11.HAFTA

YAPAY SİNİR AĞLARI

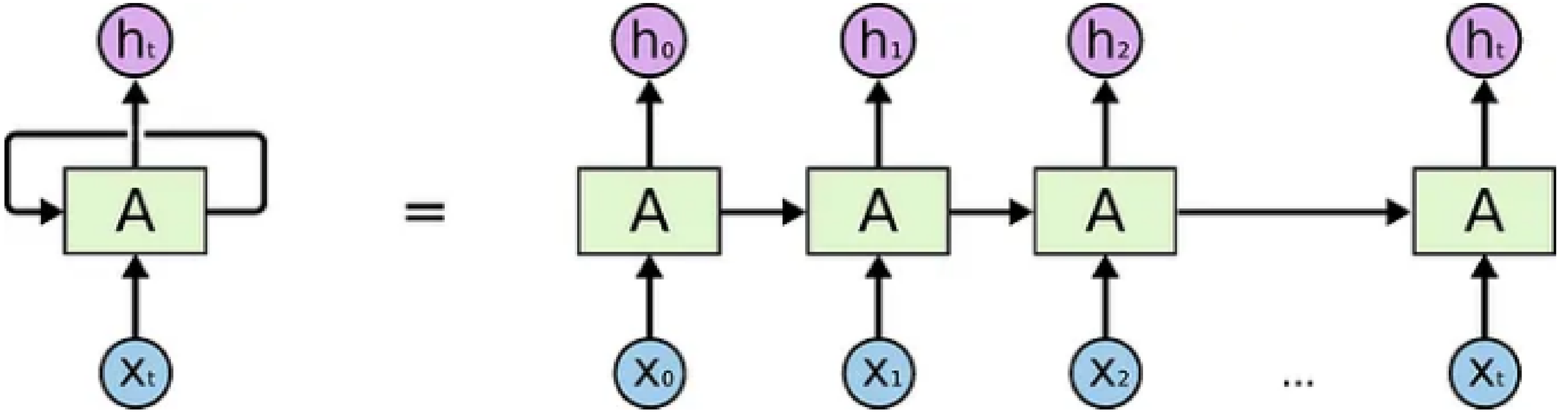
SAMSUN ÜNİVERSİTESİ

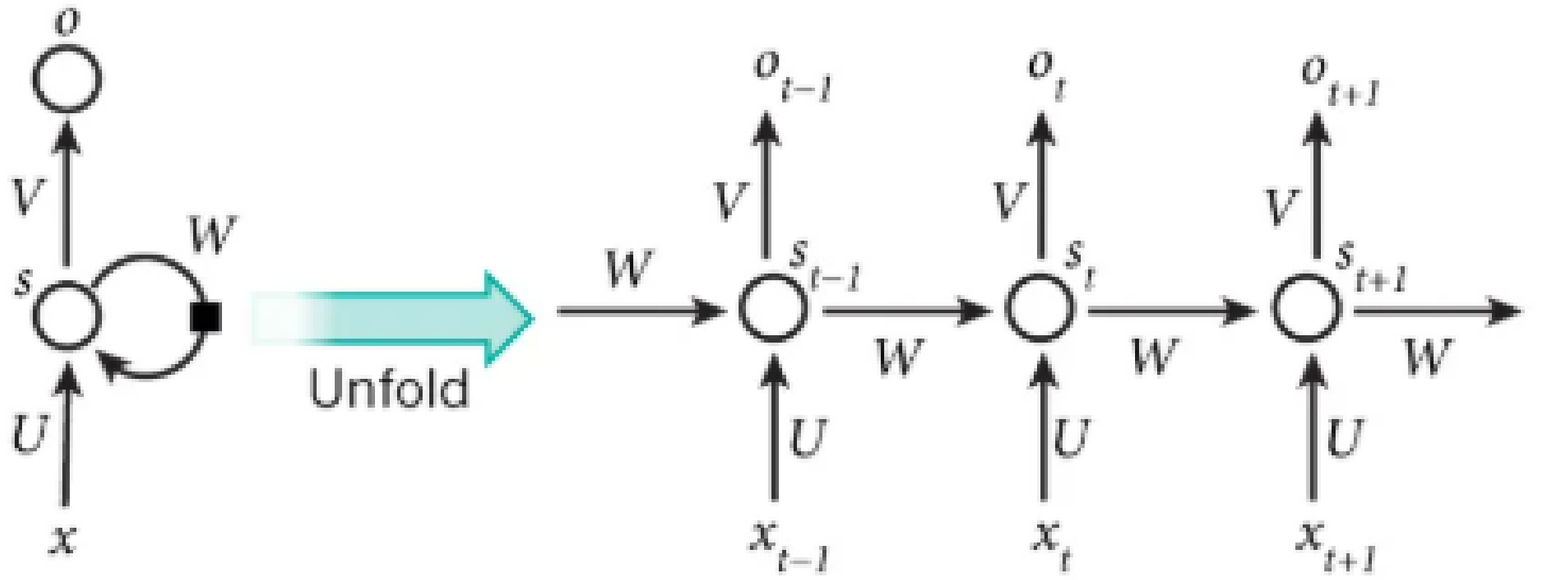
DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

Tekrar Eden Sinir Ağları (RNN)

Tekrar Eden Sinir Ağları (Recurrent Neural Networks – RNN), ardışık verilerdeki zaman bağımlılıklarını öğrenmek amacıyla geliştirilmiş sinir ağı yapılarıdır. Geleneksel yapay sinir ağlarından farklı olarak, RNN'ler geçmiş adımlardan gelen bilgiyi gizli durum (hidden state) aracılığıyla saklayarak yeni girişlerle birlikte işler.

- RNN'in en önemli özelliği, geri besleme (feedback) döngüsü sayesinde geçmiş bilgiyi saklamasıdır. Böylece her zaman adımı sadece o anki girdiye bağlı değil, aynı zamanda geçmiş girdilerin de bir fonksiyonudur.


h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t


o
V
W
s
U
x
Unfold
o_{t-1}
o_t
o_{t+1}
V
V
V
W
s_{t-1}
s_t
s_{t+1}
W
W
W
U
U
U
x_{t-1}
x_t
x_{t+1}

Matematiksel olarak RNN'in temel yapısı:

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

$$y_t = W_y \cdot h_t + b_y$$

Burada:

- x_t: t anındaki giriş
- h_t: gizli durum (hafıza)
- y_t: t anındaki çıktı
- W_x, W_h, W_y: ağırlık matrisleri
- b_h, b_y: bias terimleri
- f: genellikle **tanh** veya **ReLU** gibi aktivasyon fonksiyonları

RNN Türleri

- RNN'lerin farklı veri işleme senaryoları için geliştirilmiş çeşitleri bulunmaktadır:

1. **Vanilla RNN (Temel RNN):**
    - En basit formudur.
    - Sadece bir gizli katmana sahiptir.
    - Kısa süreli bağımlılıkları öğrenebilir, ancak uzun bağımlılıklarda gradyan sönmesi/patlaması yaşar.
2. **LSTM (Long Short-Term Memory):**
    - Uzun süreli bağımlılıkları öğrenmek için geliştirilmiştir.
    - "Hücre durumu" (cell state) ve kapılar (giriş, çıkış, unutma) kullanır.
    - Uzun dizilerde bilgi kaybını engeller.

3. **GRU (Gated Recurrent Unit):**
    - LSTM'e benzer fakat daha basit yapıya sahiptir.
    - Sadece güncelleme ve sıfırlama kapıları vardır.
    - Daha az hesaplama gerektirir, eğitim süresi daha kısadır.
4. **Bidirectional RNN (Çift Yönlü RNN):**
    - Veriyi hem ileri hem geri yönde işler.
    - Geçmiş ve gelecek bağlamı aynı anda dikkate alır.
    - Doğal dil işleme gibi alanlarda kullanılır.
5. **Deep RNN (Derin RNN):**
    - Birden fazla RNN katmanının üst üste eklenmesiyle oluşur.
    - Daha karmaşık ilişkileri öğrenebilir.

RNN Kullanım Alanları

RNN'ler, ardışık veya zaman bağımlı verilerin işlendiği alanlarda yaygın şekilde kullanılmaktadır. Başlıca kullanım alanları:

- **Doğal Dil İşleme (NLP):**
  - Metin tamamlama
  - Makine çevirisi
  - Duygu analizi
- **Konuşma ve Ses İşleme:**
  - Sesli asistanlar
  - Otomatik konuşma tanıma
  - Müzik oluşturma

- **Zaman Serisi Analizi:**
  - Finansal veri tahmini (örneğin hisse senedi fiyatları)
  - Hava durumu tahmini
  - IoT sensör verilerinin analizi
- **Diğer Uygulamalar:**
  - Video analizi (ardışık kareler arasındaki bağımlılık)
  - Sağlık alanında biyomedikal sinyallerin (EEG, EKG) işlenmesi

Tekrar Eden Sinir Ağları (RNN) ve Zaman Serisi

Zaman Serisi :

Zaman serisi, belirli zaman aralıklarında ölçülen ardışık veri noktalarından oluşan bir veri türüdür.

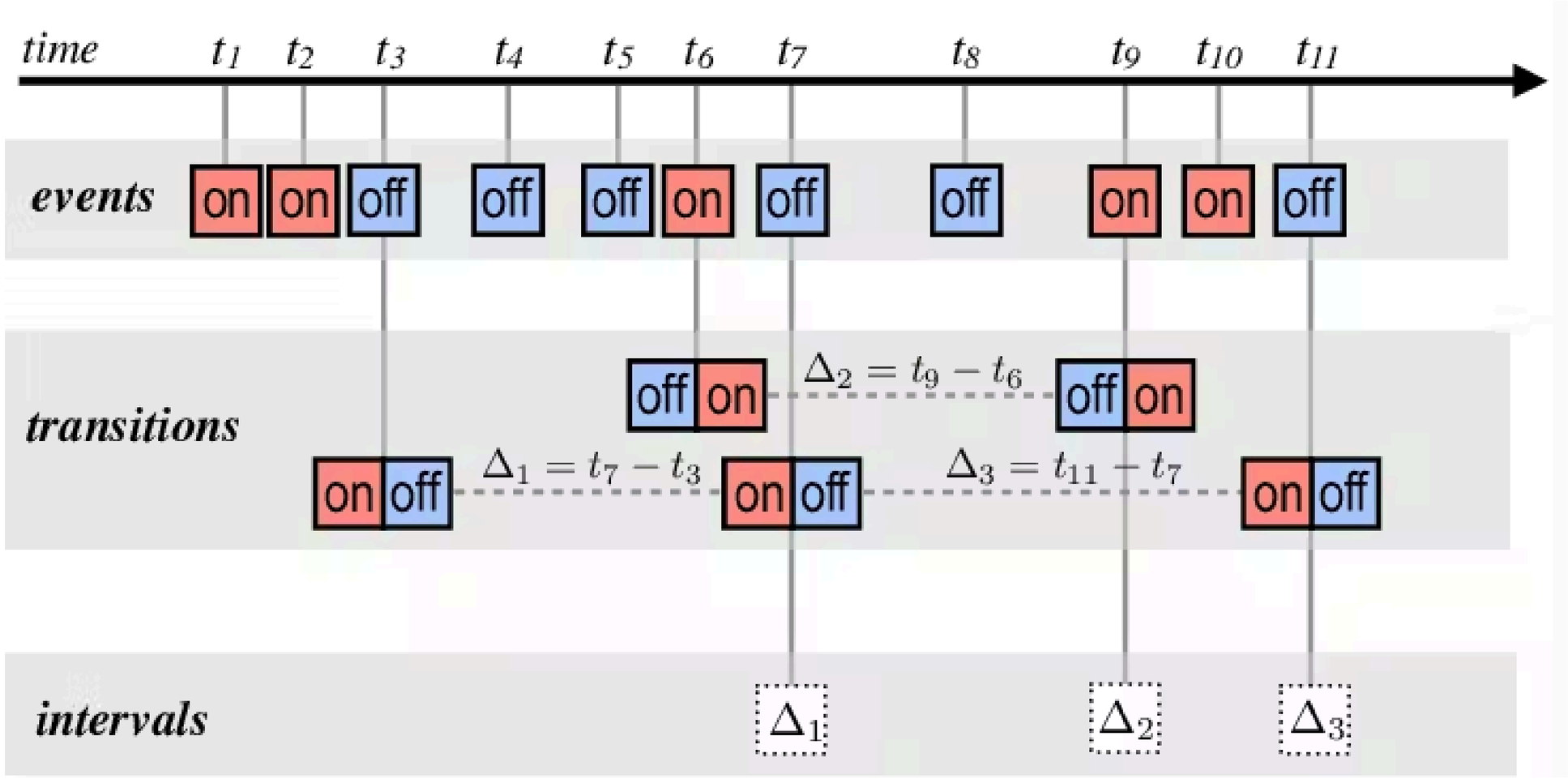
Örnekler:

- Günlük sıcaklık ölçümleri
- Saatlik elektrik tüketimi
- Dakikalık hisse senedi fiyatları
- Dakikada atılan adım sayısı (sağlık sensörleri)

Zaman serilerinin temel özelliği, veriler arasında zaman bağımlılığı bulunmasıdır. Yani mevcut bir değerin anlamı, geçmişteki değerlerle ilişkilidir.

RNN ve Zaman Serisi İlişkisi

Tekrar Eden Sinir Ağları (RNN), geçmiş bilgiyi hafızada tutabilme özelliğine sahip oldukları için zaman serilerinin modellenmesinde oldukça etkilidir.



Bir zaman serisi tahmininde RNN'in çalışma prensibi şu şekildedir:

1. **Girdi:** Zaman serisinin önceki değerleri ($x_{t-1}, x_{t-2}, \ldots$) ağa sırayla verilir.
2. **Gizli Durum:** Her zaman adımında gizli durum (h_t), geçmiş bilgileri saklayarak güncellenir.
3. **Çıktı:** Tahmin edilen değer (y_t), hem o anki girdi hem de geçmiş bilgilerin birleşiminden elde edilir.
4. **Tahmin:** Öğrenilen bağımlılıklar kullanılarak, gelecekteki değerler (x_{t+1}, x_{t+2}) tahmin edilebilir.

Matematiksel olarak:

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

$$\hat{y}_{t+1} = W_y \cdot h_t + c$$

RNN’in Zaman Serisi Üzerindeki Avantajları

- Geçmiş gözlemleri dikkate alarak geleceği tahmin edebilir.
- Veriler arasındaki zaman bağımlılıklarını öğrenir.
- Klasik istatistiksel yöntemlere (örneğin ARIMA) göre daha karmaşık ilişkileri yakalayabilir.
- Eksik veri veya gürültülü veriyle çalışırken daha esnektir

Karşılaşılan Zorluklar

- Uzun zaman serilerinde **gradyan sönmesi/patlaması** sorunları yaşanır.
- Hesaplama maliyeti yüksektir, eğitim süresi uzundur.
- Gürültülü ve düzensiz zaman serilerinde performans düşebilir.

Bu zorluklar nedeniyle pratikte genellikle RNN'in gelişmiş türleri olan **LSTM (Long Short-Term Memory)** ve **GRU (Gated Recurrent Unit)** tercih edilir.

Uygulama Alanları

- **Finans:** Hisse senedi fiyat tahmini, döviz kuru öngörüsü
- **Enerji:** Elektrik tüketim tahmini, yenilenebilir enerji üretimi
- **Sağlık:** Kalp atış hızı (EKG), beyin dalgaları (EEG) analizi
- **Meteoroloji:** Sıcaklık, yağış, rüzgar tahmini
- **IoT ve Sensör Verileri:** Akıllı şehir uygulamaları, trafik tahminleri

LSTM ve GRU Katmanları

- *RNN'deki uzun süreli bağımlılık sorunlarını çözmek üzere tasarlanmışlardır.*

LSTM (Long Short-Term Memory)

LSTM, 1997 yılında geliştirilmiş, klasik RNN'lerde görülen gradyan sönmesi (vanishing gradient) problemine çözüm getiren özel bir tekrar eden sinir ağı türüdür. LSTM, uzun vadeli bağımlılıkları öğrenebilmek için kapılar (gates) ve hücre durumu (cell state) kullanır.

Yapısı

LSTM hücresi üç ana kapıya sahiptir:

- **Unutma Kapısı (Forget Gate):** Hangi bilgilerin unutulacağını belirler.
- **Giriş Kapısı (Input Gate):** Hangi yeni bilgilerin hafızaya ekleneceğini belirler.
- **Çıkış Kapısı (Output Gate):** Hücreden hangi bilginin çıktı olarak kullanılacağını belirler.

Matematiksel olarak:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(unutma kapısı)}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(giriş kapısı)}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(aday hücre durumu)}$$

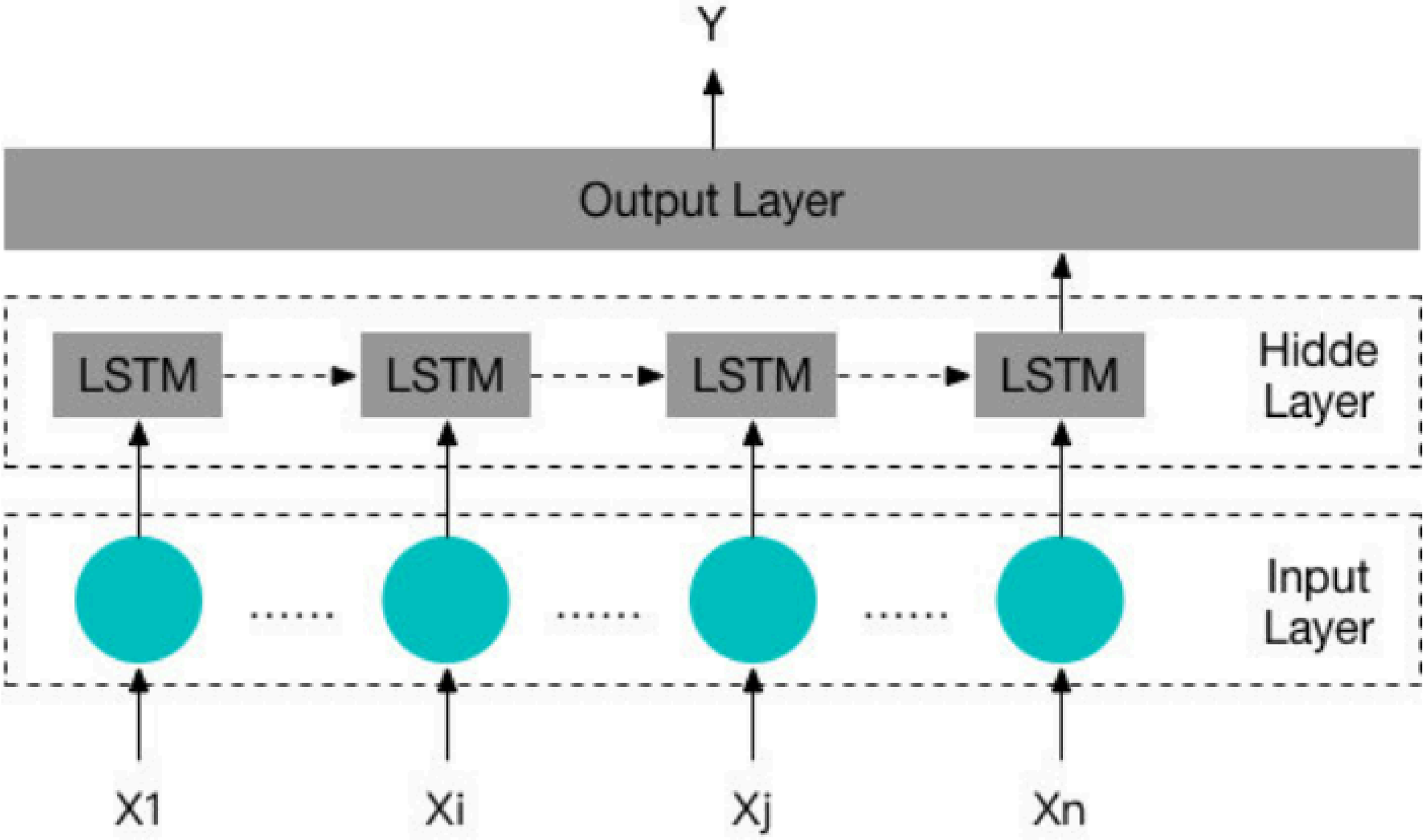
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad \text{(hücre güncellemesi)}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(çıkış kapısı)}$$

$$h_t = o_t \cdot \tanh(C_t) \quad \text{(gizli durum)}$$

Avantajları

- Uzun vadeli bağımlılıkları öğrenebilir.
- Metin, konuşma ve zaman serisi gibi alanlarda yüksek başarı sağlar.


Y
Output Layer
LSTM
LSTM
LSTM
LSTM
Hidde Layer
Input Layer
X1
Xi
Xj
Xn

GRU (Gated Recurrent Unit)

GRU, 2014 yılında önerilen ve LSTM'in daha basitleştirilmiş bir versiyonudur. LSTM gibi kapılar kullanır ancak daha az parametreye sahiptir. Bu sayede daha hızlı eğitim sağlar.

Yapısı

GRU, iki kapı içerir:

- **Güncelleme Kapısı (Update Gate):** Hangi bilgilerin korunacağını ve hangilerinin güncelleneceğini belirler.
- **Sıfırlama Kapısı (Reset Gate):** Geçmiş bilgilerin ne kadar unutulacağını belirler.

Matematiksel olarak:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad \text{(güncelleme kapısı)}$$

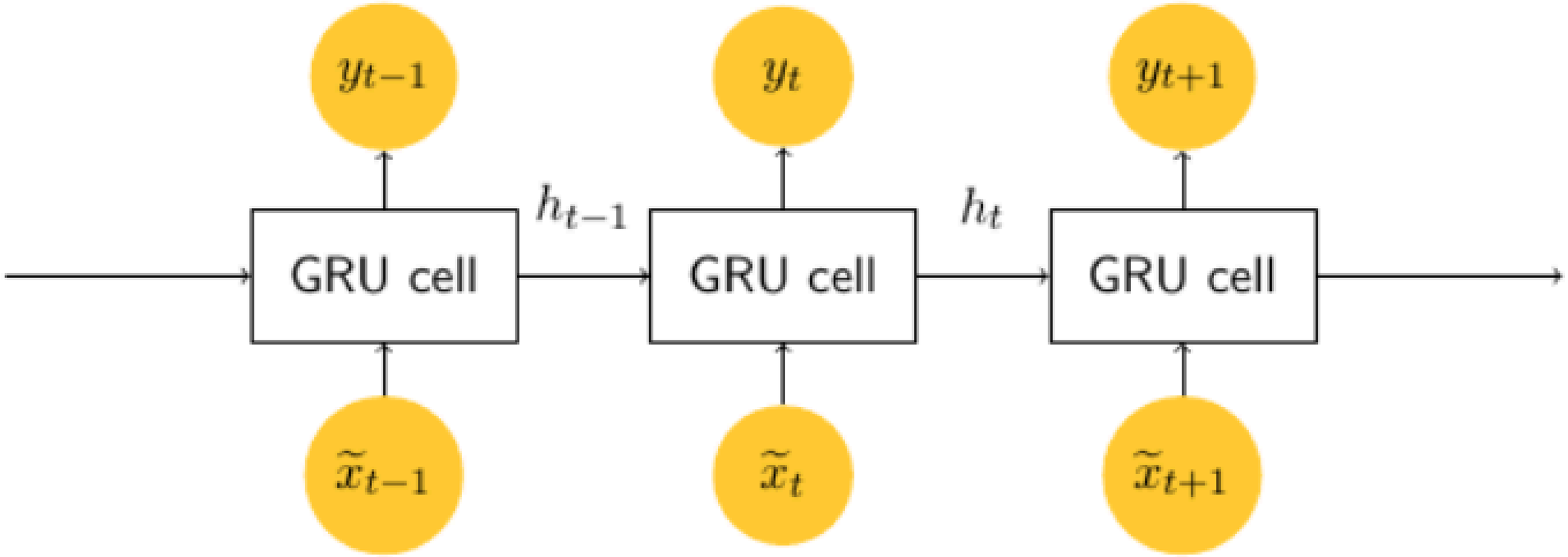
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad \text{(sıfırlama kapısı)}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \quad \text{(aday gizli durum)}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \quad \text{(gizli durum güncellemesi)}$$

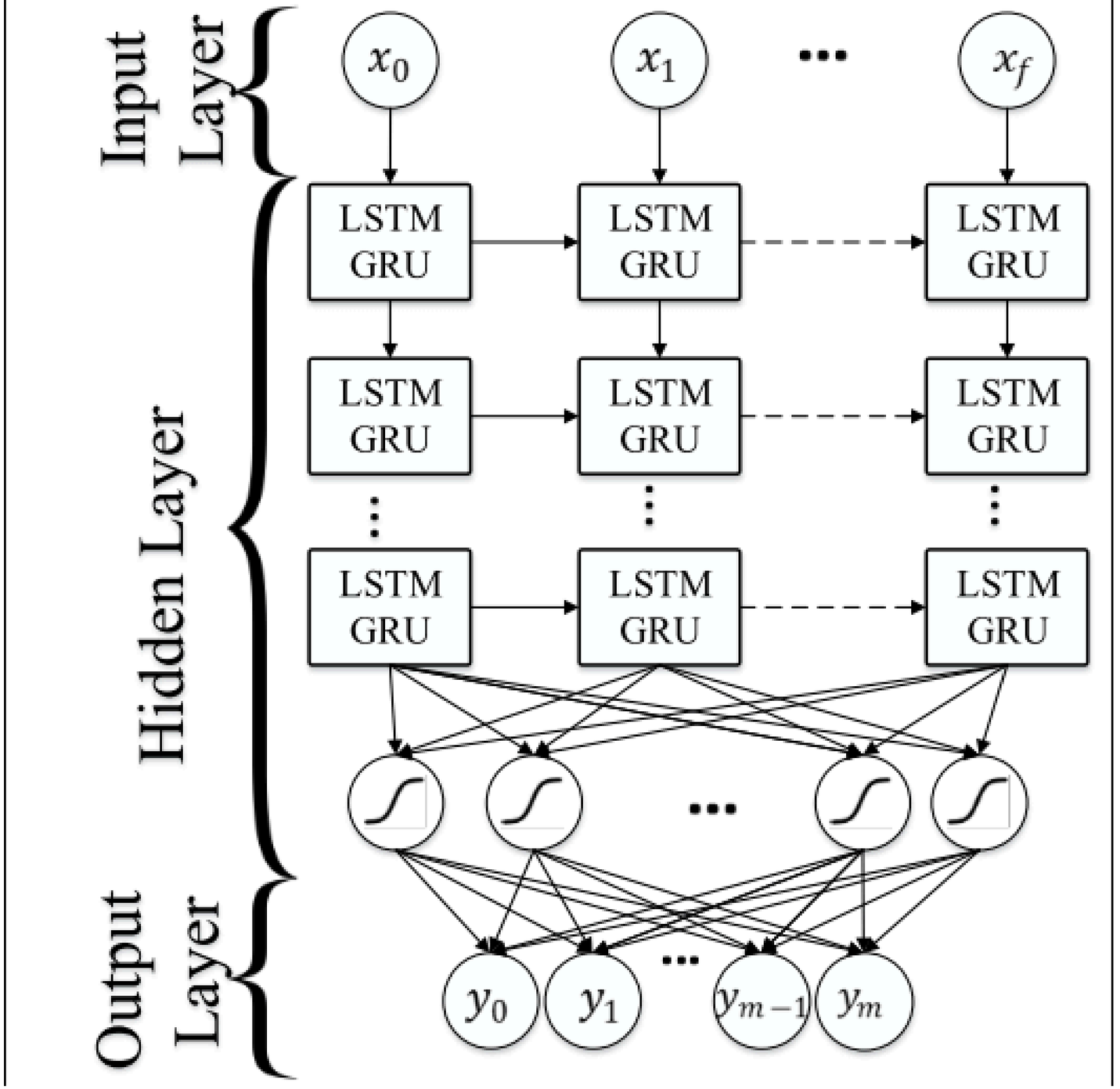
Avantajları

- LSTM'e göre daha basit ve hızlıdır.
- Daha az parametre içerir, bu nedenle küçük veri setlerinde daha etkilidir.


y_{t-1}
y_t
y_{t+1}
h_{t-1}
h_t
GRU cell
GRU cell
GRU cell
$\widetilde{x}_{t-1}$
$\widetilde{x}_t$
$\widetilde{x}_{t+1}$

LSTM ve GRU Karşılaştırması

Özellik	LSTM	GRU
Kapı Sayısı	3 (Unutma, Giriş, Çıkış)	2 (Güncelleme, Sıfırlama)
Parametre Sayısı	Daha fazla	Daha az
Hafıza Yapısı	Hücre durumu + gizli durum	Sadece gizli durum
Hesaplama Hızı	Daha yavaş	Daha hızlı
Performans	Uzun dizilerde daha güçlü	Daha küçük veri setlerinde yeterli
Kullanım Alanları	NLP, konuşma, uzun bağımlılıklar	Zaman serisi, daha basit görevler


Input Layer
Hidden Layer
Output Layer
x_0
x_1
x_f
LSTM GRU
y_0
y_1
y_{m-1}
y_m

IMDB veri setinde duygu analizi (LSTM örneği)

```
# Gerekli kütüphaneleri içe aktar
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt

# Rastgelelik için seed değerini ayarla (tekrarlanabilirlik için)
tf.random.set_seed(42)
np.random.seed(42)
```

```python
# Veri setini yükleme ve ön işleme
def load_and_preprocess_data():
    """
    IMDB veri setini yükler ve ön işler
    """
    # En sık kullanılan 10000 kelimeyi al
    num_words = 10000

    # Veri setini yükle (train ve test olarak)
    (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)

    # Kelime indeksini al (kelimeleri sayılara eşleyen sözlük)
    word_index = imdb.get_word_index()

    # İndeks-kelime sözlüğü oluştur (sayılardan kelimelere)
    reverse_word_index = {value: key for key, value in word_index.items()}

    return (X_train, y_train), (X_test, y_test), reverse_word_index, num_words
```

```python
# Dizileri aynı uzunluğa getir (padding)
def pad_sequences_data(X_train, X_test, maxlen=500):
    """
    Dizileri belirli bir uzunluğa padler
    """
    X_train = pad_sequences(X_train, maxlen=maxlen)
    X_test = pad_sequences(X_test, maxlen=maxlen)
    return X_train, X_test

# Sayı dizisini tekrar metne çeviren fonksiyon
def decode_review(numbers, reverse_word_index):
    """
    Sayı dizisini orijinal metne çevirir
    """
    # 0, 1, 2 özel indeksler olduğu için 3 çıkarırız
    return ' '.join([reverse_word_index.get(i - 3, '?') for i in numbers])
```

```python
# LSTM modelini oluştur
def create_lstm_model(vocab_size, embedding_dim=128, lstm_units=64):
    """
    LSTM modelini oluşturur
    """
    model = Sequential([
        # Gömme katmanı: kelimeleri vektörlere dönüştürür
        Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=500),

        # LSTM katmanı: zaman serisi verilerini işler
        LSTM(lstm_units, dropout=0.2, recurrent_dropout=0.2),

        # Dropout: overfitting'i önlemek için
        Dropout(0.5),

        # Çıkış katmanı: sigmoid aktivasyonu (0-1 arası çıktı)
        Dense(1, activation='sigmoid')
    ])

    return model
```

```python
# Modeli derle
def compile_model(model):
    """
    Modeli derler ve optimizer, loss fonksiyonunu ayarlar
    """
    model.compile(
        optimizer='adam',           # Optimizasyon algoritması
        loss='binary_crossentropy', # İkili sınıflandırma kayıp fonksiyonu
        metrics=['accuracy']        # Doğruluk metriği
    )
    return model
```

```python
# Modeli eğit
def train_model(model, X_train, y_train, X_test, y_test, epochs=5, batch_size=64):
    """
    Modeli eğitir ve geçmişi döndürür
    """
    history = model.fit(
        X_train, y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(X_test, y_test),
        verbose=1  # İlerleme çubuğunu göster
    )
    return history
```

```python
# Eğitim sonuçlarını görselleştir
def plot_training_history(history):
    """
    Eğitim ve validation kaybını/doğruluğunu görselleştirir
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

    # Kayıp grafiği
    ax1.plot(history.history['loss'], label='Eğitim Kaybı')
    ax1.plot(history.history['val_loss'], label='Validation Kaybı')
    ax1.set_title('Model Kaybı')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Kayıp')
    ax1.legend()

    # Doğruluk grafiği
    ax2.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
    ax2.plot(history.history['val_accuracy'], label='Validation Doğruluğu')
    ax2.set_title('Model Doğruluğu')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Doğruluk')
    ax2.legend()

    plt.tight_layout()
    plt.show()
```

```python
# Örnek tahminler yap
def make_predictions(model, X_sample, y_sample, reverse_word_index, num_samples=5):
    """
    Rastgele örnekler üzerinde tahminler yapar ve sonuçları gösterir
    """
    indices = np.random.choice(len(X_sample), num_samples, replace=False)

    for i, idx in enumerate(indices):
        review = X_sample[idx]
        true_label = y_sample[idx]

        # Tahmin yap (batch boyutu ekle)
        prediction = model.predict(np.array([review]), verbose=0)[0][0]
        predicted_label = 1 if prediction > 0.5 else 0

        # İlk 20 kelimeyi göster
        decoded_text = decode_review(review, reverse_word_index)
        preview_text = ' '.join(decoded_text.split()[:20]) + "..."
```

```
        print(f"Örnek {i+1}:")
        print(f"Metin: {preview_text}")
        print(f"Gerçek Etiket: {'Pozitif' if true_label == 1 else 'Negatif'}")
        print(f"Tahmin: {prediction:.4f} -> {'Pozitif' if predicted_label == 1 else 'Negati
f'}")
        print(f"Doğru Tahmin: {'Evet' if predicted_label == true_label else 'Hayır'}")
        print("-" * 50)

# Ana fonksiyon
def main():
    print("IMDB Duygu Analizi - LSTM Modeli")
    print("=" * 50)
```

```
    # 1. Veriyi yükle ve ön işle
    print("1. Veri yükleniyor ve ön işleniyor...")
    (X_train, y_train), (X_test, y_test), reverse_word_index, vocab_size = load_and_preprocess_
data()

    # 2. Dizileri padle
    print("2. Diziler aynı uzunluğa getiriliyor...")
    X_train, X_test = pad_sequences_data(X_train, X_test)

    print(f"Eğitim verisi şekli: {X_train.shape}")
    print(f"Test verisi şekli: {X_test.shape}")

    # 3. Modeli oluştur
    print("3. LSTM modeli oluşturuluyor...")
    model = create_lstm_model(vocab_size + 1)  # +1 çünkü indeksler 0'dan başlar
```

```python
# 4. Modeli derle
print("4. Model derleniyor...")
model = compile_model(model)

# Model özeti
model.summary()

# 5. Modeli eğit
print("5. Model eğitiliyor...")
history = train_model(model, X_train, y_train, X_test, y_test, epochs=5)

# 6. Sonuçları değerlendir
print("6. Model değerlendiriliyor...")
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Kaybı: {test_loss:.4f}")
print(f"Test Doğruluğu: {test_acc:.4f}")
```

```python
    # 7. Eğitim geçmişini görselleştir
    print("7. Eğitim geçmişi görselleştiriliyor...")
    plot_training_history(history)

    # 8. Örnek tahminler yap
    print("8. Örnek tahminler yapılıyor...")
    make_predictions(model, X_test, y_test, reverse_word_index)

    return model, history

# Programı çalıştır
if __name__ == "__main__":
    model, history = main()
```

```
IMDB Duygu Analizi - LSTM Modeli
==================================================
1. Veri yükleniyor ve ön işleniyor...
2. Diziler aynı uzunluğa getiriliyor...
Eğitim verisi şekli: (25000, 500)
Test verisi şekli: (25000, 500)
3. LSTM modeli oluşturuluyor...
4. Model derleniyor...
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 128)          1280128
 lstm (LSTM)                 (None, 64)                49408
 dropout (Dropout)           (None, 64)                0
 dense (Dense)               (None, 1)                 65
=================================================================
Total params: 1,329,601
Trainable params: 1,329,601
Non-trainable params: 0
_________________________________________________________________
5. Model eğitiliyor...
Epoch 1/5
391/391 [==============================] - 100s 250ms/step - loss: 0.4928 - accuracy: 0.7496 -
val_loss: 0.3578 - val_accuracy: 0.8460
...
6. Model değerlendiriliyor...
Test Kaybı: 0.3245
Test Doğruluğu: 0.8612
```