



Kardiyovasküler Hastalık Tahmini için Nöron Ağları

- **Eren ÇALBAY** - 200202006
- **Ömer Emircan AYVAZ** - 200202009
- **Mehmet Ali AKDOĞAN** - 200202017
- **Eren SEZER** - 200202025
- **Ömer Faruk ULUSOY** - 200202032
- **Zeynep Aslı ERHAN** - 200202047
- **Muhammed Fatih ÖZEN** - 200202050
- **Muhammet Eren GÜR** - 200202082



İçindekiler

- Giriş
- Problemin İfadesi
- Kullanılan Kütüphaneler
- Nöron Ağları (NA)
 - Nöron Ağları Yapısı
- Veri Seti
 - Veri Seti Özellikleri
 - Veri Önisleme
- Model Yapısı
- Aktivasyon Fonksiyonları
- Kayıp Fonksiyonu
- Model Eğitimi
- Sonuç

Giriş

- Yapay zeka ve makine öğrenimi alanlarında, kardiyovasküler hastalık tahmini son yıllarda büyük önem kazanmıştır.
- Bu çalışma, nöron ağlarının (NA) kardiyovasküler hastalık tahminindeki potansiyelini ele alacak ve bu alandaki bir projenin ayrıntılarını sunmayı amaçlamaktadır.





Problemin ifadesi

Günümüzde kardiyovasküler hastalıklar dünya çapında önemli bir sağlık sorunu haline gelmiştir. Bu hastalıkların erken teşhisi ve etkili yönetimi, hem bireylerin sağlığı hem de sağlık sistemlerinin sürdürülebilirliği açısından hayatı önem taşımaktadır.

Amaçlar

- Kan tahlili ve bazı fiziksel özelliklerden elde edilen verilerin kullanılarak kardiyovasküler hastalıkların tespit edilmesi
- Hastaların risk seviyelerini belirleyerek erken müdahale imkanı sağlamak ve kaynakları daha etkin bir şekilde yönlendirmektir

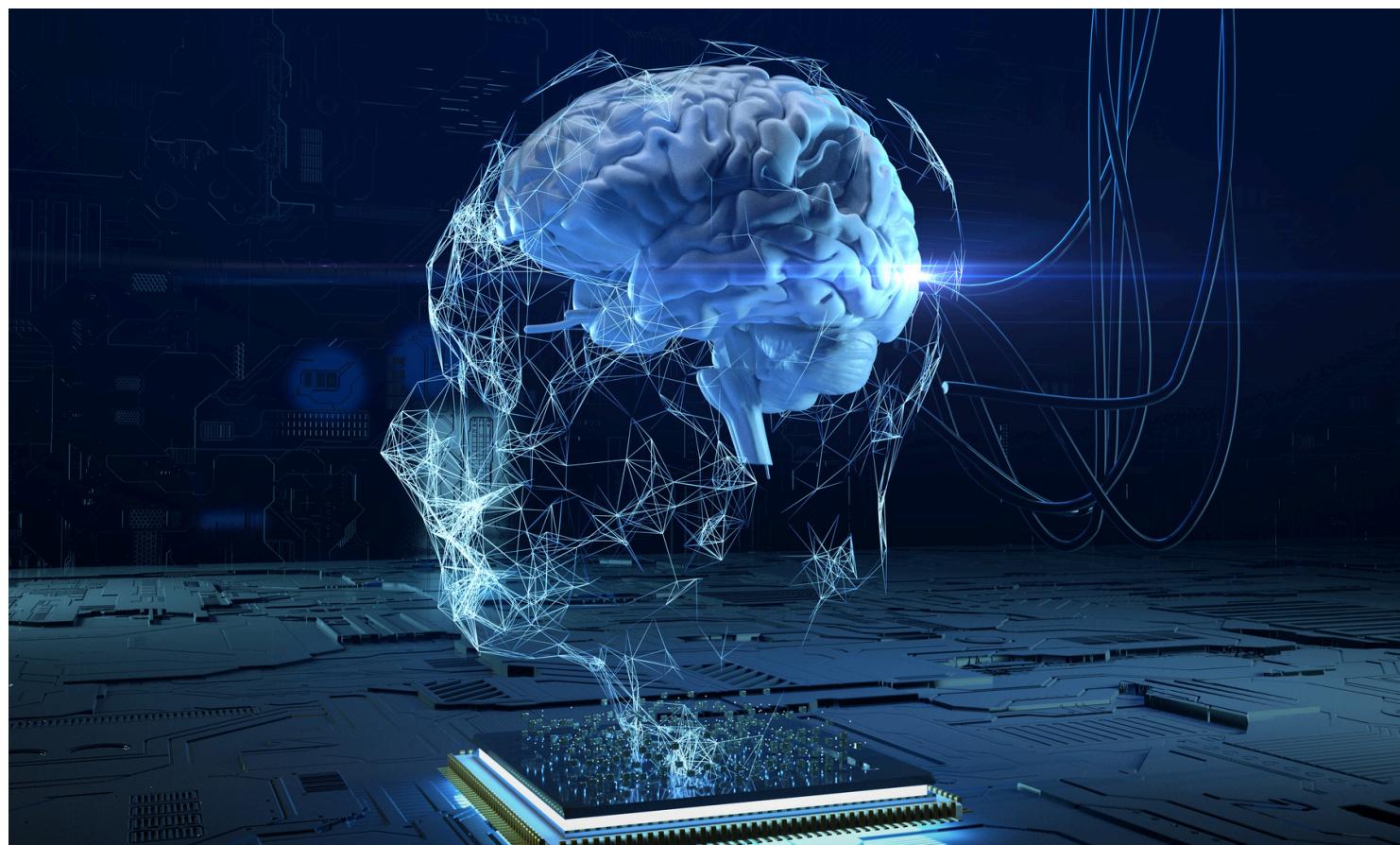


Kullanılan Kütüphaneler

Kütüphaneler

- **Pandas:** Veri setlerini okuma, yazma, temizleme, dönüştürme ve analiz etme işlemleri için kullanılır.
- **NumPy:** Diziler, matrisler ve çok boyutlu veri yapıları oluşturmak ve işlemek için kullanılır. Vektör ve matris operasyonları, rastgele sayı üretimi, lineer cebir işlemleri gibi birçok matematiksel işlemi destekler.
- **Seaborn:** Matplotlib üzerine inşa edilmiş ve daha yüksek seviyeli veri görselleştirme arayüzü sağlar.
- **Matplotlib.pyplot:** Veri görselleştirme için kullanılır.
- **Scikit-learn (sklearn):** Sınıflandırma, regresyon, kümeleme, boyut azaltma ve model seçimi gibi birçok makine öğrenimi algoritması ve aracı içerir. Model eğitimi, doğrulama ve tahmin için kullanılan araçlar sağlar.

Nöron Ağları



Nöron ağları, temel olarak nöron adı verilen birimlerden ve bu nöronların birbirleriyle bağlantılarından oluşur

Eğitimmiş bir NA modeli, çeşitli klinik verileri analiz ederek kardiyovasküler hastalık riskini tahmin edebilir. Projemizde, kardiyovasküler hastalık tahminini gerçekleştirmek için NA'lar kullanılacak ve modelin etkinliği inceleneciktir.





Korelasyon
kısımında hangi
özelliklerin neden
kullanıldığı
anlatılacaktır.

Veri Seti Özellikleri

AKSHAT ve arkadaşlarının hazırlamış olduğu yaklaşık 70 bin kişi için 14 fieldden oluşan veri seti kullanılmıştır.

Yaş (Age)

- Kişinin yaşıını belirtir (gün cinsinden tam sayı).

Boy (Height)

- Kişinin boyunu belirtir (cm cinsinden tam sayı).

Kilo (Weight)

- Kişinin kilosunu belirtir (kg cinsinden ondalık sayı).

Cinsiyet (Gender)

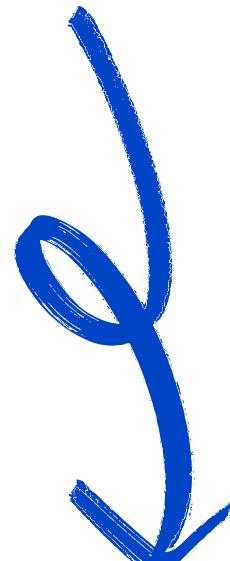
- Kişinin cinsiyetini belirtir (erkek veya dişi).

Sistolik Kan Basıncı (Systolic Blood Pressure)

- Kişinin sistolik kan basıncını belirtir (tam sayı).

Diyastolik Kan Basıncı (Diastolic Blood Pressure)

- Kişinin diyastolik kan basıncını belirtir (tam sayı).





Korelasyon
kısımında hangi
özelliklerin neden
kullanıldığı
anlatılacaktır.

Veri Seti Özellikleri

Kolesterol (Cholesterol)

- Kişinin kolesterol seviyesini belirtir (1: normal, 2: normal üstü, 3: iyi üstü).

Glikoz (Glucose)

- Kişinin glikoz seviyesini belirtir (1: normal, 2: normal üstü, 3: iyi üstü).

Sigara İçme Durumu (Smoking)

- Kişinin sigara içip içmediğini belirtir (0: hayır, 1: evet).

Alkol Tüketimi (Alcohol Intake)

- Kişinin alkol tüketip tüketmediğini belirtir (0: hayır, 1: evet).

Fiziksel Aktivite Durumu (Physical Activity)

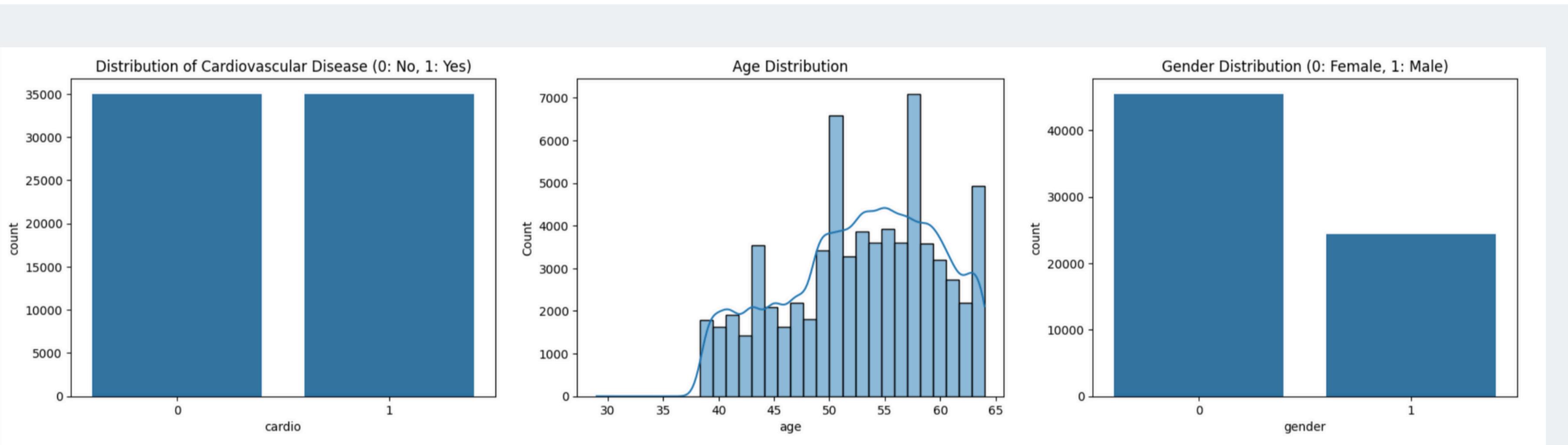
- Kişinin fiziksel aktivite durumunu belirtir (0: hayır, 1: evet).

Kardiyovasküler Hastalık Varlığı veya Yokluğu (Presence or Absence of Cardiovascular Disease)

- Kişinin kardiyovasküler hastalık varlığını veya yokluğunu belirtir (0: yok, 1: var).

Veri Önüşleme

* Kardiyovasküler Hastalık tahmini veri seti birden çok parametre ve etiketler içermektedir. Bu kısımda pandas kütüphanesi ile veri görselleştirme ve inceleme aşamaları getirilmiştir. Pandasın sunduğu DataFrame grafiği ile veri önizleme, kaldırma işlemleri gerçekleştirildi.



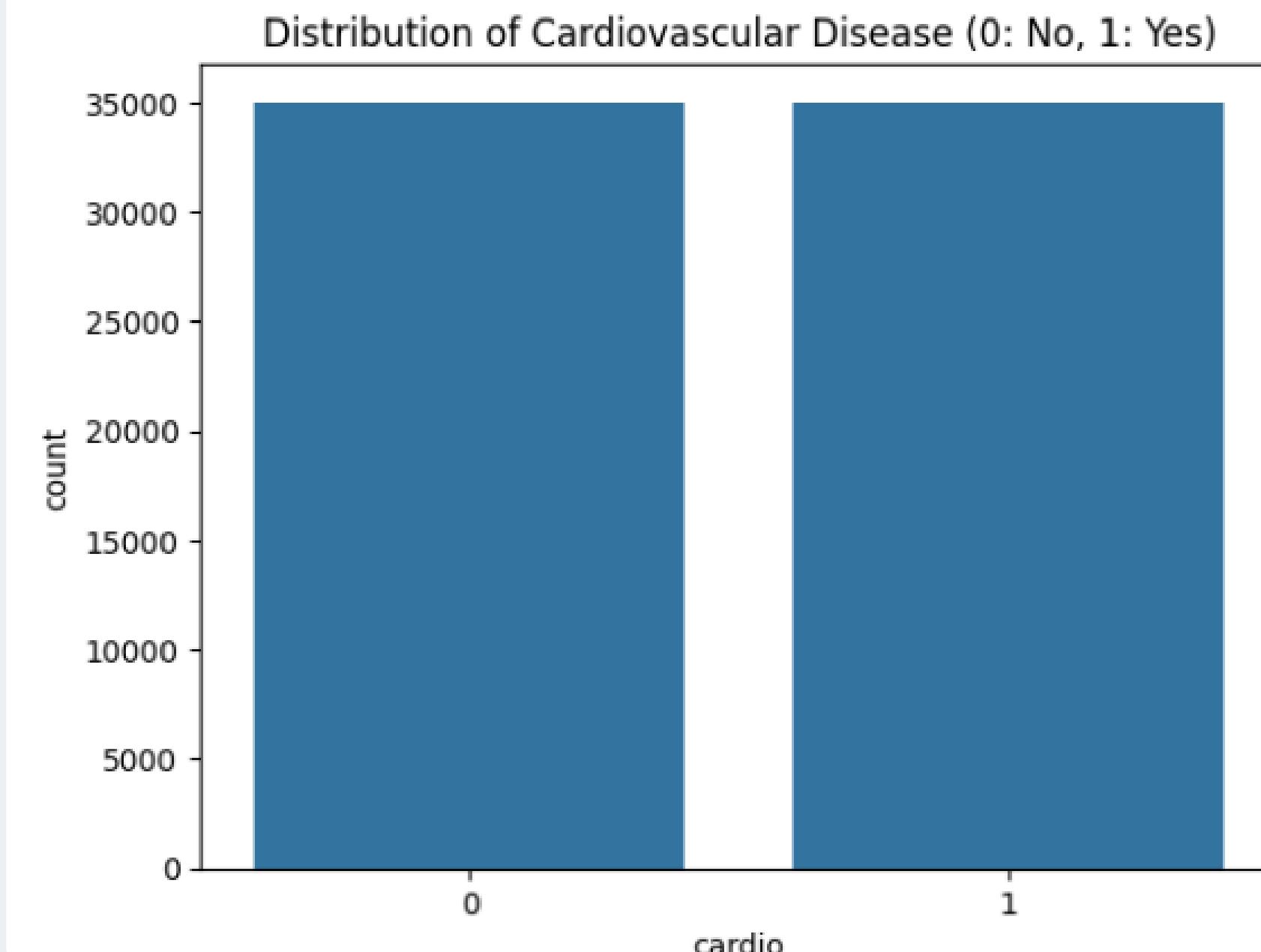
Veri Önüşleme

Burada yapılan işlem, 'cardio' adlı hedef değişkenin dağılımını görselleştirmektir. Bu değişkenin dağılımı, veri setindeki kardiyovasküler hastalığa (kalp-damar hastalığı) sahip olan ve olmayan bireylerin sayısını gösterir. sns.countplot() fonksiyonu kullanılarak, 'cardio' değişkeninin değerlerine göre gözlemlerin sayısı sayılmakta ve bu değerler çubuk grafik olarak gösterilmektedir.

* Bu gösterme, veri setindeki kardiyovasküler hastalık sınıflarının dengesini veya dengesizliğini anlamak için önemlidir.

```
# Distribution of the target variable 'cardio'  
print("\nDistribution of the target variable 'cardio':")  
sns.countplot(x='cardio', data=df)  
plt.title('Distribution of Cardiovascular Disease (0: No, 1: Yes)')  
plt.show()
```

Distribution of the target variable 'cardio':



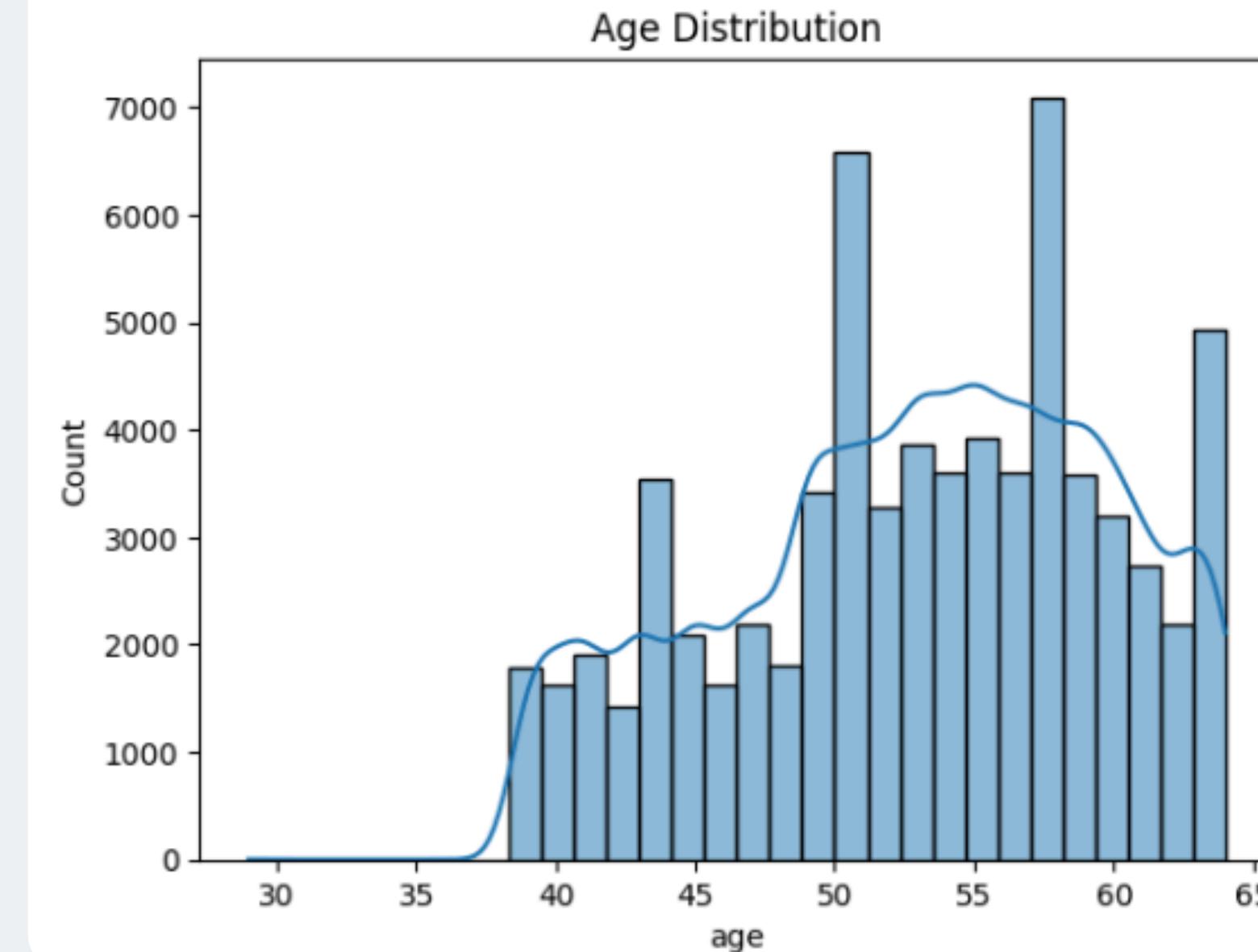
Veri Önüşleme

`sns.histplot()` fonksiyonu kullanılarak, 'age' değişkeninin değerlerinin histogramı oluşturulmaktadır.

* Bu görselleştirme, veri setindeki yaş dağılımının nasıl olduğunu gösterir. Histogram, yaş değerlerinin frekansını belirli aralıklara bölgerek bu aralıklardaki gözlem sayısını gösterir. Bu şekilde, yaş dağılımının genel şekli ve merkezi eğilimi hakkında bilgi edinilebilir

```
# Age distribution
print("\nAge distribution:")
sns.histplot(df['age'], bins=30, kde=True)
plt.title('Age Distribution')
plt.show()
```

Age distribution:

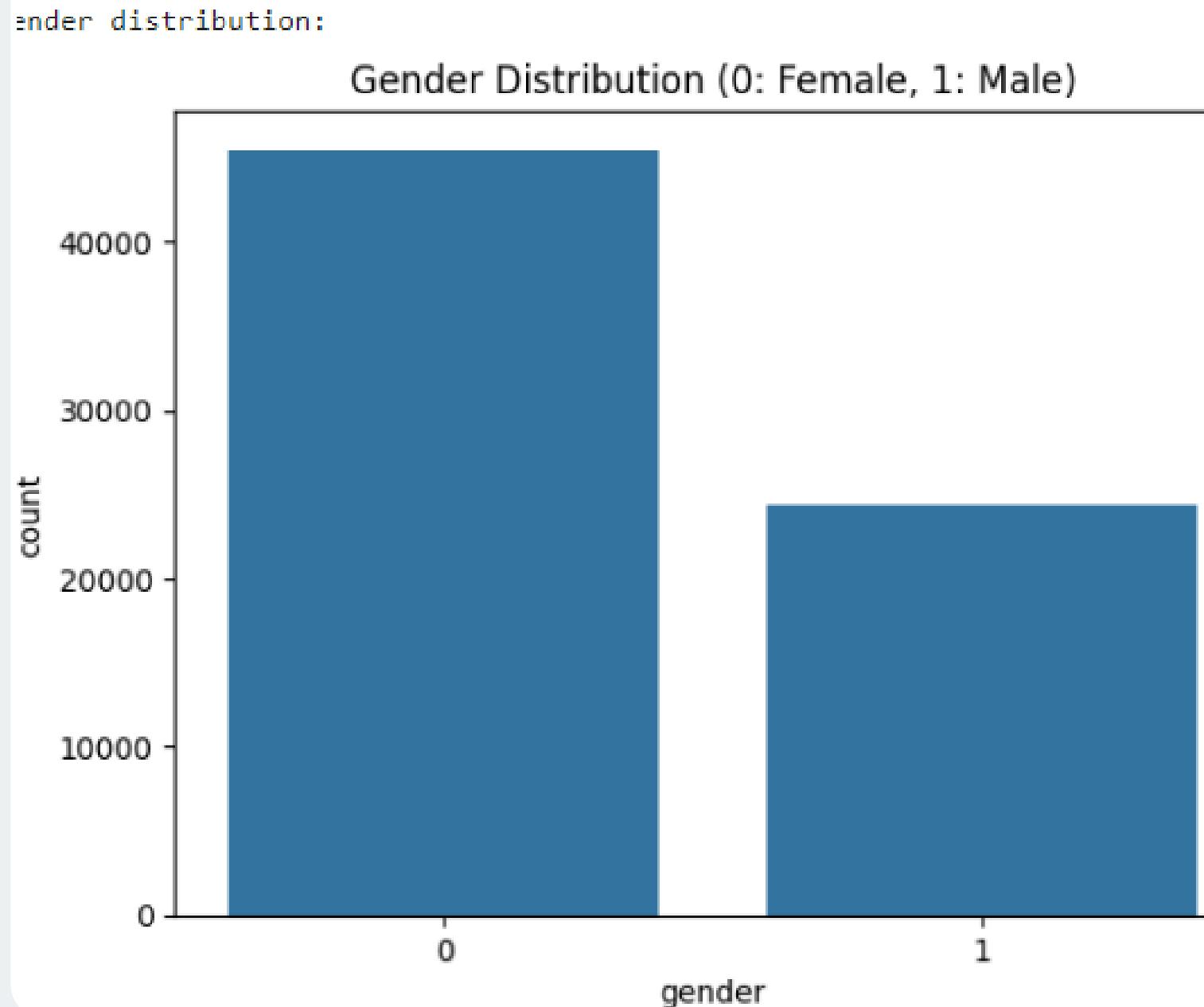


Veri Önisleme

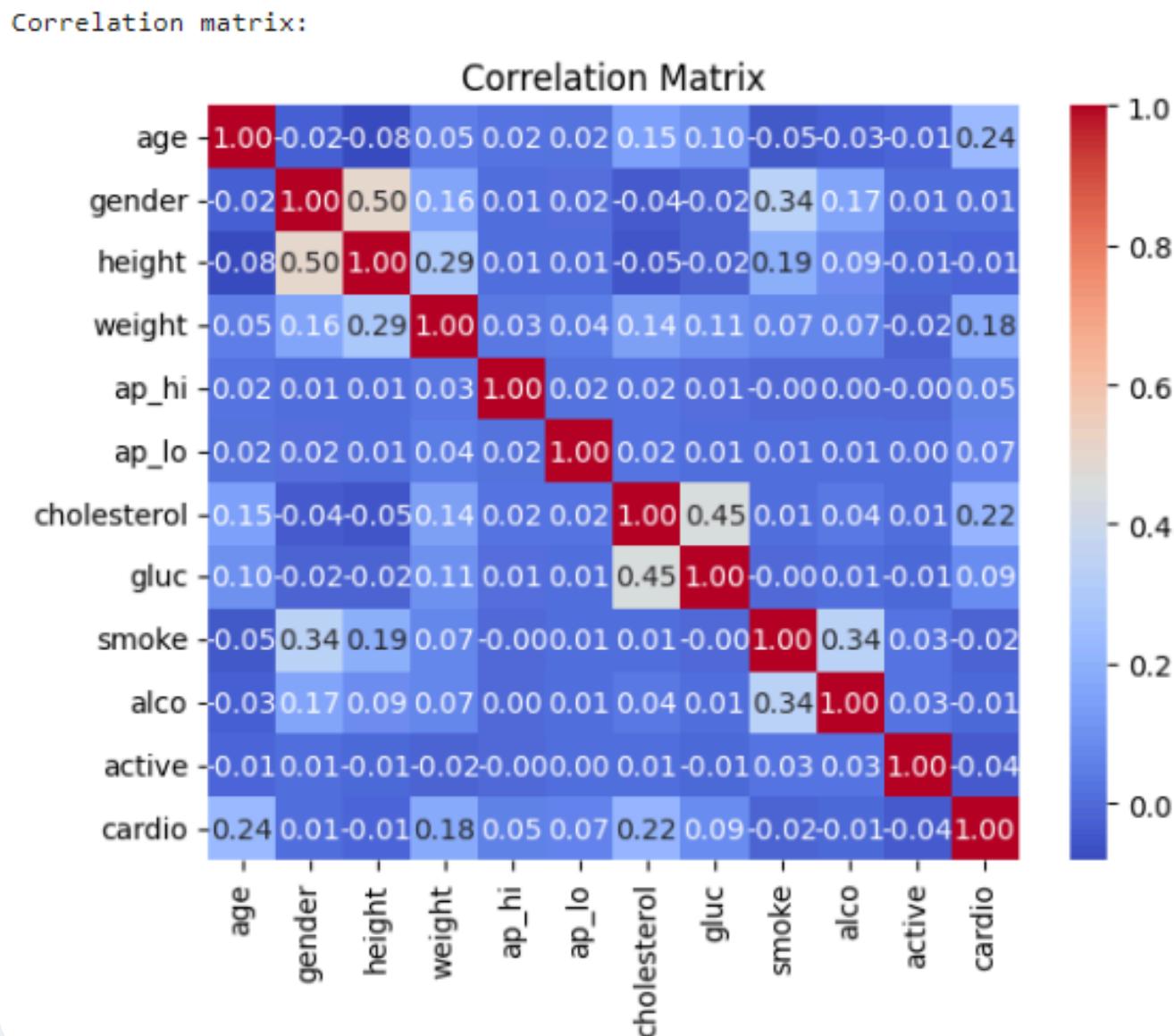
'gender' (cinsiyet) değişkeninin dağılımını görselleştirmektedir. sns.countplot() fonksiyonu kullanılarak, 'gender' değişkeninin değerlerine göre gözlemlerin sayısı sayılmakta ve bu değerler çubuk grafik olarak gösterilmektedir.

* Bu görselleştirme, veri setindeki cinsiyet dağılımını gösterir. Veri setindeki kadın ve erkek sayılarının karşılaştırılması ve cinsiyet dağılımının dengeli olup olmadığıın belirlenmesi için kullanılabilir.

```
gender distribution
print("\nGender distribution:")
sns.countplot(x='gender', data=df)
lt.title('Gender Distribution (0: Female, 1: Male)')
lt.show()
```



```
print("\nCorrelation matrix:")
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



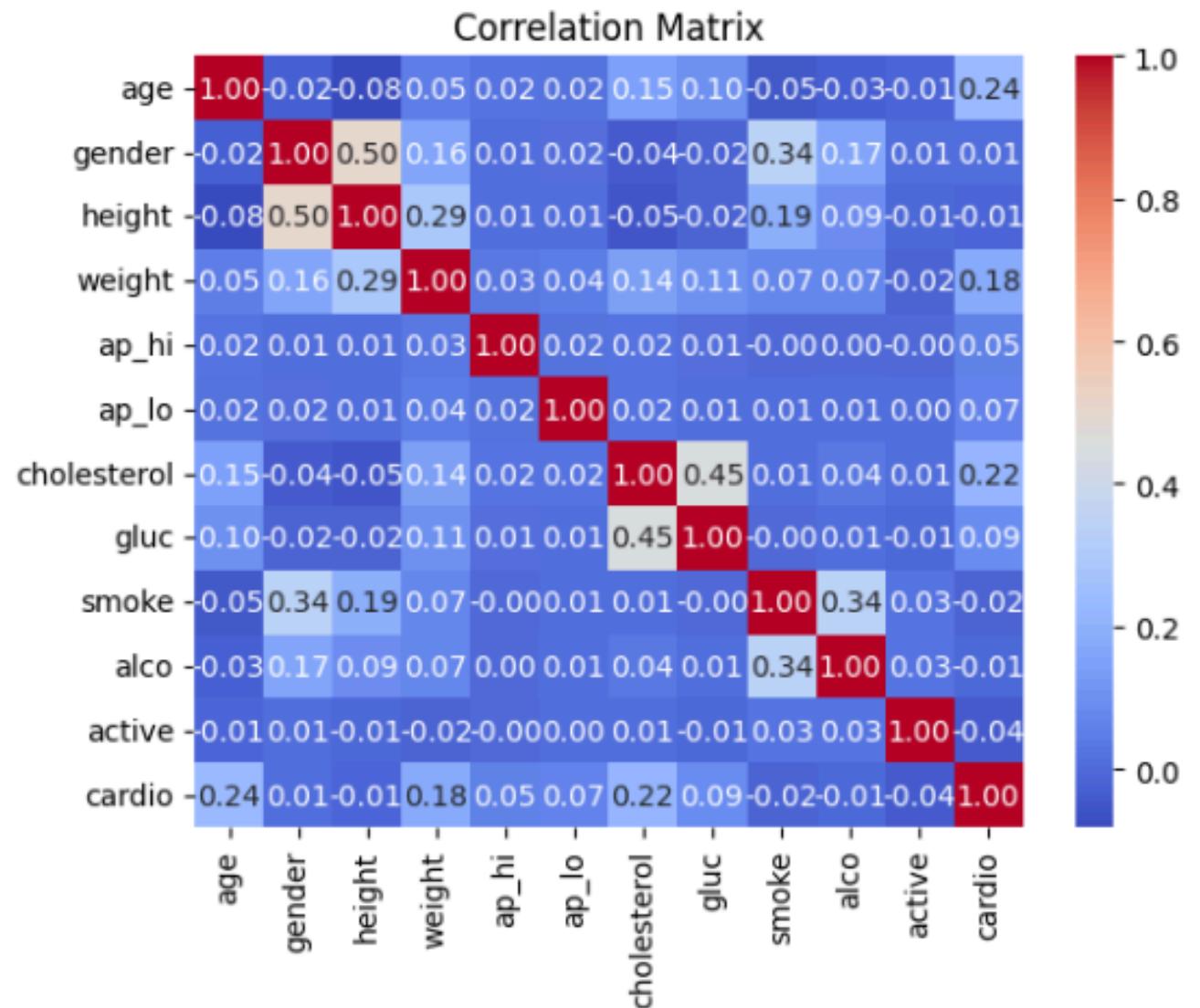
Veri Önüşleme

Veri setinde birden çok parametre bulunabilir ve bu parametreler arasında bir bağılilik veya korelasyon olabilir. Diğer bir deyişle, bazı parametreler aynı veri noktasında birlikte aktivasyon gösterebilir.

Bu durumu kontrol etmek için genellikle "Korelasyon Matrisi" adı verilen bir çıktıya bakılır. Korelasyon matrisi, değişkenler arasındaki ilişkinin gücünü ve yönünü gösteren istatistiksel bir araçtır.

```
print("\nCorrelation matrix:")
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

Correlation matrix:



- Pearson korelasyon katsayısı, iki değişken arasındaki ilişkinin gücünü ve yönünü ölçen istatistiksel bir ölçümdür. Genellikle -1 ile +1 arasında bir değer alır.

Veri Önisleme

1. **df.corr()** yöntemi kullanılarak değişkenler arasındaki Pearson korelasyon katsayıları hesaplanır ve bir korelasyon matrisi oluşturulur.
2. **sns.heatmap()** fonksiyonu kullanılarak korelasyon matrisi ısı haritası olarak görselleştirilir. Isı haritası, korelasyon katsayılarını renk tonlarıyla gösterir; pozitif korelasyonlar genellikle sıcak renklerle (örneğin, kırmızı), negatif korelasyonlar ise genellikle soğuk renklerle (örneğin, mavi) temsil edilir.
3. **annot=True** parametresi, her bir hücreye korelasyon katsayısının değerini ekler. **cmap='coolwarm'** parametresi, ısı haritasının renk harmasını belirler. **fmt=".2f"** parametresi, korelasyon katsayılarının virgülüden sonra iki basamakla gösterilmesini sağlar.

Veri Önüşleme

* **Normalizasyon**, veri setinin özelliklerinin birbirine benzer ölçeklere sahip olmasını sağlamak için kullanılır. Bu, bir özelliğin diğerine göre ağırlığının daha fazla olmamasını veya bir özelliğin modelin eğitimini domine etmemesini sağlar.

Min-Max normalizasyonu, veri özelliklerini belirli bir aralığa dönüştürmek için kullanılan bir yöntemdir. Bu yöntem, her özellik için minimum ve maksimum değerleri kullanarak veriyi 0 ile 1 arasında bir aralığa ölçeklendirir.

Min-Max normalizasyonu genellikle aşağıdaki formülle ifade edilir:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Veri Önüşleme



Min-Max normalizasyonu:

```
def min_max_normalization(series):
    min_value = series.min()
    max_value = series.max()
    normalized_series = (series - min_value) / (max_value - min_value)
    return normalized_series
```

```
df['normalized_ap_hi'] = min_max_normalization(df['ap_hi'])
df['normalized_ap_lo'] = min_max_normalization(df['ap_lo'])
df['normalized_height'] = min_max_normalization(df['height'])
df['normalized_weight'] = min_max_normalization(df['weight'])
df['normalized_age'] = min_max_normalization(df['age'])
df['normalized_cholesterol'] = min_max_normalization(df['cholesterol'])
df['normalized_gluc'] = min_max_normalization(df['gluc'])
```





Veri Önüşleme

Veriler model eğitimine sokulmadan önce ‘Train’, ‘Test’ ve ‘Val’ olarak 3'e ayrılmıştır.

Hiperparametreler, bir makine öğrenimi modelinin yapılandırılması ve performansını etkileyen, modelin eğitimi sırasında değil, önceden belirlenmiş olan ve genellikle deneme-yanılma yöntemiyle ayarlanan parametrelerdir. Örneğin, bir sinir ağı modelindeki katman sayısı, her katmandaki nöron sayısı, öğrenme hızı, düzenleme parametreleri, aktivasyon fonksiyonları vb.

- **Train seti**, bir makine öğrenimi modelinin öğrenme sürecinde kullanılan ve modelin genel veri desenlerini anlaması için büyük miktarda veri içeren bir veri setidir.
- **Test seti**, modelin eğitim sırasında görmediği ve performansını ölçmek için kullanılan ayrı bir veri setidir. Modelin gerçek dünya verilerine ne kadar iyi uyarlandığını değerlendirmek için test seti kullanılır.
- **Validation seti**, modelin hiper parametrelerini ayarlamak ve aşırı uyumu önlemek için kullanılan ayrı bir veri setidir; bu set, genellikle modelin eğitim setinden ayrılan bir parçadır ve farklı hiper parametre ayarlarının performansını değerlendirmek için kullanılır.



Veri Önüşleme

X_train, X_val ve X_test veri kümeleri, özellik matrisleridir (**girdiler**). y_train, y_val ve y_test ise hedef değişkenlerin vektörleridir (**çıktılar**).

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, random_state=42)
```

```
x_train_transposed=x_train.T
x_val_transposed=x_val.T
x_test_transposed=x_test.T
```

```
y_train=y_train.values.reshape(-1,1)
y_val=y_val.values.reshape(-1,1)
y_test=y_test.values.reshape(-1,1)
```

```
x_test_transposed.shape
```



Veri Önisleme

Veri Setindeki '**cardio**' sütununu verilerin etiketi olarak kullanılacak ve modelin tespit etmesini amaçlanan sütun olacak. Geri kalan veriler ise etikete göre modelin anlaması sağlanacak veriler olacak. Bu durumda yapılan işlemler sırayla:

- • Etiket ve diğer verilerin ayrıştırılması.
- • Verilerin Train, Test ve Validation olarak bölünmesi.
- • Modele uygunluğun sağlanması için transpose ve reshape işlemlerinin yapılması

Tüm veri kullanılmadı:
cardio ile **korelasyonu**
0.04'ün altındaki
kullanılmadı.

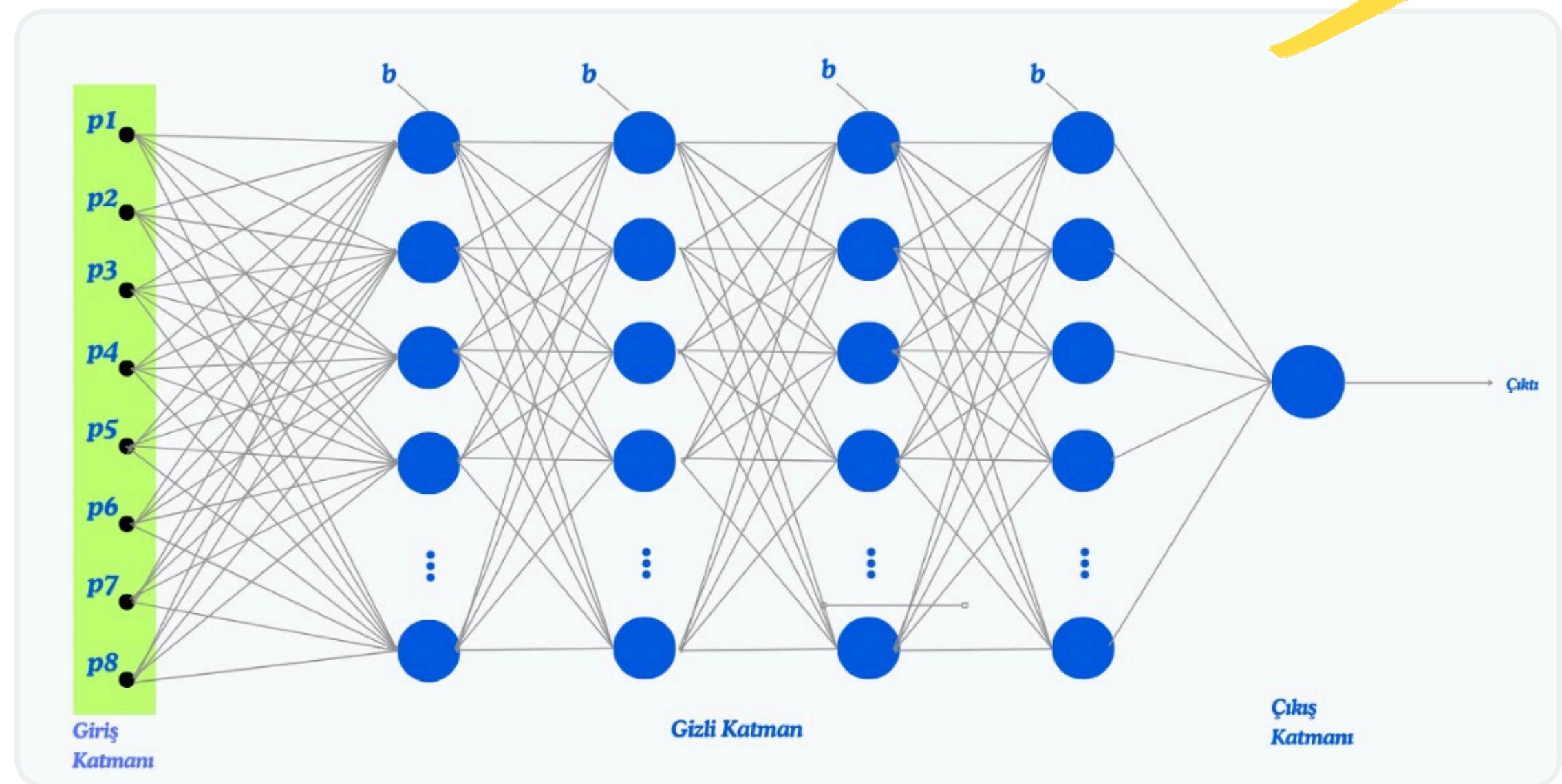


Model Yapısı

* Modelin yapısı dinamik bir şekilde *kullanıcının seçeceği sayıda* olacak gizli katmandan ve 1 çıkış katmanından oluşturulmuştur.

* Yapının dinamik bir şekilde oluşturulmasının istenmesindeki amaç farklı veri setleri ve farklı görevler için ya da aynı görev için optimum model yapısını oluştururken statik yapının zorluğundan kurtulunmak istenmesi ve derin öğrenmede kullanılan hazır kütüphanelerin kısıtlı bir seviyede de olsa taklit edilmesini sağlamaya çalışmaktadır.

Model Yapısı



Modelin yapısını oluşturan **gizli katmanlar** için ReLU, çıktı katmanı için ise sigmoid aktivasyon fonksiyonu kullanılmasına karar verilmiştir. Dinamik bir şekilde gizli katman sayıları ve katmanlardaki nöron sayıları ayarlanabilmektedir.

Model için **Adam optimizasyon** algoritması kullanılmıştır. Adam algoritması modele implemente edilirken **beta1**, **beta2** ve **epsilon** parametreleri sırasıyla **0.9**, **0.999** ve **1e-8** olarak belirlenmiştir. **Öğrenme oranı** **0.001** olarak belirlenmiştir.

Bunlar algoritma performansını etkileyen hiperparametrelerdir.



Model Yapısı

- **Beta1 ve Beta2:** Beta1, momentum teriminin eski gradyan değerleri ile yeni gradyan değerleri arasındaki dengeyi belirler. Beta2 ise momentumun karesi teriminin eski gradyan karesi değerleri ile yeni gradyan karesi değerleri arasındaki dengeyi belirler.
- **Epsilon:** Epsilon, sıfıra bölme hatasını önlemek için eklenen küçük bir değerdir. Bu değer, gradyanın karesinin kökü alınırken sıfıra bölme hatası olmasını engeller.
- **Öğrenme Oranı (Learning Rate):** Öğrenme oranı, modelin her iterasyonda parametrelerini ne kadar güncelleyeceğini belirler. Yüksek bir öğrenme oranı, daha hızlı öğrenme sağlayabilir, ancak aşırı öğrenmeye neden olabilir. Düşük bir öğrenme oranı ise daha istikrarlı bir öğrenme sağlayabilir, ancak daha yavaş bir eğitim sürecine neden olabilir.

Model için **Adam optimizasyon** algoritması kullanılmıştır. Adam algoritması modele implemente edilirken **beta1, beta2 ve epsilon** parametreleri sırasıyla **0.9, 0.999** ve **1e-8** olarak belirlenmiştir. **Öğrenme oranı 0.001** olarak belirlenmiştir.

Bunlar algoritma performansını etkileyen hiperparametrelerdir.



Model Yapısı

Adam optimizer adı verilen bir optimizasyon algoritması sınıfını tanımlar. Bu sınıf, gradient tabanlı optimizasyon yöntemleri kullanarak makine öğrenimi modellerinin eğitimini hızlandırmak için kullanılır.

- **t değişkeni:** Optimizer'in iç durumu olarak, güncelleme adımını izlemek için kullanılır.
- **m ve v sözlükleri:** Her bir parametrenin gradyanlarının ortalaması ve varyansını depolar. Bu değerler, optimize edilen parametrelerin hız ve yönünü temsil eder.



```
import numpy as np

class AdamOptimizer:
    def __init__(self, learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.learning_rate = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.t = 0
        self.m = {} #gradyanların ortalaması hız ve yönü temsil eder
        self.v = {} #gradyanların varyansı

    def initialize_params(self, params):
        for key in params:
            self.m[key] = np.zeros_like(params[key])
            self.v[key] = np.zeros_like(params[key])

    def update_params(self, params, gradients):
        self.t += 1
        lr_t = self.learning_rate * np.sqrt(1 - self.beta2 ** self.t) / (1 - self.beta1 ** self.t)

        for key in params:
            gradient_key = 'd' + key.capitalize()
            self.m[key] = self.beta1 * self.m[key] + (1 - self.beta1) * gradients[gradient_key] #ağırılık
            self.v[key] = self.beta2 * self.v[key] + (1 - self.beta2) * (gradients[gradient_key] ** 2)

            m_corrected = self.m[key] / (1 - self.beta1 ** self.t) # momentlerin düzeltılması
            v_corrected = self.v[key] / (1 - self.beta2 ** self.t)

            params[key] -= lr_t * m_corrected / (np.sqrt(v_corrected) + self.epsilon)
```



Model Yapısı

- Yapay sinir ağı sınıfını tanımlar ve eğitimini gerçekleştirir.
- Yapay sinir ağı, gizli katmanlar ekleyerek ve aktivasyon fonksiyonlarını kullanarak veriyi işler.
- Geriye yayılım algoritmasını kullanarak ağırlıkları günceller ve eğitim verisinin kaybını (loss) azaltır.
- Verilen girdi verileri için tahminler yapar.

```
class NeuralNetwork:  
    def __init__(self, input_size):  
        self.input_size = input_size  
        self.hidden_layers = []  
        self.optimizer = AdamOptimizer(learning_rate=0.001)  
  
    def add_hidden_layer(self, layer_type, size):  
        if len(self.hidden_layers) == 0:  
            # First hidden layer  
            prev_layer_size = self.input_size  
        else:  
            prev_layer_size = self.hidden_layers[-1]['weights'].shape[0]  
  
        if layer_type == 'relu':  
            weights = np.random.randn(size, prev_layer_size) * np.sqrt(2 / prev_layer_size) #HE başlatma  
        else:  
            weights = np.random.randn(size, prev_layer_size) * np.sqrt(1 / prev_layer_size) #xavier başlatma  
        biases = np.zeros((size, 1))  
        self.hidden_layers.append({'weights': weights, 'biases': biases})  
  
    def relu(self, x):  
        return np.maximum(0, x), (x)  
  
    def relu_derivative(self, x):  
        return np.where(x > 0, 1, 0)  
  
    def sigmoid(self, x):  
        return 1 / (1 + np.exp(-x)), (x)  
  
    def sigmoid_derivative(self, x):  
        return self.sigmoid(x)[0] * (1 - self.sigmoid(x)[0])
```



Model Yapısı

Gradyan kaybolması, eğitim sırasında, ağdaki daha derin katmanlara ilerledikçe, gradyanlar giderek küçülür veya tamamen kaybolabilir. Bu durum, geriye doğru yayılım sırasında gradyanların katmanlardan katmanlara geçerken eksponansiyel olarak küçülmesi veya sıfır yaklaşması sonucu oluşur.

- * Modelimizin ağırlıklarını herhangi bir ağırlık başlatması metodu kullanmadan rastgele olarak başlattık. Ancak modelin gradyan çıktılarını incelediğimizde gradyanların **aşırı küçük olduğunu** gözlemledik. Gerçekleştirdiğimiz literatür çalışmamızda bu problemin **gradyan kaybolması** olarak adlandırıldığını tespit ettik.
- * Sorunun çözümü için **Saiprasad Koturwar ve Shabbir N Merchant**, önerdiği yöntemde çeşitli ağırlık başlatma yöntemleri kullanılarak bu sorunun çözülebildiğini göstermişlerdir.



Model Yapısı

XAVIER GOLROT AĞIRLIK BAŞLATMASI

- Xavier Golrot ve Yoshua Bengio, sigmoid ve tanh aktivasyon fonksiyonları gibi lineer olmayan aktivasyon fonksiyonları kullanılan katmanlardaki ağırlıkların gradyan kaybolmasına sebebiyet vermeyecek şekilde başlatılması için bir yöntem geliştirdiler. Sigmoid aktivasyon fonksiyonu kullanılan bir sinir ağında yaptıkları incelemelerde 3 katmana kadar olan yapay sinir ağlarında aktivasyon fonksiyonunun çıktı değerlerinin 100 iterasyondan sonra sökülmeye başladığını gözlemlemişlerdir. 4 katmanlı olan bir yapay ağında ise iterasyon 20 ye bile gelmemişken sökümlenmenin gerçekleştiğini gözlemlemişlerdir. Geliştirdikleri yönteme göre standart normal dağılım ile başlatılan ağırlıkları önceki katmanın boyutnun çarpmaya göre tersinin kareköküyle çarparak normalize ederek bu sorunu aşmışlardır.



Model Yapısı

HE AĞIRLIK BAŞLATMASI

- Kaiming He ve arkadaşları ReLU ve türevi aktivasyon fonksiyonlarındaki gibi kısmi lineer aktivasyon fonksiyonları kullanılan katmanlardaki ağırlıkların gradyan kaybolmasına sebebiyet vermeyecek şekilde başlatılması için bir yöntem geliştirdiler. Geliştirdikleri ağırlık başlatma yöntemiyle ReLU aktivasyon fonksiyonu kullanan daha derin sinir ağları oluşturulabilmesinin önünü açmışlardır. Geliştirdikleri yönteme göre standart normal dağılım ile başlatılan ağırlıkları önceki katmanın boyutnun çarpamaya göre tersinin iki katınının kareköküyle çarparak normalize ederek bu sorunu aşmışlardır.



Model Yapısı

HE ve Xavier Golrot Ağırlık Başlatması Uygulaması *le*

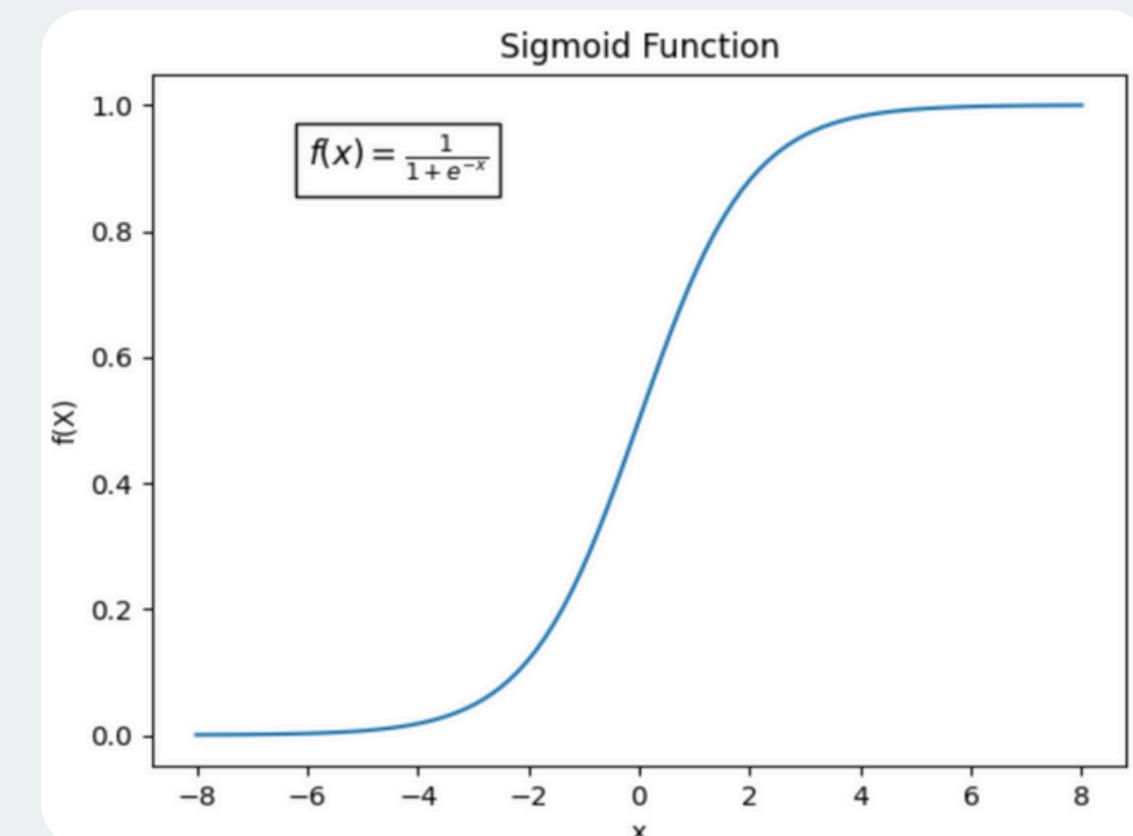
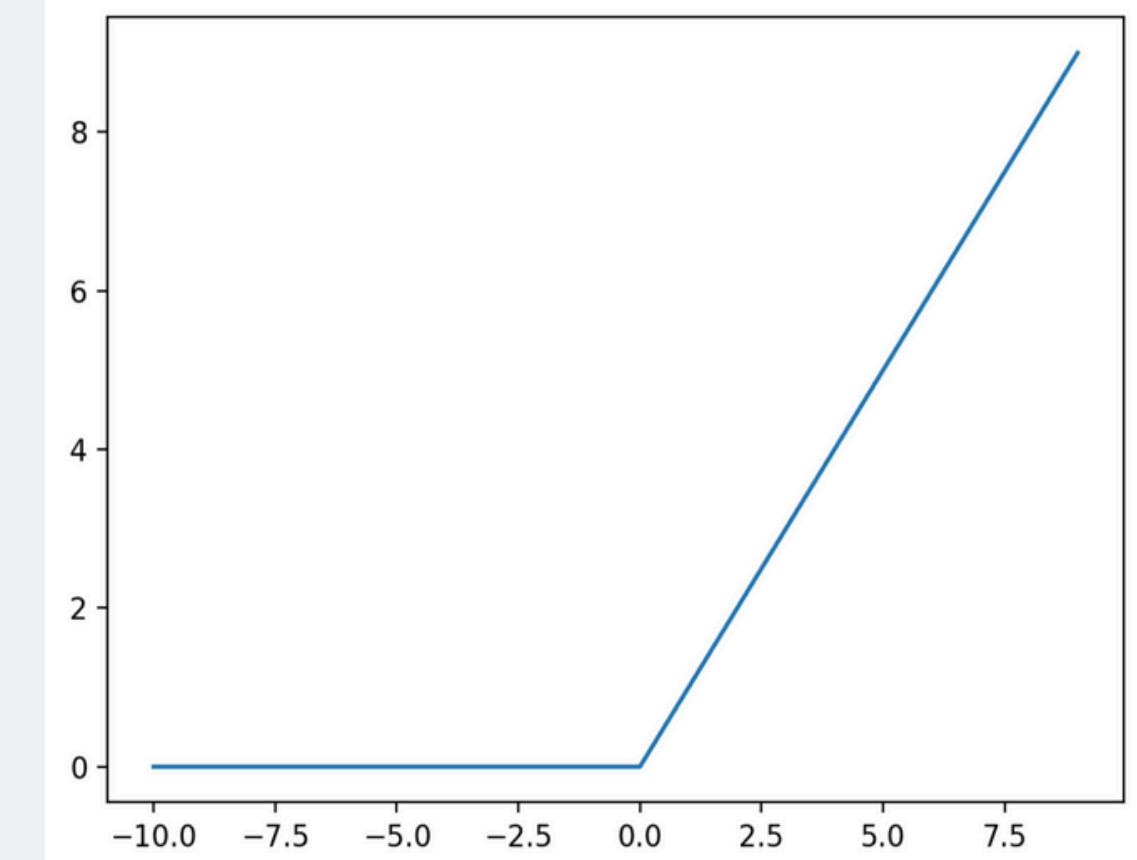
```
def add_hidden_layer(self, layer_type, size):
    if len(self.hidden_layers) == 0:
        # First hidden layer
        prev_layer_size = self.input_size
    else:
        prev_layer_size = self.hidden_layers[-1]['weights'].shape[0]

    if layer_type == 'relu':
        weights = np.random.randn(size, prev_layer_size) * np.sqrt(2 / prev_layer_size) #HE başlatma
    else:
        weights = np.random.randn(size, prev_layer_size) * np.sqrt(1 / prev_layer_size) #xavier başlatma
    biases = np.zeros((size,1))
    self.hidden_layers.append({'weights': weights, 'biases': biases})
```

Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları, derin öğrenme modellerinde bilgi işleme sürecinin önemli bir parçasını oluşturur ve modelin karmaşıklığını, performansını ve öğrenme hızını etkiler.

- RELU
- Sigmoid

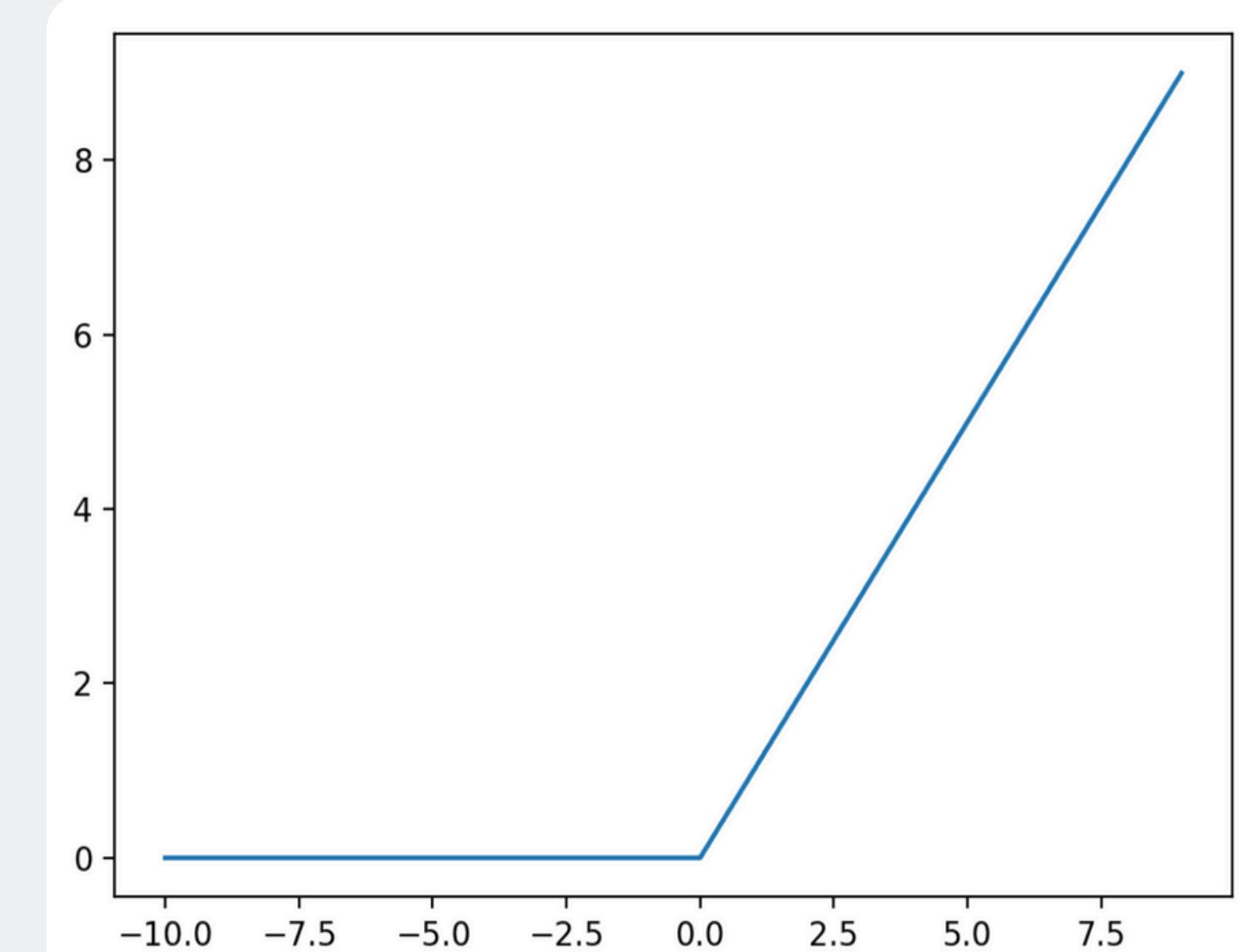


Aktivasyon Fonksiyonları

RELU



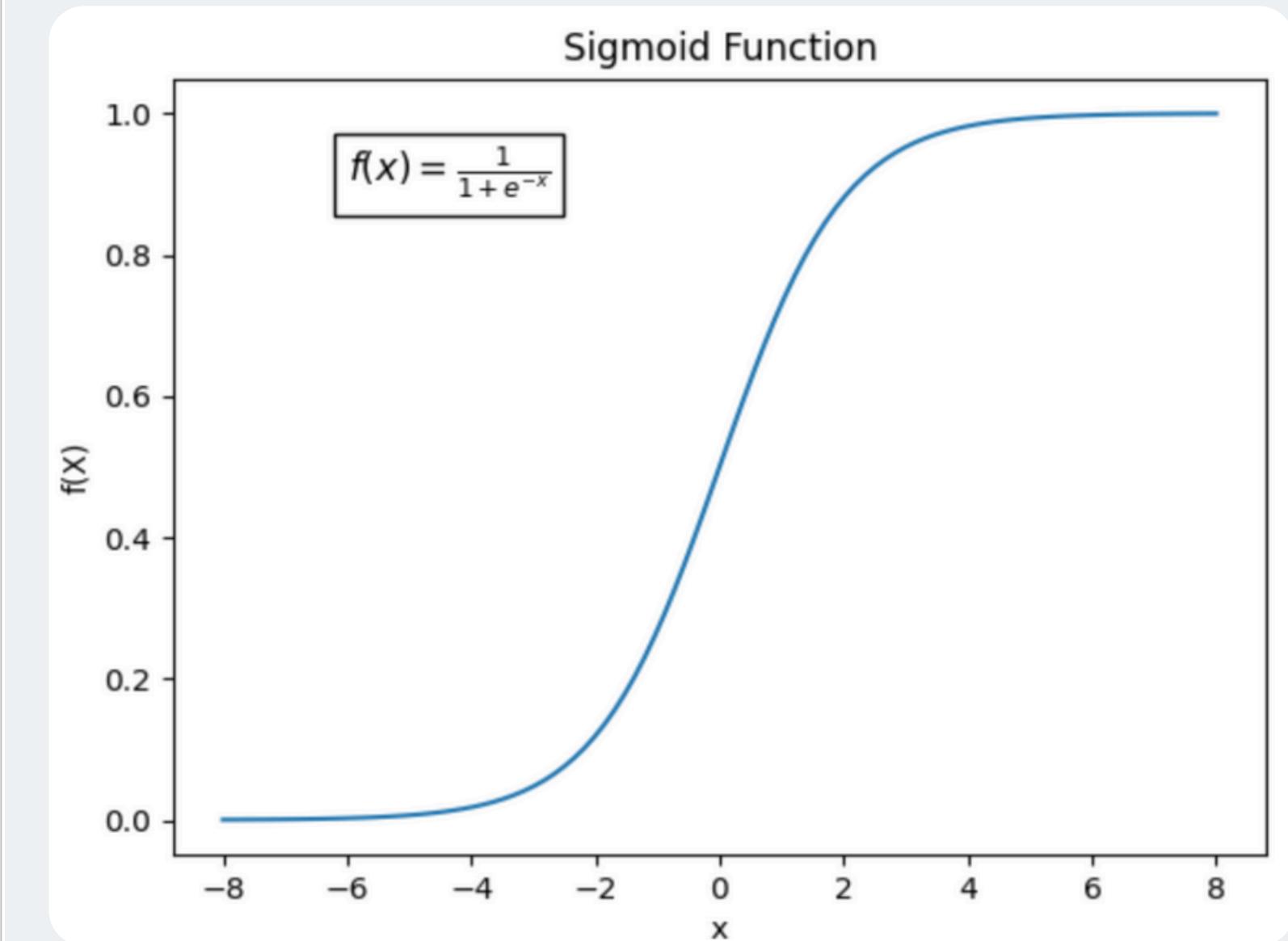
- ReLU fonksiyonu, girdi negatif ise sıfırı, pozitif ise girdiyi olduğu gibi geri döndürür. ReLU'nun temel avantajlarından biri, hesaplama maliyetinin düşük olmasıdır. Bu fonksiyon, sıfır olmayan değerlerde gradyanları geçirir, bu da geriye yayılım sürecinde daha hızlı öğrenmeyi sağlar.



Aktivasyon Fonksiyonları

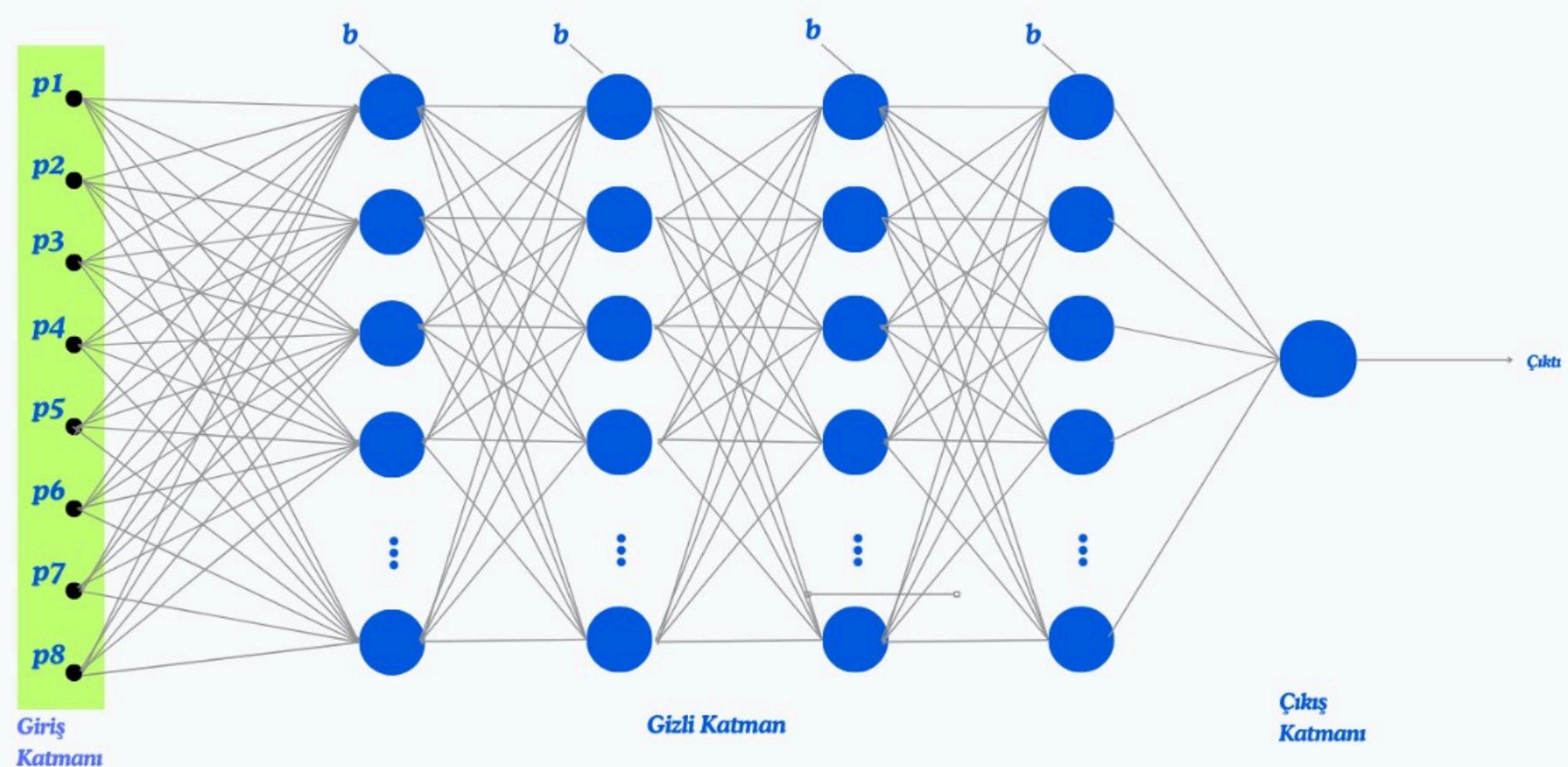
Sigmoid

- Sigmoid fonksiyonu, girdiyi $[0, 1]$ aralığına sıkıştırın bir aktivasyon fonksiyonudur. Bu fonksiyon, genellikle çıktı sınıflarının olasılık dağılımını modellerken kullanılır. Sigmoid fonksiyonu, özellikle ikili sınıflandırma problemlerinde çıktı katmanında kullanılır.



Aktivasyon Fonksiyonları

nn



```
nn = NeuralNetwork(input_size=8)
nn.add_hidden_layer('relu',32)
nn.add_hidden_layer('relu',128)
nn.add_hidden_layer('relu',256)
nn.add_hidden_layer('relu',64)
nn.add_hidden_layer('sigmoid',1)
```



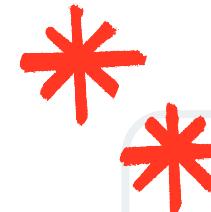
Kayıp Fonksiyonu

- Binary Cross Entropy (BCE) kayıp fonksiyonu, ikili sınıflandırma problemlerinde sıkılıkla kullanılan bir kayıp fonksiyonudur ve modelin tahmin ettiği sonuçlar ile gerçek etiketler arasındaki farkı ölçerek bir hata değeri hesaplar.
- BCE kayıp fonksiyonu, özellikle ikili sınıflandırma problemlerinde kullanılır, yani çıktı sınıflarının iki olası durumdan birine ait olduğu durumlarda etkilidir. Örneğin, bir resmin kediyi içerip içermediğini sınıflandırmak gibi.

BCE kayıp fonksiyonu, sigmoid aktivasyon fonksiyonu ile birlikte çıktı katmanında kullanılır. Sigmoid fonksiyonu, çıktıları $[0, 1]$ aralığında bir olasılık dağılımına dönüştürür. Bu, her bir sınıf için bir olasılık değeri elde etmemizi sağlar. Örneğin, bir resmin kediyi içermeye olasılığı gibi.



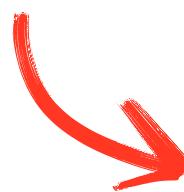
Kayıp Fonksiyonu



$$\text{Loss} = -\frac{1}{\frac{\text{output size}}{\text{size}}} \sum_{i=1}^{\frac{\text{output size}}{\text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Bu formülde, her bir örnek için gerçek etiket ve tahmin edilen olasılık arasındaki farkı ölçeriz.

- Eğer gerçek etiket 1 ise, yani örneğin pozitif sınıfına aitse, bu kısım $\log(y^i)$ şeklinde ifade edilir.
- Eğer gerçek etiket 0 ise, yani örneğin negatif sınıfına aitse, bu kısım $\log(1-y^i)$ şeklinde ifade edilir.



Toplam kayıp değeri, tüm örnekler üzerinden ortalama alınarak hesaplanır.



Kayıp Fonksiyonu

*
*

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

```
def binary_cross_entropy(self,A,Y):
    m = A.shape[1]

    return ((-1/m)*(np.dot(np.log(A), Y) + np.dot(np.log(1-A), 1-Y)))[0][0]
```



Kayıp Fonksiyonu

Eğitim verisi üzerindeki çıktıının, gerçek etiketlerle karşılaştırılması için binary cross entropy maliyeti hesaplanır (binary_cross_entropy yöntemi).

```
def train(self, X_train, y_train, X_val, y_val, epochs):

    self.initialize_optimizer()

    costs_train = []
    costs_val = []
    for epoch in range(epochs):
        output, caches = self.feedforward(X_train)
        gradients = self.backpropagation(y_train, output, caches)
        cost_train = self.binary_cross_entropy(output, y_train)
        costs_train.append(cost_train)
        self.update_params(gradients)
        # self.update_params(gradients, learning_rate)
        output_val, _ = self.feedforward(X_val)
        cost_val = self.binary_cross_entropy(output_val, y_val)
        costs_val.append(cost_val)
        print(f"epoch: {epoch+1}, cost_train: {cost_train}, cost_val: {cost_val}")
    return costs_train, costs_val
```



- Her bir epoch (iterasyon) boyunca, modelin eğitim verisi üzerinde **ileri yayılım (feedforward)** ve **geriye yayılım (backpropagation)** yaparak *gradyanlarını* hesaplar.
- *Eğitim ve doğrulama verileri* için **binary cross entropy maliyetini** hesaplar ve bu maliyetleri kaydeder.
- **Optimizasyon algoritması** kullanılarak *modelin parametreleri* güncellenir.
- Her epoch sonunda eğitim ve doğrulama maliyetleri ekrana yazdırılır.

Model EĞİTİMİ

```
def train(self, X_train, y_train,X_val,y_val, epochs):  
  
    self.initialize_optimizer()  
  
    costs_train = []  
    costs_val = []  
    for epoch in range(epochs):  
        output, caches = self.feedforward(X_train)  
        gradients = self.backpropagation(y_train, output, caches)  
        cost_train = self.binary_cross_entropy(output, y_train)  
        costs_train.append(cost_train)  
        self.update_params(gradients)  
        # self.update_params(gradients,learning_rate)  
        output_val, _ = self.feedforward(X_val)  
        cost_val = self.binary_cross_entropy(output_val, y_val)  
        costs_val.append(cost_val)  
        print(f"epoch: {epoch+1}, cost_train: {cost_train}, cost_val: {cost_val}")  
    return costs_train,costs_val
```



Sonuçlar

mm

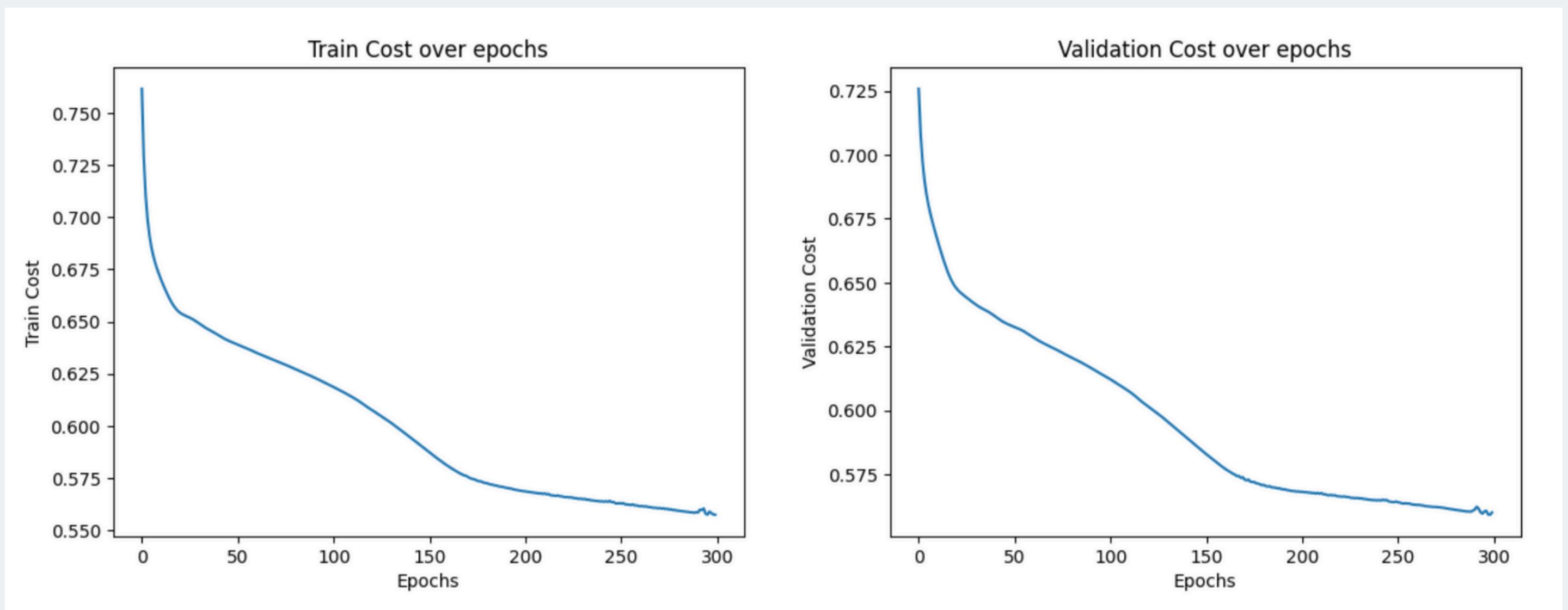
Train ve **Validation** olarak ayrılan verileri modele uygulayarak eğitime başlanması sağlandı. Epoch '300', learning rate ise '0.001' belirlendi. Epoch'ların her ilerleyişinde 'loss' yani 'cost_train' ve 'cost_val' olmak üzere train ve validation loss oranı elde edildi.

```
epoch: 290, cost_train: 0.5587204774528861, cost_val: 0.5607545582836679
epoch: 291, cost_train: 0.5585390407286539, cost_val: 0.5611637159936629
epoch: 292, cost_train: 0.5599000468392694, cost_val: 0.5622732385141546
epoch: 293, cost_train: 0.5597283539409621, cost_val: 0.5615403528610386
epoch: 294, cost_train: 0.5605249209088945, cost_val: 0.5601335984739713
epoch: 295, cost_train: 0.5579715052570564, cost_val: 0.5595890209818323
epoch: 296, cost_train: 0.5575543075240604, cost_val: 0.5604003187931748
epoch: 297, cost_train: 0.5590384545666129, cost_val: 0.5606976153621059
epoch: 298, cost_train: 0.5582813744304665, cost_val: 0.559244354233953
epoch: 299, cost_train: 0.557637256384701, cost_val: 0.5591429896053056
epoch: 300, cost_train: 0.5575112016117704, cost_val: 0.5600299799526989
```

Sonuçlar

mm

Her bir Epoch sonunda elde edilen 'loss' değerlerinin grafiğini çıkarttığımızda:



Sonuçlar

- TP : True Positive
- TN : True Negative
- FP : False Positive
- FN : False Negative

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- F1 SCORE: 0.727

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

- ACCURACY: 0.729

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
X_test_transposed = X_test.T
predictions, _ = nn.predict(X_test_transposed)

#Apply a threshold to turn predictions into binary output
threshold = 0.5
binary_predictions = (predictions > threshold).astype(int)
accuracy = accuracy_score(y_test, binary_predictions.T)
f1 = f1_score(y_test, binary_predictions.T) # 'y_test' gerçek etiketlerdir
print("F1 Score: ", round(f1,3))
print("Accuracy: ", round(accuracy,3))
```

Sonuçlar

Model Confusion Matrix

```
import seaborn as sns
import matplotlib.pyplot as plt

# Seaborn ile confusion matrix'i görselleştir

cm = confusion_matrix(y_test, binary_predictions.T)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues') # 'annot=True' değerleri gösterir
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

