DEEP LEARNING SPECIALIZATION

By: Andrew Ng

Course - 1

Neural Networks & Deep Learning

Practice Notes By:
Nishant Gupta

Scanned with CamScanner

Table of Contents

Week 1: Introduction to Deep Learning

1. What is a Neural Network [1,2]
2. Supervised Learning with NN [3]
3. Why is Deep learning taking off [4]

Week 2: Logistic Regression as a NN

1. Binary Classification & Logistic Regression [5,7]
2. Logistic Regression Cost Function [8,10)
3. Gradient Descent (10,11]
4. Derivatives [12]
5. Derivatives with Computation Graph [13,15]
6. Gradient Descent for Logistic Regression [16,17]
7. Vectorization on Logistic Reg^n & it's Gradient [18,21]
8. Broadcasting in Python [22]
9. Avoiding "Rank-1" Array [23]
10. Explanation of Logistic Reg^n Cost function [24,25]

Week 3: Shallow Neural Network

1. NN Overview (1-Layer, 2-Layer) [26]
2. NN Representation [27]
3. Computing a NN's Output [28,29]
4. Vectorizing across Multiple Examples [30,31]
5. Explanation for Vectorized Implementation [32]

Scanned with CamScanner

6. Activation Functions [33]
7. Why do you need Non Linear Activation f^{ns}. [34]
8. Derivatives of Activation Functions [35,37]
9. Gradient Descent for NN [38,39]
10. Backpropagation Intuition [40,41]
11. Summary of Gradient Descent [42]
12. Random Initialization [43,44]

Week 4: Deep Neural Network

1. Deep L-Layer NN [45]
2. Forward Propagation in a Deep Network [46]
3. Getting your Matrix Dimensions Right [47,48]
4. Why Deep Representations [49]
5. Building Blocks of Deep NN [50,51]
6. Forward & Backward Propagation [52,53]
7. Parameters Vs. Hyperparameters. [54]

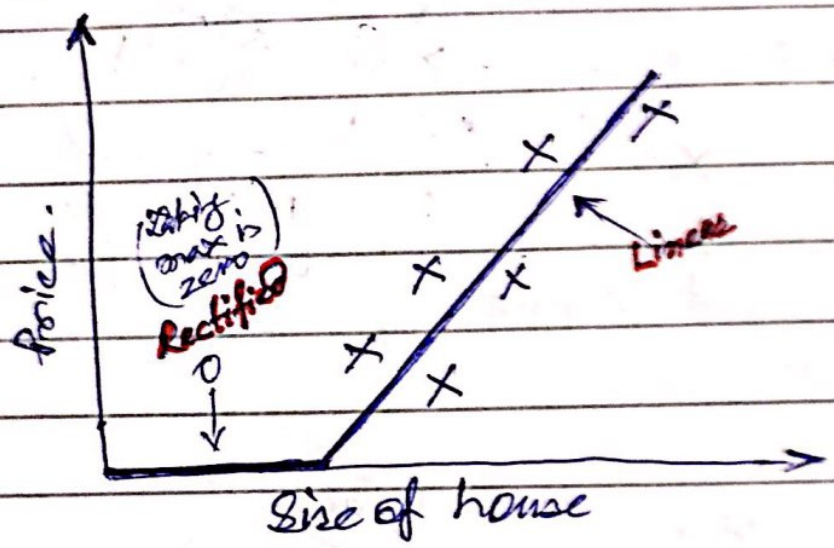
Scanned with CamScanner

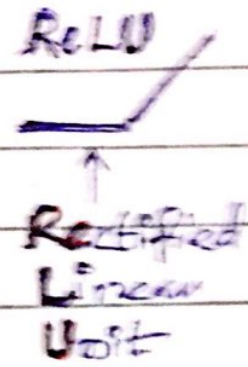
Deep Learning Specialization

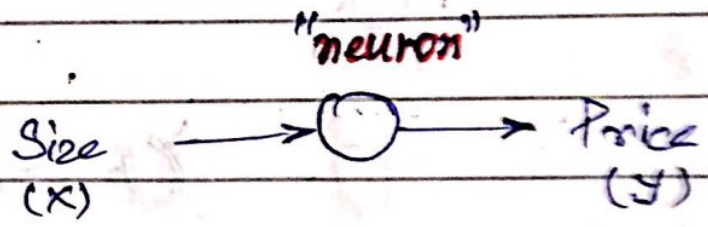
1. Neural Networks & Deep Learning

*. AI is the new electricity.
(will bring an equally big transf^n)

*. eg. Housing Price Prediction.







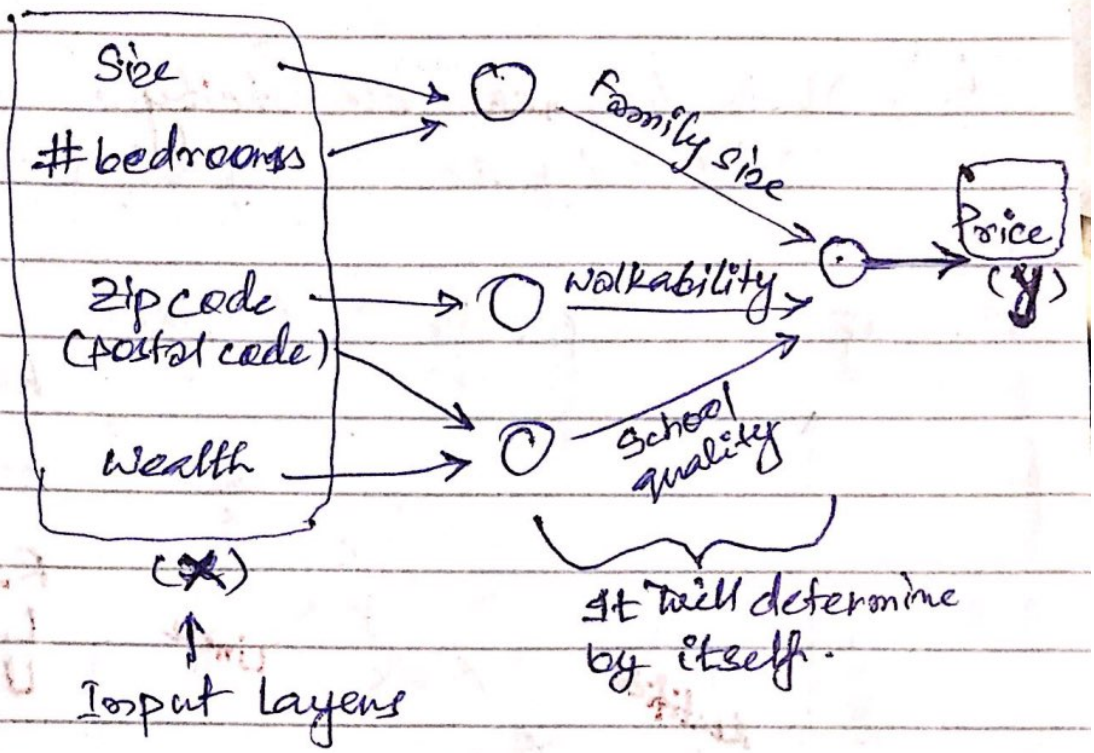
The above is a single "neuron" if we stack together multiple neurons then we will get a NEURAL NETWORK.

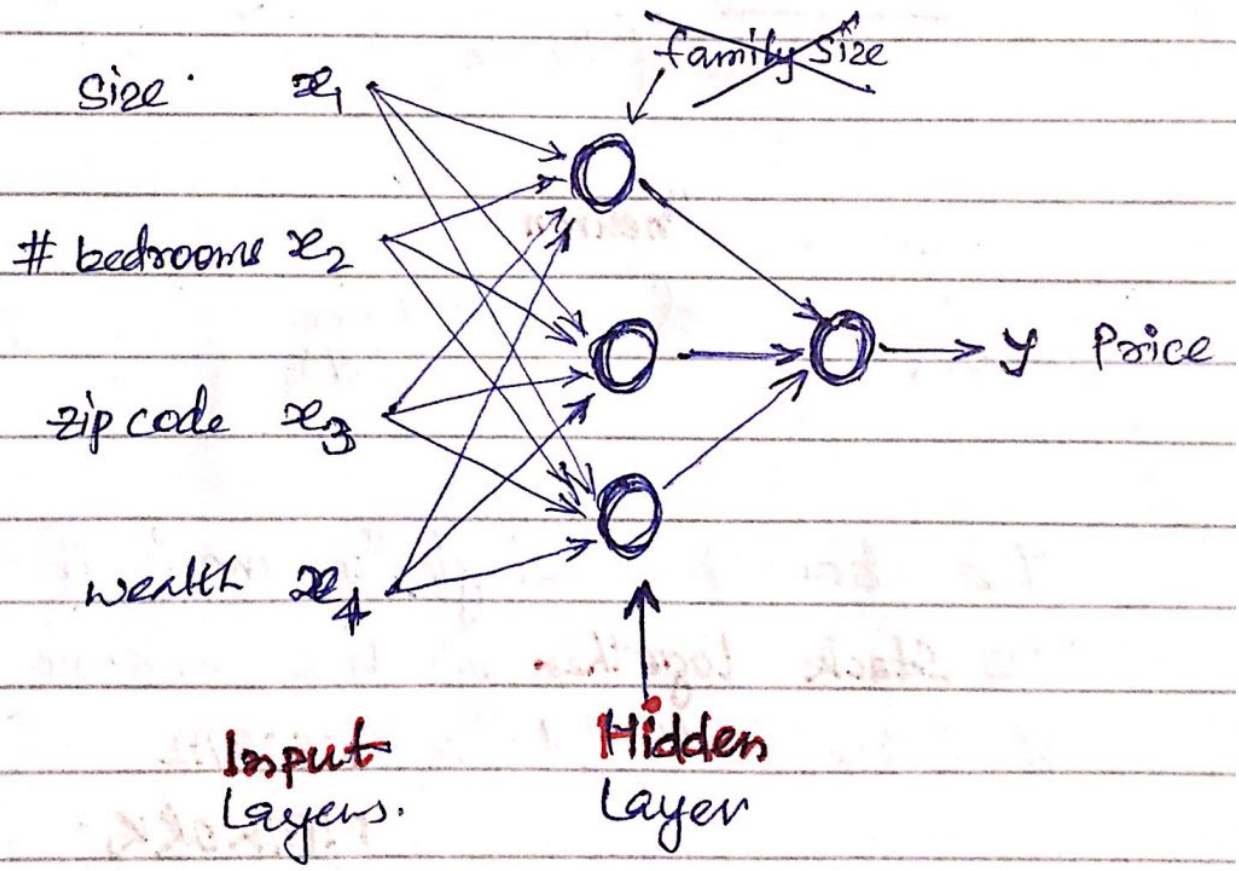
eg.

Scanned with CamScanner

Example of Neural Network (Basic)

eg

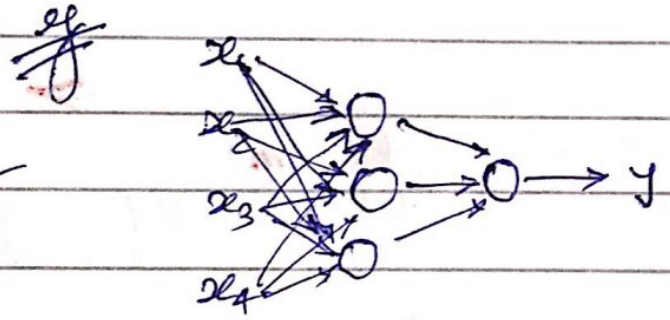




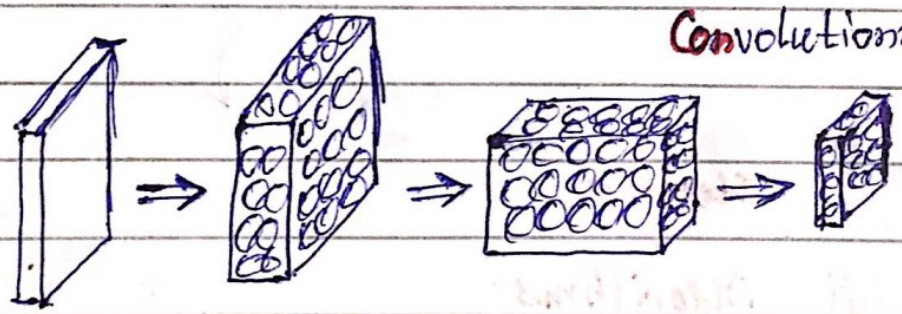
Scanned with CamScanner

A. Supervised learning

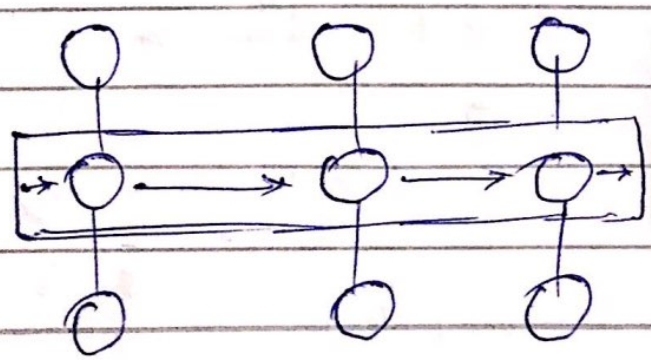
INPUT (X)	OUTPUT (Y)	APPLICATION	
Home features	Price	Real Estate	Std. NN
Ad, user info	Click on ad (0/1)?	Onl. Adv.	Std. NN
Image	Object (1,---1000)	Photo tagging	CNN
Audio	Text transcript	Speech Recog.	RNN
English	Chinese	M/c Trans^{n}	RNN
Image, Radar Info	Position of other cars	Aut. Driving	Hybrid or custom NN



Standard NN



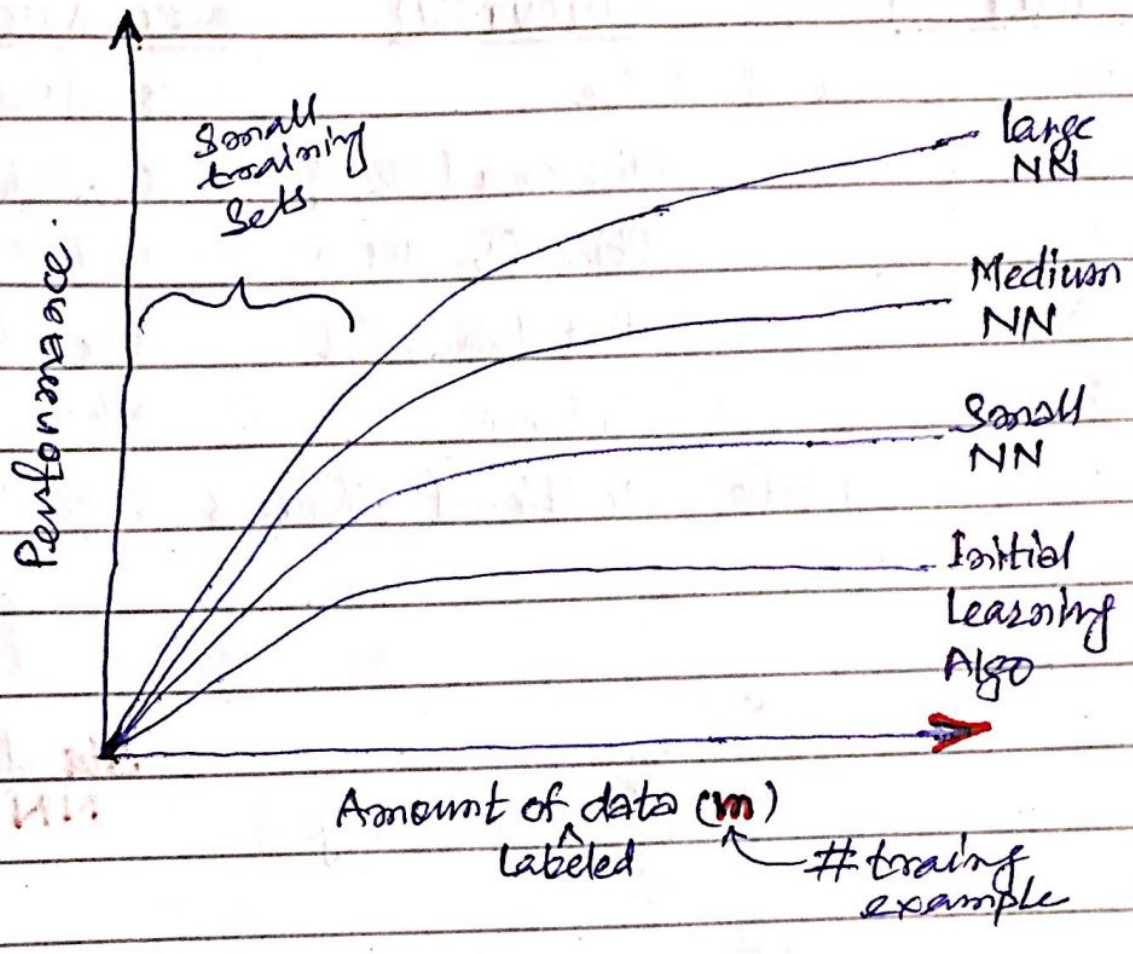
Convolutional NN



Recurrent NN

Scanned with CamScanner

* Scale drives deep learning progress



(i) **Data.**

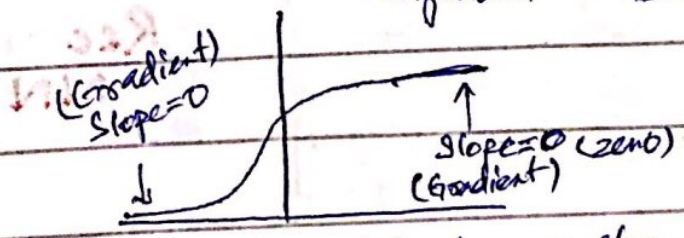
(ii) Computation (**Iterative**)

idea → Experiment → Code

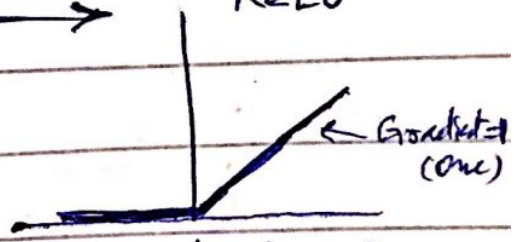
(iii) **Algorithms.**

"Gradient Descent" (work much faster)

Sigmoid → ReLU



As gradient = 0, learning becomes slow. parameters change very slow



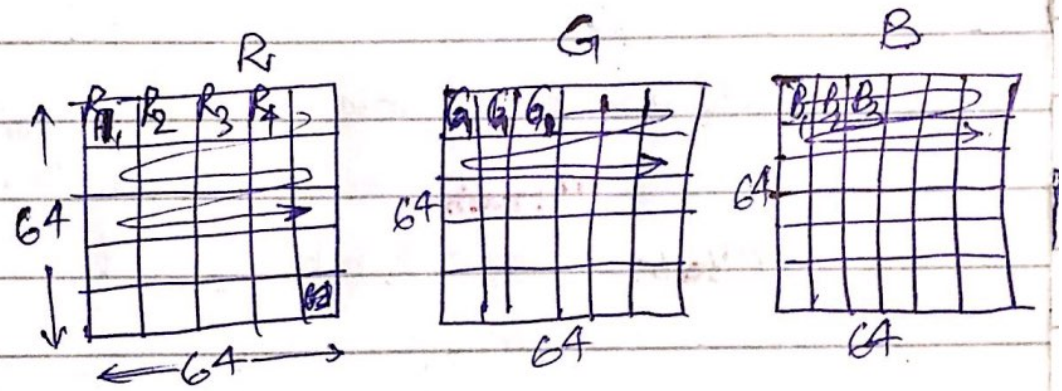
Gradient = 1 for all positive values

Scanned with CamScanner

* Logistic Regression as a Neural Network

* Images are represented in R, G, B channel matrices in Computer.

Eg. If image is of 64×64 pixels.
Then it will be represented as.
3 (64×64) R, G, B matrices.



Input pixel values.

Change into by unrow.

Input feature Vector

$$X = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ G_1 \\ G_2 \\ \vdots \\ B_1 \\ B_2 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

We have to predict $X \rightarrow Y$

Scanned with CamScanner

* Notation.

✓ (x, y) ; $x \in \mathbb{R}^{n_x}$ (n_x dimensional feature vector)

$$y \in \{0, 1\}$$

✓ m training example.

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

✓ Sometimes we denote.

$m = m_{train}$ to denote number training example

$m_{test} \longrightarrow$ number of test examples

✓

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

$$\longleftarrow m \longrightarrow$$

$$X \in \mathbb{R}^{n_x \times m}$$

In python, $X.shape = (n_x, m)$

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

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

In python, $Y.shape = (1, m)$

Scanned with CamScanner

y will tell that image is a cat or not

If x is an Image

✓ Given X, want $\hat{y} = P(y=1|x)$

$(0 \le \hat{y} \le 1)$

$X \in \mathbb{R}^{n_x}$

Parameters $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

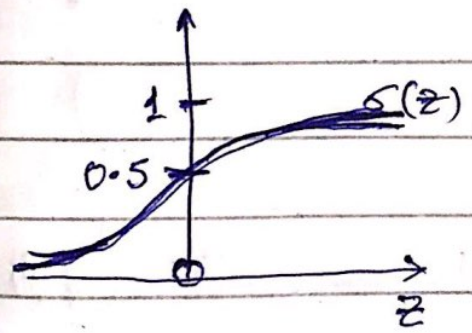
Real Number

✓ Output $\hat{y} = w^T x + b$ it is a linear fn.

here $\hat{y}$ can be much bigger than 1 or any value (-ve) not satisfying $(0 \le y \le 1)$

Therefore using sigmoid fn.

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



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

If z is very large.

$\sigma(z) \approx 1$

If z is large -ve number

$$\sigma(z) = \frac{1}{1+\text{Big Number}}$$

$$\approx 0$$

Scanned with CamScanner

* Logistic Regression Cost Function

✓ To train the parameter w & b
we need to define a cost function.

We have

$$\hat{y} = \sigma(w^T x + b) \quad \text{where} \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})\}$ m training Ex.

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

(predicted value closer to actual)

✓ We have x training example

$$\hat{y} = \sigma(\underbrace{w^T x + b}_{z}) \quad \text{where } \sigma(z) = \frac{1}{1+e^{-z}}$$

→ for i^{th} training example

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b) \quad \text{where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$$

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

Scanned with CamScanner

computes the error for a single training example

Loss (error) function:

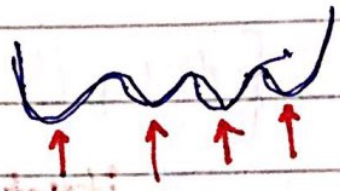
Square error

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

loss fn.

to measure how good $\hat{y}$ as true label is y

By using this fn. optimization problem will become NON CONVEX



So gradient descent may not find GLOBAL OPTIMA

We use

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

loss fn. applied to single training example

its informal justification. (We will look in details later)

If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} = \log\left(\frac{1}{\hat{y}}\right)$

To keep it maximum. $\hat{y}$ should be Large

so, $\log\left(\frac{1}{\text{Large}}\right) \approx \log(0) \approx 1$

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

To keep it minimum. $\hat{y}$ should be small

so, $\log\frac{1}{(1-\text{small})} \approx \log(1) \approx 0$

Scanned with CamScanner

is the **average** of the loss functions of the entire training set.

→ **Cost function** → (cost of parameters)

↳ we need to **minimize** it

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

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

Gradient Descent is an **iterative method** to minimize cost function by improving model parameters (weights & biases)

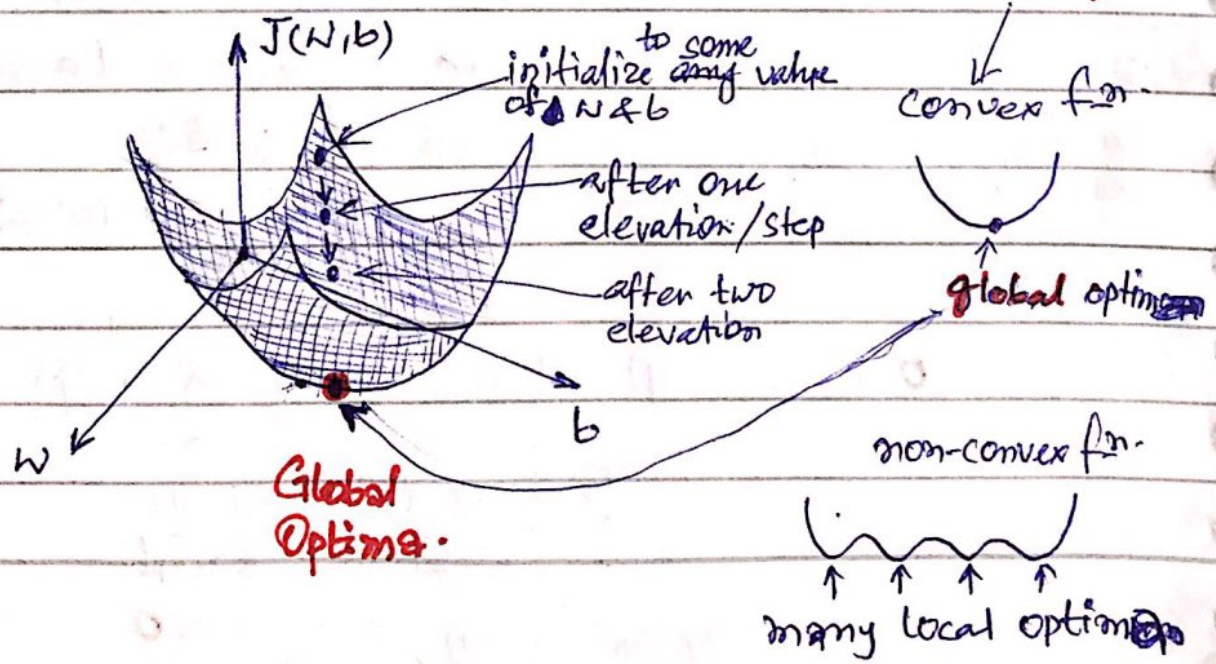
→ **Gradient Descent**

Recap:

$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\boxed{\begin{aligned} J(w,b) &= \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \\ &= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) \end{aligned}}$$

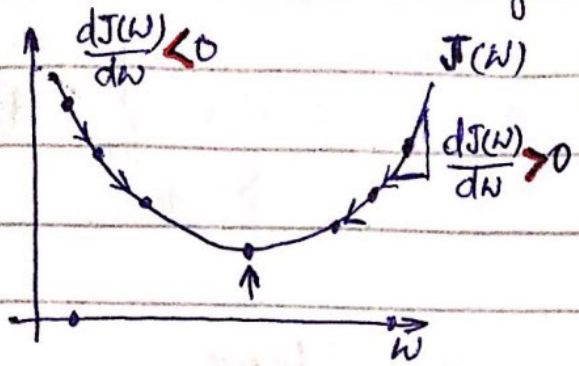
want to **find w & b** to **minimize J(w,b)**



Scanned with CamScanner

$w := \longrightarrow$ denotes updating w

for illustration we are ignoring b for now.



Repeat {

$$w := w - \alpha \frac{d(J(w))}{dw}$$

}

Learning Rate.

notation

dw

$$w := w - \alpha dw$$

for

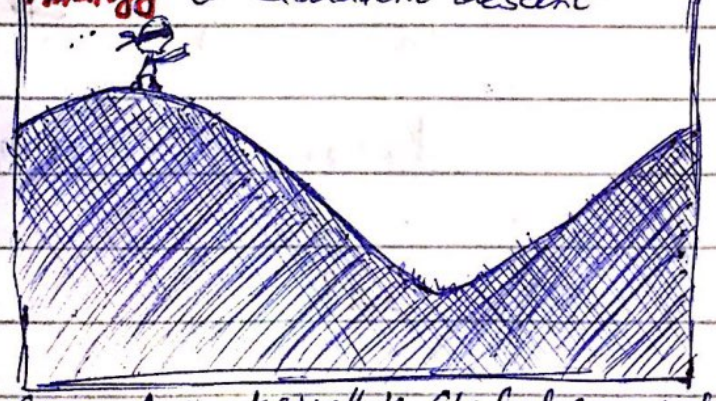
$$\boxed{\begin{aligned} J(w,b) = \quad & w := w - \alpha \frac{\partial J(w,b)}{\partial w} \\ & b := b - \alpha \frac{\partial J(w,b)}{\partial b} \end{aligned}}$$

partial derivative symbol.

because J is a fn. of two or more variable

Otherwise it will be denoted by "d" as derivative. (shown above) for One variable

Analogy to Gradient descent



Source: Andrej Karpathy's Stanford Cs231n Lecture 3

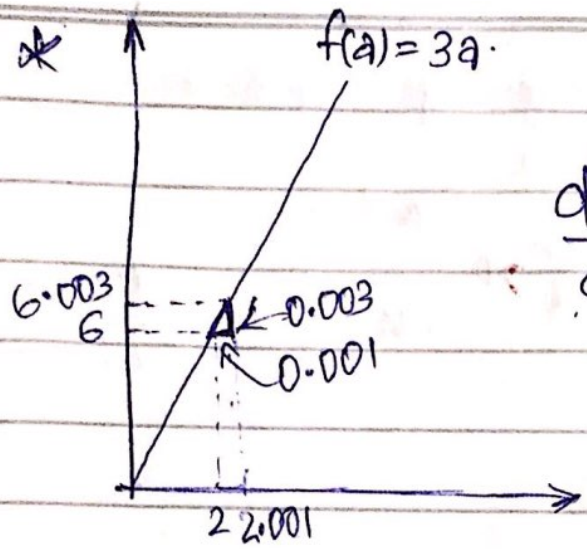
Rough terrain $\longrightarrow$ cost function

Minimizing Cost fn. $\longrightarrow$ trying to reach lower altitude

Feeling slope of terrain around you $\longrightarrow$ calculating Gradient

Taking a Step to the lowest altitude around you. $\longrightarrow$ One iteration of update to the model parameters (weights, biases)

Scanned with CamScanner



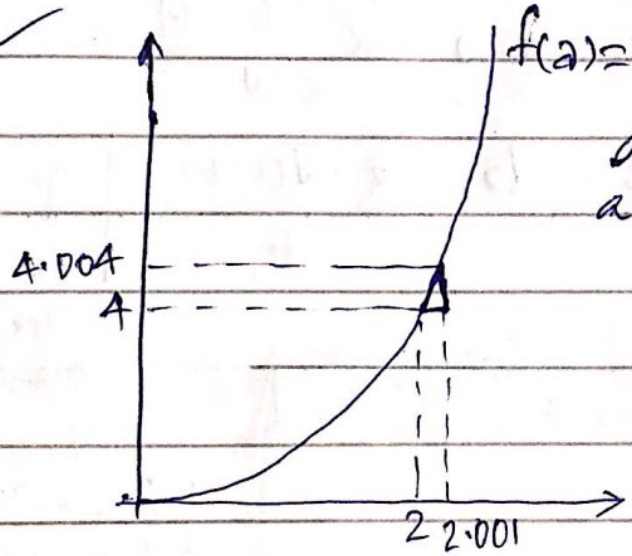
Derivative

$$\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$$

Slope

derivative of f(a) does not change irrespective of 'a' value.

Linear fn



$a = 2$, $f(a) = 4$

$a = 2.001$, $f(a) \approx 4.004$ (4.004001)

$$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$$

$\frac{d}{da} f(a)$ at $a = 5$ is 10

$f(a) = a^3$; $\frac{d}{da} f(a) = 3a^2$

if $a = 2.001$, $f(a) = 8.012$

$f(a) = \log_e(a)$; $\frac{d}{da} f(a) = \frac{1}{a}$.

if $a = 2$ $f(a) = 0.69315$

$a = 2.001$ $f(a) = 0.69365$

$\Delta a = 0.001$ $\Delta f(a) = 0.0005$

Scanned with CamScanner

How to compute derivatives of more complex functions.

Computation Graph.

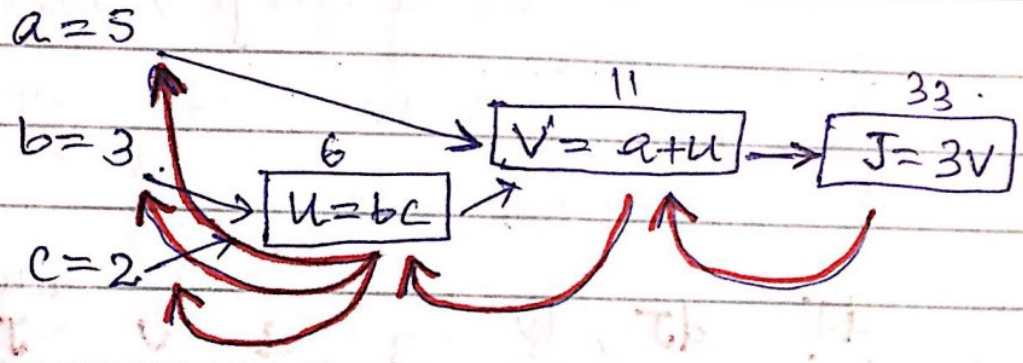
$$J(a,b,c) = \underbrace{3(\underbrace{a+\underbrace{bc}_{u}}_{v})}_{J}$$

computing this fn. includes 3 Steps:

$u = bc$

$v = a + u$

$J = 3v$



Computation graph **helps to optimize** some special output fn. $\boxed{J}$

In case of logistic Regn. it is $\boxed{\text{cost fn}}$

FORWARD

A **Left to Right** Pass.

we can compute the value of J **cost fn**

BACKWARD

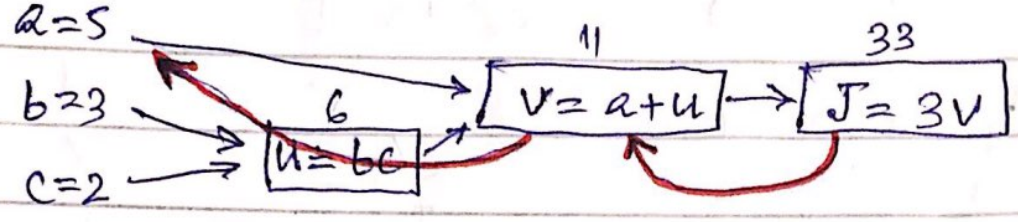
In order to **compute the derivative**.

it will be.

Right to left Pass

Scanned with CamScanner

Derivatives with a Computation Graph.



✓ $\frac{dJ}{dv}$ ⟶ how the value of J change. when we change v value slightly

$\frac{dJ}{dv} = 3.$ $J = 3v$ $v = 11 \rightarrow 11.\underline{001}$
$J = 33 \rightarrow 33.\underline{003}$

✓ $\frac{dJ}{da} = \frac{dJ}{dv} \times \frac{dv}{da} = 3$ $\left\{ a \xrightarrow{\text{affects}} v \xrightarrow{\text{affects}} J \right.$ Chain Row.

$\frac{dJ}{dv} \rightarrow 3$, $\frac{dv}{da} \rightarrow 1$

$a = 5 \rightarrow 5.001$
$v = 11 \rightarrow 11.001$
$J = 33 \rightarrow 33.003$

✓ $\frac{d\ \text{finalOutputVar}}{d\ \text{var}}$ ⟶ "dvar"

Notation for coding.

$\frac{dJ}{db} \rightarrow db = 6$ $\frac{dJ}{dv} \rightarrow dv = 3$

$\frac{dJ}{dc} \rightarrow dc = 9$ $\frac{dJ}{da} \rightarrow da = 3.$

Scanned with CamScanner

Similarly,

$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

$u = 6 \rightarrow 6.001$
$v = 11 \rightarrow 11.001$
$J = 33 \rightarrow 33.003$

$$\frac{dJ}{db} = \underbrace{\frac{dJ}{du}}_{3} \times \underbrace{\frac{du}{db}}_{2} = 6$$

$$\frac{du}{db} = 2.$$

$b = 3 \rightarrow 3.001$
$u = bc = 6 \rightarrow 6.002$ as $c = 2$.

$J = 33 \rightarrow 33.006$

$$\frac{dJ}{dc} = \underbrace{\frac{dJ}{du}}_{3} \times \underbrace{\frac{du}{dc}}_{3.} = 9.$$

$$\frac{du}{dc} = 3$$

$c = 2 \rightarrow 2.001$
$u = bc = 6 \rightarrow 6.003$ as $b = 3$

$J = 33 \rightarrow 33.009$

finally see here

Derivative $\frac{dJ}{dv} = 3$; $\frac{dJ}{da} = 3$; $\frac{dJ}{db} = 6$; $\frac{dJ}{dc} = 9$

Notation $dv = 3$ $da = 3$ $db = 6$ $dc = 9$

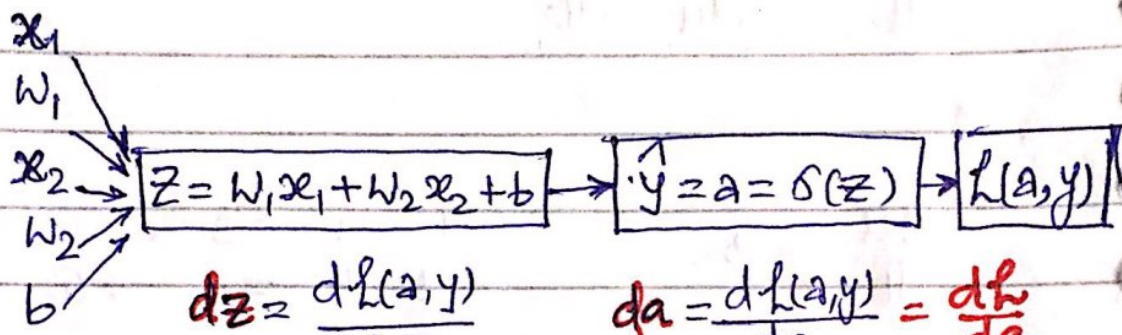
Scanned with CamScanner

* Gradient Descent for Logistic Regression

Recap: $\hat{y} = a = \sigma(z)$

$z = w^T x + b$

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



$dz = \frac{d\mathcal{L}(a,y)}{dz}$

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

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

Refer Pg 40

$= \frac{d\mathcal{L}}{da} \times \frac{da}{dz}$

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

$= a - y$

$\frac{\partial \mathcal{L}}{\partial w_1} = \text{"}dw_1\text{"} = x_1 dz$

$\text{"}dw_2\text{"} = x_2 dz$

$db = dz$

Scanned with CamScanner

Vectorization will help to omit explicit for loops in gradient descent. As for loop slows down execution in dealing with large datasets.

$$\boxed{\begin{aligned} w_1 &:= w_1 - \alpha dw_1 \\ w_2 &:= w_2 - \alpha dw_2 \\ b &:= b - \alpha db \end{aligned}}$$

* Gradient Descent on m examples

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

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

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)}}$$

If

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$

for $i = 1$ to m

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

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

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

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

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

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

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

Two for loops

for $i = 1$ to 2

for $n_x = n = 2$

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

Scanned with CamScanner

*. VECTORIZATION: a method to omit explicit for loops. (wherever possible)

✓ In logistic regression,

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

$w \leftarrow \mathbb{R}^{n_x}$ $x \leftarrow \mathbb{R}^{n_x}$

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

✓ GPU, CPU } SIMD (Single input multiple data.)

→ GPU & CPU use parallelized instructions

SIMD operation

→ GPU are much faster on SIMD calculations

another example ✓

$u = Av$ $u_i = \sum_j A_{ij} v_j$ $u = np.zeros((n,1))$ for i ---- for j ---- $u[i] \mathrel{+}= A[i][j] * v[j]$	$u = np.dot(A, v)$ eliminates two explicit for loops. (much faster)

Scanned with CamScanner

✓ another example : applying exponential operation on every element of a matrix or vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \qquad u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
u = np.exp(v)
```

much faster & eliminates one for loop with use of numpy built-in fn.

```
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i])
```

✓ Using Vectorization on Logistic Regression Derivatives

$J=0,\ \boxed{\cancel{dw_1=0,\ dw_2=0}},\ db=0$

for $i = 1$ to m:

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

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

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

$dz^{(i)} = a^{(i)}(1-a^{(i)})$

for $j=1$ to n_x

$dw_j \mathrel{+}= x^{(i)} dz^{(i)}$

$n_x = 2$

$\boxed{\cancel{dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}} \quad \cancel{dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}}}$

$dw \mathrel{+}= x^{(i)} * dz^{(i)}$

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

$J = J/m,\ \boxed{\cancel{dw_1 = dw_1/m,\ dw_2 = dw_2/m}},\ db = db/m$

$\Downarrow$

$dw = np.zeros((n\text{-}x, 1))$

Scanned with CamScanner

* VECTORIZING LOGISTIC REGRESSION

Let's examine forward propagation step if we have 'm' training examples.

for 1st training ex	for 2nd ---	for 3rd ---
$z^{(1)} = w^T x^{(1)} + b$	$z^{(2)} = w^T x^{(2)} + b$	$z^{(3)} = w^T x^{(3)} + b$
$a^{(1)} = \sigma(z^{(1)})$	$a^{(2)} = \sigma(z^{(2)})$	$a^{(3)} = \sigma(z^{(3)})$

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

training input stack together

(n_x, m)

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

$$= \underset{(1\times m)}{[w^T x^{(1)} + b\ \ w^T x^{(2)} + b\ \ \cdots\ w^T x^{(m)} + b]}$$

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

$$Z = \underbrace{\text{np.dot}(w.T, X)} + b$$

(1,1)

When you add this vector to this row number 'b' python automatically expands it to (1, m) from (1,1)

This is known as "Broadcasting"

"Python Broadcasting"

(1, m)

Scanned with CamScanner

Also

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

* VECTORIZING LOGISTIC REGRESSION'S GRADIENT OUTPUT (examine Backward propagation.)

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ ------ for m training examples

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

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

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

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

→ Previously we omit one for loop. for second for loop, we do the following:-

Initialize $dw = 0$

$$\begin{cases} dw \mathrel{+}= x^{(1)} dz^{(1)} \\ dw \mathrel{+}= x^{(2)} dz^{(2)} \\ \vdots \\ dw \mathrel{+}= x^{(m)} dz^{(m)} \\ dw \mathrel{/}= m \end{cases}$$

for loop involved here

$db = 0$

$$\begin{cases} db \mathrel{+}= dz^{(1)} \\ db \mathrel{+}= dz^{(2)} \\ \vdots \\ db \mathrel{+}= dz^{(m)} \\ db \mathrel{/}= m \end{cases}$$

Let's vectorize it

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

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

$$= \frac{1}{m} [x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)}]_{(1\times n)}$$

Let's vectorize it

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

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

Scanned with CamScanner

* BROADCASTING IN PYTHON

✓ (m,n) matrix $\pm$ $(1,n) \rightsquigarrow (m,n)$

$*$ / $(m,1) \rightsquigarrow (m,n)$

✓ $(m,1) + \mathbb{R}$

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

$$[1,2,3] + 100 = [101, 102, 103]$$

Eg

$$A = \begin{array}{c} \\ \text{Carb} \\ \text{Protein} \\ \text{Fat} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Apples} & \text{Beef} & \text{Eggs} & \text{Potatoes} \end{array} \\ \begin{bmatrix} 56.0 & 0.0 & 4.4 & 68.0 \\ 1.2 & 104.0 & 52.0 & 8.0 \\ 1.8 & 135.0 & 99.0 & 0.9 \end{bmatrix} \end{array}$$

$(3,4)$

cal = A.sum(axis=0)

axis = 0 ↓, 1 →

Percentage = 100* A/(cal.reshap(1,4))

Scanned with CamScanner

* AVOID "RANK-1" ARRAY

a = np.random.randn(5)

print(a.shape) = (5,)

"Rank-1 Array"

It is neither a row vector nor a column vector.

A **funny data structure** with non-intutive effects

use reshape to convert Rank-1 array into some vector

a = a.reshape((5,1))

OR, you can use "**assert**" to check on its shape

code: assert(a.shape == (5,1)), "error msg"

Otherwise **use**,

a = np.random.randn(5,1) → a column vector — print(a.shape) = (5,1)

or, a = np.random.randn(1,5) → a row vector — print(a.shape) = (1,5)

Scanned with CamScanner

* EXPLANATION OF LOGISTIC REGRESSION COST FUNCTION.

$\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

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

$$\begin{cases} \text{If } y=1 : & P(y|x) = \hat{y} \\ \text{If } y=0 : & P(y|x) = 1-\hat{y} \end{cases}$$

Summarize it in one eqn.

$$\boxed{P(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}}$$

Taking log on both the sides,

maximize $\log P(y|x) = \log \hat{y}^{y} (1-\hat{y})^{(1-y)}$

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

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

minimize = -ve of LOSS FUNCTION

Now for cost function,

cost on m examples

$$P(\text{labels in training set}) = \prod_{i=1}^{m} P(y^{(i)} | x^{(i)})$$

Taking log on both the sides,

Scanned with CamScanner

$$\log p(\text{----}) = \sum_{i=1}^{m} \underbrace{\log p(y^{(i)} | x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

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

The above justifies the cost
We have,

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

(minimize)

For convenience that our quantities are better scaled we just add "$\frac{1}{m}$" extra scaling vector

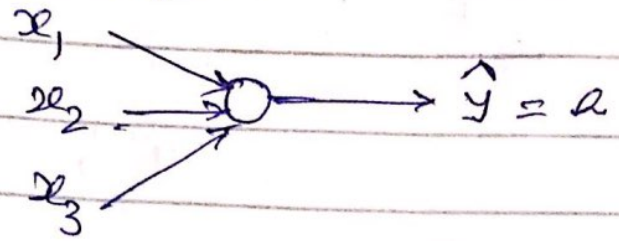
Because we need to minimize the cost and due to maximize likelyhood, we get to rid off minus sign.

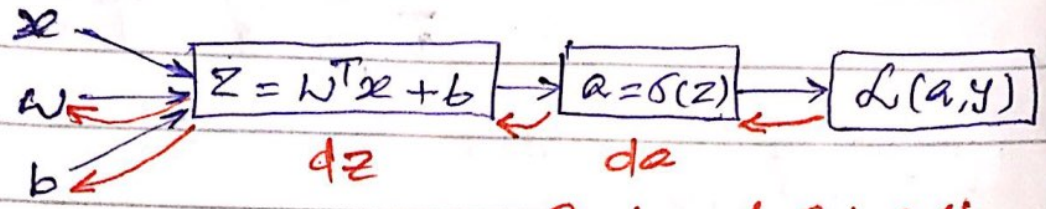
Assumption: Our m training examples are IID (Identically, Independently distributed)

Scanned with CamScanner

* NN Overview (One hidden Layer NN).

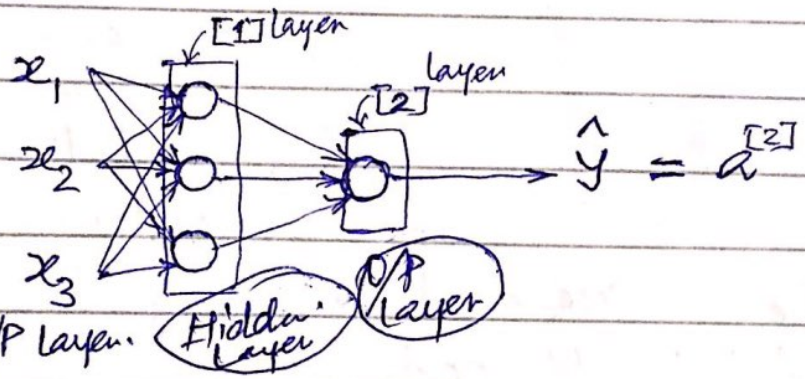
One layer.



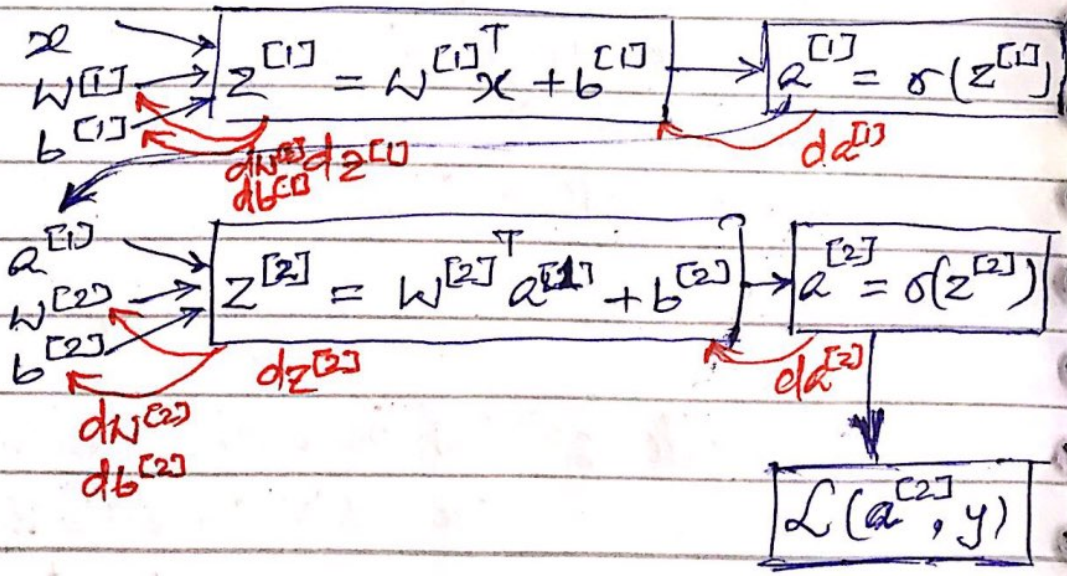


Backward Calculation for computing Derivatives

Two Layer.



do not count I/P layer. (layer 0)



Scanned with CamScanner

* NN Representation

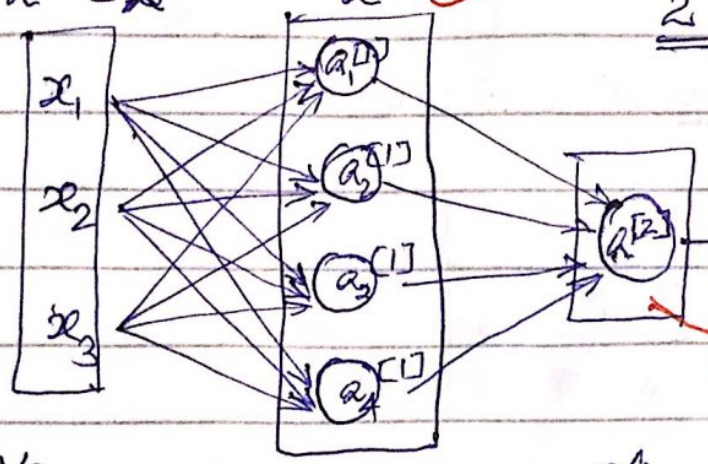
Parameters associated

$W^{[1]}$ (4,3), $b^{[1]}$ (4,1)

4 nodes of hidden layers

3 i/p features

2 Layer NN



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

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

1 node of layer

4 i/p features

i/p Layer $a^{[0]}$

Hidden Layer $a^{[1]}$

o/p Layer $a^{[2]}$

✓ → Notation "a" denotes "activations"
Here i/p layer passes the value X to the hidden layer (Subsequent layer)
Therefore $a^{[0]}$ = activation of i/p layer.

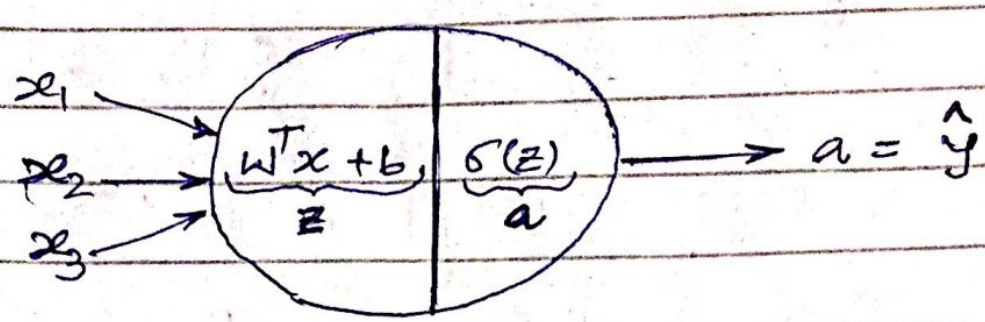
layer zero

✓ $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$ → 4 dimensional matrix
4 nodes

Scanned with CamScanner

* Computing a NN's Output.

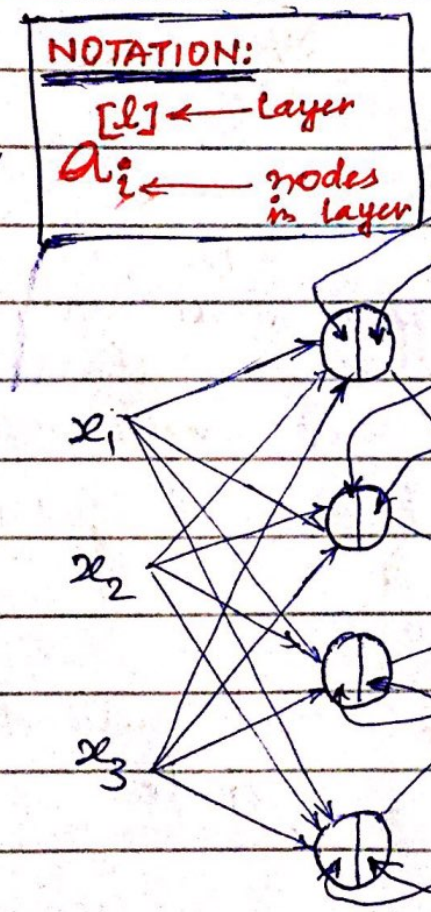
The circle in logistic Regression NN involves <u>two step</u> of calculation



NOTATION:

$[l] \leftarrow$ layer

$a_i \leftarrow$ nodes in layer



$z_1^{[1]} = W_1^{[1]T} x + b_1^{[1]}$

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

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

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

$z_1^{[2]} = W_1^{[2]T} x + b_1^{[2]}$

$a_1^{[2]} = \sigma(z_1^{[2]})$

$z_3^{[1]} = W_3^{[1]T} x + b_3^{[1]}$

$a_3^{[1]} = \sigma(z_3^{[1]})$

$z_4^{[1]} = W_4^{[1]T} x + b_4^{[1]}$

$a_4^{[1]} = \sigma(z_4^{[1]})$

i/p Layer	Hidden Layer	O/p Layer.
Layer-0	Layer-1	Layer-2
$a^{[0]}$	$a^{[1]}$	$a^{[2]}$

Scanned with CamScanner

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

$$z^{[1]} = \underbrace{\begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ \vdots \\ - w_4^{[1]T} - \end{bmatrix}}_{W^{[1]}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{(3,1)} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}_{(4,1)}$$

$$= \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}_{(4,1)} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}_{(4,1)}$$

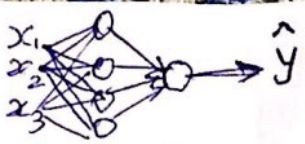
As input $x(x_1, x_2, x_3) = a^{[0]}$

= activation @ Layer 0

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} a^{[0]}_{(3,1)} + b^{[1]}_{(4,1)}$$
$$a^{[1]}_{(4,1)} = \sigma(z^{[1]})_{(4,1)}$$

$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$
$$a^{[2]}_{(1,1)} = \sigma(z^{[2]})_{(1,1)}$$

Scanned with CamScanner



* Vectorizing across multiple ex$^{\underline{s}}$

→ for single training example.

for a given input feature vector x, we can generate $a^{[2]} = \hat{y}$

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

→ for m training examples.

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

NOTATION:

Choose Carefully

$$\boxed{a^{[l](i)}}$$

l → Layer

i → training example i

→ As previously discussed, we have:

$$z^{[1]} = W^{[1]} \overset{a^{[0]}}{x} + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

→ If we have un-vectorized implementation for m training example;

Scanned with CamScanner

Lower Case Vector axis $\xrightarrow{\text{Vectorize.}}$ Capital Case Matrix by stacking lower case vectors in columns.

lower Case

$a^{[0](i)}$

for $i=1$ to m

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

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

$a^{[1](i)}$

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

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

→ Vectorize the implementation of above for loop:

capital

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

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

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

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

Corresponds to different training examples

Horizontal Index

Where,

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

Vertical Index Corresponds to hidden units

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

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

different input features

Scanned with CamScanner

* Explanation for Vectorized Implementation:

We have

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \to 0$$
$$z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \to 0$$
$$z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \to 0$$

Let's keep $b=0$ for convenience of calculation.

If we take different training examples and stack them in different columns.

$$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}, \quad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

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

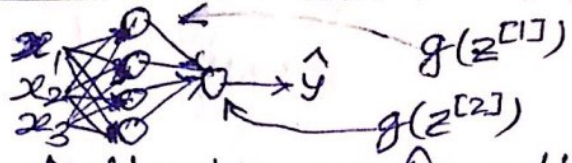
$$W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

Then the corresponding results, also stacked up in columns. (as Z)

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

$$= Z^{[1]}$$

Scanned with CamScanner

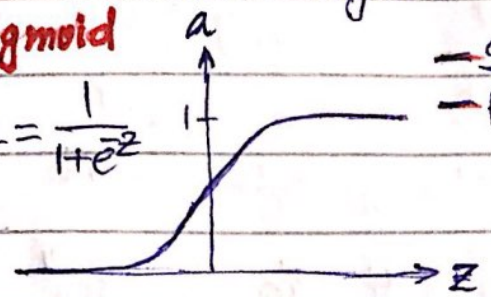


* Activation functions

What Activation f^{n} you will use in hidden layer(s) and o/p layer.

Sigmoid

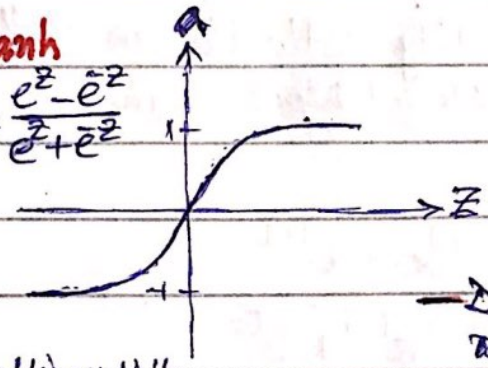
$a = \frac{1}{1+e^{-z}}$



- Suit when o/p is $\{0,1\}$
- For binary classification

tanh

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



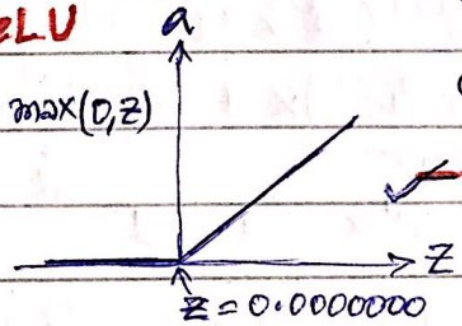
- tanh is almost superior than sigmoid except $\hat{y} \in \{0,1\}$

- Downsize of Sigmoid & tanh is when z is very large or small then the gradient (derivative or slope of this f^{n}) becomes very small (close to zero) → slow down Gradient Descent

Rectified Linear Unit

ReLU

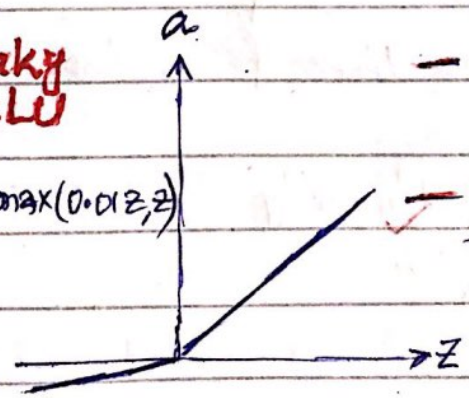
$a = \max(0, z)$



Technically, Derivative is not well defined when z is exactly zero. In practice you can pretend derivative as 1 or 0

Leaky ReLU

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



- ~~As z is~~ for -ve z, derivative is zero for ReLU. Therefore you can use Leaky ReLU.

- In practice, maximum no. of training examples has $z > 0$ therefore ReLU works best.

For given x: (In a forward Propagation)

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

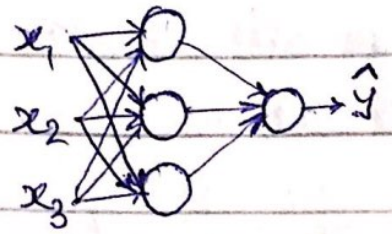
$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$ $\qquad$ $a^{[2]} = \cancel{\sigma(z^{[2]})}\ g(z^{[2]})$

a non-linear function

Scanned with CamScanner

Is a critical part of NN's

* Why do we need **non-linear** activation functions



Forward Propg^n. Equation

for given x:

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

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

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

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

→ If $g(z) = z$ → "Linear activation function" or, "Identity activation function" (as we get Output (z) as given Input (z))

Then,

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

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

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

$$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{b'}$$

$$= W'x + b'$$

So, if we take linear activation f^n. then it turns out that output $\hat{y}$ is a linear f^n of input features

If we do not add non-linearity then we are not computing interesting f^{n}s.

Exception: You might use linear activation f^n. when you use m/c learning on regression problems.

Scanned with CamScanner

* Derivatives of Activation fns.

To implement Backward Propagation in a N.N, we need to compute derivative of Activation fns.

→ **Sigmoid**

$$g(z) = \frac{1}{1+e^{-z}} = a. \quad \left\{ \begin{array}{l} \text{In NN,} \\ \text{activation fn.} \\ a = g(z) \end{array} \right.$$

$$\therefore g'(z) = \frac{d}{dz} g(z) = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right)$$

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

$$= (1+e^{-z})^{-2} \cdot (-1) \cdot \frac{d}{dz}(1+e^{-z})$$

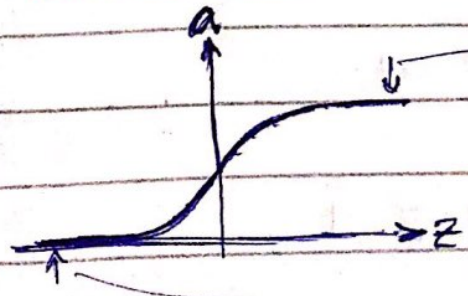
$$= (1+e^{-z})^{-2} \cdot (-1) \cdot \left[0 + (-1)(e^{-z})\right]$$

$$= (1+e^{-z})^{-2} \cdot e^{-z}$$

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

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

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



$z = 10 \quad g(z) \approx 1$

$\frac{d}{dz} g(z) \approx (1(1-1)) \approx 0$

$z = -10 \quad g(z) \approx 0$

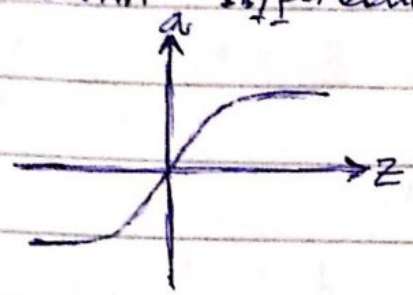
$\frac{d}{dz} g(z) \approx 0(1-0) \approx 0$

$z = 0 \quad g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = 1/2(1-1/2) = 1/4$

Scanned with CamScanner

→ Tanh Hyperbolic Tangent Activation Fn.



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

As per Quotient Rule, $\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x) \cdot g(x) - g'(x) \cdot f(x)}{[g(x)]^2}$

$$\therefore g'(z) = \frac{d}{dz}(\tanh(z)) = \frac{d}{dz}\left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)$$

$$= \frac{\frac{d}{dz}(e^{z} - e^{-z}) \cdot (e^{z} + e^{-z}) - (e^{z} - e^{-z}) \cdot \frac{d}{dz}(e^{z} + e^{-z})}{(e^{z} + e^{-z})^2}$$

$$= \frac{[e^{z} - (-1)e^{-z}](e^{z} + e^{-z}) - (e^{z} - e^{-z})[e^{z} + (-1)e^{-z}]}{(e^{z} + e^{-z})^2}$$

$$= \frac{(e^{z} + e^{-z})^2 - (e^{z} - e^{-z})^2}{(e^{z} + e^{-z})^2}$$

$$= 1 - \frac{(e^{z} - e^{-z})^2}{(e^{z} + e^{-z})^2}$$

$$\boxed{g'(z) = 1 - [\tanh(z)]^2 = 1 - a^2}$$

If $z = 10$, $\tanh(z) \approx 1$

$g'(z) \approx 0$

$z = -10$, $\tanh(z) \approx -1$

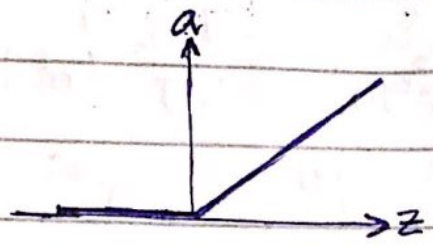
$g'(z) \approx 0$

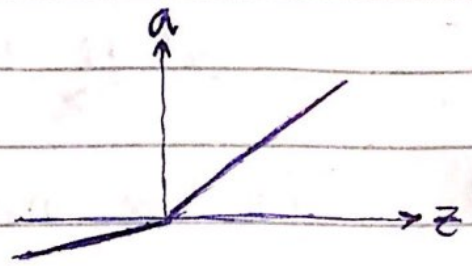
$z = 0$, $\tanh(z) = 0$

$g'(z) = 1$

Scanned with CamScanner

→ ReLU and Leaky ReLU





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

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

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

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

Technically derivative of $g(z)$ is Undefined when z is exactly zero $(z=0)$

But,

Practically you can take

$$z = 0.0000000$$

Hence $\boxed{g'(z) = 1.}$

Scanned with CamScanner

* Gradient descent for NN's

For a single hidden layer NN:

→ Parameters: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

If we have, $n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$ {for this example}

Input Features, Hidden Units, Output Units

Then

→ Cost function:

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

$a^{[2]}$

for Binary Classifn.

→ Gradient Descent

Repeat {

Compute Predictions ($\hat{y}^{(i)}$, for $i = 1 \ldots m$)

$$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}, \quad db^{[1]} = \frac{\partial J}{\partial b^{[1]}}, \ldots$$

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

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

Similarly for $W^{[2]}$ & $b^{[2]}$

→ Summarizing Eqns.

Forward Propagation

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

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

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

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

for Binary Classification Activation fn. is Sigmoid

Scanned with CamScanner

Backward Propagation

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

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

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

Summing horizontally

It prevents python for outputting funny Rank-1 arrays.

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

In this case, $= (1, 1)$

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

Refer. Pg. 40

element-wise product

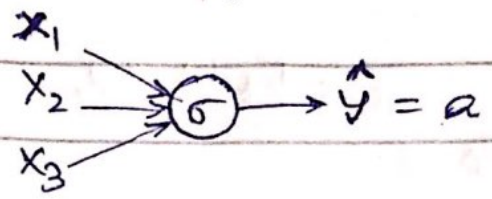
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$\underset{(n^{[1]},1)}{db^{[1]}} = \underbrace{\frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)}_{\text{reshape it into dimension of } db^{[1]} \text{ i.e } (n^{[1]}, 1)}$$

Scanned with CamScanner

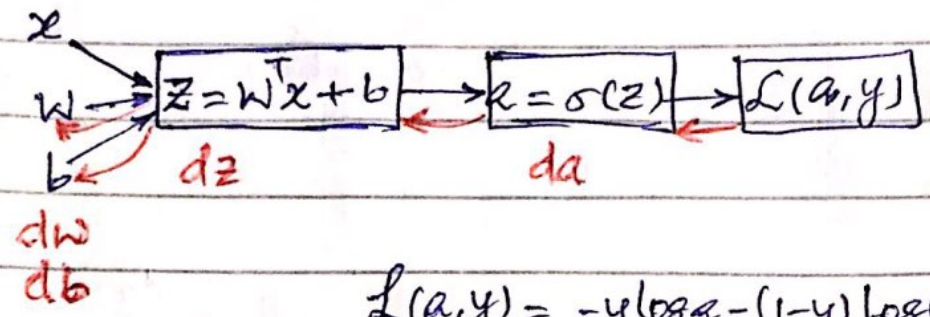
* Backpropagation intuition

→ Logistic Regression:



{ One Layer NN

Computation Graph:



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

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

$$\boxed{dz = a - y = \frac{d}{dz}\mathcal{L}(a,y)}$$

Refer Pg.16

$$\boxed{dz = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}}$$

$$dz = da \cdot dz$$

$$\boxed{dz = da \cdot g'(z)}$$

$a = \sigma(z)$

$a = g(z)$

$\therefore \frac{da}{dz} = \frac{d(g(z))}{dz}$

$= g'(z)$

$$dw = \frac{d\mathcal{L}(a,y)}{dw}$$

$$= \frac{d\mathcal{L}}{dz} \cdot \frac{dz}{dw}$$

$$\boxed{dw = dz \cdot x}$$

$$db = d\mathcal{L}(a,y)$$

$$= \frac{d\mathcal{L}}{dz} \cdot \frac{dz}{db}$$

$$= dz \cdot 1$$

$$\boxed{db = dz}$$

Scanned with CamScanner

$$x \xrightarrow{W^{[1]},\ b^{[1]}} \boxed{z^{[1]} = W^{[1]}x + b^{[1]}} \rightarrow \boxed{a^{[1]} = \sigma(z^{[1]})} \xrightarrow{W^{[2]},\ b^{[2]}} \boxed{z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}} \rightarrow \boxed{a^{[2]} = \sigma(z^{[2]})} \rightarrow \boxed{\mathcal{L}(a^{[2]}, y)}$$

$dW^{[1]}$, $db^{[1]}$, $dz^{[1]}$, $da^{[1]}$, $dW^{[2]}$, $db^{[2]}$, $dz^{[2]}$, $da^{[2]} = \frac{-y}{a^{[2]}} + \frac{(1-y)}{(1-a^{[2]})}$

Dimension is same for a vector and its derivative.

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

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

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

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

similar as

$$dW = dz \cdot x$$

$a^{[0]}$

$$W^{[2]} \xrightarrow{\text{dimension}} (n^{[2]}, n^{[1]})$$

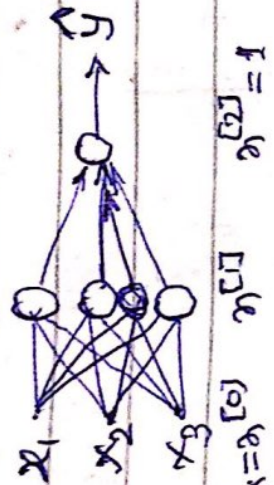
$$z^{[2]}, dz^{[2]} \rightarrow (n^{[2]}, 1) \rightarrow (1, 1)$$

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

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

$$db^{[1]} = dz^{[1]}$$

Two Layer NN



In case of Logistic Reg^n W is a column vector. $W^{[2]}$ is a row vector. Therefore Transpose is used.

$W^{[2]}$ [———]

→ Summary of Gradient Descent { Two layer NN

Vectorize

$$dz^{[2]} = a^{[2]} - y \qquad dZ^{[2]} = A^{[2]} - Y$$

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

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

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]}) \qquad dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

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

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

Gradient Derivatives

Scanned with CamScanner

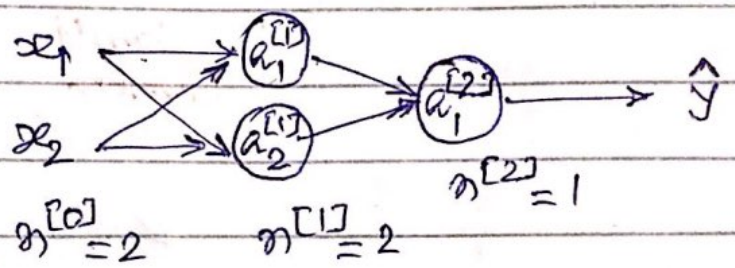
* Random Initialization

✓ for training NN $\xrightarrow[\text{Imp}]{}$ Initialize weights Randomly

✓ for Logistic Regression,

It was okay to initialize weights to zero

→ What happens if we initialize weights to zero



If we initialize, $W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Therefore, $a_1^{[1]} = a_2^{[1]}$ $\qquad dz_1^{[1]} = dz_2^{[1]}$

Both of the hidden units are computing **same function**

So, $a_1^{[1]}$ & $a_2^{[1]}$ are identical or **symmetric**

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\qquad \boxed{W^{[1]} = W^{[1]} - \alpha dW}$ → after one elevation

Every row takes on the same value

$\therefore W^{[1]} = \begin{bmatrix} - & - & - \\ - & - & - \end{bmatrix}$

We want different hidden units to compute different functions **The solⁿ to this is to** Initialize your parameters **RANDOMLY**

Scanned with CamScanner

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

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

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

$b^{[2]} = 0$

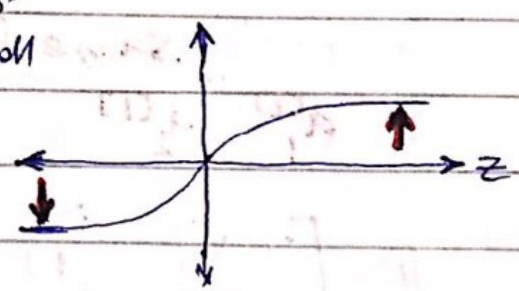
Small Number

If we take large weights value
then when we compute activation values

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

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

If $W^{[1]}$ is big then
$\therefore z^{[1]}$ is very big/small
$\therefore a^{[1]}$ is very big



In case we are doing Binary classification then activation fn. (Sigmoid or tanh) will be saturated (gradient/slope = 0)

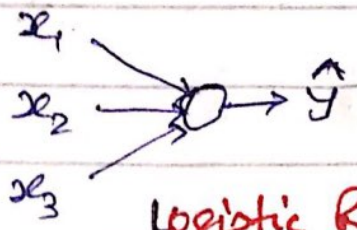
So learning will be slow

Scanned with CamScanner

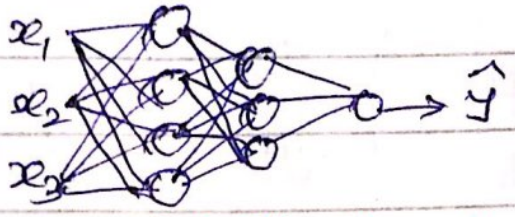
* Deep L-Layer NN

"Shallow"

1 layer NN

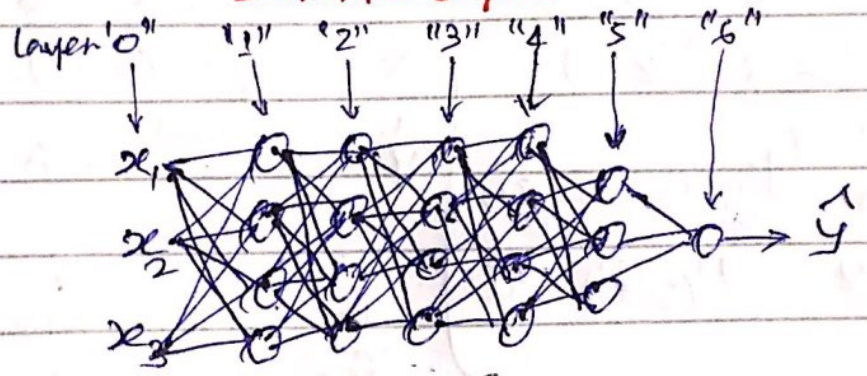


Logistic Regⁿ.



3 layer NN

2 hidden layers



5 hidden layers.

"deep"

6 layer NN

NOTATION:

$L = 6$ (# layers)

$n^{[l]}$ = # of units in layer 'l'

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

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

$n^{[5]} = 3$

$n^{[6]} = n^{[L]} = 1$

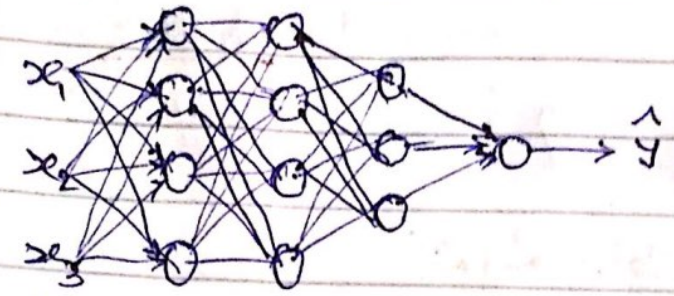
$a^{[0]} = x$, $a^{[L]} = \hat{y}$

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

$a^{[l]} = g^{[l]}(z^{[l]})$ Similarly $w^{[l]}$ & $b^{[l]}$ — weight, bias

Scanned with CamScanner

* Forward Propagation in a Deep Network.



$x = a^{[0]}$

$$Z^{[1]} = W^{[1]} \overset{a^{[0]}}{x} + b^{[1]}$$
$$a^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(Z^{[2]})$$

Similarly,

$$Z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$$
$$a^{[4]} = g^{[4]}(Z^{[4]}) = \hat{y}$$

Vectorized Version:

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]} \quad l=1$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]} \quad l=2$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\hat{y} = g^{[4]}(Z^{[4]}) = A^{[4]} \quad l=4$$

Seems that there is an explicit 'for' loop for forward propagation for layer. $l = 1$ to layer 'L'

This is one place where we can not get rid off **'for' loop**. It is OKAY & Acceptable

$$\boxed{\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned}}$$

Generalized Eq^n. (Vectorized)

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

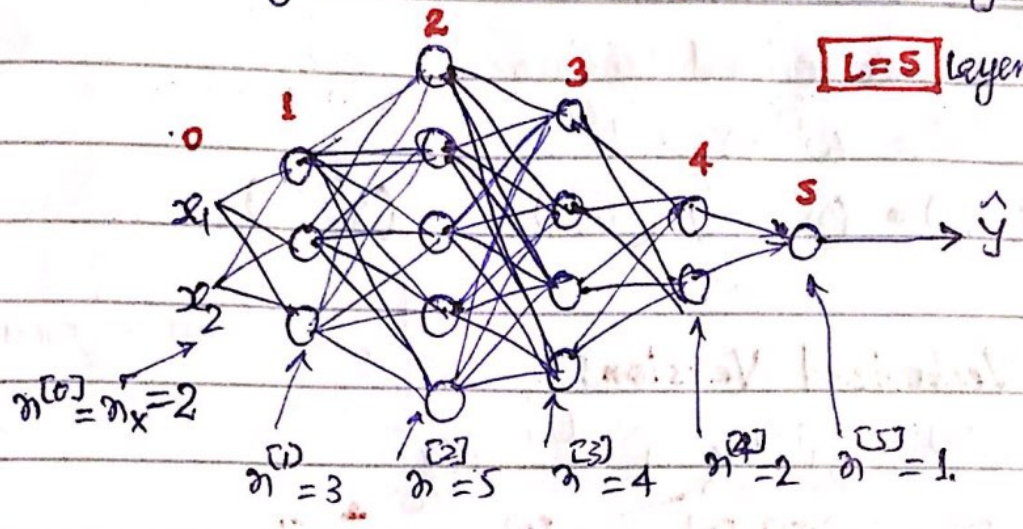
Vectorized version comes by stacking vertically all z for 'm' training examples.

Scanned with CamScanner

One of Debugging Tool

* Getting Your Matrix Dimensions Right

$\boxed{L=5}$ layer 5 NN.



As Z is vector of activations, a column vector.

$\therefore$ $Z^{[1]}$. $(3,1)$ → 3 hidden units in layer '1'

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

$(3,1) \leftarrow (3,2) \cdot (2,1) \quad (3,1) \rightarrow$ This should be

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

Therefore

$Z^{[l]}$	:	$(n^{[l]},1)$	: $dZ^{[l]}$
$W^{[l]}$	:	$(n^{[l]},n^{[l-1]})$	: $dW^{[l]}$
$a^{[l]}$	:	$(n^{[l]},1)$	: $da^{[l]}$
$b^{[l]}$	:	$(n^{[l]},1)$	: $db^{[l]}$

for Backward Propogation

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

Both will have same dimension

See what happens when we Vectorized the given implementation.

Scanned with CamScanner

Non-Vectorized Version.

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

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

Vectorized Version:

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

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

still same

$$(n^{[1]}, m) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, m) \quad (n^{[1]}, 1)$$

By Python Broadcasting It will get duplicated to

(Refer Pg. 22)

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

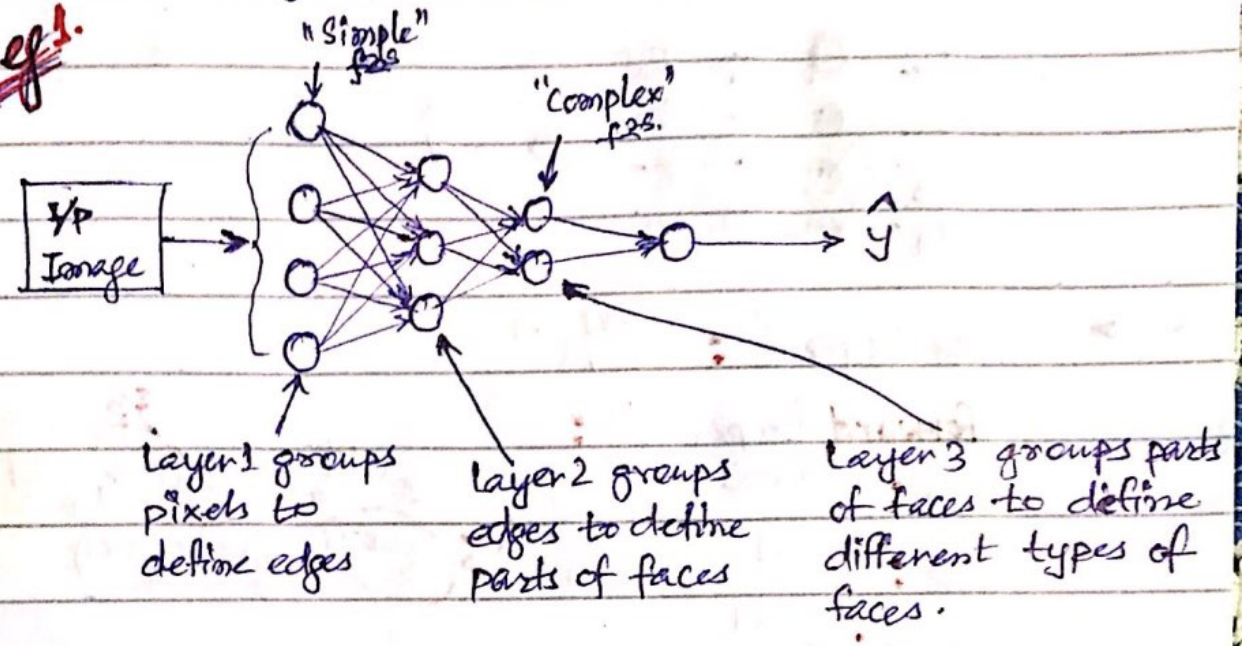
for vectorized version:

$Z^{[l]}$	: $(n^{[l]}, m)$	: $dZ^{[l]}$
$W^{[l]}$	: $(n^{[l]}, n^{[l-1]})$	: $dW^{[l]}$
$A^{[l]}$	: $(n^{[l]}, m)$	: $dA^{[l]}$
$b^{[l]}$	: $(n^{[l]}, 1)$	: $db^{[l]}$

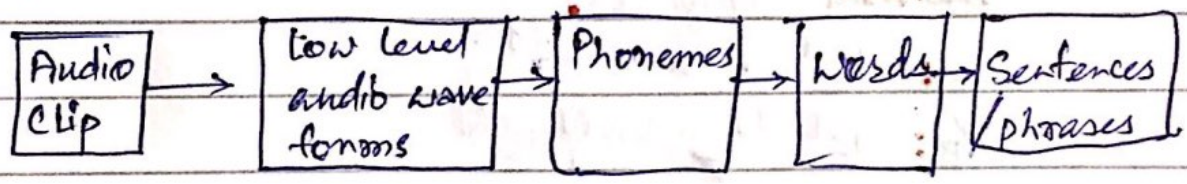
Scanned with CamScanner

* Why Deep Representations

eg 1.



eg 2. Similarly for Speech Recognition:



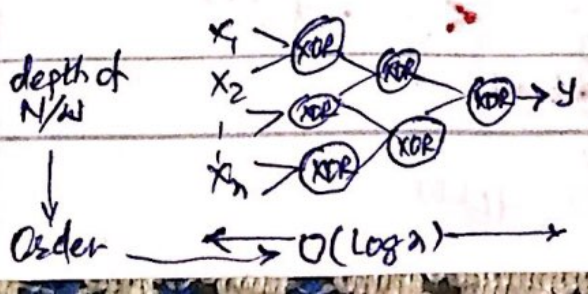
→ Circuit theory & Deep learning

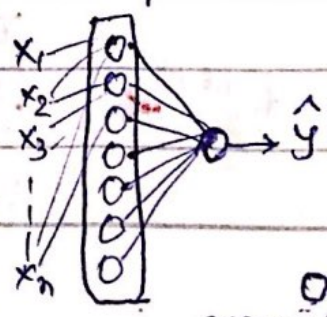
There are fns you can compute with a "Small" L-Layer deep neural network.

that

"Shallower" networks require exponentially more hidden units to compute.

$y = x_1 \text{ XOR } x_2 \text{ — XOR } x_n$

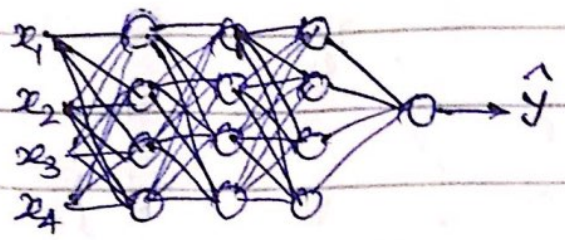




$O(2^n) \approx 2^{n-1}$ exponentially large

Scanned with CamScanner

* Building Blocks of Deep NNs



→ for layer 'l' : $W^{[l]}, b^{[l]}$

Forward Propagation:

I/p : $a^{[l-1]}$

O/p : $a^{[l]}$

Cache : $z^{[l]}$

$$z^{[l]} = W^{[l]} \overset{I/p}{a^{[l-1]}} + b^{[l]}$$

$$\overset{O/p}{a^{[l]}} = g^{[l]}(z^{[l]})$$

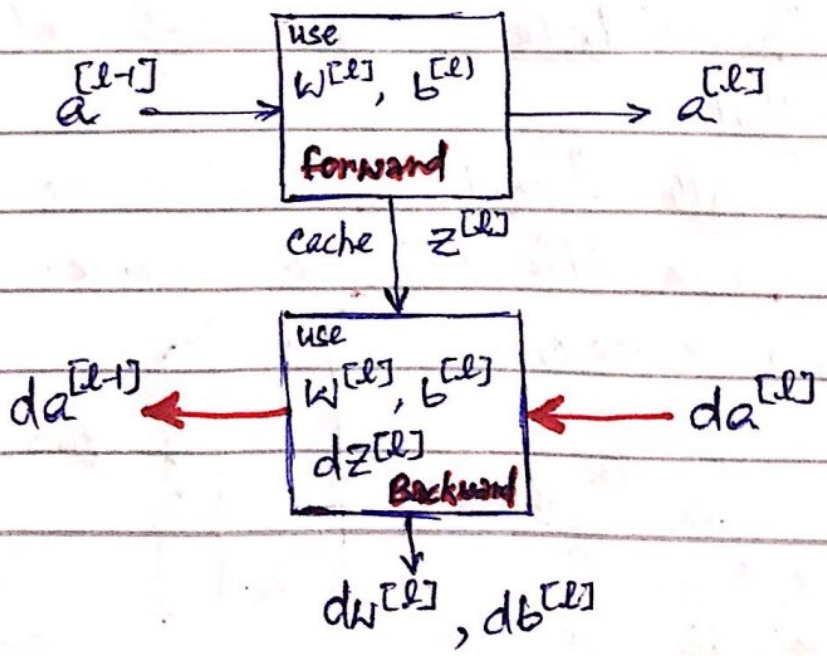
Backward Propagation:

I/p : $da^{[l]}$, cache $z^{[l]}$

O/p : $da^{[l-1]}$, $dW^{[l]}, db^{[l]}$

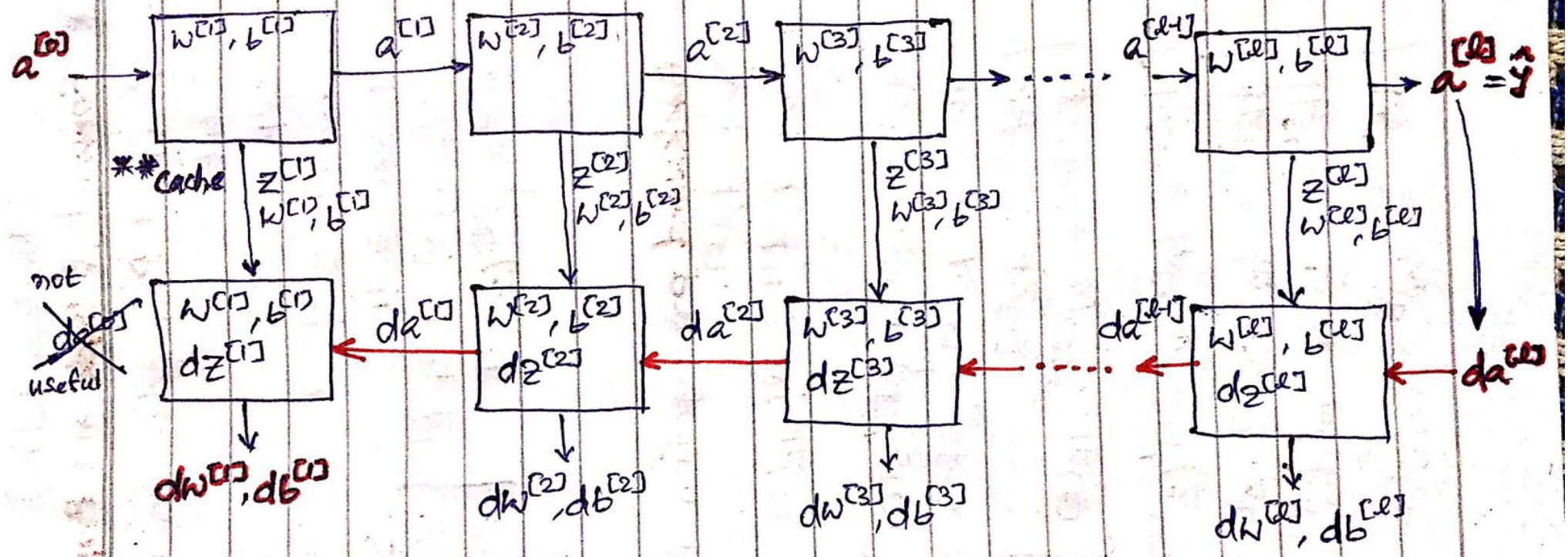
cache :

→ for layer 'l'



Scanned with CamScanner

→ for an 'l' layer Deep Neural Network.



$$\boxed{\begin{aligned} W^{[l]} &= W^{[l]} - \alpha dW^{[l]} \\ b^{[l]} &= b^{[l]} - \alpha db^{[l]} \end{aligned}}$$

** for an implementation (programming) point of view cache $W^{[l]}, b^{[l]}$ too along with $Z^{[l]}$ so that it can be used in backward propagation step(s).

Scanned with CamScanner

(Implementation)

* Forward & Backward Propagation.

→ **Forward Propagation for Layer l**

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

O/P : $a^{[l]}$

cache: $z^{[l]}, w^{[l]}, b^{[l]}$ — For implementation point of view cache it too

Non Vectorized:

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

Vectorized:

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

→ **Backward propagation for Layer l**:

I/P : $da^{[l]}$

O/P : $da^{[l-1]}, dw^{[l]}, db^{[l]}$

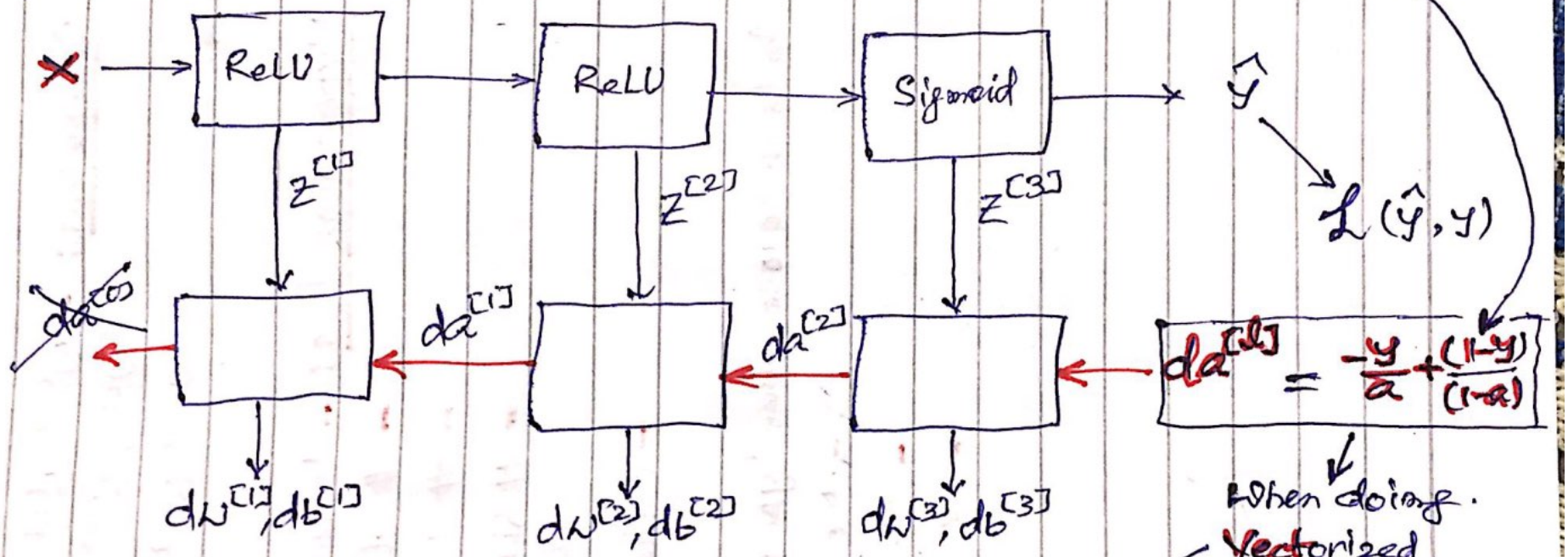
$$\left.\begin{cases} dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]}) \\ dw^{[l]} = dz^{[l]} \cdot a^{[l-1]} \\ db^{[l]} = dz^{[l]} \\ da^{[l-1]} = w^{[l]T} \cdot dz^{[l]} \end{cases}\right\} \text{Non-Vectorized}$$

$$\text{Vectorized} \begin{cases} dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]}) \\ dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T} \\ db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True) \\ dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]} \end{cases}$$

Scanned with CamScanner

Summary:

When doing Logistic Reg.
(for Binary Classification)



When doing Vectorized Implementation

$$dA^{[L]} = -\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots\cdots - \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}$$

Scanned with CamScanner

* Parameters Vs. Hyperparameters.

Parameters: - $W^{[1]}, b^{[1]}$
$W^{[2]}, b^{[2]}$
$W^{[3]}, b^{[3]}$ .....

Controls.

Hyperparameters:
- Learning Rate α
- # iterations
- # hidden Layers L
- # hidden Units
  $n^{[1]}, n^{[2]}, ----$
- Choice of Activation fn

In later course, other hyperparameters:
- Momentum
- Minibatch Size
- Regularizations

Scanned with CamScanner