

# SIFIRDAN YAPAY SİNİR AĞI EĞİTİLMESİ

## DERS:BİLGİSAYAR TASARIM VE UYGULAMALARI

NEVŞEHİR HACI BEKTAŞ VELİ ÜNİVERSİTESİ  
LEYLA KIZILKAYA – BİLGİSAYAR MÜHENDİSLİĞİ  
20260810035  
AKADEMİSYEN:DR.EMRE BENDEŞ

```
main.py x
1 from typing import List
2
3 import pandas as pd
4 import numpy as np
5 from sklearn.model_selection import KFold
6 import matplotlib.pyplot as plt
7 from sklearn.preprocessing import StandardScaler
8
```

✚ Bu satırlar, kullanılan kütüphaneleri içeri aktarır. Özellikle, pandas (veri çerçeveleri için), numpy (sayısal hesaplamalar için), KFold (çapraz doğrulama için), matplotlib.pyplot (grafikleme için) ve StandardScaler (verileri ölçeklendirmek için) kütüphaneleri kullanılmıştır.

```
2 usages
1 class Activation:
2     def run(self, x):
3         pass
4
5     def derivative(self, x):
6         pass
```




✚ Bu kısım, aktivasyon fonksiyonlarını temsil eden bir temel sınıf olan **Activation** sınıfını tanımlar. Aktivasyon fonksiyonları genellikle bir ağı geçtikten sonra kullanılan matematiksel fonksiyonlardır.

```
main.py x
15 1 usage
16 class Sigmoid(Activation):
17 def run(self, x):
18     return 1 / (1 + np.exp(-x))
19
20 def derivative(self, x):
21     return self.run(x) * (1 - self.run(x))
22
23 1 usage
24 class Relu(Activation):
25 def run(self, x):
26     return np.maximum(0, x)
27
28 def derivative(self, x):
29     return 1. * (x > 0)




1 usage
class Tanh(Activation):
def run(self, x):
    return np.tanh(x)

def derivative(self, x):
    return 1.0 - np.tanh(x) ** 2
```



### Sigmoid Sınıfı:

-  Sigmoid, aktivasyon fonksiyonu olarak sigmoid fonksiyonunu uygular.
-  run fonksiyonu, giriş verilerini alır (x) ve sigmoid fonksiyonunu uygulayarak çıkışı hesaplar.
-  derivative fonksiyonu, sigmoid fonksiyonunun türevidir. Sigmoid fonksiyonunun türevi, sigmoid fonksiyonunu girişiyle ve 1 eksiğiyle çarparak hesaplanır.

### Relu Sınıfı:

-  Relu, Rectified Linear Unit (ReLU) aktivasyon fonksiyonunu uygular.
-  run fonksiyonu, giriş verilerini alır (x) ve ReLU fonksiyonunu uygulayarak çıkışı hesaplar.
-  derivative fonksiyonu, ReLU fonksiyonunun türevidir. Eğer giriş pozitifse, türev 1'dir; negatifse, türev 0'dır.

### Tanh Sınıfı:

-  run metodunda, hiperbolik tanjant fonksiyonu olan **np.tanh** fonksiyonu kullanılarak giriş verileri **x** üzerinde işlem yapılır. Bu işlem sonucu, hiperbolik tanjant fonksiyonunun çıkışı elde edilir. Yani, bu metot, tanh(x) değerini hesaplar.
-  derivative fonksiyonu, hiperbolik tanjant fonksiyonunun türevidir.

```

1 usage
class Layer:
    def __init__(self, input_size, output_size, activation):
        # Katmanın ağırlıklarını rastgele başlattım.
        self.weights = np.random.randn(input_size, output_size) * 0.01
        # Katmanın bias'ını sıfırladım.
        self.bias = np.zeros(output_size)
        # Katmanın aktivasyon fonksiyonunu belirledim.
        self.activation = activation
        # Katmanın giriş ve çıkışlarını saklamak için değişkenler
        self.inputs = None
        self.output = None

    def forward(self, inputs):
        # Girişleri saklar.
        self.inputs = inputs
        # Girişleri ağırlıklarla çarpar, bias ekler ve aktivasyon fonksiyonunu uygular.
        self.output = self.activation.run(np.dot(inputs, self.weights) + self.bias)
        # Çıkışı döndürür.
        return self.output

```

- ✚ `__init__` metodu: Bu metot, Layer sınıfının bir örneği oluşturulduğunda çağrılır. İlgili katmanın ağırlıklarını rastgele başlatır, bias'ı sıfırlar, aktivasyon fonksiyonunu belirler ve giriş/çıkış değişkenlerini başlatır.
- ✚ `forward` metodu: Bu metot, katmanın ileri geçişini (forward pass) gerçekleştirir. Girişleri (inputs) alır, ağırlıklarla çarpar, bias'ı ekler ve aktivasyon fonksiyonunu uygular. Son olarak, çıkışı döndürür.
- ✚ Bu sınıf, genellikle sinir ağı modellerinde kullanılan katmanları temsil etmek için kullanılır. `activation` parametresi olarak, `tanh`, `sigmoid`, `ReLU` gibi aktivasyon fonksiyonlarını alabilir. Bu sınıfın kullanımı, bir sinir ağı modeli oluşturmak için bir dizi katmanı bir araya getirmemiz gerektiğinde gerçekleşir.

```

main.py x
57
58 def backward(self, error, learning_rate):
59     # Aktivasyonun türevini hesapladım
60     activation_derivative = self.activation.derivative(self.output)
61
62     # Delta hesapladım
63     delta = error * activation_derivative * self.activation.run(self.output)
64
65     # Ağırlık güncellemelerini yaptım
66     self.derivative_weights = np.outer(self.inputs, delta) # Ağırlık türevi
67     # Bu ifade, giriş vektörü ile delta vektörünün dış çarpımını hesaplar.
68     # Yani, her bir giriş özelliği ile delta arasındaki ilişkiyi gösterir. Elde edilen matris, her bir ağırlığın ne kadar değişmesi gerektiğini belirtir.
69     # Örneğin, eğer giriş vektörü 3 özellik içeriyorsa ve delta vektörü de 2 çıkış özelliği içeriyorsa, bu matris 3x2 boyutunda olacaktır.
70     # Matrisin her bir elemanı, bir ağırlığın ne kadar değişmesi gerektiğini gösterir.
71
72
73     self.derivative_bias = delta # Bias türevi
74

```

```

# Ağırlıkları ve bias'ı güncelledim
self.weights += learning_rate * np.outer(self.inputs.T, delta)

# Ağırlıkları güncelle
# NOT:self.derivative_weights = np.outer(self.inputs, delta) satırı, ağırlıkların türevidir ve
# her bir ağırlığın ne kadar değişmesi gerektiğini gösterir. np.outer(self.inputs, delta) ifadesi,
# giriş vektörü ile delta vektörünün dış çarpımını temsil eder. Bu çarpım, her bir giriş özelliğinin delta ile çarpılması sonucu elde edilen matrisi ifade eder.
# Her bir sütun, bir ağırlık için hesaplanan türevi temsil eder.
# Örneğin, eğer giriş vektörü 3 özellik içeriyorsa ve delta vektörü de 2 çıkış özelliği içeriyorsa,
# elde edilen matris 3x2 boyutunda olacaktır. Bu matrisin elemanları, her bir ağırlığın ne kadar değişmesi gerektiğini belirtir.

self.bias += learning_rate * delta # Bias'ı güncelle
# Bias'ı güncelledim. Yine öğrenme oranı ile delta çarpılarak güncelleme yapıldı.

# Geriye yayılım hatası (bir önceki katmana gönderilecek hata)

backward_error = delta @ self.weights.T # Vektör çarpımı ile geriye yayılım hatasını hesapladım

```

```

main.py x
90
91 # Geriye yayılım hatası (bir önceki katmana gönderilecek hata)
92
93 backward_error = delta @ self.weights.T # Vektör çarpımı ile geriye yayılım hatasını hesapladım
94
95 #backward_error = delta @ self.weights.T satırı, bir önceki katmana gönderilecek geriye yayılım hatasını hesaplar.
96 # @ işlemi, matris çarpımını ifade eder. Delta, mevcut katmandaki hata vektörünü temsil eder. self.weights.T ise ağırlıkların transpozunu ifade eder.
97 #Bu matris çarpımı sonucunda elde edilen vektör, bir önceki katmandaki nöronlara geriye doğru iletilmesi gereken hata miktarını belirtir.
98 # Her bir eleman, bir önceki katmandaki bir nörona düşen hata miktarını ifade eder. Yani, bu vektör, ağırlıkların türevi ile çarpıldığı için,
99 # ağırlıkların katkısıyla bir önceki katmandaki hataların nasıl değişeceğini gösterir.
100
101 return backward_error

```

- activation\_derivative: Aktivasyon fonksiyonunun türevidir. Bu türev, aktivasyon fonksiyonunun çıkışına göre hesaplanır.
- delta: Geriye yayılım (backpropagation) hata sinyali. Bu, önceki katmandan gelen hata (error) ile aktivasyon türevi ve mevcut aktivasyonun çarpımıdır. Bu, gradient descent algoritması için bir güncelleme sinyali olarak kullanılacaktır.
- derivative\_weights ve derivative\_bias: Ağırlık ve bias için türev değerlerini hesaplar. Bu değerler, ağırlıkları ve bias'ı güncellerken kullanılacaktır.
- Ağırlıkları ve bias'ı günceller: Gradient descent algoritması kullanılarak ağırlıkları ve bias'ı günceller. Her bir ağırlık ve bias, önceki değeri ile öğrenme oranı ve türev değeri çarpılarak güncellenir.
- backward\_error: Geriye yayılım hatasıdır. Bu, ağırlıkların transpozu ile delta'nın çarpımıdır. Bu, bir önceki katmana gönderilecek hata miktarını temsil eder.

```

102 class Model:
103     def __init__(self):
104         self.layers = []
105
106     def add_layer(self, layer):
107         self.layers.append(layer)
108

```

- \_\_init\_\_ metodu: Sınıfın başlatıcı metodudur. Model oluşturulduğunda, boş bir katman listesi (self.layers) oluşturulur.
- add\_layer metodu: Bu metodun amacı, modele bir katman eklemektir. layer parametresi olarak bir Layer nesnesi alır ve bu katmanı modelin katman listesine ekler.

```

38
39 def forward(self, inputs):
40     for layer in self.layers:
41         inputs = layer.forward(inputs)
42     return inputs
43
44 def backward(self, error, learning_rate):
45     for layer in reversed(self.layers):
46         error = layer.backward(error, learning_rate)
47

```

- forward metodu: Bu metodun amacı, modele bir giriş vektörü (inputs) verildiğinde, sırasıyla tüm katmanlardan geçerek çıkışı hesaplamaktır. Bu işlem, her bir katmanın forward metodunu çağırarak gerçekleştirilir.
- backward metodu: Bu metodun amacı, bir hata (error) ve öğrenme oranı (learning\_rate) verildiğinde, geriye yayılım algoritması kullanarak modelin tüm katmanlarını güncellemektir. Bu işlem, her bir katmanın backward metodunu çağırarak gerçekleştirilir. reversed(self.layers) ifadesi, katmanları tersten (son katmandan başlayarak) dolaşmamıza olanak tanır, çünkü geriye yayılım işlemi genellikle ağıın sonundan başlayarak yapılır.

```

main.py x
usage
118 def get_user_input():
119     # Kullanıcıdan katman sayısını aldım.
120     num_layers = int(input("Kaç katman eklemek istiyorsunuz? "))
121     # Boş listeler oluşturdum.
122     layer_sizes = [] # Katman boyutları
123     activation_functions = [] # Aktivasyon fonksiyonları
124
125     # Katman sayısı kadar döngü yaptım.
126     for i in range(num_layers):
127         while True:
128             try:
129                 # Kullanıcıdan her bir katmanın boyutunu aldım.
130                 layer_size = int(input(f"{i + 1}. katman boyutunu girin: "))
131                 # Katman boyutu pozitif bir tam sayı olmalıdır.
132                 if layer_size <= 0:
133                     print("Katman boyutu pozitif bir tam sayı olmalıdır. Tekrar deneyin.")
134                     continue
135                 break
136             except ValueError:
137                 # Hata durumunda kullanıcıya bilgi verdim.

```

```

main.py x
137     # Hata durumunda kullanıcıya bilgi verdim.
138     print("Geçersiz bir değer girdiniz. Lütfen bir tam sayı girin.")
139
140     # Katman boyutunu listeye ekler.
141     layer_sizes.append(layer_size)
142
143     # Kullanıcıdan her bir katmanın aktivasyon fonksiyonunu aldım.
144     activation = input(f"{i + 1}. katman aktivasyon fonksiyonunu girin (sigmoid, relu veya tanh): ")
145
146     # Girilen aktivasyon fonksiyonuna göre uygun aktivasyon nesnesini oluşturdum ve listeye ekledim.
147     if activation.lower() == "sigmoid":
148         activation_functions.append(Sigmoid())
149     elif activation.lower() == "relu":
150         activation_functions.append(ReLU())
151     elif activation.lower() == "tanh":
152         activation_functions.append(Tanh())
153     else:
154         # Geçersiz bir aktivasyon fonksiyonu girilirse hata mesajı verdirip programı sonlandırdım.
155         print("Geçersiz aktivasyon fonksiyonu. Lütfen 'sigmoid', 'relu' veya 'tanh' kullanın.")
156         exit()
157

```

```
# Oluşturulan bilgileri döndürdüm.  
return layer_sizes, activation_functions
```

- ✚ Bu fonksiyon, kullanıcıdan sinir ağı modeli için katman sayısını, her bir katmanın boyutunu ve aktivasyon fonksiyonunu alarak, bu bilgileri kullanıcı girişine dayalı olarak toplar.
- ✚ Bu fonksiyonun amacı, kullanıcıdan sinir ağı modeli için gerekli bilgileri almak ve bu bilgileri döndürmektir. Kullanıcıdan alınan bilgiler daha sonra bir sinir ağı modeli oluşturmak için kullanılabilir.

```
main.py x  
160  
161 # Veri setini yükleme  
162 dosya_yolu = r"C:\Users\kizil\OneDrive\Belgeler\Uygulama_akciğer_kanseri\housing_price_dataset.csv"  
163 df = pd.read_csv(dosya_yolu)  
164 df = df.dropna(subset=['Price'])  
165 df['Price'] = pd.to_numeric(df['Price'], errors='coerce')  
166  
167 # Giriş ve çıkış sütunlarını seçme  
168 inputs = pd.concat([df[['SquareFeet', 'Bedrooms', 'Bathrooms', 'YearBuilt']], df.filter(regex='Neighborhood_*')], axis=1)  
169 outputs = df['Price'].values.reshape(-1, 1)  
170  
171 # Verileri ölçeklendirin  
172 scaler = StandardScaler()  
173 inputs_scaled = scaler.fit_transform(inputs)  
174 outputs_scaled = scaler.fit_transform(outputs)  
175  
176 # Kullanıcıdan sinir ağı yapısı hakkında bilgileri al  
177 layer_sizes, activation_functions = get_user_input()  
178
```

- ✚ dosya\_yolu: Veri setinin dosya yolu. r ön eki, bir raw string ifadesidir ve kaçış dizilerini yok sayar.
- ✚ pd.read\_csv(dosya\_yolu): Pandas kütüphanesini kullanarak CSV dosyasını okudum.
- ✚ df.dropna(subset=['Price']): 'Price' sütunundaki NaN değerlere sahip satırları çıkarttım.
- ✚ pd.to\_numeric(df['Price'], errors='coerce'): 'Price' sütununu sayısal veriye dönüştürme işlemi yaptım. Hatalı dönüşen değerleri NaN ile değiştirme yaptım.
- ✚ inputs: Giriş verilerini seçme. 'SquareFeet', 'Bedrooms', 'Bathrooms' ve 'YearBuilt' sütunları ile 'Neighborhood\_\*' regex deseni ile eşleşen sütunları birleştirme yaptım.
- ✚ outputs: Çıkış verilerini seçme. 'Price' sütununu kullanma ve reshape işlemi ile tek sütunlu bir veri çerçevesine dönüştürme işlemi yaptım.
- ✚ scaler = StandardScaler(): Verileri ölçeklendirmek için bir StandardScaler nesnesi oluşturdum.
- ✚ inputs\_scaled: Giriş verilerini ölçeklendirdim.
- ✚ outputs\_scaled: Çıkış verilerini ölçeklendirdim.
- ✚ get\_user\_input(): Kullanıcıdan sinir ağı modelinin yapısı hakkında bilgileri almak için önce tanımlanan get\_user\_input fonksiyonunu çağırma. Kullanıcıdan katman sayısını, her bir katmanın boyutunu ve aktivasyon fonksiyonunu alır.
- ✚ Bu bilgileri layer\_sizes ve activation\_functions değişkenlerine atar. Bu bilgiler, sinir ağı modelinin oluşturulmasında kullanılacaktır.

```

9 # Model oluşturma bölümü
0 model = Model()
1 input_size = inputs.shape[1]
2 for i in range(len(layer_sizes)):
3     layer = Layer(input_size, layer_sizes[i], activation_functions[i])
4     model.add_layer(layer)
5     input_size = layer_sizes[i]
6
7 # Eğitim bölümü
8 learning_rate = 0.001
9 batch_size = 100
0 num_epochs = 5
1
2 # Çapraz doğrulama için KFold nesnesini oluştur
3 kfold = KFold(n_splits=100, shuffle=True, random_state=42)
4

```

- ✚ `model = Model()`: Bir Model nesnesi oluşturulur. Bu nesne, sinir ağı modelini temsil eder.
- ✚ `input_size = inputs.shape[1]`: Giriş verisinin özellik sayısını alır.
- ✚ `for i in range(len(layer_sizes))`: Katman sayısına göre bir döngü oluşturulur.
- ✚ `layer = Layer(input_size, layer_sizes[i], activation_functions[i])`: Her bir iterasyonda, Layer sınıfından bir katman oluşturulur. Katmanın giriş boyutu `input_size`, çıkış boyutu `layer_sizes[i]`, aktivasyon fonksiyonu ise `activation_functions[i]` olarak belirlenir.
- ✚ `model.add_layer(layer)`: Oluşturulan katman, model nesnesine eklenir.
- ✚ `input_size = layer_sizes[i]`: Bir sonraki iterasyonda kullanılmak üzere giriş boyutu, eklenen katmanın çıkış boyutuna ayarlanır.
- ✚ `learning_rate`: Öğrenme oranı, gradient descent algoritmasındaki adım büyüklüğünü belirleyen bir hiperparametredir.
- ✚ `batch_size`: Her bir eğitim iterasyonunda kullanılacak örnek sayısını belirler. Minibatch gradient descent kullanılacaksa, bu sayı küçük bir değer olabilir.
- ✚ `num_epochs`: Eğitim sürecinde kaç kez tüm veri setinin kullanılacağını belirleyen bir hiperparametredir.
- ✚ `kfold = KFold(n_splits=100, shuffle=True, random_state=42)`: K-fold çapraz doğrulama için KFold nesnesi oluşturulur. Bu, veri setini belirtilen sayıda parçaya böler ve her bir parçayı test seti olarak kullanır. `n_splits` parametresi, kaç parçaya bölüneceğini belirler. `shuffle=True` ise her bölmeyi oluşturmadan önce veriyi karıştırır. `random_state=42` ise tekrarlanabilirliği sağlamak için kullanılan bir rastgele başlatma değeridir.



```

195 # Çapraz doğrulama işlemi
196 all_errors = [] #Tüm k-fold iterasyonlarının hata değerlerini tutmak için bir liste
197 fold_errors_list = [] #Her bir k-fold iterasyonunun hata değerlerini tutmak için bir liste
198
199 fold = 1 # Kaçınıcı k-fold iterasyonu olduğunu takip etmek için bir değişken
200
201 for train_index, test_index in kfold.split(inputs_scaled):
202     print(f"Fold-{fold}")
203
204     # Eğitim ve test setlerini oluştur
205     X_train, X_test = inputs_scaled[train_index], inputs_scaled[test_index]
206     y_train, y_test = outputs_scaled[train_index], outputs_scaled[test_index]
207
208     # Yeni bir Model nesnesi oluştur
209     model = Model()
210
211     # Her bir k-fold iterasyonu için hata değerlerini tutacak bir liste
212     fold_errors = []
213

```

```

main.py x
214 # Modelin eğitimi
215 for epoch in range(num_epochs):
216     fold_epoch_errors = []
217
218     for _ in range(0, len(X_train), batch_size):
219         # Mini-batch oluşturdum.
220         input_batch = X_train[_:_ + batch_size]
221         output_batch = y_train[_:_ + batch_size]
222
223         # İleri geçiş (forward pass)
224         layer_inputs = input_batch
225         for layer in model.layers:
226             layer_inputs = layer.forward(layer_inputs)
227         output = layer_inputs
228
229         # Hata hesapladım.
230         error = output_batch - output
231
232         # Geriye yayılım (backward pass)
233         for i in reversed(range(len(model.layers))):
234             if i == len(model.layers) - 1:

```

```

main.py x
232 # Geriye yayılım (backward pass)
233 for i in reversed(range(len(model.layers))):
234     if i == len(model.layers) - 1:
235         layer_error = error
236     else:
237         layer_error = np.dot(layer_error, model.layers[i + 1].weights.T) * model.layers[
238             i + 1].activation.derivative(model.layers[i + 1].output)
239         model.layers[i].backward(layer_error, learning_rate)
240
241     # Hatanın karesinin ortalamasını al ve listeye ekledim.
242     epoch_error = np.mean(np.square(error))
243     fold_epoch_errors.append(epoch_error)
244
245     # Bu epoch'un hata değerini listeye ekledim.
246     fold_errors.append(np.mean(fold_epoch_errors))
247     # Tüm hata değerlerini genel liste olan all_errors'a ekledim.
248     all_errors.extend(fold_epoch_errors)
249
250     # Her epoch sonunda o epoch'un hata değerini yazdırdım.
251     print(f"Epoch-{epoch + 1} MSE: {np.mean(fold_epoch_errors)}")
252

```

```

# Her epoch sonunda o epoch'un hata değerini yazdırdım.
print(f"Epoch-{epoch + 1} MSE: {np.mean(fold_epoch_errors)}")

fold_errors_list.append(fold_epoch_errors)
print(f"Fold-{fold} MSE: {np.mean(fold_epoch_errors)}\n")

fold += 1

```

- ✚ Bu kod, çapraz doğrulama işlemini gerçekleştirir.
- ✚ Dıştaki döngü, her bir k-fold iterasyonunu temsil eder. Her bir iterasyonda, model yeniden oluşturulur ve eğitim/test setleri belirlenir.
- ✚ İçteki döngü, her bir epoch için eğitim işlemini gerçekleştirir. Her epoch'ta, mini-batch'ler kullanılarak ileri geçiş, hata hesaplama ve geriye yayılım işlemleri yapılır. Epoch'un sonunda elde edilen hata değeri listeye eklenir.
- ✚ Bu işlemlerin sonucunda, her bir k-fold iterasyonu için hem eğitim setinde hem de test setinde elde edilen hata değerleri ayrı ayrı listelere eklenir. Bu listeler `all_errors` ve `fold_errors_list` olarak adlandırılmıştır.

```

# Grafikler
plt.figure(figsize=(10, 6))
for i, fold_errors in enumerate(fold_errors_list):
    # Her bir k-fold iterasyonu için bir çizgi çizdim.
    plt.plot(*args: np.arange(len(fold_errors)) + 1 + i * num_epochs, fold_errors, label=f'Fold {i + 1}', alpha=0.7)

plt.xlabel('Epoch') # x eksen etiketi
plt.ylabel('Hata Değeri (MSE)') # y eksen etiketi
plt.title('Çapraz Doğrulama Foldlarına Göre Eğitim Sırasında Hata Değeri Değişimi (MSE)') # Grafiğin başlığı
plt.legend() # Legend (çizgi açıklamaları)
plt.grid(True) # Izgara eklenmiş mi kontrolü
plt.show() # Grafiği göster

```

- ✚ `plt.figure(figsize=(10, 6))`: Yeni bir matplotlib figürü oluşturur ve boyutunu belirler.
- ✚ `for i, fold_errors in enumerate(fold_errors_list):`: Her bir k-fold iterasyonu için bir döngü başlatılır ve bu iterasyonların hata değerleri üzerinde gezilir.
- ✚ `np.arange(len(fold_errors)) + 1 + i * num_epochs`: x-ekseni değerleri, epoch sayısını temsil eder. Her bir k-fold iterasyonu için x değerleri bu iterasyonun epoch sayısına göre artar.
- ✚ `fold_errors`: Y-ekseni değerleri, her bir epoch'taki hata değerlerini temsil eder.
- ✚ `label=f'Fold {i + 1}'`: Legend'da gösterilecek etiket, k-fold iterasyonunu temsil eder.
- ✚ `alpha=0.7`: Çizgilerin saydamlığı belirlenir.
- ✚ `plt.xlabel('Epoch')`: x eksen etiketi eklenir.
- ✚ `plt.ylabel('Hata Değeri (MSE)')`: y eksen etiketi eklenir.
- ✚ `plt.title('Çapraz Doğrulama Foldlarına Göre Eğitim Sırasında Hata Değeri Değişimi (MSE)')`: Grafiğin başlığı belirlenir.
- ✚ `plt.legend()`: Legend, yani çizgi açıklamaları eklenir. Bu, her bir çizginin neyi temsil ettiğini gösterir.
- ✚ `plt.grid(True)`: Grafiğe ızgara eklenip eklenmediği kontrol edilir. True durumunda ızgara eklenir.
- ✚ `plt.show()`: Grafiği görüntüler.

- ✚ Bu kod bloğu, çapraz doğrulama sırasında her bir k-fold iterasyonunun eğitim sürecinde elde ettiği hata değerlerini karşılaştıran bir çizgi grafiği oluşturur. Her bir çizgi, bir k-fold iterasyonunu temsil eder ve x eksenini epoch sayısını, y eksenini ise hata değerlerini gösterir. Grafiği incelediğinizde, eğitim sürecinin her kısmının hata değerlerindeki değişiklikleri görebilir ve modelin performansını değerlendirebilirsiniz.

```
273 # Toplam hata değerlerini gösteren bir grafik
274 plt.figure(figsize=(10, 6))
275 plt.plot(*args: all_errors, label='Toplam Hata', color='orange')
276 plt.xlabel('Epoch')
277 plt.ylabel('Hata Değeri (MSE)')
278 plt.title('Eğitim Sırasında Toplam Hata Değeri Değişimi (MSE)')
279 plt.legend()
280 plt.grid(True)
281 plt.show()
282
283
284
```

- ✚ `plt.figure(figsize=(10, 6))`: Yeni bir matplotlib figürü oluşturur ve boyutunu belirler.
- ✚ `plt.plot(all_errors, label='Toplam Hata', color='orange')`: Grafik üzerine toplam hata değerlerini temsil eden bir çizgi eklenir. `color='orange'` ifadesi, çizginin rengini turuncu olarak belirler.
- ✚ `plt.xlabel('Epoch')`: x eksenini etiketi eklenir.
- ✚ `plt.ylabel('Hata Değeri (MSE)')`: y eksenini etiketi eklenir.
- ✚ `plt.title('Eğitim Sırasında Toplam Hata Değeri Değişimi (MSE)')`: Grafiğin başlığı belirlenir.
- ✚ `plt.legend()`: Legend, yani çizgi açıklamaları eklenir. Bu durumda sadece bir çizgi olduğu için legend'a gerek yoktur, ancak genellikle birden fazla çizgi olduğunda kullanılır.
- ✚ `plt.grid(True)`: Grafiğe ızgara eklenip eklenmediği kontrol edilir. True durumunda ızgara eklenir.
- ✚ `plt.show()`: Grafiği görüntüler.
- ✚ Bu grafik, eğitim süreci boyunca toplam hata değerinin nasıl değiştiğini gösterir. Epoch sayısına bağlı olarak hata değerlerindeki dalgalanmaları ve genel bir düşüş eğilimini değerlendirmek için kullanılabilir.

## Özet:

**Aktivasyon Fonksiyonları:** Activation sınıfından türetilmiş olan Sigmoid, ReLU ve Tanh aktivasyon fonksiyonları tanımlanmıştır. Bu fonksiyonlar, sinir ağı katmanlarında kullanılmak üzere ileri geçiş ve geriye yayılım işlemlerini yönetir.

**Katman Sınıfı:** Layer sınıfı, sinir ağı katmanının temel özelliklerini ve işlevlerini tanımlar. Her bir katmanın ağırlıkları, bias'ı, aktivasyon fonksiyonu, giriş ve çıkışları bu sınıf içinde bulunur.

**Model Sınıfı:** Model sınıfı, sinir ağı modelini oluşturan katmanları ve bu katmanları yöneten işlemleri içerir. Katmanlar eklemek, ileri geçiş ve geriye yayılım işlemleri bu sınıf içinde gerçekleşir.

**Kullanıcıdan Giriş Almak İçin Fonksiyon:** `get_user_input` fonksiyonu, kullanıcıdan sinir ağı yapısı hakkında bilgileri (katman sayısı, her katmanın boyutu ve aktivasyon fonksiyonları) alır.

**Veri Setini Yükleme ve Ön İşleme:** `pandas` kütüphanesi kullanılarak bir CSV dosyasından veri seti yüklenir. Giriş ve çıkış sütunları belirlenir ve veriler ölçeklendirilir.

**Modelin Oluşturulması ve Eğitimi:** Model sınıfı kullanılarak sinir ağı modeli oluşturulur. Ardından, çapraz doğrulama ile model eğitilir. Her epoch'ta hata değerleri kaydedilir.

**Grafiklerle Sonuçların Görselleştirilmesi:** Çapraz doğrulama sonuçları ve toplam hata değerleri `matplotlib` kullanılarak grafiğe dökülür. İki farklı grafik oluşturulur: biri çapraz doğrulama foldlarına göre eğitim sırasında hata değeri değişimi, diğeri ise eğitim sürecindeki toplam hata değeri değişimidir.

Bu kod, bir sinir ağı modelinin temelini oluşturur ve eğitir. Çapraz doğrulama ile modelin performansını değerlendirir ve eğitim sırasında hata değişimini görselleştirir.

Ben yazmış olduğum kodda ilk olarak Sigmoid ve Relu kullandım. Daha sonra Tanh fonksiyonu kullandım.

MSE değerini; batch, fold, epoch değerlerini değiştirerek gözlemledim. B

Örneğin; batch değerim 25, fold değerim 30 iken aşağıdaki değerleri buldum.

#### **TERMINAL-1**

```
C:\Users\kizil\PycharmProjects\ysa\venv\Scripts\python.exe
C:\Users\kizil\PycharmProjects\ysa\main.py
```

Kaç katman eklemek istiyorsunuz? 5

1. katman boyutunu girin: 1

1. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): relu

2. katman boyutunu girin: 2

2. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): sigmoid

3. katman boyutunu girin: 2

3. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): relu

4. katman boyutunu girin: 1

4. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): relu

5. katman boyutunu girin: 2

5. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): sigmoid

Fold-1

Epoch-1 MSE: 1.5740165390922192

Epoch-2 MSE: 1.5740165390922192

Epoch-3 MSE: 1.5740165390922192

Epoch-4 MSE: 1.5740165390922192

Epoch-5 MSE: 1.5740165390922192

Fold-1 MSE: 1.5740165390922192

Fold-2

Epoch-1 MSE: 1.5747113474080912

Epoch-2 MSE: 1.5747113474080912

Epoch-3 MSE: 1.5747113474080912

Epoch-4 MSE: 1.5747113474080912

Epoch-5 MSE: 1.5747113474080912

Fold-2 MSE: 1.5747113474080912

Fold-3

Epoch-1 MSE: 1.575392925466499

Epoch-2 MSE: 1.575392925466499

Epoch-3 MSE: 1.575392925466499

Epoch-4 MSE: 1.575392925466499

Epoch-5 MSE: 1.575392925466499

Fold-3 MSE: 1.575392925466499

Fold-4

Epoch-1 MSE: 1.5772532760988731

Epoch-2 MSE: 1.5772532760988731

Epoch-3 MSE: 1.5772532760988731

Epoch-4 MSE: 1.5772532760988731

Epoch-5 MSE: 1.5772532760988731

Fold-4 MSE: 1.5772532760988731

Fold-5

Epoch-1 MSE: 1.57763060976049

Epoch-2 MSE: 1.57763060976049

Epoch-3 MSE: 1.57763060976049

Epoch-4 MSE: 1.57763060976049

Epoch-5 MSE: 1.57763060976049

Fold-5 MSE: 1.57763060976049

Fold-6

Epoch-1 MSE: 1.5747040795807419

Epoch-2 MSE: 1.5747040795807419

Epoch-3 MSE: 1.5747040795807419

Epoch-4 MSE: 1.5747040795807419

Epoch-5 MSE: 1.5747040795807419

Fold-6 MSE: 1.5747040795807419

Fold-7

Epoch-1 MSE: 1.5752120452004132

Epoch-2 MSE: 1.5752120452004132

Epoch-3 MSE: 1.5752120452004132

Epoch-4 MSE: 1.5752120452004132

Epoch-5 MSE: 1.5752120452004132

Fold-7 MSE: 1.5752120452004132

Fold-8

Epoch-1 MSE: 1.5759047783882922

Epoch-2 MSE: 1.5759047783882922

Epoch-3 MSE: 1.5759047783882922

Epoch-4 MSE: 1.5759047783882922

Epoch-5 MSE: 1.5759047783882922

Fold-8 MSE: 1.5759047783882922

Fold-9

Epoch-1 MSE: 1.5763744616227124

Epoch-2 MSE: 1.5763744616227124

Epoch-3 MSE: 1.5763744616227124

Epoch-4 MSE: 1.5763744616227124

Epoch-5 MSE: 1.5763744616227124

Fold-9 MSE: 1.5763744616227124

#### Fold-10

Epoch-1 MSE: 1.5753149090451173

Epoch-2 MSE: 1.5753149090451173

Epoch-3 MSE: 1.5753149090451173

Epoch-4 MSE: 1.5753149090451173

Epoch-5 MSE: 1.5753149090451173

Fold-10 MSE: 1.5753149090451173

#### Fold-11

Epoch-1 MSE: 1.577625007050419

Epoch-2 MSE: 1.577625007050419

Epoch-3 MSE: 1.577625007050419

Epoch-4 MSE: 1.577625007050419

Epoch-5 MSE: 1.577625007050419

Fold-11 MSE: 1.577625007050419

#### Fold-12

Epoch-1 MSE: 1.575144309224201

Epoch-2 MSE: 1.575144309224201

Epoch-3 MSE: 1.575144309224201

Epoch-4 MSE: 1.575144309224201

Epoch-5 MSE: 1.575144309224201

Fold-12 MSE: 1.575144309224201

#### Fold-13

Epoch-1 MSE: 1.576354288267861

Epoch-2 MSE: 1.576354288267861

Epoch-3 MSE: 1.576354288267861

Epoch-4 MSE: 1.576354288267861

Epoch-5 MSE: 1.576354288267861

Fold-13 MSE: 1.576354288267861



#### Fold-14

Epoch-1 MSE: 1.5750309252239088

Epoch-2 MSE: 1.5750309252239088

Epoch-3 MSE: 1.5750309252239088

Epoch-4 MSE: 1.5750309252239088

Epoch-5 MSE: 1.5750309252239088

Fold-14 MSE: 1.5750309252239088

#### Fold-15

Epoch-1 MSE: 1.5751370636232345

Epoch-2 MSE: 1.5751370636232345

Epoch-3 MSE: 1.5751370636232345

Epoch-4 MSE: 1.5751370636232345

Epoch-5 MSE: 1.5751370636232345

Fold-15 MSE: 1.5751370636232345

#### Fold-16

Epoch-1 MSE: 1.5761854625377554

Epoch-2 MSE: 1.5761854625377554

Epoch-3 MSE: 1.5761854625377554

Epoch-4 MSE: 1.5761854625377554

Epoch-5 MSE: 1.5761854625377554

Fold-16 MSE: 1.5761854625377554

#### Fold-17

Epoch-1 MSE: 1.577678000688969

Epoch-2 MSE: 1.577678000688969

Epoch-3 MSE: 1.577678000688969

Epoch-4 MSE: 1.577678000688969

Epoch-5 MSE: 1.577678000688969

Fold-17 MSE: 1.577678000688969

#### Fold-18

Epoch-1 MSE: 1.5754889513976063

Epoch-2 MSE: 1.5754889513976063

Epoch-3 MSE: 1.5754889513976063

Epoch-4 MSE: 1.5754889513976063

Epoch-5 MSE: 1.5754889513976063

Fold-18 MSE: 1.5754889513976063

#### Fold-19

Epoch-1 MSE: 1.5769490378095175

Epoch-2 MSE: 1.5769490378095175

Epoch-3 MSE: 1.5769490378095175

Epoch-4 MSE: 1.5769490378095175

Epoch-5 MSE: 1.5769490378095175

Fold-19 MSE: 1.5769490378095175

#### Fold-20

Epoch-1 MSE: 1.5739505124194695

Epoch-2 MSE: 1.5739505124194695

Epoch-3 MSE: 1.5739505124194695

Epoch-4 MSE: 1.5739505124194695

Epoch-5 MSE: 1.5739505124194695

Fold-20 MSE: 1.5739505124194695

#### Fold-21

Epoch-1 MSE: 1.5762325265652246

Epoch-2 MSE: 1.5762325265652246

Epoch-3 MSE: 1.5762325265652246

Epoch-4 MSE: 1.5762325265652246

Epoch-5 MSE: 1.5762325265652246

Fold-21 MSE: 1.5762325265652246

#### Fold-22

Epoch-1 MSE: 1.5758981060634933

Epoch-2 MSE: 1.5758981060634933

Epoch-3 MSE: 1.5758981060634933

Epoch-4 MSE: 1.5758981060634933

Epoch-5 MSE: 1.5758981060634933

Fold-22 MSE: 1.5758981060634933

#### Fold-23

Epoch-1 MSE: 1.5742780073611928

Epoch-2 MSE: 1.5742780073611928

Epoch-3 MSE: 1.5742780073611928

Epoch-4 MSE: 1.5742780073611928

Epoch-5 MSE: 1.5742780073611928

Fold-23 MSE: 1.5742780073611928

#### Fold-24

Epoch-1 MSE: 1.5755369102011816

Epoch-2 MSE: 1.5755369102011816

Epoch-3 MSE: 1.5755369102011816

Epoch-4 MSE: 1.5755369102011816

Epoch-5 MSE: 1.5755369102011816

Fold-24 MSE: 1.5755369102011816

#### Fold-25

Epoch-1 MSE: 1.5730694744282603

Epoch-2 MSE: 1.5730694744282603

Epoch-3 MSE: 1.5730694744282603

Epoch-4 MSE: 1.5730694744282603

Epoch-5 MSE: 1.5730694744282603

Fold-25 MSE: 1.5730694744282603

Fold-26

Epoch-1 MSE: 1.5757858796874495

Epoch-2 MSE: 1.5757858796874495

Epoch-3 MSE: 1.5757858796874495

Epoch-4 MSE: 1.5757858796874495

Epoch-5 MSE: 1.5757858796874495

Fold-26 MSE: 1.5757858796874495

Fold-27

Epoch-1 MSE: 1.576328175068352

Epoch-2 MSE: 1.576328175068352

Epoch-3 MSE: 1.576328175068352

Epoch-4 MSE: 1.576328175068352

Epoch-5 MSE: 1.576328175068352

Fold-27 MSE: 1.576328175068352

Fold-28

Epoch-1 MSE: 1.5770860300532161

Epoch-2 MSE: 1.5770860300532161

Epoch-3 MSE: 1.5770860300532161

Epoch-4 MSE: 1.5770860300532161

Epoch-5 MSE: 1.5770860300532161

Fold-28 MSE: 1.5770860300532161

Fold-29

Epoch-1 MSE: 1.5748595612375662

Epoch-2 MSE: 1.5748595612375662

Epoch-3 MSE: 1.5748595612375662

Epoch-4 MSE: 1.5748595612375662

Epoch-5 MSE: 1.5748595612375662

Fold-29 MSE: 1.5748595612375662

Fold-30

Epoch-1 MSE: 1.575828077456027

Epoch-2 MSE: 1.575828077456027

Epoch-3 MSE: 1.575828077456027

Epoch-4 MSE: 1.575828077456027

Epoch-5 MSE: 1.575828077456027

Fold-30 MSE: 1.575828077456027

Process finished with exit code 0

Şimdi ise;batch değerim 25,fold değerim 30 iken aşağıdaki değerleri buldum.

### TERMINAL-2

```
C:\Users\kizil\PycharmProjects\ysa\venv\Scripts\python.exe
```

```
C:\Users\kizil\PycharmProjects\ysa\main.py
```

Kaç katman eklemek istiyorsunuz? 2

1. katman boyutunu girin: 1

1. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): relu

2. katman boyutunu girin: 3

2. katman aktivasyon fonksiyonunu girin (sigmoid veya relu): relu

Fold-1

Epoch-1 MSE: 1.5740829989778327

Epoch-2 MSE: 1.5740829989778327

Epoch-3 MSE: 1.5740829989778327

Epoch-4 MSE: 1.5740829989778327

Epoch-5 MSE: 1.5740829989778327

Fold-1 MSE: 1.5740829989778327

#### Fold-2

Epoch-1 MSE: 1.5738514082863346

Epoch-2 MSE: 1.5738514082863346

Epoch-3 MSE: 1.5738514082863346

Epoch-4 MSE: 1.5738514082863346

Epoch-5 MSE: 1.5738514082863346

Fold-2 MSE: 1.5738514082863346

#### Fold-3

Epoch-1 MSE: 1.5781786578127759

Epoch-2 MSE: 1.5781786578127759

Epoch-3 MSE: 1.5781786578127759

Epoch-4 MSE: 1.5781786578127759

Epoch-5 MSE: 1.5781786578127759

Fold-3 MSE: 1.5781786578127759

#### Fold-4

Epoch-1 MSE: 1.5750779922527254

Epoch-2 MSE: 1.5750779922527254

Epoch-3 MSE: 1.5750779922527254

Epoch-4 MSE: 1.5750779922527254

Epoch-5 MSE: 1.5750779922527254

Fold-4 MSE: 1.5750779922527254

#### Fold-5

Epoch-1 MSE: 1.5749253107272612

Epoch-2 MSE: 1.5749253107272612

Epoch-3 MSE: 1.5749253107272612

Epoch-4 MSE: 1.5749253107272612

Epoch-5 MSE: 1.5749253107272612

Fold-5 MSE: 1.5749253107272612

#### Fold-6

Epoch-1 MSE: 1.57629382045419

Epoch-2 MSE: 1.57629382045419

Epoch-3 MSE: 1.57629382045419

Epoch-4 MSE: 1.57629382045419

Epoch-5 MSE: 1.57629382045419

Fold-6 MSE: 1.57629382045419

#### Fold-7

Epoch-1 MSE: 1.5760850574572518

Epoch-2 MSE: 1.5760850574572518

Epoch-3 MSE: 1.5760850574572518

Epoch-4 MSE: 1.5760850574572518

Epoch-5 MSE: 1.5760850574572518

Fold-7 MSE: 1.5760850574572518

#### Fold-8

Epoch-1 MSE: 1.57581157736065

Epoch-2 MSE: 1.57581157736065

Epoch-3 MSE: 1.57581157736065

Epoch-4 MSE: 1.57581157736065

Epoch-5 MSE: 1.57581157736065

Fold-8 MSE: 1.57581157736065

#### Fold-9

Epoch-1 MSE: 1.5747757603184838

Epoch-2 MSE: 1.5747757603184838

Epoch-3 MSE: 1.5747757603184838

Epoch-4 MSE: 1.5747757603184838

Epoch-5 MSE: 1.5747757603184838

Fold-9 MSE: 1.5747757603184838

#### Fold-10

Epoch-1 MSE: 1.5755194036100915

Epoch-2 MSE: 1.5755194036100915

Epoch-3 MSE: 1.5755194036100915

Epoch-4 MSE: 1.5755194036100915

Epoch-5 MSE: 1.5755194036100915

Fold-10 MSE: 1.5755194036100915

#### Fold-11

Epoch-1 MSE: 1.577398896429366

Epoch-2 MSE: 1.577398896429366

Epoch-3 MSE: 1.577398896429366

Epoch-4 MSE: 1.577398896429366

Epoch-5 MSE: 1.577398896429366

Fold-11 MSE: 1.577398896429366

#### Fold-12

Epoch-1 MSE: 1.5759199791144944

Epoch-2 MSE: 1.5759199791144944

Epoch-3 MSE: 1.5759199791144944

Epoch-4 MSE: 1.5759199791144944

Epoch-5 MSE: 1.5759199791144944

Fold-12 MSE: 1.5759199791144944

#### Fold-13

Epoch-1 MSE: 1.5753155498687434

Epoch-2 MSE: 1.5753155498687434

Epoch-3 MSE: 1.5753155498687434

Epoch-4 MSE: 1.5753155498687434

Epoch-5 MSE: 1.5753155498687434

Fold-13 MSE: 1.5753155498687434



#### Fold-14

Epoch-1 MSE: 1.5756741436602404

Epoch-2 MSE: 1.5756741436602404

Epoch-3 MSE: 1.5756741436602404

Epoch-4 MSE: 1.5756741436602404

Epoch-5 MSE: 1.5756741436602404

Fold-14 MSE: 1.5756741436602404

#### Fold-15

Epoch-1 MSE: 1.5744013632889347

Epoch-2 MSE: 1.5744013632889347

Epoch-3 MSE: 1.5744013632889347

Epoch-4 MSE: 1.5744013632889347

Epoch-5 MSE: 1.5744013632889347

Fold-15 MSE: 1.5744013632889347

#### Fold-16

Epoch-1 MSE: 1.5753880235024373

Epoch-2 MSE: 1.5753880235024373

Epoch-3 MSE: 1.5753880235024373

Epoch-4 MSE: 1.5753880235024373

Epoch-5 MSE: 1.5753880235024373

Fold-16 MSE: 1.5753880235024373

#### Fold-17

Epoch-1 MSE: 1.5730748811850985

Epoch-2 MSE: 1.5730748811850985

Epoch-3 MSE: 1.5730748811850985

Epoch-4 MSE: 1.5730748811850985

Epoch-5 MSE: 1.5730748811850985

Fold-17 MSE: 1.5730748811850985

Fold-18

Epoch-1 MSE: 1.5759741612590958

Epoch-2 MSE: 1.5759741612590958

Epoch-3 MSE: 1.5759741612590958

Epoch-4 MSE: 1.5759741612590958

Epoch-5 MSE: 1.5759741612590958

Fold-18 MSE: 1.5759741612590958

Fold-19

Epoch-1 MSE: 1.5774315751412287

Epoch-2 MSE: 1.5774315751412287

Epoch-3 MSE: 1.5774315751412287

Epoch-4 MSE: 1.5774315751412287

Epoch-5 MSE: 1.5774315751412287

Fold-19 MSE: 1.5774315751412287

Fold-20

Epoch-1 MSE: 1.574417042522723

Epoch-2 MSE: 1.574417042522723

Epoch-3 MSE: 1.574417042522723

Epoch-4 MSE: 1.574417042522723

Epoch-5 MSE: 1.574417042522723

Fold-20 MSE: 1.574417042522723

Process finished with exit code 0

Figure 1

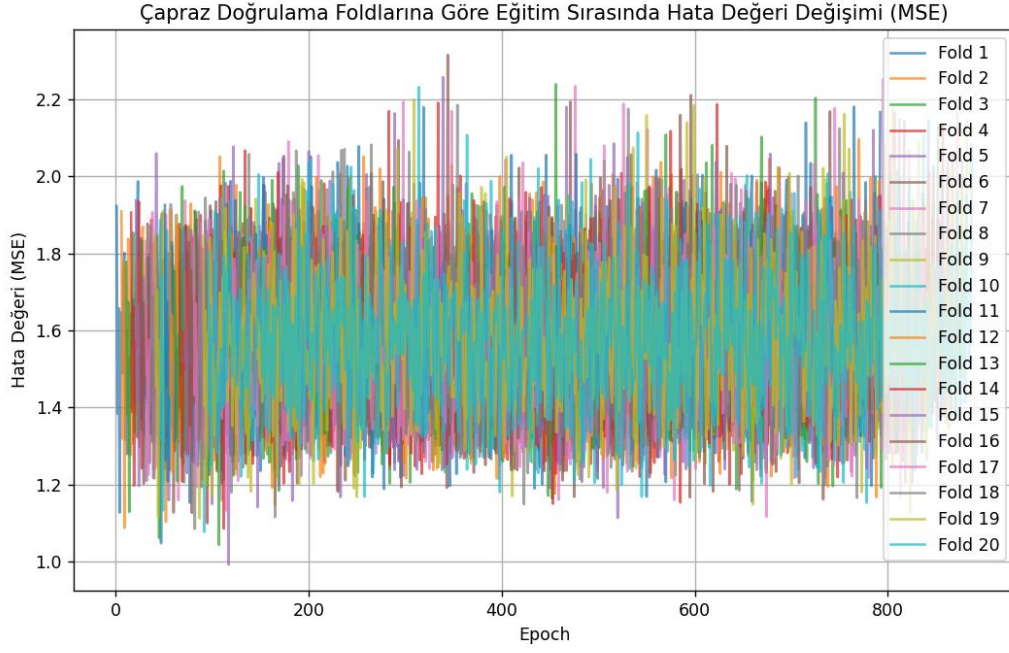
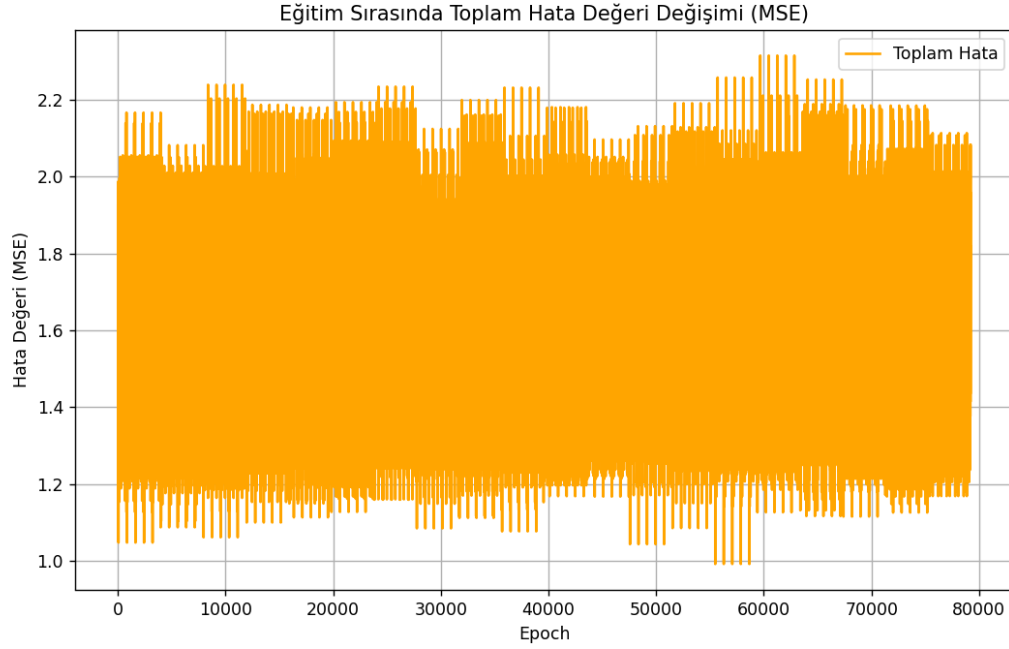
**ŞEKİL:TERMİNAL-2 ÖRNEĞİNİN GRAFİĞİ**

Figure 1

**ŞEKİL:TERMİNAL-2 ÖRNEĞİNİN TOPLAM HATA DEĞERİ DEĞİŞİMİNİN GRAFİĞİ**