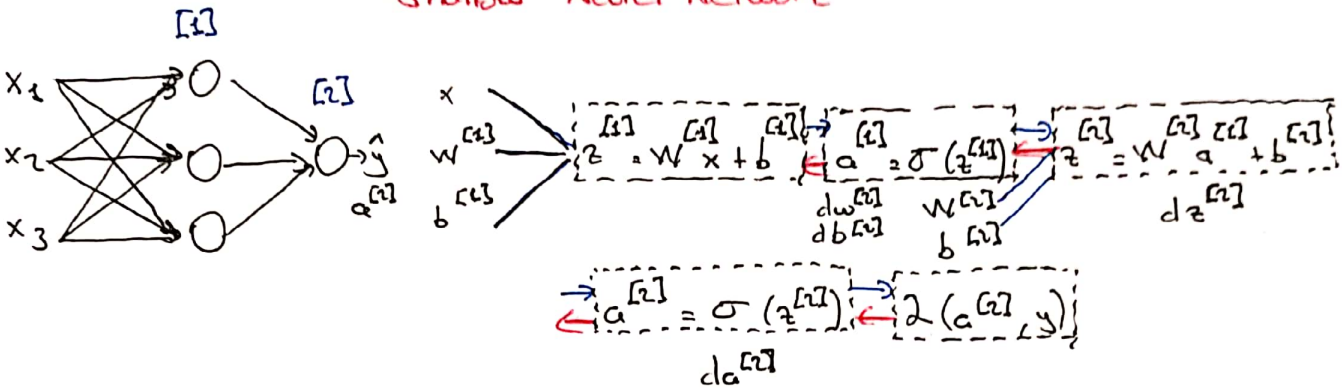
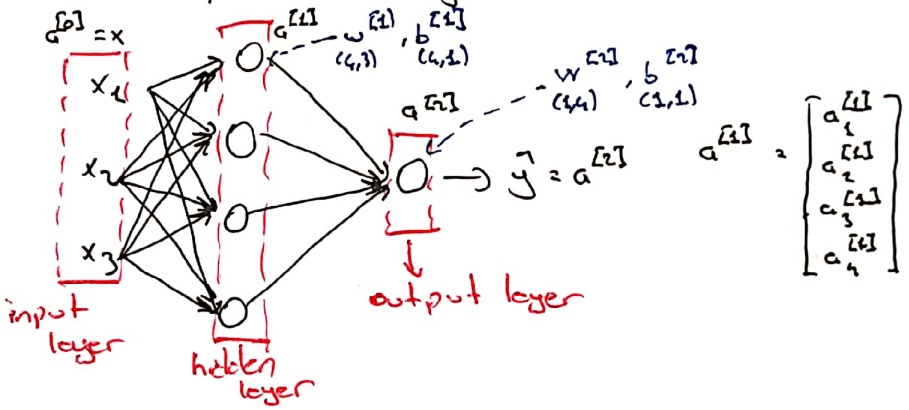
Shallow Neural Network



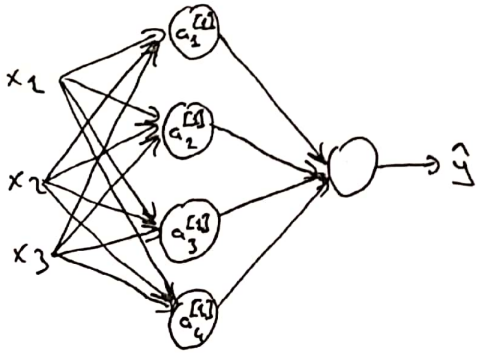
Sinir ağında her bir nöron hem doğrusal bir fonksiyon hesaplar hem de bir aktivasyon fonksiyonu hesaplar.
Katmanların gösterimi köşeli parantez ile gösterilmektedir.

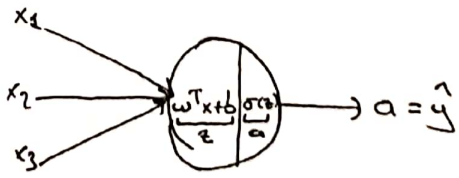


Input layer, özellik girdilerimiz. Output layer, çıktı verimiz. Bu iki sütun eğitim örneklerimizde bulunurken hidden layer girdilerimiz bulunmaz. Bu bir nevi neden gizli olarak adlandırıldığını açıklar.

Yukarıdaki örnek sinir ağı 2 katmanlıdır. Çünkü sinir ağının giriş katmanları sayılmaz.

Computing a Neural Network Output





$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

Bir sinir ağında bir katmandaki nöronların nasıl hesaplandığını gördük. Gelin bunun vektörizasyonunu yapalım.

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ $\qquad$ $b^{[1]}$

1 layer $\Rightarrow$ $z^{[1]} = W^{[1]} x + b^{[1]}$, $a^{[1]} = \sigma(z^{[1]})$ $\qquad$ 2 layer $\Rightarrow$ $z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$, $a^{[2]} = \sigma(z^{[2]})$

Yukardaki dört denklemi kullanarak gizli katmanlı sinir ağını eğitebilirsiniz. Logistic regresyondan çokda büyük bir farkı yoktur.

Multiple Examples Vectorizing

En başta tek bir örnek üzerinde tek katmanlı gizli bir sinir ağının nasıl vectorize edildiğini görmüştük. Şimdi çoklu örnekte nasıl vectörize edip eğiticeğimize bakacağız.

$x \longrightarrow a^{[2]} = \hat{y}$

$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[2](m)}$: layer 2, example i

```
for i = 1 to m,
    z[1](i) = W[1] x(i) + b[1]
    a[1](i) = σ(z[1](i))
    z[2](i) = W[2] a[1](i) + b[2]
    a[2](i) = σ(z[2](i))
```

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

train examples →

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$ ↓ hidden units

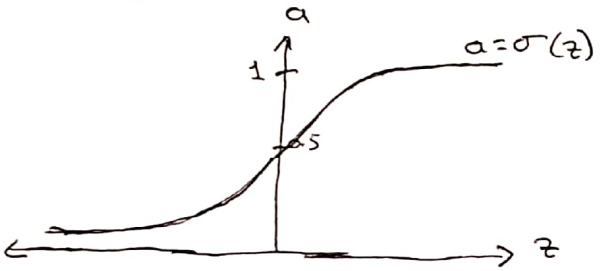
$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Activation Functions

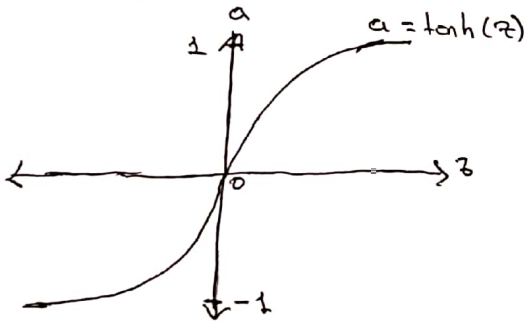
Halihazırda çokça kullandığımız sigmoid fonksiyonu bir aktivasyon fonksiyonudur. Ancak bazı aktivasyon fonksiyonları daha iyi sonuçlar verebiliyor.

1. Sigmoid Function



$$a = \frac{1}{1+e^{-z}}$$

2. Tangent Function (Hiperbolik, Tanjant)

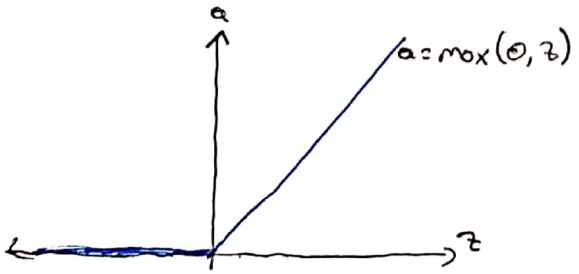


$$a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

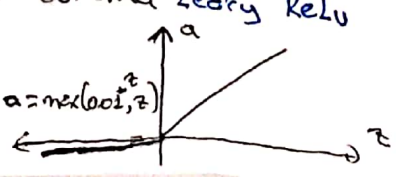
- Sigmoid fonksiyonundan daha iyi çalışır.
- Sigmoid fonksiyonunun matematiksel olarak kaydırılmış halidir.
- Verilerinizi merkeze ortalayacağından (0) bir sonraki katman için öğrenmeyi kolaylaştırır.
- Tanjant aktivasyon fonksiyonu her halükarda sigmoid fonksiyonun çok üstünde bir verimliliği verdir. Tek istisna ikili bir sınıflandırma yapacaksanız değer $0 < \hat{y} < 1$ arasında olmalıdır. Ancak tanjant bize $-1 < a < 1$ arasında sonuç döndürür. Bunun için sinir ağlarının son çıkış katmanında sigmoid fonksiyon kullanabilirsiniz.

Not Hem sigmoid fonksiyonunun hem de tanh fonksiyonunun dezavantajlarından biri, z çok büyük veya çok küçükse, bu fonksiyonun gradyan veya türevi (eğim) 0'a yakın olur. Bu da gradyan inişini yavaşlatabilir.

3. ReLU (Rectified Linear Unit)



- Pozitif olduğu sürece (z) türevi 1'dir. Negatif veya 0 ise türevi 0'dır. z=0 iken tanımsızdır aslında fonksiyonla çözeriz.
- Tek dezavantajı z negatif olduğunda türevi 0 olmasıdır. Bu da ~~gradyan~~ inişinde hafif bir eğimle inmesine neden olur. Bu duruma Leaky ReLU (Sızdıran Relu) deniyor.



- İşlem yükünü oldukça hafiftir.

Neden Aktivasyon Fonksiyonlarına İhtiyaç Duyarız?

$$a^{[1]} = z^{[1]} = w^{[1]} x + b^{[1]}$$
$$a^{[2]} = z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = w^{[2]}\left(w^{[1]} x + b^{[1]}\right) + b^{[2]}$$
$$= \underbrace{(w^{[2]} w^{[1]})}_{w'} x + \underbrace{(w^{[2]} b^{[1]} + b^{[2]})}_{b'}$$
$$= w' x + b'$$

Eğer aktivasyon fonksiyonlarını kullanmazsak birçok gizli katmandaki nöronun hesaplanması yine bize bir lineer fonksiyon verir.
Lineer bir gizli katman bir işe yaramaz. Çünkü iki lineer fonksiyonun birleşimi kendisine yani bir lineer fonksiyona eşittir.

Not Lineer fonksiyon regresyon problemlerinde kullanılabilir.

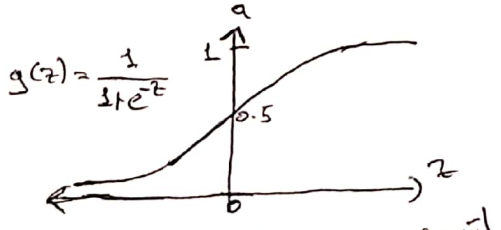
Daha derin ağlara ilerlerken anlamlı fonksiyonlar elde edebilmek için lineer fonksiyonu non-lineer şekle çevirerek lineer olmayan problemlere yanıt arayabiliriz.

Aktivasyon Fonksiyonlarının Türevi

Yapay sinir ağı için geri yayılımı (back propagation) uygularken, aktivasyon fonksiyonlarının türevlerini hesaplayabilmeniz gerekir.

1. Sigmoid Function

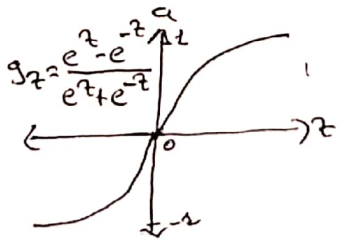
$g(z) = \frac{1}{1+e^{-z}}$



$$\frac{d}{dz} g(z) = -\frac{\frac{d}{dz}[e^{-z}+1]}{(e^{-z}+1)^2} \xrightarrow{\text{formül}} -\frac{1'}{1^2}$$
$$= \frac{e^{-z}}{(1+e^{-z})^2}$$
$$= \left(\frac{1}{1+e^{-z}}\right)\cdot\left(1 - \frac{1}{1+e^{-z}}\right)$$
$$a(1-a) = g(z)\cdot(1-g(z))$$

2. Tanh Function

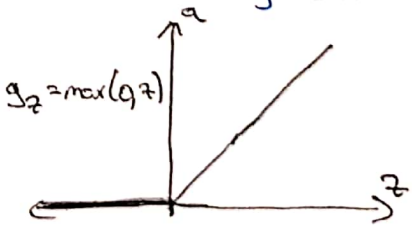
$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



$$g'(z) = \frac{(e^z+e^{-z})\cdot(e^z+e^{-z}) - (e^z-e^{-z})\cdot(e^z-e^{-z})}{(e^z+e^{-z})^2}$$
$$= \frac{(e^z+e^{-z})^2 - (e^z-e^{-z})^2}{(e^z+e^{-z})^2}$$
$$= 1 - \frac{(e^z-e^{-z})^2}{(e^z+e^{-z})^2}$$
$$= 1 - (g(z))^2 = 1 - a^2$$

3. ReLU ve Leaky ReLU

$g(z) = \max(0, z)$



$$g'(z) = \begin{cases} 0, & z<0 \\ 1, & z \geq 0 \end{cases}$$

$g(z) = \max(0.01z, z)$

$$g'(z) = \begin{cases} 0.01 & z<0 \\ 1 & z \geq 0 \end{cases}$$

Gradient Descent for Neural Networks

Parameters: $W^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $W^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$ $(n^{[2]}, 1)$ $\quad n_x = n^{[0]} = 3$, $n^{[1]} = 4$, $n^{[2]} = 1$

katmandaki düğüm sayısı

Cost Function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$ $\quad \hat{y} = a^{[2]}$

Gradient Descent:

Repeat {

Compute predicts ($\hat{y}^{(i)}$, $i = 1 \dots m$)

$dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$

$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

$dW^{[2]} = \frac{dJ}{dW^{[2]}}$, $db^{[2]} = \frac{dJ}{db^{[2]}}$

$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$

$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$

}

Forward Propagation

$z^{[1]} = W^{[1]} x + b^{[1]}$

$A^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(z^{[2]})$

Backpropagation

$dz^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[2]T}$

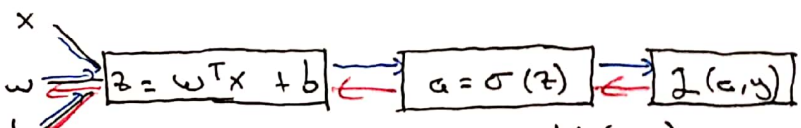
$db^{[2]} = \frac{1}{m}\, np.sum(dz^{[2]}, axis = 1, keepdims = True)$

$dz^{[1]} = \underbrace{W^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(z^{[1]})}_{(n^{[1]}, m)}$ $\quad *$ eleman çarpımı

$dW^{[1]} = \frac{1}{m} dz^{[1]} x^T$

$db^{[1]} = \frac{1}{m} \cdot np.sum(dz^{[1]}, axis = 1, keepdims = True)$

Backpropagation Sezgi



$dz = a - y$ $\quad \sigma(z)$

$da = \frac{d\mathcal{L}(a,y)}{da} = -y \log a - (1-y) \log(1-a)$

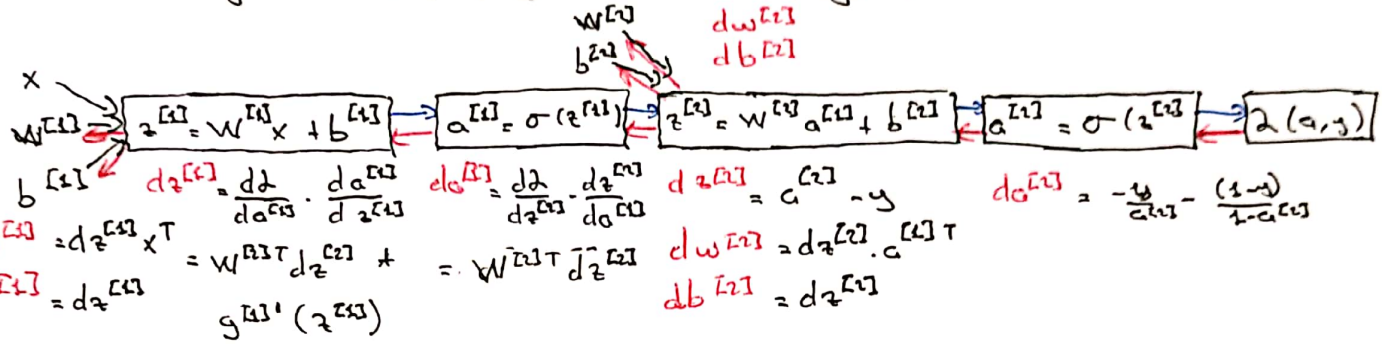
$dz = da \cdot g'(z) = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$

$= da \cdot \frac{d}{dz} g(z)$ $\quad = -\frac{y}{a} + \frac{1-y}{1-a}$

$dw = dz \cdot x$

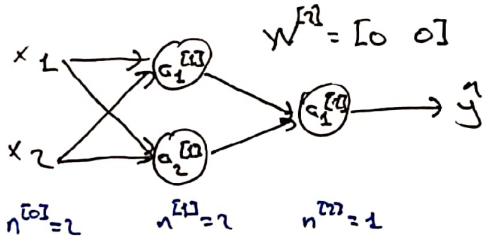
$db = dz$

Lojistik regresyonda back propagation tek seferde gerçekleşir. Ancak sinir ağlarında katman seviyesi arttıkça işlem tekrarında artan Gelin 2 katmanlı bir sinir ağın back propagation'ını hesaplayalım.



Parametre Ağırlıklarının Random Ayarlanması

Lojistik regresyonda ağırlıkların sıfıra ayarlanması ile gradient çalışır ancak sinir ağında uygulanan gradyan inişi çok fazla işe yaramaz.



Eğer ağırlık parametrelerini aynı değerler ile başlatırsanız. Her iki nöronda birbirinin simetriği olacaktır. Nöronlar aynı değerleri yolleyeceğinden $z^{[1]}$ ve $z^{[2]}$ türevleride aynı çıkar.

$$dw^{[1]} = \begin{bmatrix} u & v \\ u & v \end{bmatrix} \qquad W^{[1]} = W^{[1]} - \alpha dW$$

$dw^{[1]}$'in türevi yukarıdaki gibi aynı iki satırdan oluşacak ve parametreleri güncelleğimizde benzerlik devam edecek ve nöronlar iterasyonlar geçsede simetrikliğini koruyacak. Bu durum sinir ağını amaçsız bir durum haline getiriyor. Bunun çözümü parametreleri rastgele başlatmaktır. Ağırlıkları rastgele başlattığımızda bias değerimizi 0 başlatmak sorun teşkil etmiyor.

```
W[1] = np.random.rdn((2,2)) * 0,01
b    = np.zero((2,1))
```

Not: Parametre ağırlıklarını 0.01 gibi küçük bir sayı ile çarparak küçültmemizin nedeni eğer çok yüksek bir sayı ile çarparsak sigmoid fonksiyonuna sokacağımız "z" çok büyük veya çok küçük olur bu da eğimin 0'a yakınsamasından gradyan inişini yavaşlatır. Bu durum sigmoid ve tanh aktivasyon fonk.'larında geçerlidir.