Camlin Page
Date / /

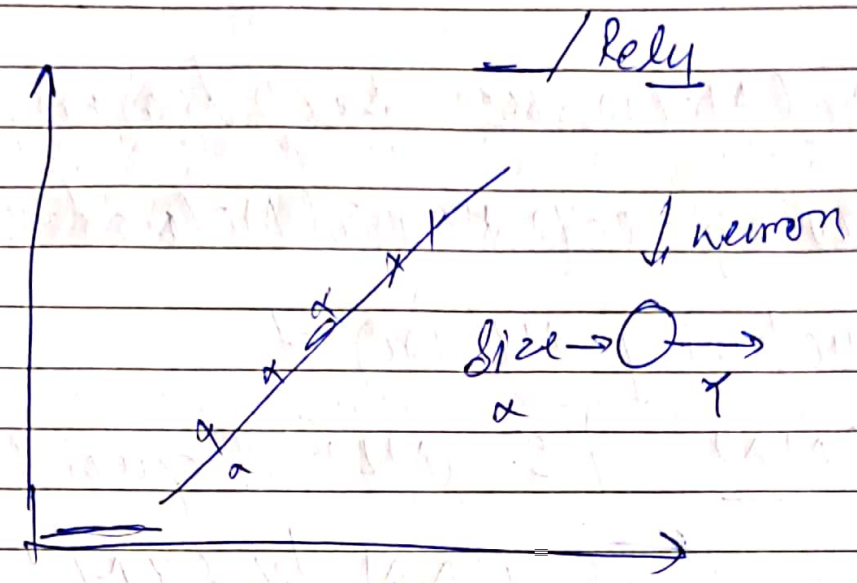
Answers

Deep learning AI

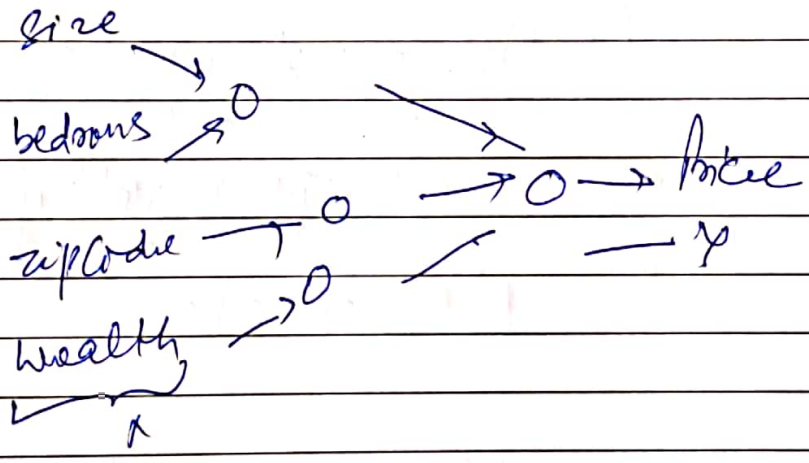
What I will Learn

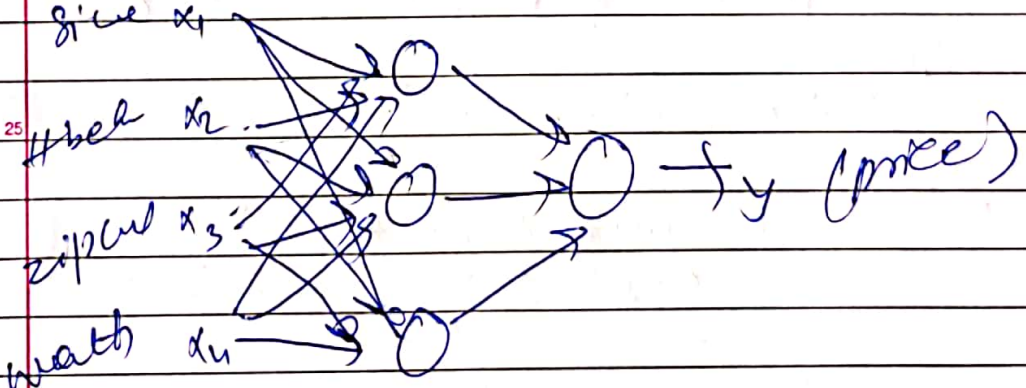
1. Neural Networks & Deep Learning
2. Improving Deep Neural Networks:-
3. Structuring ML Project
4. CNN
5. NLP: Seqnen Models.
   RNN / LSTM

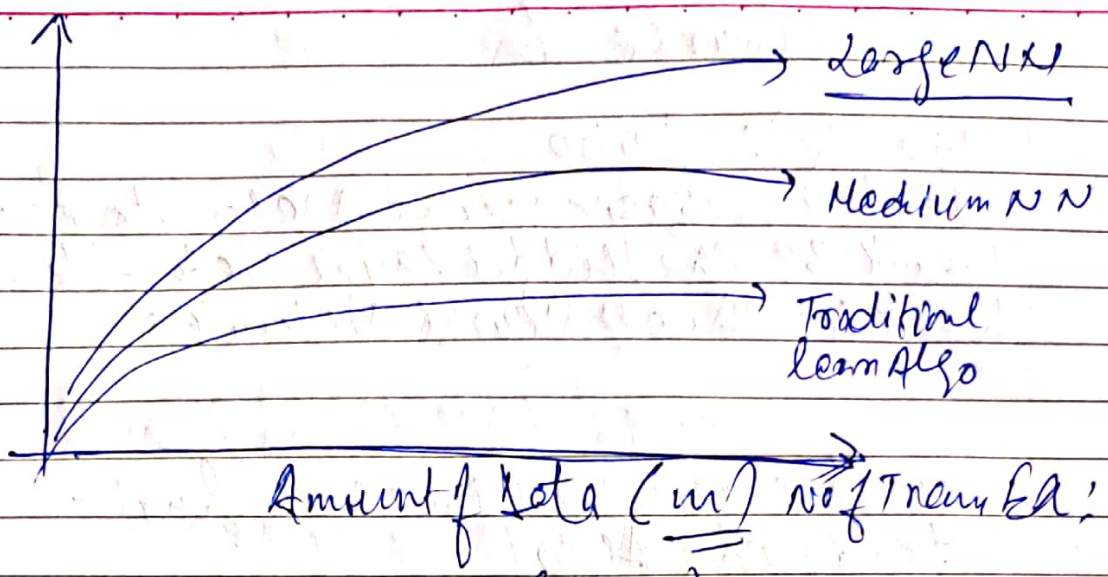
Neural Network



Housing price prediction







Small Training Set (SVM) can do better where as Neural Network Improve by more & more data.

Idea

Experiment ⟷ Code

Camlin Page
Date / /

Course: 2

Week 1: Intro
Week 2: Basic Neural Network Prog.
Week 3: One Hidden Layer Network.
Week 4: Deep Neural Network.

Increase Train size Doesn't hurt traditional Algo.

Increase size of Neural Network does not hurt but it can help.

Week: 2

[Logistic Regression as a Neural Network]

Notation - Single Training example is denoted (x, y) $x \in \mathbb{R}^n$

where x is n dimensional feature vector. $y \in (0, 1)$

m (no of train example)

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

$M = M_{train}$ / No of train example
M_{test} / No. of test example

$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ ← taking each train example stacking up into col.

notice that in OpenCV

$\leftarrow m \rightarrow$

we do it for ROW wise stacking.

→ X.shape (n_x, m)

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

stacking labels into cols/-

Y.shape $(1, m)$

Logistic Regression

Given X, what is $\hat{y}$ (i.e) label

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

Param: $w \in R^n$, $b \in R$

Output: $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$

σ = Sigmoid function.

Logistic Regression Cost function

$\hat{y} = \sigma(z)$ where $z = w^T x + b$

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})$

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

$\mathcal{L}(\hat{y}, y)$ = ($\hat{y}$ = predicted lbl, y = ground lbl)

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

if $y = 1$

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

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

Cost Function

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right)$$

Camlin Page
Date / /

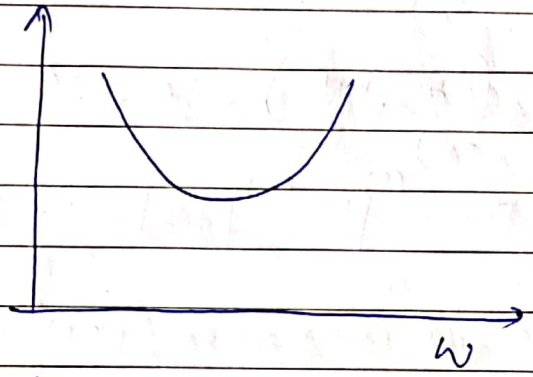
Gradient Descent

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

want to find w, b which min $J(w,b)$

Usually for "Logistic Regression" we init by "zeros".



repeat {

$$w = w - \alpha \frac{\partial J(w)}{\partial w}$$

}

α = learning Rate.

we'll use convention $dw = \frac{\partial J(w)}{\partial w}$

so $w = w - \alpha\, dw$. / so

$$w = w - \alpha \left[\frac{\partial J}{\partial w}\right] \rightarrow dw$$

$$b = b - \alpha \left[\frac{\partial J}{\partial b}\right] \rightarrow db$$

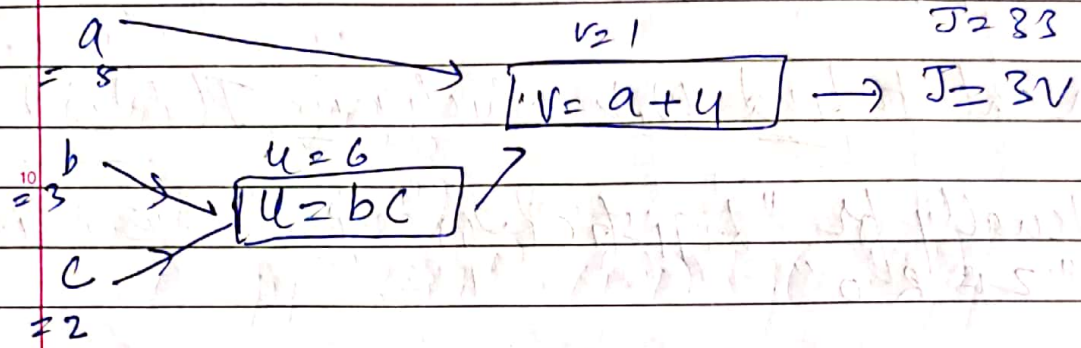
Computation Graph

$J(a, b, c) = 3(a + bc)$

$u = bc$

$v = a + u$

$J = 3v$



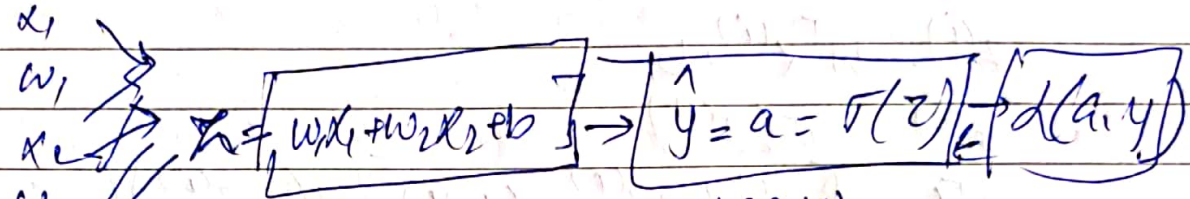
$\frac{\partial J}{\partial v} =$

Logistic Regression Gradient Descent

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

$L(a, y) = -(y \log(a) + (1-y) \log(1-a))$

Let say we have feature length '2' (x_1 & x_2)



$dz = \frac{\partial L(a,y)}{\partial z}$

$= (a - y)$

"da" $= \frac{\partial L(a,y)}{\partial a}$

$= -\left(\frac{y}{a}\right) + \left(\frac{1-y}{1-a}\right)$

$\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} = a(1-a)$

$\frac{\partial L(a,y)}{\partial w_1}$ = "dw_1" $= x_1 \cdot dz$ | $dw_2 = x_2 \cdot dz$ | $db = dz$

$w_1 = w_1 - \alpha\, dw_1$

$w_2 = w_2 - \alpha\, dw_2$

$b = b - \alpha\, db$

Logistic regression on "m" examples

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y)$$

Init $J=0,\ dw_1=0 \mid dw_2=0 \mid db=0$

for $i=1 \rightarrow m$

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J += -\left[y^{(i)} \log a^{(i)} + (1-y)\log(1-a^{(i)})\right]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 += x_1^{(i)} dz^{(i)}$$
$$dw_2 += x_2^{(i)} dz^{(i)}$$
$$db += dz^{(i)}$$

} $n=2$

$J /= m \mid dw_1 /= m \mid$

$$w_1 = w_1 - \alpha dw_1$$
$$w_2 = w_2 - \alpha dw_2$$
$$b = b - \alpha db$$

Logistic Regression Vectorization

Vectorization of code enable SIMD instruction.

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

$$[z^{(1)}\ z^{(2)}\ \cdots\ z^{(m)}] = w^T X + [b_1\ b_2\ \cdots\ b] \quad 1 \times m$$

$$w^T \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ x^{(1)} & x^{(2)} & \cdots & x^{(n)} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \quad (n_x, m)$$

w^T is a row vector.

$$[\quad] \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ x_1 & x_2 & \cdots & x_m \\ b & b & & b \end{bmatrix}$$

$$= [w^T x_1\ \ w^T x_2\ \cdots\ w^T x_m] + [b_1\ b_2\ \cdots\ b_m]$$

$$Z = [w^T x_1 + b_1\ \ w^T x_2 + b_2\ \cdots\ w^T x_m + b_m]$$

$$= [Z_1\ Z_2\ \cdots\ Z_m]$$

$Z = np.dot(w.T, X) + b$ "Broadcasting"

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

$dz^{(1)} = a^{(1)} - y^{(1)}$ | $dz^{(2)} = a^{(2)} - y^{(2)}$ ---

$dZ = [dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}]$ $1 \times m$

$A = [a^{(1)}\ a^{(2)} \cdots a^{(m)}]$ $Y = [y^{(1)}\ y^{(2)} \cdots y^{(m)}]$

$dZ = A - Y$

$dZ = [a^{(1)} - y^{(1)},\ a^{(2)} - y^{(2)} \cdots]$

$dw = 0$

$dw += x^{(1)} dz^{(1)}$ - first train example

$dw += x^{(2)} dz^{(2)}$ - second train example

!

$dw += x^{(m)} dz^{(m)}$ --- m^{th} train example

$dw = dw/m$

$db += dz^{(1)}$ - 1st train ean

$db += dz^{(2)}$ - 2nd train example

$db = db/m$

$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$

$= \frac{1}{m}$ ~~np.dot~~ $np.sum(dZ)$

$dw = \frac{1}{m} X dZ^T$

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

$(n_x, m) \qquad (m, 1) \qquad = (n_x, 1)$

$$= \frac{1}{m} \left[x_1 dz_1 + x^{(2)} dz^{(2)} \cdots x^{(m)} dz^{(m)} \right]$$

Non Vectorize

$J=0, dw_1=0, dw_2=0, db=0$

for $i=1 \rightarrow m$

$z^{(i)} = w^T x^{(i)} + b$

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

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

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

$dw_1 += x_1^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$

$db += dz^{(i)}$

$J = J/m \mid dw_1 = dw_1/m$

$dw_2 = dw_2/m$

$db = db/m$

Vectorize

$Z = W^T X + b$

$= np.dot(W.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m} np.sum(dz)$

$w = w - \alpha\, dw$

$b = b - \alpha\, db$

↓ Single

iteration of gradient descent.

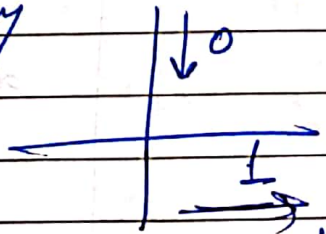
<Broadcasting Python>

	Apple	Beef	Eggs	Potato
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

Y. Calu from Carb / protein / Fat for each food.

```
A = [ [56,  0.0,  4.4,  68.0,]
    - [1.2,  104.0,  52.0,  8.0 ]
      [1.8   135.0,  99   0.9 ] ]

cal = A.sum(axis=0)

[ 59   239   155.4   76.9 ]

per = A / cal.reshape(1,4)
pes = 100 x pes:

[ [ 9.9.9   0   2.83   88.4]
  [2.03   43.51   33.46   10.40]
  [3.05   57.48   63.70   1.17 ] ],
```

A.sum(axis=0) means python should sum vertically



cal.reshape(1,4) no need it's already is

Camlin Page
Date / /

Broadcasting Ex

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 = \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

$$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix} + \begin{bmatrix}100&200&300\end{bmatrix} \rightarrow \begin{bmatrix}100&200&300\\100&200&300\end{bmatrix}$$

=

(m,n) $\begin{matrix}+\\-\\*\\/\end{matrix}$ $(1,n)$ $\Rightarrow$ copy m times (m,n)

and then multiplies the action $(+,-,*,/)$

(m,n) $\begin{matrix}+\\-\\*\\/\end{matrix}$ $(m,1)$ $\Rightarrow$ copy n times (m,n) & apply operation.

Python Rule of Thumbs

`a = np.random.rand(5)` (X) [rank 1 array] not use

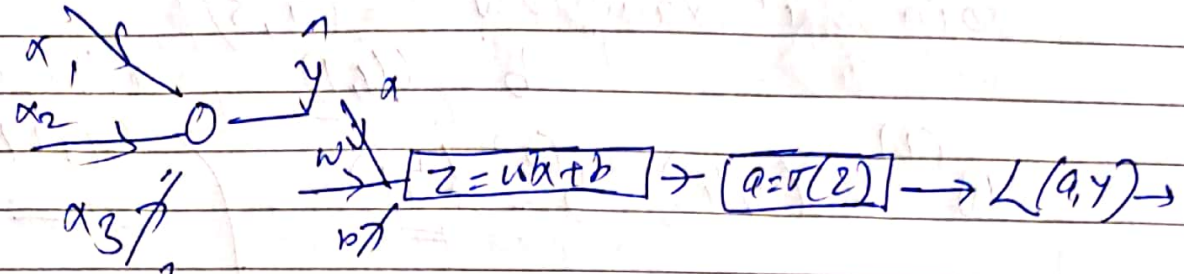
`a = np.random.rand(5,1)` ✓

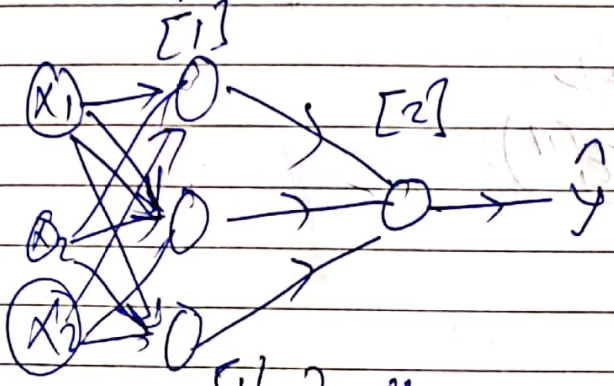
Use shape assert:

`assert (a.shape == (5,1))`

Shallow Neural Network

[Week-03]

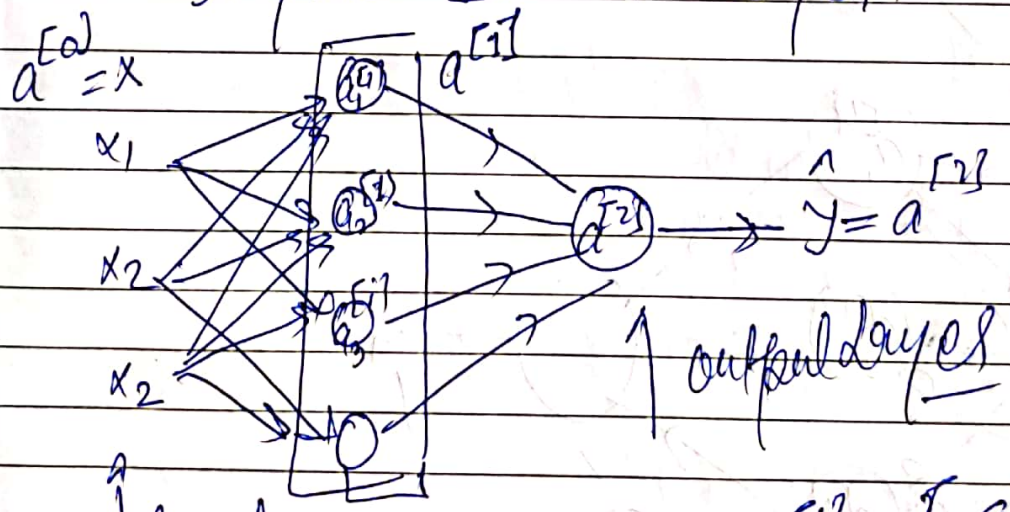




$w^{[1]}$, $b^{[1]}$ } allow us to find $z^{[1]} = w^{[1]} x + b^{[1]}$

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

Superscript [i] weight & bias correspond to layer no: i^{th} layer:



$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

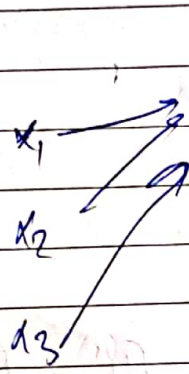
$a^{[i]}$: i^{th} layer activation

"2 layer NN" / we don't count input layer"

Hidden layer parameters $W^{[1]}, b^{[1]}$

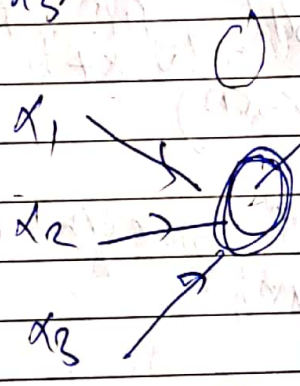
So in this example $W^{[1]} = (4,3)$ $b = (4,1)$ — dimensions.

$W^{[2]} = (1,4)$ $b = (1,1)$



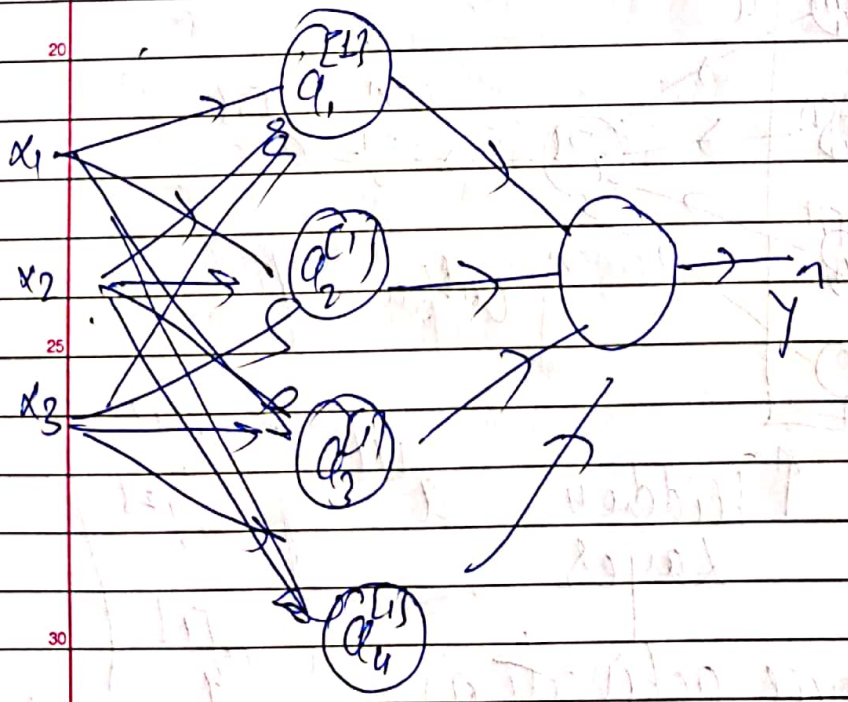
$z_1^{[1]} = W^{[1]} x + b^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$



$z_2^{[1]} = W_2^{[1]T} x + b_2^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$



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

each neuron ACT as one logistic unit $(w^T x + b)$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times 1} \quad \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \quad w^T = (w_1\ w_2\ w_3) \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

So for each neuron will put a row hence W will be (4,3) matrix.

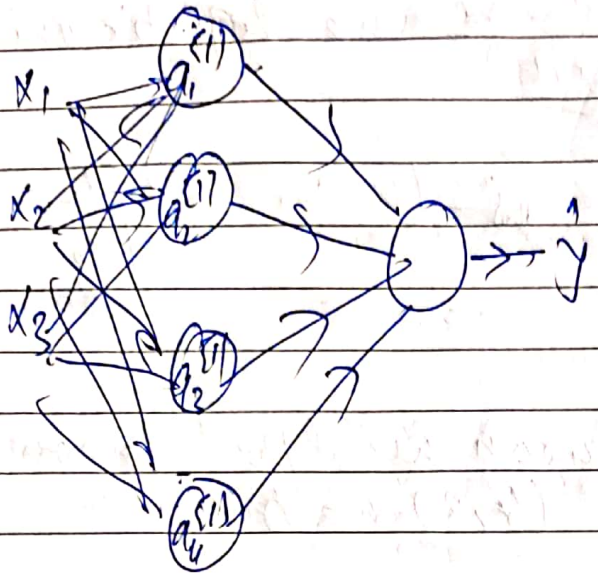
$$z^{[1]} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix}_{(4\times 3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{(3\times 1)} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x_1 + b_1^{[1]} \\ - \\ - \\ - \end{bmatrix}$$

$= (4\times 1)$ output

$$= \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} \xrightarrow{g} a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z_1)$$

Camlin Page
Date / /

Given Input X



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

$$= (4,3)(3,1) + (4,1)$$

$$= (4,1) + (4,1)$$

$$Z^{[1]} = (4,1)$$

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

$$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} = (1\times4)(4\times1) + (1,1) = (1,1)$$

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

$$= (1\times1)$$

Across Multiple examples Vectorized

for $i = 1 \to m$

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

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

(n_x, m)

m = No of Train exm

n_x = feature length.

Vectorizing It

$W = (4,3)$

$W^T = (3,4)$

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

$$= (4,3)(3,m) + (4,1)$$

$$= (4,m) + (4,1)$$

$$Z^{[1]} = (4,m)$$

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

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

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

← Across Train Ex. →

↕ different Hidden Unit in a layer

$W^{[1]} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$ $x^{(1)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

$W^{[1]} x^{(1)} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} \\ \\ \end{bmatrix}$

$= (4 \times 3)(3 \times 1) \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

$= 4 \times 1$

$W^{[1]} x^{(2)} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$ $\quad W^{[1]} x^{(3)} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$

$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix}$

X

$W^{[1]} X$

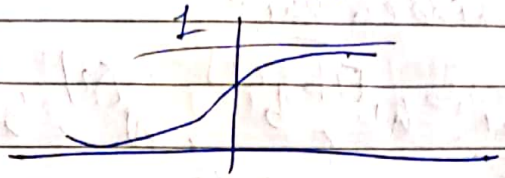
$W^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$

$= \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix}$

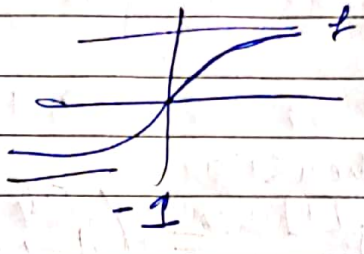
$= Z^{[1]}$

Activation functions

Sigmoid(x) =

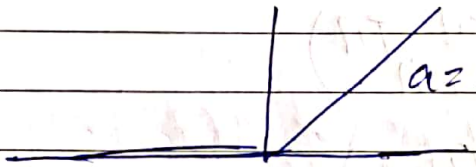


tanh(x) = $\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



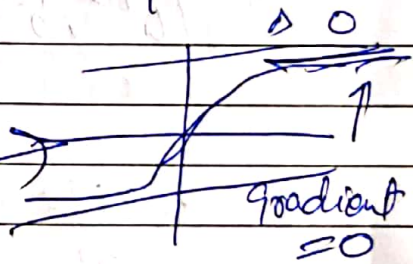
tanh is almost superior than sigmoid fun-

Relu



Derivatives of Activation

$\frac{d}{dz} g(z) = g(z)(1 - g(z))$ g(z) = sigmoid



g = tanh(z)

$\frac{d}{dx} g(z) = 1 - (\tanh(z))^2$

g(z) = Relu

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

Impractical

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

Random init weights should be very small tanh or sigmoid weight large gradient ≈ 0

Gradient Descent NN

1 hidden unit

Param: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$ | $h_x = h^{[0]}$, $h^{[1]}$ hidden unit

$h^{[2]} = 1$ output

dimension

$w^{[1]} = (n^{[1]}, n^{[0]})$

$b^{[1]} = (n^{[1]}, 1)$

$w^{[2]} = (n^{[2]}, n^{[1]})$

$b^{[2]} = (n^{[2]}, 1)$

Cost function :-

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

($a^{[2]}$)

while (1) {

$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$

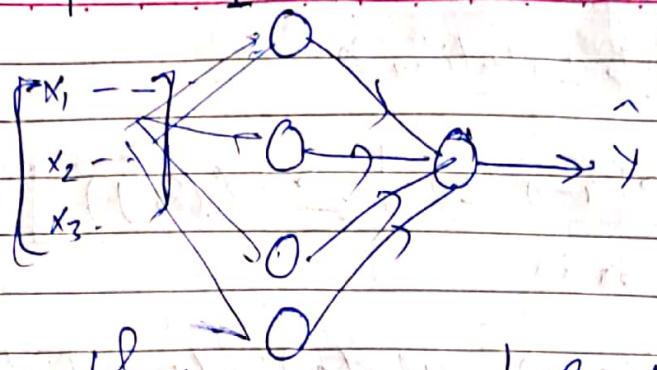
$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$

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

$w^{[2]} = w^{[2]} - \alpha\, dw^{[2]}$

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

}



Ground Truth y

Forward Prop

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

$= (4,3)(3,m) + (4,1)$

$Z^{[1]} = (4,m) + (4,1)$

$Z = (4,m)$

$\sigma(Z^{[1]}) = A^{[1]} = (4,m)$

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

$= (1,4)(4,m) + (1,1)$

$= (1,m) + (1,1)$

$= (1,m)$

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

$[1, n \ldots m]$

Back propagation :-

$dZ^{[2]} = A^{[2]} - Y \quad Y = [y^{(1)} \ldots y^{(m)}]$

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

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

↓ Here it will give $(4,1)$ matrix

So i.e $(1,1)$ $n^{[2]} = 1$

$(n^{[1]}, m)$

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

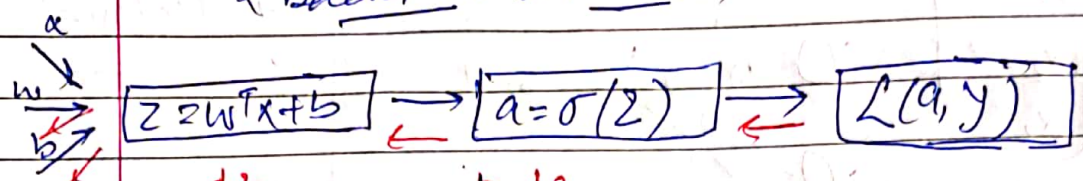
↑ element wise product

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

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

Computing Gradients

(Backward Intuition)



dw
db

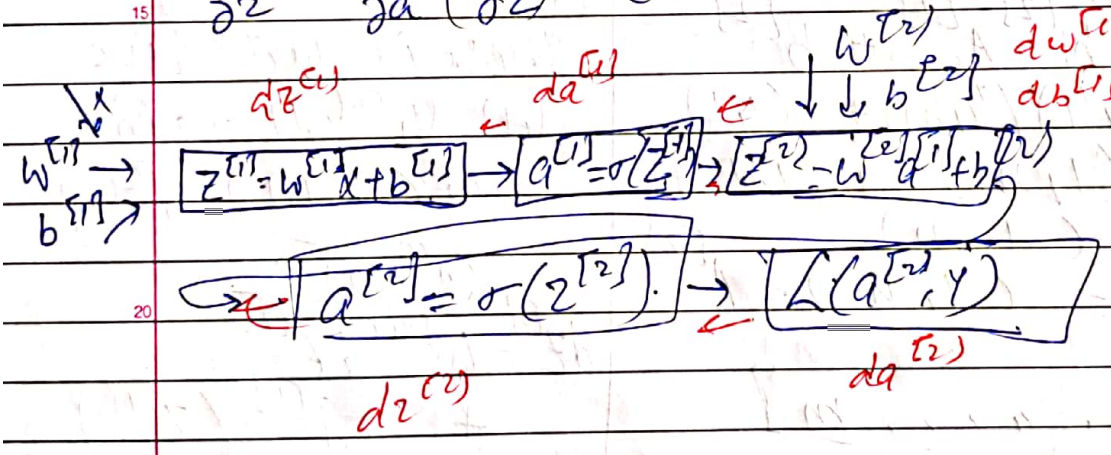
$$\frac{d}{da} da = \frac{d}{da} \mathcal{L}(a, y) = -y \log(a) - (1-y) \log(1-a)$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = a - y$$

$$dz = da \cdot g'(z) =$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \left(\frac{\partial a}{\partial z}\right) \rightarrow \left(\frac{d}{dz} g(z)\right) = g'(z)$$



$$dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$w^{[2]} = (n^{[2]}, n^{[1]})$$

$$z^{[2]}, d^{[2]} = (n^{[2]}, 1).$$

$$z^{[1]} = (n^{[1]}, 1) = dz^{[1]}$$

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

$= (n^{[1]}, n^{[2]})(n^{[2]}, 1) * (n^{[1]}, 1)$

$= (n^{[1]}, 1) * (n^{[1]}, 1)$

Summary of Gradient descent

$dz^{[2]} = a^{[2]} - y$

$dw^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

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

$dw^{[1]} = dz^{[1]} x^T$ $\quad db^{[1]} = dz^{[1]}$

< Vectorized >

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

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

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

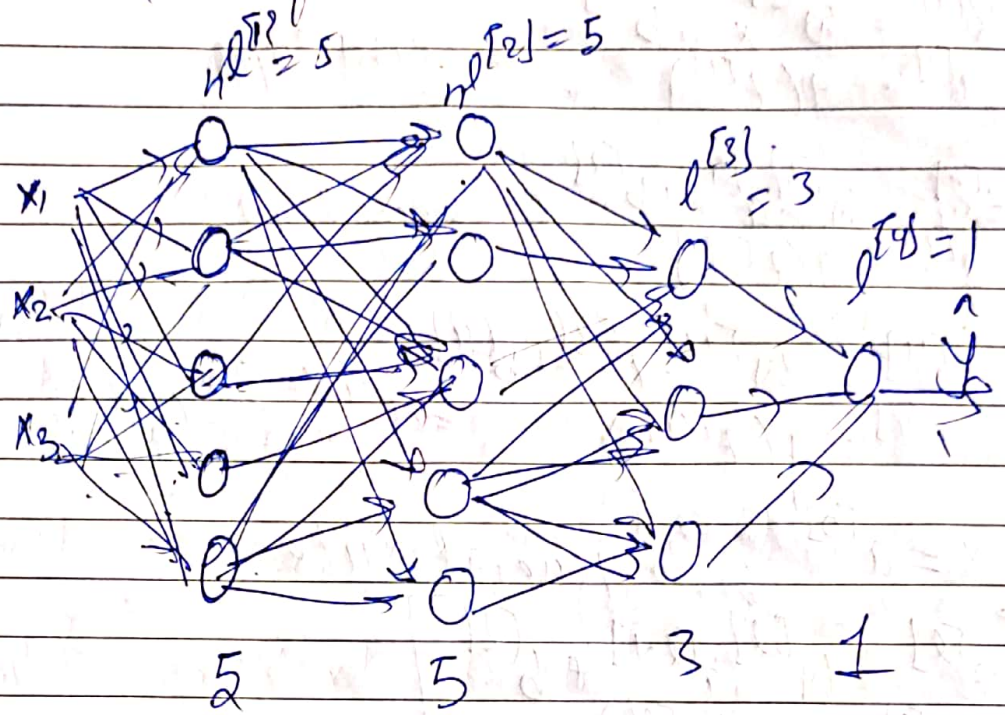
$axis=1, keepdims=True)$

$dZ^{[1]} = W^{[2]T} dZ^{[2]}$

$* g^{[1]'}(Z^{[1]})$

Week-04

< Deep Neural Network >



4 layer NN | L = 4 (No of layers)

$n^{[l]}$ = # units in layer (l)

$n^{[1]} = 5$; $n^{[2]} = 5$; $n^{[3]} = 3$; $n^{[4]} = n^{[L]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activation in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$ | $w^{[L]}$ =

x = input feature = $a^{[0]}$

$\hat{y}$ = output layer = $a^{[L]}$

Forward Propagation in a Deep Network

$z^{[1]} = W^{[1]} x + b^{[1]}$
$a^{[1]} = g(z^{[1]})$ = first layer

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$
$a^{[2]} = g(z^{[2]})$ = second layer

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$
$a^{[4]} = g(z^{[4]})$ = 4th layer

$x = a^{[0]}$; So general Rule

$$\begin{cases} z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \\ a^{[l]} = g(z^{[l]}) \end{cases}$$

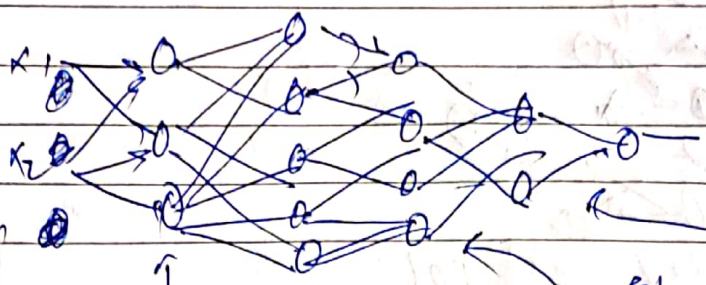
Vectorize

$Z^{[1]} = W^{[1]} X + b^{[1]}$ $\quad X = A^{[0]}$

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

$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$

2 Getting Dimension of Matrix Right



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

$(3,1) = (3,2)(2,1)$

$(n^{[1]},1) = (n^{[1]},n^{[0]})(n^{[0]},1)$

$w^{[1]} = (n^{[1]}, n^{[0]})$

$w^{[2]} = (n^{[2]}, n^{[1]}) (5,3)$

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

$(5,1) \quad (5,3)^{*}(3,1)$

$w^{[3]} = (4,5) \quad | \quad w^{[4]} = (2,4) \quad | \quad w^{[5]} = (1,2)$

$w^{[l]} = (n^{[l]}, n^{[l-1]})$

$b^{[l]} = (n^{[l]}, 1)$

$dw^{[l]} = (n^{[l]}, n^{[l-1]})$

$\Rightarrow z^{[l]}, A^{[l]}$

$(n^{[l]}, m)$

$dz^{[l]}, dA^{[l]} : (n^{[l]}, m)$

(Vectorize form)

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

$(n^{[1]}, 1) = (n^{[1]}, n^{[0]}) \cdot (n^{[0]}, 1)$.

↓ vectorize

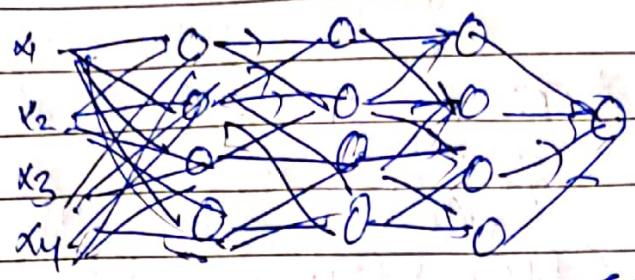
$Z^{[1]} = W^{[1]} X + b^{[1]}$

↓

$(n^{[1]}, m) = (n^{[1]}, n^{[0]})(n^{[0]}, m) + (n^{[1]}, 1)$ (broadcasting)

Neural Network

(Building Blocks of Deep Learning)



for layer l: $w^{[l]}$, $b^{[l]}$

Forward: input $a^{[l-1]}$, output $a^{[l]}$

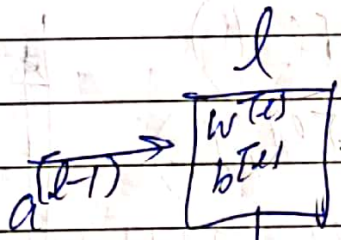
$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

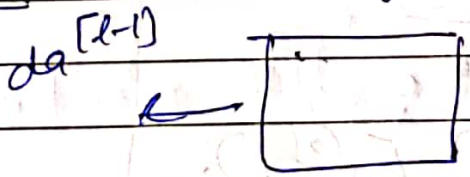
Backward: - input $da^{[l]}$ output $da^{[l-1]}$

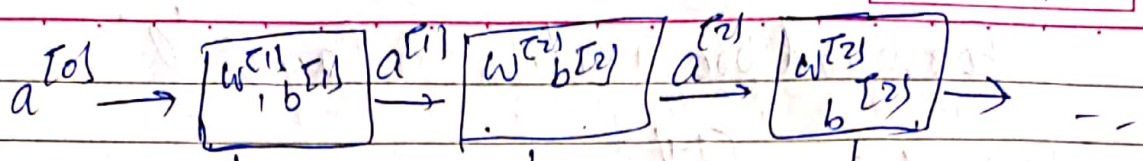
cache $z^{[l]}$ $dw^{[l]}$ $db^{[l]}$

forward:



Backward:

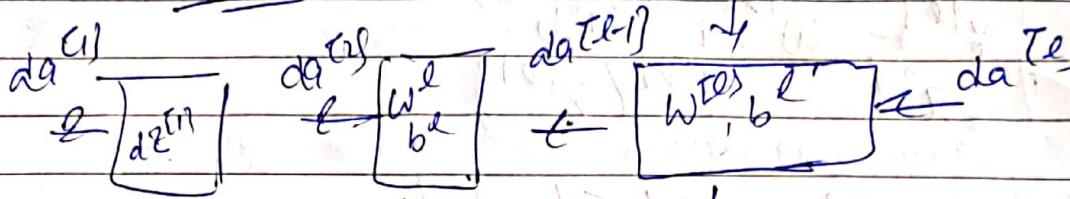




cache $z^{[1]}$
cache $[w^{[1]}, b^{[1]}, z^{[1]}]$

cache $z^{[2]}$

Cache $z^{[3]}$



dw^2, db^2

dw^3, db^3

$dw^{[l]}, db^{[l]}$

$$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha\, db^{[l]}$$

P.T.O.

Forward propagation for layer "l"

Input: $a^{[l-1]}$

Output: $a^{[l]}$, cache ($z^{[l]}$)

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

vectorize

$$Z^{[l]} = W^{[l]} A^{[l]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Backward propagation for layer "l"

Input :- $da^{[l]}$; Output :- $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} dz^{[l]}$$

vectorize

$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$$

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

$$da^{[l]} = -\frac{y}{a} + \left(\frac{1-y}{1-a}\right)$$

$$dA^{[l]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{1-y^{(1)}}{1-a^{(1)}} \cdots - \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right.$$

P.T.O

Camlin Page
Date / /

{ Forward & Backward loop }

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]}) = \hat{Y}$$

{ Backward loop }

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{T}$$
$$db^{[L]} = \frac{1}{m} np.sum(dz^{[L]}, axis=1, keepdims=True)$$
$$dZ^{[L-1]} = dW^{[L]T} dZ^{[L]} * g'^{[L]}(Z^{[L-1]})$$
$$\vdots$$
$$dZ^{[1]} = dW^{[2]T} dZ^{[2]} * g'^{[1]}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[1]T}$$
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$