HAFTA 1 • PAZAR

Keras ile Derin Öğrenme

Sequential API → BatchNorm & Dropout → Callbacks & LR Scheduling → Functional API → Custom Loop

09:00 — 17:00 · 7 Saat · 4 Python Dosyası · 600+ Satır Kod

Günün Planı — Hafta 1 Pazar

09:00–09:30

Dün Özeti & Derin Öğrenmeye Giriş

09:30–10:30

Keras Sequential API — Derinlemesine

10:30–10:45

Kısa Mola

10:45–11:45

BatchNorm, Dropout Stratejileri

11:45–13:00

Callbacks & LR Scheduling Lab

13:00–14:00

Öğle Arası

14:00–15:15

Functional API & Çoklu Giriş/Çıkış

15:15–16:30

Custom Training Loop (GradientTape)

16:30–17:00

Mini Proje Sunumu & Kapanış

Gün Sonu: Çalışan Keras modeli + 5 farklı Callback + LR Scheduler + Functional API projesi

Yapay Sinir Ağı — Temel Bileşenler

Katmanlar · Aktivasyon · İleri Besleme · Geri Yayılım

⚡ Aktivasyon Fonksiyonu

Doğrusal olmayanlık katar. ReLU, Sigmoid, Softmax, ELU…

Kayıp Fonksiyonu (Loss)

Tahmin ile gerçek arasındaki farkı ölçer. MSE, CrossEntropy…

Optimizer

Ağırlıkları hangi yönde ne kadar güncelleyeceğini belirler. Adam, SGD…

Geri Yayılım (Backprop)

Hatayı zincir kuralıyla katmanlara dağıtır, gradyan hesaplar.

İleri besleme: Girdi → Ağırlıklar → Aktivasyon → Çıkış | Geri yayılım: Gradyan ile ağırlıkları güncelle

Keras Sequential API — Derinlemesine

Dense · Aktivasyon · Initializer · Regularizer

keras_sequential.py

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers

model = keras.Sequential([
    # Giriş katmanı — shape=özellik sayısı
    layers.Input(shape=(30,)),

    # Gizli katman 1: 256 nöron
    layers.Dense(256,
        activation='relu',
        kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(1e-4)),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    # Gizli katman 2: 128 nöron
    layers.Dense(128, activation='relu',
        kernel_initializer='he_normal'),
    layers.BatchNormalization(),
    layers.Dropout(0.25),

    # Çıkış: binary → sigmoid
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', 'AUC'])
```

`kernel_initializer='he_normal'`

ReLU için önerilen. Varyansı katman boyutuna göre ölçekler, gradyan patlama/sönmesini önler.

`kernel_regularizer=l2(1e-4)`

L2 regularization: büyük ağırlıkları cezalandırır, overfitting'i azaltır.

`BatchNormalization()`

Katman çıktılarını normalize eder. Eğitimi hızlandırır, daha büyük LR kullanmayı sağlar.

`Dropout(0.3)`

Eğitimde rastgele %30 nöronu kapatır. Test'te tüm nöronlar aktif (ama ağırlıklar 0.7 ile ölçeklenir).

Aktivasyon Fonksiyonları — Karşılaştırmalı Analiz

Neden Lineer Aktivasyon Yetmez?

ReLU

$\max(0, x)$

✓ Hızlı, basit. Derin ağlarda standart.

⚠ x<0 için gradyan=0 (dying ReLU)

📌 **Çoğu gizli katman**

Leaky ReLU

$x>0 \rightarrow x,\ x<0 \rightarrow 0.01x$

✓ Dying ReLU sorununu çözer.

⚠ α hiperparametresi seçimi

📌 **ReLU yetersiz kalınca**

ELU

$x>0 \rightarrow x,\ x\leq 0 \rightarrow \alpha(e^{x}-1)$

✓ Negatif değerler; sıfır ortalama.

⚠ Hesaplama maliyeti yüksek

📌 **Derin ağlarda iyi**

Sigmoid

$1 / (1 + e^{-x})$

✓ [0,1] çıktı. Binary sınıflandırma.

⚠ Vanishing gradient. Yavaş.

📌 **Sadece çıkış katmanı**

Softmax

$e^{x}_{i} / \Sigma e^{x}_{j}$

✓ Çıktılar toplamı=1. Olasılık.

⚠ Çok sınıflıda sadece çıkış

📌 **Çok sınıflı çıkış**

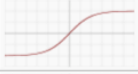
SELU

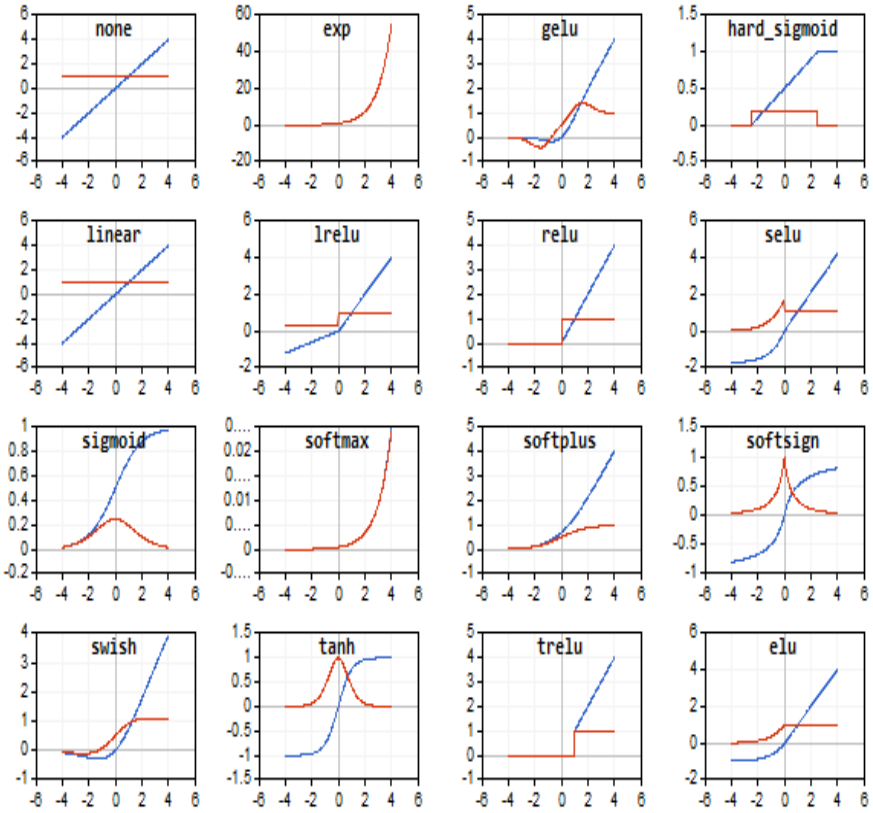
$\lambda \cdot ELU(x,\alpha)$

✓ Self-normalizing! BN gereksiz.

⚠ Özel init gerektirir.

📌 **Dense ağlarda alternatif**

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$



Batch Normalization — Neden Bu Kadar Güçlü?

Internal Covariate Shift · Hızlı Eğitim · Regularization Etkisi

⚠ Sorun: Internal Covariate Shift

Her katmanın girişi, önceki katmanın ağırlıkları değiştikçe farklılaşır. Model eğitim sırasında sürekli değişen bir dağılımla çalışmak zorunda kalır → yavaş öğrenme, düşük LR zorunluluğu.

✓ Çözüm: BatchNorm

Her mini-batch için aktivasyonları normalize eder. Öğrenilebilir γ (ölçek) ve β (kaydırma) ile orijinal kapasiteyi korur.

$$\hat{x} = (x - \mu_B) / \sqrt{(\sigma^2_B + \varepsilon)} \quad \rightarrow \quad y = \gamma \cdot \hat{x} + \beta$$

10x Daha Hızlı Eğitim

Daha büyük öğrenme hızı (LR) kullanmayı mümkün kılar.

Daha Yüksek Doğruluk

Her katmanda stabil dağılım; gradyanlar daha sağlıklı akar.

Hafif Regularization

Batch gürültüsü hafif düzenleme etkisi yaratır.

⚙ Daha Az Hassas Init

Başlangıç ağırlık seçimi kritikliği azalır.

BatchNorm konumu: Dense(…) → BatchNorm() → Activation() VEYA Dense(activation=relu) → BatchNorm() — ikisi de yaygın!

Dropout Stratejileri — 4 Farklı Yaklaşım

Regularization · Co-adaptation Önleme · Monte Carlo Dropout

Standart Dropout

`layers.Dropout(rate=0.3)`

🔧 Eğitimde her ileri geçişte nöronların %rate'ini rastgele kapatır.

📌 **Gizli Dense katmanlardan sonra. rate: 0.2–0.5**

SpatialDropout1D

`layers.SpatialDropout1D(0.2)`

🔧 Tam özellik haritası kanallarını kapatır. Ardışık örnekler arasında korelasyon varsa daha etkili.

📌 **LSTM/CNN çıkışlarından sonra. Metin verisi için ideal.**

Alpha Dropout

`layers.AlphaDropout(rate=0.1)`

🔧 SELU aktivasyonu ile kullanılmak üzere tasarlandı. Self-normalizing özelliğini korur.

📌 **SELU aktivasyonlu katmanlar için özel tercih.**

MC Dropout (Inference)

`model(x, training=True)`

🔧 Test sırasında Dropout açık bırakılır. Birden fazla tahmin yapılır → belirsizlik kestirimi.

📌 **Bayesian yaklaşım. Güven aralığı hesaplamak için.**

⚠️ Dropout sadece eğitimde (training=True) aktif! Değerlendirmede (model.evaluate) otomatik kapanır.

Callbacks — Eğitimi Akıllıca Yönet

EarlyStopping · ModelCheckpoint · ReduceLROnPlateau · Custom

EarlyStopping

```
monitor='val_loss'
patience=15
restore_best_weights=True
min_delta=1e-4
```

val_loss 15 epoch boyunca iyileşmezse eğitimi durdur. En iyi ağırlıkları geri yükle. validasyon metriği iyileşmediğinde eğitimi durduran callback

ModelCheckpoint

```
filepath='best_model.keras'
monitor='val_auc'
save_best_only=True
mode='max'
```

val_auc maksimum olduğunda modeli kaydet. Her epoch'ta overwrite eder. en iyi modeli diske kaydeden callback

ReduceLROnPlateau

```
monitor='val_loss'
factor=0.3
patience=7
min_lr=1e-7
```

val_loss iyileşmeyince LR'yi 0.3 ile çarp. Plato noktasında öğrenmeyi sürdürür. learning rate düşürme mekanizması

Custom Callback

```
on_epoch_end(epoch, logs)
on_batch_end(batch, logs)
on_train_begin(logs)
self.model.stop_training
```

BaseCallback'den miras al. İstediğin olayda istediğin kodu çalıştır.

```
callbacks=[early_stop, checkpoint, reduce_lr, custom_cb]  →  model.fit(…, callbacks=callbacks)
```

Learning Rate Scheduling — LR'yi Dinamik Yönet

Sabit LR Neden Yetmez? · 4 Popüler Strateji

Sabit LR Problemi

Büyük LR → hızlı öğrenir ama minimum'u atlar
Küçük LR → yavaş, platoda takılır
Çözüm: büyük başla, küçült!

LR Range Test (LR Finder)

LR'yi 1e-7'den 1'e logaritmik artır. Loss düşmeye başladığı → minimum LR, kayıp artmadan önce → maksimum LR.

Cosine Annealing

```
CosineDecay(initial=1e-3, decay_steps=1000)
```

LR'yi cosinus eğrisiyle düşürür. Periyodik restartlarla yerel minimumlardan kaçar.

Warmup + Decay

```
WarmupCosineDecay(warmup_steps=200)
```

Başta küçük LR → lineer ısıtma → cosine decay. Transformerlar için standart.

Exponential Decay

```
ExponentialDecay(rate=0.96, steps=1000)
```

Her 1000 adımda LR'yi 0.96 ile çarp. Basit, öngörülebilir, etkili.

Cyclical LR

```
CyclicalLR(base=1e-4, max=1e-2)
```

LR'yi periyodik olarak min-max arasında döngüye sokar. Super-convergence sağlar.

Keras Functional API — Sequential'ın Ötesine

Çoklu Giriş · Dal Yapısı · Artık Bağlantılar (Skip Connections)

Sequential — Sınırlı	Functional — Tam Kontrol
✗ Tek girdi	✓ Çoklu girdi
✗ Tek çıktı	✓ Çoklu çıktı
✗ Lineer katman sırası	✓ İstediğin topoloji
✗ Dal yapısı YOK	✓ Dal & birleştirme
✗ Artık bağlantı YOK	✓ Skip connections

Functional API (directed acyclic graph mantığında model kurma yöntemi) çoklu giriş ve çıkışlara izin verir. Skip connection (katman çıktısını ileri katmanlara doğrudan bağlama yöntemi) residual learning (artık öğrenme yaklaşımı) sağlar ve gradyan akışını iyileştirir.

functional_api.py

```
# Girdi katmanları
inp_num = keras.Input(shape=(20,))
inp_cat = keras.Input(shape=(10,))

# Sayısal dal
x1 = layers.Dense(64,'relu')(inp_num)
x1 = layers.BatchNormalization()(x1)
x1 = layers.Dropout(0.3)(x1)

# Kategorik dal
x2 = layers.Dense(32,'relu')(inp_cat)

# Birleştir (Concatenate)
merged = layers.Concatenate()([x1,x2])
x = layers.Dense(64,'relu')(merged)

# Çok çıktı
out_main  = layers.Dense(1,'sigmoid',
name='main')(x)
out_extra = layers.Dense(3,'softmax',
name='aux')(x)
model = keras.Model(
    inputs=[inp_num, inp_cat],
    outputs=[out_main, out_extra])
```

Custom Training Loop — tf.GradientTape

Maksimum Kontrol · Özel Metrik · Araştırma Düzeyinde Esneklik

custom_training_loop.py

```
optimizer = keras.optimizers.Adam(lr=1e-3)
loss_fn   = keras.losses.BinaryCrossentropy()
train_acc = keras.metrics.BinaryAccuracy()
val_acc   = keras.metrics.BinaryAccuracy()

for epoch in range(EPOCHS):
    # ── EĞİTİM DÖNGÜSÜ ──────────────────────
    for X_batch, y_batch in train_dataset:
        with tf.GradientTape() as tape:
            y_pred = model(X_batch, training=True)
            loss   = loss_fn(y_batch, y_pred)
            loss  += sum(model.losses)  # L2 reg

        # Gradyanları hesapla ve uygula
        grads = tape.gradient(loss,
                    model.trainable_variables)
        optimizer.apply_gradients(
            zip(grads, model.trainable_variables))
        train_acc.update_state(y_batch, y_pred)

    # ── VALİDASYON ──────────────────────────
    for X_v, y_v in val_dataset:
        y_vp = model(X_v, training=False)
        val_acc.update_state(y_v, y_vp)
```

tf.GradientTape()

Bu blok içindeki tüm işlemleri kaydeder. tape.gradient() ile gradyanları hesaplar.

apply_gradients()

zip(gradyanlar, değişkenler) ile her ağırlığı günceller. optimizer.step() benzeri.

training=True/False

Dropout ve BatchNorm eğitim/test modunu bu parametre ile anlar.

Metrics

update_state() ile biriktir, result() ile oku, reset_state() ile sıfırla.

✓ model.fit() → hızlı prototipleme | Custom Loop → araştırma, GAN, meta-learning, özel kayıp fonksiyonları

Bugünkü Uygulamalar — 4 Python Dosyası

Sırasıyla çalıştır · Her dosya bağımsız · Ortak veri seti: MNIST + Breast Cancer

01 Keras Sequential — Derinlemesine

`uygulama_01_keras_sequential.py`

- Dense, BatchNorm, Dropout tam mimari
- Aktivasyon fonksiyonları karşılaştırması
- He / Glorot initializer deneyi
- L1, L2, ElasticNet regularization
- model.summary() ve param sayma

02 Callbacks & LR Scheduling

`uygulama_02_callbacks_lr.py`

- EarlyStopping + restore_best_weights
- ModelCheckpoint .keras formatı
- ReduceLROnPlateau uygulaması
- Custom Callback (LRLogger + Plotter)
- Cosine Annealing + Warmup scheduler

03 Functional API & Çoklu Çıkış

`uygulama_03_functional_api.py`

- Çoklu girdi mimarisi
- Concatenate / Add birleştirme
- Skip connection (residual block)
- Çoklu çıkış + özel loss ağırlıkları
- Auxiliary loss ile eğitim

04 Custom Training Loop

`uygulama_04_custom_loop.py`

- tf.GradientTape ile tam kontrol
- Gradient clipping uygulaması
- Mixed precision training
- Custom metrik ve loss fonksiyonu
- Mini proje: Tüm teknikleri birleştir

```
pip install tensorflow scikit-learn pandas numpy matplotlib seaborn
```

Hafta 1 Tamamlandı!

Bu haftanın kazanımları:

- ✓ Scikit-Learn Pipeline + ColumnTransformer + SHAP
- ✓ EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
- ✓ Functional API — Çoklu Giriş/Çıkış, Skip Connections
- ✓ Keras Sequential API — BatchNorm + Dropout
- ✓ Learning Rate Scheduling (Cosine, Warmup, Cyclical)
- ✓ Custom Training Loop (tf.GradientTape)

Hafta 2: Data Augmentation · CNN · CIFAR-10 · Transfer Learning