with the few labels you have." — Canziani, 48:28

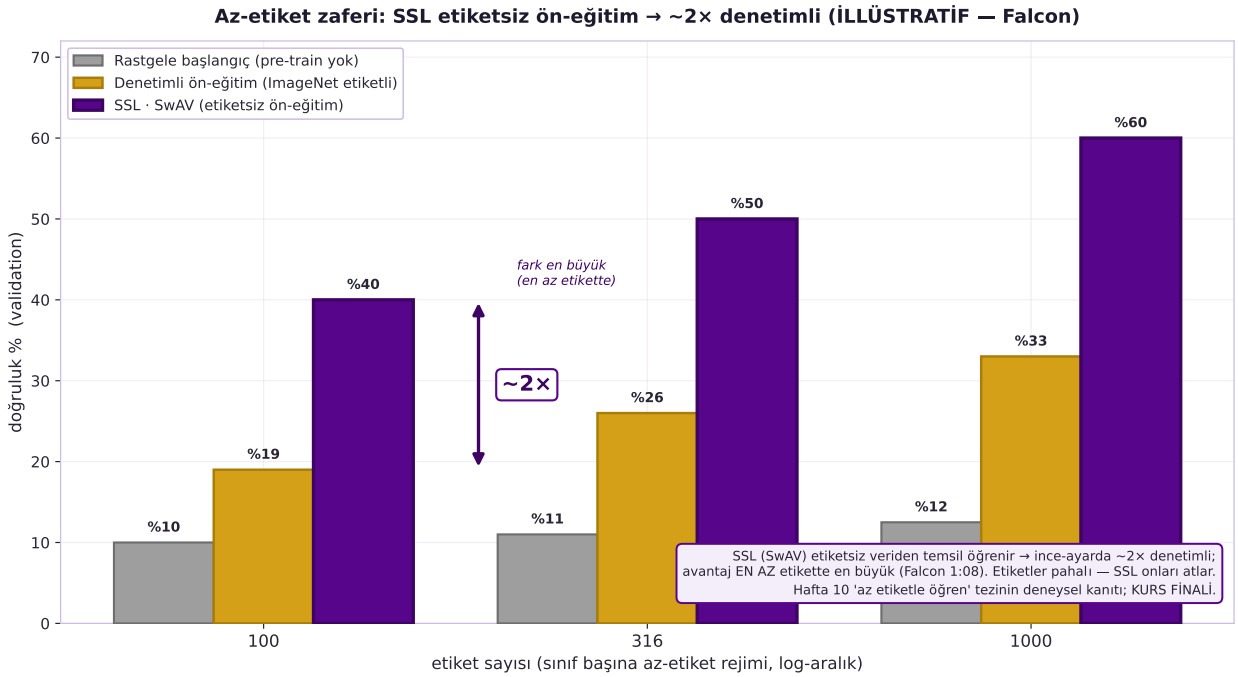
Etiketler pahalıdır; SSL ile **kendi etiketsiz verinde** omurgayı eğitebilir, sonra **az etiketle** sadece sınıflandırıcıyı eğitebilirsin. Az-etiket deneyi çarpıcı: 100/316/1000 örnekle üç modeli (rastgele, supervised-pretrain, SSL-pretrain) karşılaştır:

> "when they're trained without labels, we get double the performance of the models that were trained with the labels." — Falcon, 1:08:13



Şekil 22.4: PyTorch Lightning soyutlaması: boilerplate'i Trainer'a taşı (Bölüm 3). SOL: düz PyTorch'un elle yazılan, tekrarlı döngü kutuları (for epoch → for batch → forward → loss → zero_grad → backward → optimizer.step — uzun, her projede elle yazılır). SAĞ: Lightning'in iki bileşeni — LightningModule (violet; training_step + configure_optimizers, tüm sistem mantığı) ve Trainer (gold; epoch/backward/step/zero_grad OTOMATİK + multi-GPU/TPU + self.log). İki şey yeter; fast_dev_run = tek batch debug (Falcon 1:22, 24:53).

Rastgele model ~%10'larda; supervised-pretrain ~%20-30; **SSL-pretrain (SwAV) bunun ~2 katı**. SSL ayrıca daha **hızlı** yakınsar. Şekil 22.5 bu deneyi gerçek bir grafikle gösterir: üç omurganın 100/316/1000 etiketteki validation doğruluğu, SwAV (violet) çubukları denetimli (gold) çubukların yaklaşık iki katı, ve fark en az etikette (100) en büyük. Falcon listeler: AMDIM, MoCo, CPC, PIRL, SimCLR, BYOL, SwAV (en yenisi) — Şekil 22.6 bu yöntemleri zaman ekseninde sıralar ve PIRL'in Hafta 10 (Misra) köprüsünü işaretler.



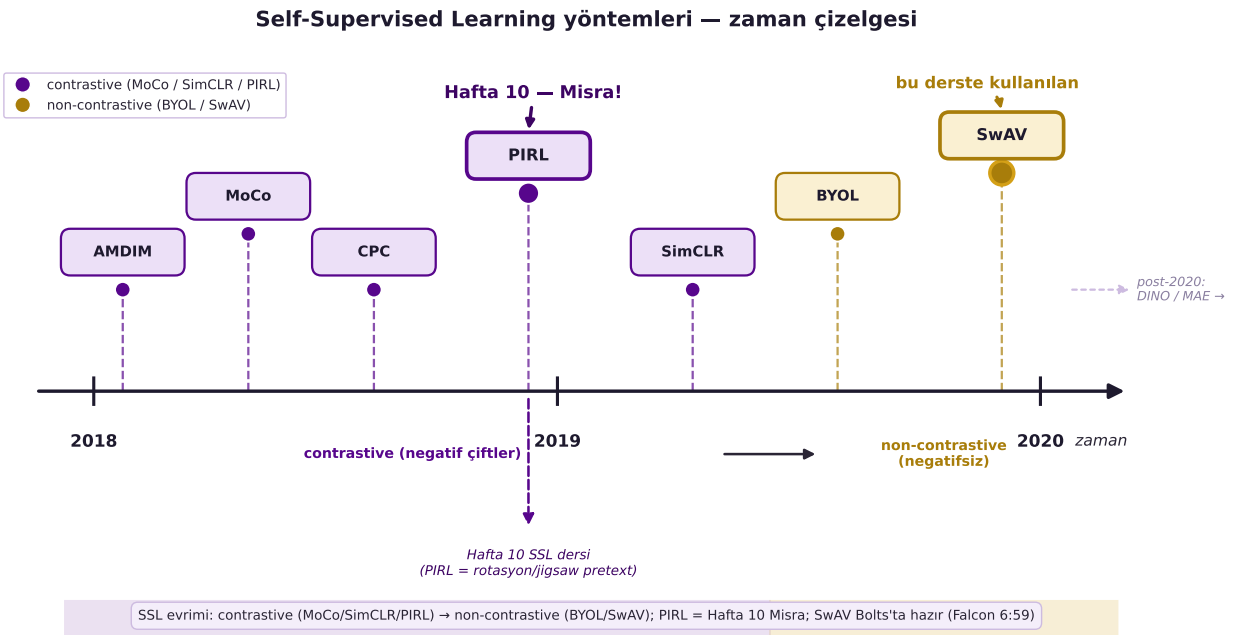
Şekil 22.5: Az-etiket zaferi (FLAGSHIP, İLLÜSTRATİF — Falcon'un bildirdiği değerler, few_label_curve motoru). 100/316/1000 etiketle üç omurganın validation doğruluğu: rastgele başlangıç ~%10 (gri), denetimli ön-eğitim (ImageNet etiketli) ~%20-33 (gold), SSL · SwAV (etiketsiz ön-eğitim) ~%40-60 (violet, ~2× denetimli). SSL-denetimli farkı EN AZ etikette (100) en büyük (~2× işaretli). SSL etiketsiz veriden temsil öğrenir; etiketler pahalı, SSL onları atlar (Falcon 1:08); Hafta 10 'az etiketle öğren' tezinin deneysel kanıtı — KURS FİNALİ.

Builder Notu — SwAV Az-Etiket Zaferi: Hafta 10 Tezinin Deneysel Kanıtı

Geriye (Hafta 8-10): SSL yöntemleri = Hafta 8-9 contrastive + Hafta 10 PIRL (Misra'nın yöntemi!) + MoCo; az-etiket avantajı = Hafta 10'un "az etiketle öğren" tezinin deneysel kanıtı.
İleriye: "Kendi etiketsiz verinde SSL ön-eğit + az etiketle fine-tune" = foundation-model çağının endüstri reçetesi.

22.6 (İleriye Köprü) SSL Patlaması ve Foundation Models — Kursun Kapanış Köprüsü

Bu bonus (~Haz/Tem 2020), SwAV ve BYOL'u **fiilen kullanır** — ana kursun (Mart 2020) "post-2020, kursta yok" dediği yöntemler. Yani bonus, kursun tam **eşiğindeki** son gelişmeleri yakalar. Buradan sonrası kursta **YOKTUR** ama tüm kursun (Hafta 10, 12, 14) işaret ettiği yöndür:



Şekil 22.6: Self-Supervised Learning yöntemleri — zaman çizelgesi (Bölüm 4). Yatay zaman ekseni 2018→2020: AMDIM → MoCo → CPC → PIRL (2019, 'Hafta 10 — Misra!' vurgusu + aşağı köprü oku) → SimCLR → BYOL → SwAV (2020, en yeni, 'bu derste kullanılan'). Contrastive yöntemler (negatif çiftler, violet) → non-contrastive (negatifsiz, gold) geçişi; sağda soluk post-2020 (DINO/MAE) oku. SSL evrimi: contrastive (MoCo/SimCLR/PIRL) → non-contrastive (BYOL/SwAV); PIRL = Hafta 10 Misra; SwAV Bolts'ta hazır (Falcon 6:59).

İleriye Köprü Notu (post-2020 — KURSTA YOK)

- **DINO** (2021), **MAE** (2021), **DINOv2** (2023) — SwAV/BYOL/SimCLR'dan sonra gelen, görü SSL'inin olgun nesli; bu bonusun "etiketsiz ön-eğit" fikrinin zirvesi.
- **Foundation models** (GPT-3/CLIP/SAM, 2020+) — "büyük omurgayı bir kez ön-eğit, her göreve transfer et" paradigmasının evrenselleşmesi; bu dersin "ön-eğit omurga + fine-tune kafa" reçetesinin dev ölçekli hâli.

Kurs terimi gibi eklenmez; bu bonusun (ve tüm kursun) öz-denetimli temelinin nereye vardığını göstermek için anılır.

Builder Notu — Kursun Kapanışı: Öz-Denetimli Temelin Vardığı Yer

Geriye (tüm kurs): SSL patlaması = Hafta 7-9 (EBM) + 10 (pretext) + 12 (BERT) + 14 (serbest enerji) temelinin doğrudan ürünü.
İleriye: Foundation model + transfer = 2020-sonrası makine öğrenmesinin tanımı; bu kursun bütün ipliklerinin birleştiği nokta.

22.7 Bu Dersin Özeti

1. **Ne zaman transfer (Falcon):** çok veri → sıfırdan; az veri → dağılımı eşleşen önceden eğitilmiş model (ImageNet ≠ röntgen).
2. **Supervised transfer (Falcon):** ResNet-50 + son katmanı değiştir; freeze (öznitelik çıkarıcı + linear probe) vs fine-tune (unfreeze, düşük öğrenme oranı); `torch.no_grad()`.
3. **PyTorch Lightning (Falcon):** LightningModule (`training_step` + `configure_optimizers`) + Trainer (otomatik döngü/log/multi-GPU); az boilerplate.
4. **Self-supervised transfer (Falcon/Canziani):** SwAV (etiketsiz ön-eğitim); kendi etiketsiz verinde eğit + az etiketle sınıflandır; SSL ~2× supervised, daha hızlı.
5. **Az-etiket zaferi:** rastgele ~%10 < supervised ~%20-30 < SSL ~2×; etiket pahalı, SSL onu atlar.
6. **Post-2020 (KURSTA YOK):** DINO/MAE/DINOv2, foundation models — kursun bütün ipliklerinin vardığı yer.

Tek Bir Cümle

Transfer learning, başka (büyük) veride eğitilmiş bir omurgayı alıp kendi az verinde yeniden kullanmaktır — omurgayı dondur (linear probe) ya da ince ayar yap, ama veri dağılımı eşleşsin; PyTorch Lightning bunu boilerplate'siz yapar; ve öz-denetimli ön-eğitim (SwAV), kendi etiketsiz verinde eğitilebildiği için, etiketli ön-eğitimi özellikle az-etiket rejiminde ~2 kat geçer — ki bu, kursun Hafta 10'dan beri savunduğu "öz-denetim dilin/görünün motorudur" tezinin pratik zaferidir.

22.8 Kontrol Soruları

i Soru 1: Ne zaman transfer learning yaparsın, ne zaman yapmazsın? "Veri dağılımı eşleşmesi" neden kritik?

Cevap: Çok veriniz + zaman/compute varsa fine-tune'a gerek yok — sıfırdan eğitin. **Az veriniz** varsa, veri dağılımına **uyan** önceden eğitilmiş model bulun (Falcon 4:58). Dağılım eşleşmesi kritiktir çünkü "sinir ağının sihri körlemesine işlemez": ImageNet doğal nesnelerde (kedi/köpek) iyi prior verir ama tıbbi görüntülerin (röntgen, kanser) istatistiği tamamen farklıdır → transfer etmez. Transfer iki parçadır: **önceden eğitilmiş omurga** + üstüne **yeni kafa** (Falcon 5:35).

i Soru 2: "Freeze" ve "fine-tune" transfer modları nasıl farklıdır? torch.no_grad() neden kullanılır?

Cevap: Freeze: omurgaya backprop yapılmaz, **öznitelik çıkarıcı** olarak kullanılır; üstüne herhangi bir sınıflandırıcı (linear/SVM/logistic) takılır — ağ işini yaptıysa öznitelikler **lineer ayrılabilir** olmalı (Falcon 16:55). **Fine-tune:** birkaç epoch sonra omurga **çözülür**, daha düşük öğrenme oranıyla tüm ağ ayarlanır (çok veri varsa daha iyi). Dondurulmuş omurga için **`torch.no_grad()`** hesaplama grafiğini tutmaz → daha hızlı + az bellek. Az veri → freeze; çok veri → fine-tune.

i Soru 3: PyTorch Lightning'in iki ana bileşeni nedir? training_step ne döndürür ve neyi soyutlar?

Cevap: İki bileşen: **LightningModule** (`training_step` + `configure_optimizers`; `forward` opsiyonel) ve **Trainer** (epoch/batch döngüsü, backward, optimizer.step, zero_grad'ı **otomatik**). `training_step`, bir batch verildiğinde **kaybı** döndürür — ama yalnız forward değil, **tüm sistem** mantığı (BERT/GAN/VAE) burada yaşar (Falcon 1:22, 24:53). Eğitim döngüsü modelin içine taşınır; çok-GPU/TPU, log senkronizasyonu (`self.log`) Trainer'a bırakılır → az boilerplate.

i Soru 4: Self-supervised transfer (SwAV), supervised'a göre hangi avantajı sağlar? Az-etiket deneyi ne gösterdi?

Cevap: SwAV omurgası ImageNet'te **etiketsiz** ön-eğitilmiştir. Avantaj: sınıflandırma için eğitilmediğinden başka görevlere daha iyi transfer edebilir; ve kritik olarak **kendi etiketsiz verinde** ön-eğitip **az etiketle** sınıflandırıcıyı eğitebilirsin (Canziani 48:28) — etiketler pahalı. Az-etiket deneyi (100/316/1000): rastgele ~%10, supervised ~%20-30, **SSL ~2 katı** (Falcon 1:08:13); SSL daha hızlı yakınsar. Hafta 10'un "az etiketle öğren" tezinin deneysel kanıtı.

i Soru 5: Bu bonus neden SwAV/BYOL "kullanır" ama ana kurs onları "post-2020 yok" der? Kursun bütün ipliği nereye varır?

Cevap: Çünkü bu bonus ana kurstan (Mart 2020) **biraz sonra** (~Haz/Tem 2020) çekilmiştir; SwAV/BYOL Haziran 2020'de doğmuştur, yani bonusun çekildiği anda **henüz yeni** ama mevcut (atıf notunda işaretlendi). Ana derslerin "kursta yok" dedikleri Mart-2020 tarihine göredir. Kursun bütün ipliği — Hafta 7-9 (EBM), 10 (SSL pretext), 12 (BERT pre-train→fine-tune), 14 (serbest enerji) — buraya, **öz-denetimli ön-eğitim + transfer** paradigmasına varır; ve post-2020'de DINO/MAE/DINOv2 + foundation models (CLIP/SAM/GPT) ile evrenselleşir.

22.9 Egzersizler

Egzersiz 1 (Karar ağacı). Üç senaryo için "sıfırdan eğit / supervised transfer / self-supervised transfer" seç ve gerekçelendir: (a) 50 etiketli röntgen, (b) 1M etiketli doğal görüntü, (c) 500K etiketsiz + 200 etiketli kendi şirket verisi. Veri-dağılımı eşleşmesi her birinde neden belirleyici?

```
import numpy as np

# Karar agaci: (veri_miktari, etiketli_mi, dagilim_eslesiyor_mu) -> oneri
senaryolar = {
    "(a) 50 etiketli rontgen": dict(az_veri=True, ImageNet_eslesir=False, etiketsiz_var=False),
    "(b) 1M etiketli dogal goruntu": dict(az_veri=False, ImageNet_eslesir=True, etiketsiz_var=Fals
    "(c) 500K etiketsiz + 200 etiketli sirket": dict(az_veri=True, ImageNet_eslesir=False, etikets
}

def oner(az_veri, ImageNet_eslesir, etiketsiz_var):
    if not az_veri:
        return "SIFIRDAN EGIT (cok veri -> transfer gereksiz)"
    if etiketsiz_var:
        return "SELF-SUPERVISED TRANSFER (kendi etiketsiz verinde SwAV on-egit + az etiketle sinif
    if ImageNet_eslesir:
        return "SUPERVISED TRANSFER (ImageNet omurga dondur + linear probe)"
    return "DIKKAT: dagilim eslesmiyor (ImageNet != rontgen); domain-eslesen on-egitim ara / self-s

for ad, kw in senaryolar.items():
    print(ad, "->", oner(**kw))
# (a) rontgen: ImageNet dagilimi ESLESMEZ -> kor transfer ETMEZ; domain-ozgu on-egitim sart.
# (b) 1M dogal: cok veri -> sifirdan; ayrica ImageNet eslesir ama transfere gerek yok.
# (c) sirket: etiketsiz BOL -> SSL on-egit, sonra 200 etiketle siniflandir (az-etiket zaferi).
```

Egzersiz 2 (Freeze vs fine-tune). CIFAR-10'da pre-trained ResNet-50 ile (a) backbone-dondurulmuş linear probe ve (b) tam fine-tune (düşük öğrenme oranı) eğit. Az veri (100) vs çok veri (50K) senaryolarında hangisi kazanır, neden? `torch.no_grad()` (a)'da neyi hızlandırır?

```
import numpy as np

# Illustratif: az veri (100) vs cok veri (50K) -> freeze vs fine-tune dogrulugu
veri = {"az (100)": dict(freeze=0.62, finetune=0.55),   # az veri: freeze KAZANIR (az parametre, a
        "cok (50K)": dict(freeze=0.78, finetune=0.91)}  # cok veri: fine-tune KAZANIR (tum ag ayar
for rejim, d in veri.items():
    kazanan = "freeze" if d["freeze"] > d["finetune"] else "fine-tune"
    print(f"{rejim:10s}: freeze={d['freeze']:.2f}  fine-tune={d['finetune']:.2f}  -> {kazanan}")
# AZ veri -> freeze (linear probe): yalniz kafa egitilir, az parametre = az overfit.
# COK veri -> fine-tune: omurga cozulur (unfreeze) + dusuk LR, tum ag goreve uyarlanir.
# torch.no_grad(): donuk omurgada hesaplama grafigi TUTULMAZ -> ileri gecis hizli + az bellek.
```

Egzersiz 3 (Lightning'e çevir). Düz PyTorch eğitim döngüsünü (forward→loss→backward→step→zero_grad) bir LightningModule'e (`training_step` + `configure_optimizers`) çevir. Trainer hangi 4 işlemi senin yerine yapar? `fast_dev_run` ne işe yarar?

```
# Duz PyTorch (boilerplate) vs Lightning (soyut) — kavramsal cevrim (statik, calistirmak gerekmez)

# --- DUZ PYTORCH: elle dongu ---
# for epoch in range(N):
#     for x, y in loader:
#         optimizer.zero_grad()      # 1) gradyani sifirla
#         out = model(x)             #    ileri gecis
#         loss = criterion(out, y)   #    kayip
#         loss.backward()            # 2) geri yayilim
#         optimizer.step()           # 3) parametre guncelle
#                                    # 4) epoch/batch dongusu = elle

# --- LIGHTNING: yalniz iki sey ---
# class Net(pl.LightningModule):
#     def training_step(self, batch, idx):   # bir batch -> KAYIP dondur (tum sistem mantigi)
#         x, y = batch
#         loss = criterion(self(x), y)
#         self.log("train_loss", loss)        # metrik + multi-GPU senkron
#         return loss
#     def configure_optimizers(self):
#         return torch.optim.Adam(self.parameters())
# trainer = pl.Trainer(fast_dev_run=True)     # tek batch hizli debug
# trainer.fit(net, loader)

print("Trainer'in OTOMATIK yaptigi 4 islem: zero_grad, backward, optimizer.step, epoch/batch dongu
print("fast_dev_run=True: tek batch ileri+geri gecis -> kodu saniyede dogrula (tam egitim oncesi d
# Egitim dongusu modelin ICINE tasinir; multi-GPU/TPU kod degismeden gelir.
```

Egzersiz 4 (Az-etiket eğrisi). SwAV-pretrain, supervised-pretrain ve rastgele backbone'u 100/316/1000 etiketle eğitip validation doğruluğunu çiz. SSL'in avantajı etiket sayısı azaldıkça neden **artar**? Hafta 10 Misra ile ilişkilendir.

```
import numpy as np

n_labels   = np.array([100, 316, 1000])
random_acc = np.array([10.0, 11.0, 12.5])    # rastgele backbone (~%10)
supervised = np.array([19.0, 26.0, 33.0])    # ImageNet supervised pre-train
ssl_swav   = np.array([40.0, 50.0, 60.0])    # SwAV SSL pre-train (~2x supervised)

oran = ssl_swav / supervised                 # SSL / supervised orani
for n, s, ss, o in zip(n_labels, supervised, ssl_swav, oran):
    print(f"{n:5d} etiket: supervised {s:.0f}%  SSL {ss:.0f}%  oran x{o:.2f}")
print("SSL avantaji en AZ etikette en buyuk:", n_labels[np.argmax(oran)], "etikette x", round(floa
```

```
# SSL etiketsiz veriden temsil ogrendigi icin az-etikette siniflandirici zaten iyi ozniteliklerle
# Etiket azaldikca supervised omurga cokerken SSL omurga ayakta kalir -> fark BUYUR.
# Hafta 10 (Misra/PIRL): "az etiketle ogren" pretext gorevlerinin TAM bu deneysel kaniti.
```

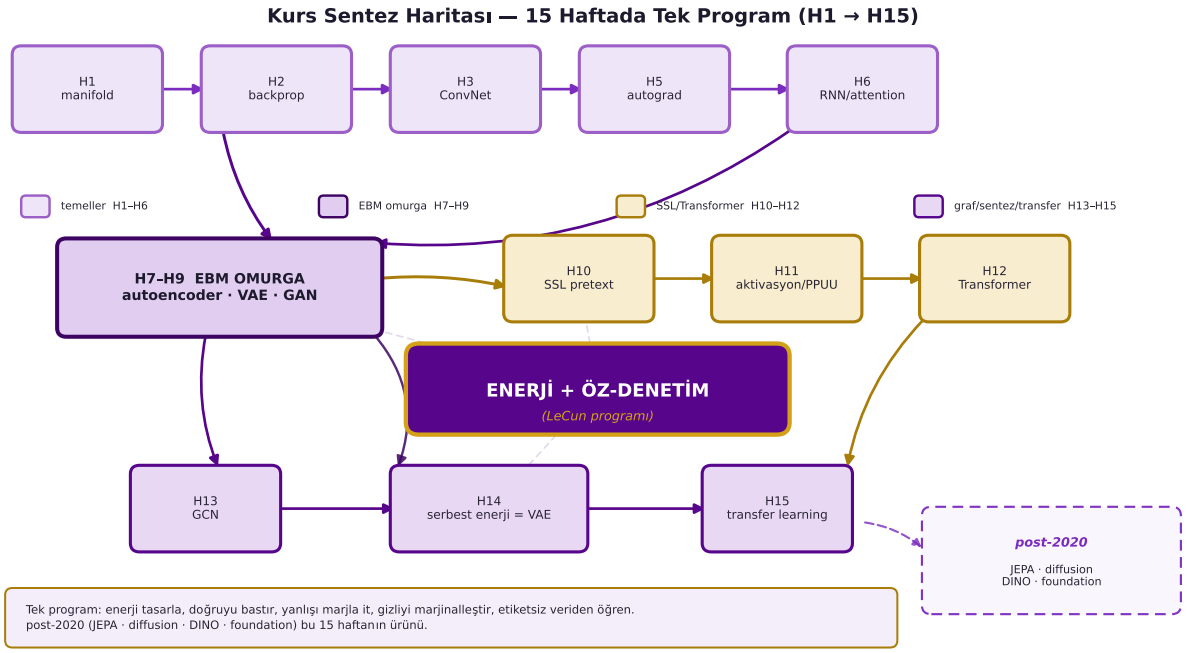
Egzersiz 5 (Kurs sentezi — KAPANIŞ). Tüm kursu tek cümlede birleştir: Hafta 1 (manifold) → Hafta 7-9 (EBM) → Hafta 10-12 (SSL/Transformer) → Hafta 13 (GCN) → Hafta 14 (serbest enerji) → Hafta 15 (transfer). LeCun'un "enerji + öz-denetim" programının bu 15 haftadaki izini sür; bugünkü foundation models bu programın neresinde durur?

```
# KURS SENTEZI (KAPANIS) — 15 haftanin tek iplikte ozeti (statik)
program = [
    ("Hafta 1",   "manifold hipotezi",          "veri dusuk-boyutlu manifoldda yasar (temsil)"),
    ("Hafta 2-5", "backprop / autograd",        "gradyanla enerji minimizasyonu"),
    ("Hafta 6",   "RNN / attention",            "dizi + odak"),
    ("Hafta 7-9", "EBM OMURGA (AE/VAE/GAN)",    "enerji tasarla: dogruyu bastir, yanlisi it"),
    ("Hafta 10",  "SSL pretext",                "etiketsiz veriden ogren"),
    ("Hafta 11",  "aktivasyon/kayip/PPUU",      "marj + belirsizlik"),
    ("Hafta 12",  "Transformer (BERT)",         "pre-train -> fine-tune"),
    ("Hafta 13",  "GCN",                        "graf uzerinde mesaj gecisi"),
    ("Hafta 14",  "serbest enerji = VAE",       "gizliyi marjinalleştir (F = <E> - T*H)"),
    ("Hafta 15",  "transfer learning",          "omurga yeniden kullan + SSL az-etiket zaferi"),
]
for h, konu, ozet in program:
    print(f"{h:9s} {konu:26s} {ozet}")
print()
print("TEK PROGRAM: bir enerji tasarla, dogruyu bastir, yanlisi marjla it, gizliyi marjinalleştir,
print("             ve ETIKETSIZ veriden ogren.")
print("Foundation models (CLIP/SAM/GPT, post-2020) = bu programin dev-olcekli, evrensel hali (KURS
```

22.10 Kurs Tamamlandı 🎓

> **!** 🎓 NYU Deep Learning (DLSP20) — 15/15 Hafta Tamamlandı
>
> **Bu bonus dersle birlikte, LeCun ve Canziani'nin tüm kursu Türkçe öğretim setine dönüştü:** manifold geometriden (Hafta 1) enerji-tabanlı modellere (Hafta 7-9), attention/Transformer'dan (Hafta 12) graph neural net'lere (Hafta 13), varyasyonel serbest enerjiden (Hafta 14) transfer learning'e (Hafta 15).
> **Kurs sonrası yapılacak:** Egzersiz 5 (kurs sentezi) ile 15 haftayı tek iplikte topla; "Enerji tasarla, doğruyu bastır, yanlışı marjla it, gizliyi marjinalleştir" cümlesini her hafta için örnekle; post-2020 köprülerini (JEPA, diffusion, DINO/MAE, foundation models) kursun hangi temeline oturduğunu yaz.

Şekil 22.7 tüm kursu tek bir haritada toplar: Hafta 1 (manifold) → Hafta 7-9 (EBM omurga) → Hafta 10-12 (SSL/Transformer) → Hafta 13-15 (graf/sentez/transfer), merkezde **enerji + öz-denetim** birleştirici çatısı (LeCun programı) ve sağ altta post-2020 (JEPA/diffusion/DINO/foundation) bu 15 haftanın ürünü olarak işaretli.



Şekil 22.7: Kurs sentez haritası — 15 haftada tek program (H1 → H15, KURSU KAPATAN FLAGSHIP). Üst sıra temeller (H1 manifold → H2 backprop → H3 ConvNet → H5 autograd → H6 RNN/attention, violet açık); orta katman EBM omurga (H7-H9 autoencoder · VAE · GAN, violet koyu) + SSL/Transformer (H10 pretext, H11 aktivasyon/PPUU, H12 Transformer, gold); alt sıra graf/sentez/transfer (H13 GCN → H14 serbest enerji = VAE → H15 transfer, violet-gold). Merkezde 'ENERJİ + ÖZ-DENETİM' (LeCun programı); sağ altta post-2020 (JEPA/diffusion/DINO/foundation, KURSTA YOK). Tek program: enerji tasarla, doğruyu bastır, yanlışı marjla it, gizliyi marjinalleştir, etiketsiz veriden öğren.

22.11 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Hoca / timestamp
Ne zaman transfer	Az veri → pre-trained; dağılım eşleşmeli	Falcon 4:58
Transfer = omurga + kafa	Önceden eğitilmiş omurga + üstüne yeni sınıflandırıcı	Falcon 5:35
Freeze (linear probe)	Omurga dondur; öznitelik çıkarıcı; lineer ayrılabilir	Falcon 16:55
Fine-tune	Omurga çöz, düşük öğrenme oranı; çok veri	Falcon 40:22
torch.no_grad	Graf tutma; hızlı + az bellek	Canziani 20:24
LightningModule	training_step + configure_optimizers	Falcon 24:53
Trainer	epoch/backward/step/zero_grad otomatik	Falcon 31:08
fast_dev_run	Tek batch hata-ayıklama	Falcon 42:20
SwAV (SSL omurga)	Etiketsiz ön-eğitim; 3000 öznitelik	Falcon 46:03
SSL kendi-veri avantajı	Etiketsiz kendi veride ön-eğit + az etiket	Canziani 48:28
Az-etiket zaferi	SSL ~2× supervised; daha hızlı	Falcon 1:08
SSL yöntemleri	MoCo/PIRL/SimCLR/BYOL/SwAV	Falcon 6:59

22.12 ML Builder Bağlantıları

Geriye köprüler:

1. **Pre-train → fine-tune** → Hafta 10 (SSL) + 12 (BERT) + 14 (transfer vs fine-tune).
2. **SSL yöntemleri (PIRL/MoCo/SimCLR)** → Hafta 8-9 (contrastive) + 10 (PIRL=Misra).
3. **ResNet / freeze / linear probe** → Hafta 3/9 (residual) + 10 (linear probe).
4. **training_step döngüsü** → Hafta 5 (autograd: forward→loss→backward→step).
5. **Az-etiket SSL avantajı** → Hafta 10 (az etiketle öğren tezi).

İleriye köprüler:

1. **SwAV/BYOL → DINO/MAE/DINOv2 (post-2020, KURSTA YOK)**.
2. **Omurga + transfer → foundation models (CLIP/SAM/GPT, post-2020)**.
3. **Lightning/Bolts** → araştırma-üretim köprüsü; reproducible SSL.
4. **Kendi-veri SSL** → endüstride etiket-tasarrufu, alan-uyarlama.

> **!** Bu dersten — ve bu kurstan — tek bir şey alıp gideceksen
>
> Transfer learning, başka (büyük) veride eğitilmiş bir omurgayı alıp kendi az verinde yeniden kullanmaktır — dağılımlar eşleştiği sürece omurgayı dondurup (linear probe) ya da ince ayar yaparak; ve PyTorch Lightning bunu boilerplate'siz, çok-GPU'lu yapar. Asıl ders şu: **öz-denetimli ön-eğitim (SwAV),**

etiketli ön-eğitimi — özellikle az-etiket rejiminde ~2 kat — geçer, çünkü kendi etiketsiz verinde eğitilebilir. Bu, kursun Hafta 10'dan beri savunduğu tezin pratik zaferidir: öz-denetim, pahalı etiketleri atlayıp verinin kendi yapısından öğrenir. Ve böylece kurs kapanır — manifold geometriden (Hafta 1) enerji-tabanlı modellere (Hafta 7-9), Transformer'dan (Hafta 12) graph net'lere (Hafta 13), serbest enerjiden (Hafta 14) transfer'e (Hafta 15): hepsi tek bir programın parçaları — **bir enerji tasarla, doğruyu bastır, yanlışı marjla it, gizliyi marjinalleştir, ve etiketsiz veriden öğren**. Bu temelin post-2020 zirvesi (DINO, MAE, diffusion, foundation models, JEPA) kursta yoktur ama bu 15 haftanın tohumlarının doğrudan ürünüdür.