**ÖRNEK UYGULAMALAR**

1. **YAPAY SİNİR AĞI UYGULAMASI**

Yapay sinir ağları (YSA), insan beyninin işleyişinden esinlenerek oluşturulmuş matematiksel modeldir. Bu ağlar, veri içindeki karmaşık ilişkileri bulmak ve öğrenmek için kullanılır. Bir YSA, girdi katmanı, gizli katman(lar) ve çıktı katmanı olmak üzere genellikle üç katmandan oluşur.

Python kullanarak basit bir sinir ağı modeli oluşturalım. İlk olarak, yapay sinir ağı modelini oluşturmak için Keras kütüphanesini kullanacağız. Keras, sinir ağlarının hızlı bir şekilde oluşturulmasını ve eğitilmesini sağlayan kullanıcı dostu bir derin öğrenme kütüphanesidir.

Bu kodda, XOR mantıksal işlemi için bir yapay sinir ağı oluşturduk. XOR mantıksal işleminin doğruluk tablosu.

| **Giriş 1** | **Giriş 2** | **Çıkış** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* **BasitSinirAgı sınıfı:** Yapay sinir ağını temsil eder. Başlangıçta rastgele ağırlıklar ve bir bias değeriyle başlatılır.
* **aktivasyon\_fonksiyonu:** Sigmoid fonksiyonunu tanımlar. Sigmoid, aktivasyon fonksiyonu olarak kullanılır.
* **aktivasyon\_fonksiyonu\_turevi:** Sigmoid fonksiyonunun türevidir. Geri yayılım (backpropagation) işlemi için kullanılır.
* **tahmin:** Verilen girişe göre çıktı tahmini yapar. İleri yayılım (forward propagation) işlemi gerçekleştirir.
* **egit:** Verilen eğitim verileri üzerinde ağı eğitir. Geri yayılım (backpropagation) kullanarak ağı günceller.
* **Örnek Kullanım:** XOR işlemi için eğitim verileri oluşturulur. Sinir ağı oluşturulur, eğitilir ve test verisi üzerinde tahminler yapılır.

import numpy as np

class BasitSinirAgı:

def \_\_init\_\_(self):

# Rastgele ağırlıklar ve bias değeriyle başlatılıyor

self.w = np.random.rand(2, 1) # 2 giriş, 1 çıkış için ağırlıklar

self.b = np.random.rand(1)

def aktivasyon\_fonksiyonu(self, x):

# Basit bir aktivasyon fonksiyonu olarak sigmoid kullanıyoruz

return 1 / (1 + np.exp(-x))

def aktivasyon\_fonksiyonu\_turevi(self, x):

# Sigmoid fonksiyonunun türevi

return x \* (1 - x)

def tahmin(self, x):

# Ağırlıkları ve biası kullanarak tahmin yapıyoruz

z = np.dot(x, self.w) + self.b

return self.aktivasyon\_fonksiyonu(z)

def egit(self, x, y, iterasyon=100, ogrenme\_orani=0.1):

for i in range(iterasyon):

# İleri yayılım (forward propagation)

z = np.dot(x, self.w) + self.b

tahmin\_degeri = self.aktivasyon\_fonksiyonu(z)

# Geri yayılım (backward propagation)

hata = y - tahmin\_degeri

gradyan = hata \* self.aktivasyon\_fonksiyonu\_turevi(tahmin\_degeri)

# Ağırlıkları ve biası güncelleme

self.w += np.dot(x.T, gradyan) \* ogrenme\_orani

self.b += np.sum(gradyan) \* ogrenme\_orani

# Her iterasyonda hata kontrolü

if i % 10 == 0:

print(f"Iterasyon {i}, Hata: {np.mean(np.abs(hata))}")

# Örnek kullanım

if \_\_name\_\_ == "\_\_main\_\_":

# Eğitim verisi

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

# Sinir ağı oluşturma ve eğitme

sinir\_agi = BasitSinirAgı()

sinir\_agi.egit(X, y)

# Test verisi için tahmin yapma

test\_verisi = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

print("Tahminler:")

for test in test\_verisi:

print(test, ":", sinir\_agi.tahmin(test))

Ağ, iki giriş alır (0 veya 1) ve XOR işleminin sonucunu çıktı olarak verir. Modeli eğittikten sonra, modelin performansını değerlendirir ve girdi verilerine dayanarak tahminler yaparız.

|  |
| --- |
| # Keras'tan gerekli modülleri içe aktarın  from keras.models import Sequential  from keras.layers import Dense  # Veri işleme için NumPy'i içe aktarın  import numpy as np  # Örnek veri setini oluşturalım (X: girdi, y: çıktı)  X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Girişler  y = np.array([[0], [1], [1], [0]]) # Çıkışlar  # Yapay sinir ağı modelini oluşturun  model = Sequential() # Boş bir model oluşturulur  # Girdi katmanı ve gizli katmanı ekleme  model.add(Dense(4, input\_dim=2, activation='relu')) # 4 nöronlu gizli katman, ReLU aktivasyon fonksiyonu  # input\_dim: Giriş boyutunu belirtir  # Çıkış katmanını ekleme  model.add(Dense(1, activation='sigmoid')) # 1 nöronlu çıkış katmanı, Sigmoid aktivasyon fonksiyonu  # Modeli derleme  model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])  # loss: Kayıp fonksiyonu (binary\_crossentropy: ikili sınıflandırma için kullanılır)  # optimizer: Optimizasyon algoritması (adam: adaptif momentum algoritması)  # metrics: Değerlendirme metrikleri (doğruluk metriği kullanılır)  # Modeli eğitme  model.fit(X, y, epochs=100, batch\_size=4)  # epochs: Eğitim döngülerinin sayısı  # batch\_size: Her güncellemede kullanılacak örnek sayısı  # Modeli değerlendirme  loss, accuracy = model.evaluate(X, y)  print("Loss:", loss)  print("Accuracy:", accuracy)  # Tahminler yapma  predictions = model.predict(X)  print("Predictions:", predictions) |

1. **DERİN ÖĞRENME UYGULMASI**

Keras kullanarak basit bir evrişimli sinir ağı (convolutional neural network - CNN) örneği

* **Conv2D():** Evrişimli katman oluşturur. İlk parametre, katmandaki filtre (kernel) sayısını belirtir. kernel\_size ise filtre boyutunu belirler.
* **MaxPooling2D():** Havuzlama katmanı oluşturur. Resmi küçültmek ve özelliklerini vurgulamak için kullanılır.
* **Flatten():** Düzleştirme katmanı, bir görüntüyü tek boyutlu bir vektöre dönüştürür.
* **Dense():** Tam bağlı (fully connected) katmanlar oluşturur.

Diğer adımlar, önceki örnekle aynıdır: model derlenir, eğitilir ve test edilir. Bu örnekte, CNN kullanılarak MNIST veri setindeki el yazısı rakamlarının sınıflandırılması hedeflenmiştir.

import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from keras.utils import to\_categorical

# MNIST veri setini yükleme

(X\_egitim, y\_egitim), (X\_test, y\_test) = mnist.load\_data()

# MNIST veri setinden örnek görüntülerin gösterilmesi

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i + 1)

plt.imshow(X\_egitim[i].reshape(28, 28), cmap='gray')

plt.title(f'Etiket: {np.argmax(y\_egitim[i])}')

plt.axis('off')

plt.show()

# Giriş verilerini normalize etme ve şeklini ayarlama

X\_egitim = X\_egitim.astype('float32') / 255

X\_test = X\_test.astype('float32') / 255

X\_egitim = np.expand\_dims(X\_egitim, axis=-1)

X\_test = np.expand\_dims(X\_test, axis=-1)

# Çıkış etiketlerini kategorik hale getirme

y\_egitim = to\_categorical(y\_egitim, 10)

y\_test = to\_categorical(y\_test, 10)

# Model oluşturma

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1))) # Evrişim katmanı

model.add(MaxPooling2D(pool\_size=(2, 2))) # Havuzlama katmanı

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu')) # Evrişim katmanı

model.add(MaxPooling2D(pool\_size=(2, 2))) # Havuzlama katmanı

model.add(Flatten()) # Düzleştirme katmanı

model.add(Dense(128, activation='relu')) # Tam bağlantılı gizli katman

model.add(Dense(10, activation='softmax')) # Çıkış katmanı

# Modeli derleme

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Modeli eğitme

model.fit(X\_egitim, y\_egitim, batch\_size=128, epochs=10, verbose=1, validation\_split=0.1)

# Modeli değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test verisi üzerinde kayıp (loss): {loss}')

print(f'Test verisi üzerinde doğruluk (accuracy): {accuracy}')

1. **DERİN ÖĞRENME UYGULMASI 2**

Bu kod, aynı MNIST veri setini kullanarak el yazısı rakamlarını sınıflandırır, yöntem olarak tam bağlantılı bir yapay sinir ağı (Multilayer Perceptron-MLP) kullanır. Modeli oluşturur, derler, eğitir ve test eder. Sonuçları değerlendirir ve kayıp (loss) ile doğruluk (accuracy) sonuçlarını ekrana yazdırır.

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Flatten

from keras.utils import to\_categorical

# MNIST veri setini yükleme

(X\_egitim, y\_egitim), (X\_test, y\_test) = mnist.load\_data()

# Giriş verilerini normalize etme ve şeklini ayarlama

X\_egitim = X\_egitim.astype('float32') / 255

X\_test = X\_test.astype('float32') / 255

X\_egitim = X\_egitim.reshape(X\_egitim.shape[0], -1)

X\_test = X\_test.reshape(X\_test.shape[0], -1)

# Çıkış etiketlerini kategorik hale getirme

y\_egitim = to\_categorical(y\_egitim, 10)

y\_test = to\_categorical(y\_test, 10)

# Model oluşturma

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(784,)))

model.add(Dense(256, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Modeli derleme

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Modeli eğitme

model.fit(X\_egitim, y\_egitim, batch\_size=128, epochs=10, verbose=1, validation\_split=0.1)

# Modeli değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test verisi üzerinde kayıp (loss): {loss}')

print(f'Test verisi üzerinde doğruluk (accuracy): {accuracy}')

1. **DERİN ÖĞRENME UYGULMASI 3**

Keras kullanarak bir duyarlılık analizi (sentiment analysis) modeli oluşturacağız. Bu model, IMDb film incelemelerinin pozitif mi yoksa negatif mi olduğunu belirlemek için kullanılabilir.

import numpy as np

from keras.datasets import imdb

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM

from keras.preprocessing import sequence

# Veri setini yükleme

max\_kelime = 10000

(X\_egitim, y\_egitim), (X\_test, y\_test) = imdb.load\_data(num\_words=max\_kelime)

# İncelemeleri aynı uzunluğa getirme

max\_uzunluk = 200

X\_egitim = sequence.pad\_sequences(X\_egitim, maxlen=max\_uzunluk)

X\_test = sequence.pad\_sequences(X\_test, maxlen=max\_uzunluk)

# Model oluşturma

model = Sequential()

model.add(Embedding(max\_kelime, 128, input\_length=max\_uzunluk))

model.add(LSTM(128, dropout=0.2, recurrent\_dropout=0.2))

model.add(Dense(1, activation='sigmoid'))

# Modeli derleme

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Modeli eğitme

model.fit(X\_egitim, y\_egitim, batch\_size=32, epochs=5, validation\_data=(X\_test, y\_test))

# Modeli değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test verisi üzerinde kayıp (loss): {loss}')

print(f'Test verisi üzerinde doğruluk (accuracy): {accuracy}')

**Embedding Katmanı:**

Embedding katmanı, kelime gömme (word embedding) işlemini gerçekleştirir. Bu işlem, metin verisini temsil etmek için kelimeleri sayısal vektörlerle eşlemek için kullanılır. Embedding katmanı, kelimeleri belirli bir boyutta (genellikle belirli bir vektör uzunluğunda) yoğun bir vektör olarak temsil eder. Bu, metin verisinin daha iyi işlenmesini ve modelin metinsel ilişkileri öğrenmesini sağlar.

**LSTM Katmanı (Long Short-Term Memory):**

LSTM katmanı, tekrarlayan sinir ağlarında (RNN) sıklıkla kullanılan bir türdür. LSTM hücreleri, zaman serisi verileri gibi sıralı verileri işlemek için tasarlanmıştır. LSTM, geleneksel RNN'lerdeki bellek sorununu çözmek için özel bir mimari kullanır. Bu sayede, uzun vadeli bağımlılıkları daha etkili bir şekilde modelleyebilirler. LSTM hücreleri, geçmiş bilgileri saklayarak ve unutmalarını kontrol ederek gelen verileri işler. Bu, metin verisi gibi sıralı verilerin işlenmesinde kullanılabilir.

1. **DERİN ÖĞRENME UYGULMASI 4**

Bu örnekte, Keras kütüphanesi kullanılarak bir evrişimli sinir ağı (Convolutional Neural Network - CNN) oluşturarak, CIFAR-10 veri setindeki görüntüleri sınıflandıracağız. CIFAR-10 veri seti, 10 farklı sınıfa ait 32x32 piksel boyutundaki renkli görüntülerden oluşur.

Bu kod, CIFAR-10 veri setindeki 32x32 piksel boyutundaki renkli görüntülerin sınıflandırılması için bir evrişimli sinir ağı (CNN) modeli oluşturur. Veri seti, arabalar, uçaklar, kuşlar, kediler, geyikler, köpekler, kurbağalar, atlar, gemiler ve kamyonlar gibi farklı nesneleri içerir. Model, evrişim (Conv2D), ölçeklendirme (MaxPooling2D) ve tam bağlantı (Dense) katmanları içerir. Veriler normalize edilir ve çıkış etiketleri kategorik forma dönüştürülür. Son olarak, model eğitilir ve test edilir, ve modelin performansı kayıp (loss) ve doğruluk (accuracy) değerleriyle değerlendirilir.

import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import cifar10

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from keras.utils import to\_categorical

# CIFAR-10 veri setini yükleme

(X\_egitim, y\_egitim), (X\_test, y\_test) = cifar10.load\_data()

# Sınıf etiketlerini belirleme

siniflar = ['Uçak', 'Otomobil', 'Kuş', 'Kedi', 'Geyik', 'Köpek', 'Kurbağa', 'At', 'Gemi', 'Kamyon']

# İlk 5 görüntüyü görselleştirme

plt.figure(figsize=(10, 5))

for i in range(5):

plt.subplot(1, 5, i + 1)

plt.imshow(X\_egitim[i])

plt.title(siniflar[int(y\_egitim[i])])

plt.axis('off')

plt.show()

# Giriş verilerini normalize etme ve şeklini ayarlama

X\_egitim = X\_egitim.astype('float32') / 255

X\_test = X\_test.astype('float32') / 255

# Çıkış etiketlerini kategorik hale getirme

y\_egitim = to\_categorical(y\_egitim, 10)

y\_test = to\_categorical(y\_test, 10)

# Model oluşturma

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Modeli derleme

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Modeli eğitme

model.fit(X\_egitim, y\_egitim, epochs=10, batch\_size=64, validation\_split=0.1)

# Modeli değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test verisi üzerinde kayıp (loss): {loss}')

print(f'Test verisi üzerinde doğruluk (accuracy): {accuracy}')

1. **DERİN ÖĞRENME UYGULMASI 5**

Doğal dil işleme (NLP) alanında duyarlılık analizi (sentiment analysis) yapabiliriz. Bu örnekte, Twitter veri setindeki tweetlerin duyarlığını (olumlu veya olumsuz) belirleyen bir model oluşturacağız. İşte bir başlangıç örneği: Bu örnekte, bir LSTM tabanlı bir model kullanarak Twitter veri setindeki tweetlerin duyarlığını belirlemeye çalışıyoruz. Veri seti, örnek tweetler ve bunların olumlu veya olumsuz olarak etiketlenmiş duyarlılıklarını içerir. Öncelikle, metin verilerini sayısal dizi formatına dönüştürmek için Tokenizer kullanıyoruz. Ardından, LSTM tabanlı bir model oluşturuyoruz. Eğitim ve test verilerini kullanarak modeli eğitiyoruz ve performansını değerlendiriyoruz.

import numpy as np

from sklearn.model\_selection import train\_test\_split

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense

from wordcloud import WordCloud

import matplotlib.pyplot as plt

# Örnek veri seti (Twitter tweetleri ve duyarlıkları)

tweets = ["Bu film harikaydı!", "Bu kitap çok kötüydü.", "Harika bir gün geçirdim.", "Üzgünüm, bu çok kötü bir haber."]

duyarliklar = [1, 0, 1, 0] # 1: Olumlu, 0: Olumsuz

# Tweetleri birleştirme

tum\_tweetler = ' '.join(tweets)

# Kelime bulutu oluşturma

kelime\_bulutu = WordCloud(width=800, height=400, background\_color='white').generate(tum\_tweetler)

# Kelime bulutunu görselleştirme

plt.figure(figsize=(10, 5))

plt.imshow(kelime\_bulutu, interpolation='bilinear')

plt.axis('off')

plt.show()

# Veri setini eğitim ve test olarak ayırma

X\_egitim, X\_test, y\_egitim, y\_test = train\_test\_split(tweets, duyarliklar, test\_size=0.2, random\_state=42)

# Kelime seviyesinde gömme (embedding) yapmak için metinleri dönüştürme

max\_kelime = 1000

tokenizer = Tokenizer(num\_words=max\_kelime)

tokenizer.fit\_on\_texts(X\_egitim)

X\_egitim\_seq = tokenizer.texts\_to\_sequences(X\_egitim)

X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

# Metinleri aynı uzunluğa (max\_len) getirme

max\_len = 20

X\_egitim\_pad = pad\_sequences(X\_egitim\_seq, maxlen=max\_len)

X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=max\_len)

# Model oluşturma

model = Sequential()

model.add(Embedding(max\_kelime, 64, input\_length=max\_len))

model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))

# Modeli derleme

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

X\_egitim\_pad = np.array(X\_egitim\_pad)

X\_test\_pad = np.array(X\_test\_pad)

y\_egitim = np.array(y\_egitim)

y\_test = np.array(y\_test)

# Modeli eğitme

model.fit(X\_egitim\_pad, y\_egitim, epochs=5, batch\_size=32, validation\_data=(X\_test\_pad, y\_test))

# Modeli değerlendirme

loss, accuracy = model.evaluate(X\_test\_pad, y\_test)

print(f'Test verisi üzerinde kayıp (loss): {loss}')

print(f'Test verisi üzerinde doğruluk (accuracy): {accuracy}')