Backpropagation Proje Raporu

Hazırlayan

Fırat Kaan Bitmez

Öğrenci Numarası

23281855

Dersin Hocası

Prof.Dr. Erdal Kılıç

Giriş

Bu rapor, veri madenciliği dersi kapsamında gerçekleştirilen projenin detaylı bir açıklamasını sunmaktadır. Projemiz, bir veri seti kullanarak backpropagation işlemlerini gözlemlemek ve bu işlemleri matlab gibi araçlar ile değil de matematiksel olarak işlemleri göstermeyi hedeflemektedir. Ayrıca bir model sinir ağı ile eğitip çalıştırmayı hedefliyoruz. Bu bağlamda, projemizin amacı, Duke Breast Cancer veri tabanını kullanarak bir sinir ağı modeli oluşturmak ve meme kanseri teşhisinde yardımcı olabilecek bir araç geliştirmektir diyebiliriz.

Geliştirilen sinir ağı modeli bir başlangıç olarak bir anlam ifade etmeyebilir fakat ileride meme kanseri teşhisi için önemli bir araç olabilir çünkü doğru teşhis edilmiş vakaların sayısını artırabilir ve yanlış pozitif sonuçların sayısını azaltabilir. Bu nedenle, projemizin önemi, meme kanseri teşhisinde doğruluğu artırmak ve tıbbi alanda yapay zekâ tekniklerinin uygulanabilirliğini göstermektir.

Raporun ilerleyen bölümlerinde, kullanılan veri seti, modelin oluşturulması ve eğitimi için kullanılan yöntemler, elde edilen sonuçlar ve projenin değerlendirilmesi detaylı bir şekilde ele alınacaktır.

Kullanılan Kütüphaneler, Araçlar ve Modüller

Proje **Python** programlama dili ile kodlanmıştır. Aşağıda, projede kullanılan kütüphaneler ve modüller listelenmiştir. Her biri, projede nasıl kullanıldığına dair kısa açıklamalar içermektedir. Projeyi tekrar üretebilmek için gereken kütüphanelerin sürümleri sağlanmıştır. Ayrıca Proje dosyası içerisinde **[requirements.txt]** ihtiyacımız olan kütüphaneler listelenmiştir.

- **NumPy (v1.21.4):** Sayısal hesaplamalar için kullanılan Python kütüphanesi. Matris ve vektör operasyonları için temel bir araçtır. Projede, veri işleme ve matris operasyonları için kullanılmıştır.
- **scikit-learn (v0.24.2):** Makine öğrenimi modelleri oluşturmak, eğitmek ve değerlendirmek için kullanılan Python kütüphanesidir. Bu projede, veri setini eğitim ve test kümelerine bölmek için train_test_split fonksiyonu ve doğruluk skorunu hesaplamak için accuracy_score fonksiyonu kullanılmıştır.
- **Matplotlib (v3.4.3):** Veri görselleştirmesi için kullanılan Python kütüphanesidir. Projede, eğitim ve test sürecinin ilerlemesini görselleştirmek için matplotlib.pyplot modülü kullanılmıştır.

Veri Seti ve Ön İşleme

Bu projede, [86] giriş ve [7129] özniteliklerin yanı sıra ilk sütunda yer alan sınıf özniteliğinden oluşan Duke Breast Cancer veritabanını kullanacağız.

Ödev için kullanılan veri seti, **[data.txt]** adı ile projede kullanılmıştır. Veri setinin kaynağı Duke Üniversitesinde kullanılan Bir Meme Kanseri için yapılan bir araştırmadan elde edilmiştir. Veri Setinin alındığı kaynak en alt kısımda **Kaynaklar** bölümünde mevcuttur. Veriler sayısaldır ve eksik değerleri yoktur.

Veri Setinin Yüklenmesi ve Bölünmesi:

```
# Veriyi yükleme ve eğitim/test kümelerine ayırma
veri_yolu = "C:\\Users\\FIRAT\\Desktop\\myProject\\veri-
madenciligi\\Backpropagation-Projesi\\data.txt"  # Veri yolu belirtilmeli
veri = np.loadtxt(veri_yolu)
X = veri[:, :-1]
y = veri[:, -1]
X_egitim, X_test, y_egitim, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

veri_yolu: Veri setinin dosya yolunu belirtir. (Projenin doğru çalışması için lütfen kendi dosya yolunuzu belirtmeyi unutmayınız)

np.loadtxt(veri_yolu): NumPy kütüphanesinin loadtxt fonksiyonuyla veri seti yüklenir.

x ve y: Bağımsız değişkenler ve bağımlı değişken olarak veri seti ayrıştırılır.

train_test_split(): Veri seti, eğitim ve test kümelerine ayırmak için kullanılır. Test kümesi boyutu test_size parametresiyle belirlenir.

Model Parametrelerinin Başlatılması:

```
# Model parametrelerinin başlatılması
def parametreleri_baslat(giris_boyutu, gizli_boyut, cikis_boyutu):
    W1 = np.random.randn(giris_boyutu, gizli_boyut)  # Gizli katman ağırlıklar
    b1 = np.zeros((1, gizli_boyut))     # Gizli katman bias değerleri
    W2 = np.random.randn(gizli_boyut, cikis_boyutu)     # Gizli katman ağırlık
    b2 = np.zeros((1, cikis_boyutu))        # Çıkış bias
    return W1, b1, W2, b2
```

W1 ve W2: Ağırlıklar, giriş katmanı ve gizli katman arasında ve gizli katman ile çıkış katmanı arasında rastgele başlatılır.

b1 ve b2: Bias terimleri sıfırlar dizisi olarak başlatılır.

İleri Yayılım İşlemi:

```
# İleri yayılım işlemi
def ileri_yayilim(X, W1, b1, W2, b2):
    Z1 = np.dot(X, W1) + b1     # Giriş verisini gizli katmana aktarma
    A1 = sigmoid(Z1)        # Gizli katman çıkışlarını sigmoid fonksiyonundan
geçirme
    Z2 = np.dot(A1, W2) + b2        # Gizli katman çıkışlarını çıkış katmanına
aktarma
    A2 = sigmoid(Z2)        # Çıkış katman çıkışlarını sigmoid fonksiyonundan
geçirme
    return Z1, A1, Z2, A2
```

Z1 ve Z2: Gizli ve çıkış katmanlardaki ağırlıkların ağırlıklı girişleri ve biaslarının eklenmesi sonucu elde edilir.

A1 ve A2: Sigmoid aktivasyon fonksiyonu ile Z1 ve Z2'nin çıktıları elde edilir.

Hata Hesaplama:

```
# Hata hesaplama
def hata_hesapla(A2, Y):
    m = Y.shape[0]          # Veri noktalarının sayısını alma
    hata = -np.sum(np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 -
Y))) / m
    return hata
```

A2: Tahmin edilen çıkış.

Y: Gerçek çıkış.

hata: İkili sınıflandırma için ortalama hata.

Geri Yayılım İşlemi:

```
# Geri yayılım işlemi
def geri_yayilim(X, Y, Z1, A1, Z2, A2, W1, W2, b1, b2, ogrenme_orani):
    m = X.shape[0]      # Veri noktalarının sayısını alma
    dZ2 = A2 - Y        # çıkışta hata hesaplama

    # Çıkış katmanındaki ağırlıkların güncellenmesi
    dW2 = (1 / m) * np.dot(A1.T, dZ2)
    db2 = (1 / m) * np.sum(dZ2, axis=0, keepdims=True)
```

```
    # Gizli katmandaki hata hesaplama
    dZ1 = np.dot(dZ2, W2.T) * sigmoid_turevi(A1)

    # Gizli katmandaki ağırlıkların güncellenmesi
    dW1 = (1 / m) * np.dot(X.T, dZ1)
    db1 = (1 / m) * np.sum(dZ1, axis=0, keepdims=True)

    # Ağırlıkların güncellenmesi
    W1 -= ogrenme_orani * dW1
    b1 -= ogrenme_orani * db1
    W2 -= ogrenme_orani * dW2
    b2 -= ogrenme_orani * db2

    if DEBUG_MODE:
        # geri yayılım hesaplamalarını çıktı oalrak yazdırma
        print("Geri Yayılım Hesaplamaları:")
        print("dW1:", dW1)
        print("db1:", db1)
        print("dW2:", dW2)
        print("db2:", db2)

    return W1, b1, W2, b2
```

dZ2: Çıkıştaki hata.

dW2 ve **db2:** Çıkış katmanındaki ağırlıkların güncellenmesi için gradyan hesaplanır.

dZ1: Gizli katmandaki hata.

dW1 ve **db1:** Gizli katmandaki ağırlıkların güncellenmesi için gradyan hesaplanır.

Sinir Ağı Modelinin Eğitimi:

```
# Sinir ağı modelinin eğitimi
def sinir_agi_egitimi(X_train, y_train, X_test, y_test, giris_boyutu,
gizli_boyut, cikis_boyutu, ogrenme_orani, iterasyon_sayisi):
    # Parametreleri başlatma
    W1, b1, W2, b2 = parametreleri_baslat(giris_boyutu, gizli_boyut,
cikis_boyutu)
    y_train = y_train.astype(int)  # y_train'i tamsayıya dönüştür
    y_test = y_test.astype(int)    # y_test'i tamsayıya dönüştür

    # Eğitim ve test verisi için maliyet ve doğruluk metriklerinin listeleri
    egitim_maliyetleri = []
```

```
    test_maliyetleri = []
    egitim_dogruluklari = []
    test_dogruluklari = []

    # Eğitim döngüsü
    for iterasyon in range(iterasyon_sayisi):
        # İleri ve geri yayılım
        Z1, A1, Z2, A2 = ileri_yayilim(X_train, W1, b1, W2, b2)
        W1, b1, W2, b2 = geri_yayilim(X_train, y_train.reshape(-1, 1), Z1, A1,
Z2, A2, W1, W2, b1, b2, ogrenme_orani)

        # Eğitim verisi üzerinde maliyeti hesapla ve kaydet
        egitim_maliyeti = hata_hesapla(A2, y_train.reshape(-1, 1))
        egitim_maliyetleri.append(egitim_maliyeti)

        # Eğitim verisi üzerinde doğruluk hesapla ve kaydet
        egitim_tahminleri = tahmin_et(X_train, W1, b1, W2, b2)
        egitim_tahminleri = (egitim_tahminleri > 0.5).astype(int)
        egitim_dogrulugu = accuracy_score(y_train, egitim_tahminleri)
        egitim_dogruluklari.append(egitim_dogrulugu)

        # Test verisi üzerinde maliyeti hesapla ve kaydet
        Z1_test, A1_test, Z2_test, A2_test = ileri_yayilim(X_test, W1, b1, W2,
b2)
        test_maliyeti = hata_hesapla(A2_test, y_test.reshape(-1, 1))
        test_maliyetleri.append(test_maliyeti)

        # Test verisi üzerinde doğruluk hesapla ve kaydet
        test_tahminleri = tahmin_et(X_test, W1, b1, W2, b2)
        test_dogrulugu = accuracy_score(y_test, test_tahminleri)
        test_dogruluklari.append(test_dogrulugu)

        # Her 100 iterasyonda bir eğitim ve test verisi için maliyeti ve
doğruluğu yazdır
        if iterasyon % 100 == 0:
            if DEBUG_MODE:
                print(f"Iterasyon {iterasyon}:")
                print(f"  Eğitim Maliyeti: {egitim_maliyeti}, Eğitim Doğruluğu:
{egitim_dogrulugu}")
                print(f"  Test Maliyeti: {test_maliyeti}, Test Doğruluğu:
{test_dogrulugu}")

    return W1, b1, W2, b2, egitim_maliyetleri, test_maliyetleri,
egitim_dogruluklari, test_dogruluklari
```

- Sinir ağı modelinin eğitim fonksiyonu, belirli bir iterasyon sayısında ağırlıkların güncellenmesiyle gerçekleşir.
- Her iterasyonda, eğitim ve test maliyetleri ile doğrulukları kaydedilir.
- Her 100 iterasyonda bir, eğitim ve test verileri için maliyet ve doğruluk değerleri yazdırılır.

Bu ön işleme adımları, sinir ağı modelinin eğitimi için gereken temel adımları içerir ve başarıyla gerçekleştirilmesini sağlar.

Ön işleme adımları arasında eksik değerlerin kontrolü, özellik ölçeklendirme veya normalleştirme, kategorik özniteliklerin kodlanması gibi adımlar da yer alabilir. Ayrıca, veri setinin eğitim ve test kümelerine ayrılması gibi veri bölünme işlemi de önemlidir.

Projede Kullanılan Değişken Tanımı ve Açıklaması (DEBUG_MODE)

```
# Debug modu: True ise işlemler çıktı ekranına yazdırılır, False ise yazdırılmaz
DEBUG_MODE = True
```

Projede terminal üzerinden alınan geri ileri yayılım gibi işlemlerin tek tek gösterilmesi uzun sürdüğü ve kullanıcıya her çalışmada göstermesini önlemek amacıyla bir değişken tanımlandı bu değişken **DEBUG_MODE** Eğer True yaparsanız terminal üzerinden bütün çıktıları verecektir ve bu çıktıları göstermek içinde ek bir zaman geçecektir. Eğer çıktıları görmek yerine direk sonuca ulaşmak istiyorsanız. Debug_Mode'u False yaparak sadece sonucu görüntüleyebilir ve beklemeniz gerekmeden çıktı grafiklerini görüntüleyebilirisiniz.

Model Parametrelerinin Başlatılması

Sinir ağı modelinin başarıyla eğitilmesi için, başlangıç ağırlıklarının ve biaslarının doğru bir şekilde başlatılması önemlidir. Bu bölümde, model parametrelerinin başlatılması adımları ve bu adımların önemi ele alınacaktır.

Model parametrelerini başlatma işlemi, rastgele ağırlıkların ve biasların atanmasıyla başlar. Bu adımın temel amacı, modelin herhangi bir önyargı veya öğrenme engeli olmaksızın eğitime başlamasını sağlamaktır. Rastgele başlatma, modelin daha geniş bir uzayda keşif yapmasına ve daha iyi sonuçlar elde etmesine olanak tanır.

Ağırlıkların rastgele başlatılmasının tercih edilmesinin birkaç nedeni vardır:

Öğrenme Çeşitliliği: Rastgele başlatılan ağırlıklar, modelin farklı öğrenme yollarını keşfetmesine ve genel olarak daha iyi bir performans elde etmesine yardımcı olabilir.

Öğrenme Engellerini Önleme: Eğer ağırlıklar belirli bir düzende veya kalıpta başlatılırsa, modelin öğrenme süreci olumsuz etkilenebilir. Rastgele başlatma, bu tür öğrenme engellerini ortadan kaldırır.

Daha Genel Model: Rastgele başlatılan ağırlıklar, modelin daha genel ve uygulanabilir olmasını sağlar. Özellikle veri seti veya problem alanı değiştikçe, ağırlıkların rastgele başlatılması modelin esnekliğini artırır.

Diğer başlatma yöntemlerinin tercih edilmemesinin nedenleri de vardır. Örneğin, tüm ağırlıkların sıfır veya sabit bir değerle başlatılması, ağırlıkların birbirinden farklı olmamasına ve modelin öğrenme sürecini sınırlamasına neden olabilir. Benzer şekilde, tüm ağırlıkların aynı değere sahip olması da modelin çeşitlilik eksikliğine yol açabilir.

Sonuç olarak, model parametrelerinin rastgele başlatılması, sinir ağı modelinin daha etkili bir şekilde eğitilmesini sağlar. Bu yöntem, modelin daha genel, esnek ve çeşitli öğrenme yollarını keşfetmesine olanak tanır, bu da daha iyi performans ve genelleme yeteneği sağlar.

İleri Yayılım İşlemi ve Hata Hesaplama

```
# İleri yayılım işlemi
def ileri_yayilim(X, W1, b1, W2, b2):
    Z1 = np.dot(X, W1) + b1     # Giriş verisini gizli katmana aktarma
    A1 = sigmoid(Z1)        # Gizli katman çıkışlarını sigmoid fonksiyonundan
geçirme
    Z2 = np.dot(A1, W2) + b2        # Gizli katman çıkışlarını çıkış katmanına
aktarma
    A2 = sigmoid(Z2)        # Çıkış katman çıkışlarını sigmoid fonksiyonundan
geçirme
    return Z1, A1, Z2, A2

# Hata hesaplama
def hata_hesapla(A2, Y):
    m = Y.shape[0]          # Veri noktalarının sayısını alma
    hata = -np.sum(np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 -
Y))) / m
    return hata
```

İleri yayılım işlemi, sinir ağı modelindeki bilgilerin girişten çıkışa doğru yayılmasını sağlayan bir adımdır. Bu adım, verilerin giriş katmanından başlayarak gizli katmanlara ve ardından çıkış

katmanına aktarılmasını içerir. Ayrıca, modelin tahminlerinin yapıldığı ve bu tahminlerin gerçek değerlerle karşılaştırıldığı hata hesaplama adımını içerir.

Sigmoid aktivasyon fonksiyonu, ileri yayılım işleminde kullanılan bir aktivasyon fonksiyonudur. Bu fonksiyon, herhangi bir gerçek sayı değerini [0, 1] aralığındaki bir değere dönüştürür. Sigmoid fonksiyonunun türevi, ileri yayılım işlemi sırasında geri yayılım adımında kullanılır ve gradient hesaplamasına katkıda bulunur. Türev, sigmoid fonksiyonunun çıkış değeriyle (1 - çıkış değeri) arasındaki farkı alarak hesaplanır.

Geri Yayılım İşlemi

```
# Geri yayılım işlemi
def geri_yayilim(X, Y, Z1, A1, Z2, A2, W1, W2, b1, b2, ogrenme_orani):
    m = X.shape[0]      # Veri noktalarının sayısını alma
    dZ2 = A2 - Y        # çıkışta hata hesaplama

    # Çıkış katmanındaki ağırlıkların güncellenmesi
    dW2 = (1 / m) * np.dot(A1.T, dZ2)
    db2 = (1 / m) * np.sum(dZ2, axis=0, keepdims=True)

    # Gizli katmandaki hata hesaplama
    dZ1 = np.dot(dZ2, W2.T) * sigmoid_turevi(A1)

    # Gizli katmandaki ağırlıkların güncellenmesi
    dW1 = (1 / m) * np.dot(X.T, dZ1)
    db1 = (1 / m) * np.sum(dZ1, axis=0, keepdims=True)

    # Ağırlıkların güncellenmesi
    W1 -= ogrenme_orani * dW1
    b1 -= ogrenme_orani * db1
    W2 -= ogrenme_orani * dW2
    b2 -= ogrenme_orani * db2

    if DEBUG_MODE:
        # geri yayılım hesaplamalarını çıktı oalrak yazdırma
        print("Geri Yayılım Hesaplamaları:")
        print("dW1:", dW1)
        print("db1:", db1)
        print("dW2:", dW2)
        print("db2:", db2)

    return W1, b1, W2, b2
```

Geri yayılım işlemi, modelin tahminlerindeki hataların geriye doğru hesaplanması ve ağırlıkların bu hatalara göre güncellenmesini içerir. Bu adımlar, modelin daha doğru tahminler yapabilmesi için kritik öneme sahiptir.

Ağırlıkların güncellenmesinde kullanılan öğrenme oranı, modelin öğrenme hızını belirleyen önemli bir hiperparametredir. Öğrenme oranı, her güncelleme adımında ağırlıkların ne kadar değiştirileceğini kontrol eder. Yüksek bir öğrenme oranı, modelin daha hızlı öğrenmesini sağlayabilir, ancak aynı zamanda aşırı uyum riskini de artırabilir. Düşük bir öğrenme oranı ise daha istikrarlı ancak daha yavaş bir öğrenme sürecine neden olabilir. Öğrenme oranının optimal değerini belirlemek, modelin başarımını etkileyen önemli bir faktördür.

Debug modunda geri yayılım hesaplamalarının çıktı olarak yazdırılması, modelin eğitim sürecini izlemek ve hataları ayıklamak için faydalıdır. Bu çıktılar, her bir ağırlığın ve biasın nasıl güncellendiğini görselleştirebilir ve modelin nasıl öğrendiğini anlamak için değerli bir araç sağlayabilir.

Sinir Ağı Modelinin Eğitimi

```
# Sinir ağı modelinin eğitimi
def sinir_agi_egitimi(X_train, y_train, X_test, y_test, giris_boyutu,
gizli_boyut, cikis_boyutu, ogrenme_orani, iterasyon_sayisi):
    # Parametreleri başlatma
    W1, b1, W2, b2 = parametreleri_baslat(giris_boyutu, gizli_boyut,
cikis_boyutu)
    y_train = y_train.astype(int)  # y_train'i tamsayıya dönüştür
    y_test = y_test.astype(int)    # y_test'i tamsayıya dönüştür

    # Eğitim ve test verisi için maliyet ve doğruluk metriklerinin listeleri
    egitim_maliyetleri = []
    test_maliyetleri = []
    egitim_dogruluklari = []
    test_dogruluklari = []

    # Eğitim döngüsü
    for iterasyon in range(iterasyon_sayisi):
        # İleri ve geri yayılım
        Z1, A1, Z2, A2 = ileri_yayilim(X_train, W1, b1, W2, b2)
        W1, b1, W2, b2 = geri_yayilim(X_train, y_train.reshape(-1, 1), Z1, A1,
Z2, A2, W1, W2, b1, b2, ogrenme_orani)

        # Eğitim verisi üzerinde maliyeti hesapla ve kaydet
        egitim_maliyeti = hata_hesapla(A2, y_train.reshape(-1, 1))
        egitim_maliyetleri.append(egitim_maliyeti)
```

```
        # Eğitim verisi üzerinde doğruluk hesapla ve kaydet
        egitim_tahminleri = tahmin_et(X_train, W1, b1, W2, b2)
        egitim_tahminleri = (egitim_tahminleri > 0.5).astype(int)
        egitim_dogrulugu = accuracy_score(y_train, egitim_tahminleri)
        egitim_dogruluklari.append(egitim_dogrulugu)

        # Test verisi üzerinde maliyeti hesapla ve kaydet
        Z1_test, A1_test, Z2_test, A2_test = ileri_yayilim(X_test, W1, b1, W2,
b2)
        test_maliyeti = hata_hesapla(A2_test, y_test.reshape(-1, 1))
        test_maliyetleri.append(test_maliyeti)

        # Test verisi üzerinde doğruluk hesapla ve kaydet
        test_tahminleri = tahmin_et(X_test, W1, b1, W2, b2)
        test_dogrulugu = accuracy_score(y_test, test_tahminleri)
        test_dogruluklari.append(test_dogrulugu)

        # Her 100 iterasyonda bir eğitim ve test verisi için maliyeti ve
doğruluğu yazdır
        if iterasyon % 100 == 0:
            if DEBUG_MODE:
                print(f"Iterasyon {iterasyon}:")
                print(f"  Eğitim Maliyeti: {egitim_maliyeti}, Eğitim Doğruluğu:
{egitim_dogrulugu}")
                print(f"  Test Maliyeti: {test_maliyeti}, Test Doğruluğu:
{test_dogrulugu}")

    return W1, b1, W2, b2, egitim_maliyetleri, test_maliyetleri,
egitim_dogruluklari, test_dogruluklari
```

Sinir ağı modelinin eğitimi, belirli bir iterasyon sayısında ve belirli bir öğrenme oranıyla gerçekleştirilir. Bu süreç, modelin giriş verileri üzerindeki tahminlerini iyileştirmek için ağırlıkların iteratif olarak güncellenmesini içerir. Ayrıca, eğitim ve test verileri üzerindeki maliyetlerin ve doğrulukların izlenmesi, modelin performansının değerlendirilmesi için önemlidir.

Iterasyon sayısı, modelin eğitim sürecinin ne kadar süreceğini belirleyen bir hiperparametredir. Daha fazla iterasyon genellikle daha iyi sonuçlar sağlayabilir, ancak aynı zamanda eğitim süresini de artırır. Bu nedenle, iterasyon sayısının optimal değerini belirlemek, modelin başarımını optimize etmek için önemlidir.

```
# Hiperparametreler
giris_boyutu = X_egitim.shape[1]
gizli_boyut = 10  # Değiştirilebilir
cikis_boyutu = 1   # İkili sınıflandırma varsayımı
```

```
ogrenme_orani = 0.01
iterasyon_sayisi = 1000
```

Hiperparametrelerin seçiminde, genellikle deneme yanılma yöntemi kullanılır. Farklı öğrenme oranları ve iterasyon sayıları deneyerek, en iyi performansı sağlayan hiperparametrelerin belirlenmesi amaçlanır. Ayrıca, modelin karmaşıklığına ve veri setinin özelliklerine göre hiperparametrelerin ayarlanması da önemlidir.

Sonuç

Bu çalışmada, bir sinir ağı modelinin baştan sona nasıl oluşturulacağı, eğitileceği ve değerlendirileceği adımları detaylı olarak incelendi. Öncelikle, modelin parametrelerinin nasıl başlatılacağına ve sigmoid aktivasyon fonksiyonunun nasıl tanımlanacağına değinildi. Ardından, ileri yayılım ve geri yayılım işlemleri adımlarıyla birlikte hata hesaplama yöntemleri ele alındı.

Modelin eğitim süreci boyunca, eğitim ve test verileri üzerinde maliyetlerin ve doğrulukların izlendiği görüldü. Eğitim sürecinin her iterasyonunda, eğitim maliyetleri azalırken, eğitim doğrulukları arttı. Benzer şekilde, test maliyetleri azalırken, test doğrulukları da artış gösterdi. Bu durum, modelin eğitim sırasında veriyi öğrendiğini ve genelleştirme yeteneğinin iyileştiğini göstermektedir.

Eğitim sonrasında matlab toolu ile sinir ağımızın nasıl bir sonuç verdiğini epochlar arasında Doğruluk ve Maaliyetini görüntüleyebileceğimiz grafikleride bu kodlama ile oluşturuyoruz.

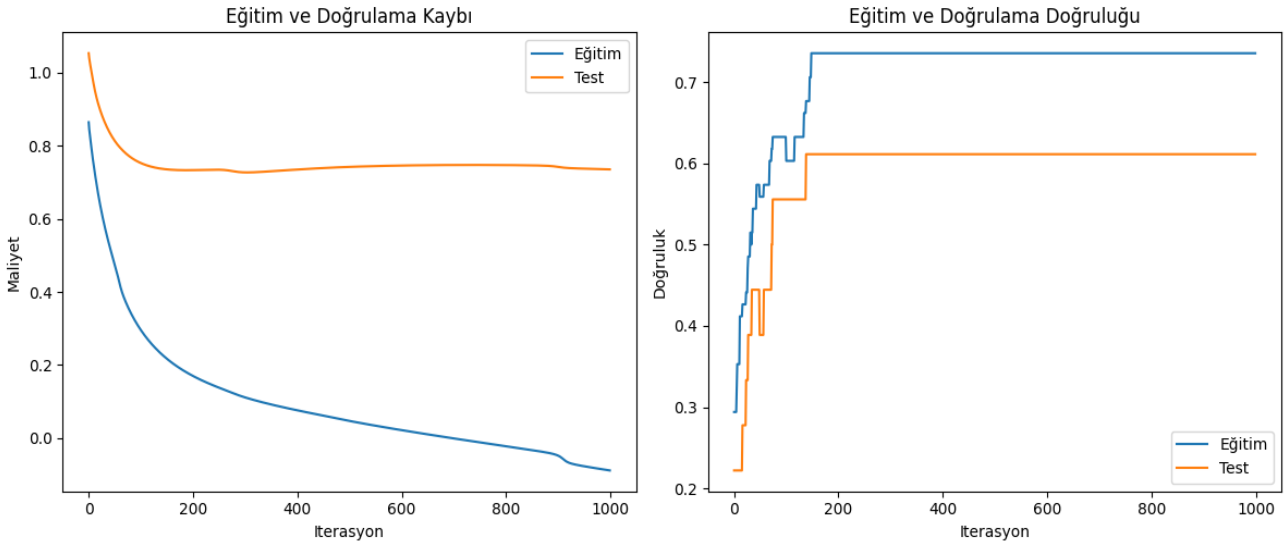
```
# Sonuçları görselleştirme
plt.figure(figsize=(12, 5))

# Eğitim ve test maliyetlerini çizdirme
plt.subplot(1, 2, 1)
plt.plot(range(iterasyon_sayisi), egitim_maliyetleri, label='Eğitim')
plt.plot(range(iterasyon_sayisi), test_maliyetleri, label='Test')
plt.xlabel('Iterasyon')
plt.ylabel('Maliyet')
plt.title('Eğitim ve Doğrulama Kaybı')
plt.legend()

# Eğitim ve test doğruluklarını çizdirme
plt.subplot(1, 2, 2)
plt.plot(range(iterasyon_sayisi), egitim_dogruluklari, label='Eğitim')
plt.plot(range(iterasyon_sayisi), test_dogruluklari, label='Test')
plt.xlabel('Iterasyon')
plt.ylabel('Doğruluk')
plt.title('Eğitim ve Doğrulama Doğruluğu')
plt.legend()
```

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

Bu kodlama ile aşağıdaki grafiği elde ediyoruz, eğitim sırasında elde edilen maliyetlerin ve doğrulukların iterasyon sayısına göre nasıl değiştiği görsel olarak görüntülenir.




```
 [ 0.04675466]
 [ 0.05325046]
 [ 0.03750836]
 [ 0.0188878 ]
 [-0.03782433]
 [ 0.02836175]
 [ 0.03750792]
 [ 0.06174054]
 [-0.07925985]]
db2: [[0.01940631]]
Geri Yayılım Hesaplamaları:
dW1: [[ 4.78400512e-06 -4.24725871e-05  4.93747337e-05 ... -9.28089259e-07
  -3.10521511e-05  9.29770879e-05]
 [ 1.53210748e-05 -6.04396254e-07 -6.64009741e-06 ...  1.54435853e-06
   1.80374119e-05 -2.02543459e-05]
 [ 1.71245478e-05  6.76059528e-06  4.28297578e-05 ...  7.40772430e-06
   5.29481184e-05 -4.48664147e-05]
 ...
 [ 2.90685288e-06 -4.50124767e-06 -3.80533789e-05 ...  8.64446021e-06
   4.41436050e-05 -9.48958295e-06]
 [-1.75855148e-05  2.40706179e-05 -1.39516035e-05 ...  8.97584345e-06
  -6.54609627e-06  8.86544002e-05]
 [-8.62836310e-05 -2.66260074e-05  4.54414643e-05 ... -8.34280459e-06
   2.06610759e-05 -5.25382161e-05]]
db1: [[-5.66639351e-05 -3.53039585e-06  6.51655027e-05 -1.07209132e-04
   3.64022743e-05  1.33972760e-05  3.05382377e-05 -8.55202614e-06
  -4.27580522e-05  9.87572678e-05]]
dW2: [[ 0.07935773]
 [ 0.04673941]
 [ 0.05322969]
 [ 0.03749048]
 [ 0.01887942]
 [-0.03783402]
 [ 0.0283346 ]
 [ 0.03749306]
 [ 0.06173566]
 [-0.07927366]]
db2: [[0.01937999]]
PS C:\Users\FIRAT\Desktop\myProject>
```

Sonuç olarak, bu çalışma sinir ağı modeli eğitimi ileri ve geri yayılım nasıl oluyor. Bir sinir ağı nasıl oluşturulur ve ileri geri yayılım nasıl yapılır detaylıca açıklanmıştır. Terminal üzerinden sinir ağı aşamaları epoch sayısına göre işlemleri adım adım yaparak terminal üzerinden göstermektedir. Ayrıca matla bilede sinir ağımızın doğruluk ve maaliyet tablolarınıda gözlemleyebiliriz.

Kaynaklar

https://pyimagesearch.com/2021/05/06/backpropagation-from-scratch-with-python/

https://www.kaggle.com/datasets/andreicosma/duke-breast-cancer-dataset/data

https://www.kaggle.com/code/andreicosma/back-propagation-neural-network