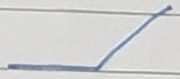
Andrew Ng

1) Neural Network

ReLU → 
Rectified linear unit

Neuron → basically takes input & gives o/p

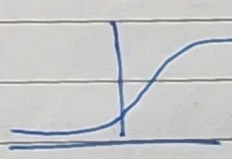
i/p → ○ → o/p
x "neuron" y

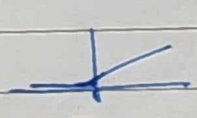
Supervised → Best

2) Supervised Learning

sequence → RNN
Structured data → table
easy for comp

image → CNN
Unstructed data → Audio, img, text
easy for humans

3) Sigmoid 
→ Problem
Here learning is slow

ReLU 

WEEK 2

4) Binary Classification

(x, y) $\quad x \in R^{n_x}$ $\quad y \in \{0, 1\}$

n → no. of training ex.

$m = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

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

$X \in R^{n_x \times m}$ $\quad$ X. Shape $= (n_x, m)$

$Y = [y^{(1)} \; y^{(2)} \ldots y^{(m)}]$

$Y \in R^{1 \times m}$

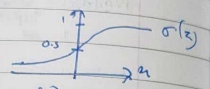
Y shape $= (1, m)$

5) Logistic Regression

$\hat{y} = P(y=1 \mid x)$ (Probability)

Parameters : $\omega \in R^{n_x}$, $b \in R$

Output : $y = \sigma(\omega^T x + b)$



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

$b \rightarrow$ intercept

$x_0 = 1$, $x \in R^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \} \omega \end{matrix}$

6) Cost Function

want, $\hat{y}^{(i)} \approx y^{(i)}$

loss (error func) :

$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

isme koi problem nai.

if $y = 1$, $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\hat{y}$ large

if $y = 0$, $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← $\hat{y}$ want min

$\therefore P(y \mid x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$

$\therefore \log P(y \mid x) = -\mathcal{L}(\hat{y}, y)$

8) Cost func^n: $J(\omega, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

loss func^n only for i^{th} step

Cost function for m steps

We need to minimize Cost Func^n

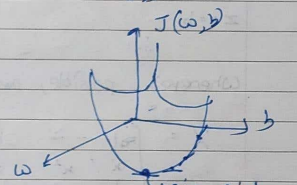
7) Gradient descent.

want to find ω, b that minimize $J(\omega, b)$

In here we repeat {

$\omega := \omega - \alpha \frac{dJ(\omega)}{d\omega}$

this means we update ω



convex function.

$\alpha \rightarrow$ learning rate

further we use $d\omega = \frac{dJ(\omega)}{d\omega}$

$\therefore \omega := \omega - \alpha d\omega$

$\frac{dJ(\omega)}{d\omega} \rightarrow$ represents slope of parameters

$\omega := \alpha - \frac{\partial J(\omega, b)}{\partial \omega}$ $\quad b := \alpha - \frac{\partial J(\omega, b)}{\partial b}$

ACTUAL

$z = \omega_1 x_1 + \omega_2 x_2 + b = \omega^T x + b$ (z is $\hat{y}$)

$a = \sigma(z)$

$\mathcal{L}(a, y)$

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

Vectorisation helps reduction in for loops.

$z = w^T x + b$ $\quad w = \begin{bmatrix} \vdots \end{bmatrix}$ $\quad x = \begin{bmatrix} \vdots \end{bmatrix}$

Non vectorized	vectorized
$z = 0$ for i in range(n-x) $z += w[i] * x[i]$ $z += b$	$z = \underbrace{np.dot(w, x)}_{w^T x} + b$

Wherever possible, avoid explicit for loops

$$X = \begin{bmatrix} | & | & | & | & | \\ x^1 & x^2 & x^3 & \cdots & x^m \\ | & | & | & | & | \end{bmatrix} \qquad R^{n \times m}$$

$$Z = [z^{(1)}\ z^{(2)} \dots z^{(m)}] = w^T X + [b\ b\ b\ b \cdots]$$

$$Z = np.dot(w.T, X) + b$$ [Broadcasting in python]

$$A = [a^{(1)}\ a^{(2)} \dots a^{(m)}] = \sigma(Z)$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dZ = [dz^{(1)}\ dz^{(2)} \dots] \qquad dA = [a^{(1)} \dots a^{(m)}] \qquad Y = [y^{(1)} \dots y^{(m)}]$$

$$dZ = A - Y$$

$$dw = \frac{1}{m} X dZ, \quad db = \frac{1}{m} np.sum(dZ)$$

$$w := w - \alpha dw \qquad b := b - \alpha db$$

8) Broadcasting

In vector is nonvector calc

$cal = A.sum(axis = 0)$ $\qquad$ percentage = 100* A/(cal reshape(1, 4))

0 ↓ → 1

0 = sum vertical

1 = " horizontal

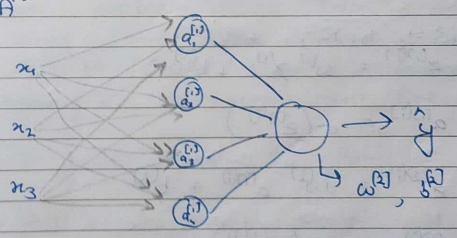
→ Dont use Rank 1 array

WEEK 3

9) NN Representation

$a^{[l]}_i$ ← layer, → node in layer

$X = A^{[0]}$



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

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

⋮

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

$4 \times 3 \qquad 3 \times 1 \qquad 4 \times 1$

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = \sigma(z^{[1]})$

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

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

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

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

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

$\vdots$

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

$a^{[2](i)}$ — i: example i; $[2]$: layer 2

for $i = 1$ to m,

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

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = w^{[2]} a^{[1](i)} + b^{[2]}$

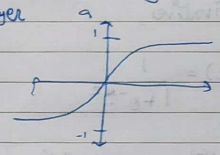
$a^{[2](i)} = \sigma(z^{[2](i)})$

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

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

horizontal $= m$, ~~vert~~ vertically depends on no. of hidden layers.

Instead of sigmoid $\sigma(z)$, using $g(z)$ i.e. $\tanh(z)$ works better, for hidden layer



$a = \tanh(z)$

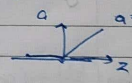
$\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

Now instead of avg at 0.5, we get average closer to 0.

For output layer it is better to use, sigmoid as $0 \leq \hat{y} \leq 1$ and of $+$ to 1.

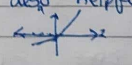
However problem with both g & σ is at very large and very small values of z the gradient descent / slope becomes closer to 0, ~~hence~~ so consumes more time.

So one better option is ReLU



$a = \max(0, z)$

~~deaky~~ ReLU is also helpful, ~~the~~ not used much, check if necessary



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

⇒ Why do NN need non-linear activation func^n.

$g(z) = z$ → linear.

If we use linear activation func^n the NN behaves as ~~good as~~ if there is no hidden layer, i.e. hidden layers are useless

If $y \in \mathbb{R}$, you may use linear activation. use it only in output layer if necessary.

→ Derivative,

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

$$g'(z) = \frac{d\,g(z)}{dz} = g(z)\left(1 - g(z)\right) = a(1-a)$$

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

$$g'(z) = \frac{d\,g(z)}{d(z)} = \text{slope of } g(z) \text{ at } z$$

$$= 1 - (\tanh(z))^2$$

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

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

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

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

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

→ Back Propogation

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

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

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

→ To prevent Rank 1 array

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{n^{[1]} \times m} \underset{\text{means element wise product.}}{*} \underbrace{g^{[1]\prime}\left(Z^{[1]}\right)}_{n^{[1]} \times m}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T \quad , \quad db^{[1]} = \frac{1}{m} \text{np.sum}\left(dZ^{[1]}, \text{axis}=1, \text{keep dims} = \text{True}\right)$$

→ We cannot initialize w as 0 for DL.
as then,
$a_1^{[1]} = a_2^{[1]}$ also, $dz_1^{[1]} = dz_2^{[1]}$

Hence, ini. it randomly.

$w^{[1]} = np.random.rand((2,2)) * \underbrace{0.01}$
$b^{[1]} = np.zero((2,1))$

↳ small because if $w \uparrow$ then $g(z)$ will have slope zero.

WEEK-4

Dimensional Analysis.

$n^{[i]}$ = no of ~~nodes~~ hidden units in layer i

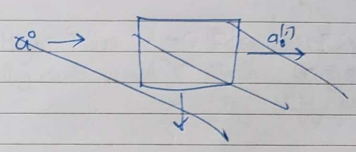
~~$z^{[1]} = w^{[1]} x + b^{[1]}$~~
~~$(n^{[1]} \times 1)\ (n^{[1]} \times n^{[0]})(n^{[0]} \times 1)\ (n^{[1]} \times 1)$~~

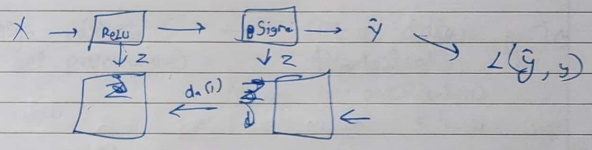
$$z^{[1]} = w^{[1]} x + b^{[1]}$$
$$(n^{[1]} \times 1)\quad (n^{[1]} \times n^{[0]})(n^{[0]} \times 1)\quad (n^{[1]} \times 1)$$

dw & db have same dimensions as w & b resp.

$$Z^{[1]} = w^{[1]} X + b^{[1]}$$
$$(n^{[1]} \times m)\quad (n^{[1]} \times n^{[0]})(n^{[0]} \times m)\quad (n^{[1]} \times 1)$$

$dZ^{[2]}, dA^{[2]}, Z^{[2]}, A^{[2]}$: $(n^{[2]} \times m)$





Parameters : w & b
Hyperparameters : α (learning rate)
iterations
hidden layers l
hidden units $n^{[1]}, n^{[2]}, \ldots$
choice of activation func^n

Later we'll see : momentum, minibatch size ...