

Derin Öğrenmeye Giriş

Kursa Başlamadan Önce

- Bu kurs tamamlandığı takdirde giriş düzeyi yapay zeka algoritmaları ve veri analizine temel oluşturabilecek genel bilgileri edinmiş olacaksınız
- Spesifik konular anlaşılması zor ve kişide ders esnasında mantığın oturması kolay olmayacağından bol bol bireysel pratik gerekmektedir
- Slaytlar yazılara boğulmadan görsellerle anlatılacaktır. Bu yüzden ders esnasında not tutulması **son derece** önemlidir
- Konu başlıkları temel düzey algoritmalar için yeterli olduğundan başlıklar araştırılmalı, bol bol uygulama ve teorik bilgiler içeren sitelerde araştırma yapılmalıdır

Bu Eğitimde Neler Öğreneceksiniz

1. Yapay sinir ağlarına derinlemesine giriş yapacak ve yapay sinir ağlarının temel taşı olan nöronları yakından tanıyacağız.
2. Bir Derin Öğrenme modelinin eğitimi boyunca gerçekleşen adımları sırasıyla göreceğiz, bir eğitim boyunca olanları anlamak için Yapay Sinir Ağları Matematiğinin kavramlarını vereceğiz.
3. Yapay sinir ağının çalışma adımlarında göreceğimiz aktivasyon fonksiyonu, optimizier ve hiper-parametrelerin nasıl çalıştığını öğrenecek ve yaygın kullanılan çeşitlerini göreceğiz.
4. Tek katmanlı yapay sinir ağlarının temel birimi Perceptron ile tanışacak ve Perceptron kodlaması yapacağız.

Mert Çobanov

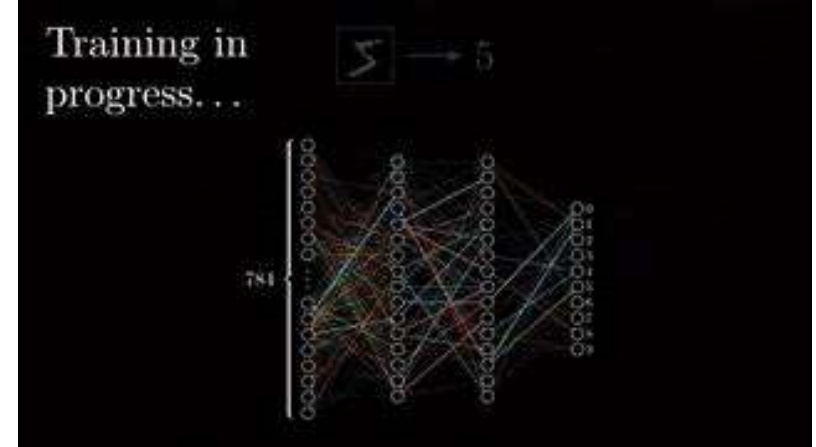
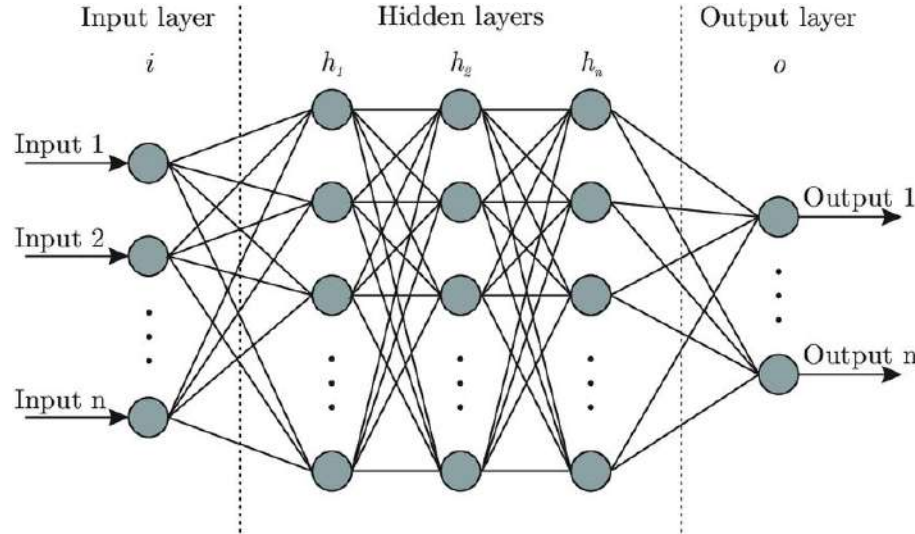
Github: cobanov

Linkedin: mertcobanoglu

Twitter: mertcobanov

Yapay Sinir Ağları

Sinir ağları, birbirine bağlı nöronlara sahip bilgi işlem sistemleridir. Belli algoritmaları kullanarak, ham verilerdeki gizli kalıpları ve korelasyonları tanıyabilir, kümeleyebilir ve sınıflandırabilir ve - zaman içinde - sürekli olarak öğrenip geliştirebilirler.



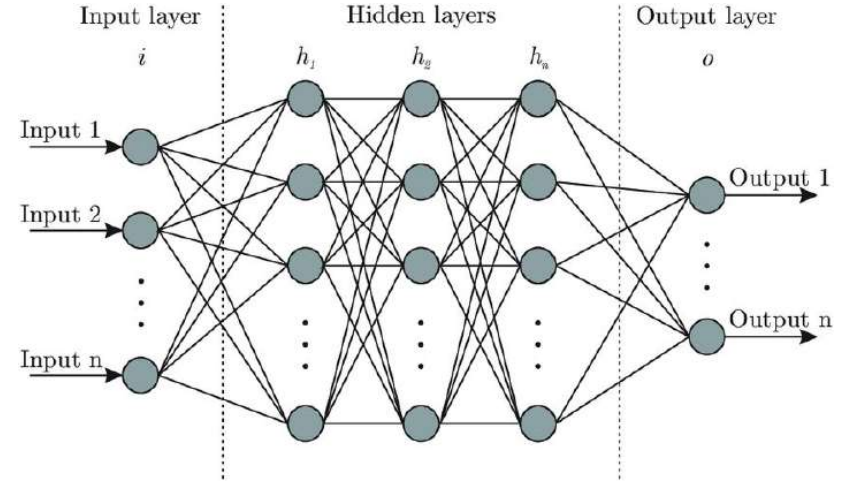
Yapay Sinir Ağları

Yapay Sinir Ağları 3 katmandan oluşur:

Input katmanı: Modelimize veriyi beslediğimiz katmandır. Her zaman vektörel veri kabul eder

Hidden katmanı: Input ve output katmanı arasındaki her katmandır. Bu katman non-lineeriteyi sağlamak için aktivasyon fonksiyonlarına sahip nöronlar içerir

Output layer: Modelimizin çıktısının verildiği katmandır. Problemin tipine göre seçilen aktivasyon fonksiyonuna uygun çıktı üretir

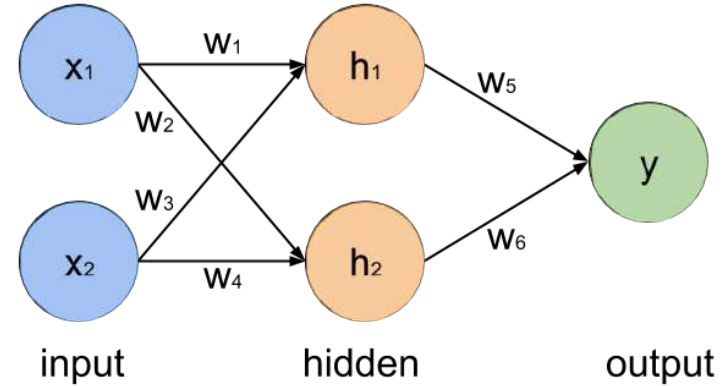


Nöron Kavramı

Nöronlar (bir diğer adıyla Perceptron), ağdaki veri ve hesaplamaların aktığı düğümlerdir. **Input**, **Hidden** ve **Output** katmanları nöronlardan oluşur.

Hidden katmandaki nöronlar **aktivasyon fonksiyonu** barındırır ve lineerite bu nöronlar içinde yapılan hesaplamalar sonrası bozulur.

Bu işlem, sınıflandırma veya tahmin problemlerinden daha kompleks problemlerin çözümünde etkin rol oynar.



Temel Regresyon Matematığı

Lineer Regresyon Matematiği

Lineer Regresyon bir bağımlı değişken (y) ile bir veya birden fazla bağımsız değişken(x) arasındaki ilişki kurar.

```
df = pd.DataFrame(data = {  
    'Metrekare':[100, 150, 120, 300, 230],  
    'Bina Yaşı':[5, 2, 6, 10, 3],  
    'Fiyat':[70, 90, 95, 120, 110]  
})
```

```
X = df.iloc[:, :-1]  
y = df.iloc[:, -1]
```

```
reg = LinearRegression().fit(X, y)
```

```
print(f"Score: {reg.score(X, y)}")
```

```
print(f"Coefficient: {reg.coef_}")
```

```
print(f"Intercept: {reg.intercept_}")
```

```
Score: 0.8465258951131542
```

```
Coefficient: [ 0.21520796 -0.15619124]
```

```
Intercept: 59.0747612003911
```

Dependent variable (DV)

Independent variables (IVs)

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$



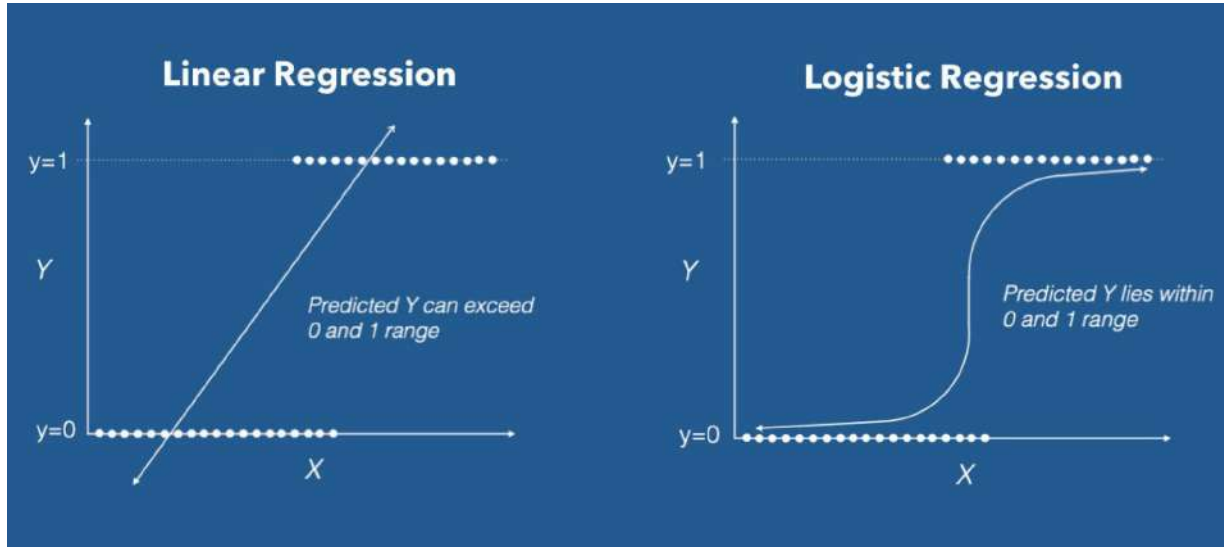
```
# y = b0 + b1x1 + b2x2
```

```
y = 59.0747612003911 + (0.21520796 * 100) + (-0.15619124 * 5)
```

```
print(y) # 79.8146010003911
```

Lojistik Regresyon Matematiği

Lojistik Regresyon, bağımlı değişkenin kategorik bir değer olduğu regresyon biçimidir. Lineer Regresyon fonksiyonu, Sigmoid aktivasyon fonksiyonuna sokularak çıktı 0 ve 1 arasına sıkıştırılır. Bu fonksiyon sayesinde ikili sınıflandırma problemleri çözülebilir

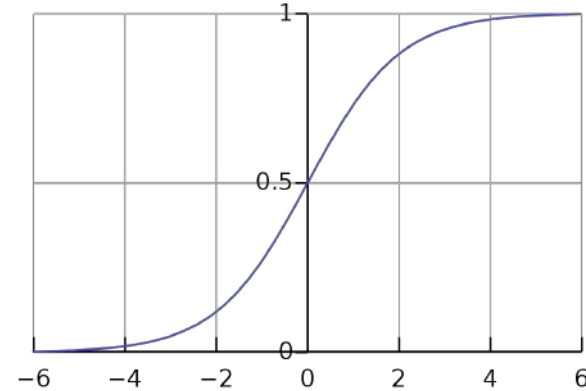


Sigmoid Aktivasyon Fonksiyonu

Sigmoid fonksiyonu, sürekli bir değeri 0 ile 1 arasına sıkıştırarak olasılık değeri çıkaran bir aktivasyon fonksiyonudur.

- S şeklinde bir eğriye sahiptir
- Değeri 0 ve 1 arasına sıkıştırır
- Lojistik regresyon probleminde Sigmoid Fonksiyonu kullanılır

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Yapay Sinir Ağlarının Komponentleri ve Matematiği

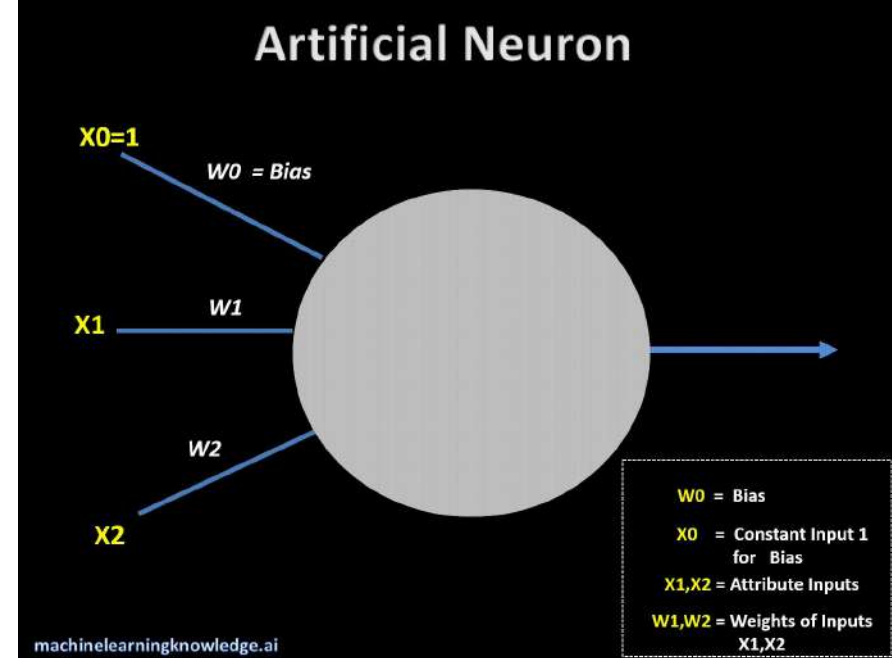
Weight

Temel makine öğrenmesinde ağırlık, bir bağımsız değişkenin bağımlı değişkeni ne kadar etkilediğini belirten bir özellik katsayısıdır.

Yapay Sinir Ağlarında ise ağırlık, her bir nöron değerinin katsayısıdır.

Bir nörona bağlanan nöronlar kendi ağırlıkları ile çarpılırlar.

$$y = w_0 * x_0 + w_1 * x_1 + w_2 * x_2$$



Bias

Bias, aktivasyon fonksiyonun verilere daha iyi uyması için sola veya sağa kaydırılmasına izin verir.

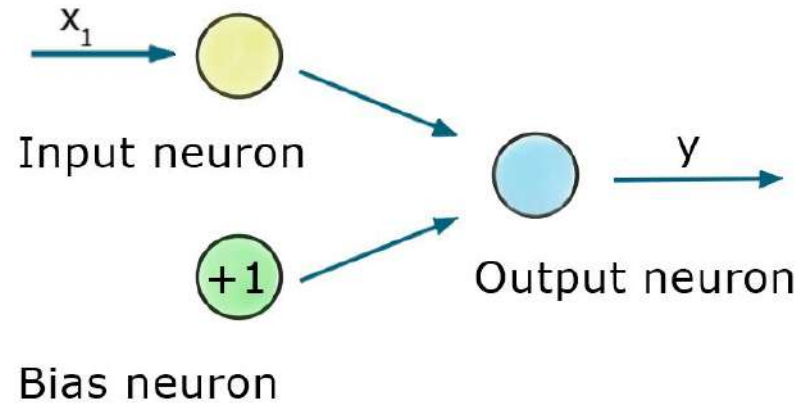
Bias aynı zamanda bir nöronun çıktısının sonraki katmana iletilip ileilmeyeceğini belirleyen bir eşiktir. Nöronumuzun çıktısı belli bir eşik değerini (bias) geçemiyorsa çıktı sonraki katmana iletilmez.

Ağda bir nörona bağlı tüm nöronlar kendi ağırlıkları ile çarpılır. yandaki örnekte şu şekilde bir denklem elde edilir:

$$y = X1W1 + X2W2 + X3W3 + b$$

Eğer bias olmasaydı şu şekilde bir denklem oluşurdu:

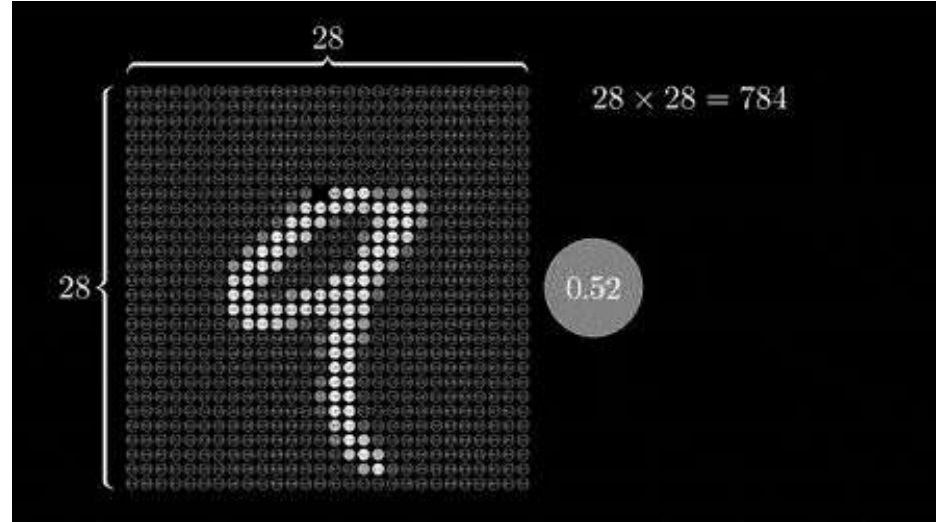
$$y = X1W1 + X2W2 + X3W3$$



Input

Bir yapay sinir ağında, input verisinin shape değeri ile yapay sinir ağının input katmanının shape değeri aynı olmalıdır. Bu katman verilerin ağa giriş yeridir.

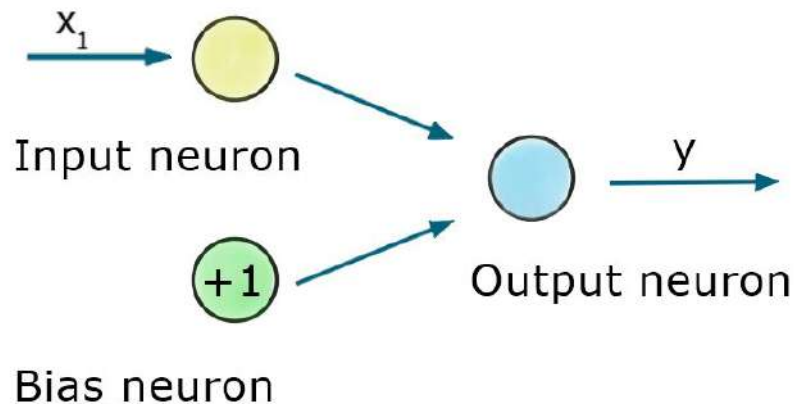
Yapay Sinir Ağı'nın input katmanı verileri vektör olarak kabul eder ve veri rank 1 tensörden daha fazla boyuta sahipse , veri reshape kullanarak rank 1 tensöre indirgenmelidir. Örneğin 28x28 boyutunda verinizi yapay sinir ağınıza vermek için 784 elemanlı bir vektöre çevirmelisiniz



Output

Çıktı katmanı, nihai sonucun üretilmesinden sorumludur. Bir sinir ağında her zaman bir çıktı katmanı olmalıdır.

Problemin çeşidine göre çıktı katmanında belirlenecek aktivasyon fonksiyonu sayesinde istenen çıktı elde edilir.



Veri Türleri ve Tensörler

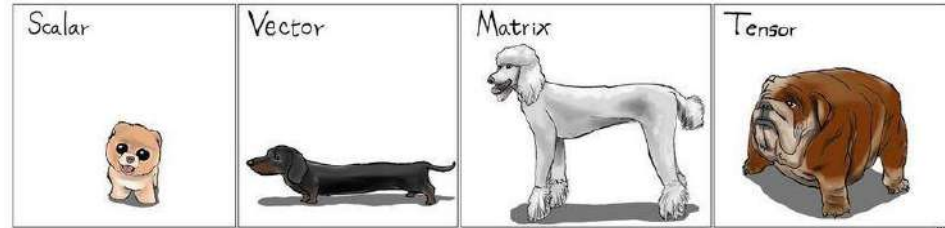
Yapay sinir ağlarına beslenen veriler, lineer cebirdeki veri türleri tarafından tanımlanabilir.

Skaler: Tekil değere sahip verilerdir. Skaler değer aynı zamanda Rank 0 Tensor'dur.

Vektör: Tek boyutlu koleksiyon verisidir. Vektör aynı zamanda Rank 1 Tensor'dur.

Matrix: İki boyutlu koleksiyon verisidir. Matrix aynı zamanda Rank 2 Tensor'dur.

Tensör: İkiden çok boyutlu koleksiyon verisidir. 2'den fazla çeşitli ranklere sahip tensörlerdir.



Matrislerde Cebir İşlemleri

Toplama İşlemi: İki matris toplandığında konumsal olarak karşılık gelen değerler toplanarak yeni matris elde edilir. Matrislerin toplanabilmesi için boyutlarının aynı olması gerekir ve toplam matrisi de toplanan matrislerle aynı boyuttadır.

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

Element-wise Product (Hadamard Product): Matrislerin Hadamard çarpımı eleman bazında bir işlemdir. Konumsal olarak karşılık gelen değerler, yeni bir matris üretmek için çarpılır.

$$\begin{bmatrix} 3 & 5 & 7 \\ 4 & 9 & 8 \end{bmatrix} \overset{G}{\circ} \begin{bmatrix} 1 & 6 & 3 \\ 0 & 2 & 9 \end{bmatrix} \overset{H}{=} \begin{bmatrix} 3 \times 1 & 5 \times 6 & 7 \times 3 \\ 4 \times 0 & 9 \times 2 & 8 \times 9 \end{bmatrix} \overset{N}{}{}$$

Matrislerde Cebir İşlemleri

Dot product: Vektörlere bu işlem uygulandığında sonuç skaler olur. Örneğin:

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 2 \\ 8 \end{bmatrix} = 2 \cdot 8 + 7 \cdot 2 + 1 \cdot 8 = 38$$

↑
Dot product

Matrislere bu işlemi uygulayabilmek için matris boyutlarının sırasıyla $m \times n$ ve $n \times m$ olması gerekmektedir. Çıktı tensörü $m \times m$ boyutta olur.

Matrislerde Cebir İşlemleri

Dot product: Matrisler vektörlere ayrılır ve vektörler çarpılarak skaler bir değer ele edilir. Vektörlere ayırma işleminde $m \times n$ boyutundaki ilk matrisin satırları, $n \times m$ boyutundaki ikinci matrisin sütunları sırayla vektör olarak alınır ve çarpım sonuçları ile boyutu $m \times m$ olan matris oluşturulur.

Input:	Result:
$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 7 \end{pmatrix}$	$\begin{pmatrix} 13 & 27 \\ 11 & 17 \end{pmatrix}$

Input:	$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 7 \end{pmatrix}$	$(1*1) + (2*3) + (3*2) = 13$
Input:	$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 7 \end{pmatrix}$	$(1*2) + (2*2) + (3*7) = 27$
Input:	$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 7 \end{pmatrix}$	$(3*1) + (2*3) + (1*2) = 11$
Input:	$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 7 \end{pmatrix}$	$(3*2) + (2*2) + (1*7) = 17$

Matrislerde Cebir İşlemleri

Transpose İşlemi: Bu işlem matrisin boyutlarını değiştirmenizi sağlar. $m \times n$ bir matrisi $n \times m$ boyuruna çevirir. Derin Öğrenme’de modelimizin inputu herhangi bir rank’te tensor olabilir ve bu tensörler boyut olarak cebir işlemlerine uygun olmayabilir. Transpose, bu uygunluğu sağlamayı amaçlar ama her zaman bu uygunluğun garantisini vermez, matrisinizi daha özelleştirilmiş bir boyuta çevirmeniz gerekebilir.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \Rightarrow \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

Matrislerde Cebir İşlemleri

Numpy'da reshape() işlemi ile tensörün boyutu değiştirilebilir. Bu işlem sırasında dikkat edilmesi gereken nokta, yeni boyutunuzun da eski boyuttaki kadar eleman tutabilmesidir.

Örneğin elinizdeki 6x4 boyutunda bir matris 24 eleman tutacaktır. Siz bu boyutu 12x2 boyutuna çevirebilirsiniz çünkü bu da 24 eleman tutacaktır. Ancak 24 elemanlı bir tensörün boyutunu 5x3 boyutuna çeviremezsiniz. Çünkü 5x3 boyutundaki bir tensör 15 eleman tutmaktadır.

data

1
2
3
4
5
6

data.reshape(2,3)

1	2	3
4	5	6

data.reshape(3,2)

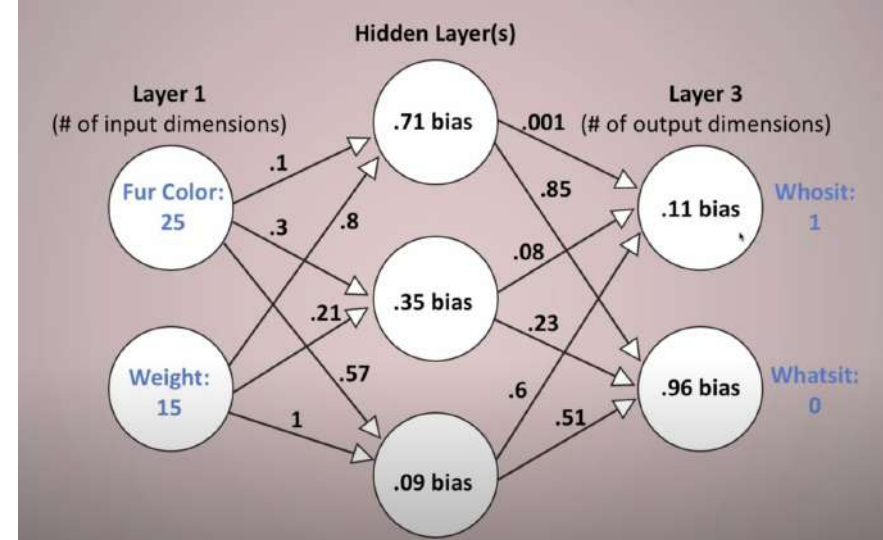
1	2
3	4
5	6

Yapay Sinir Ağı Nasıl Çalışır?

Yapay Sinir Ağları Nasıl Çalışır?

Forward Propagation:

1. Input verimiz yapay sinir ağlarına verilir. Yapay sinir ağıımızdaki ağırlık deęerleri ve nöronlarımızdaki bias deęerleri rastgele verilir.
2. Input nöronundaki veriler ağırlık deęerleriyle bir sonraki nörona gönderilir.
3. Her nöron, kendisine baęlı nöronların deęerlerini ve ağırlıklarını çarpıp toplar. Bu sonuç aktivasyon fonksiyonuna sokulur ve çıktı, her bir nöronun içinde bulunan bias deęeri ile kıyaslanır. Eęer çıktı, bias deęerini geçemiyorsa nöron sonraki katmana bilgi iletemez.



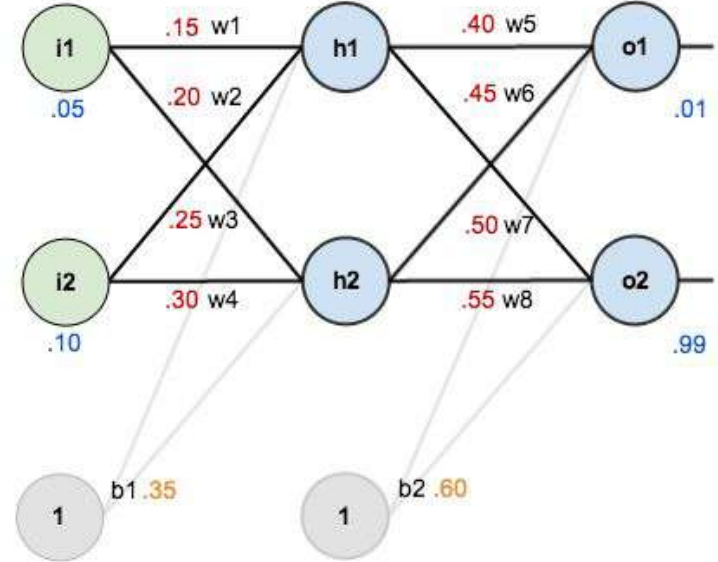
Yapay Sinir Ağları Nasıl Çalışır?

Back Propagation:

1. Çıktı katmandaki nöronlardan çıkan tahmin sonuçları, gerçek değerle kıyaslanıp hata hesaplanır. Bu hata değerleri ve önceki katmandaki (gizli katman) ağırlık değerleri kullanılarak önceki katmanın hatası hesaplanır.

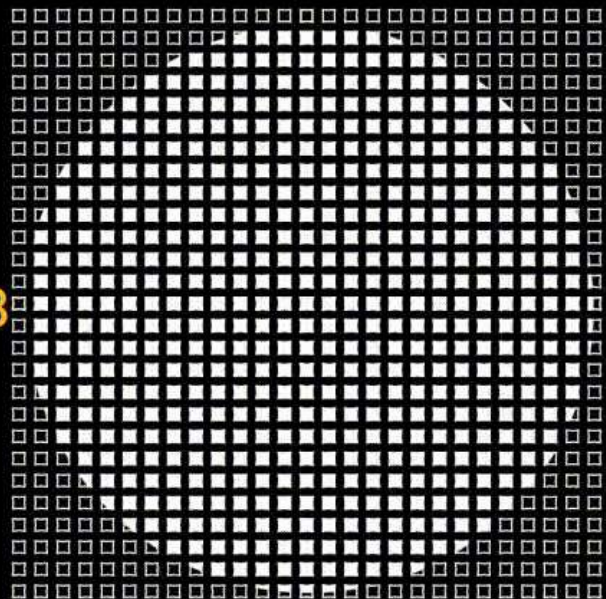
$$error(h1) = \frac{w5}{w5 + w6} * error(o1) + \frac{w7}{w7 + w8} * error(o2)$$

2. Hesaplanan hata değerleri kullanılarak gradyanlar hesaplanır. Bu gradyanlar kullanılarak ağırlık değerleri yeniden hesaplanır.



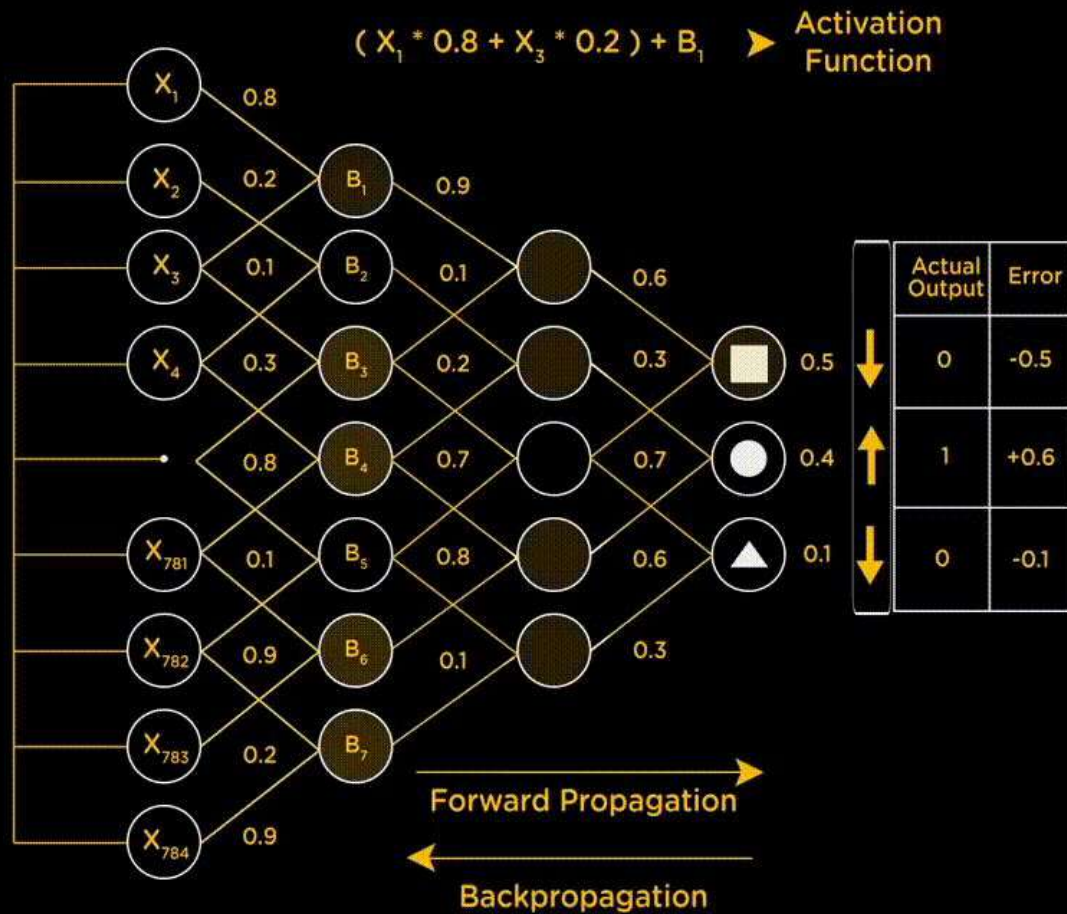
Hata formülü, şekildeki ağ baz alınarak verilmiştir.

28



28

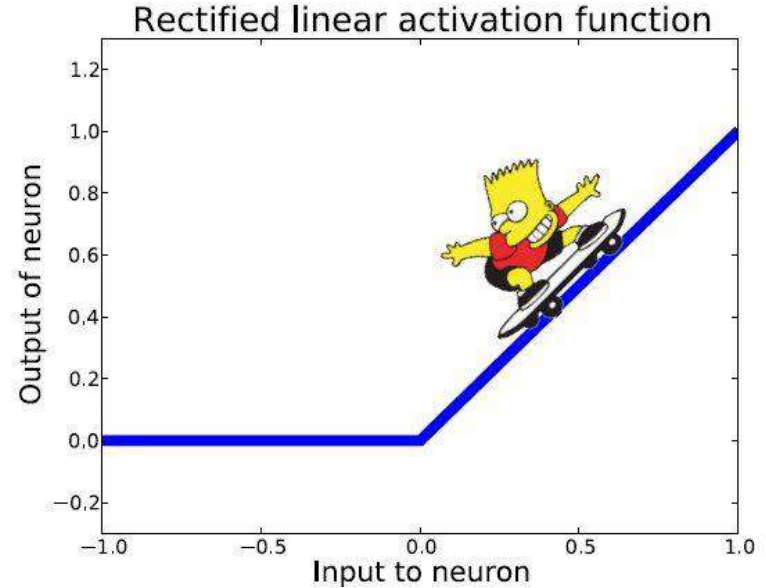
28 x 28 = 784 Pixels



Vanishing Gradient Problemi

Sinir ağılarına belirli aktivasyon fonksiyonlarını kullanan daha fazla katman eklendikçe, kayıp fonksiyonunun gradyanları sıfıra yaklaşır ve ağı eğitilmesi zorlaşır.

Sinir ağlarının gradyanları, back propagation kullanılarak bulunur. Son katmandan ilk katmana katman katman hareket ederek ağı türevlerini bulur. Zincir kuralına (Chain Rule) göre, ilk katmanların türevlerini hesaplamak için her katmanın türevleri ağıda çarpılır (son katmandan ilk katmana kadar).

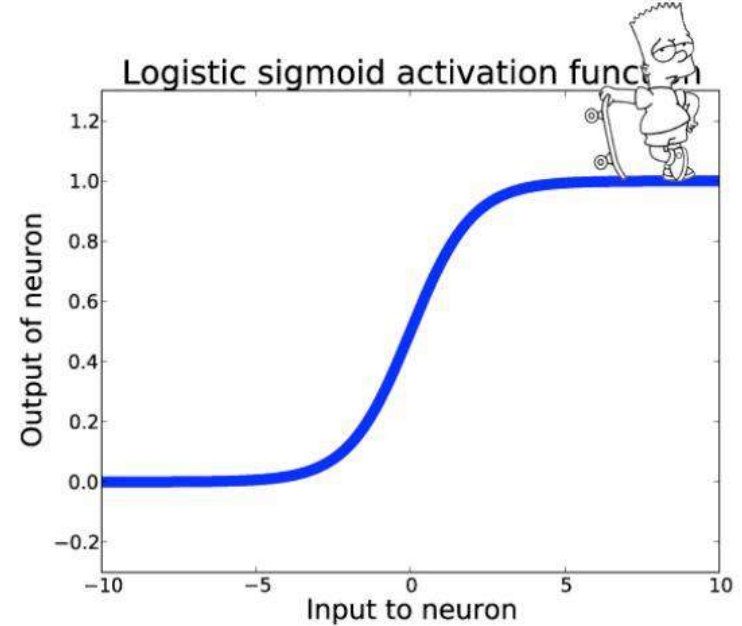


Vanishing Gradient Problemi

Sigmoid fonksiyonu çıktıyı 0 ile 1 arasında sıkıştırır. Bu nedenle, sigmoid işlevinin inputundaki büyük bir değişiklik, çıktıda küçük bir değişikliğe neden olur. Dolayısıyla türev küçük olur.

n tane gizli katman ile beraber sigmoid fonksiyonu gibi bir aktivasyon kullandığında, n tane küçük türev birlikte çarpılır. Böylece, back propagation sırasında başlangıç katmanlarına doğru ilerledikçe gradyan katlanarak azalır.

Vanishing Gradient problemine getirilebilecek çözümlerden biri, sigmoid fonksiyonundaki gibi küçük bir türev değeri üretmeyen ReLU aktivasyon fonksiyonunu kullanmaktır.



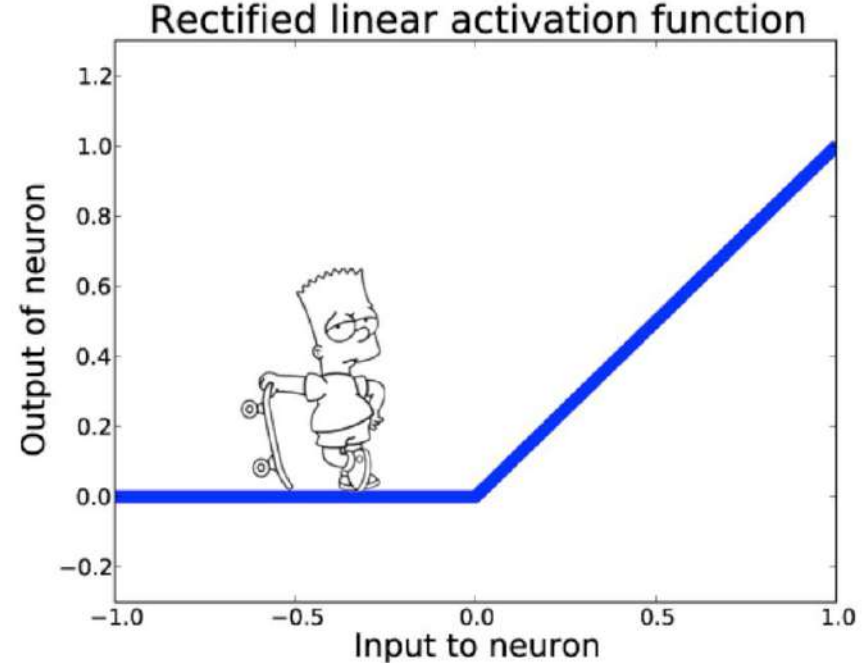
Dying ReLU Problemi

ReLU, Vanishing Gradient sorununa çözüm getirse de başka bir probleme neden olmaktadır; Dying ReLU.

ReLU negatif inputları sıfır değerine eşitlediği için negatif değerleri için gradyan 0 olur.

Ağda çok sayıda 0 çıktısı üreten nöron varsa, öğrenme olması gerektiği şekilde gerçekleşemeyecektir.

Bu karakteristik özellik ReLU'ya gücünü veren şey olsa da çok fazla negatif input olması durumunda probleme yol açacaktır.



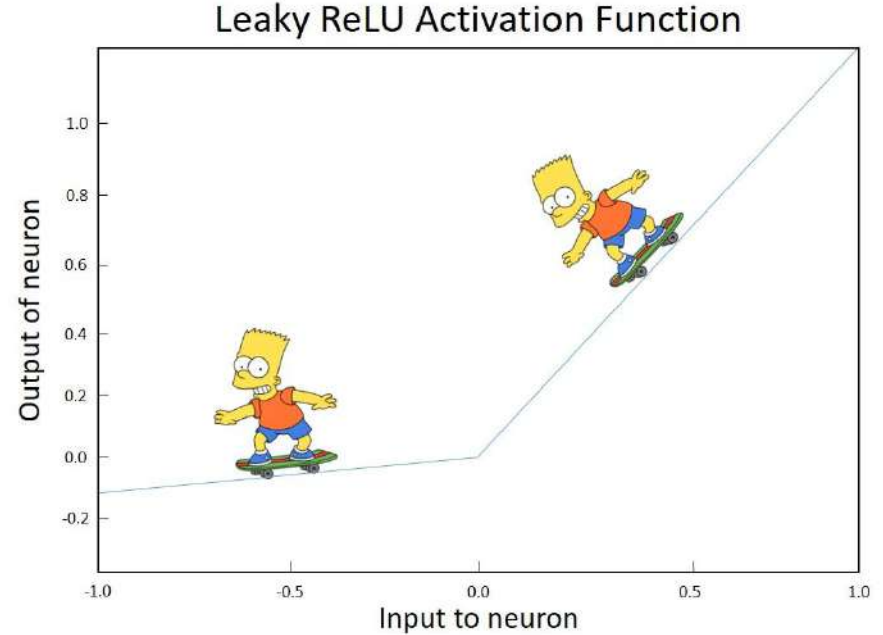
Dying ReLU Problemi

Dying ReLU’da problem olan şey, nöronların çoğu negatif input yüzünden sıfırlandığında, back propagation sırasında ağırlıkların güncellenemeyecek olmasıdır.

ReLU’nun yetersiz kaldığı bu durumda Leaky ReLU kullanılabilir.

Leaky ReLU, negatif inputlar için 0 değeri üretmeyen bir aktivasyon fonksiyonudur.

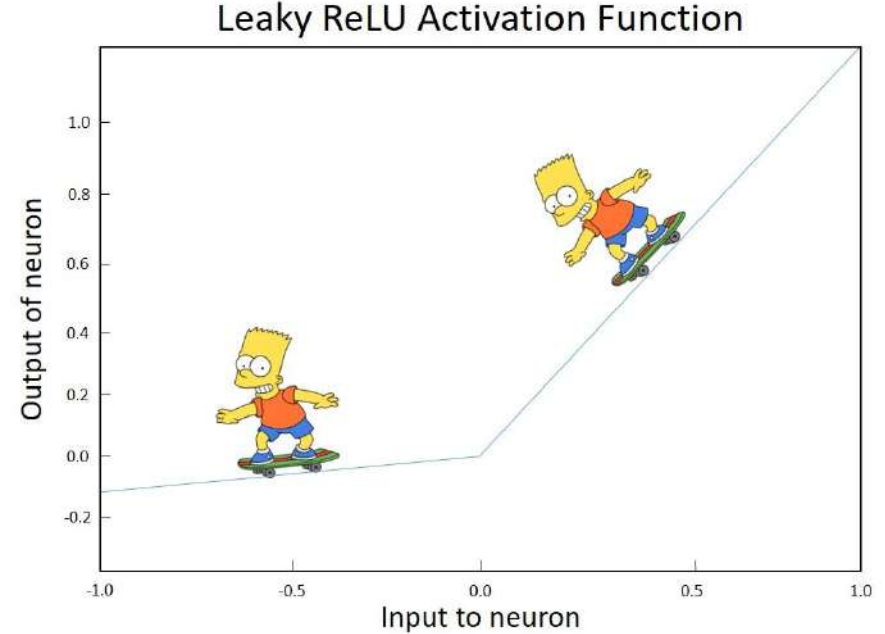
Negatif inputların fazla olduğu senaryolarda bu inputlar da türevlenebileceğinden, back propagation işlemi ağırlıkları güncelleyebilecektir.



Dying ReLU Problemi

Optimizerlar her seferinde birden çok giriş değerini dikkate aldığından, Dying ReLU problemi her zaman gerçekleşmez.

Tüm inputlar ReLU'yu negatif segmente itmediği sürece nöronlar aktif kalabilir, ağırlıklar güncellenebilir ve ağ öğrenmeye devam edebilir.



Basit bir Perceptron Kodlaması

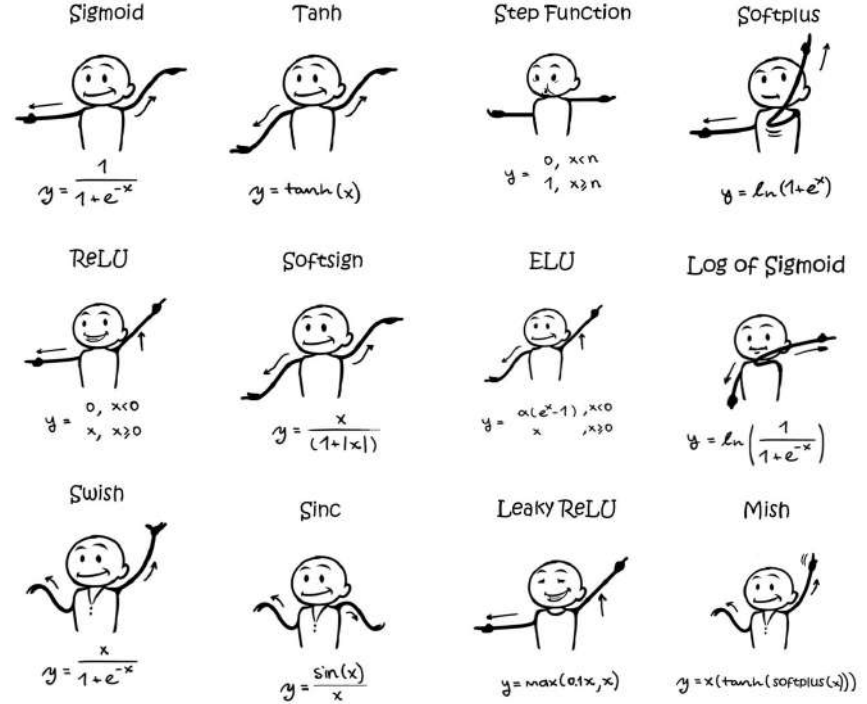
Aktivasyon Fonksiyonları

Aktivasyon Fonksiyonu Nedir?

Aktivasyon fonksiyonu en basit haliyle bir nöronun girdilerini kullanarak hesaplanan lineer bir fonksiyonun, nöronun nasıl çıkacağına karar veren fonksiyondur.

Lineerliği bozmak için kullanılır ve lineerliğin nasıl bozulacağını belirler.

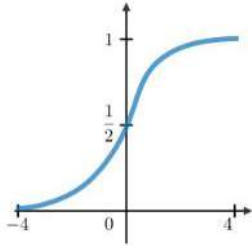
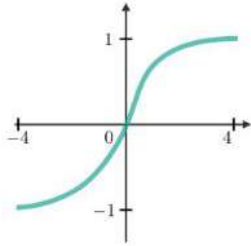
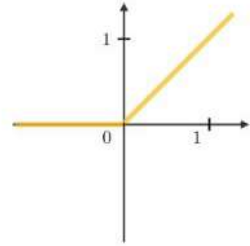
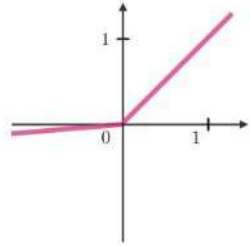
Klasik bir nöron manuel olarak verilmedikçe bir aktivasyon fonksiyonuna sahip değildir ve lineer bir hesaplama yapar.



Aktivasyon Fonksiyonu Neden Kullanılır?

Aktivasyon fonksiyonu olmadan yapay sinir ağıma eklenen katmanlar ağın gücünü arttırmamaktadır. Başka bir deyişle, modelimizdeki katmanlar lineer çıktılar üretiyorsa eklenen katmanlar derin temsillerden yararlanan hipotez uzayını genişletmez ve özellikler öğrenilmez.

Yapay sinir ağının sunduğu konsept, her gizli katmanda yer alan nöronların aktivasyon fonksiyonu yardımıyla belli matematiksel işlemler sonucunda input verisinin lineerliğini bozmasıdır.

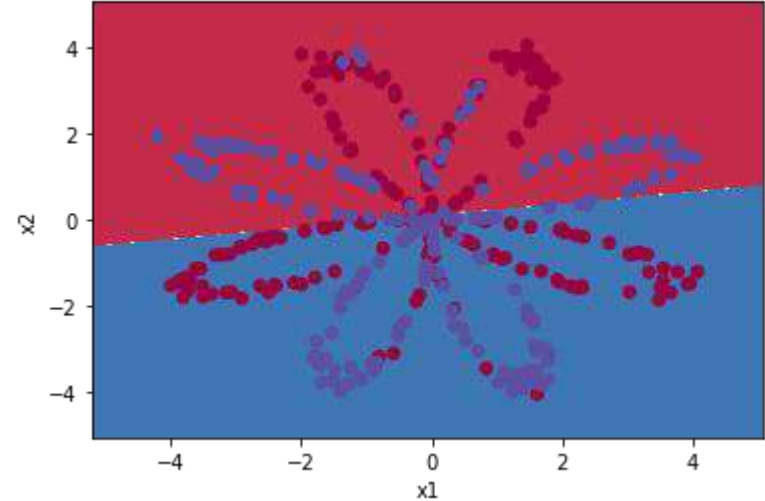
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Aktivasyon Fonksiyonu Neden Kullanılır?

Yandaki örnek için lojistik regresyon modeli oluşturulduğunda modelin accuracy değeri %47 çıkmıştır. (Veri dağılımı eşit olduğundan accuracy metriği kullanılmıştır)

Örneğe bakıldığında, iki sınıf arasındaki ayrımın lineer bir çizgiyle ayrılamayacağı kolayca görülebilir.

Bu tip bir sorunu yapay sinir ağı ile çözerek gereken non-lineerlik gizli katmanlardaki aktivasyon fonksiyonları sayesinde sağlanabilir.



Yaygın Aktivasyon Fonksiyonları

ReLU (Düzeltilmiş Doğrusal Birim)

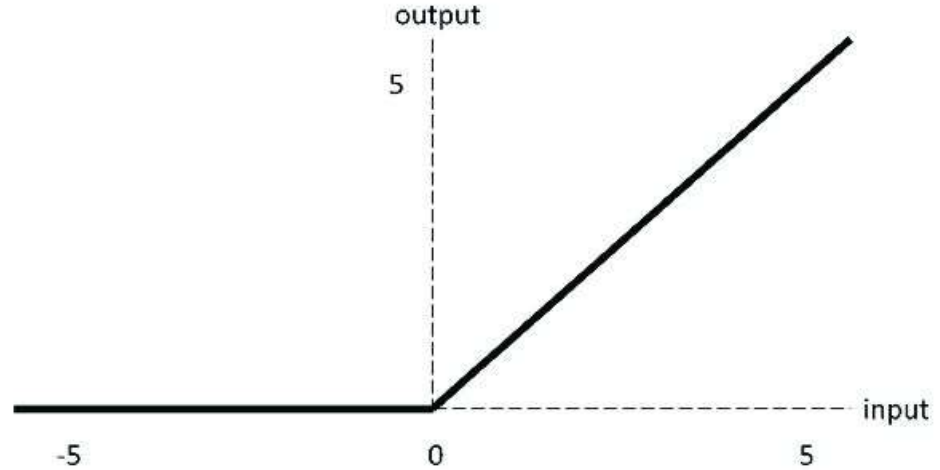
ReLU, negatif değerler için 0, pozitif değerler için ise değerini döndüren bir aktivasyon fonksiyonudur.

ReLU, tanh ve sigmoid ile karşılaştırıldığında daha basit matematiksel işlemler içerir, böylece hesaplama performansını daha da artırır.

ReLU, küçük çıktılar üretmediği için Vanishing Gradient sorununa çözüm olarak kullanılır.

ReLU ağı yalnızca gizli katmanlarında kullanılabilir.

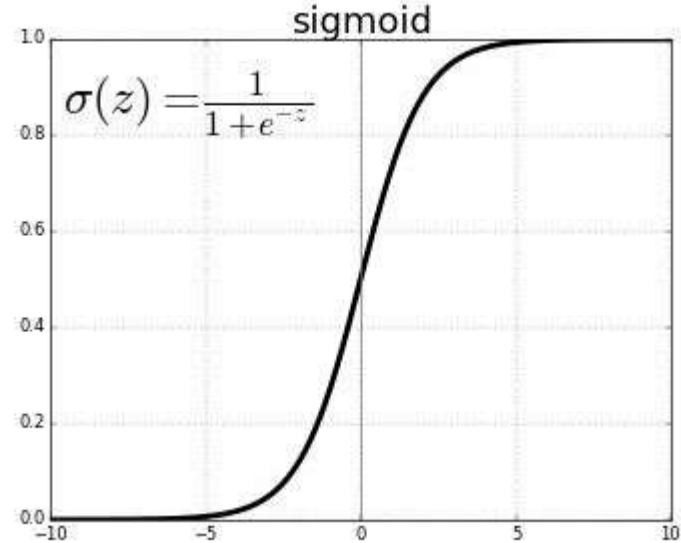
$$f(x) = \max(0, x)$$



Sigmoid Fonksiyonu

Doğrusal olmayan(**non-linear**) bir fonksiyondur. Karar vermeye yönelik olasılıksal bir yaklaşıma sahip bir yapıda olan bu fonksiyonun değer aralığı $[0,1]$ arasındadır.

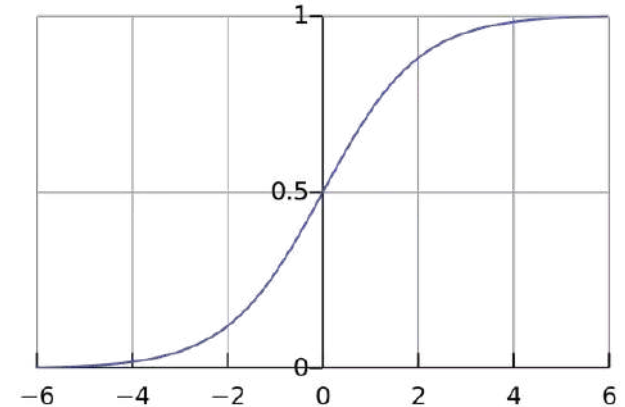
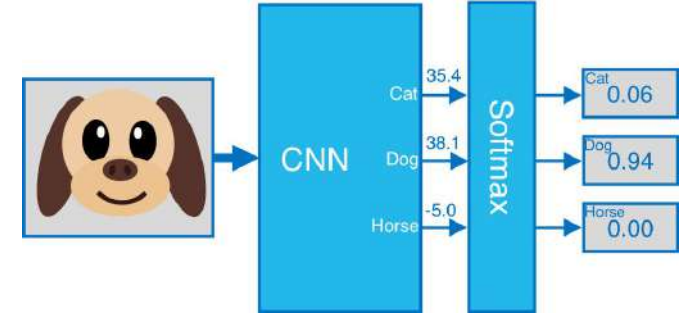
- Bu fonksiyonlardaki x değerinin değişimleri y değeri üzerinde ciddi değişikliğe neden olacaktır
- x (-sonsuz, +sonsuz) arasında ne kadar değişirse değişsin aktivasyon değeri her zaman 0 ile 1 arasında olacaktır



Softmax

Softmax, Çok sınıflı bir sınıflandırma probleminde, çıktı olarak her sınıfa ait olasılık sonucu döndürür.

Çok sınıflı işlemlerde ağınız bir **softmax aktivasyon fonksiyonu** ile bitmelidir, böylece N çıkış sınıfı üzerinde bir olasılık dağılımı verecektir.



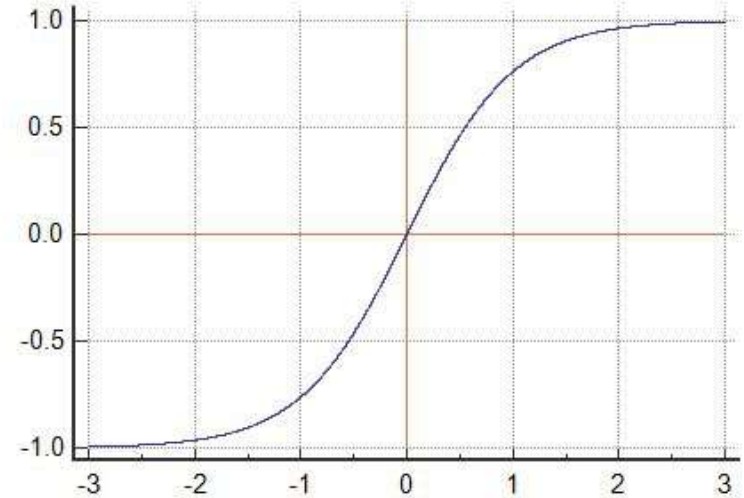
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Tanh Fonksiyonu (Hiperbolik Tanjant)

Karakteristik olarak Sigmoid fonksiyonuna benzese de çıktı 0'da ortalanan ve sınırlar -1 ve 1 olur.

Tanh fonksiyonu ölçeklenmiş Sigmoid fonksiyonudur ve Sigmoid fonksiyonuna göre daha hızlı yakınsar.

Çıktı sınırları Sigmoid'e göre genişlemiş olsa da Vanishing Gradient problemine takılmaya devam eder.

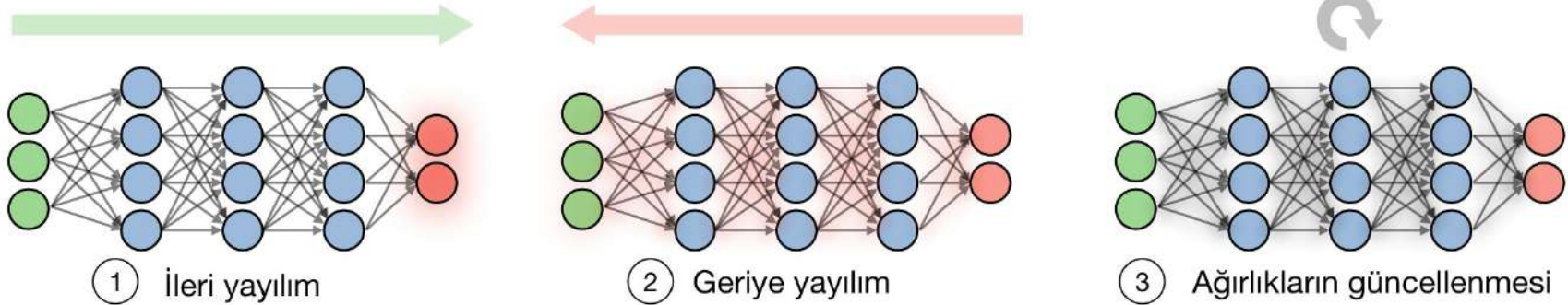


Sinir Ağlarının Optimizasyonu

Backpropagation (Geri Yayılım)

Geri yayılım, istenen çıktıyı dikkate alarak sinir ağındaki ağırlıkları güncellemek için kullanılan bir yöntemdir. Her bir ağırlığa göre türev, zincir kuralı kullanılarak hesaplanır.

- Adım 1: Bir küme eğitim verisi alın ve kaybı hesaplamak için ileriye doğru ilerleyin.
- Adım 2: Her ağırlığa göre kaybın derecesini elde etmek için kaybı tekrar geriye doğru yayın.
- Adım 3: Ağırlıkların güncellenmesi için gradyanları kullanın.



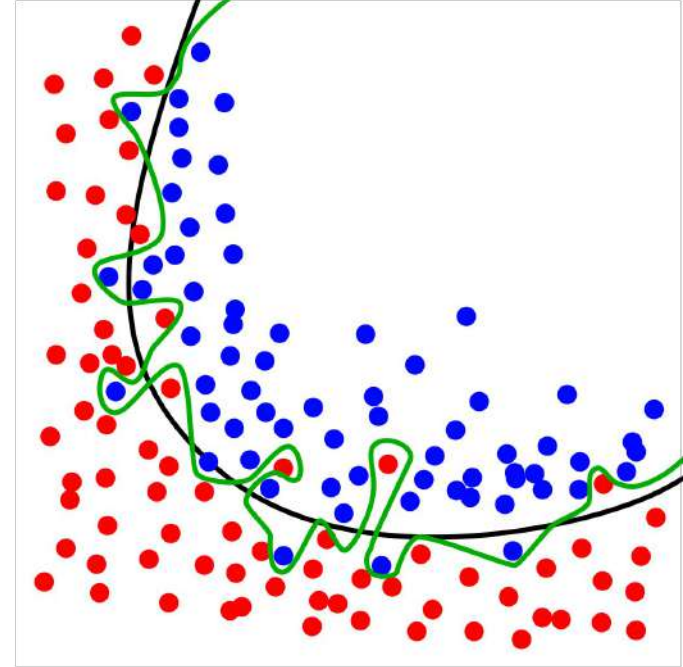
Aşırı Öğrenme (Overfitting)

Algoritma eğitim verilerini sonuçları ile birlikte ezberlediğinde ortaya çıkan bir durumdur. Aşırı öğrenmeye sebep olabilecek durumlar;

- Verinin yetersiz olması
- Fazla epoch değeri verilmesi ve uzun eğitim süresi
- Çok fazla değişken kullanılması

Aşırı öğrenme durumunda model verilerdeki gürültüyü de tanır ve gürültüyü test verilerinde arayarak doğruluğu düşürür.

Aşırı öğrenme oluştuğunda algoritma ilk defa gördüğü verilere karşı verimli bir şekilde çalışamaz.

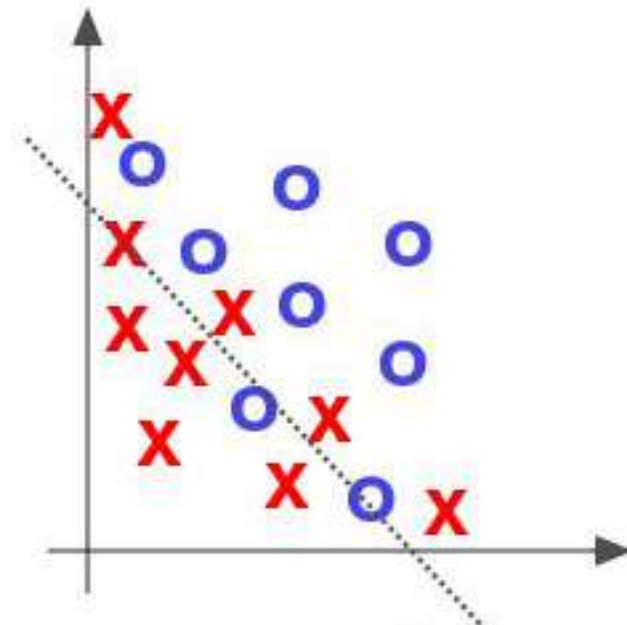


Eksik Öğrenme (Underfitting)

Algoritma, eğitim verilerine yeterince uymaz ve verileri genellemede sorun yaşar. Eksik öğrenme yaşamamak için;

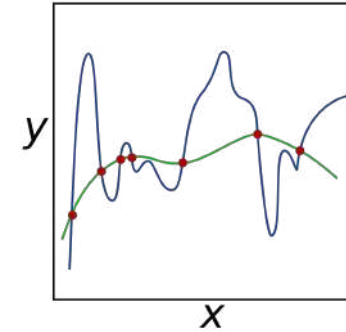
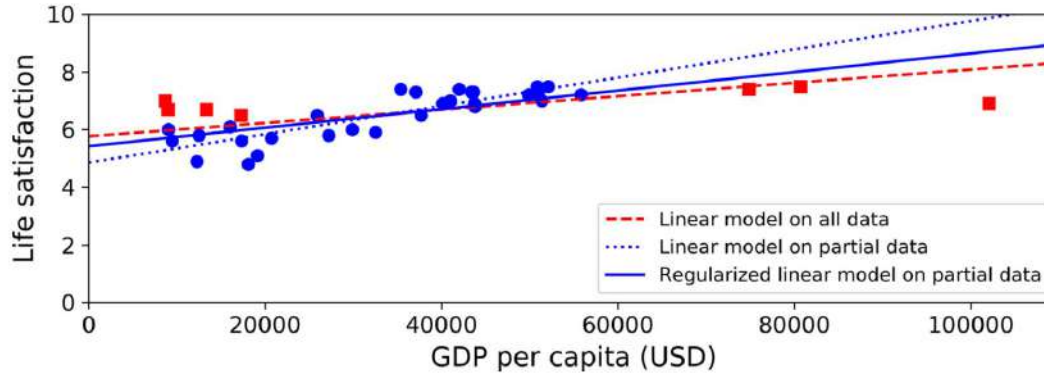
- Eğitim verilerimizi arttırmak
- Daha fazla değişken eklemek
- Değişken ağırlığını arttırmak için Regularization değerini azaltmak

Underfitting yaşandığında, eğitim ve test veri setinde hata oranı yüksektir, düşük varyans ve yüksek bias'a sahiptir



Regülerizasyon

Regülerizasyon, bir modeli basitleştirmek ve buna bağlı olarak **overfit'i önlemek** amacıyla **kısıtlanma** işlemidir. Bu sayede modelin görünmeyen veriler üzerindeki performansı iyileştirir. Bu yöntemde cost functiondaki ağırlıklar, regülerizasyon terimi olarak bilinen başka bir terim eklenerek güncellenir yani cezalandırılır.



Öğrenme sırasında uygulanacak regülerizasyon miktarı bir **regularization term (regülerizasyon parametresi)** isimli hiper parametre ile kontrol edilebilir.

Regülerizasyon

L1 Lasso

- L1 Regülerizasyonda hata fonksiyonu, ağırlıkların mutlak değeri ile cezalandırılır
- L1'in türevi k'dir (değeri ağırlıktan bağımsız olan bir sabit)
- L1'in türevini her seferinde ağırlıktan bir miktar sabit çıkaran bir kuvvettir

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right]$$

}

L2 Ridge

- L2 Regülerizasyonda hata fonksiyonu, ağırlıkların karesi ile cezalandırılır
- L2'nin türevi 2 * ağırlıktır
- L2'nin türevini her seferinde ağırlığın %x'ini kaldıran bir kuvvettir

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_0^{(i)}$$

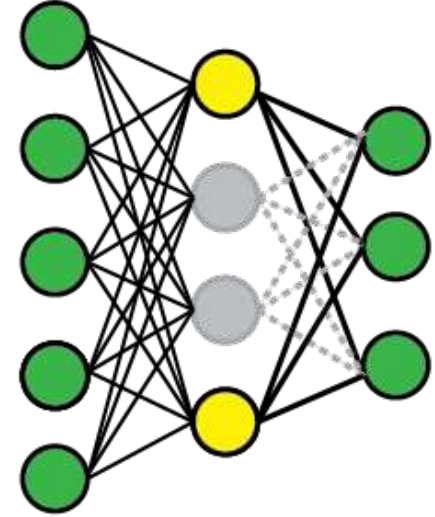
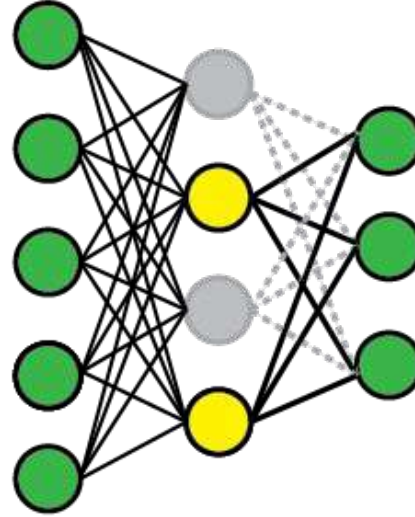
$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j^2 \right]$$

}

Dropout

Dropout, işlemeyi ve sonuçlara ulaşma süresini iyileştirmek için bir sinir ağından kasıtlı olarak bırakılan verileri veya gürültüyü ifade eder.

Yapay sinir ağları içinde birbiriyle iletişim kuran milyonlarca nöron vardır. Bir ağ, üzerinde çalıştığı problem ile doğrudan ilgili olmayan nöron düğümleri tarafından iletilen tüm iletişimleri ortadan kaldırır.

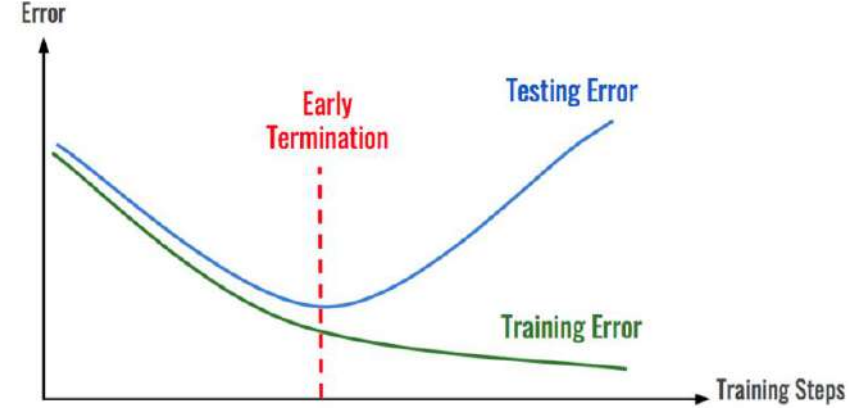


Erken Durdurma (Early Stopping)

Gradient Descent gibi yinelemeli öğrenme algoritmalarını düzenli hale getirmenin çok farklı bir yolu, doğrulama hatası **minimuma** ulaşır ulaşmaz eğitimi durdurmaktır

2018 Turing Ödülü'nü Yoshua Bengio ve Yann LeCun'la birlikte almaya hak kazanmış Geoffrey Hinton'ın "**No Free Lunch Theorem**" dediği basit ve etkili bir düzenleme tekniğidir

Minimum bir hata değeri belirlenir ve validasyon seti hatası bu sınırın altına indiğinde veya sınıra eşit olduğunda n kadar daha iterasyon gerçekleşir. Eğer performansımız iyileşmezse hata değerimiz en iyi noktaya geri döner ve eğitim durur (*callback*)



Data Augmentation

Veri analizinde veri arttırma, halihazırda var olan verilerin biraz değiştirilmiş kopyalarını veya mevcut verilerden yeni oluşturulan sentetik verileri ekleyerek veri miktarını arttırmak için kullanılan bir tekniktir.

Bir makine öğrenimi modelini eğitirken overfitting oluşmamasına yardımcıdır.

Veri analizinde aşırı örnekleme (oversampling) ile yakından ilgilidir.

Orjinal	Çevirme	Rotasyon (Yönlendirme)	Rastgele kırpmak/kesme
			
<ul style="list-style-type: none"> Herhangi bir değişiklik yapılmamış görüntü 	<ul style="list-style-type: none"> Görüntünün anlamının korunduğu bir eksene göre çevrilmiş görüntü. 	<ul style="list-style-type: none"> Hafif açılı döndürme Yanlış yatay kalibrasyonu simüle eder 	<ul style="list-style-type: none"> Görüntünün bir bölümüne rastgele odaklanma Arka arkaya birkaç rasgele kesme yapılabilir

Renk değişimi	Gürültü ekleme	Bilgi kaybı	Kontrast değişimi
			
<ul style="list-style-type: none"> RGB'nin nüansları biraz değiştirilmesi Işığa maruz kalırken oluşabilecek görüntü 	<ul style="list-style-type: none"> Gürültü ekleme Girdilerin kalite değişkenliğine daha fazla toleranslı olması 	<ul style="list-style-type: none"> Yok sayılan görüntüler Görüntünün parçalardaki olası kayıplarını kopyalanması 	<ul style="list-style-type: none"> Gün içindeki ışık ve renk değişiminin kontrolü

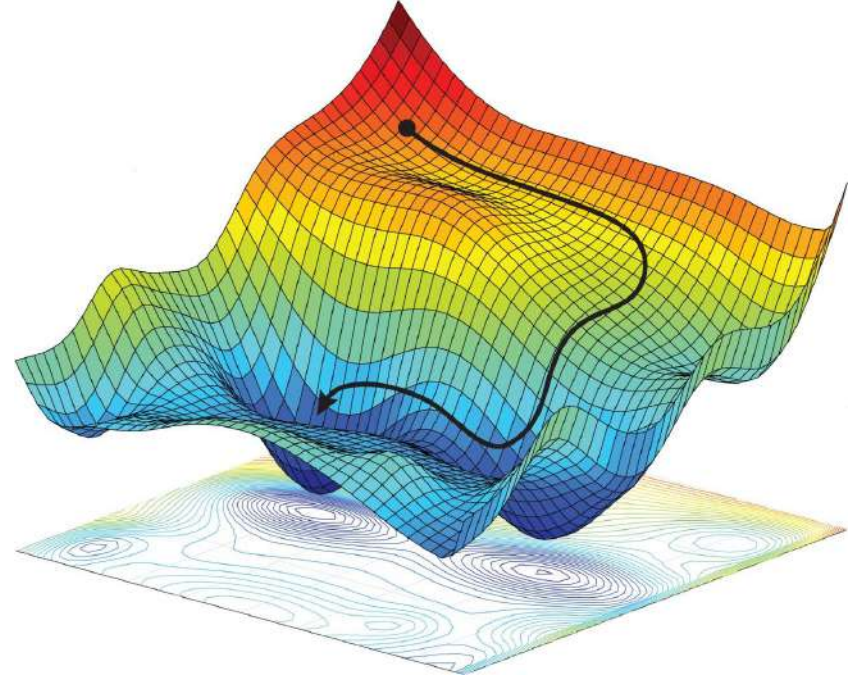
Optimizer

Optimizer Nedir?

Optimize ediciler, bir hata fonksiyonunu (loss function) en aza indirmek ve doğruluk oranını en üst seviyeye çıkartmak için kullanılan yöntemlerdir.

Optimizerler modelin öğrenilebilir parametrelerine (Weights & Biases) bağlı matematiksel işlevlerdir.

Optimize ediciler kayıpları azaltmak için sinir ağının ağırlıklarının ve öğrenme oranının nasıl değişeceğini bilmeye yardımcı olurlar.



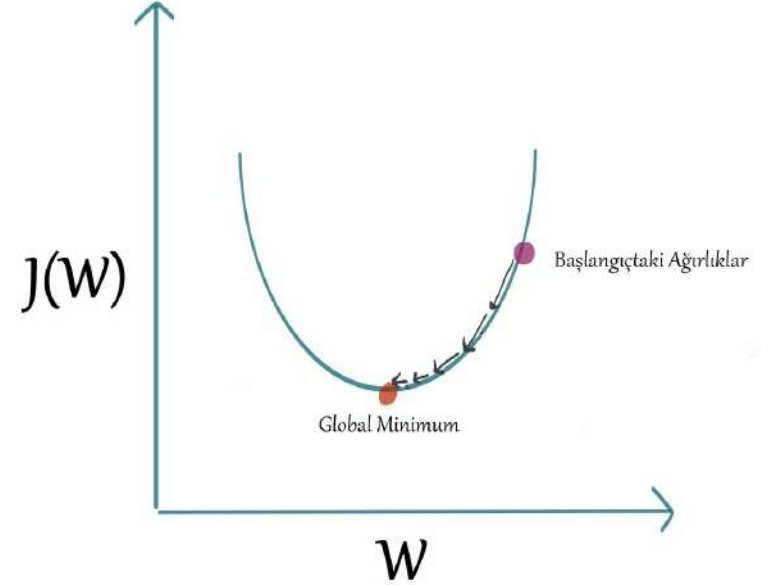
Yaygın Optimizasyon Algoritmaları

Gradient Descent

Gradyan iniş, dışbükey bir fonksiyona dayalı bir optimizasyon algoritmasıdır ve verilen bir fonksiyonu yerel minimuma en aza indirmek için parametrelerini yinelemeli olarak ayarlar.

Bu yöntem, tüm veri kümesi için gradyanı tek bir güncellemede hesapladığı için, hesaplama çok yavaş ve pahalıdır. Bu nedenle büyük bellek gerektirir.

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

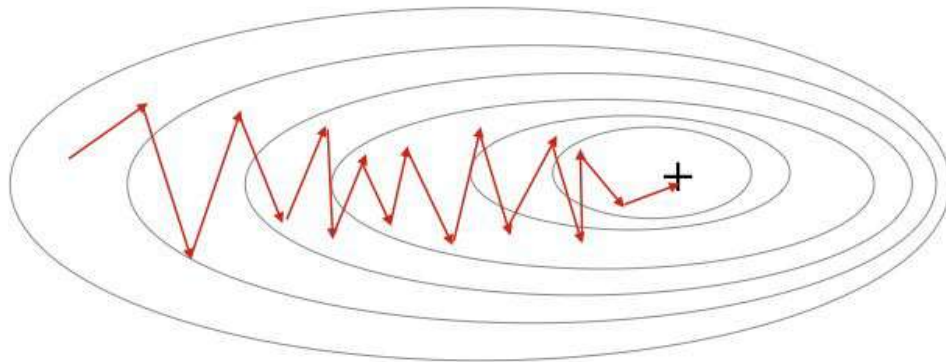


Stochastic Gradient Descent

Gradient Descent algoritmasının bir çeşidi olan Stokastik gradyan Descent (SGD), her eğitim örneği için bir parametre güncellemesi gerçekleştirir.

Daha az bellek gerektirmesi, model parametrelerinin sürekli güncellenmesi, büyük veri kümelerinin kullanımına daha uygun olması Gradient Descent Algoritmasına kıyasla avantajlarındandır.

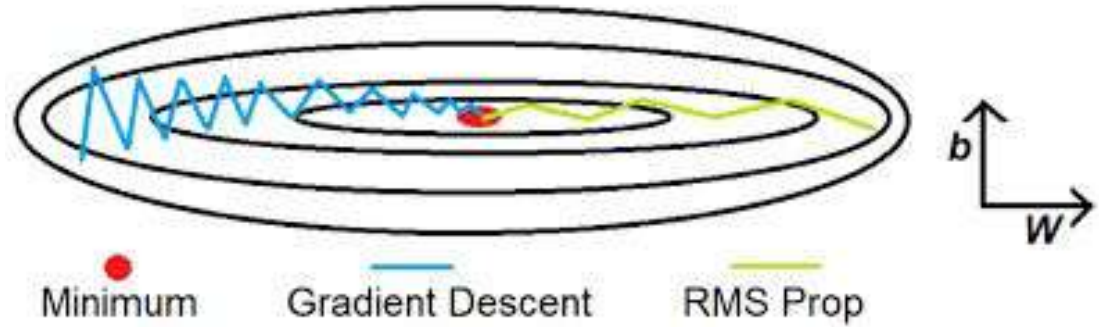
Modelde 10K veri kümesi varsa, SGD model parametrelerini 10K kez güncelleyecektir.



RMS-Prop (Root Mean Square Propagation)

RMSprop, sinir ağlarının eğitiminde kullanılan gradyan tabanlı bir optimizasyon tekniğidir.

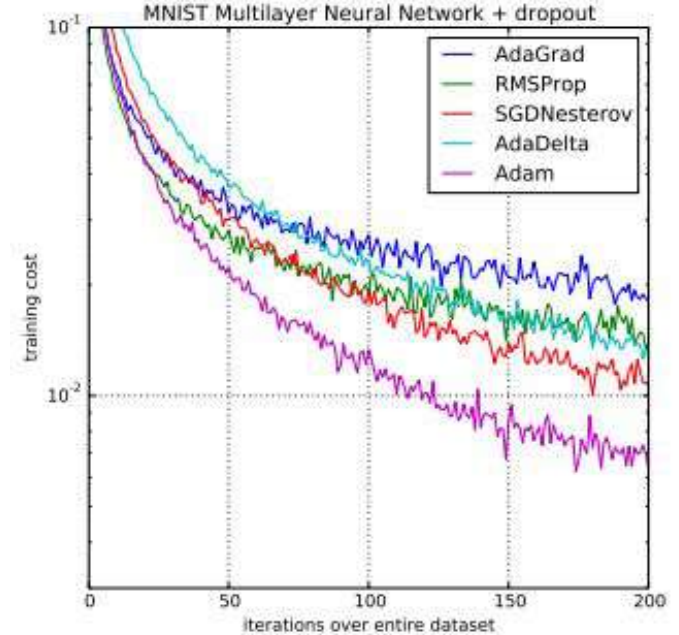
RMSprop, öğrenme oranını bir hiper parametre olarak ele almak yerine uyarlanabilir bir öğrenme hızı kullanır. Bu, öğrenme hızının zamanla değiştiği anlamına gelir.



Adam (Adaptive Moment Estimation)

Adam, uyarlanabilir bir öğrenme oranı yöntemidir, yani her parametre için uyarlanabilir öğrenme oranlarını hesaplar

Adam optimizier adı, uyarlanabilir moment tahmininden türetilmiştir ve bu şekilde adlandırılmasının nedeni, Adam'ın sinir ağının her ağırlığı için öğrenme oranını uyarlamak için birinci ve ikinci gradyan anlarının tahminlerini kullanmasıdır.



Yapay Sinir Ağında Eğitim Parametreleri

Dönem (Epoch)

Verinin tamamının eğitim için modelden geçme sayısıdır.

Tüm eğitim setini kullanarak, modelin ağırlıklarını güncellemek için eğitim adımlarını yinelemeyi amaçlar.

Her epoch yenilemesinde ağırlıklar güncellenir. Kayıp değeri düşürülmeye, accuracy değeri arttırılmaya çalışılır. Bu işlemler belirtilen optimizer ve loss fonksiyonu ile yapılır.

*Üniversite sınavına hazırlanan bir öğrencinin elindeki soru bankasını baştan sona bitirme sayısı, her bitirmesinde bazı eksikliklerini kapatması ve doğruluğu yükseltmeye çalışması **Epoch** kavramına bir örnektir. Baştan hedeflediği sayıda kez soru bankasını çözecektir.*

```
Epoch 1/3
32/32 [=====] - 0s 721us/step - loss: 0.5791 - mae: 0.6232
Epoch 2/3
32/32 [=====] - 0s 601us/step - loss: 0.2739 - mae: 0.4296
Epoch 3/3
32/32 [=====] - 0s 576us/step - loss: 0.2547 - mae: 0.4078

<tensorflow.python.keras.callbacks.History at 0x1423856d0>
```

Batch Size

Verimizin sayısı fazla olduğunda tek seferde eğitmeden geçirmek bellekte daha yüksek işlem maliyetine ve dolayısı ile performans kaybına yol açmaktadır.

Batch size hiper-parametresi, ağ üzerinden tek seferde yayılacak veri sayısını tanımlar. Veriniz, batch size kadar belleğinize verilerek eğitim gerçekleşir ve tüm veriniz belleğinize verilene kadar epoch tamamlanmış olmaz.

Üniversite sınavına hazırlanan bir öğrencinin elindeki soru bankasını baştan sona bitirmek için kitabı kaç sayfaya böleceği ve kaçar sayfayı aynı anda çalışacağı Batch Size parametresine örnektir.

```
Epoch 1/3
32/32 [=====] - 0s 721us/step - loss: 0.5791 - mae: 0.6232
Epoch 2/3
32/32 [=====] - 0s 601us/step - loss: 0.2739 - mae: 0.4296
Epoch 3/3
32/32 [=====] - 0s 576us/step - loss: 0.2547 - mae: 0.4078

<tensorflow.python.keras.callbacks.History at 0x1423856d0>
```

Soru Vakti!