BSM622 · HAFTA 10

Sinir Ağları

Neural Networks

Yapay nöronlar · Aktivasyon fonksiyonları · Backpropagation · Düzenlileştirme · Yorumlama

Kaynak: Bruce, Bruce & Gedeck (2026) "AI-Assisted Statistics for Data Scientists", 3rd ed. — Bölüm 8

Doç. Dr. Onur Polat

Hacettepe Üniversitesi · Bilişim Enstitüsü

Güz 2026-2027 · 2 saatlik lisansüstü ders · 32 slayt

Ders Akışı

7 bölüm · yaklaşık 2 saat · Python + scikit-learn + PyTorch uygulamaları

01

Sinir Ağlarının Temel Yapısı

Katmanlar, düğümler, ağırlıklar

02

Veriye Ağ Uydurmak

Ağırlıklar, bias, aktivasyon fonksiyonu kavramı

03

Aktivasyon Fonksiyonları

ReLU, Sigmoid, Softmax, tanh

04

Eğitim Süreci

Kayıp fonksiyonu, backpropagation, gradient descent

05

Aşırı Uyum ve Düzenlileştirme

Dropout, L1/L2, early stopping

06

Yorumlama ve Değişken Önemi

SHAP değerleri

07

Hiperparametre Optimizasyonu

Katman sayısı, öğrenme oranı, batch büyüklüğü

BÖLÜM 1

Sinir Ağlarının Temel Yapısı

İnsan beyni mimarisinden ilham alan tahmin modelleri

Sinir Ağı Nedir?

Esnek, güçlü, parametrik tahmin modeli

Tanım

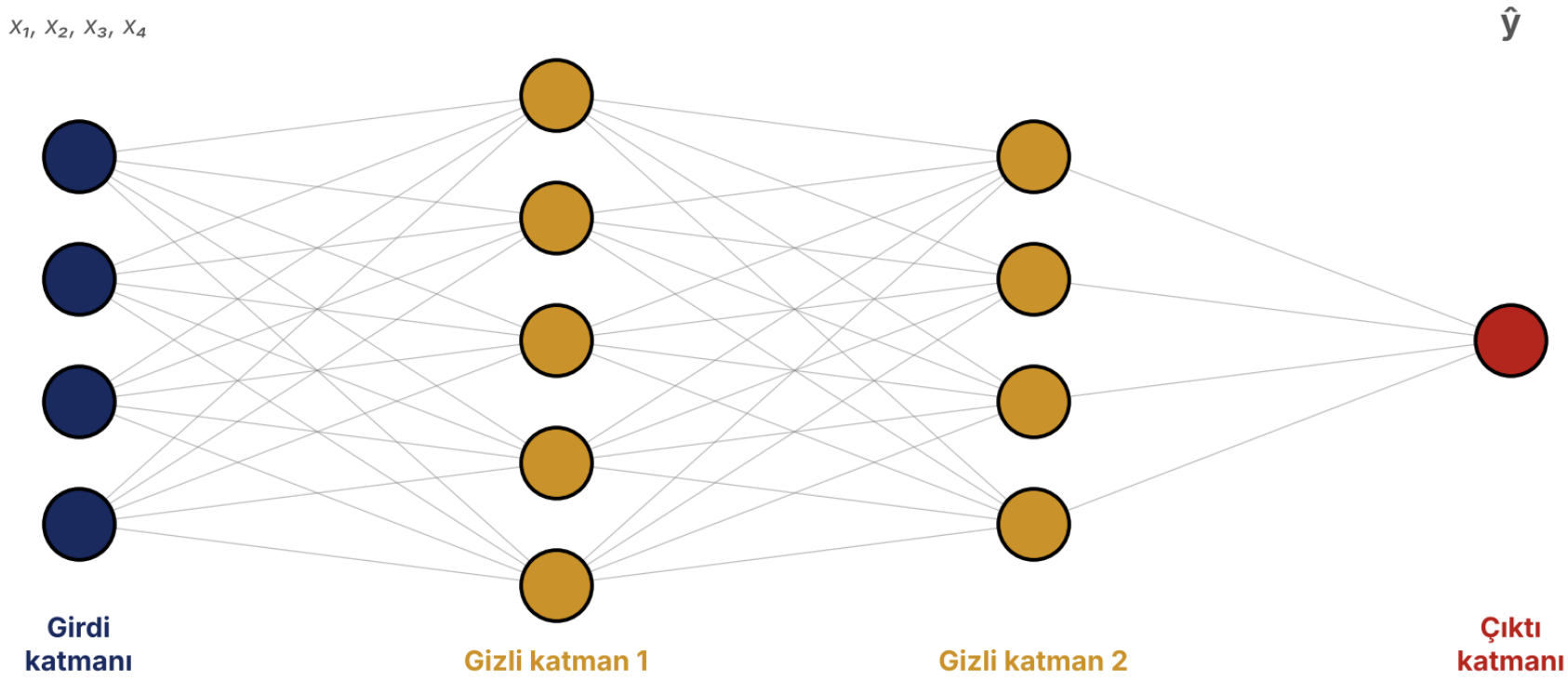
Sinir ağları, girdiyi birbirine bağlı düğümler ve ara hesaplamalardan oluşan karmaşık bir yol üzerinden çıktıya dönüştüren esnek, parametrik modellerdir. İlham kaynağı insan beyninin basitleştirilmiş bir modelidir, ancak modern yapay sinir ağları beynin biyolojik karmaşıklığına kıyasla hesaplama verimliliği için optimize edilmiş çok daha sade yapılardır.

Neden önemli?

- Verideki **çok karmaşık ilişkileri** (lineer olmayan, etkileşimli, hiyerarşik) yakalayabilir
- Günümüz yapay zekâ devriminin merkezinde yer alır — metin, görüntü, ses sınıflandırmadan üretken AI'a
- Derin öğrenme modelleri (LLM, CNN, Transformer) bu bölümdeki temel yapıya dayanır
- Lineer regresyon gibi birkaç parametre yerine **milyonlarca ya da milyarlarca** parametre öğrenir
- Karşılığında: yorumlanabilirlik azalır, veri ve hesaplama ihtiyacı artar

Basit Bir İleri-Beslemeli Sinir Ağı

Multilayer feedforward neural network



- **Girdi katmanı:** özellik sayısı kadar düğüm (p adet)
- **Gizli katmanlar:** katman sayısı ve her katmandaki düğüm sayısı HİPERPARAMETREDİR (kullanıcı belirler)
- **Çıktı katmanı:** regresyon için 1 düğüm, çok sınıflı sınıflandırma için K düğüm

Diğer Yöntemlerle Karşılaştırma

Sinir ağları esneklik–yorumlanabilirlik takasında nerede?

Yöntem	Esneklik	Yorumlanabilirlik	En iyi kullanım
Lineer regresyon	Düşük	Çok yüksek	Lineer ilişkiler, küçük veri
Spline / GAM	Orta	Yüksek	Lineer olmayan tek-değişken etkisi
Karar ağacı	Orta-Yüksek	Yüksek (kurallar)	Etkileşim, yorumlanabilir çıktı
Random Forest / XGBoost	Yüksek	Düşük	Yapılandırılmış veride en iyi performans
Sinir Ağı	Çok yüksek	Çok düşük	Görüntü, metin, ses, çok büyük veri

Yapılandırılmış veride (tablolu) gradient boosting genelde sinir ağından daha iyidir. Sinir ağının gerçek gücü GÖRÜNTÜ, METİN, SES gibi yüksek boyutlu verilerde ortaya çıkar.

BÖLÜM 2

Veriye Ağ Uydurmak

Ağırlıklar, bias ve düğüm hesabı

Tek Bir Düğümün Hesabı

Her düğüm bir mini-regresyon + aktivasyon fonksiyonudur

Ağırlıklı toplam (weighted sum)

$$z = b + \sum_{i=1}^{p} w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

Aktivasyon uygulandıktan sonra düğüm çıktısı

$$a = f(z) = f(b + \mathbf{w}^\top \mathbf{x})$$

Lineer regresyon ile benzerlik

- Ağırlıklar w_i ↔ regresyon katsayıları β_i
- Bias b ↔ sabit terim (intercept) α — düğümün çıktısını girdiden bağımsız olarak kaydırır
- Fark: aktivasyon fonksiyonu f sayesinde çıktı NONLİNEER — asıl güç buradan gelir

Katman Düzeyinde Vektörel Gösterim

Her katman bir matris çarpımı + aktivasyon

ℓ-inci katmanın aktivasyonu

$$\mathbf{a}^{(\ell)} = f\left(\mathbf{W}^{(\ell)}\,\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)$$

Burada:

- **a^(ℓ-1) :** önceki katmanın çıktı vektörü (bir sonraki katmana girdi olur)
- **W^(ℓ) :** ℓ-inci katmanın ağırlık matrisi (önceki katman × bu katman düğüm sayısı)
- **b^(ℓ) :** bu katmanın bias vektörü
- **f :** eleman-bazlı aktivasyon fonksiyonu (ReLU, sigmoid, tanh...)

⟹ Veri ileriye doğru katmandan katmana akar — **forward propagation**

BÖLÜM 3

Aktivasyon Fonksiyonları

ReLU, Sigmoid, Softmax, tanh

Neden Lineer Olmayan Aktivasyon?

Lineer bileşke hâlâ lineerdir

Kritik gözlem

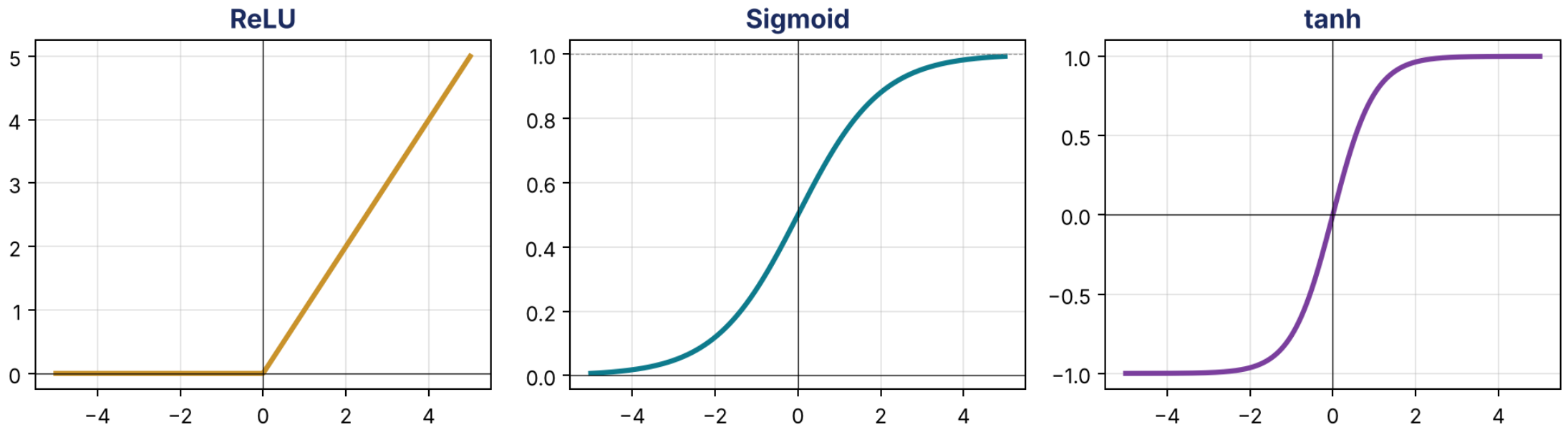
Eğer bütün aktivasyon fonksiyonları lineer olsaydı, 100 katmanlı bir sinir ağı tek bir lineer dönüşüme indirgenebilirdi — yani gizli katmanların hiçbir avantajı kalmazdı. Model yine basit bir lineer regresyona eşdeğer olurdu.

Lineer olmayan aktivasyon fonksiyonları sayesinde

- Her katman önceki katmanın gösteriminden **yeni bir özellik uzayı** (feature representation) oluşturur
- Yeterince geniş tek-katmanlı bir sinir ağı, **evrensel yaklaşıklayıcıdır** (Universal Approximation Theorem — Cybenko 1989, Hornik 1991)
- Birden çok katman, aynı fonksiyonu **çok daha az nöronla** temsil eder → "derin" öğrenmenin matematiksel dayanağı

Üç Temel Aktivasyon Fonksiyonu

Grafik karşılaştırması



Aktivasyon seçim rehberi

- **Gizli katmanlar için varsayılan: ReLU** — hızlı, kaybolan-gradient problemine karşı dirençli
- **İkili sınıflandırma çıktısı: Sigmoid** — (0, 1) aralığında bir olasılık verir
- **Çok sınıflı çıktı: Softmax** (sonraki slaytta)
- **tanh:** (−1, 1) aralığında sıfır-merkezli; eskiden yaygın, bugün sadece bazı RNN uygulamalarında

ReLU — Rectified Linear Unit

Günümüzün en yaygın gizli-katman aktivasyonu

$$\mathrm{ReLU}(z) = \max(0,\ z)$$

Neden ReLU bu kadar başarılı?

- **Hesaplama ucuzdur** — sadece bir karşılaştırma işlemi; sigmoid/tanh'taki üstel fonksiyonlardan 5–10 kat hızlı
- **Türevi basittir:** z > 0 için 1, z ≤ 0 için 0 → backpropagation hızlı
- **Kaybolan gradient (vanishing gradient) problemine daha az maruz kalır** — sigmoidde gradient uç değerlerde neredeyse 0 olurken ReLU'da aktif bölgede sabit 1
- **Seyrek aktivasyon** — negatif girdiler sıfırlanır, bu da ağa "sparsity" (seyreklik) kazandırır

⚠ *"Dying ReLU" problemi:* çok sayıda nöron sürekli negatif girdi alır ve sıfır üretmeye başlarsa artık öğrenemez. Çözüm: Leaky ReLU, ELU, GELU gibi varyantlar.

Sigmoid Aktivasyon

İkili sınıflandırma çıktısı için standart

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \sigma(z) \in (0, 1)$$

Özellikleri

- Girdiyi (0, 1) aralığına sıkıştırır → çıktı doğrudan bir **olasılık olarak yorumlanabilir**
- Lojistik regresyonun aynısı — tek düğümlü bir sinir ağı = lojistik regresyon
- Türevi: σ'(z) = σ(z) · (1 − σ(z)), maksimum değeri 0.25 (z=0 noktasında)
- z çok küçük ya da çok büyükken türev ≈ 0 → **kaybolan gradient problemi**

Nerede kullanılır?

- **İkili sınıflandırmanın çıkış katmanında** (son nöron)
- Gizli katmanlarda artık çoğunlukla kullanılmaz (ReLU tercih edilir)

Softmax Aktivasyon

Çok sınıflı sınıflandırma için olasılık dağılımı

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \qquad \sum_i \text{softmax}(z_i) = 1$$

Nasıl çalışır?

- K tane sınıf için K adet skor (logit) alır ve hepsini birden **[0, 1] aralığında olasılıklara** dönüştürür
- Çıktıların toplamı tam olarak 1 → geçerli bir olasılık dağılımı
- `Örnek: 3-sınıf problemde z = (2.0, 1.0, 0.1) → softmax ≈ (0.659, 0.242, 0.099)`
- Üstel fonksiyon farkları *abartır* — en büyük skor daha da öne çıkar
- Softmax + **cross-entropy loss** kombinasyonu çok sınıflı sınıflandırmanın standart çıkış tasarımıdır
- Sayısal kararlılık: `softmax(z - max(z))` olarak hesaplanır — overflow'u önler

Çıktı Katmanı Tasarımı

Göreve göre doğru çıktı + aktivasyon + kayıp fonksiyonu seçimi

Görev	Çıktı nöron sayısı	Aktivasyon	Kayıp fonksiyonu
Regresyon	1	Kimlik (none)	MSE / MAE
İkili sınıflandırma	1	Sigmoid	Binary Cross-Entropy
Çok sınıflı sınıflandırma (K sınıf)	K	Softmax	Categorical Cross-Entropy
Çok etiketli sınıflandırma	K	Sigmoid (her biri ayrı)	K adet BCE toplamı
Zaman serisi tahmini (h-adım)	h	Kimlik	MSE

BÖLÜM 4

Eğitim Süreci

Kayıp fonksiyonu, backpropagation, gradient descent

Kayıp Fonksiyonları

Tahminin gerçekten ne kadar uzak olduğunu ölçer

Regresyon: Mean Squared Error (MSE)

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

İkili sınıflandırma: Binary Cross-Entropy (BCE)

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$$

Çok sınıflı sınıflandırma: Categorical Cross-Entropy (CCE)

$$\mathcal{L}_{\text{CCE}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \log \hat{y}_{i,k}$$

⟹ Eğitim = kayıp fonksiyonunu minimize edecek ağırlık ve bias değerlerini bulmak.

Gradient Descent

Kayıp yüzeyinde yokuş-aşağı yürüyüş

Her ağırlık için güncelleme kuralı

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$$

Fikir

- **∂ℒ/∂w :** kayıbın w değişkenine göre türevi — kaybı en çok ARTIRAN yön
- Bu nedenle o yönün **tersine** (eksi işareti) bir adım atılır — kayıp azalır
- **η (öğrenme oranı, learning rate):** her adımın büyüklüğü — en önemli hiperparametre
  – η çok büyük → kayıp etrafta zıplar, yakınsama olmaz
  – η çok küçük → eğitim çok yavaş ilerler

Pratikte **Adam, RMSProp, SGD + momentum** gibi uyarlanabilir optimizatörler η'yi otomatik ayarlar.

Backpropagation

Kayıbın gradyanını tüm ağa yaymak — zincir kuralı

Zincir kuralı (chain rule)

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

Algoritma (Rumelhart, Hinton & Williams, 1986)

1. **Forward pass:** Girdi ağ boyunca ileri akar, her katmanın çıktısı saklanır
2. **Kayıp hesaplanır:** $\mathcal{L}(\hat{y},\ y)$
3. **Backward pass:** Zincir kuralı ile gradyan son katmandan ilk katmana doğru GERİYE yayılır
4. **Ağırlık güncellemesi:** her w için gradient descent uygulanır

💡 Modern çerçeveler (PyTorch, TensorFlow) backprop'u **automatic differentiation (autograd)** ile otomatik yapar — sizin elle hesaplamanız gerekmez.

Mini-Batch Stochastic Gradient Descent

Hesaplama ve gürültü arasında denge

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}(x^{(t)}, y^{(t)})$$

Üç gradient descent varyantı

- **Batch GD:** her adımda TÜM eğitim verisini kullanır — kararlı ama ağır ve yavaş
- **Stochastic GD (SGD):** her adımda TEK bir gözlem kullanır — hızlı ama çok gürültülü
- **Mini-batch GD:** 32, 64, 128 vs. küçük parti üzerinde gradyan hesaplanır — ***standart tercih***

Önemli terimler:

- **`Batch size:`** bir güncellemede kullanılan gözlem sayısı
- **`Epoch:`** tüm eğitim veri setinin bir kez geçilmesi
- **`Iteration:`** tek bir mini-batch üzerinden tek bir ağırlık güncellemesi

BÖLÜM 5

Aşırı Uyum ve Düzenlileştirme

Overfitting & Regularization

Sinir Ağları Neden Aşırı Uyum Eğilimli?

Milyonlarca parametre, kısıtlı veri

⚠ Aşırı uyum (overfitting)

Bir sinir ağı yeterince büyükse eğitim verisini EZBERLEYEBİLİR — hatta rastgele gürültüyü bile. Eğitim kaybı sıfıra yaklaşırken doğrulama (validation) kaybı artmaya başlar. Model, verideki gerçek örüntüyü değil, o özel eğitim örneklerinin tesadüfi özelliklerini öğrenir.

Üç ana düzenlileştirme (regularization) yöntemi

1. **Weight decay (L2 cezası)** — büyük ağırlıklara ceza verir
2. **Dropout** — her eğitim adımında nöronların bir kısmını rastgele devre dışı bırak
3. **Early stopping** — doğrulama kaybı artmaya başladığında eğitimi durdur

Ek yöntemler: batch normalization, veri artırma (augmentation), transfer learning, model ensembling.

L1 ve L2 Düzenlileştirme

Ağırlık cezası ekleyerek modeli basitleştirmek

L2 cezası (ridge / weight decay)

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_j w_j^2$$

L1 cezası (lasso)

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_j |w_j|$$

L1 vs L2 — pratik fark

- **L2:** tüm ağırlıkları eşit ölçüde SIFIRA DOĞRU çeker — sıfır yapmaz; *yoğun (dense) çözüm*
- **L1:** bazı ağırlıkları TAM OLARAK SIFIR yapar → otomatik özellik seçimi; *seyrek (sparse) çözüm*
- **λ:** ceza şiddeti — tipik değerler 10^{-5} ile 10^{-2} arası, cross-validation ile seçilir
- PyTorch / Keras'ta L2 genelde optimizatörün `weight_decay` parametresiyle açılır
- Elastic Net: L1 + L2 kombinasyonu — her iki dünyadan da yararlanır

Dropout

Nöronları rastgele devre dışı bırakarak ensemble etkisi

$$\tilde{a}_j = \frac{m_j}{1-p} a_j, \qquad m_j \sim \text{Bernoulli}(1-p)$$

Srivastava et al. (2014) — Dropout algoritması

- **Eğitim sırasında** her nöron, her iterasyonda p olasılıkla (tipik 0.2–0.5) rastgele sıfırlanır
- Kalan nöronlar `1/(1-p)` ile ölçeklenir ki aktivasyonların beklenen değeri korunsun (inverted dropout)
- **Çıkarım (inference) sırasında** dropout kapatılır — tüm nöronlar aktif
- Her mini-batch için farklı bir "alt ağ" eğitiliyor → ***implicit ensemble*** etkisi
- Nöronlar birbirlerine aşırı güvenmekten vazgeçer — "co-adaptation" azalır
- Modern CNN ve Transformer'larda dropout batch normalization ile yer değiştirdi

Early Stopping

En basit ve en etkili düzenlileştirme

Algoritma

1. Veriyi eğitim / doğrulama / test olmak üzere üçe ayır (tipik 70/15/15)
2. Her epoch sonunda **doğrulama kaybını** hesapla
3. Doğrulama kaybı N epoch (patience) boyunca azalmıyorsa eğitimi DURDUR
4. En iyi doğrulama kaybını veren ağırlıkları GERİ YÜKLE (restore best weights)

Neden işe yarıyor?

- Eğitim ilerledikçe model önce kaba ilişkileri (sinyal), sonra gürültüyü öğrenir
- Doğrulama kaybı U-şekli çizer: önce azalır, sonra artmaya başlar — **o nokta tam da aşırı uyumun başladığı andır**
- Ek hiperparametre gerektirmez; L2/dropout ile birlikte kullanılabilir
- Keras: `EarlyStopping(patience=5, restore_best_weights=True)`

BÖLÜM 6

Yorumlama ve Değişken Önemi

Kara-kutu modelden içgörü çıkarmak — SHAP

SHAP — SHapley Additive exPlanations

Oyun teorisinden gelen bir atfetme yöntemi

Bir özelliğin SHAP değeri

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\ (|F| - |S| - 1)!}{|F|!} \left[v(S \cup \{i\}) - v(S) \right]$$

Fikir (Lundberg & Lee, 2017)

- Her özelliği bir **"oyuncu"** olarak düşün; tahminin değeri bir **"koalisyonun ödülü"** (Shapley 1953)
- Shapley değeri = her özelliğin, olası tüm koalisyonlar üzerinden ortalama katkısı
- Tahmin = ortalama tahmin + her özelliğin SHAP katkılarının toplamı (additive)
- **Yerel yorumlama (local):** tek bir gözlem için her özelliğin katkısı
- **Küresel yorumlama (global):** |SHAP| ortalaması → genel değişken önemi
- Python: `shap.DeepExplainer(model, X).shap_values(X_test)`

BÖLÜM 7

Hiperparametre Optimizasyonu

Katman sayısı, öğrenme oranı, batch büyüklüğü

Ayarlanması Gereken Hiperparametreler

Modelin başarısını büyük ölçüde belirleyen seçimler

Hiperparametre	Tipik aralık	Nasıl ayarlanır?
Katman sayısı (derinlik)	`2 – 20 (tablular)`	Küçükten başla, overfitting görene kadar artır
Katman genişliği (nöron sayısı)	`32 – 512`	Her katman öncekinin yaklaşık yarısı
Öğrenme oranı η	`1e-5 – 1e-2`	Log-uniform grid veya öğrenme-oranı range testi
Batch size	`32, 64, 128, 256`	GPU belleğine göre; genelde 2'nin kuvveti
Dropout oranı p	`0.1 – 0.5`	Overfitting varsa artır
L2 λ (weight decay)	`1e-5 – 1e-2`	Log-uniform grid
Optimizer	`Adam / AdamW`	Varsayılan: Adam, lr=1e-3
Epoch sayısı	`Early stopping`	Patience = 5–10

Ana Çıkarımlar ve Haftalık Çalışma

Hatırlamanız ve uygulamanız gerekenler

🧠 Hatırlanması gereken kavramlar

1. Sinir ağı = **ağırlıklı toplam + bias + aktivasyon** işleminin katmanlar boyunca tekrarı
2. Aktivasyon fonksiyonu nonlineerliği sağlar — **ReLU gizli katmanda, Sigmoid/Softmax çıkışta**
3. Eğitim = kayıp fonksiyonunu **mini-batch gradient descent + backpropagation** ile minimize etmek
4. Aşırı uyum kaçınılmaz — **L2, dropout, early stopping** birlikte kullanılır
5. Yorumlama için **SHAP değerleri** yerel ve küresel önem ölçümü sağlar
6. En önemli hiperparametre: **öğrenme oranı η;** diğerleri sonra gelir

💻 Colab pratiği

- scikit-learn `MLPClassifier` ile Diabetes veri setinde ikili sınıflandırma
- PyTorch ile elden yazılmış 2-katmanlı ağ — forward + backward adımları açıkça
- California Housing veri setinde regresyon — MSE loss + Adam optimizer
- Dropout, L2, early stopping karşılaştırması — eğitim/doğrulama eğrileri
- SHAP ile değişken önem analizi — summary plot

Teşekkürler

Sorular?

onurpolat@hacettepe.edu.tr