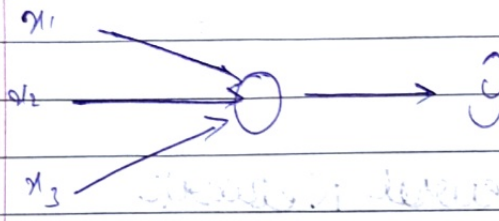
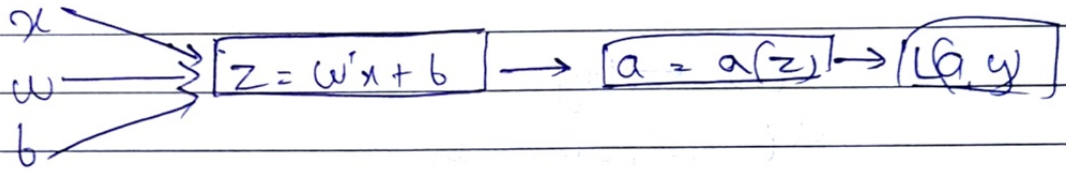
Page No.
Date

Week 3

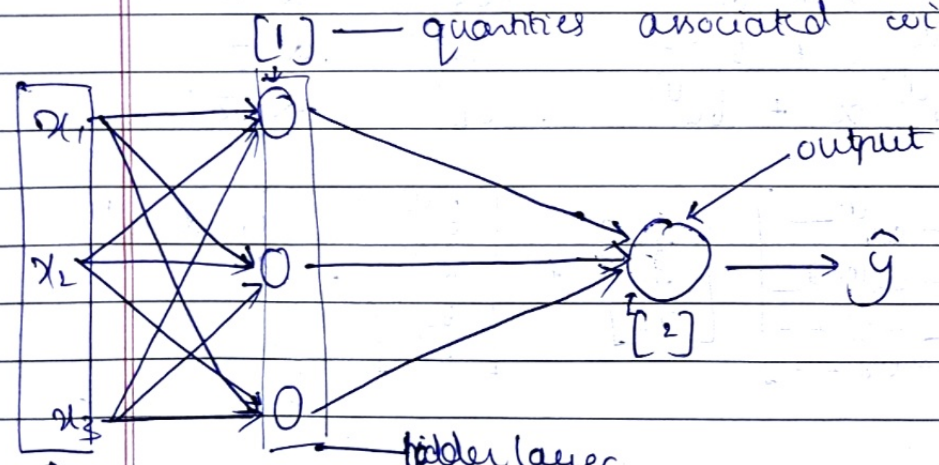
Vid 1: Overview + Vid 2

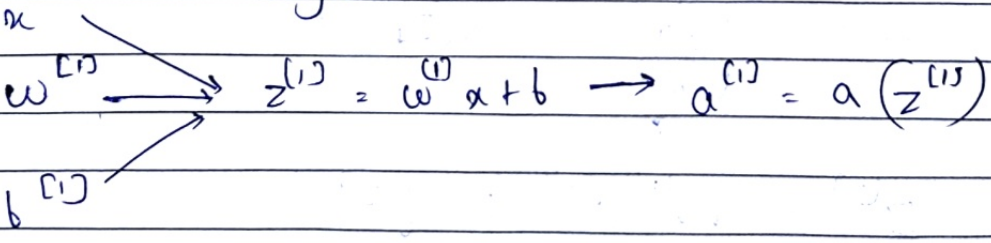




neural networks: stacking of sigmoid units

$[1]$ — quantities associated with 1st layer





Date

hidden layer - you can't see in training set

activation - A → values passed on to the subsequent layers

Computing Output of Neural Network

(i) ← layer

a_i ← node in layer

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

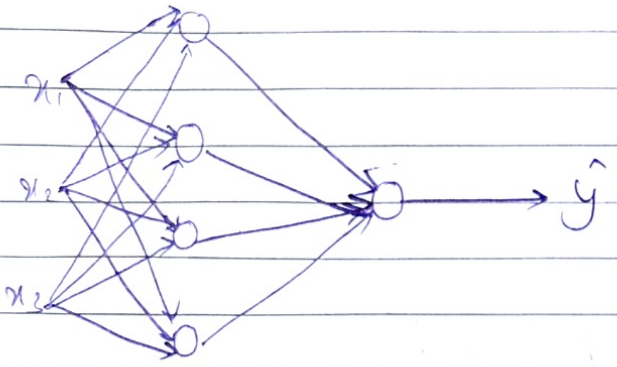
$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ \vdots \\ \vdots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

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = a(z^{[1]})$$

$$W^T = W^{[1]}$$

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

$$a^{[2]} = a(z^{[2]})$$



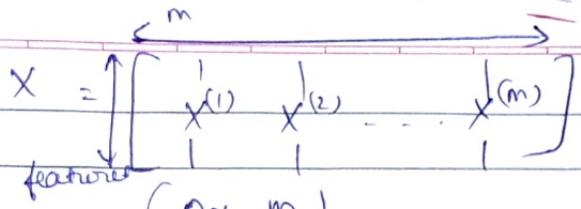
$$x \longrightarrow a^{[2]} = \hat{y}$$
$$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$
$$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$$
$$\vdots$$
$$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

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

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma z^{[2]}$$

for $i = 1$ to m

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

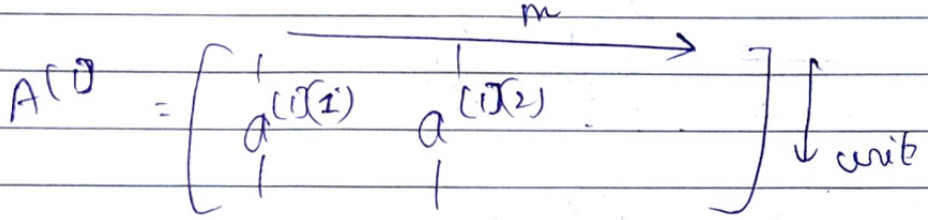


$$Z^{[1]} = W^{[1]} X + b^{[1]} \quad \rightarrow \quad \begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \dots \end{bmatrix}$$

$$A^{[1]} = g(Z^{[1]}) \quad \rightarrow \quad \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \dots \end{bmatrix}$$

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

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



$$W^{[1]} \begin{bmatrix} | & | & | & & | \\ x^1 & x^2 & x^3 & \dots & x^m \\ | & | & | & & | \end{bmatrix} =$$

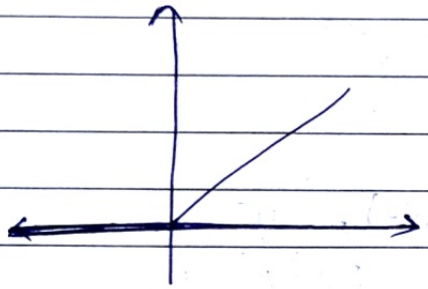
Activation Functions

* So far we have worked with sigmoid functions but it is not preferred.
* Its range is $[0, 1]$
* Another activation function is tanh function whose range is $[-1, 1]$
* Tanh function is better than sigmoid because the avg of its o/p is closer to 0. so it helps centering data better for next layer

* It cannot be used for binary classification where output is between 0 & 1 and tanh gives output between (-1, 1)

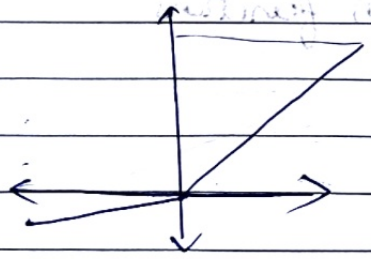
* The disadvantage of sigmoid or tanh function is that if input is too large / too small, the slope will near 0 which will cause gradient descent problem.

* ReLU — most preferred



$\max(0, z)$

* Leaky ReLU



- instead of returning 0 for -ve values it returns a small negative number

In hidden layers using linear activation function is rare.

Linear activation function produces linear activation. It is used only in output linear.

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

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

$$= w^{[2]}(w^{[1]} x + b^{[1]}) + b^{[2]}$$
$$= w^{[2]} w^{[1]} x + w^{[2]} b^{[1]} + b^{[2]}$$

Derivatives of activation functions

Sigmoid:

$$g'(z) = g(z)(1 - g(z))$$
$$= a(1-a)$$

tanh:

$$g'(z) = 1 - g(z)^2$$
$$= 1 - a^2$$

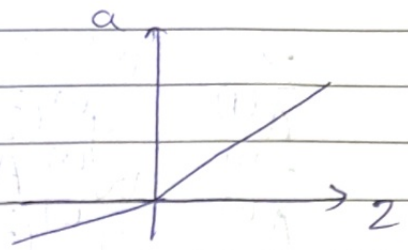
ReLU

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

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

at 0

Leaky ReLU



Leaky ReLU

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

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

Gradient Descent for Neural Networks.

Parameters: $w^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $w^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$ $(n^{[1]}, 1)$

$n_x = n^{[0]}$ - input feature

$n^{[1]}$ - hidden units

$n^{[2]}$ - output units

Cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

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

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$ $\quad db^{[1]} = \frac{dJ}{db^{[1]}}$

$w^{[1]} = w^{[1]} - \alpha dw^{[1]}$

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

Page No.
Date

For forward propagation

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

$g^{[2]}$ ← because binary classifier

Back-Propagation Step

$$dz^{[2]} = A^{[2]} - Y$$
$$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims = True)$$
$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'} z^{[1]}}_{(n^{[1]}, m)}$$

element wise product

Page No.
Date

Random Initialisation

* In logistic regression, it wasn't important to initialise weights randomly.

* However, in NN - we initialise randomly.

* Initialising all weights to 0 → ✗ — Symmetric hidden units
  Initialising bias (b) to 0 → ✓

This is because:

- all hidden units will be identical & compute same function
- same update on iteration with gradient descent.

* ∴ W is initialised randomly,

example:

$w^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

⋮

$w^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$