**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.

**İleri Yayılım (Forward Propagation)**

İleri yayılım, sinir ağının girdilerden başlayarak çıktılara doğru ilerleyen bir süreçtir. Bu süreçte, her katmandaki nöronlar, girdileri ve ağırlıkları kullanarak çıktı üretirler. İleri yayılım aşağıdaki adımları izler:

Girdi Katmanı: İlk adımda, sinir ağının girdi katmanına gelen veri işlenir.

Gizli Katman(lar): Girdi katmanından sonra gelen gizli katmanlar, ağırlıklı girdi toplamlarını ve aktivasyon fonksiyonlarını hesaplarlar.

Çıktı Katmanı: Son olarak, gizli katmanlardan geçen bilgiler çıktı katmanında işlenir ve ağın nihai çıktısı üretilir.

Matematiksel olarak, ileri yayılım aşağıdaki formüllerle ifade edilebilir:

z(l)=W(l)a(l−1)+b(l)

a(l)=f(z(l))

Burada z(l), l. katmandaki nöronların ağırlıklı girdi toplamını, a(l), l. katmandaki nöronların aktivasyon fonksiyonundan geçmiş çıktısını, W(l), l. katmandaki nöronların ağırlıklarını, b(l), l. katmandaki nöronların bias'ını, ve f, aktivasyon fonksiyonunu temsil eder.

**Aktivasyon Fonksiyonu**

Yapay Sinir Ağları, giriş katmanı, gizli katman(lar) ve çıkış katmanından oluşur. Her katmanda bir veya daha fazla düğüm bulunur ve bu düğümler birbirleriyle bağlantılıdır. Bu bağlantılar ağırlıklarla temsil edilir. Ağ, giriş verisini alır, bu ağırlıklarla işlemler yapar ve bir çıkış üretir. Bu işlemler genellikle aktivasyon fonksiyonları kullanılarak gerçekleştirilir. Bu algoritmada kullanılan aktivasyon fonksiyonu genellikle sigmoid fonksiyonudur. Sigmoid fonksiyonu, herhangi bir gerçek sayıyı (−∞, +∞) aralığındaki bir değere dönüştürerek 0 ile 1 arasında bir değer üretir. Bu, çıkışı olasılık değerlerine dönüştürmek için idealdir.

**Hata Hesaplama**

Hata hesaplama, ağın performansını değerlendirmek için kullanılan bir ölçüdür. Genellikle, ortalama kare hata gibi bir fonksiyon kullanılarak gerçek ve tahmin edilen çıktılar arasındaki fark hesaplanır.

**Geri Yayılım (Backward Propagation)**

Geri yayılım, ağın çıktı katmanından başlayarak hata geriye doğru yayıldığı ve ağırlıkların güncellendiği süreci ifade eder. Bu süreç, ağın tahminlerindeki hata ile başlar ve bu hatayı minimize etmek için ağın içinde geriye doğru yayılır. Geri yayılımın temel adımları şunlardır:

Hata Hesaplama: Öncelikle, ağın çıktıları ile gerçek değerler arasındaki fark hesaplanır. Bu fark genellikle bir hata fonksiyonu kullanılarak belirlenir.

Geri Yayılım: Hatanın kökenine doğru geriye doğru yayılır ve her katmandaki nöronlara ait ağırlıkların ve bias'ların hata katkıları belirlenir.

Ağırlık Güncelleme: Elde edilen hata katkıları, gradyan inişi gibi bir optimizasyon algoritması kullanılarak ağırlıkların güncellenmesi için kullanılır.

Geri yayılım formülleri şu şekildedir:

∂z(l)∂J​=∂a(l)∂J​∂z(l)∂a(l)​

∂W(l)∂J​=m1​∂z(l)∂J​a(l−1)T

∂b(l)∂J​=m1​∑i=1m​∂z(l)∂J​

∂a(l−1)∂J​=W(l)T∂z(l)∂J​

Burada, ∂z(l)∂J​ l. katmandaki hata, ∂W(l)∂J​ l. katmandaki ağırlıkların türevi, ∂b(l)∂J​ l. katmandaki bias'ların türevi, ∂a(l−1)∂J​ l. katmanın öncesindeki katmandaki hata ve T transpozunu temsil eder.

**Geri Yayılım adımları şu şekildedir:**

* İleri Yayılım: Veri, ağa verilir ve her katmanda ağırlıklarla işlemler yaparak bir tahmin üretilir.
* Hata Hesaplama: Elde edilen tahmin ile gerçek değer arasındaki hatayı hesaplamak için bir kayıp fonksiyonu kullanılır.
* Geri Yayılım: Hata, ağı geriye doğru yayılarak, ağırlıkların güncellenmesi için kullanılır.

**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.

metin, çizgi, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

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 matlab kütüphaneleri ile sinir ağımızın doğruluk ve maaliyet tablolarını da 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