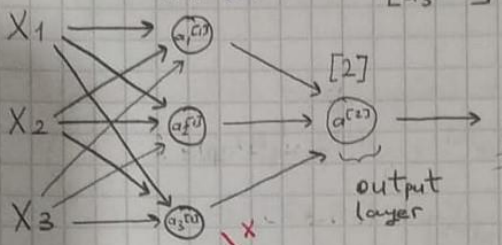
3) Shallow Neural Network

$a^0 = X$ $\quad a^{[1]}$ [1] $\rightsquigarrow a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$

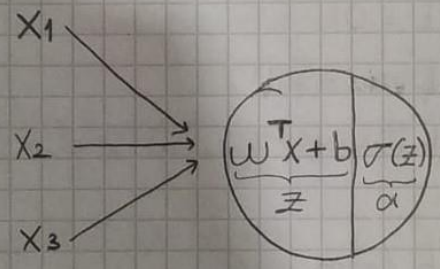


$x \rightarrow W^{[1]}$, $b^{[1]}$

$z^{[1]} = W^{[1]}x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$ (sigmoid; $W^{[2]}$, $b^{[2]}$)

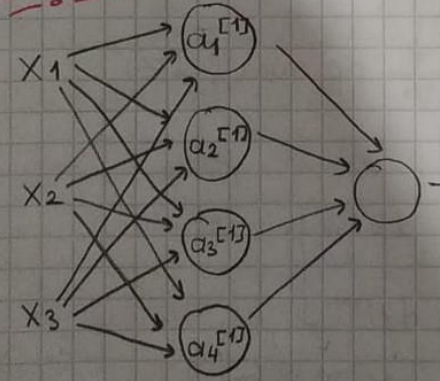
$\downarrow$

$\mathcal{L}(a^{[2]}, y) \leftarrow a^{[2]} = \sigma(z^{[2]})$



$\rightarrow a = \hat{y}$ $\quad z = w^T x + b$, $\quad a = \sigma(z)$

Gösterim



$\rightarrow \hat{y}$

$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}$, $a_1^{[1]} = \sigma(z_1^{[1]})$

$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}$, $a_2^{[1]} = \sigma(z_2^{[1]})$

$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}$, $a_3^{[1]} = \sigma(z_3^{[1]})$

$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}$, $a_4^{[1]} = \sigma(z_4^{[1]})$

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

Hepsi single Training example için.

Vectorizing across Multiple Training Examples:

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

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

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

$\vdots$

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

ÖR

$a^{[2](i)}$ → Refers to training example i.

↓ layer 2

Different Training Examples →

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

$$Z = \begin{bmatrix} z^{[1](1)} & \cdots \\ \vdots & \\ z^{[n](1)} & \cdots \end{bmatrix}$$ ↓ Hidden layers

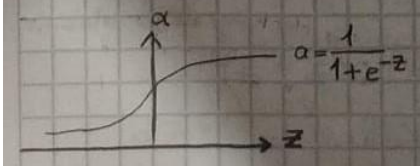
Different Training Examples →

$$A = \begin{bmatrix} a^{[1](1)} & \cdots \\ \vdots & \\ a^{[n](1)} & \cdots \end{bmatrix}$$ ↓ Hidden layers

Different training examples →

Activation Functions:

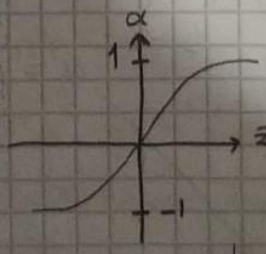
Sigmoid function'dan farklı olarak, başka non linear fonksiyonlarda kullanılabilir.



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

Sigmoid Func.

$\sigma(z)$



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

$g(z)$

tanh Func.

tanh, değerleri 0'a center'ladığı için Sigmoid'e göre daha iyidir. tanh'da -1 ile 1 arasında değişir değerler.

↳ output layer'da "Sigmoid func." kullanmak gerekir, hidden'da tanh kullansak bile. Çünkü 0 ile 1 arasında olmasını isteriz Output'un.

* Activation functions can be different for different layers.

* Hem tanh hem Sigmoid fonksiyonun z'nin çok küçük veya çok büyük olduğu durumlarda slope of fonksiyon ends up being close to 0. Bu da Gradient Descent'i yavaşlatır.

↳ Örneğin

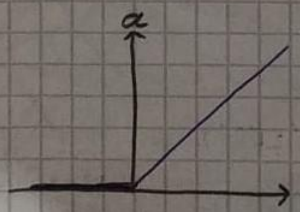
$$w_{(i)} = w_{(i)} - \alpha \, dw_{(i)}$$

$$dw_{(i)} = \frac{\partial \mathcal{L}}{\partial w_{(i)}} = x \cdot dz$$

dz → çok küçük veya büyük olduğu durumda Gradient Descent çok yavaşlar.

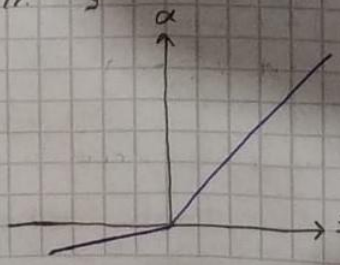
Bu sorunların üstesinden gelmek için;

"Relu" fonksiyonu kullanılır.



Rule of Thump.
Hidden layer için Relu kullan;
Output layer için Sigmoid kullan.

Leaky Relu diye bir aktiv. func var. Relu'dan daha iyi çalışır.

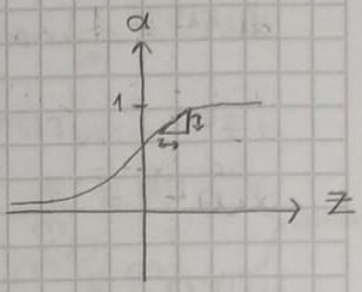


Leaky Relu

Örnek

Örneğin sigmoid func. derivative'sini almak isteyelim.

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



$$\frac{d}{dz} g(z) = \text{slope of } g(x) \text{ at } z.$$

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

$$= g(z)\,(1 - g(z))$$

örneğin Relu fonk. derivative almak isteyelim:

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

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

Şimdi de Leaky ReLU derivative alalım.

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

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

Gradient Descent for Neural Networks (one hidden layer)

Forward Propagation:

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

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

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

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

↳ sigmoid (output layer)

Backpropagation:

$$dZ^{[2]} = A^{[2]} - Y \qquad Y = [\, y^{(1)} \dots y^{(m)} \,]$$

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

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

↳ element wise product

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

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

Summary of Gradient Descent: (one hidden layer)

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

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

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

loss func = ...

$$J(..) = \frac{1}{m} \sum_{i=1}^{n} L(\hat{y}, y)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

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

! Network'e başlarken, W ve b random initialize edilmelidir. (log reg'deki gibi 0'a set edilmemelidir.)

! Başlangıç değerleri 0'a set edilmemelidir, ancak küçük değerlere set edilmelidir.