

KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Nöron Ağlarına Giriş

Yapay Sinir Ağları ile Diyabet Hastası Tespiti Modellenmesi

160201077 TUĞBA AYDEMİR

190201105 ALPEREN İLERİ

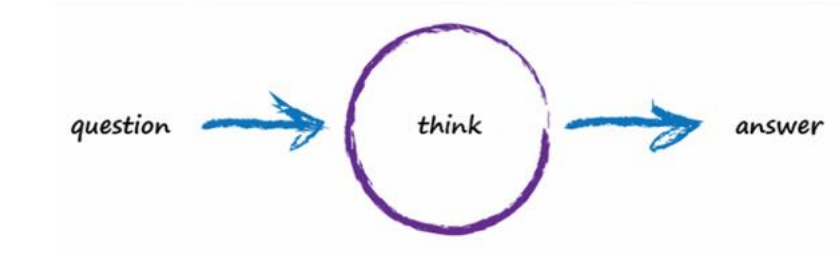
170201093 İLAYDA DIŞIÇIK

170201026 İSMAİL UTKU SAYAN

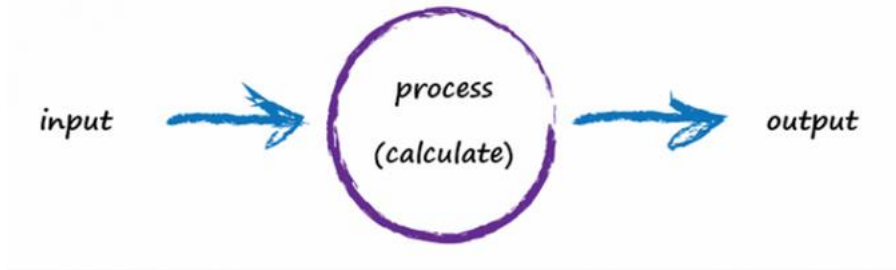
Yapay Sinir Ağları

Sinir Ağları (Neural Networks), en popüler makine öğrenimi algoritmalarından biridir (1). Yapay sinir ağının genel bir tanımının yapılması gerekirse; yapay sinir ağları, nöron adı verilen hücrelerle işlemleri gerçekleştiren ve insan beyninin çalışma şekli örnek alınarak geliştirilmiş bir ağ modelidir. (2) İlk nörobilgisayarın mucidi Dr. Robert Hecht-Nielsen ise bir sinir ağını şu şekilde tanımlar: "Bilgileri harici girdilere dinamik durum tepkileri ile işleyen, basit, birbiriyle yüksek oranda birbirine bağlı işlem öğelerinden oluşan bir bilgi işlem sistemi". (3) Yapay sinir ağları, bir başka deyişle, biyolojik sinir ağlarını taklit eden bilgisayar programlarıdır (4).

İnsan beyni dünyayı anlamaya çalışırken sorunlar ve var olan sorunlara cevap aramaktadır. Makinelerde bu şekilde çalışmaktadır. Makineye bir problem veriyoruz ve makinenin bu problemi çözmesini bekliyoruz. Örneğin bir tahmin makinesini düşünelim. Bu makineden, sisteme girilen soruların cevabını tahmin etmesi beklenmektedir.



Aynı şekilde bilgisayarlarında belli problemlere çözüm getimesi beklenmektedir (5).



Yapay sinir ağlarının dayandığı ilk hesaplama modelinin temelleri 1940'ların başında araştırmalarına başlayan sinir hekimi W.S. McCulloch ve matematikçi W.A. Pitts'in, 1943 yılında yayınladıkları bir makaleyle atılmış olmuştur. (6) Daha sonra 1954 yılında B.G. Farley ve W.A. Clark tarafından bir ağ içerisinde uyarılara tepki veren, uyarılara adapte olabilen model oluşturulmuştur. 1960 yılı ise ilk neural bilgisayarın ortaya çıkış yılıdır. 1963 yılında basit

modellerin ilk eksiklikleri fark edilmiş, ancak başarılı sonuçların alınması 1970 ve 1980’lerde termodinamikteki teorik yapıların doğrusal olmayan ağların geliştirilmesinde kullanılmasına kadar gecikmiştir. 1985 yapay sinir ağlarının oldukça tanındığı, yoğun araştırmaların başladığı yıl olmuştur (7).

1980’li yılların sonlarından başlamak üzere Yapay Sinir Ağları öngörümleme tekniği günümüzde birçok alanda yaygın bir şekilde kullanılmaya başlanmıştır. Yapay Sinir Ağları metodolojisi veriden öğrenebilme, genelleme yapabilme, sınırsız sayıda değişkenle çalışabilme vb birçok önemli özelliğe sahiptir. Bu özellikleri sayesinde oldukça önemli avantajlar sağlayan Yapay Sinir Ağları metodolojisi diğer alanlarda olduğu gibi öngörü modellemesi alanında da yaygın bir şekilde kullanılmaktadır. Yapay Sinir Ağları, girdi ve çıktı değişkenleri arasındaki herhangi bir ön bilgiye gereksinim duymadan doğrusal ve doğrusal olmayan modellemeyi sağlayabilmektedir (8).

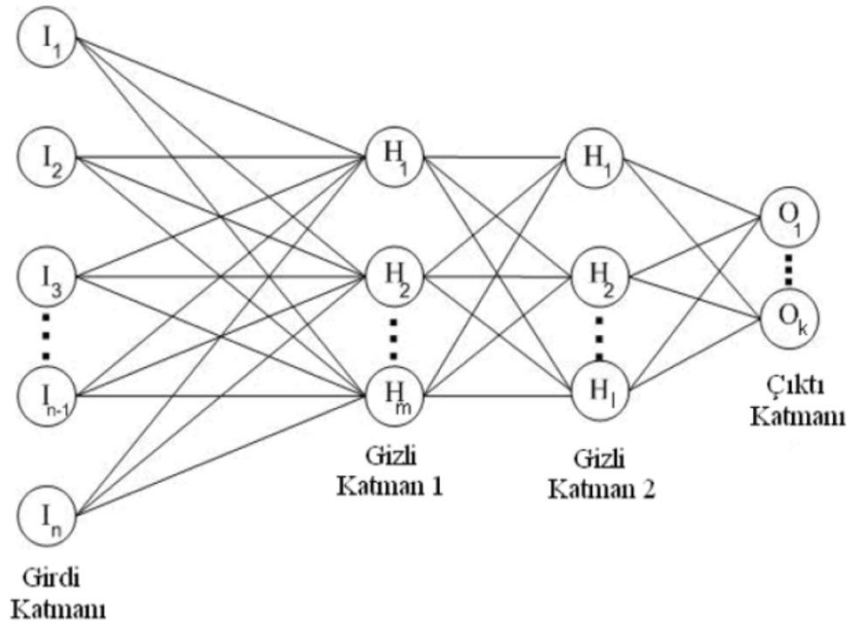
Yapay Sinir Ağlarının Bazı Avantajları

- Yapay sinir ağları bir çok hücreden meydana gelir ve bu hücreler eş zamanlı olarak çalışarak karmaşık işlemleri gerçekleştirir.
- Geleneksel yöntemlere göre daha hızlı ve literatüre göre daha başarılı tahmin(öngörü) yapabilmektedirler.
- Daha önce görülmemiş örnekler hakkında bilgi üretebilirler. YSA’lar eğitimleri sırasında kendilerine verilen örneklerden genellemeler çıkarırlar ve bu genellemeler ile yeni örnekler hakkında bilgi üretebilirler. (9)
- YSA’lar eğitildikten sonra veriler eksik bilgi içerse dahi, çıktı üretebilirler. Burada performans kaybı eksik bilginin önemine bağlıdır.
- YSA’ların bir ya da birden fazla hücrenin bozulması çıktı üretmesini engellemez. Bu özellik ağları hata toleransına sahip kılar. (10)

Gerçek hayatta yapay sinir ağlarının ortak uygulamalarına örnek olarak kalite kontrol, finansal tahmin, ekonomik tahmin, kredi notu, anormal durum tahmini, iflas tahmini gibi alanları (11) ve

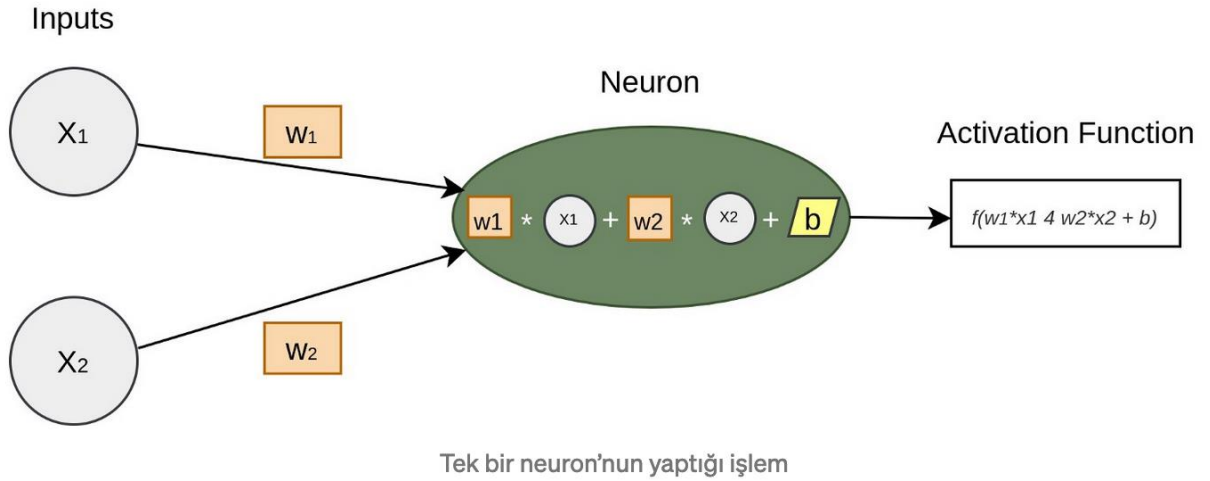
otomatik yüz tanıma, konuşma tanıma, doğal dil işleme, Go oyunu gibi birçok karmaşık problemlerin teknoloji ile çözümlenmesini (12) gösterebiliriz.

Basit bir ağ'da Girdi Katmanı, Gizli Katman ve Çıktı Katmanı bulunmaktadır. Girdi katmanı girdi verilerimizin bulunduğu katmandır. Girdi katmanında ağa girdi bilgisini veren girdi kümesi bulunur. Girdi katmanı ile çıktı katmanı arasında birden fazla ara katman bulunabilir. Bir katmandaki her nöron bir üst katmanda ki nörona çeşitli ağırlıklar ile bağlıdır. Girdi katmanındaki her bir veri ağırlık değeriyle çarpılarak gizli katmana iletilir. Sonuç gizli katmanda bir transfer fonksiyonundan geçirilir ve çıktı katmanına gönderilir. (13) Örneğin Boy ve Kilo verileri üzerinden cinsiyet tahmini yapan bir modelin eğitildiği bir ağda, Girdi Katmanında iki adet girdi (Boy ve Kilo), Gizli Katmanda her birinin ayrı ayrı boy ve kilo girdilerine bağlı olduğu iki nöron ve çıktı katmanında sonuç olarak cinsiyeti veren bir nöron bulunmaktadır (Gizli katman sayısı ve içerisindeki nöronlar isteğe bağlı olarak artırılabilir) (14).



Nöronlar **beslendiği** nörondan aldıkları değerleri ağırlıklandırarak **beslediği** nörona iletmektedirler. Son katman olan Çıktı katmanında yer alan nöronlar ise son ağırlıklandırma işlemini yaparak tahmin değerini çıktı olarak vermektedir. Elde edilen tahmin değeri gerçek değer ile karşılaştırılarak, ağırlıklandırma işlemi için kullanılan değerler elde edilen bu fark değerine göre güncellenir ve diğer girdi verileri ağ üzerinden geçer. **Bu işlem tahmin değeri gerçek değere en yakın olan noktaya gelinceye kadar veya belirtilen devir sayısı kadar devam eder.** Bu türde bir ağa İleri Beslenmeli ağ (Feedforward) denilmektedir (14). İleri beslenmeli sinir ağında, birimler arasındaki bağlantılar bir döngü oluşturmaz. Bu ağda bilgi, giriş düğümlerinden gizli düğümler üzerinden ve çıkış düğümlerine yalnızca bir yönde (ileri) hareket eder. Geliştirilen yapay sinir ağının ilk ve en basit türüdür. (15)

Nöronlar, içerisinde her bir bağlantının “Ağırlık” (W) değerini ve “Bias” (b) değeri barındırmaktadır. Nöronlar “Girdi Katmanı” üzerinden aldıkları değerleri o girdiye ait Ağırlık “W” değeri ile çarpmak ve ardından “bias” değeri ile toplamakla görevlilerdir. Burada “W”**ağırlık** değerleri o nöronun girdiler için katsayıları “b” **bias** değeri de nörondaki sabit sayı olarak düşünülebilir.












Yukarıdaki şekilde dikkat edildiğinde nöron her bir “W” değerini denk gelen girdi değeri ile çarparak toplamaktadır. Bu işlem sonrasında elde edilen değer “b” bias değeri toplanmaktadır. Buradan elde edilen sonuç girdilerin ağırlıklandırılmış toplam değerleridir. Aşağıdaki tabloda

kullanılabilecek toplayıcı fonksiyonlar verilmiştir. Bu fonksiyonlar içerisinde en yaygın olarak kullanılan fonksiyon “**Ağırlıklı Toplama**” dır (16).

Toplam $Net = \sum_{i=1}^N X_i * W_i$	Ağırlık değerleri girdiler ile çarpılır ve bulunan değerler birbirleriyle toplanarak Net girdi hesaplanır.
Çarpım $Net = \prod_{i=1}^N X_i * W_i$	Ağırlık değerleri girdiler ile çarpılır ve daha sonra bulunan değerler birbirleriyle çarpılarak Net Girdi Hesaplanır.
Maksimum $Net = \text{Max}(X_i * W_i)$	n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en büyüğü Net girdi olarak kabul edilir.
Minimum $Net = \text{Min}(X_i * W_i)$	n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en küçüğü Net girdi olarak kabul edilir.
Çoğunluk $Net = \sum_{i=1}^N \text{Sgn}(X_i * W_i)$	n adet girdi içinden girdilerle ağırlıklar çarpıldıktan sonra pozitif ile negatif olanların sayısı bulunur. Büyük olan sayı hücrenin net girdisi olarak kabul edilir.
Kümülatif Toplam $Net = \text{Net}(\text{eski}) + \sum_{i=1}^N X_i * W_i$	Hücreye gelen bilgiler ağırlıklı olarak toplanır. Daha önce hücreye gelen bilgilere yeni hesaplanan girdi değerleri eklenerek hücrenin net girdisi hesaplanır.

Nöron, son olarak bir aktivasyon fonksiyonu kullanarak elde edilen değeri 0 ile 1 arasında standartlaştırmaktadır. Aktivasyon fonksiyonunun çıkış değeri aynı zamanda nöronun çıkışıdır. Yapay sinir ağında kullanılacak olan aktivasyon fonksiyonunu belirlemede belirli bir kural yoktur. Veri setinin durumuna göre karar verilmektedir. (17) Bu sayede **kontROLSÜZ** veya **sınırsız (unbounded)** olarak gelen veri tahmin edilebilir bir değere dönüştürülmektedir. **Sigmoid** ve **Softmax** fonksiyonları en çok kullanılan aktivasyon fonksiyonlarındandır. Bununla birlikte aşağıda verilen aktivasyon fonksiyonları da kullanılmaktadır (18).

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

```
# Numpy kütüphanesi içeri aktarıyoruz
import numpy as np

# Nöron sınıfımızı oluşturuyoruz

class Noron:
    # def __init__ ile başlangıç ağırlık ve bias değerlerimizi sınıfımıza ekliyoruz

    def __init__(self , ağırlık,yanlilık):
        self.ağırlık=ağırlık
        self.yanlilık=yanlilık

    # Nöron içerisinde yapılan toplam işlemi sonrası
    # değerimizi 0 ile 1 arasında standartlaştırmak için
    # sigmoid metodumuzu oluşturuyoruz
    def sigmoid(self,x):
        return 1 / (1 + np.exp(-x))

    def ileriBesleme(self,veri):
        # Girdilerin ağırlıkla ve ardından bias ile toplama işlemi
        # En son olarak sonucu sigmoid fonksiyonundan geçirerek
        # 0 ile 1 arasında standartlaştırılmış sonuç değerini elde ediyoruz
        return self.sigmoid(np.dot(veri,self.ağırlık)+self.yanlilık)
```

```
x = [4,15]      # X1 = 4 , X2 = 5
w = [0,1]       # W1 = 0 , W2 = 1
b = 0           # Bias = 0

# Oluşturduğumuz class'ı kullanarak ağırlık ve
# bias değerleri ile Nöronu muz oluşturuyoruz
n = Noron(w ,b)

# İleri besleme yöntemi ile değerimizi elde ediyoruz
Tahmin = n. ileriBesleme(x)

# Sonucu yazdırıyoruz
print(Tahmin)
```

Tahmin: 0.999999694097773

```
#weight-->[1,0], bias=0, data=[4,15]
n2=Noron([1,0],0)
print('Tahmin:',n2.ileriBesleme([4,15]))
```

Tahmin: 0.9820137900379085


```
#weight-->[0,1], bias=0, data-->listem
listem=[[4,15],[2,3],[6,1],[-1,2]]
for i in listem:
    print(n1.ileriBesleme(i))
0.999999694097773
0.9525741268224334
0.7310585786300049
0.8807970779778823
```

```
n=Noron([0,1,3,5],0)
#weight-->[0,1,3,5], bias=0, data-->listem
listem=[[4,15,7,8],[2,3,-2,5],[6,1,0,6],[-1,2,-7,3]]
for i in listem:
    print(n.ileriBesleme(i))
1.0
0.9999999997210531
0.9999999999999656
0.01798620996209156
```

Elde edilen tahmin değeri ile gerçek değer arasındaki farkın standardize edilmesi için ise çeşitli metrikler kullanılmaktadır. Bu metriklerden elde edilen değerlere ise genel olarak “Loss” adı verilmektedir. “Loss” gerçek değer tahmin değerinden ne kadar farklı olduğunu göstermektedir. **MSE** (Mean Squared Error) yani Ortalama Standart Hata en çok kullanılan “Loss” metriklerin biridir.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_{true} - Y_{prediction})^2$$

Ortalama Standart Hata formülü

Ağın eğitilebilmesi için Loss miktarına göre Nöronlarda bulunan **W** ve **b** değerlerinin güncellenmesi ve ardında diğer gözlem girdilerinin ağ içeresinden geçirilip yeni tahmin değerleri ile tekrar güncelleme yapılması gerekmektedir. Bu işlem tahmin değeri ile gerçek değer arasındaki oran minimuma gelene kadar devam etmesi gerekir. Bu sayede yapay sinir ağıımızı

eğitmiş oluyoruz. Ağırlıkların ayrı ayrı nörondan elde edilen değerlere göre güncellenebilmesi için Loss değerinin her bir W ve b değerine olan etkisinin hesaplanması gerekmektedir. Bu işlemi ise Loss değerinin W ve b değerlerine göre kısmi türevinin alınmasıyla gerçekleştirebiliriz.

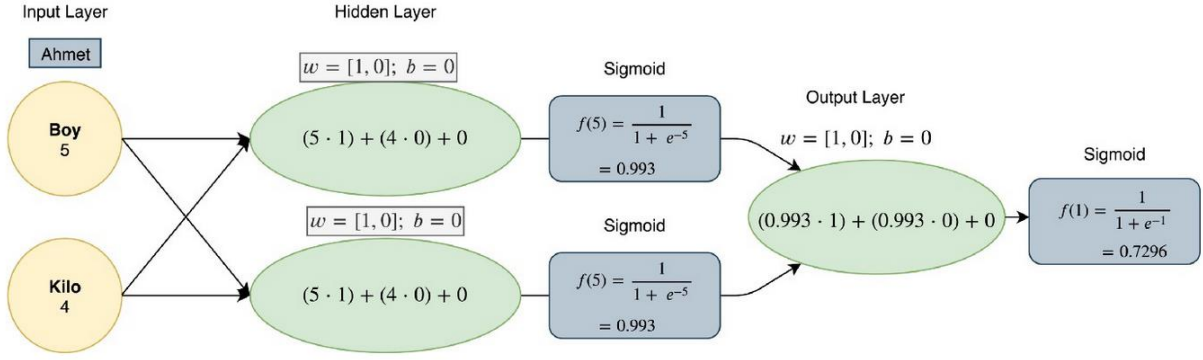
Aşağıdaki veri setini kullanarak bu durumu ifade edelim.

İsim	Boy	Kilo	Cinsiyet
Ahmet	165	75	E (0)
Veli	175	76	E (0)
Nazan	154	45	K (1)
Elif	136	56	K (1)
Kamil	145	70	E (0)
Suzan	167	56	K (1)
...

Bu veri setinde öğrenme süresini kısaltmak ve daha anlamlı sonuçlar elde etmek için basit bir normalleştirme işlemi yapalım. Bunun için boy değerlerden 160, kilo değerlerinden 71 çıkaralım.

İsim	Boy(-160)	Kilo(-71)	Cinsiyet
Ahmet	5	4	E (0)
Veli	15	5	E (0)
Nazan	-6	-26	K (1)
Elif	-24	-15	K (1)
Kamil	-15	-6	E (0)
Suzan	7	-15	K (1)
...

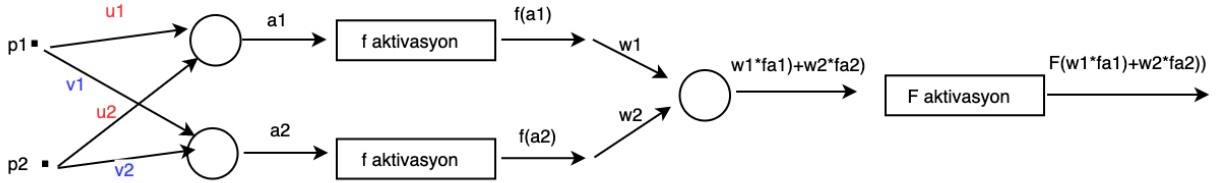
Bu veri setinde Ahmet isimli kişinin normalleştirilmiş boy ve kilo değerleri olan $[5,4]$ vektörünü 3 nöronlu yapay sinir ağı üzerinden geçirelim.



Ahmet'in cinsiyeti için elde edilen 0.7296 değeri 1'e daha yakın bir değerdir. Sadece tek elemanlı bir veri kümesi gibi düşünerek **MSE** (Mean Squared Error) yani Ortalama Standart Hata değeri hesaplanırsa,

$$MSE = L = \frac{1}{1} \sum_{i=1}^1 (Y_{gerçek} - Y_{tahmin})^2 = \frac{1}{1} \sum_{i=1}^1 (0 - 0.7296)^2 = 0.5323$$

O halde Ahmet gözlemi için MSE Loss değeri 0.5223 olur.



$$MSE = L = \frac{1}{1} \sum_{i=1}^1 (Y_{gerçek} - Y_{tahmin})^2$$

$$Y_{tahmin} = F(w1 + f(a1) + w2 * f(a2)) = F(w1 + f(u1 * p1 + u2 * p2) + w2 * f(v1 * p1 + v2 * p2))$$

$$\frac{\partial L}{\partial u_1} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial u_1}$$

$$\frac{\partial Y_{tahmin}}{\partial u_1} = \frac{\partial Y_{tahmin}}{\partial f} \frac{\partial f}{\partial u_1} = \frac{\partial Y_{tahmin}}{\partial f} p1$$

$$\frac{\partial L}{\partial u_2} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial u_2}$$

$$\frac{\partial Y_{tahmin}}{\partial u_2} = \frac{\partial Y_{tahmin}}{\partial f} \frac{\partial f}{\partial u_2} = \frac{\partial Y_{tahmin}}{\partial f} p2$$

$$\frac{\partial L}{\partial v_1} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial v_1}$$

$$\frac{\partial Y_{tahmin}}{\partial v_1} = \frac{\partial Y_{tahmin}}{\partial f} \frac{\partial f}{\partial v_1} = \frac{\partial Y_{tahmin}}{\partial f} p1$$

$$\frac{\partial L}{\partial v_2} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial v_2}$$

$$\frac{\partial Y_{tahmin}}{\partial v_2} = \frac{\partial Y_{tahmin}}{\partial f} \frac{\partial f}{\partial v_2} = \frac{\partial Y_{tahmin}}{\partial f} p2$$

$$\frac{\partial L}{\partial w_1} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial w_1}$$

$$\frac{\partial Y_{tahmin}}{\partial w_1} = \frac{\partial Y_{tahmin}}{\partial w_1}$$

$$\frac{\partial L}{\partial w_2} = 2(Y_{gerçek} - Y_{tahmin}) \frac{\partial Y_{tahmin}}{\partial w_2}$$

$$\frac{\partial Y_{tahmin}}{\partial w_2} = \frac{\partial Y_{tahmin}}{\partial w_2}$$

Burada f ve F fonksiyonu seçilen aktivasyon fonksiyonunu temsil etmektedir. Örneğin sigmoid fonksiyonu olursa türevi de aşağıdaki gibidir.

$$f(x) = \frac{1}{1+e^{-x}}, f'(x) = f(x)(1 - f(x))$$

$$F(x) = \frac{1}{1+e^{-x}}, F'(x) = F(x)(1 - F(x))$$

Herbiri için

$$u_{iYENİ} = u_i - \eta * \frac{\partial L}{\partial u_i}$$

$$v_{iYENİ} = v_i - \eta * \frac{\partial L}{\partial v_i}$$

$$w_{iYENİ} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

Şeklinde yukarıdaki türevler kullanarak güncelleme gerçekleşecektir.

Ağı eğitmek için, Loss miktarına göre Nöronlarda bulunan **W** ve **b** değerlerinin güncellenmesi ve ardında diğer gözlem girdilerinin ağ içeresinden geçirilip yeni tahmin değerleri ile tekrar güncelleme yapılması gerekmektedir. Bu işlem tahmin değeri ile gerçek değer arasındaki oran minimuma gelene kadar devam etmelidir.

Ağırlıkların ayrı ayrı nörondan elde edilen değerlere göre güncellenebilmesi için Loss değerinin her bir W ve b değerine olan etkisinin hesaplanması gerekmektedir. Bu işlemi ise Loss değerinin **W** ve **b** değerlerine göre kısmi türevinin alınmasıyla gerçekleştirebilir.

Yukarıdaki örneğimizde 3 adet nöron ve her bir nöronda 2 adet **W**, 1 adet **b** bulunmaktaydı. Bu durumda Loss fonksiyonu her bir ağırlık ve bias değeri için aşağıdaki gibi olmaktadır.

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Kısmi türev çok değişkenli bir fonksiyonda bir değişkenin o fonksiyonu ne kadar etkilediğini elde etmek için kullanılmaktadır. Bu durumda kısmi türev, her bir ağırlık ve bias değerinin Loss değerini nasıl etkilediğini ifade etmektedir. Yukarıda verilen L Loss fonksiyonun ağırlık ve bias değerlerine göre kısmi türevleri aşağıdaki gibidir.

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial w_5}, \frac{\partial L}{\partial w_6}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

W' in Loss'a olan etkisi $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial Y_{tahmin}} \frac{\partial Y_{tahmin}}{\partial w_1}$ olur. Bu türevi hesaplamak için önce

$$MSE = L = \sum_{i=1}^n (Y_{gerçek} - Y_{tahmin})^2, \text{den}$$

$\frac{\partial L}{\partial Y_{tahmin}} = -2 \sum_{i=1}^n (Y_{gerçek} - Y_{tahmin})$ hesaplanır. Daha sonra

$$\frac{\partial Y_{tahmin}}{\partial w_1} = \frac{\partial Y_{tahmin}}{\partial H_1} \frac{\partial H_1}{\partial w_1} \text{ türevi;}$$

$Y_{tahmin} = f((w_5H_1) + (w_6H_2) + b_3)$ ve $H_1 = f((w_1x_1) + (w_2x_2) + b_2)$ 'den sırasıyla

$$\frac{\partial Y_{tahmin}}{\partial H_1} = w_5 * f'((w_5H_1) + (w_6H_2) + b_3)$$

$$\frac{\partial H_1}{\partial w_1} = x_1 * f'((w_1x_1) + (w_2x_2) + b_2) \text{ türevleri alınarak hesaplanır.}$$

Burada fonksiyonu sigmoid fonksiyonunu temsil etmektedir ve türevi de aşağıdaki gibidir.

$$f(x) = \frac{1}{1+e^{-x}}, f'(x) = f(x)(1 - f(x))$$

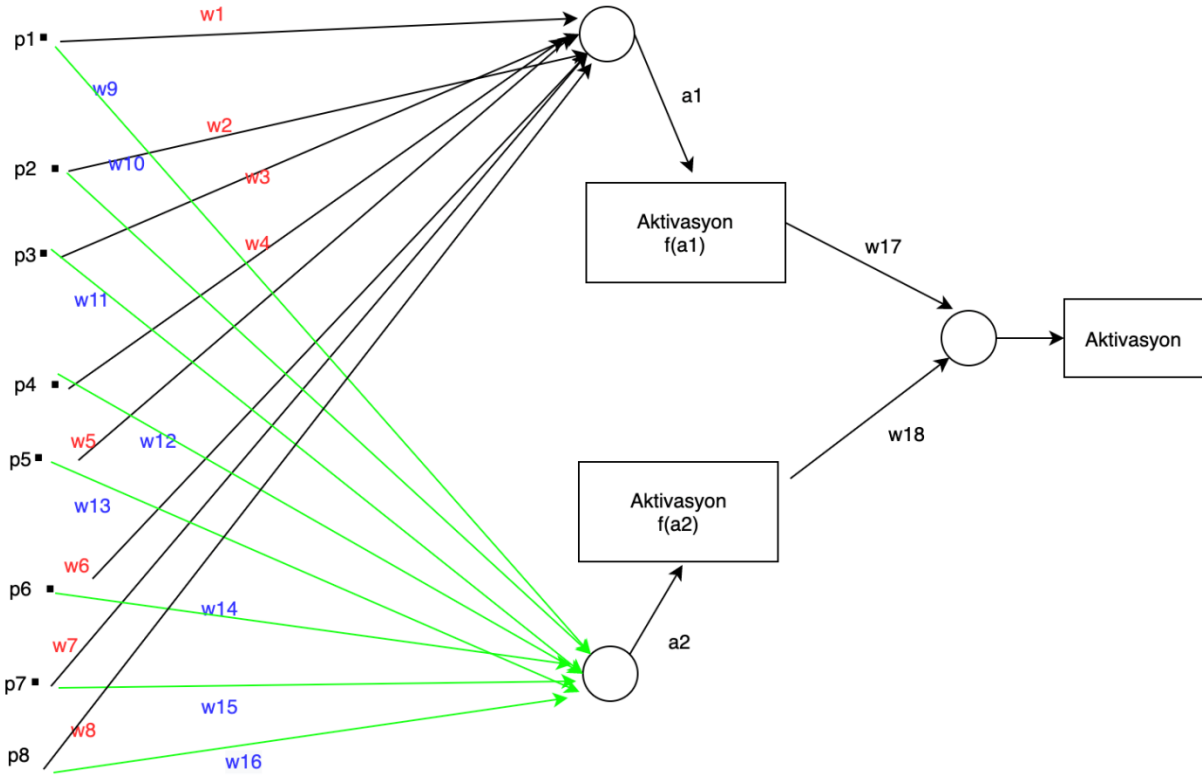
W_1 'in Loss değerine olan etkisi, $\frac{\partial L}{\partial w_1}$ hesaplanmasının ardından ağırlığın bu oranda güncellenmesi gerekmektedir. Bu güncelleme işlemi de “Learning Rate” yani öğrenme oranı düzeyinde gerçekleşmektedir. Etki hesaplamasında elde edilen sonuç öğrenme oranı ile çarpılarak güncellenecek ve ağırlık değerinden çıkarılarak yeni ağırlık değeri bulunacaktır.

$\eta = \text{Learning rate}$ olmak üzere,

$$w_{1YENİ} = w_1 - \eta * \frac{\partial L}{\partial w_1} \text{ 'dir.}$$

Yukarıdaki işlemler 3 nöronlu ve 3 ağırlık fonksiyonlu bir ağda ağırlık fonksiyonlarından birinin sadece ilk bileşeni için gereken aşamalardı. Bu aşamalar ağda olan bütün ağırlık ve bias değerleri için uygulanmalıdır.

Uygulama:



Ağımız üzerinde 3 adet nöron bulunduğundan dolayı, ilk 2 nöronda 8 adet ağırlık ve son nöronda 2 adet ağırlık olmak üzere toplam 18 adet ağırlık değer ve her bir nörona 1 adet bias olmak üzere toplam 3 adet bias değerini `__init__` methodu içerisinde rastgele olarak oluşturalım;

#Ağırlıklarımızı ve bias değerlerimiz burada oluşturulmaktadır.

```
def __init__(self):  
    # Ağ üzerinden 3 adet nöron olduğu için  
    # 8 adet ağırlık ve 3 adet bias değeri olmalı  
    self.w1 = np.random.normal()  
    self.w2 = np.random.normal()  
    self.w3 = np.random.normal()  
    self.w4 = np.random.normal()  
    self.w5 = np.random.normal()  
    self.w6 = np.random.normal()  
    self.w7 = np.random.normal()  
    self.w8 = np.random.normal()  
    self.w9 = np.random.normal()  
    self.w10 = np.random.normal()  
    self.w11 = np.random.normal()  
    self.w12 = np.random.normal()  
    self.w13 = np.random.normal()  
    self.w14 = np.random.normal()  
    self.w15 = np.random.normal()  
    self.w16 = np.random.normal()  
    self.w17 = np.random.normal()  
    self.w18 = np.random.normal()  
  
    self.b1 = np.random.normal()  
    self.b2 = np.random.normal()  
    self.b3 = np.random.normal()
```

Şimdi nörondaki toplam değeri 0 ile 1 arasında standartlaştıran sigmoid hesaplamasını yapacağımız methodumuzu oluşturalım. Burada girdi olarak x, nörondaki toplam değeri temsil etmektedir.

```
()#Sigmoid fonksiyonu
```

```
def sigmoid(self , x):
    #Sigmoid aktivasyon fonksiyonu :  $f(x) = 1 / (1 + e^{(-x)})$ 
    return (1/(1+np.exp(-x)))
```

Ardından ağırlık ve bias güncelleme eşitliklerinde kullanmak üzere sigmoid fonksiyonun türev'i olan methodu oluşturalım

```
#Sigmoid fonksiyonunun türevi
def sigmoid_turev(self , x):
    # Sigmoid fonksiyonunun türevi:  $f'(x) = f(x) * (1 - f(x))$ 
    sig = self.sigmoid(x)
    return sig * (1 - sig)
```

Geliştirdiğimiz sinir ağıının genel olarak nasıl eğitildiğini gözlemlemek için tahmin edilen değerler ile gerçek değerler arasındaki farkı inceleyerek, her bir devirde veri setindeki toplam Loss'u hesaplamak için MSE metodunu oluşturabiliriz.

```
def mse_loss(self , y_real , y_prediction):
    # y_real ve y_prediction aynı boyutta numpy arrayleri olmalıdır.
    return ((y_real - y_prediction) ** 2).mean
```

Şimdi ise feedforward methodu yani **İleri Besleme** yöntemini oluşturmamız gerekli

```
#İleri beslemeli nöron değerleri hesaplamaları

def feedforward(self , data):
    # h1 nöronunun değeri
    h1 = self.sigmoid((self.w1 * data[0]) + (self.w2 * data[1]) + (self.w3 * data[2]) + (self.w4 * data[3]) + (self.w5 * data[4]) + (self.w6 * data[5]) + (self.w7 * data[6]) + (self.w8 * data[7]) + self.b1 )

    # h2 nöronunun değeri
    h2 = self.sigmoid((self.w9 * data[0]) + (self.w10 * data[1]) + (self.w11 * data[2]) + (self.w12 * data[3]) + (self.w13 * data[4]) + (self.w14 * data[5]) + self.w15 * data[6]) + (self.w16 * data[7]) + self.b2 )

    # Tahmin değeri 01 nöronun değeri
```

```

        o1 = self.sigmoid((self.w17 * h1 ) + (self.w18 * h2 ) +
self.b3 )
    return o1

```

Bütün yöntemlerin belirlenmesinden sonra “train” yöntemimiz kodlayamaya başlayabiliriz. “train” kısmında her bir ağırlığın Loss değerine göre güncelleneceği değer belirlenmelidir. Bu bağlamda train metodunun içerisinde aşağıdaki değerlerin hesaplanması gerekmektedir.

- Loss değerinin tahmin değerine göre türevi.
- Tahmin değerinin hangi nöronun ağırlık değerine göre türev alıyorsa o nöronun toplam değerine göre türevi.
- Hangi nöronun ağırlık değerine göre türev alınıyorsa, o nöronun değerinin ağırlık değerine göre türevi.
- Hangi nöronun ağırlık değerine göre türev alınıyorsa, o nöronun değerinin “bias” değerine göre türevi.

Ayrıca eğitimin ne kadar tekrar edeceğini belirlemek ve her bir devirde ağırlık ve biasların hangi oranda güncelleneceğini belirlemek için “epoch” yani devir sayısı ve “learning rate” öğrenme oranı belirlenmesi gerekmektedir. Bu örnekte epoch = 1000 ve learning_rate= 0.001 olarak belirleyebiliriz.

```

#Belirtilen iterasyon sayısı kadar model eğitimi
def train(self , data , labels):

    learning_rate = 0.001
    epochs = 10000

    for epoch in range(epochs):

        for x, y in zip(data , labels):

            # Neuron H1
            sumH1 = (self.w1 * x[0]) + (self.w2 * x[1])
            +(self.w3 * x[2]) + (self.w4 * x[3])
            +(self.w5 * x[4]) + (self.w6 * x[5])
            +(self.w7 * x[6]) + (self.w8 * x[7])+ self.b1
            H1 = self.sigmoid(sumH1)

            # Neuron H2
            sumH2 = (self.w9 * x[0])+(self.w10 * x[1])
            +(self.w11 * x[2])+(self.w12 * x[3])
            +(self.w13 * x[4])+(self.w14 * x[5])
            +(self.w15 * x[6])+(self.w16 * x[7])+ self.b2

```

```

H2      = self.sigmoid(sumH2)

# Neuron O1
sumO1   = (self.w17 * H1) + (self.w18 * H2) +
self.b3

O1      = self.sigmoid(sumO1)

# Tahmin değerimiz
prediction = O1

# Türevlerin Hesaplanması
# dL/dYpred : y = doğru değer | prediciton:
tahmin değeri
dLoss_dPrediction = -2*(y - prediction)

# Aynı zamanda tahmin değerinin H1 ve H2'ye göre
türevlerinin de
# hesaplanması gerekmektedir.
dPrediction_dH1      = self.w17 *
self.sigmoid_turev(sumO1)
dPrediction_dH2      = self.w18 *
self.sigmoid_turev(sumO1)

# Nöron H1 için ağırlık ve bias türevleri
dH1_dW1 = x[0] * self.sigmoid_turev(sumH1)
dH1_dW2 = x[1] * self.sigmoid_turev(sumH1)
dH1_dW3 = x[2] * self.sigmoid_turev(sumH1)
dH1_dW4 = x[3] * self.sigmoid_turev(sumH1)
dH1_dW5 = x[4] * self.sigmoid_turev(sumH1)
dH1_dW6 = x[5] * self.sigmoid_turev(sumH1)
dH1_dW7 = x[6] * self.sigmoid_turev(sumH1)
dH1_dW8 = x[7] * self.sigmoid_turev(sumH1)
dH1_dB1 = self.sigmoid_turev(sumH1)

# Nöron H2 için ağırlık ve bias türevleri
dH2_dW9 = x[0] * self.sigmoid_turev(sumH2)
dH2_dW10 = x[1] * self.sigmoid_turev(sumH2)
dH2_dW11 = x[2] * self.sigmoid_turev(sumH2)
dH2_dW12 = x[3] * self.sigmoid_turev(sumH2)
dH2_dW13 = x[4] * self.sigmoid_turev(sumH2)
dH2_dW14 = x[5] * self.sigmoid_turev(sumH2)
dH2_dW15 = x[6] * self.sigmoid_turev(sumH2)
dH2_dW16 = x[7] * self.sigmoid_turev(sumH2)
dH2_dB2 = self.sigmoid_turev(sumH2)

```

```

# Nöron O1 (output) için ağırlık ve bias
türevleri
    dPrediction_dW17 = H1 *
self.sigmoid_turev(sumO1)
    dPrediction_dW18 = H2 *
self.sigmoid_turev(sumO1)
    dPrediction_dB3 = self.sigmoid_turev(sumO1)

## Ağırlık ve biasların güncellenmesi

# H1 nöronu için güncelleme
self.w1 = self.w1 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW1)
self.w2 = self.w2 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW2)
self.w3 = self.w3 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW3)
self.w4 = self.w4 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW4)
self.w5 = self.w5 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW5)
self.w6 = self.w6 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW6)
self.w7 = self.w7 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW7)
self.w8 = self.w8 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dW8)
self.b1 = self.b1 - (learning_rate *
dLoss_dPrediction * dPrediction_dH1 * dH1_dB1)

# H2 nöronu için güncelleme
self.w9 = self.w9 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW9)
self.w10 = self.w10 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW10)
self.w11 = self.w11 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW11)
self.w12 = self.w12 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW12)
self.w13 = self.w13 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW13)
self.w14 = self.w14 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW14)
self.w15 = self.w15 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW15)

```

```

        self.w16 = self.w16 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dW16)
        self.b2 = self.b2 - (learning_rate *
dLoss_dPrediction * dPrediction_dH2 * dH2_dB2)

        # 01 nöronu için güncelleme
        self.w17 = self.w17 - (learning_rate *
dLoss_dPrediction * dPrediction_dW17)
        self.w18 = self.w18 - (learning_rate *
dLoss_dPrediction * dPrediction_dW18)
        self.b3 = self.b3 - (learning_rate *
dLoss_dPrediction * dPrediction_dB3)

    predictions = np.apply_along_axis(self.feedforward ,1,
data)
    loss = self.mse_loss(labels , predictions)
    print("Devir %d loss: %.7f" % (epoch, loss))    loss:
%.7f" % (epoch, loss))

```

Artık Network class yapısını tamamladıktan sonra, ağı eğitmek için gerçek verilerle besleyerek daha sonra verilecek bir yeni veri için sonuç tahminleyebiliriz. Bu ağı verilen hasta verilerini kullanarak hastanın diyabet olup olmadığını tahmin edeceğiz (19). Veri kümesini aşağıdaki gibi import edelim.

```
a=pd.read_csv('/Users/tugbaaydemir/python_kodlar/KeskinVahit/dia
betes.csv')
```

a

Unnamed: 0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	0	6	148	72	35	0	33.6	0.627	50	1
1	1	1	85	66	29	0	26.6	0.351	31	0
2	2	8	183	64	0	0	23.3	0.672	32	1
3	3	1	89	66	23	94	28.1	0.167	21	0
4	4	0	137	40	35	168	43.1	2.288	33	1
...
763	763	10	101	76	48	180	32.9	0.171	63	0
764	764	2	122	70	27	0	36.8	0.340	27	0
765	765	5	121	72	23	112	26.2	0.245	30	0
766	766	1	126	60	0	0	30.1	0.349	47	1
767	767	1	93	70	31	0	30.4	0.315	23	0

768 rows x 10 columns

Burada ağda kullanacağımız 8 özneteliğin 366 değerini yani 366x8'lik bir matrisi ayıralım. Çünkü veriler temizlememiz gerekiydi. Gerçekte bir insanın sahip olamayacağı gerçek dışı

verileri kümemizden attık. Örneğin yaşayan hiçbir insanın tansiyonu, deri kalınlığı, insülin miktarı vb. öznitelikleri 0 değerini alamaz.

```
a=a[(a['Glucose']>20)&(a['BloodPressure']>20)&(a['SkinThickness']>10)&(a['Insulin']>40)&(a['BMI']>10)&(a['DiabetesPedigreeFunction']>0.1)]
```

a

Unnamed: 0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
3	3	1	89	66	23	94	28.1	0.167	21	0
4	4	0	137	40	35	168	43.1	2.288	33	1
6	6	3	78	50	32	88	31.0	0.248	26	1
8	8	2	197	70	45	543	30.5	0.158	53	1
13	13	1	189	60	23	846	30.1	0.398	59	1
...
751	751	1	121	78	39	74	39.0	0.261	28	0
753	753	0	181	88	44	510	43.3	0.222	26	1
755	755	1	128	88	39	110	36.5	1.057	37	1
763	763	10	101	76	48	180	32.9	0.171	63	0
765	765	5	121	72	23	112	26.2	0.245	30	0

366 rows x 10 columns

```
#indekleri duzenledik
```

```
a.reset_index(inplace=True)
```

Bu verideki öznitelik değerlerinin farklılıkları nedeniyle bu değerleri önce normalize etmeliyiz.

Bu aynı zamanda bize işleme de kolaylık da sağlayacaktır.

```
data=a.iloc[:,2:10]
```

```
data
```

```
#normalizasyon
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
sc=MinMaxScaler()
```

```
x_scaled = sc.fit_transform(data)
```

```
x_scaled
```

```
array([[0.05882353, 0.23239437, 0.45      , ..., 0.20245399, 0.0270027 ,
        0.          ],
       [0.          , 0.57042254, 0.125    , ..., 0.50920245, 0.98154815,
        0.2         ],
       [0.17647059, 0.15492958, 0.25     , ..., 0.26175869, 0.06345635,
        0.08333333],
       ...,
       [0.05882353, 0.50704225, 0.725    , ..., 0.37423313, 0.42754275,
        0.26666667],
       [0.58823529, 0.31690141, 0.575    , ..., 0.3006135 , 0.02880288,
        0.7         ],
       [0.29411765, 0.45774648, 0.525    , ..., 0.16359918, 0.06210621,
        0.15        ]])
```

Daha sonra veriden sonuç kısmını veren etiket değişkenini ayıralım.

```
labels=a['Outcome']
labels
0      0
1      1
2      1
3      1
4      1
..
361    0
362    1
363    1
364    0
365    0
Name: Outcome, Length: 366, dtype: int64
```

Artık ağımızı veri ile eğitebiliriz. Eğitim verisi olarak 240 veri kullanalım. Kalan veriyi de test için kullanacağız.

```
network = Network()
network.train(x_scaled[:240],labels[:240])
```

Devir 9999 loss: 0.2971985

Test verimizi kullanarak karmaşıklık matrisimizi görelim.

```
X_test=x_scaled[240:]
y_test=a['Outcome'][240:]
```

```
y_pred=[]
for i in range(len(X_test)):
    if(network.feedforward(X_test[i])> 0.5):
        y_pred.append(1)
```



```

else:
    y_pred.append(0)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
cm.T
array([[82, 31],
       [ 5,  8]])

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

```

	precision	recall	f1-score	support
0	0.73	0.94	0.82	87
1	0.62	0.21	0.31	39
accuracy			0.71	126
macro avg	0.67	0.57	0.56	126
weighted avg	0.69	0.71	0.66	126

Artık eğitilmiş ağıma yeni bir hastaya ait bilgileri vererek tahminleme yapabiliriz. Bu yeni veriyi ağa vermeden önce normalize etmeyi unutmamamız gerekir. Çünkü ağa verileri normalize ederek vermiştik.

```
data.max()
```

Pregnancies	17.000
Glucose	198.000
BloodPressure	110.000
SkinThickness	60.000
Insulin	846.000
BMI	67.100
DiabetesPedigreeFunction	2.329
Age	81.000
dtype: float64	

```
data.min()
```

Pregnancies	0.000
Glucose	56.000
BloodPressure	30.000
SkinThickness	11.000
Insulin	41.000
BMI	18.200
DiabetesPedigreeFunction	0.107
Age	21.000
dtype:	float64

```
#normalizasyon
X = [6,148,72,35,0,33.6,0.627,50]
X=[(X[0]-data.min()[0])/(data.max()[0]-data.min()[0]),
    (X[1]-data.min()[1])/(data.max()[1]-data.min()[1]),
    (X[2]-data.min()[2])/(data.max()[2]-data.min()[2]),
    (X[3]-data.min()[3])/(data.max()[3]-data.min()[3]),
    (X[4]-data.min()[4])/(data.max()[4]-data.min()[4]),
    (X[5]-data.min()[5])/(data.max()[5]-data.min()[5]),
    (X[6]-data.min()[6])/(data.max()[6]-data.min()[6]),
    (X[7]-data.min()[7])/(data.max()[7]-data.min()[7]),
    ]
X

prediction = network.feedforward(X)

if(prediction > 0.5):
    print('X Diyabet var   |' , 'Value : ' , prediction)
else:
    print('X Diyabet yok   |' , 'Value : ' , prediction)
```

X Diyabet yok | Value : 0.013292619584133754

Verilen hasta verisi sonucunda ağ bu hastanın diyabet hastası olmadığı sonucunu vermiştir.

Bibliography

1. **Güzel , Hasan.** Makine ÖğrenmesiPythonR EğitimleriYapay Zeka R ve Python ile Sıfırdan Sinir Ağlarını (NNs) Anlama ve Kodlama. [Online] Kasım 24, 2020. [Cited: Şubat 18, 2021.] <https://www.datascienceearth.com/r-ve-python-ile-sifirdan-sinir-aglarini-nns-anlama-ve-kodlama/>.
2. *Mevduat Bankalarının Karlılığının Yapay Sinir Ağları ile Tahmini: Bir Yazılım Modeli Tasarımı.* **SÖNMEZ, Ferdi.** 1, 2015, Vol. 9.
3. **Tutorialspoint.** *tutorialspoint.com.* [Online] https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm.
4. *Yapay Sinir Ağlarının Kullanım Alanları ve Bir Uygulama.* **Ağyar, Zafer.** 662, Mühendis ve Makine, Vol. 56, pp. 22-23.
5. **Makinist, Semiha.** Derin Öğrenme (Yapay Sinir Ağları-1). [Online] Nisan 11, 2018. [Cited: Mart 11, 2021.] <http://buyukveri.firat.edu.tr/2018/04/11/derin-ogrenme-part-1/>.
6. *Yapay Sinir Ağları ve Yapay Zekâ'ya Genel Bir Bakış.* **Öztürk, Kadir.** 2, 2018 : s.n., Vol. 6.
7. **Şengöz, Nilgün.** Yapay Sinir Ağları. [Online] Mart 4, 2017. [Cited: Mart 18, 2021.] <https://www.derinogrenme.com/2017/03/04/yapay-sinir-aglari/>.
8. *Yapay Sinir Ağları ile Öngörü Modellemesi.* **Ataseven, Burçin.** 39, 2013, Öneri Dergisi, Vol. 10.
9. **ÇINAR, Utku Kubilay.** *veribilimiokulu.* [Online] Eylül 30, 2018. <https://www.veribilimiokulu.com/yapay-sinir-aglari/>.
10. **Aymaz, Osman.** <https://www.elektrikport.com>. [Online] Şubat 15, 2015. <https://www.elektrikport.com/teknik-kutuphane/yapay-sinir-aglari-avantaj-ve-dezavantajlari/15007#ad-image-0>.
11. **Filiz, Fahrettin.** *medium.com.* [Online] Ağustos 19, 2017. <https://medium.com/@fahrettinf/4-1-1-artificial-neural-networks-6257a7a54bb3>.
12. *Uyarlanır Yerel Bağlı Nöron Modelinin İncelemesi.* **TEK, Faik Boray.** 4, 2019, Vol. 12.
13. **Altunbas, Cansu.** <https://medium.com>. [Online] Şubat 13, 2020. <https://medium.com/türkiye/yapay-sinir-ağları-ysa-caec1cc0bf9f>.

14. **Cansız, Sergen.** Sıfırdan Python İle Neural Network (Yapay Sinir Ağ) Oluşturmak : Bölüm 1 — Yapay Sinir Ağları ve Nöronlar Nasıl Çalışır? [Online] Mart 24, 2020. [Cited: Şubat 14, 2021.] <https://medium.com/datamming/s%C4%B1f%C4%B1rdan-python-i%C4%B1le-neural-network-yapay-sinir-a%C4%9F-olu%C5%9Fturmak-b%C3%B6l%C3%BCm-1-yapay-sinir-a%C4%9Flar%C4%B1-6209b27a17c1>.
15. **wikipedia.org.** [Online] https://en.wikipedia.org/wiki/Feedforward_neural_network.
16. **Makinist, Semiha.** Derin Öğrenme (Yapay Sinir Ağları-3). [Online] Nisan 16, 2018. [Cited: Mart 5, 2021.] <http://buyukveri.firat.edu.tr/2018/04/16/derin-ogrenme-yapay-sinir-aglari-3/>.
17. **Çıtır, Ennur.** <https://ennurcitir.wordpress.com>. [Online] Şubat 19, 2020. <https://ennurcitir.wordpress.com/2020/02/19/yapay-sinir-aglari/>.
18. **DevHunter.** Python ile Derin Öğrenme. [Online] Mayıs 18, 2018. [Cited: 12 Mart, 2021.] <https://devhunteryz.wordpress.com/2018/05/18/python-ile-derin-ogrenme/>.
19. **Saurabh , Singh.** Kaggle. [Online] Kasım 13, 2017. [Cited: Mart 5, 2021.] <https://www.kaggle.com/saurabh00007/diabetescsv/metadata>.