

HAFTA 2 • PAZAR

RNN, LSTM & GRU

Dikkat Mekanizması

Vanishing Gradient → LSTM Kapıları → GRU → BiLSTM → Dikkat → Zaman Serisi

09:00 — 17:00 • 7 Saat • 4 Python Dosyası • 750+ Satır Kod • IMDB + Hisse Senedi Tahmini

Günün Planı — Hafta 2 Pazar

09:00–09:30



Geçen Hafta Özeti & Sıralı
Veriye Giriş

09:30–10:45



Vanilla RNN, LSTM Kapıları,
GRU

10:45–11:00



Kısa Mola

11:00–12:00



BiLSTM, Yığılmış LSTM,
Dropout

12:00–13:00



Dikkat Mekanizması & Self-
Attention

13:00–14:00

Öğle Arası

14:00–15:30



Metin Sınıflandırma (IMDB)
Uygulaması

15:30–16:30



Zaman Serisi Tahmini
(LSTM+Conv1D)

16:30–17:00

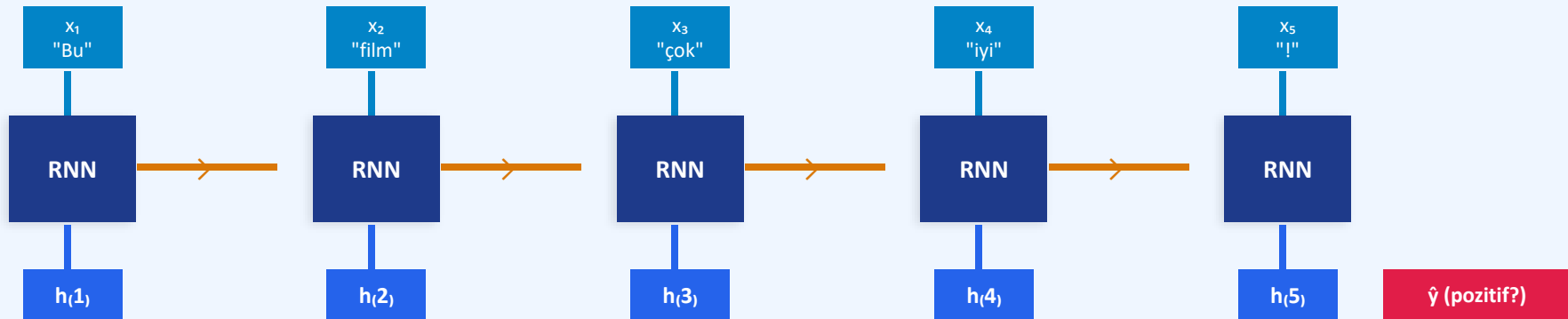


Proje Sunumu & Hafta
Kapanışı

🎯 Gün Sonu: IMDB'de %88+ accuracy + Hisse senedi 30 günlük tahmin + Custom Attention katmanı

Vanilla RNN — Sıralı Hafıza ve Kırılgan Gradyanlar

$$h_t = \tanh(W_h \cdot h_{t-1} + W_x \cdot x_t + b)$$



⚠️ Vanishing Gradient

Çok uzun sekanslarda (>50 adım) gradyan, geri yayılırken her RNN hücresinde küçülür. İlk kelimeler öğrenilemez.

⚠️ Exploding Gradient

Tam tersi: gradyan katlanarak büyür. Model ıraksıyor. Gradient clipping ile önlenir ama kök sorun çözülmez.

⚠️ Kısa Süreli Hafıza

Vanilla RNN pratikte sadece son 5-10 adımı 'hatırlayabilir'. Uzun bağımlılıklar (cümle başı↔sonu) yakalanamaz.

✅ Çözüm: LSTM / GRU

Kapı mekanizmaları bilginin ne zaman tutulup, ne zaman silineceğini öğrenir. Yüzlerce adım geriye bakabilir.

LSTM — Dört Kapı, Uzun Süreli Hafıza

Forget Gate · Input Gate · Cell State · Output Gate

Forget: $f = \sigma(W_f \cdot [h, x] + b_f)$ **Input:** $i = \sigma(W_i \cdot [h, x] + b_i)$ **Cell:** $\tilde{C} = \tanh(W_c \cdot [h, x] + b_c)$ **Output:** $o = \sigma(W_o \cdot [h, x] + b_o)$
 $h_t = o_t \cdot \tanh(C_t)$

Forget Gate (Unutma Kapısı)

Önceki hücre durumunun (C_{t-1}) ne kadarının devam edeceğine karar verir. σ çıkışı $[0,1]$: 0=tamamını unut, 1=tamamını tut.

```
f_t = sigmoid(W_f @ [h_{t-1}, x_t] + b_f)
```

+ Input Gate (Giriş Kapısı)

Yeni bilginin hücre durumuna ne kadar ekleneceğini belirler. Sigmoid (filtre) + Tanh (yeni değer) kombinasyonu.

```
i_t = sigmoid(W_i @ [h, x]) # filtre  
C_tilde = tanh(W_c @ [h, x]) # yeni değer
```

Cell State (Hücre Durumu)

Uzun süreli hafıza. Gradyanlar hücre durumu üzerinden neredeyse doğrudan akar — vanishing gradient minimum. 'Bilgi otoyolu'.

```
# Önceki bil. unut + yeni bil. ekle:  
C_t = f_t * C_{t-1} + i_t * C_tilde
```

Output Gate (Çıkış Kapısı)

Hücre durumunun hangi bölümünü çıkışa (h_t) vereceğine karar verir. Sonraki hücreye aktarılan gizli durum.

```
o_t = sigmoid(W_o @ [h, x])  
h_t = o_t * tanh(C_t)
```

GRU — Basitleştirilmiş Kapılar ve Karşılaştırma

Reset Gate · Update Gate · LSTM vs GRU vs SimpleRNN

GRU — Gated Recurrent Unit (2014)

Reset Gate (Sıfırlama)

$z_r = \sigma(W_r \cdot [h, x])$ – geçmiş hafızayı ne kadar 'sıfırla'

Update Gate (Güncelleme)

$z = \sigma(W_z \cdot [h, x])$ – ne kadar 'güncelle' vs geçmişi koru

Yeni Gizli Durum

$\tilde{h} = \tanh(W \cdot [z_r \odot h, x])$ – aday gizli durum

GRU'nun LSTM Avantajları:

- ✓ Daha az parametre (hücre durumu yok)
- ✓ Daha hızlı eğitim
- ✓ Küçük veri setlerinde genellikle daha iyi
- ⚠ Çok karmaşık/uzun sekanslarda LSTM geride bırakabilir

Özellik	SimpleRNN	GRU	LSTM
Parametre Sayısı	Az	Orta	Çok
Eğitim Hızı	Hızlı	Hızlı	Yavaş
Uzun Bağımlılık	Kötü	İyi	Çok İyi
Kapı Sayısı	0	2	4
Hücre Durumu	Yok	Yok	Var
Küçük Veri	Zayıf	İyi	İyi
Büyük Veri	Zayıf	İyi	Çok İyi
Metin Üretimi	Hayır	Evet	Evet
Tercih Durumu	Demo	Hız öncelikliyse	Doğruluk öncelikliyse

Keras kullanımı:

`layers.SimpleRNN(64)` | `layers.GRU(64, return_sequences=True)` | `layers.LSTM(128, dropout=0.2, recurrent_dropout=0.1)`

BiLSTM & Yığılmış LSTM — Daha Derin Bağlam

İleri + Geri Yön · Çok Katmanlı · Dropout Stratejileri

Bidirectional LSTM (BiLSTM)

Diziyi hem ileri (\rightarrow) hem geri (\leftarrow) yönde işler. Her adımda iki gizli durum çıktısı birleştirilir (concat/sum/avg). Cümle ortasındaki bir kelimenin hem önceki hem sonraki bağlamını görür.

★ NER, POS tagging, Q&A — bağlamı tam görmek kritik

```
x = layers.Bidirectional(  
    layers.LSTM(128, return_sequences=True),  
    merge_mode='concat' # (256,) çıkış  
) (x)
```

Yığılmış LSTM (Stacked)

Birden fazla LSTM katmanı üst üste yığılır. Alt katmanlar düşük seviye kalıpları (kelime, hece), üst katmanlar yüksek seviye anlam yapılarını öğrenir.

★ Metin üretimi, makine çevirisi — hiyerarşik özellik

```
x = layers.LSTM(128, return_sequences=True)(x)  
x = layers.Dropout(0.3)(x)  
x = layers.LSTM(64, return_sequences=True)(x)  
x = layers.LSTM(32)(x) # son katman
```

LSTM Dropout Stratejileri

dropout=0.2

Girdi bağlantıları ($x_t \rightarrow h_t$). Standart.

recurrent_dropout=0.1

Yineleme bağlantıları ($h_{t-1} \rightarrow h_t$).

Katmanlar arası

Dropout

LSTM blokları arasına Dropout(0.3) koy.

⚠ RNN'de Dropout

Her adımda aynı mask! tf-keras otomatik.

Dikkat Mekanizması — 'Nereye Bak?'

Bahdanau Additive Attention · Scaled Dot-Product · Multi-Head

Motivasyon: Encoder-Decoder Darboğazı

Geleneksel Seq2Seq'te encoder tüm girdiyi tek bir vektöre sıkıştırır (darboğaz). Uzun cümlelerde bilgi kaybolur. Dikkat mekanizması decoder'ın her adımında encoder'ın farklı konumlarına doğrudan bakmasına izin verir.

Bahdanau (Additive)

$$e_{ij} = V \cdot \tanh(W1 \cdot h_j + W2 \cdot s_i) \\ \alpha = \text{softmax}(e) \cdot \sum \alpha h = \text{context}$$

Encoder ve decoder gizli durumları toplanır. Öğrenilmiş ağırlık matrisi V ile puan hesaplanır.

★ Makine çevirisi, metin özetleme

Luong (Multiplicative)

$$e_{ij} = s_i \cdot W \cdot h_j \quad (\text{dot}) \\ \text{veya } s_i \cdot h_j \quad (\text{general})$$

Vektör çarpımı ile puan hesaplanır. Bahdanau'dan daha hızlı, büyük gizli boyutlarda etkili.

★ NMT, büyük seq2seq modeller

Scaled Dot-Product

$$\text{Attention}(Q, K, V) = \frac{\text{softmax}(QK^T / \sqrt{d_k})}{\sqrt{d_k}} \cdot V$$

Transformer'ın temeli. Q(sorgu), K(anahtar), V(değer). $\sqrt{d_k}$ ölçekleme gradyan patlamasını önler.

★ Transformer, BERT, GPT tabanlı modeller



Multi-Head Attention: Aynı girdiyi h farklı 'perspektiften' aynı anda işler — her kafa farklı ilişki örüntüsü yakalar. Transformer = Self-Attention + FFN

Keras ile Dikkat Mekanizması — Uygulama Kodu

Custom Attention Katmanı · MultiHeadAttention · Text Classification

custom_attention.py

```
# — ÖZEL DİKKAT KATMANI —————
class BahdanauAttention(keras.layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.W1 = layers.Dense(units, use_bias=False)
        self.W2 = layers.Dense(units, use_bias=False)
        self.V = layers.Dense(1, use_bias=False)

    def call(self, query, values):
        # query: (batch, d) values: (batch, T, d)
        q = tf.expand_dims(query, 1) # (B,1,d)
        score = self.V(tf.nn.tanh(
            self.W1(values) + self.W2(q))) # (B,T,1)
        weights = tf.nn.softmax(score, axis=1)
        context = weights * values # (B,T,d)
        context = tf.reduce_sum(context, axis=1) # (B,d)
        return context, weights

# — KERAS BUILT-IN MULTI-HEAD ATTENTION —————
mha = layers.MultiHeadAttention(
    num_heads=4,
    key_dim=64, # her kafa boyutu
    dropout=0.1
)
out, weights = mha(query=x, value=x, key=x,
```

Self-Attention (aynı dizi)

query=key=value=x. Dizinin kendi içindeki ilişkileri öğrenir. Transformer, BERT temeli.

Cross-Attention (farklı dizi)

query=decoder, key=value=encoder. Seq2Seq çeviride decoder encoder'a bakar.

Global Average Pooling ile Attention

Attention → context vektörü → GAP yerine. Metin sınıflandırmada etkili.

Attention Görselleştirme

return_attention_scores=True ile ağırlıkları çek. Isı haritasıyla hangi kelimenin hangisine baktığını gör.

✓ Keras MultiHeadAttention GPU optimize · attention_scores ile görselleştirme · causal_mask=True ile otoregresif modeller

Sekans Modeli Paternleri — Hangi Mimariyi Seçmeli?

One-to-Many · Many-to-One · Many-to-Many · Seq2Seq

One-to-One

Standart İleri Beslemeli Ağ. RNN gereksiz.

✦ Görüntü sınıflandırma, tabular veri

Keras: `return_sequences=False`

One-to-Many

Tek giriş, çoklu çıkış. Her adımda çıktı.

✦ Görüntü altyazı üretimi (image captioning)

Keras: `return_sequences=True`

Many-to-One

Dizi girişi, tek son çıkış.
`return_sequences=False`.

✦ Duygu analizi, spam tespiti, belge sınıflandırma

Keras: `return_sequences=False`

Many-to-Many (eş uzunluk)

Her adımda çıktı. `return_sequences=True`.

✦ Video karesi etiketleme, NER (ad varlık tanıma)

Keras: `return_sequences=True`

Seq2Seq (farklı uzunluk)

Encoder+Decoder. Farklı uzunluk diziler.

✦ Makine çevirisi, soru-cevap, özetleme

Keras: `return_sequences=True`



Birden fazla LSTM katmanında: aradaki tüm katmanlar `return_sequences=True`, sadece SON katman görevinize göre `True/False`

Metin Ön İşleme & Kelime Gömme (Embedding)

Tokenization · Padding · Embedding Layer · Pre-trained GloVe

metin_on_isleme.py

```
# — METİN VERİSİ HAZIRLAMA —————
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 1. Tokenizer — kelime → indeks
tokenizer = Tokenizer(
    num_words=20000, # en sık 20K kelime
    oov_token='<OOV>', # bilinmeyen kelime
)
tokenizer.fit_on_texts(X_train) # SADECE train!

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# 2. Padding — eşit uzunluk
X_train_pad = pad_sequences(X_train_seq,
    maxlen=256, # maksimum uzunluk
    padding='post', # sona sıfır ekle
    truncating='post') # sondan kırp

# 3. Embedding Layer (öğrenilebilir)
emb = layers.Embedding(
    input_dim=20000, # kelime sayısı
    output_dim=128, # vektör boyutu
    input_length=256, # sekans uzunluğu
```

Tokenization nedir?

Metni kelime/alt kelime birimlerine böler. 'Merhaba dünya' → ['merhaba', 'dünya'] → [45, 312]. Her kelimenin benzersiz indeksi var.

Padding neden gerekli?

RNN'ler sabit boyutlu batch bekler. Farklı uzunluktaki cümleler sıfırlarla doldurulur. mask_zero=True ile model padding'i yok sayar.

Embedding vs One-Hot

One-hot: 20000 boyutlu seyrek. Embedding: 128 boyutlu yoğun vektör. Anlamsal yakınlık: king-man+woman≈queen.

Pre-trained Embedding

GloVe, Word2Vec, FastText: milyarlarca cümleden önceden öğrenilmiş. Küçük veri setlerinde çok daha iyi.

Modern Yaklaşım: tf.keras.layers.TextVectorization → adaptasyon API · Sub-word tokenization (BPE) daha sağlam OOV kontrolü

```
# 4. Pre-trained: hub.KerasLayer('GloVe')
```

Zaman Serisi Tahmini — LSTM + Conv1D Hibrit

Pencere Yöntemi · Çoklu Adım · Kovaryantlar · Ölçekleme

zaman_serisi.py

```
# — PENCERELİ VERİ HAZIRLAMA —
def create_sequences(data, n_steps_in, n_steps_out):
    X, y = [], []
    for i in range(len(data)-n_steps_in-n_steps_out+1):
        X.append(data[i : i+n_steps_in])      # girdi penceresi
        y.append(data[i+n_steps_in :          # hedef penceresi
                    i+n_steps_in+n_steps_out])
    return np.array(X), np.array(y)

# — CONV1D + LSTM HİBRİT MODEL —
def build_conv_lstm(n_steps_in, n_features, n_steps_out):
    inp = keras.Input(shape=(n_steps_in, n_features))
    # Conv1D: yerel zaman kalıpları
    x = layers.Conv1D(64, kernel_size=3,
        padding='causal', activation='relu')(inp)
    x = layers.Conv1D(32, kernel_size=3,
        padding='causal', activation='relu')(x)
    # LSTM: uzun süreli bağımlılık
    x = layers.LSTM(128, return_sequences=True)(x)
    x = layers.Dropout(0.2)(x)
    x = layers.LSTM(64)(x)
    x = layers.Dropout(0.2)(x)
    x = layers.Dense(64, activation='relu')(x)
    out = layers.Dense(n_steps_out)(x) # n adım ilerisi
    return keras.Model(inp, out)
```

Veri Sızıntısı (Leakage)

Test verisinin istatistiği eğitimde kullanılmamalı. Scaler.fit() YALNIZCA train split'e. Temporal split zorunlu, random split yasak.

Causal Padding

padding='causal' → gelecek bilgisi sızmaz. Standart 'same' padding gelecek adımları görebilir — bu hata!

Çok Adımlı Tahmin

Direkt: tüm adımları aynı anda tahmin et (önerilen). Recursive: t+1'i tahmin, onu girdi yap — hata birikir.

Kovaryant Özellikler

Sadece hedef değişken değil; ilgili değişkenler de girdi olarak kullanılır. n_features > 1. İstatistiksel özellik mühendisliği önemli.

Metrikler: MAE, RMSE, MAPE (yüzde hata). Baseline: naive (son değeri kopyala). Model iyi mi? → Baseline'ı geçmeli.

Bugünkü Uygulamalar — 4 Python Dosyası

Sırasıyla çalıştır · IMDB + Sentetik Zaman Serisi verisi

01 RNN, LSTM, GRU — Temel Karşılaştırma

uygulama_01_rnn_lstm_gru.py

- ▶ Vanishing gradient demosu (gradyan normu izleme)
- ▶ SimpleRNN vs LSTM vs GRU ablasyon — IMDB
- ▶ Dizi uzunluğu etkisi analizi
- ▶ Gizli durum boyutu ablasyonu
- ▶ return_sequences davranışı görselleştirme

02 BiLSTM, Yığılmış LSTM & Dikkat

uygulama_02_bilstm_attention.py

- ▶ BiLSTM vs tek yönlü LSTM karşılaştırması
- ▶ 2/3/4 katmanlı stacked LSTM ablasyonu
- ▶ Custom Bahdanau Attention katmanı
- ▶ MultiHeadAttention ile metin sınıflandırma
- ▶ Dikkat ağırlıkları görselleştirme

03 Metin Sınıflandırma — IMDB Tam Pipeline

uygulama_03_metin_siniflandirma.py

- ▶ TextVectorization + Embedding + BiLSTM+Attention
- ▶ Pre-trained GloVe embedding deneyi
- ▶ Eşik optimizasyonu (precision-recall trade-off)
- ▶ Yanlış tahmin analizi + sınır vakaları
- ▶ F1-Score, AUC, Confusion Matrix görsel

04 Zaman Serisi Tahmini — LSTM+Conv1D

uygulama_04_zaman_serisi.py

- ▶ Hisse senedi verisi + sentetik çoklu değişken
- ▶ Conv1D → LSTM hibrit mimari
- ▶ Tek adım vs çok adım tahmin karşılaştırması
- ▶ Naive baseline vs LSTM karşılaştırması
- ▶ 30 günlük gelecek tahmini görselleştirme

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



Hafta 2 Tamamlandı!

Bu haftanın tam kazanım listesi:

- ✓ CNN: Conv2D, ResNet, DSConv, Transfer Learning, Grad-CAM
- ✓ GRU, BiLSTM, Yığılmış LSTM karşılaştırması yaptık
- ✓ Multi-Head Attention ve Keras entegrasyonu
- ✓ Conv1D+LSTM hibrit zaman serisi modeli
- ✓ RNN Vanishing Gradient → LSTM 4 kapısı tam kavradık
- ✓ Bahdanau Attention custom katman yazdık
- ✓ IMDB metin sınıflandırma tam pipeline
- ✓ Causal padding, veri sızıntısı önleme



Hafta 3: Transformer Mimarisi · BERT · GPT · Fine-Tuning · LLM Uygulamaları