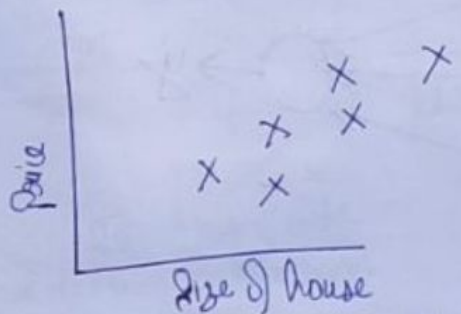
INTRODUCTORY GUIDE TO DEEP LEARNING & NEURAL NETWORK

#1

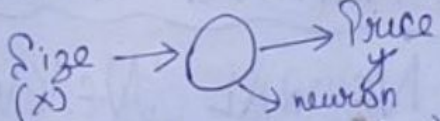
· What is Neural Network?

Suppose we want to predict the price of a house. The variables we are given Size of house & price.

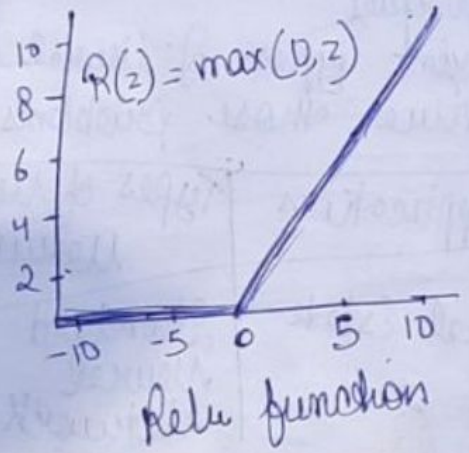


Here, a linear model will try to draw a straight line to fit the data

So, how to solve this problem using Neural Networks



Here neuron will take an input (size), apply some activation function (Relu or Sigmoid). One of the commonly used activation function is Relu (Rectified Linear Unit)



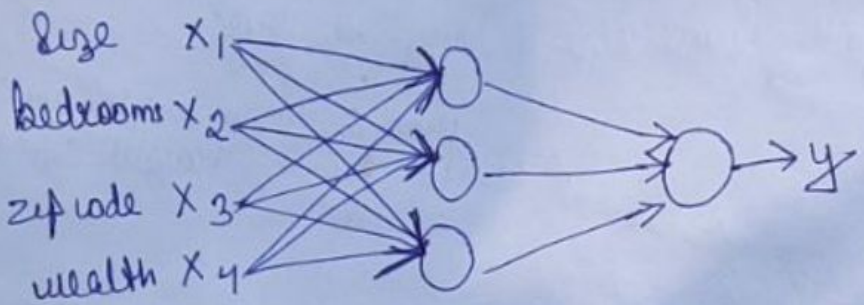
Relu function

Relu takes a real number as input and returns the maximum of 0 or that number. So, if we pass 10, the output will be 10, and if the input is -10, the output will be 0.

Well consider multiple features like number of bedroom, size, postal code.

Here, we have pass 4 features as input to the neural network.

This is how a neural Network with 4 inputs and an output with single hidden layer will look like:



SUPERVISED LEARNING WITH NEURAL NETWORKS

you have input x and you want to learn a function mapping to some output

Supervised learning refers to ~~to~~ find a function that can map ~~input and~~ to corresponding output. We have to ~~defined~~ define output for each given input and we train the model on these examples.

Following are the different types of Neural Network that can be used to solve those problems.

	Input	Output	Application	Types of Neural Network
D	Home features	Price	Real Estate	Standard Neural Network

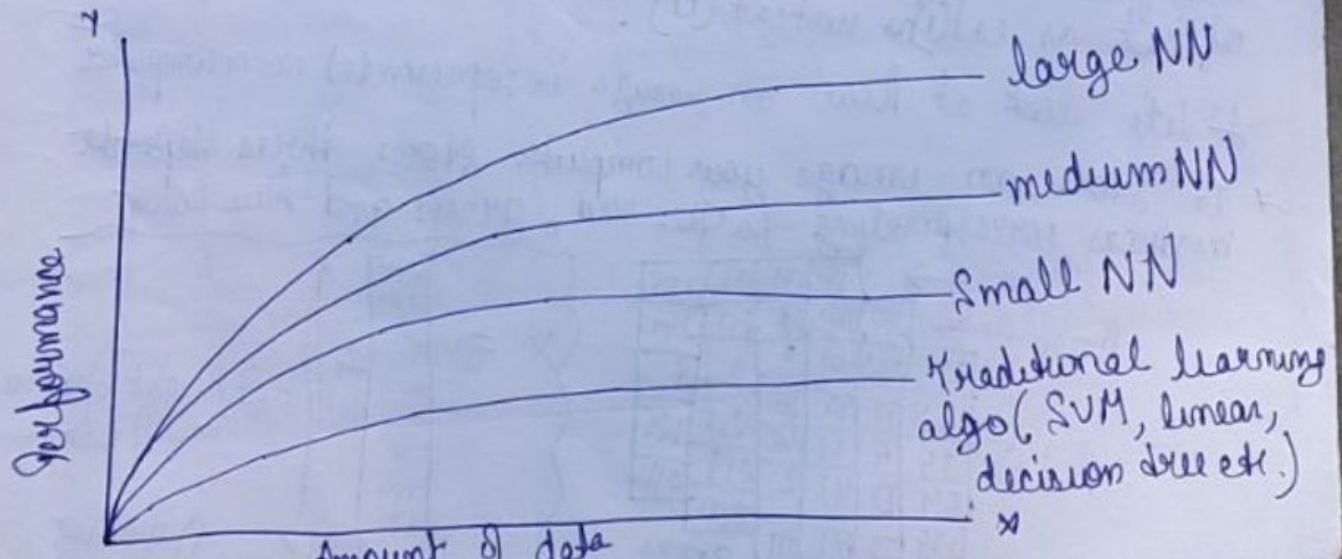
②	Ad, user info	Click prediction (1/0)	Online Advertising	Standard Neural Network
③	Image	Image class	Photo Tagging	CNN
④	Audio	Image class	~~Photo Tagging~~ Speech Recognition	RNN
⑤	English	Chinese	Machine Translation	RNN
⑥	Image, Radar ~~info~~ information	Position of car	Autonomous Driving	Custom/ Hybrid NN

Supervised learning

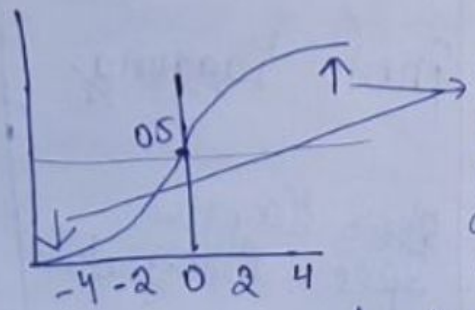
Structured data
- data in table form

Unstructured data
- audio
- text
- video
- Picture

WHY DEEP LEARNING ~~PROGRESS~~ TAKING OFF?



To improve the computation time of the model, activation function plays an important role. If we use sigmoid function



where region here with the slope of the function the gradient is nearly zero and so learning become very slow

The slope or gradient of this function, at the extreme ends is close to zero. Therefore, the parameter are updated very slowly. resulting in very slow learning. Hence, switching from sigmoid to Relu (Rectified Linear Unit) is one of the biggest breakthroughs we have seen in Neural Network this is the primary reason for faster ~~com~~ computation of the model

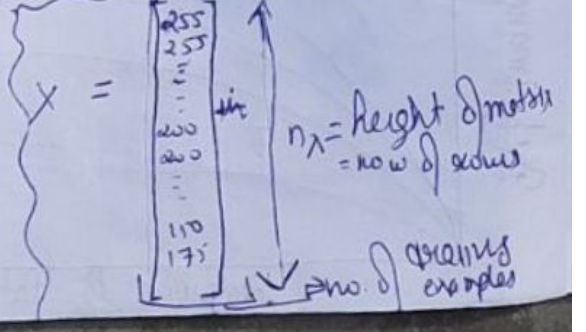
LOGISTIC REGRESSION AS A NEURAL NETWORK

① BINARY CLASSIFICATION

Logistic Regression is an algorithm for binary classification. Suppose we have to recognise a photo of cat as cat(1) or not cat (0).

let's look at how an image is represented in a computer

* To store an image your computer stores three seperate matrices corresponding to the red, green and blue color.



So. if your input image is 64 pixels by 64 pixels then we'll have $3 \times 64 \times 64$ matrices.
↳ Red, Green, Blue.

Now, what we are going to do is define a feature vector x corresponding to this image.

NOTATION

A single training example is represented by a pair of (x, y) where x is an x-dimensional feature and y the label is either 0 or 1 (according to binary classification)

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

[n_x = no. of rows]
[Here we have only 2 column] $\times 64$

m = no. of training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

and your training sets will be written $(x^{(1)}, y^{(1)})$ which is the input and output for your first training example $(x^{(2)}, y^{(2)})$ for the up which is upto $(x^{(m)}, y^{(m)})\}$

↳ all together is your training set

m_{train} = No. of training examples
m_{test} = No. of test examples

we are going to define a matrix, capital X. As defined by taking your training set input $x^{(1)}, x^{(2)}$.

X_1 = first column
X_2 = second column

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

n_x = height of matrix = no. of rows.

m = no. of training examples

we can also use transpose of X (X^T) but it will make our calculations more complex.

$X = \mathbb{R}^{n_x \times m}$ n_x = number of rows
m = number of column

Now what about the output labels Y?

To make your implementation of a neural Network easier it would be convenient to also stack Y in columns

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

$Y \in \mathbb{R}^{1 \times m}$ → no. of columns
→ no. of rows.

Y.shape = (1, m)

② Logistic Regression

formula for logistic Regression is

$$\hat{y}(x) = \frac{1}{1 + e^{-(ax+b)}}$$ a = Slope, b = Antercept
e = euclers

x = matrix of df
y = ~~output~~ target out of dataframe
$\hat{y}$ = (y hat) = predicted output
w = weight
b = bias

what are weights?

different different weights are given to neurons associated with input layer. Random weights are given in forward propagation and weights changed during Backword propagation

what are bias?

Bias are like ~~inc~~ intercept in linear regression they are the value which are added in activation function. During initialising process random weights and bias are put up.

~~Lo Output~~

Lo predictive output for sigmoid function is

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

$z = w^T x + b$

σ → sigmoid function

b = bias

x = Input matrics

w^T = transpose of weight.

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

(3) LOGISTIC REGRESSION COST FUNCTION

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

here $\mathcal{L}$ = loss
$\hat{y}$ = predicted y value
y = Actual output

#To train the parameter w & B of the logistic regression model, we need cost function

loss function is a function measure how good our output ($\hat{y}$) is when the true label of y (actual y value ~~output~~)

Lobe we can also define our loss function as $\frac{1}{2}(\hat{y}-y)^2$ But this will give us an optimization problem that is convex.

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

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

Here, $\mathcal{L}$

This is only for binary classification. We use cross-entropy for multiclassification. Cross entropy → multiclassification

~~If~~ $y=1$ then we want $\hat{y}$ as large as possible

~~If~~ $y=0$ then we want $\hat{y}$ as small as possible

Note: the loss function ~~was~~ we define here is with respect to a single training example. It measure how well you're doing on a single training example

Cost function

* It measures how you're doing on entire training set. So cost function J applied to your parameter W(weight) and B(bias).

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

[i = training example]

(applied to each of training example)

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

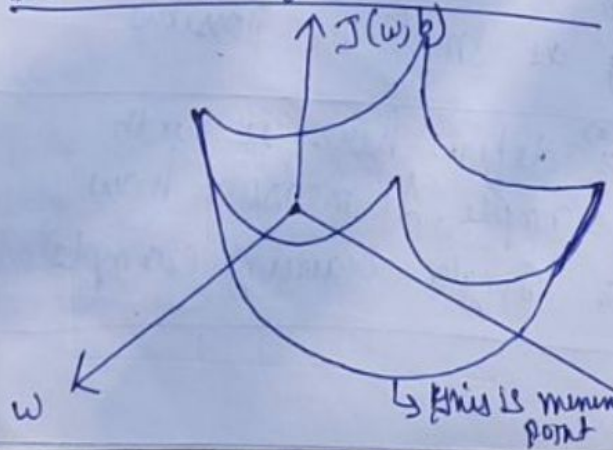
* loss function is applied to just single training example & the cost function is the cost of the parameter. So in training your logistic regression model we're going to try to find parameter (w & B) that minimize the overall cost function

(4) GRADIENT DESCENT

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log y^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})\right]$$

Here its natural that we want to minimise the cost functions.

illustration of a gradient descent (This is convex function)



the height of this graph is J(w,b)

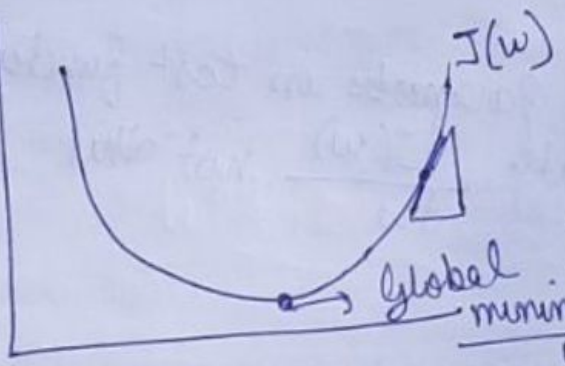
Here we want to find the value of w & b which is minimum

Note: derivative = slope = Gradient descent

* So to find a good value for the parameter what we'll do is initialize w and b to some initial value. For logistic regression any initial value will work
* and what gradient descent does is it starts at that initial point and then takes a step in the steepest downhill direction
* So after taking some steps you'll reach the steepest point. (at minimum value of) w and b

Let's go into bit details

* For the purpose of illustration, let's say that there's some function . J(w) that you want to minimize



minimum point is Global minimum

Here we are ignoring b to plot 2D plot

w := means updated w

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

So the slope of the function is really the height divided by the width.

Note: $\frac{d}{dw}$ are called derivative

How we wrote our gradient descent for J(s) if only w was your parameter

α = alpha = learning rate
α = It tells how big a step we take on each gradient descent

$\frac{d}{dw}$: This is basically the update or the change we want to make to parameter w.

Note: When we'll write the code you wrote "dw" in the code which represents $\frac{d}{dw}$.

In logistic regression your cost function is a function of both w and b ($J(w,b)$)

$$w := w - \alpha \frac{dJ(w,b)}{dw}$$

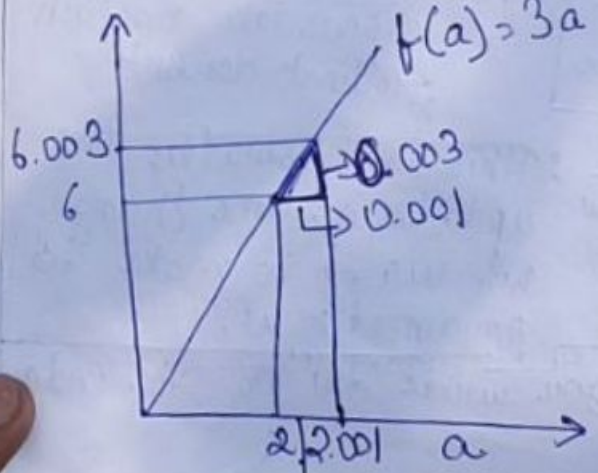
$$b := b - \alpha \frac{dJ(w,b)}{db}$$

$w :=$ means updated weights
$b :=$ means updated bias

Some derivates information

* when we have a parameter in our cost function we have to write $\frac{\partial J(w,b)}{\partial w}$ instead of this $\frac{dJ(w,b)}{dw}$. This is called Partial derivatives.
* If we have only one parameter in cost function we can have to write $\frac{dJ(w)}{dw}$ not this $\frac{\partial J(w)}{\partial w}$

(5) DERIVATIVES



If we put value of $a = 2$ we will get $f(a) = 6$

at $a = 2$, $f(a) = 6$

at $a = 2.001$, $f(a) = 6.003$

Its nothing but Height divided by width

Slope (derivative) of $f(a)$ = $\frac{\text{height}}{\text{width}} = \frac{\text{change in } y}{\text{change in } x}$

$= \frac{0.003}{0.001}$

$= 3$ (here slope is 3)

↑ This means that when we add 0.001 to the x axis Y will increase by 0.003.
watch video for complete

So this is what this equation means ($f(a) = 3a$)

at $a = 5$ $\quad f(a) = 15$

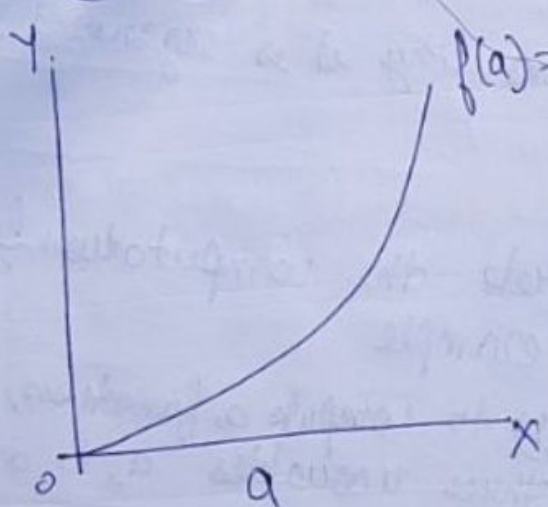
$a = 5.001$ $\quad f(a) = 5.003$

So the way we write this, that the slope of the function f is equal to three is

$\frac{d\,f(a)}{da}$ (da = tiny little amount) what this means is the slope of a function $f(a)$ when we add a with a tiny little amount this is equal to 3

we can also write $\frac{df(a)}{da}$ as $\frac{d}{da} f(a)$

(6) MORE DERIVATIVE EXAMPLES



$f(a) = a^2$ $\quad a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) = 4.004$

↓

How?

because here line is not straight because of this values of three after decimals changes
Here is how

~~a = 2.001~~ ~~f(a) = 4.004~~

~~a = 2 + 0.001~~ ~~f(a) = a²~~

~~a 0.001 0.00~~

① $f(a) = a^2$ $\qquad \frac{d}{da} f(a) = 2a$ $\qquad$ for $a = \underline{2}.001$

$f(a) = 2a \rightarrow 4$

$f(a) = 4.004$

lets take another example to understand this clearly

② $f(a) = a^3$ $\qquad \frac{d}{dx} f(a) = 3a^2 \rightarrow 6a$ $\qquad a = 2.001$

$f(a) = 8.012$

⑦ COMPUTATION GRAPH

Neural networks are organised in terms of a forward pass (forward propogation) followed by backward pass (back propagation)

computation graph explains why it is organized this way. ⟶ Imp

~~Bac...~~

In order to ~~comp~~ illustrate the computation graph lets take a simple example

let's say that we're trying to compute a function, J, which is the function of three variables a, b and c

Let's say

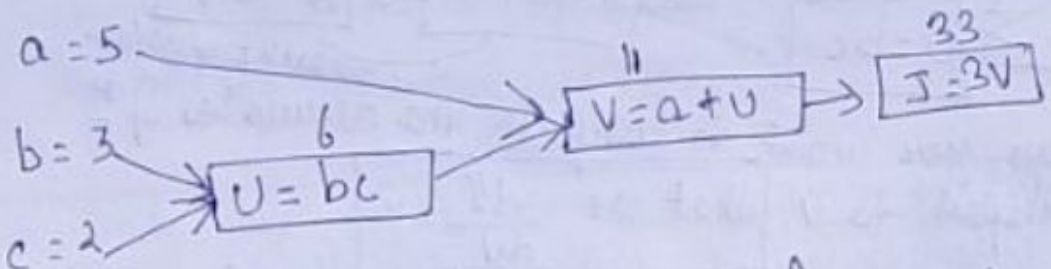
$J(a,b,c) = 3(a+bc)$

$u = bc$

$V = a + u$

$J = 3V$

Let's take a example here



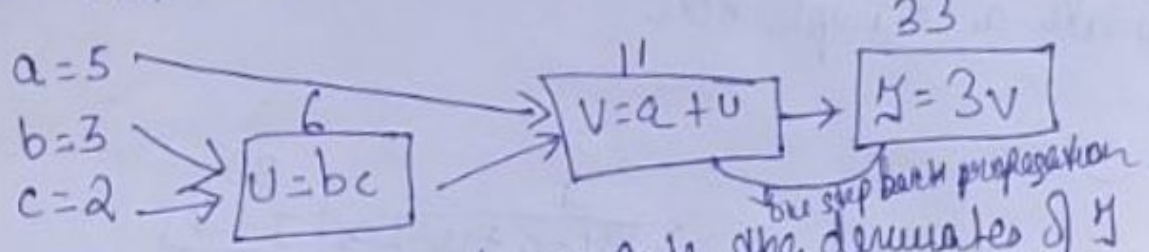
So, the computation graph comes in handy when there is some distinguished or some special output variable such as J in this case that you want to optimize.

Note: One step of backward propagation on a computation graph yields derivate of final output variable

COMPUTATION GRAPH

⑧ DERIVATIVES WITH COMPUTATION GRAPH

Let's, here's a computation graph



let's say you want to compute the derivates of J with respect to v. that is $\frac{dJ}{dv} = ?$

↳ what this means that if we take this value of v and change it a little bit how would the value of J change.

J = 3v ↳ according to formula
and right now v = 11

* So if we want to increase a little bit value of v from 11 to 11.001 then J will pump up from 33 to 33.003

* So the derivates of J with respect to v is equal to 3 that is $\frac{dJ}{dv} = 3$ because the increase in J is 3 times the increase in v. $\frac{dJ}{dv} = \frac{3dv}{dv} = 3$

* In the terminology of backpropagation is that if you want to compute the derivates of this final output variable with respect to v, then we've done one step of backpropagation

* Now, what's the value of $\frac{dJ}{da} = ?$

Mode: when we calculate respect to $\frac{dJ}{dv}$ then this is one step of backpropagation

$\frac{dJ}{da} = ?$ $a = 5$ (In our example) If we pump up the value of a, how does that effect the value of J

$V = a + v = 11$ (In our example)

~~Let's call a~~

* If we increase the value of a ~~to~~ from 5 to 5.001 v which was a + v. This would get increased to 11.001 and also J move up from 33 to 33.003

This means the derivate $\left(\frac{dJ}{da} = 3\right)$

Note: One way to break this down is to say that if you change a then it will change v and through changing v that would change J.
→ in calculus this is called chain rule

* if you increase a by 0.001 v changes by the same amount. So, $\frac{dv}{da} = 1$

wrapping up:-

Here: $\frac{dv}{dJ} = 3$ and $\frac{dv}{da} = 1$

So the product of these is 3 time 1.

So, thats gives you a the correct value of $\frac{dJ}{da}$

$= 3.$

$\frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du}$ ← multiplication

$3 = 3 \times 1$

$3 = 3$

* $\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db}$

$= 3 \times 2$ $\boxed{c=2}$

$db = 6$

* $\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc}$

$= 3 \cdot b$ $\boxed{b=3}$

$= 3 \times 3$

$\boxed{dc = 9}$

⑨ LOGISTIC REGRESSION GRADIENT DESCENT

logistic regression recap

$z = w^T x + b$

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

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

Logistic regression derivatives

→ # Here we have only 2 feature x_1 & x_2

x_1, w_1, x_2, w_2, b → $z = w_1x_1 + w_2x_2 + b$ → $a = \sigma(z)$ → $\mathcal{L}(a, y)$

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

"da" [Python] $da = \frac{d\mathcal{L}(a,y)}{da}$ explained below

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

$= a - y$

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

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

(So in logistic regression we want to modify our parameter)

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

$dw_2 = x_2 \cdot dz$

$db = dz$

GRADIENT DESCENT On m ~~Exp~~ Examples

[illegible]

In in previous topic, you saw how to compute derivative and implement of gradient descent with respect to just one training example. In this we just implement in all training examples.

we can define the predictions and cost function for 'm' training examples as:

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

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

~~The derivative of a loss function have we can apply logistic regression for in example~~

The derivate of a loss function can be written as:

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

From the above we can conclude the logestic regression code:

```
J=0; dw1=0; dw2=0, db=0; #Deve
w1=0; w=0, b=0;
for i=1 to m
    # for-ward pass
    z(i) = w1*x1(i) + w2*x2(i) + b
    a(i) = Sigmoid(z(i))
    J += -(y(i)*log(a(i)) + (1-y(i))*log(1-a(i)))
    #Backward pass
    dz(i) = a(i) - y(i)
    dw1 += dz(i) * x1(i)
    dw2 += dz(i) * x2(i)
    db += dz(i)
```

$J/ = m$

$dw1/ = m$

$dw2/ = m$

$db/ = m$

Gradient descent

$w1 = w1 - alpa * dw1$

$w2 = w2 - alpa * dw2$

$b = b - alpa * db$

PYTHON AND VECTORIZATION

(1) Vectorization

Vectorization is basically the art of getting rid of explicit for loop in your code. It executes the code fast.

$z = w^t x + b$ — w and x are vectors

Non-vectorization takes more time than vectorized

Non-vectorized implementation

```
z = 0
for i in range(n,x)
    z += w(i) * x(i)
z += b
```

Vectorized implementation

```
z = np.dot(w, x) + b
```

$w^t x$

Code implementation for both together so that we can see time taken by each of them

```
import numpy as np
a = np.array([1,2,3,4])
print(a)
```

Out | [1,2,3,4]

```
import time
a = np.random.rand(1000000)  → It creates an array of specified shape and fills it with random values
b = np.random.rand(1000000)
tic = time.time()  → time taken to implement this code
c = np.dot(a,b)  → a^T x + b
toc = time.time()
print(c)
print("Vectorized version:" + str(1000*(toc-tic)) + "ms")  (here we have multiplied by 1,000 to get milliseconds)

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()
print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

Out:

```
250286.989866
Vectorized version: 1.5027523... ms
250286.989866
For loop: 474.29513931... ms
```

So, the non-vectorize version took something like 300 times longer than the vectorize version.

② MORE VECTORIZATION EXAMPLES

(Whenever possible, avoid explicit for-loops)

If you want to compute a vector u as the product of matrix A and another vector v

$$\boxed{u = Av}$$

But this is define as $u_i = \sum_j A_{ij} v_j$

Implementation of this in numpy will be

$u = np.dot(A, v)$ ⇒ This is vectorized

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector:

> exponential function have the form $f(x) = b^x$, where $b > 0$ and $b \neq 1$

Ex.
```
import numpy as np
u = np.exp(v)
```

↳ If we apply this as non-vectorized in python we have use 2 for loops and for loops are time consuming

Whenever you are tempted to write a for loop take a look, and see if there's a way to call to call a Numpy built-in function to do it without that for-loop

(3) VECTORIZING LOGISTIC REGRESSION

for ~~drawing~~ loop

$z^{(1)} = w^T x^{(1)} + b$ $z^{(2)} = w^T x^{(2)} + b$ $z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $a^{(2)} = \sigma(z^{(2)})$ $a^{(3)} = \sigma(z^{(3)})$

> we might need to do this m times if we have m training examples

we can we can calculate all the value of $z^{(1)}$ without using for loop in python.

Let's see how we can do it.

X = [illegible] b

X ↓ training input

$\begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$ this is (n_x, m) matrix

How to calculate $z^{(1)}$ all at same time?

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

w^T will be a row vector

X ↓ training input

b ↓ bias ↓ $1 \times m$ vector

This is equal to the equation on starting of topic

we'll end up adding b at every element

An numpy we can calculate

$z^{(1)} = w^T x^{(1)} + b$

as

```
z = np.dot(w.T, x) + b
```

↳ b is 1X1 matrix (just one number)

(4) BROADCASTING IN PYTHON

Broadcasting is an another technique that you can use to make your Python codes run faster

example:-

Calories from carb, protein, fat in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
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

But what if we want to calculate Average of these number in python

In
```
import numpy as np
A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]])
print(A)
```

Out
```
[[ ____ , ____ , ____ , ____ ]
 [ ____ , ____ , ____ , ____ ]
 [ ____ , ____ , ____ , ____ ]]
```

In
```
cal = A.sum(axis=0)
print(cal)
```

Out
```
[59.  239.  155.4  76.9]
```

In
```
percentage = 100*A/cal.reshape(1,4)
print(percentage)
# We don't have to use reshape here because our
  matrix cal is already (1,4)
```

Out
```
[[ ____ , ____ , ____ , ____ ]
 [ ____ , ____ , ____ , ____ ]
 [ ____ , ____ , ____ , ____ ]]
```

Broadcasting more examples

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

~~100~~ ↓ dont use this

(2)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(m,n)} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$$

(don't use this)

(5) A NOTE ON NUMPY VECTORS

Tips and tricks to eleminate all the strange looking bugs in code :

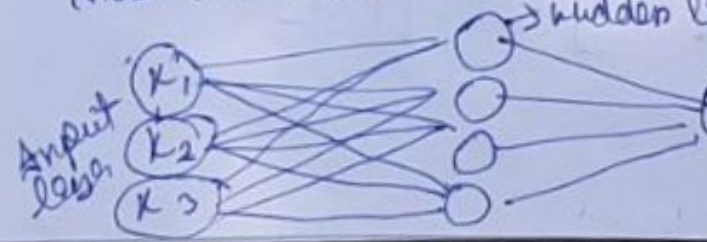
* when you want to create a random matrix of 5 row and 1 colum we generally write
  a = np.random.randn(5)
but when we print this variable a we'll get 5 column instead of 5 row.
So always use this
  a = np.random.randn(5,1)

Note: Don't use rank 1 array

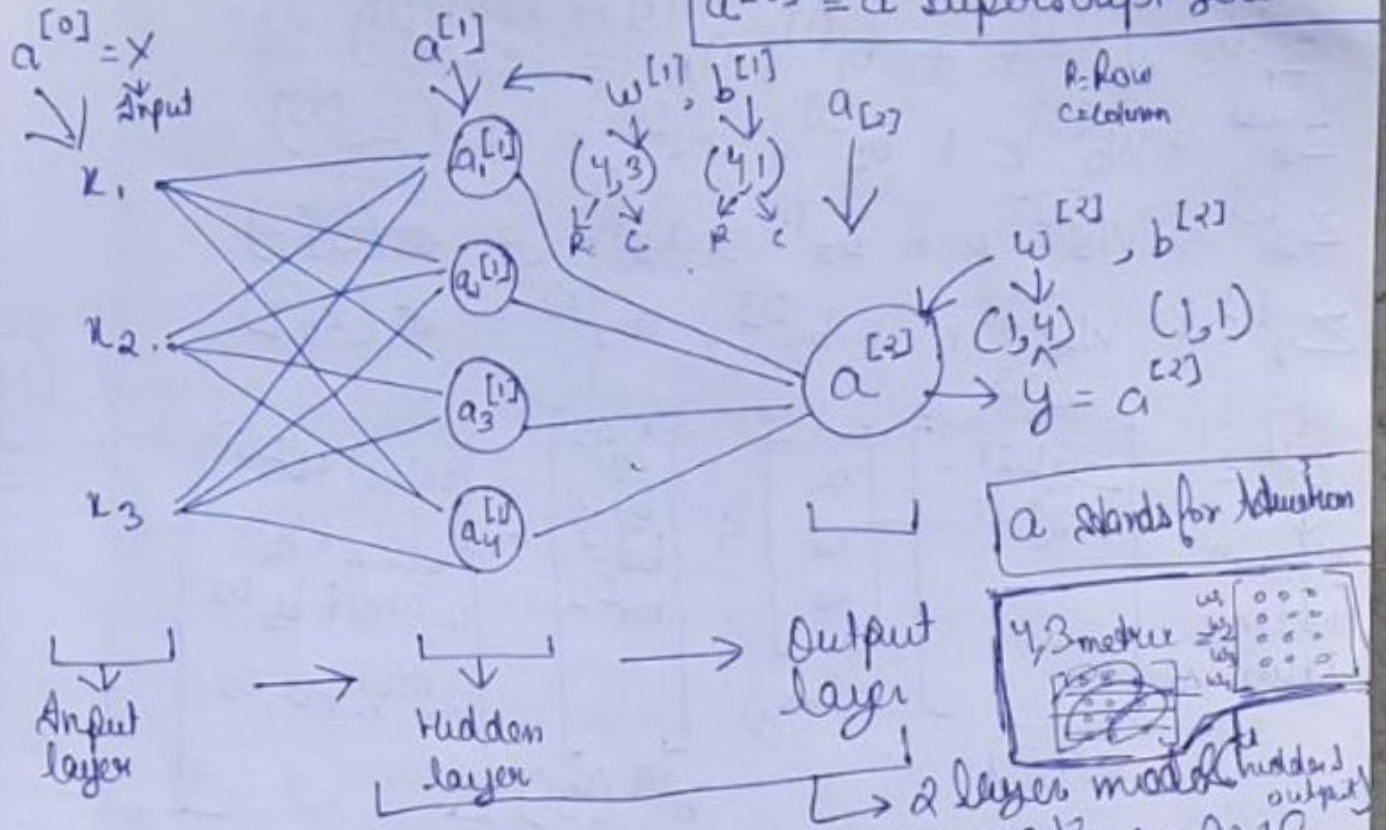
WEEK 3

(1) NEURAL NETWORK REPRESENTATION

This is how neural network looks like :-

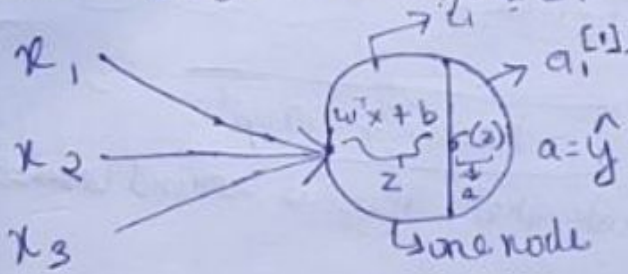


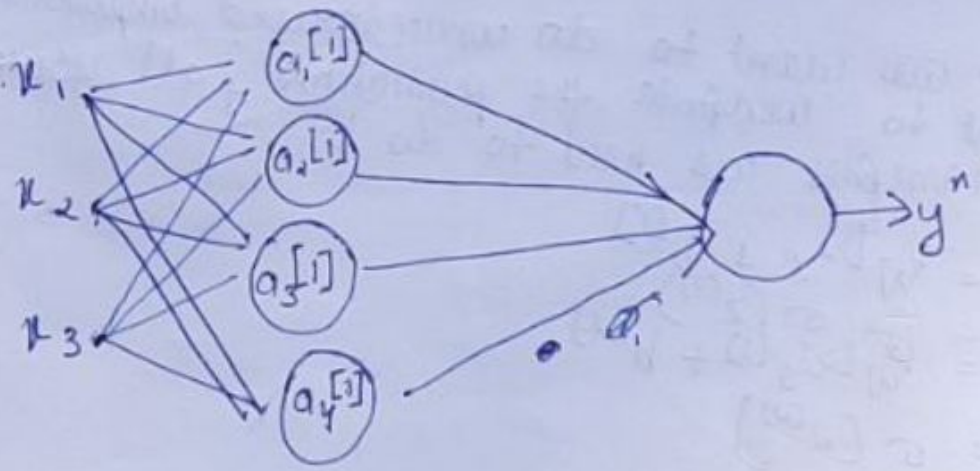
$a^{[0]}$ = a superscript zero



(2) COMPUTING A NEURAL NETWORKS OUTPUT

Each node/neuron in the hidden has Activation function. (Sigmoid or Relu)





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

$w^{[1]}$

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

$z^{[1]}$ → i^{th} or 1^{st} Hidden layer

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

$$= \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

③ VECTORIZING ACROSS MULTIPLE EXAMPLES

i = training example

$a^{[2](i)}$ → example i (please note that this is round bracket)
→ layer 2

Suppose we want to do unvectorized implementation and want to compute the predictions of all your training examples we need to do is

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

m = training examples

for i = 1 to m:

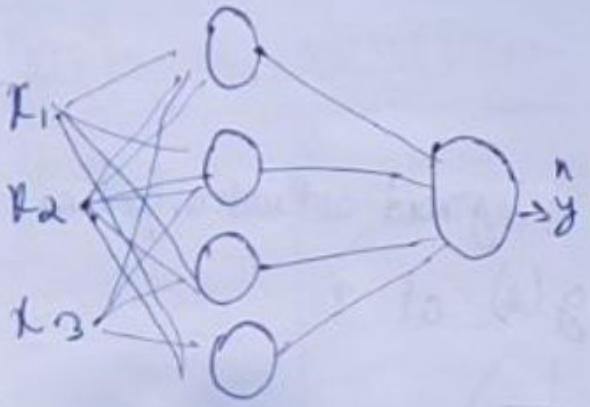
$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$

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

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

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

(4) EXPLANATION FOR VECTORIZED IMPLEMENTATION



For i = 1 to m

same

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

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

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

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

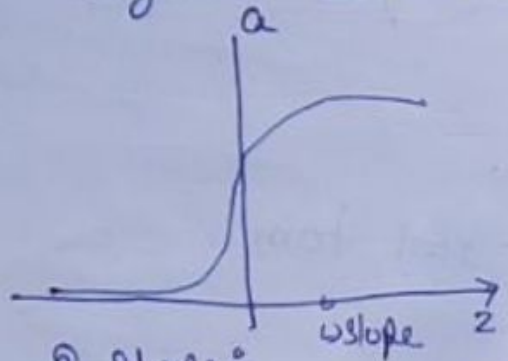
$$A^{[2]} = \sigma(Z^{[2]})$$

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

(5) ACTIVATION FUNCTIONS

In this we'll see the choices of activation function and how to find the slope of it.

① Sigmoid activation function



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

② Slope:

Slope of function in case of sigmoid activation function is

$$\frac{d}{dz} g(z) = \text{Slope of } g(z) \text{ at } z$$

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

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

* If z is very large z=10

z=10 $g(z) \approx 1$ ($g(z)$ close to 1)

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

* If z=-10

When $g(z) \approx 0$ (close to 0)

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

$\approx 0 \cdot (1-0) \approx 0$

* If z=0

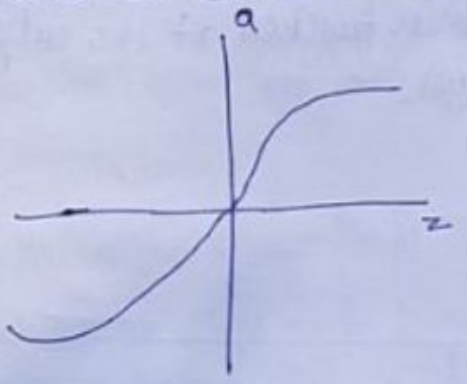
$g(z) = \frac{1}{2}$

$$\frac{d}{dt} g(t) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$$

Note: An Calculus instead of writing $\frac{d}{dt} g(z)$ the shorthand for the derivative is a prime of z. which is written as $g'(t) = \frac{d}{dt} g(t)$

An neural network $a = g(t) = \frac{1}{1+e^{-t}}$

Tanh activation function



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

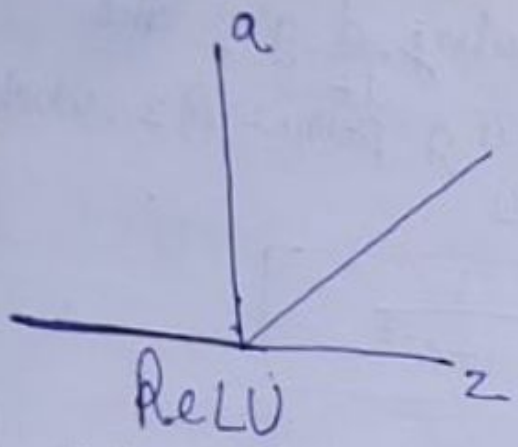
$\frac{d}{dt} g(t)$ = Slope of $g(z)$ at particular point at z

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

* At $t = 10$ $\tanh(t) \approx 1$
  $g'(z) \approx 0$
* $t = -10$ $\tanh(t) \approx -1$
  $g'(z) \approx 0$
* $t = 0$ $\tanh(z) = 0$
  $g'(z) = 1$

Note: $a = g(t)$, $g'(t) = 1 - a^2$

ReLU and Leaky ReLU



$g(z) = Max(0, z)$

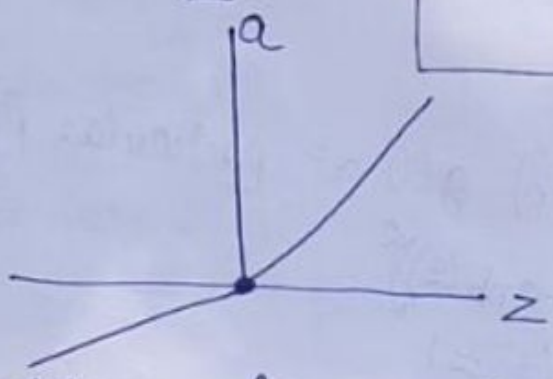
$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Remaining 6 is Here

There is only one scenario where we tend to use a linear activation like price of house.

If we use a sigmoid function the output will range from (0,1) But the price will be more than 1 also. In this case we'll use a linear function at the output layer.

Leaky Relu



$g(z) = max(0.001z, z)$

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

(6) Why do we need non-linear functions?

If we use linear activation function on output layer it will compute the output as a linear function of input features. It will be calculated as: $z = Wx + b$

also the output will be equal to z that is $A = z$

using linear activation is pointless. The composition of 2 linear functions is itself a linear function

⑧ GRADIENT DESCENT FOR NN

with one hidden layer

Note: ⑦ is on last page

NN parameters (with single hidden layer)

$n^{[0]} = n_x$

$n^{[1]}$ = No. of hidden layer

$n^{[2]}$ = No. of Output Neurons = 1

W_1 = shape is $(n^{[1]}, n^{[0]})$

b_1 = shape as $(n^{[1]}, 1)$

$W_2 = (n^{[2]}, n^{[1]})$, b_2 = shape $(n^{[2]}, 1)$

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

$\hat{y}$ → predict $a^{[2]}$

To train the parameters of algorithm you need to perform gradient descent

Gradient descent steps can be summarize as

① Compute predictions $(\hat{y}^{(i)}, i = 1 \dots m)$ ② Get derivatives: $dw^{[1]}, db^{[1]}, dw^{[2]}, db^{[2]}$

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db = \frac{dJ}{db}$ (we partial derivatives here)

③ Update $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$

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

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

Lets look at the forward & backward propagation

Forward Propagation

$Z^{[1]} = W^{[1]} * A^{[0]} + b^{[1]}$ # $A^{[0]}$ is X

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

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

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

Backward propagation

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

$dw^{[2]} = (dZ^{[2]} * A^{[1]}.T)/m$

$db^{[2]} = \text{Sum}(dZ^{[2]})/m$

$dZ^{[1]} = (W^{[2]}.T * dZ^{[2]}) * g^{[1]}{}'(Z^{[1]})$ # element wise product

$dw^{[1]} = (dZ^{[1]} * A^{[0]}.T)/m$ # $A^{[0]} = X$

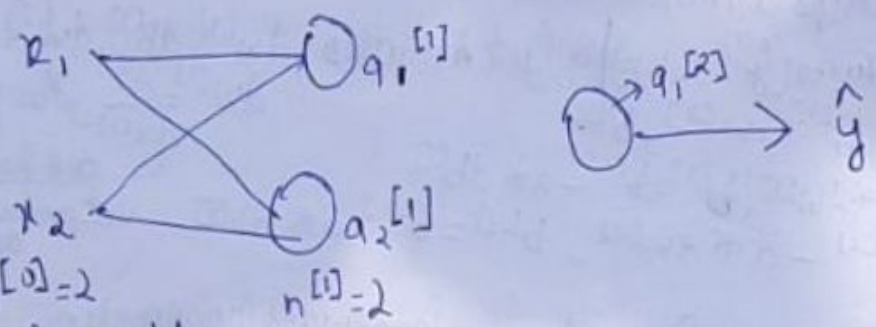
$db^{[1]} = \text{Sum}(dZ^{[1]})/m$

Watch video for proper explanation

⑨ RANDOM INITIALIZATION

When you change your Neural Network, it's important to initialize the weight randomly. For logistic regression, it was okay to initialize the weight to zero.

What happens if you initialize weights to zero



$n^{[0]}=2$ $n^{[1]}=2$

(Only 2 variable output)

Suppose we initialise w and b to 0.

$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]}$$

If we initialize w at 0 then $a_1^{[1]} = a_2^{[1]}$

$$dz_1^{(1)} = dz_1^{[1]}$$

If we initialize w at 0 then $dz_1^{[1]} = dz_2^{[1]}$

The solution for this is to initialize your parameters randomly

If we initialize w and b at 0 then both nodes will give you same output. and all of the nodes will be symetric.

$w^{[1]} = np.random.randn((2,2)) * 0.01$
↳ multiplies by small no.

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

$w^{[1]} = \ldots$

$b^{[1]} = \ldots$

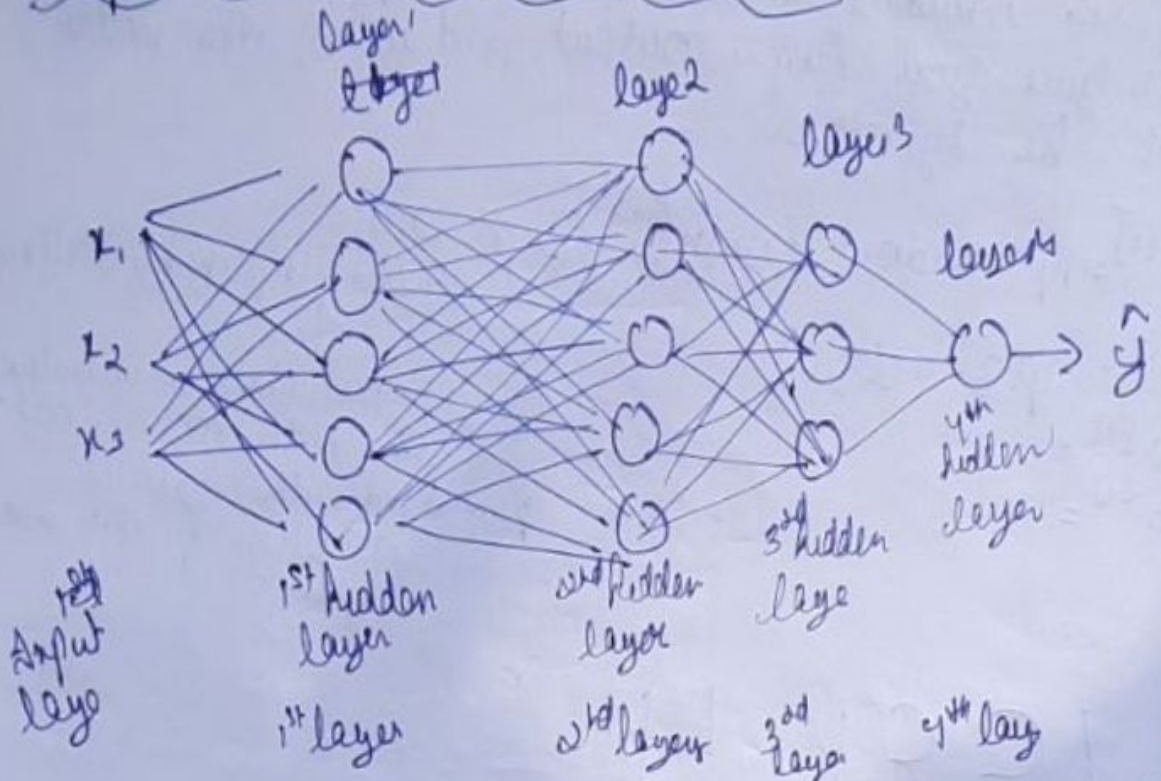
↓
But why don't we multiply this with 100 or 200?
Because this will make of activation function more complex.

WEEK 4

① Deep L-layer NN.

* we can have any number of hidden layer in our model
* when we don't have any hidden layer in our model it is known as shallow NN.
* when we have 1 hidden layer it is called 2 layer NN (1 hidden layer + output layer)
* when we have many hidden layers it is called deep NN

Deep Neural Network notation

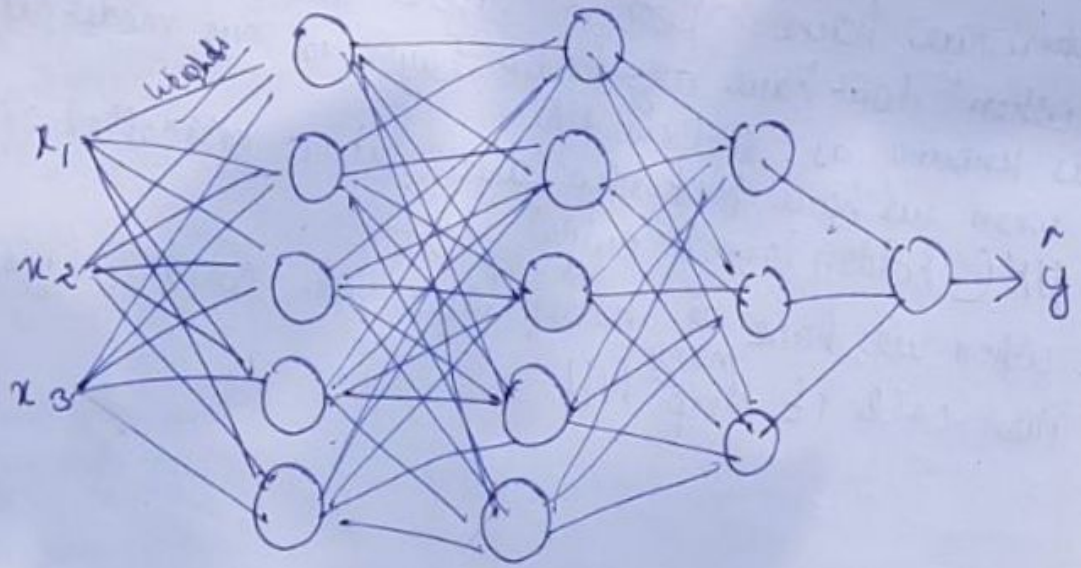


No. of layer = 4 (# layer)

$n^{[l]}$ = # number of units in layer (l) $n^{[1]} = 5,\ n^{[2]} = 5,\ n^{[3]} = 3,\ n^{[4]} = 1$

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

② FORWARD PROPAGATION IN A DEEP NETWORK



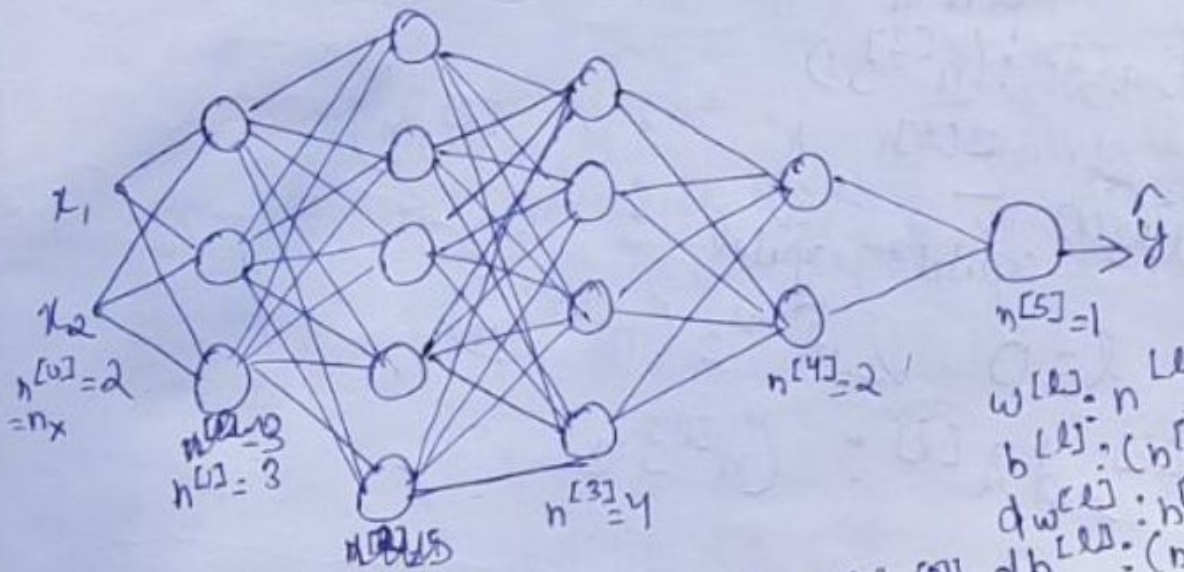
$x: z^{[1]} = w^{[1]} x + b^{[1]}$
$a^{[1]} = g^{[1]}(z^{[1]})$] → for first layer

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

= this we have to use for every layer

③ GETTING YOUR MATRIX DIMENSIONS RIGHT

l = no. of layer = 5



$w^{[l]} : n^{[l]}, n^{[l-1]}$
$b^{[l]} : (n^{[l]}, 1)$
$dw^{[l]} : n^{[l]}, n^{[l-1]}$
$db^{[l]} : (n^{[l]}, 1)$

$z^{[1]} = w^{[1]} \cdot x + b^{[1]}$
$(3,1)$ $(3,2)$ $(2,1)$ $(3,1)$
$(n^{[1]}, 1)$ $(n^{[1]}, n^{[0]})$ $n^{[0]}, 1$

$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$

$w^{[1]} : [n^{[1]}, n^{[0]}]$
$w^{[2]} : (5,3)$ $(n^{[2]}, n^{[1]}]$
$z^{[2]} = w^{[2]} \cdot x^{[1]} + b^{[2]}$
$(5,1)$ $(5,3)$ $(3,1)$ $(5,1)$

↳ ek pichey layer ka chalta hai yeah

$w^{[3]} : (4,5)$
$w^{[4]} : (2,4)$

Vectorized implementation

Same diagram

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

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

$[z^{[1](1)} \; z^{[1](2)} \ldots]$

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

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

$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

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

capital $Z^{[l]}$ capital $A^{[l]}$

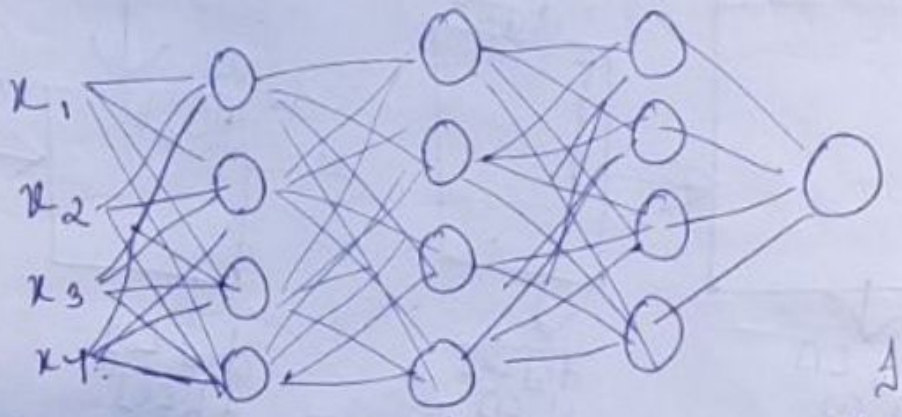
$l = 0 \quad A^{[0]} = (n^{[0]}, m)$

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

④ WHY DEEP REPRESENTATION

↳ used in speech recognition
↳ used in ~~face~~ object detection
↳ used in audio files

⑤ BUILDING BLOCKS OF DEEP N.N.



In this we cache the value of $z^{[l]}$ which will be useful in back-propagation

For layer l (any layer)
you have some parameter $w^{[l]}$, $b^{[l]}$
For forward prop you ~~will~~ will input
Forward: $a^{[l-1]}$ output $a^{[l]}$
↳ activation from previous layer

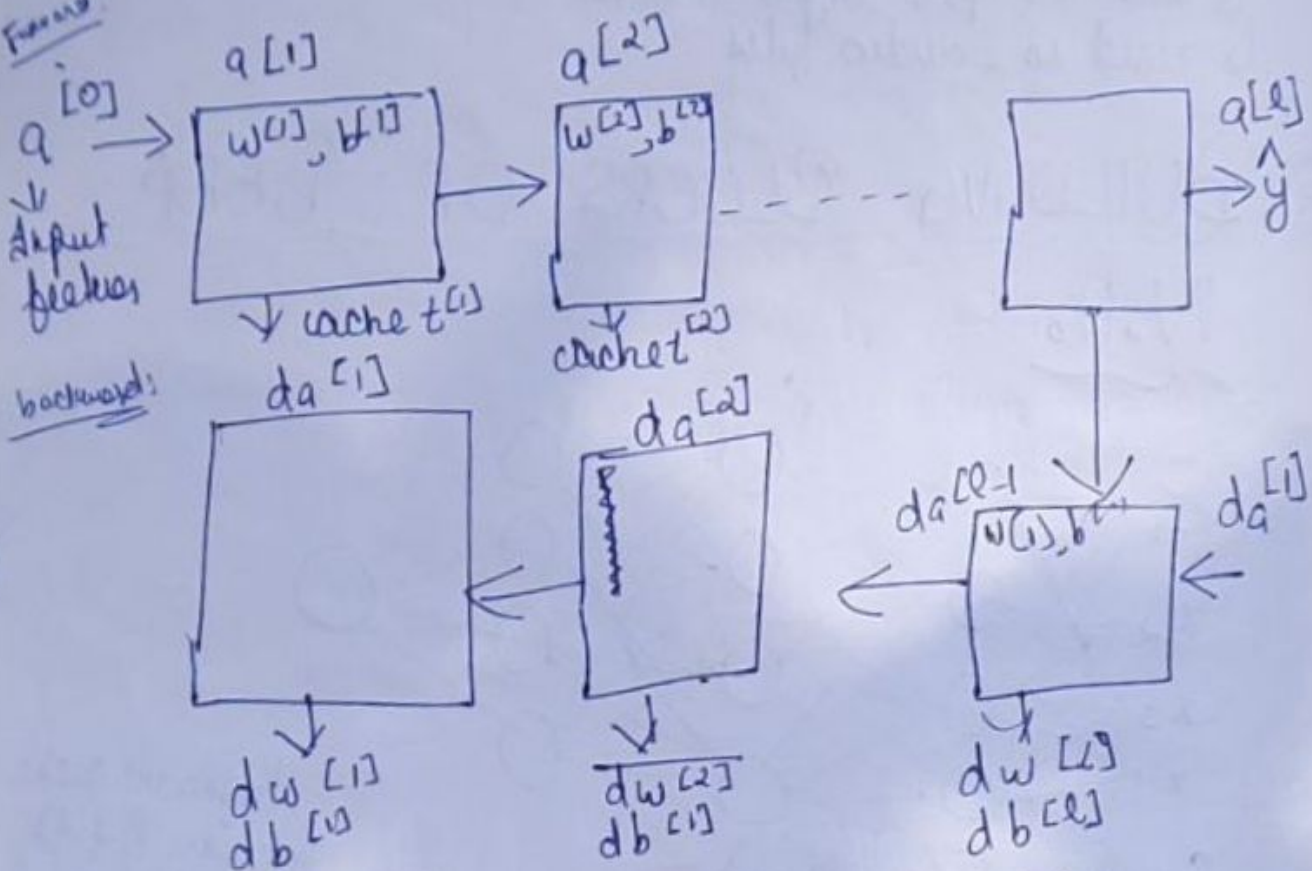
$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

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

↳ that's how you go from the input $a^{(l-1)}$ to the output $a^{[l]}$

Backward: Input $da^{[l]}$, output $da^{[l-1]}$, $dw^{[l]}$

Forward and backward functions



(6) FORWARD AND BACKWARD FUNCTIONS

Cache = values are saved of w and b in this we for back propagation

it just save value we can use to save any value like variable

Forward propagation for layer l

→ input $a^{[l-1]}$

→ output $a^{[l]}$, & cache ($z^{[l]}$ → it include $w^{[l]}, b^{[l]}$ as well

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

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

and if you want vectorize implementation of this.

$Z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]}$

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

Backward Propagation for layer l

⇒ Input $da^{[l]}$

⇒ Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]}(z^{[l]})$

$dw^{[l]} = dz^{[l]} \cdot (a^{[l-1]})^T$

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

$da^{[l-1]} = w^{[l]T} * dz^{[l]}$

Vectorize version

$dZ^{[l]} = dA^{[l]} * g^{[l]}(Z^{[l]})$

$dw^{[l]} = \frac{1}{m} dz^{[l]} A^{[l-1]T}$

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

$dA^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$

⑥ PARAMETERS VS HYPERPARAMETERS

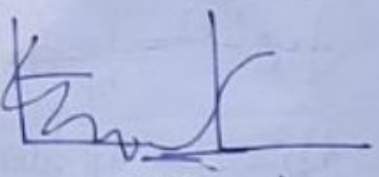
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \ldots$

Hyperparameters: learning rate (α)
Iterations
hidden layer L
hidden units $n^{[1]}, n^{[2]}$
choice of activation function

WEEK 3

⑦ DERIVATIVES OF ACTIVATION FUNCTION

For Sigmoid activation function.



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

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

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

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

Thus can also be written as $g'(z)$

Thus can also written as $a(1-a)$

Sanity test for Sigmoid about this expression makes sense:

If $z = 10$ $g(z) \approx 1$ ($\approx$ close to)

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

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

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

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

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

For Tanh function

$g(z) = \tanh(z)$

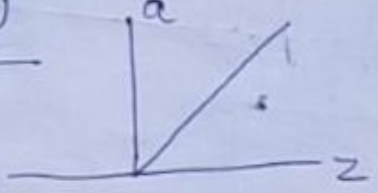
$g'(z) = \frac{d}{dz} g(z)$ = Slope of $g(z)$ at z

$= 1 - (\tanh(z))^2$ → can also be written as $1 - a^2$

If $z = 10$ $\tanh \approx 1$	If $z = -10$ $\tanh \approx -1$	If $z = 0$ $\tanh = 0$
$g'(z) \approx 0$	$g'(z) \approx 0$	$g'(z) = 1$

ReLU and Leaky Relu

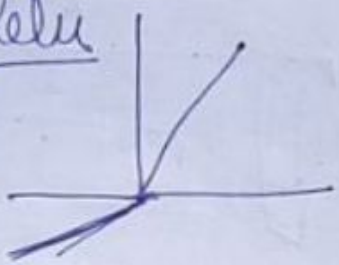
ReLU



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

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

Leaky Relu



$g(z) = \max(0.001z, z)$

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

#2 IMPROVING DEEP N.N.

HYPERPARAMETER TUNING, REGULARI-ZATION AND OPTIMIZATION

WEEK 1

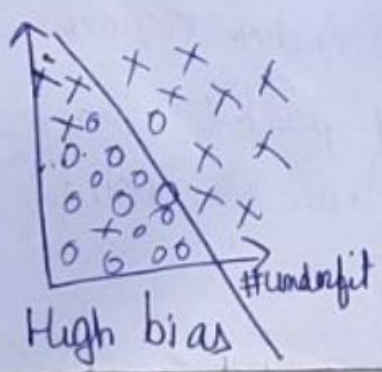
① Train / Dev / Test sets
- Train → Training set
- Dev → development set (for validation)
- Test → Test dataset

Applied ML is a highly interative process
we can change:
1. layes
2. learning rate
3. activation function
and many more

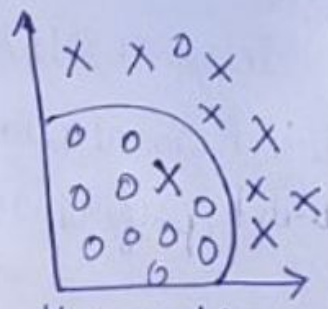
Note: the testing data must be from the same dataset

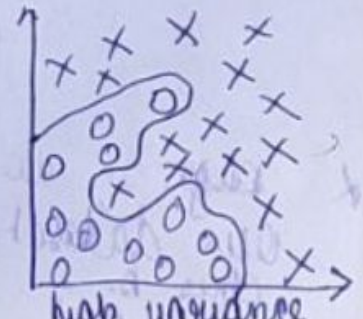
② Bias / Variance

In this we'll learn Bias and variance tradeoff



High bias


Just right


high variance
overfitting

Bias and Variance

Cat classification

	y=1	y=0		
	Cat picture	Dog Pictur		
Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	high variance	high Bias	high bias and high variance	low bias and low variance

* When our model overfits some parts of the data like this

Then it has both high bias & variance

③ BASIC RECIPE FOR M.L.

* If we have high bias in our data (training data performance) we can solve this by:-
  1. Bigger Network
  2. Train longer
  3. Or trying different NN. architech structure
* ~~high~~ If we have high variance problem in our dataset (dev-set performance) we can solve this by:-

① More data
② Regularization
③ NN. architect structure

* If we want to decrease Bias in our data this will leads to little increase in Variance & if we want to decrease Variance in our data this will leads to little increase in Bias this is called Tradeoff b/w bias & Variance. We have to create a proper balance b/w Bias & Variance.

④ REGULARIZATION

→ If our NN has overfitting problem (high variance) then we should try Regularization

Let's take the example of logestic regression. We try to minimize the loss function

$$\min_{w,b} J(w,b)$$

get more training data for high variance but sometimes it's not possible & so then we use regularization

This is how our cost function will look like if we add regularization -

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

↳ norm of w squared

$$L_2 \text{ regularization} = \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$$

norm is also known as Frobenius norm

If we add $\|w\|$ in our cost function it is called L2 regularization and if we add $\|b\|$ in our cost function it is called L1 regularization

We generally use L2 regularization because it is related to weights and weights has higher dimension then bias so generally we use L2 regularization

Note: We can also use L1 & L2 regularization together

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \frac{\lambda}{2m} b^2$$

Note: we call this as $\|w\|_2^2$ as norm of w squared

So, why don't you regularize just the parameter w? why don't we add b as well like this

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|_2^2 + \frac{\lambda}{2m} b^2$$

because w is high dimension parameter and b is not that will give us ~~not~~ much more difference

* We can also use L1 regularizat So what is L1 regulization?

L1 regularution = when we add $\frac{\lambda}{2m}\sum_{i=1}^{n_x} |w_i| =$

$\frac{\lambda}{2m}\|w\|_1$ in our cost function. when we use L1 regularuktion this will has lots of zeros which will helps in compressing our model

So, L2 is used generally

* λ (lambda) in regularuzation is called regularuzation parameter. and lambda is another hyper parameter that you might have to tune

Let's add a regularuzation term to our cost functo this is how it will look like now

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{[i]}\right) + \frac{\lambda}{2m}\sum_{l=1}^{l} \|w^{[l]}\|^2$$

$$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} \left(w^{[l]}\right)^2 \qquad w : (n^{[l]}, n^{[l-1]})$$

let's see how regularization works for a gradient descent algorithm:

previously we complete dw using backprop

$dw = (\text{from backprop})$ [It gives the value of $\frac{dJ}{dw^{[l]}}$]

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

But this is before we added the extra regularization

$dw^{[l]} = (\text{from Backprop}) + \frac{\lambda}{m} w^{[l]}$

$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]} + \frac{\lambda}{m} w^{[l]}$

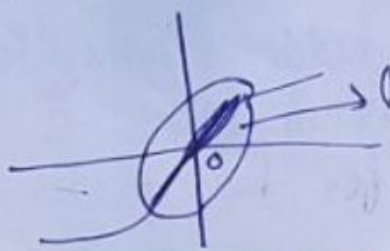
L2 regularization is also called weight decay'

$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right]$

$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha(\text{from backprop})$

(5) WHY REGULARIZATION REDUCE OVERFITTING

If the regularization becomes very large, the parameter W very small, so z will be relatively small (ignore B for now). So z will take small range of value and so the activation function if it is tanh will be relatively linear and your whole N.N. will be computing something not too far from a big linear function, which is very much less useful

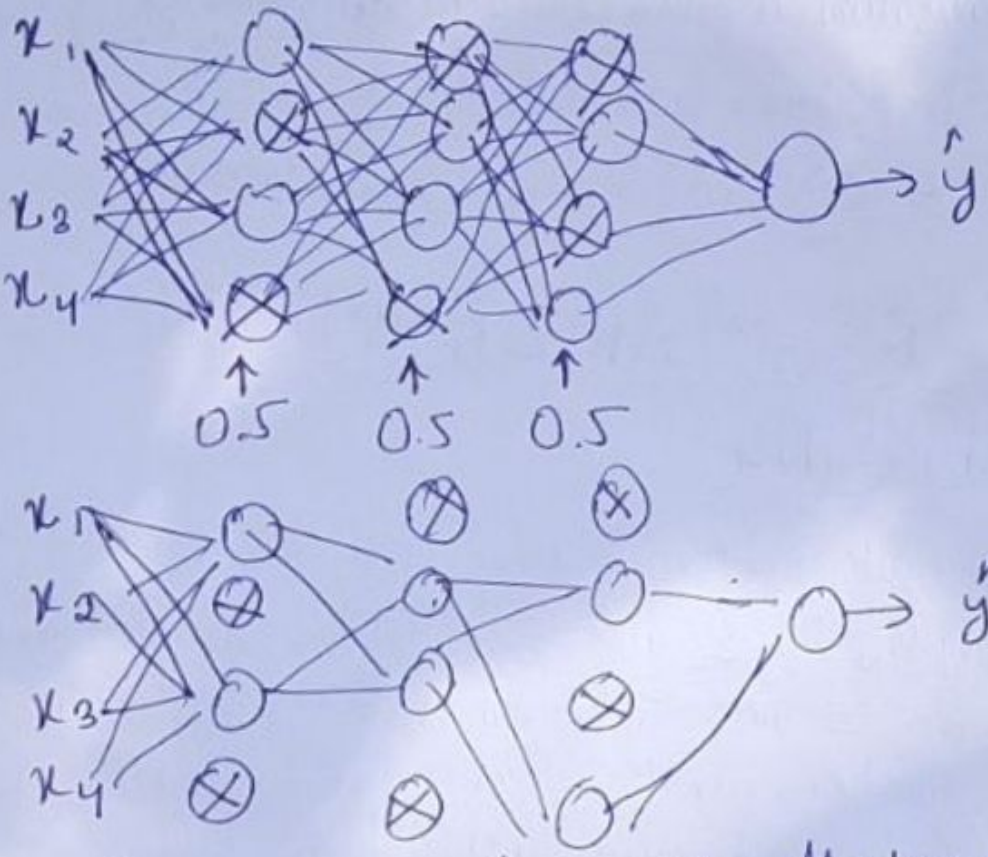


How? Because these values of $z^{[l]}$ will be near the origin

(6) DROPOUT REGULARIZATION

This is a very powerful regularization technique called dropout.

Suppose this neural network is overfitting on the training data. Suppose we add a dropout of 0.5 to all these images. The model will randomly remove 50% of the units from each layer and we finally end up with a much simpler network.



This has proven to be a very effective regularization technique. How can we implement lets check

* Suppose we we have 3 hidden layers. For now, we will consider the third layer. l=3
The dropout vector d for third hidden layer

can be written as:

d^3 = dropout vector for the layer 3

d^3 = np.random.rand(a^3.shape[0], a^3.shape[1]) < keep_prob

Here, keep_prob is the probability of keeping a unit
So, if keep_prob = 0.8 then this means that there is a 0.2 chance of elimination an hidden unit.

$a_3 = a_3$ has the activation that you computed

a_3 = np.multiply(a^3, d_3)

this a^3 value will be reduced by a factor of (1 - keep prob).
So to get the expected value of a^3, we divide the value

a^3 /= keep_dim

example:-

No. of ~~unt~~ units in the layer = 50

keep_prob = 0.8

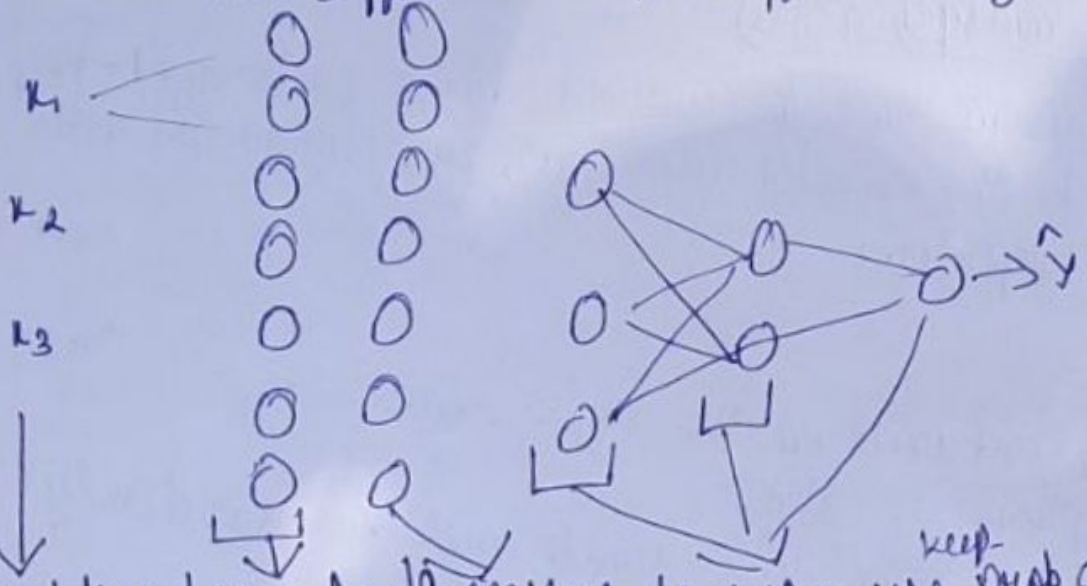
So, 20% of the total units will be randomly shut off.

Note that dropout is only done at the time of training the model (not during the test phase). The reason for doing this is because:

- We don't want our output to be random.
- Dropout adds noise to the prediction.

(7) UNDERSTANDING DROPOUT

* When we use dropout it just knock neurons in a hidden layer randomly.
* If we have more nodes in hidden layer, then we can use knock more neuron using Dropout
* following is the example of different knockoff value for different layer of N.N.
* we can set different value for different layer



For input layer keep-prob should be higher because generally we don't want our data to dropout easily.

We can have keep-prob higher or lower because we have many nodes here suppose our keep-prob = 0.7 then dropout value = 0.3 30% of the nodes will drop out

Suppose we have value of keep-prob is 0.5 that means that 50% of the neuron can or will drop

We can use keep-prob at 1 also here no neuron will be dropped.

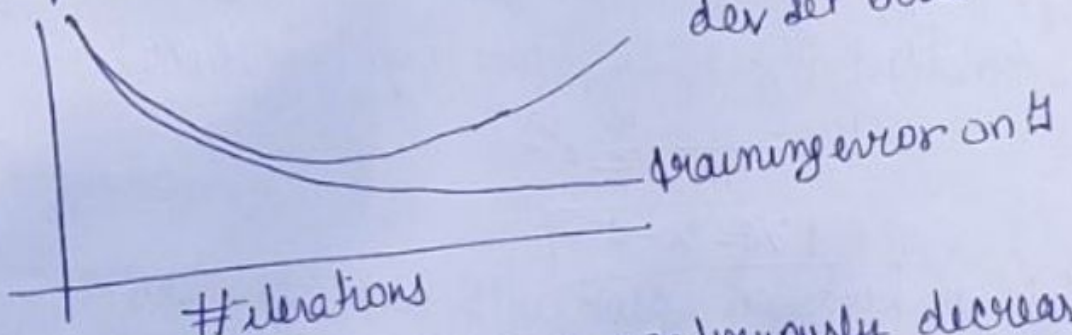
* We don't use this when our model is not overfitted (High bias or high Variance)

(8) OTHER REGULARIZATION METHOD

(1) data Augmentation: Suppose we are building an image classification model and are lacking with the training data. In such cases, we can use data augmentation i.e applying some changes in the image by flipping, random crops, rotating or adding distortions in the image etc. These can help in getting more training data and helps in reducing overfitting.

(2) Early Stopping:

example -



Here, the training error is continuously decreasing with respect to time. On the other hand, the dev set error is decreasing initially before increasing after a few iterations. We can stop the training at the point where the dev set error start to increase. This is called early stopping.

~~Setting up your optimization problem~~

~~① MINI-BATCH GRADIENT DESCENT~~

⑨ NORMALIZING INPUTS → It helps in speeding up our NN

Here we have 2 input features x_1 & x_2

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

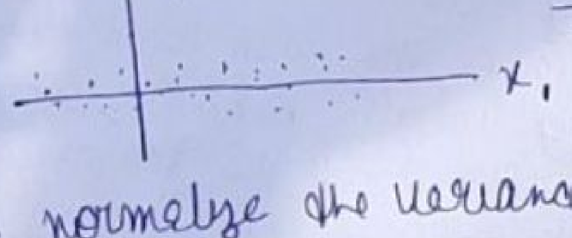
Following are the steps for normalization

1. Subtract the mean from input features:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\boxed{x := x - \mu}$$

This is how our plot will look after this



→ Here x_1 has the large variance than x_2

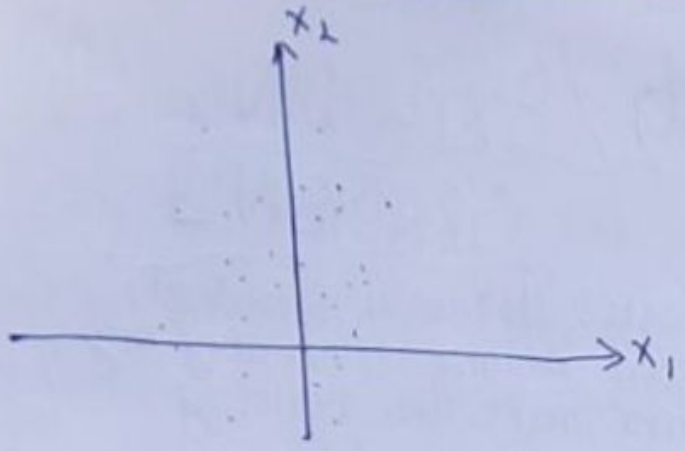
② Next we normalize the variance

Here, feature x_1 has larger variance than x_2

$$\sigma = \frac{1}{m} \sum x^{(i)} {**2}$$ → This is elementwise squaring → square (thus multiplication)

we divide the feature with the variance

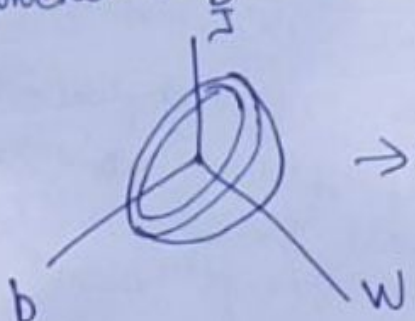
This is how our input looks like this after that



The same mean and variance to normalize the test set as well. because we want the same transformation to happen on both the train & test data.

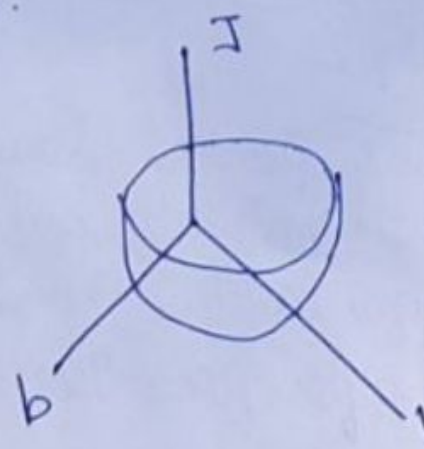
How does normalizing the data make the algorithm faster?

An unnormalized data, the scale of features will vary and hence there will be variation in the parameters learnt from each features. This will make the cost function asymmetric



→ this is asymmetric

whereas in the case of normalized data, the scale will be the same & cost function will also be symmetric.



Normalizing the inputs makes the cost function symmetric. This makes it easier for the gradient descent algorithm to find the global minima more quickly. This makes algorithm to run much faster.

(10) VANISHING / EXPLODING GRADIENTS

While training deep Neural Networks, sometimes the derivates (slopes) can become either very big or very small. It can make the training phase quite difficult. This is the problem of vanishing / exploding gradients. Suppose a N.N with i-layer with 2 input and we initialize the large weights:

$$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

The final output of l^{th} layer will be

$$\hat{y} = W^{[l]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{l-1} x$$

For deeper network, L will be large, making the gradient very large and the learning process slow. Similarly, using small weights make the gradients very small as a result learning will be slow. So how to deal with this problem? How to initialise weights?

(11) WEIGHT INITIALIZATION FOR DEEP NETWORK

It turns out that the partial solution to this doesn't solve it entirely but helps a lot is better or more careful choice of the random initialization for your Neural Network.

example:-

For Sigmoid

$z = w_1 x_1 + w_2 x_2 + \ldots w_n x_n + b$

larger → smaller w_i ; let $b = 0$

$Var(w_i) = \frac{1}{n}$

Note: We always want our weight to be smaller because it would do lots of multiplication

$w[l] = np.random.randn(shape) * np.sqrt\left(\frac{1}{n^{[l-1]}}\right)$

In case of Relu Action function:-

$Var(w_i) = \frac{2}{n}$

$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

? The primary reason behind initializing the weights randomly is to break symmetry. We want to make sure that different hidden units learns different patterns. There is one more technique that can help ensure that our implementation are current and will run quickly

(12) NUMERICAL APPROXIMATION OF GRADIENTS

When you implement Back propagation you'll find that there's a test called gradient checking that can really help you make sure that your implementation of back prop is correct.

WEEK 2

① MINI-BATCH GRADIENT DESCENT

what happens if we have a large training set say $m = 5,000,000$? if we train our model with all training examples ~~at~~ it will take lot of time. So Mini-Batch Gradient descent is the solution for this.

So, we split of data sets in mini-batch of upto of 1000 examples each it means for 1,000 examples we have to create 5,000 batches

This is how how our training data set will look:

$$X_{(n_x, m)} = [x^{(1)}\ x^{(2)}\ x^{(3)} \ldots x^{(1,000)} \mid x^{(1001)} \ldots \ldots x^{(2000)} \mid \ldots . x^{(m)}]$$

$x^{\{1\}}$ → 1st Batch

$x^{\{2\}}$

$x^{\{5,000\}}$

$$Y_{(1,m)} = [y^{(1)}\ y^{(2)}\ y^{(3)} \ldots y^{(1,000)} \mid y^{(1001)} \ldots \ldots y^{(2,000)} \ldots \mid \ldots y^{(m)}]$$

$y^{\{1\}}$ → 1st Batch → Y input

$y^{\{2\}}$

$y^{\{5,000\}}$

Here $X^{\{3\}}$, $Y^{\{3\}}$ represents 3rd mini-batch input & output.

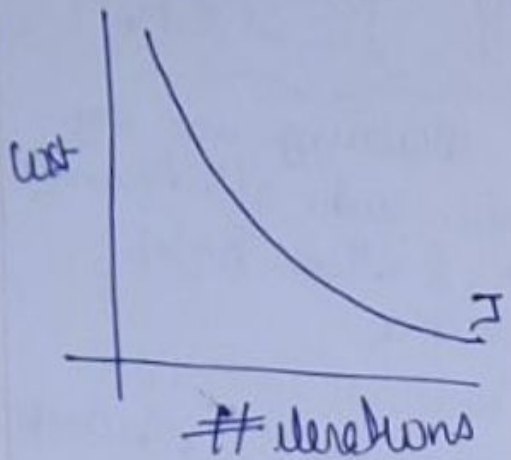
② UNDERSTANDING MINI-BATCH GRADIENT DESCENT

Batch

→

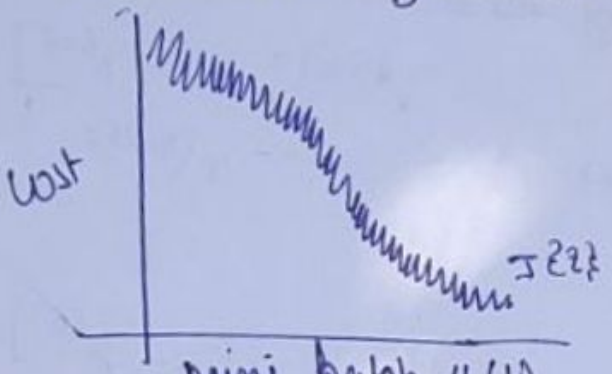
Training with mini batch gradient descent

Batch gradient descent



with batch gradient descent on every iteration you go through the entire training set and you'd expect the cost to go down on every single iteration.

Mini-batch gradient descent



On mini batch gradient descent if you plot progress on your cost function, then it will not decrease on every iteration.

If we plot cost function $J^{\{t\}}$ using $X^{\{t\}}, Y^{\{t\}}$ (This is how our plot will looks like)

It tends to go downwards but it is going to be noisy

Reason: In mini-batch gradient descent is noisy because maybe there there are mislabel in 1st batch and easy fel on second

Choosing your mini-batch size

① If your minibatch size = m : Then it is called batch gradient descent
$(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

minibatch = size of training rows

② If your mini-batch size = 1 : This is called Stochastic gradient descent

each training example is its own mini-batch

It can be extremely noisy and takes more time to reach global minima.

3. If the mini-batch size is Between 1 to m:
It is mini-batch gradient descent. The size of the mini-batch should not be too large or too small.

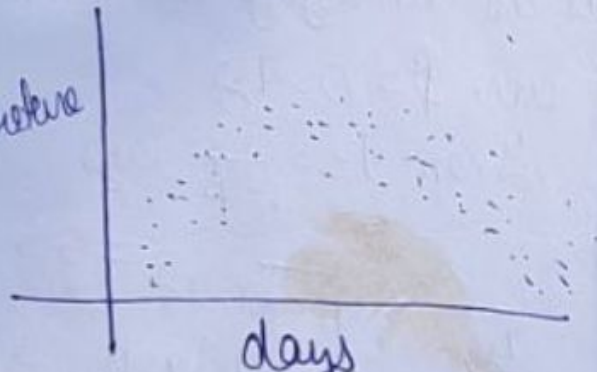
Below are a few general guidelines to keep in mind while deciding the mini batch size:

1. If the training set is small, we can choose a mini-batch ~~size~~ size of m <2000.
2. For a larger training set typical mini-batch size are 64, 128, 256, 512
   ↳ these are define by power of 2 for example $2^6, 2^7, 2^8, 2^9$ and it is to make sure that the mini-batch size fits your GPU/CPU memory

(3) EXPONENTIALLY WEIGHTED AVERAGE

following are the daily temperature of London 2011

$\theta_1 = 40F$ (1st day of year)
$\theta_2 = 49°F$
$\theta_3 = 45°F$
⋮
$\theta_{180} = 60°F$
$\theta_{181} = 56°F$ (181st day of year)
⋮
$\theta_{365} = ?$



when we plot this we'll get this graph.

Exponentially weighted averages are used by many optimization algorithms.

If you want to compute the trends the local average or a moving average of the temperature

Lets initialize $V_0 = 0$

Then on every day $V_1 = 0.9V_0 + 0.1\theta_1$ [θ_1 = that day temperature]

$V_2 = 0.9V_1 + 0.1\theta_2$ (2nd day temperature)

$V_3 = 0.9V_2 + 0.1\theta_3$

If we plot this we'll get

you'll get a moving average of what's called an exponentially weighted average of the daily temperature

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

θ_t → temperature of that day

V_{t-1} → V from the previous day

$\boxed{\beta = 0.9}$

V_t = approximately average over $\frac{1}{1-\beta}$ days temperature

So, for example when beta goes 0.9 you could think of this as averaging over the last 10 days temperature.

* If our $\beta = 0.98$

Then $V_t = \frac{1}{1-\beta}$ days

$= \frac{1}{1-0.98} = 50$

$\therefore$ × 50 days So this is averaging over roughly the last 50 days temperature

A high value of Beta the plot you'll get is much smoother because you're now averaging over more days of temperature

(4) UNDERSTANDING WEIGHTED AVERAGE

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$$V_{100} = 0.9 V_{99} + 0.1 \theta_{99}$$
$$V_{98} = 0.9 V_{97} + 0.1 \theta_{98}$$

$$V_{100} = 0.1\theta_{100} + 0.9\theta_{99}(0.1\theta_{99} + 0.9\theta_{98}$$ → again same form

This is how V_{100} is calculated

(5) BIASCORRECTION IN EXPONENTIALLY WEIGHTED AVERAGE

There is one technical detail called biased correction that can make your computation of these averages more accurately.

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

when you're implementing a moving average you initialise it with $V0 = 0$

$V_0 = 0$
$V_1 = 0.98 V_0 + 0.002\theta_1$ (if $\theta_1 = 40$)
$V_1 = \beta(V_0 - 0)$

It is not a good estimate of the first day temperature

$V_2 = 0.98 V_1 + 0.02\theta_2$
$= \ldots$

It is not a good estimate for the initial phase of your estimate.

So, here instead of using the previous equation i.e.,

$$V_t = \beta * V_{(t-1)} + (1-\beta) * \theta_t$$

we include a bias correction term:

$$V_t = [\beta * V_{(t-1)} + (1-\beta) * \theta_t] / (1-\beta_t)$$

(divide)

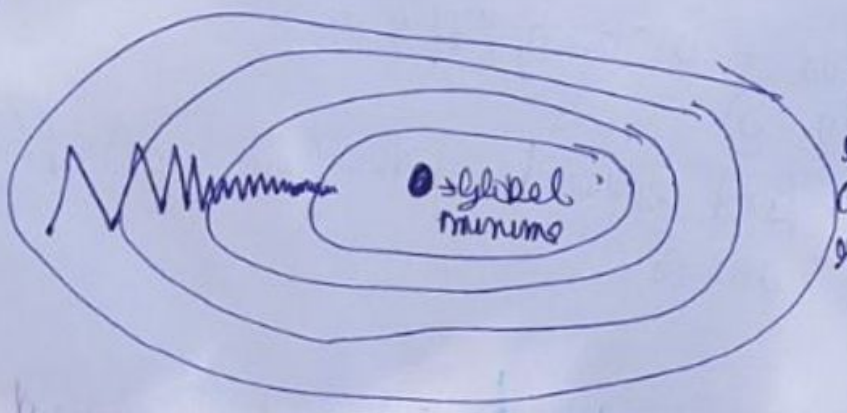
When t is small, β_t will be large, resulting in a smaller value of $(1-\beta_t)$. This will make the V_t value large which ensures that our predictions are accurate.

⑥ GRADIENT DESCENT WITH MOMENTUM

There's an algorithm called momentum, or gradient descent with momentum that almost always works faster than the standard gradient descent algorithm.

In one sentence, the basic idea is to compute an exponentially weighted average of your gradients, and then use that gradient to update your weights instead

The idea of gradient descent with momentum is to calculate the exponential weighted average of gradients and use them to update weights. Suppose we have a cost function looks like this



Here we want to reach global minima and we want to reach that part.

One more way could be to use a larger learning rate But that could result in large upgrade steps, and we might not reach global minima. Also too small a learning rate makes the gradient descent slower.

& We want a slower learning in the vertical direction and a faster learning in the horizontal direction which will help us to reach the global minima much faster

Compute dw, dB on current mini batch using momentum

$$Vdw = \beta * Vdw + (1-\beta) * dw$$

$$Vdb = \beta * Vdb + (1-\beta) * db$$

Update weights

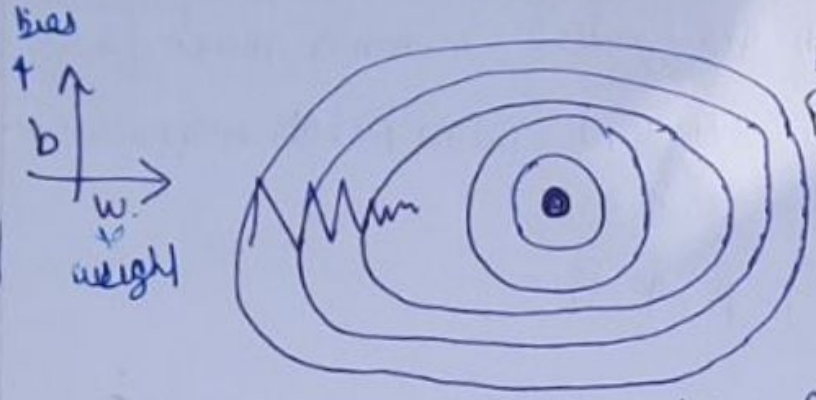
$$w = w - \alpha * Vdw$$

$$b = b - \alpha * Vdb$$

Here we have two hyperparameters i.e, α & β. The role of dw and db in the above equation is to provide momentum Vdw and Vdb provides velocity, and β acts as friction and prevents speeding over the limit. Consider a ball rolling down - Vdw and Vdb provide velocity to the ball and make it move faster. We do not want our ball to speed up so much that it misses the global minima and hence β acts as friction

⑦ RMSPROP

RMSPROP means root mean Square Prop that can also Speed up gradient descent.



the vertical axis is the parameter b and horizontal axis is the parameter w.

and we want to slow down the learning in b direction or vertical direction and speed up the to Horizontal direction.

On iteration t:

Compute dw, db / on current mini-batch ← previous value

this is element wise Squaring operation

$S_{dw} = \beta_2 * S_{dw} + (1-\beta_2) * dw^2$ ← small → large

$V_{db} = \beta_2 * S_{db} + (1-\beta_2) * db^2$ ← large

S_{dw} is relatively small than V_{db}

Update weights

$W = W - \alpha * (dW / \sqrt{S_{dw}})$

$b = b - \alpha * (db / \sqrt{S_{db}})$

when Vertical movement is done by b(bias) & Horizontal by w(bias)

Here we are dividing W with a small number and b with are large numbe to boost Horizontelly

The slope in the vertical direction (db) is steeper, resulting in a large value of S_{db}. As we want slow learning in the vertical direction, dividing db with S_{db} in update step will result in smaller change in b. Hence, learning in the vertical direction will be less. Similarly, a small value of S_{dw} will result in faster learning in the horizontal direction.

⑧ ADAM OPTIMIZATION ALGORITHM

Adam is combination of momentum and RMSprop

There are a range of hyperparameter used in adam and some of the common ones are:

- learning rate α : needs to be tuned
- Momentum term β_1 : common choice is 0.9
- RMS prop term β_2 : common choice is 0.999

Adam helps to train a NN model much more quickly than the techniques we have seen earlier

To implement Adam you would initialize $V_{dw}=0, S_{dw}=0$ and similarly $V_{db}=0$ & $S_{db}=0$. and then on iteration T you would compute the derivates: compute dw, db

① $V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw$ (momentum), $V_{db} = \beta_1 V_{db} + (1-\beta_1)db$

② $S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2)$

So ① is momentum like update with hyperparameter β_1

and ② is RMSprop with hyperparameter β_2

An the typical implementation of Adam, you do implement bias correction.

corrected means after bias correction

$$V_{dw}^{corrected} = V_{dw}/(1-\beta_1^t), \quad V_{db}^{corrected} = V_{db}/(1-\beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw}/(1-\beta_2^t), \quad S_{db}^{corrected} = S_{db}/(1-\beta_2^t)$$

$$w := w - \alpha * \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

ε is very small number Suppose of denominator become 0 then we'll don't get value of w: & b:. that's why we we ε (Epsilone)

This algorithm has a numbers of hyper parameters.

↳ given in starting

beta1 : - 0.9 is recommended

beta2 : - 0.999 is recommended

epsilon : 10^{-8} is n

M = Adaptive Moment Estimation

— (dw) — known as first movement.

— (dw) — movement.

⑨ LEARNING RATE DECAY

If we slowly reduce the learning rate over time, we might speed up the learning process. This process is called learning rate decay.

1 epoch ≈ 1 pass through the data
2 epoch = 2nd pass through the data

$$\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch\_num}} \alpha_0$$

α → learning rate

α_0 → Some initial learning rate alpha0 (α_0).

* decay_rate has become another hyperparameter which you might need to tune.

If $\alpha_0 = 0.02$
decay_rate = 1

Then

Epoch	α
1	$\frac{1}{1 + 1*1} \times \alpha_0 = 0.1$
2	0.67
3	0.5
4	0.4

Here learning rate is decreasing

There are also many formula to calculate learning rate decay

10. LOCAL OPTIMA IN N.N.

It refers to when we try to find the minimum value of our cost function we get stuck in a local optima means stuck in near minimum value of D. But this is not the minimum value for our cost function

WEEK 3

② HYPERPARAMETERS

① Tuning PROCESS

Following are the few hyperparameters :-

1. learning rate - α (most important)
2. Momentum - β
3. Adams Hyperparameter - β_1, β_2
4. Number of hidden layer
5. & learning rate decay
6. Mini-Batch size

② PICKING HYPERPARAMETERS AT RANDOM

(a) Hidden layer

You should have 2-4 hidden layer in your ~~mode~~ N.N.

(b) learning rate

α = can be low as 0.00001 ... to 1

↳ ~~0.0001~~ or any small number

BATCH NORMALIZATION

① NORMALIZING ACTIVATIONS IN A NETWORK

An deep N.N. we have lots of layers (hidden layers) which ~~as~~ has nodes ~~to~~ means loss of actuation

In this we normalize the mean an variance ~~in order~~ so that the weights of the next layer can be update faster

Quick Recap

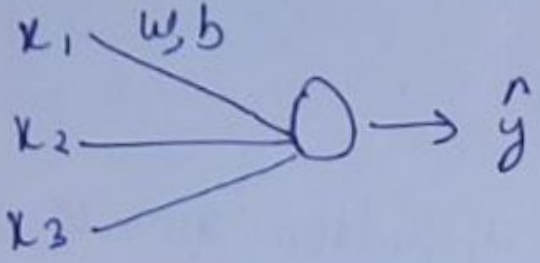
$$\mu = \frac{1}{m} \sum x^{(i)}$$

$$X = X - M$$

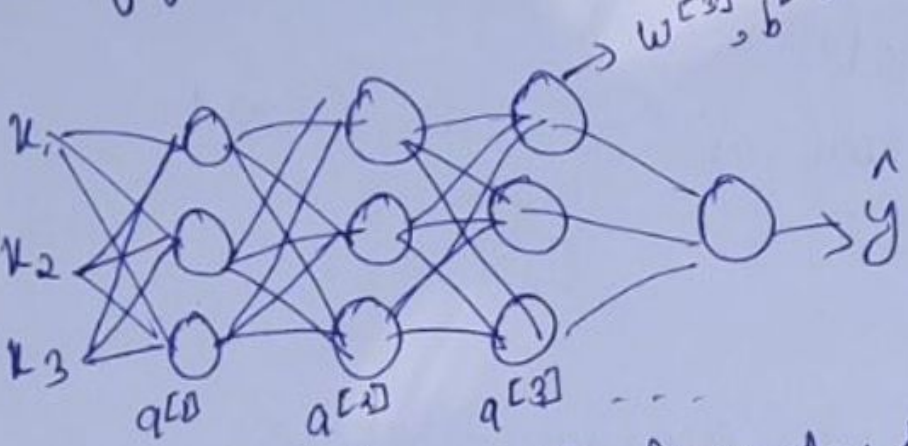
$$\sigma^2 = \frac{1}{m} \sum x^{(i)2} \rightarrow \text{elements by squaring}$$

$X = X / \sigma^2$

this work only with this type of the model



what if we have hidden layers in our model.



So, we want do learn the parameters of w3 and b3 (3rd layer w & b)

* We can normalize the mean and variance of $a^{[2]}$ to $a^{[2]}$ to make the learning of $w^{[3]}$ & $b^{[3]}$ more efficient

* By normalizing the $a^{[2]}$ layer w & b it will make $a^{[3]}$, $b^{[3]}$ work faster

$$\mu = \frac{1}{m} \sum z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - M}{\sqrt{\sigma^2 + \varepsilon}}$$

[Normalizing z value]

$$z^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

Here γ & β are learnable parameters

(2) FITTING BATCH NORM INTO A N.N.

(3) WHY DOES BATCH NORMALIZATION WORK?

- The first reason is the same reason why we normalize X
- The second reason is that batch normalization reduces the problem of input values changing (shifting)
- Batch normalization does some regularization:
  - Each mini batch is scaled by mean/variance computed of that mini-batch
  - This adds some noise to the values $Z^{[l]}$ within that mini batch. So similar to dropout it adds some noise to each hidden layers activation.

- This has a slight regularization effect. (4)
- Using bigger size of the mini-batch you are reducing noise and therefore regularization effect.
- Don't rely on batch normalization as a regularization. It's intended for normalization of hidden units, activations & therefore speeding up learning. For regularization use other regularization technique (L2 or dropout)

(4) BATCH NORMALIZATION AT TEST TIME

- When we train a N.N. with batch normalization, we compute the mean & variance of the mini-batch.
- In testing we might need to process examples at a time. Mean & variance of one example won't make sense.

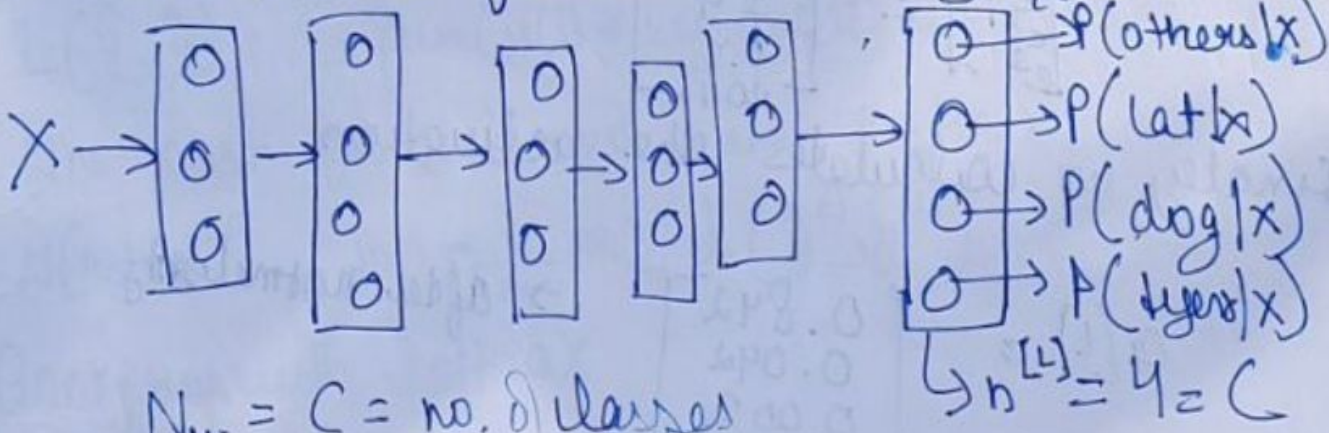
- We can use weighted average across the mini-batches
- We have to compute an estimated value of mean & variance to use it in testing time.
- We can use the weighted average across the mini-batches.
- We can use the weighted the weighted average across the mini-batches
- We will use the estimated value of the mean & variance, to test
- This method is called running average
- In practice we'll use deep learning framework and it will implement these thing by default

MULTI-CLASS CLASSIFICATION

① SOFTMAX REGRESSION

Binary classification means dealing with just 2 classes. But when we have more than one classification we use Softmax (multi class function). Suppose we have to recognise cats, dogs & tiger in set of images

At the output layer, instead of having a single unit, we have units equal to the total number of class which is 4 in our example not 3 because one class will represent none of them option Each unit tells us the probability of the image falling in different class. Since it tells the probability, the sum of value from unit is always equal to 1.



$N_y = C$ = no. of classes

In the last layer we will have to activate the Softmax activation function instead of the Sigmoid activation

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

$t = e^{(z^{[1]})}$ → t will be $(4,1)$ matrix

↓ temporary variable

output $= a^{[1]} = \frac{e^{z[1]}}{\sum_{i=1}^{4} t}$ (it is also $4,1$ vector)

- let's understand this with an example.
Consider the output from the last hidden layer:

$$z^{[1]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

- We then calculate t using the formula given above:

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$$

- Finally we calculate the activation

$$a[1] = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$ → after normalizing

(2) TRAINING A SOFTMAX CLASSIFIER

- There is an activation which is called hard max, which gets 1 for the maximum & zeros for others
- The Softmax is a generalization of logistic activation function to C classes. If C=2 Softmax reduces to logistic regression
- ~~The Softmax is generalization of logistic activation function to C classes~~
- The Softmax name came from softening the values & not hardning them like hard max

The loss function used with Softmax

$$L(\hat{y}, y) = -\text{Sum}(y(i) * \log(\hat{y}(i))) \# i = 0 \text{ to } C-1$$

The cost function used with Softmax.

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

Backprop with Softmax

$$dz^{[L]} = \hat{y} - y$$

derivate of Softmax

$$\hat{y} * (1 - \hat{y})$$

#3 COURSE-3

WEEK 1

INTRODUCTION TO ML STRATEGY

① WHY ML Strategy

Things we can do to improve ~~performance~~ accuracy of our NN:

1. Collect more data
2. Collect more training, test data
3. Train longer with Gradient descent
4. Try bigger, smaller Neetwork
5. L2 regularization
6. change network architecture
7. dropout

② ORTHOGONALIZATION

* Some deep learning developers knows exactly what parameter do tune in order to try to achieve one effect. It is called orthogonalization.

* 1] You'll have to fit training set well on cost function
  - If it's not achieved you could try bigger network, another optimization (like adam)

2] Fit dev set well on cost function.
- if its not achieved you could try regularization . bigger training etc.

iii) Fit test set well on cost function
- If its not achieved you can try bigger dev. set

iv) Performs well in real world
- If its not achieved you could try could try change dev. set change cost functions.

SETTING UP YOUR GOAL

1) SINGLE NUMBER EVALUATION METRIC

- Its better and faster to set a single number evaluation metric for your project before you start it
- what are precision and recall
  - Suppose we run the classifier on 10 images which are 5 cats & 5 non-cat. The classifier identifies that there are 4 cats, but it identified 1 wrong cat
  - Then the confusion matrix will looks like this

	Predicted cat	Predicted non-cat
Actual cat	3	2
Actual non-cat	1	4

  - Precision : Percentage of true cats in the recognized result : P = 3/(3+1)
  - Recall : Percentage of true recognition cat of the all cat predictions: R = 3/(3+2)
  - Accuracy : (3+4)/10

- Using precision/recall for evaluation is good in lot of cases, but separately they don't tell you which algorithm is better. But one this we can do is combining the precision and recall called F1 score.
- You can think of F1 score as an average of precision and recall

$$F1 = 2/((1/P) + (1/R))$$

(2) [illegible]

Classifier	Precision	Recall	F1 Score
A	95%	90%	92.4%
B	98%	85%	91.0%

(2) SATISFICING AND OPTIMIZING METRIC

Classifier	Accuracy (optimizing)	Running time (satisficing)
A	90%	80ms
B	92%	95ms
C	95%	1500ms

- Optimizing is the F1 score
- Satisficing is the running time of that particular accuracy. It has to be less than 100ms

iii) ③ TRAIN / DEV / TEST DISTRIBUTION

following are the guidelines for train/dev/test

IV) ① always choose a dev set and test set from a same distribution
② and reflect data you expect to get in the future and consider important to do well on.
③ Setting up the dev set, as well as the validation metric is really defining what target you want to aim at.

① ④ SIZE OF THE DEV AND TEST SETS

→ * Old way of splitting data

70% train	30% test

60% train	20% Dev	20% Test

* But in the modern machine learning era we are now used to working with much larger data set size. Suppose we 1,00,00,000 data example then we can use 98% to train our model, 1% to for development and 1% for training.

1% of 1,00,00,000 is 100000 which is quite high

5) WHEN TO CHANGE DEV/TEST SETS AND METRICS.

Example: (Cat classification)

Metric: classification error

Algo A: 3% error (But lot of porn images are treated as cat image here)

Algo B: 5% error

* we can easily say that our algo A is during By looking at error value but lot of porn images are also treated as cat images here

* Thus, in this case, we ~~need~~ want and need to change our metric

↳ we put less weight on images which are not porn

$w[i] = 1$ if $x[i]$ is not porn

↳ we will put more weight on images which are porn

$w[i] = \underbrace{10}_{\text{+ any big number}}$ if $x[i]$ is porn

* This is actually a example of an orthogonalization where you should take a Machine learning problem and break it into distinct steps:

i) Figure out how to define a metric that captures what you want to do - place that target

ii) Worry about how to actually do well on this metric - how to aim/shoot accurately at the target.

* If doing well on your metric + dev/test set doesn't correspond to doing well on in your application, change the metric or dev/test or both.

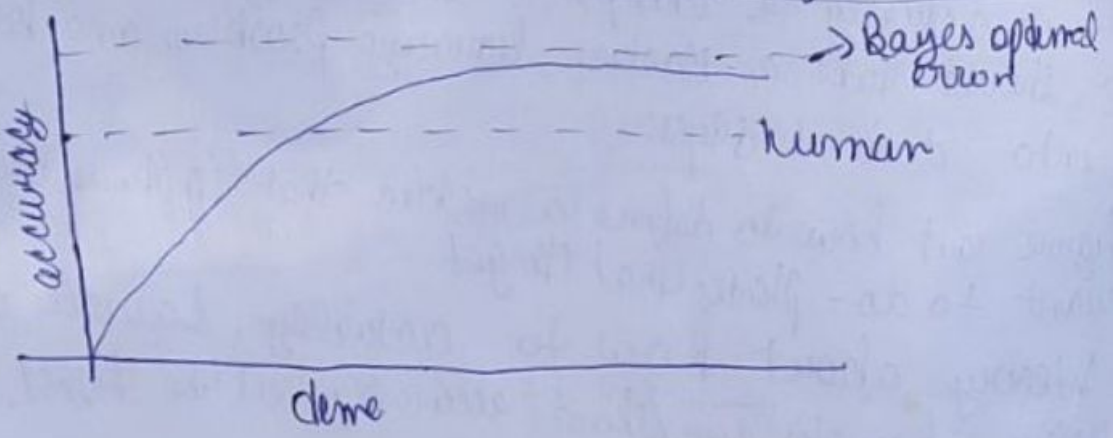
COMPARING TO HUMAN-LEVEL PERFORMANCE

① WHY HUMAN-LEVEL PERFORMANCE

Because of advances in deep learning, ML algorithms are suddenly working so much better and so it has become much more feasible in a lot of application areas for machine learning algorithms to actually become competitive with human level performance

At turns out that the workflow of designing and building a machine learning system is much more efficient when you're trying to do something that human can also do.

Comparing to human-level performance



explanation of above diagram :-

① The more we give time, the accuracy of an algorithm increases and it surpass human level performance. after that we surpass human level performance there is not that much head room to still improve.

② A line will never meet Bayes optimal error because suppose we have blurry picture whether it is a cat or not even our algorithm can't tell whether it is cat or not.

③ Humans are quite good at lot of tasks. So as long as ML is worse than humans, you can:
① Get labeled of images by humans
② Gain insight from manual error analysis: why did a person get it right?
③ Better analysis of bias/variance.

② AVOIDABLE BIAS

Suppose that cat classification algorithm gives these results:

Humans error	I^st 1%	II^nd 7.5
① Training error	8%	8%
② Dev error	10%	10%

* maybe we has 7.5% of human error & 8% training error & Dev error = 10%. Because of blurry images.

* suppose we have 1% human error on I^st we can easily say that our model is not doing good
* If we have 7.5% (II^nd) human error & 8% training error & Dev error = 10%, we can say that our model is doing great

* In the left example (1% human error) we have do focus on the bias
* In the right example (7.5 human error) then we have to focus on the variance.
* The human-level error is a proxy (estimate) for Bayes optimal error. Bayes optimal error is always less but human-level in most cases is not far from it.
* Also you can't do better than Bayes error unless you are overfitting
* Avoidable bias = Training error - Human error
* Variance = Dev error - Training error

(3) UNDERSTANDING HUMAN-LEVEL PERFORMANCE

* We have multiple human-level performance based on the human experience then you choose the human-level performance (proxy for Bayes error) that is more suitable for the system you're trying to build.
* Human-level error (proxy for Bayes error)
  - Calculate avoidable bias = training error - human level error
  - If avoidable bias difference is the bigger, the bias problem
* Training error
  - Calculate variance = dev error - training error
  - If if variance difference is bigger then

you should use a strategy for variance reducing

- So having an estimate of human-level performance gives you an estimate of Bayes error and this allows you to more quickly make decisions as to whether you should focus on trying to reduce a bias or trying to reduce the variance of ~~your~~ your algo.

* These techniques will tend to work well until you surpass human-level performance.

(9) SURPASSING HUMAN-LEVEL PERFORMANCE

Example :-	Error %
Team of humans	0.5%
One human	1%
Training error	0.6%
Dev error	0.8%

avoidable bias = difference b/w Team of humans & training error
= 0.1%

variance = difference b/w training error & Dev error
= 0.2%

In some problems, deep learning has surpassed human-level performance. Like:
- Online advertising
- Product recommendation
- Logistics

- Humans are far better in natural perception tasks like computer vision & speech recognition
- But de. M.L. are doing better in some cases like s diseases cancer, skin diseases etc

(5) IMPROVING YOUR MODEL PERFORMANCE

Two fundamental assumptions of supervised learning is →

(1) You can fit the training set pretty well with low avoidable bias
(2) The training set performance generalizes pretty well to dev/test set with variance is not too bad.

* difference b/w human level & training set error is called avoidable bias
* difference b/w the dev/test ~~and~~ set and training set error is called variance

- If avoidable bias is large you have these options:
  (1) Train bigger model
  (2) Train longer/better optimization algo (Momentum, RMSprop, adam)
  (3) find better NN Architecture/hyperparameter search.

- If variance is large you have these options:
  - Get more training data.
  - Regularization (L2, dropout, data augmentation)
  - Find better NN architecture / hyperparameter search

WEEK - 2

ERROR ANALYSIS

(1) CARRYING OUT ERROR ANALYSIS

Error analysis refers to the process of manually examining mistakes that your algorithm is making.

example:

- You discovered that some of the mislabeled data are dog pictures that look like cats. Should you try to make your cat classifier do better on dogs

- Error analysis approach:
  - Get 100 mislabeled dev set examples at random.
  - Count up how many are dogs.
  - If 5 of 100 are dogs then training your classifier to do better on dogs will decrease your error up to 9.5% which can be too little
  - If 50% of 100 are dogs then you could decrease your error up to 5% which is reasonable and you should work on that.

- Based on last example, error analysis helps you to analyse the error before taking an action

that could take lot of time with no need.

- Sometimes, you can evaluate multiple error analysis ideas in parallel & choose the best idea. Create a spreadsheet to do that and decide eg:-

Image	Dog	Great cats	Blurry	Instagram filters	Comments
1	✓			✓	Pitbulls
2	✓		✓		Rainy day at zoo
3					
4		✓			
...					
% total	8%	43%	61%	12%	

- In the last example you will decide to work on Great cats or Blurry images to improve you performance
- This quick counting procedure, which you can often do in at most, small numbers of hours can really help you make much better prioritization decisions, and understand how promising different approaches are to work on.

② CLEANING UP INCORRECTLY LABELED DATA

If your learning algorithm is not yet at the performance of a human then manually examining mistakes that your algorithm is making can give you insights into what to do to do next. This process is called error analysis

* If you are working on a brand new M.L application then you should build your first system quickly and then iterate.

* ① Set up dev/test set and metric
② Build initial system quickly
③ Use bias/variance analysis & Error analysis to prioritize next steps.

MISMATCHED TRAINING AND DEV/TEST SET

① TRAINING AND TESTING ON DIFFERENT DISTRIBUTIONS

Suppose we want to build a mobile application in which we want in which a application will show to dell weather the photo clicked by user is a cat or not.

Following are the two strategies:-

① Option one (not recommended)
In this we train our model using the pictures we find of cat on internet and take picture clicked by users and then we shuffle and then we train our model but in this strategy ~~train~~ examples in testing and dev set will be low and even if our model is not good with pictures clicked by user it will show high accuracy.

(c) Option two (Recommended)

Train your model using pictures taken of cats from internet and then we can use picture of cats clicked by users on dest Data. This will give us the real accuracy of the model.

(2) BIAS AND VARIANCE WITH MISMATCHED DATA DISTRIBUTIONS

The way you analyze Bias and variance changes when your training set comes from different distribution than your dev and test sets.

- The cat classification example. Suppose you've worked in the example and reached this
  - Human error - 0%.
  - Train error - 1%
  - Dev error = 10%

In this example, you'll think that this is a variance problem, but because the distribution aren't the same you can't tell for sure, because it could be that train set was easy to train on, but the dev set was more difficult.

Here difference is 9%. which is big but it might be good.

So, the problem with this analysis is that
- you went from the training error to dev error two things changed at a time.

- One, is that the algorithm saw data in the training set but not in the dev set.
- Two, the distribution of data in the dev set is different.

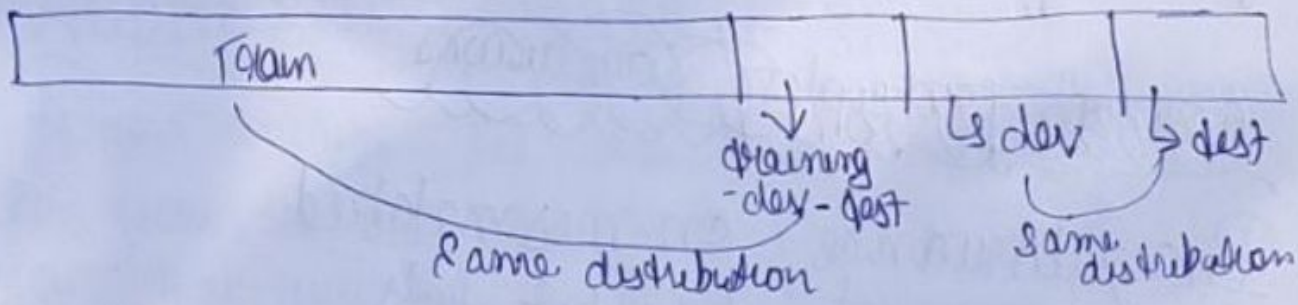
and because you changed two things at the same time, it's difficult to know of this 9% increase in error how much of it is because the algorithm didn't see the data in the dev set and how much of it, is because the dev set data is just different

In order to tease out those two effects it will be useful to define a new piece of data which we'll call the training-dev set (new subset of data which we carve out that should have the same distribution as training set)

> Training-dev set: Same distribution as training set but not used for training

example

* Now we're going to do is randomly shuffle the training sets and then carve out just a piece of the training set to be the training-dev set



To carry out error analysis, what you should do is now look at the error of your classifier on the

training set, on the training-dev set, as well as on the dev set.

Example:
Now like have:

Human error : 0%.
Train error : 1%.
Train-dev error : 9%.
Dev error : 10%

Now we are sure that this a high variance problem

another example.

Human error - 0%.
Train error - 1%.
Train-dev error - 1-5%.
Dev error - 10%.

In this case we have something called data mismatch problem because your learning algorithm was not trained explicitly on data from training-dev or dev. but these two data sets come from different distribution.

~~General principles / conclusions~~

Bias/variance on mismatched training and dev/test sets

Human level	4%	] - avoidable bias
Training set error	7%	] → variance
Training dev set error	10%	] data mismatch
Real error	12%	] degree of overfitting to dev set.
Test error	12%	

→ for avoidable bias

* If the difference is big then its avoidable bias problem then you should use a strategy for high bias
* If the difference is big then its high variance problem then you should use a strategy for solving it. (for variance)
* If difference is much bigger than train-dev error its data mismatch problem.
* degree of overfitting to dev set = test error - dev error
  If the difference is big (positive) then maybe you need to find a bigger dev set (dev set and test set come from the same distribution)

3) ADDRESSING DATA MISMATCH

If your training set comes from a different distribution than your dev and test set, and if error analysis shows you that you have a data mismatch problem.
what to do?

#There is no completely systematic solution to this, but there some things you could try.

Addressing data mismatch

- Carry out manual error analysis to try to understand difference b/w training & dev/test sets
- Make training data more similar; or collect more data similar to dev/test sets

If your goal is to make the training data more similar to your dev set one of the techniques you can use artificial data synthesis that can help you make more training data

- Combine normal audio with car noise to get audio with car noise example
- But we do need to be cautious and bear in mind whether or not you might be accidentally simulating data from a tiny subset of the space of all possible examples because your NN might overfit these generated data

LEARNING FROM MULTIPLE TASKS

① Transfer learning

Transfer learning refers to using the NN knowledge for another application.

When to use transfer learning?
① Task A & B have same input x.
② A lot more data for Task A than Task B
③ low level features from task A could be helpful for task B

example 1:
The following network is trained for cat recognition, but we want to adapt it for radiology diagnosis the neural network will learn the structure & the nature of images. This initial phase of training on image recognition called pre-training, since it ends pre-initialize the weights of NN updating all the weight afterwards is called fine-tuning.

example 2:
let's say that you've trained a speech recognition system. So now X is input of audio & Y is some ink transcript. So, you've trained in speech recognition system to output your transcripts. And let's say that you now want to build a 'wake words' or trigger words detection system. Such saying Alexa, Ok Google, to wake up device

Transfer learning

MULTI-TASK LEARNING

In multi-task learning, you start off simultaneously trying to have one neural network do several things at the same thing and then each of these task helps hopefully all of the other task.

Example:

Let's say you're building an autonomous vechile, building a self driving car. Then your self driving car would need to detect several different things such as pedastrians, detect other cars, detect stop signs. Then Y shape will be $(4, m)$.

We can also trained 4 NN seperately but if some of the earlier features in neural network can be shared b/w these different of object then you find that NN. to do 4 things

$$Y = \begin{bmatrix} 1 & ? & 1 & \dots \\ 0 & 0 & 1 & \dots \\ ? & 1 & ? & \dots \end{bmatrix}$$

Why multi-tasking?

- Training on a set of tasks that could benefit from having shared lower-level features
- Usually, amount of data you have for each task is quite similar
- Can train a big enough network to do well on all the tasks.

- If you train a big enough NN, the performance of the multi-tasking learning compared to splitting the task is better.
- Today, transfer learning is used more often than multi-tasking learning.

LEARNING FROM MULTIPLE TASKS

END-TO-END DEEP LEARNING

1) What is end to end deep learning?

Some system have multiple stages to implement. An end-to-end deep learning system implements all these stages with single NN.

example :-

- Speech recognition system

audio → features → phonemes → words → transcript # non-end-to-end system

audio ——————————→ transcript # end-to-end learning system.

- End-to-end learning gives data more freedom, it might not use phonemes when training.

* To build the end-to-end deep learning system that works well, we need a big dataset (more data than in non end-to-end system). If we have a small dataset the ordinary implementation could work just fine.

6 @Wheather to use end-to end deep learning

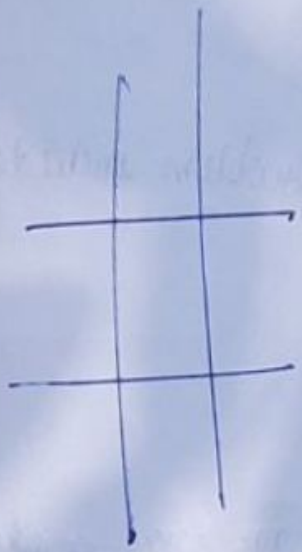
- Pros of end-to-end deep learning
  - By having a pure ML approch, your NN learning input from X to Y may be more able to capture whatever statistics are in the data, rather than being forced to reflect human preconceptions.
  - less hand-designing of components needed.

Cons of end to end learning

- May need a large amount of data.
- Excludes potentially useful hand-design components

Applying end-to-end deep learning.

- Key question: Do you have sufficient data to learn a function of the complexity needed to map X to Y?
- Use DL/ML to learn some individual components
- When aplying supervised learning you should carefully choose what types of X to Y mapping you want to learn depending on what data you can get data for:



#4

CONVOLUTIONAL NEURAL NETWORKS

WEEK 1

① COMPUTER VISION

What is computer vision?

Computer vision is a field of computer science that works on enabling computer to see, identify and process in the same way that human vision does.

Some of the applications of computer vision that are using deep learning are:

- Self driving cars.
- Face recognition.

Examples of a computer vision problem include

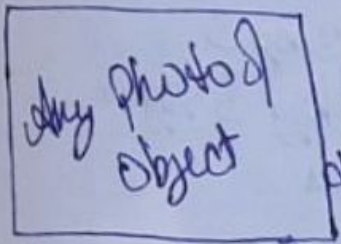
- Image classification
- Object detection
  - Detect object & localize them
- Neural style transfer
  - changes the style on an image using another image (filters) ~~adding animate ch~~ (creating ~~and adding animate the~~

~~Suppose we have a picture of cat or that 64 by 64 by 3 (64×64×3~~
Suppose we have a cat picture of 64×64×3 (that's 12288) that means we say 12288 on x (input).
But what if we have 1000×1000×3 image of cat (1000 = 1 Mega Pixel)
if we have 3 million input features than this means that x will be 3 million dimensional

~~To do that, you~~
One of the solution is to build this using convolution layers instead of the fully connected layer.

2) EDGE DETECTION EXAMPLE

In this we'll see how convolution operation works.
Example:-



Given a picture like that for a computer to figure out what are the objects in this picture, the first thing you'll do is maybe detect vertical edges in this image and we also want to find horizontal edges. So how we detect edges of an image?

Here is an 6×6 grayscale image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

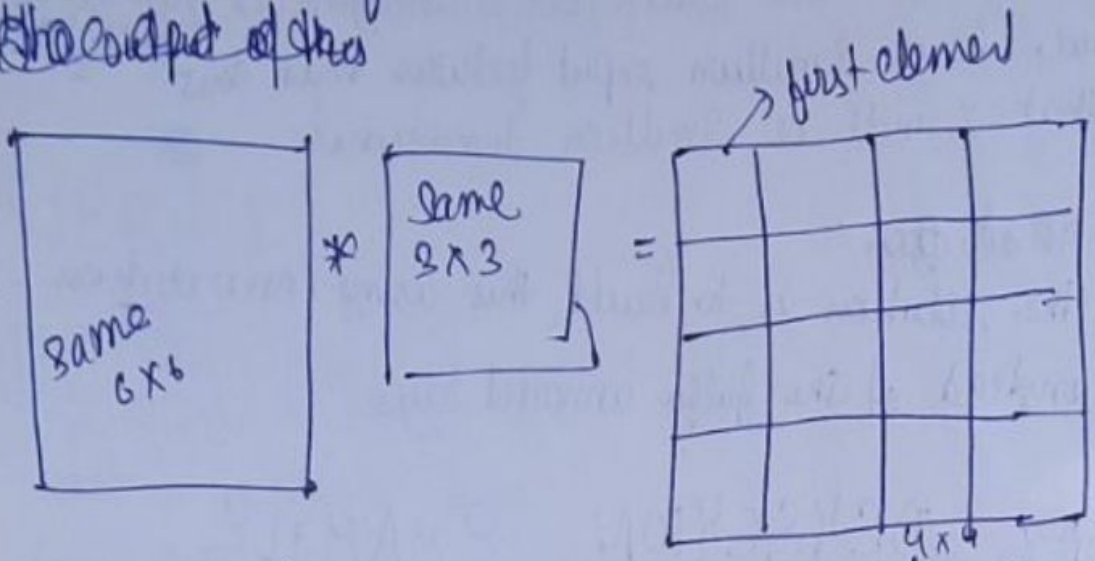
In order to detect edges or lets say vertical edges in this image what we'll do construct a 3×3 matrix known as filter

1	0	-1
1	0	-1
1	0	-1

Sometimes filter is also called Kernel

and what you are going to do is take the 6 by 6 image and convolve it and the convolution operation is denoted by asterisk (*) (This is not multiplication) this astrik is refers to convolution.

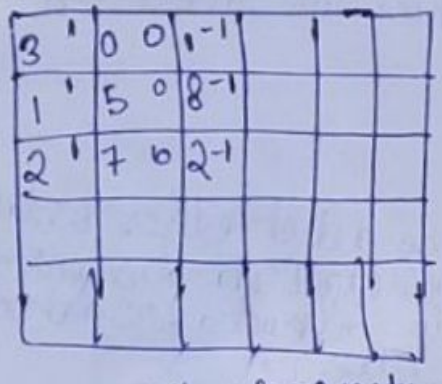
The output of this



The output of the convolution operator will be a 4X4 matrix

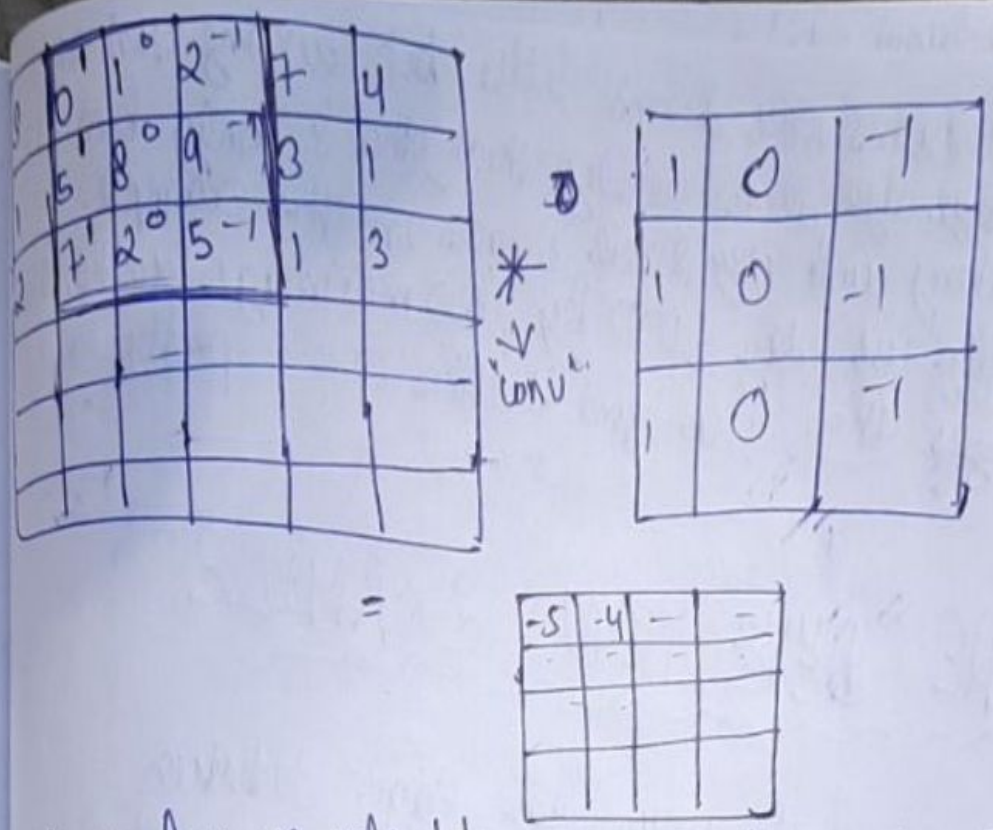
The way to compute this 4X4 output is as follow :-

The first element of 4X4 matrix is calculated as what you are going to do is take the 3 by 3 filter and paste it on top of the 3 by 3 region of your original input images like this



$3\times1 + 1\times1 + 2\times1 + 0\times0 + \text{-----} = -5$

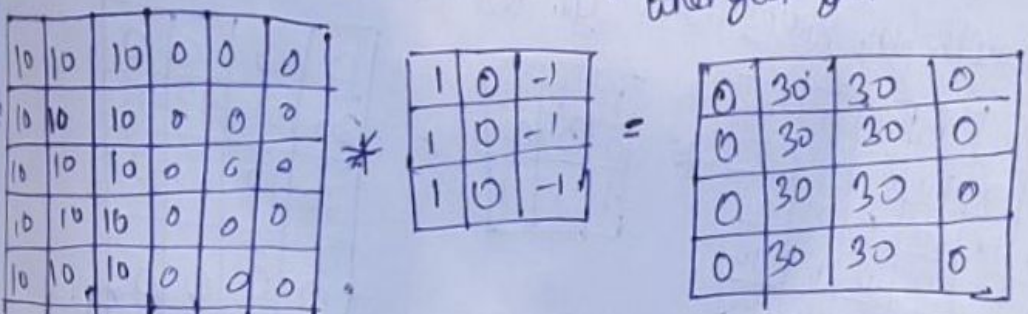
and to compute 2nd element we do the same calculation but now we take one step right like this :-



this is how we calculate all the value and this turns out to be a vertical edge detector

we can do this tensorflow stages by using this function
tf.nn.conv2d

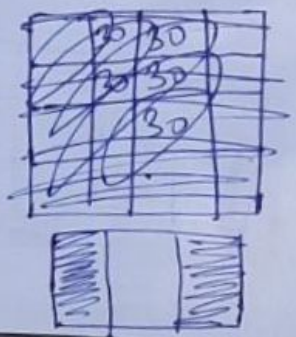
Let's look at another example:
what you'll get



If we plot this image it might look like this

simplified can be written as follows

If we plot this it will look like this



In case dimensions here seems little bit wrong that the detected image edge seems really thick that's only because we are working with very small images in this example. will Generally use say 1,000 by 1,000 more image rather than 6X6 images then you find that this does a pretty good job.

② MORE DETECTION EXAMPLE

Vertical and Horizontal Edge detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

example for horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

So in summary, different filters allow you to find vertical and horizontal edges. It turns out that the three by three vertical edge detection filter we've used is just one possible choice

In computer vision literature there was a fair amount of debate about what is the best set of numbers to use. So here's some of it

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

And this is just for vertical edge detection if we flip this it will become horizontal edge detector. ~~>~~ 90°

When you really want to detect edges in some complicated image, maybe you don't need to have computer vision researches handpick these nine numbers. Maybe you can just learn them and treat the nine numbers of this matrix as parameters, which you can then learn using back propagation and the goal is to learn nine parameters so that when you take the image the 6 × 6 one image, and convolve it with your three by three filter then this will give you a good edge detector and by just letting all of these numbers be parameters and learning them.

automatically from data, we find that NN can actually learn low level features.

(3) PADDING

$(n \times n)$ $\rightarrow (f \times f)$

If we have 6×6 image and 3×3 filter then the dimension of the output will be

$n-f+1 \times n-f+1$

$6-3+1 * 6-3+1$

4×4 (output matrix)

Note: Our image shrunks after applying filter and also the pixel at the corner or the edge is used only in one of the output (which is 1st element for apply filter for the first time) whereas if the pixel in the middle which is taken by filter many times

In short, pixels on the corners or on the edge are used much less than pixels in the middle. So here we are throwing away lot of information.

To solve this problem we use padding.

Padding refers to adding padding to the image with additional border.

→ This will change the pixel dimension of input

4×5 ~~5×6~~

to

6×7 after padding

So how to calculate the dimension of output after padding?

Here is formula for this

$n+2p-f+1 \times n+2p-f+1$

{ p = padding
n = input
f = filter

So after padding edges/corner pixel are used many times for calculation of output.

We can also do padding of 2 pixel. this will add 2 borders to our input pixel.

In terms of how much to pad, it turns out there are two common choice that are called Valid and Same convolutions.

Valid convolution → No padding

input will be $n \times n$ filter will be $f \times f$
and output dimension will be $n-f+1 \times n-f+1$

Same convolution → Pad so that output size is same as the input size.

when we pad with p pixel

$n+2p-f+1 \times n+2p-f+1$

$n+2p-f+1 = n$ (So the output size is same as input size)

$= p = \frac{f-1}{2}$

f is usually odd and you'll rarely see even numbered filters

We have 2 reasons for that :-

① If f is was even then you need some asymmetric padding. So only if f is odd that this type of same convolution gives a natural padding region. (pad more on the left pad more less on right or opposite called asymmetric padding)

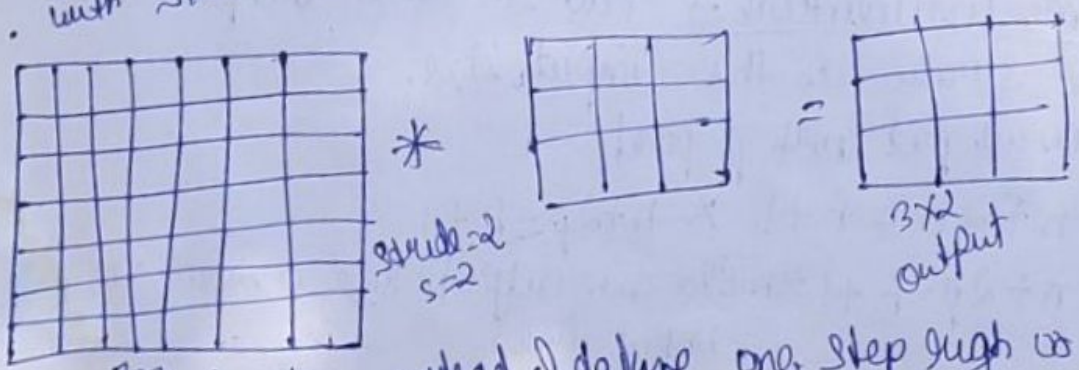
② when you have an ~~as~~ odd dimension filter, such as 3×3 then it has a central position and sometimes in computer vision its nice to have a distinguisher (central pixel)

Maybe none of this is a great reason to say 3×3 filters. 3×3 are very common

It will give fine performance even if we use 4×4 ~~ff~~ filter

④ STRIDED CONVOLUTION

We want to convolve this 7×7 image by 3×3 filter with stride of 2



In this instead of taking one step right or down we take 2 steps

If we stride at 2 (S=2) then what is the dimension of the output? Here the formula

$$\frac{n+2p-f}{s}+1 \quad \times \frac{n+2p-f}{s}+1$$

$$\frac{7+0-3}{2}+1 \quad \times \quad -$$

$$= \underbrace{3}_{\text{If this value is not int in that case we'll round this down}} \times 3$$

$$\lfloor z \rfloor = \text{floor}(z)$$

↑ floor symbol

Your 3×3 filter must lie entirely within your image or image + padding

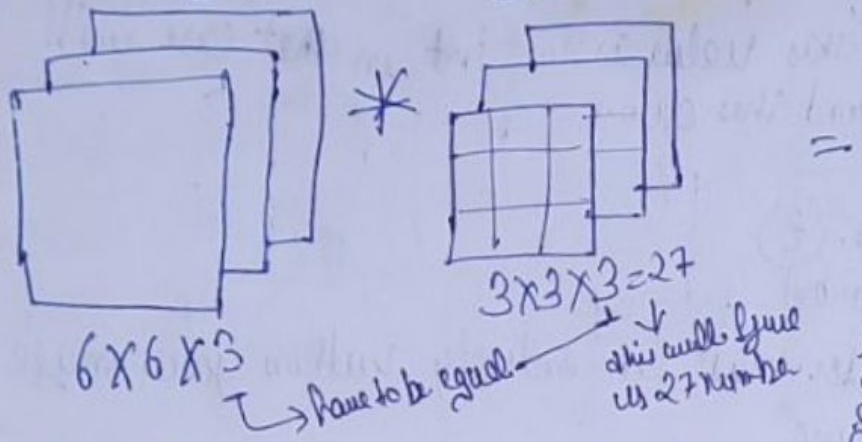
Technical note on cross-correlation vs. convolution

In math textbooks the conv operation is flipping the the filter before using it. what we were doing is called cross-correlation operation but the state of art of deep learning is using this as conv operation.

Technically, what we're actually doing, the operation we've been using for the last few videos is sometimes cross-correlation instead of convolution. But in deep learning literature by convention we just call this a convolutional operation

(5) CONVOLUTIONS OVER VOLUME

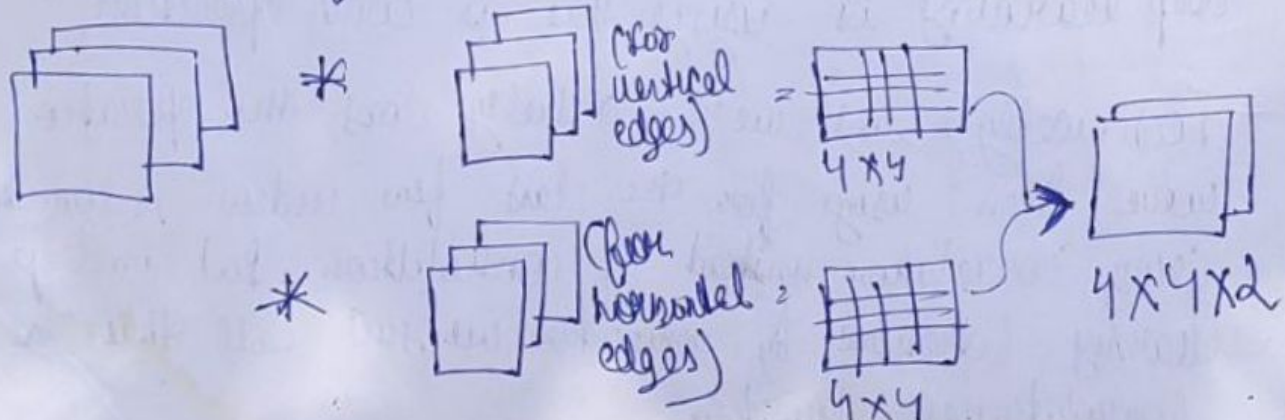
Lets say you want to detect features, not just Grey scale image but in RGB image like (6X6X3) then will have feature image of (3X3X3)



This is calculated as same as discussed but for every layer. Then add up all the number this will give you first number

Now we have learned to convolve on volumes.

what if we don't just want to detect vertical edges? What if we wanted to detect vertical edges and horizontal edges in short what if you want to use multiple filters at the same time?



Summary:

Input $= n \times n \times n_c$ (number of channels) $= 6 \times 6 \times 3$

* filter

Filters $f \times f \times n_c$ (number of channels) $3 \times 3 \times 3$

= output (no. of filters)

$n-f+1 \times n-f+1 \times n_c'$

$4 \times 4 \times 2$

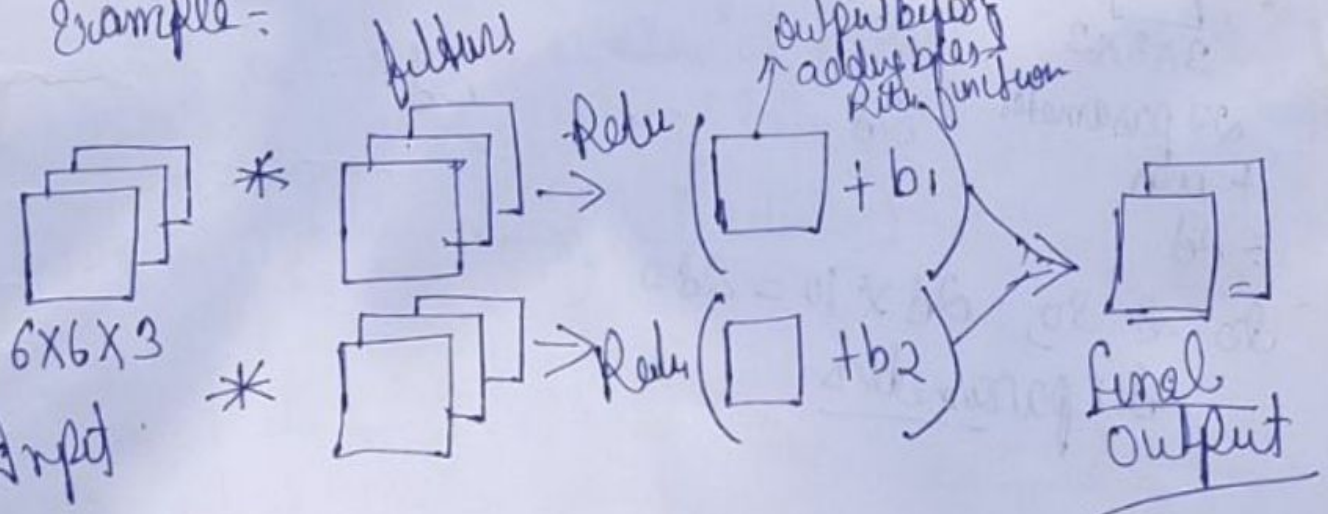
n_c of input should be same as n_c of filters

If we use strip this more than one (1) then this formula will change

* The output have number of channels equal to the number of filters you are detecting.
* The last dimension in input layer in M.L. lit reader people often call this the depth of this 3D volume

(6) ONE LAYER OF A CONVOLUTIONAL NETWORK

Example:-



This is one layer of Convolutional network

So to map this back to one layer of four propagation in the standard ~~neural~~ N.N, in a non-convolution network remember.

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

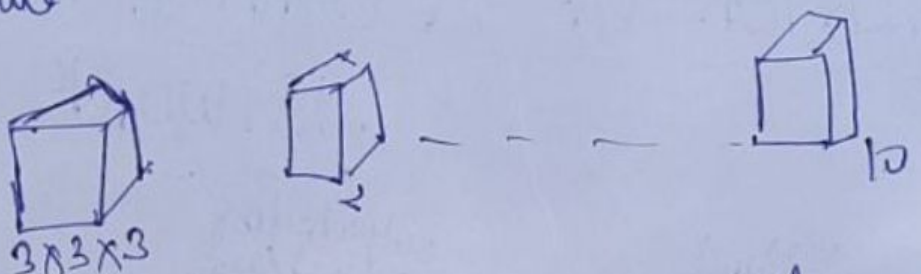
$$a^{[1]} = g(z^{[1]})$$

* filters are similar to $w^{[1]}$
* $a^{[0]}$ are similars to input
* $b^{[1]}$ are bias
* $a^{[1]}$ is our final output

Number of parameters in one layer

Q) If you have 10 filters that are 3×3×3 in one layer of NN. How many parameters does that layer have

Sol)



27 parameter + bias = 28

28 ---> 28

So 28 so, 28 × 10 = 280

280 parameters

Summary of notation

$f^{[l]}$ = filter size
$p^{[l]}$ = padding
$s^{[l]}$ = stride
$n_c^{[l]}$ = nume of filter

Input : $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c$ → because, that's the activation from the previous layer.

n_c ↓ no. of channels in an image

H = height
w = weigh

Output : $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

l = layer of the output

Each filters is : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations : $a^{[l]} \rightarrow n_H^{[l]} * n_w^{[l]} * n_c^{[l]}$

and when you are using a vectorised implementation or batch gradient then your actually output a is -

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_w^{[l]} + n_c^{[l]}$

↓ m examples

Weights : $f^{[l]} \times f^{[l]} \times n_c^{(l-1)}$ (that's the dimension of one filter)

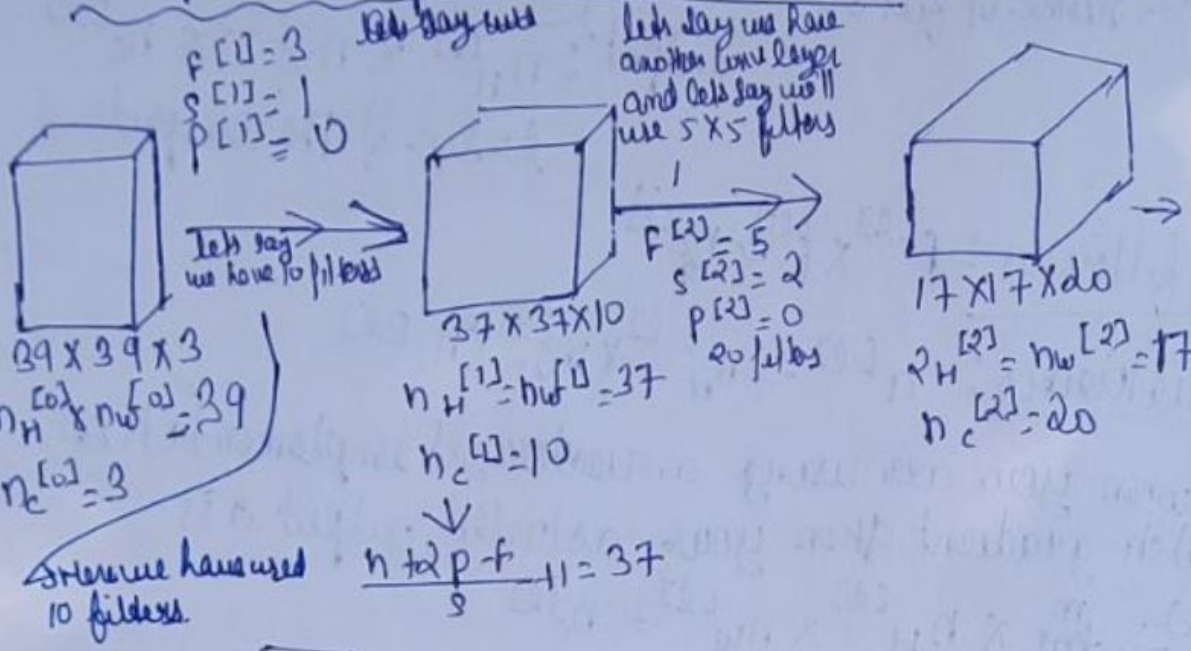
$n_c^{[l]}$ are the number of filter

so, $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

Bias : $n_c^{[l]}$ → (Real number)

(6) SIMLE CONVOLUTIONAL NETWORK

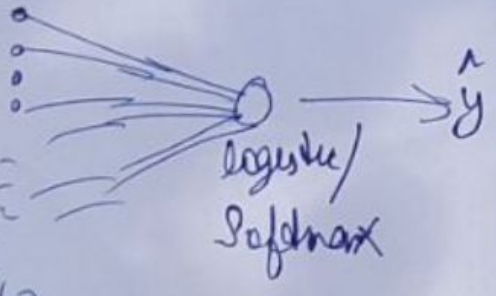
example of deep convolutional network (ConvNet)



Otherwise have used 10 filters.

$F^{[3]} = 5$
$S^{[2]} = 2$
40 filters

7 times 7 times 40 is 1960.
And so what we can do is take this volume and flatten it



1960
↳ It is a very long vector

Note: the size of the images are do trend to go down 39x39 to 37x37 to 17x17 to 7x7 while the channels will generally increase 3 to 10 to 20 to 40

Types of layer in a convolutional network:

- convolution (Conv)
- Pooling (Pool)
- Fully connected

You can also create a convNet only using convolution

(7) POOLING LAYERS

other than convolutional layers, ConvNets often also use pooling layers to reduce the size of the representation ConvNets often use pooling layers to reduce the size of the representation to speed the computation

example:

Suppose you have a 4x4 input and you want to apply a type of pooling called max pooling.

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

→ max of each partition

9	2
6	3

So, this is as if you apply a filter size of two

f = 2
s = 2

} Hyperparameters of max pooling

So here's the intuition behind what max pooling is doing. If you think of this 4×4 region as some set of features, the activations in some layer of NN, then a large number means that it's maybe detected a particular feature. So, what the max operation does is a lots of features detected anywhere and one of these quadrants, it then remains preserved in the output of max pooling. If these features are detected anywhere in this filter then keep a high number. But if this features doesn't is not detected keep low number. So, that's the intuition behind max pooling.

One interesting property of max pooling is that it has a set of parameter (S and F) but it has no parameters to learn. Gradient descent will not change this parameter.

Pooling layer : Max Pooling

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

5×5

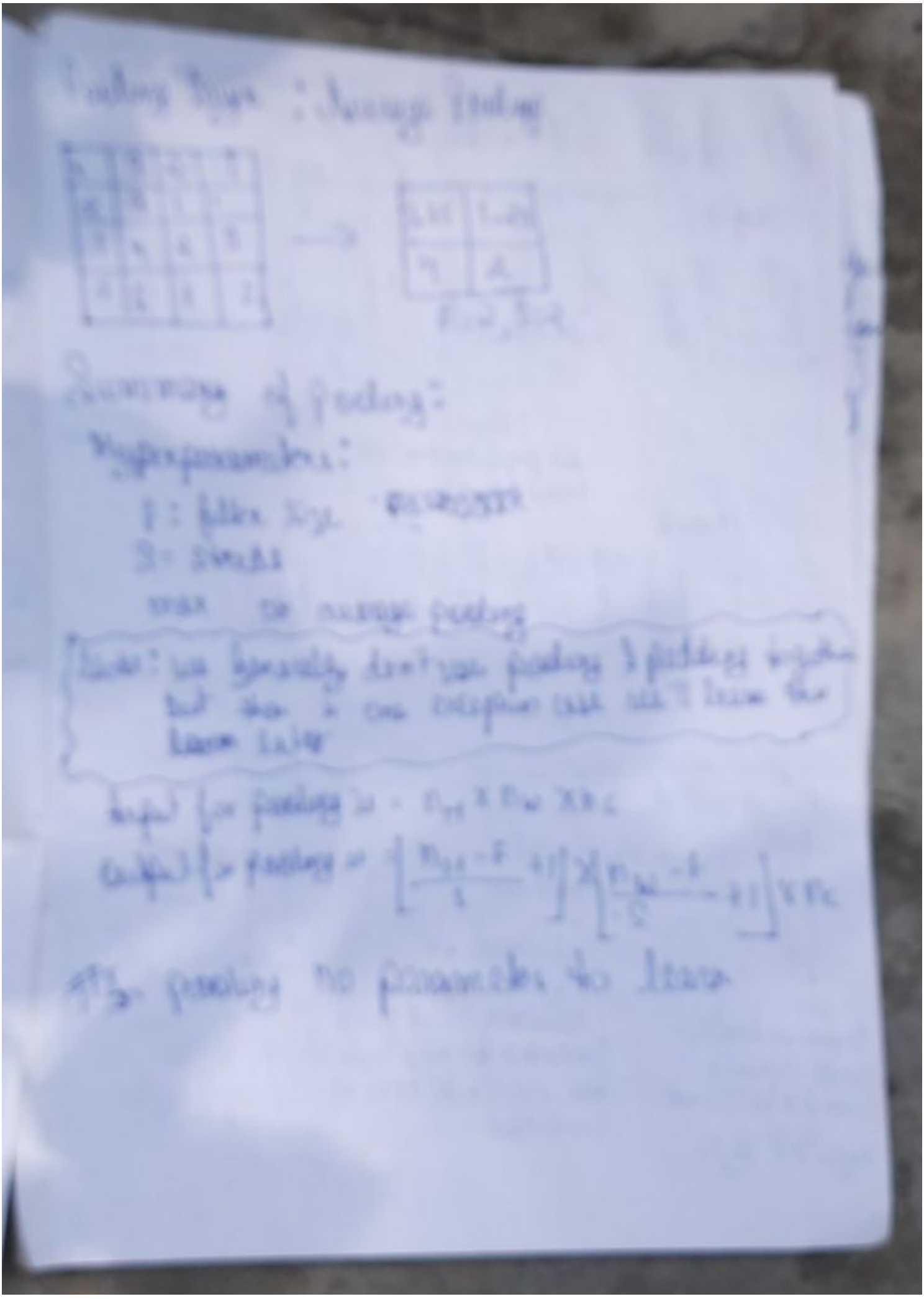
→

9	9	5
9	9	5
8	6	9

3×3

f = 3, S = 1

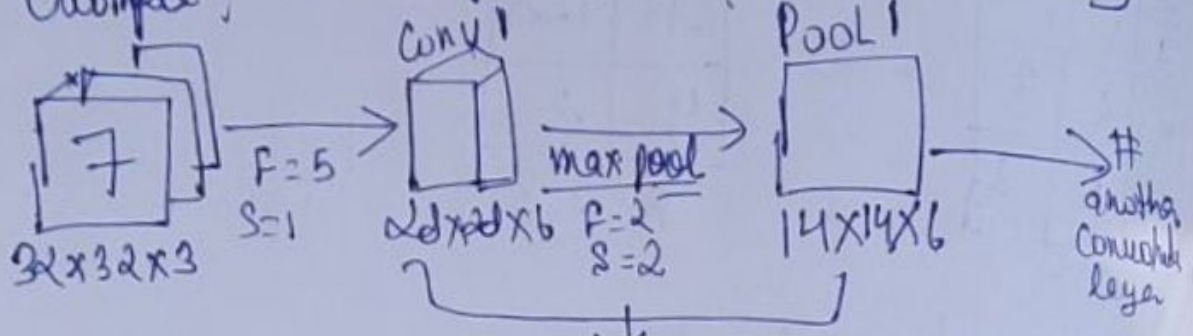
It is called named in 2D input.



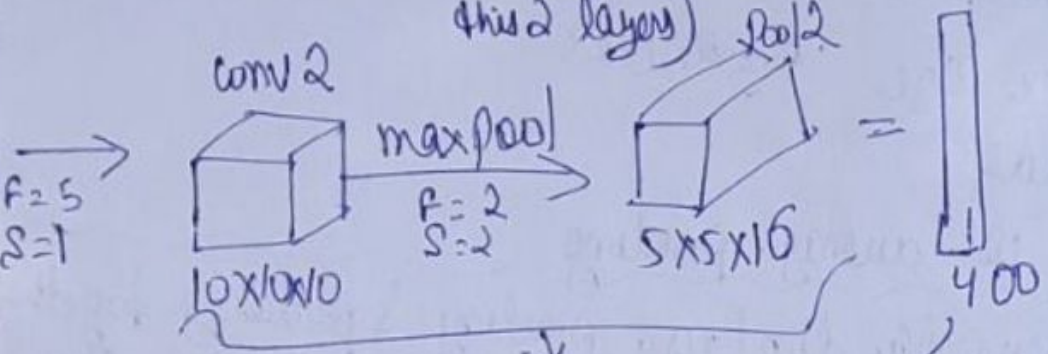
(8) CNN EXAMPLES

(This example is so

Example :- (This example is something like LeNet-5)



1 layer
(But people sometimes call this 2 layers)



layer 2

Fully Connected

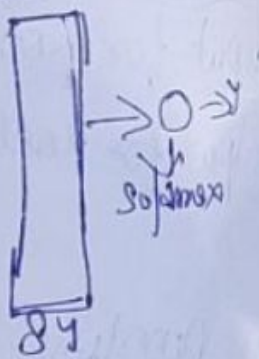
FC3

a weight matrix that's called $W^{[3]}$ of dimension 120×400

$W^{[3]} : (120, 400)$, $b^{[3]} : 120$

Because we have 400 units densely connected to 120 units.

This is called fully connected because each of the 400 units here is connected



A single NN layer

Some statistics about the last example:

	Activation shape	Activation size	# Parameters
Input	(32,32,3)	3072	0
Conv (F=5,s=1)	(28,28,8)	6272	208
Pool1	(14,14,8)	1568	0
Conv2 (F=5,s=1)	(10,10,16)	1600	416
Pool2	(5,5,16)	400	0
FC3	(120,1)	120	48001
FC4	(84,1)	84	10081
Softmax	(10,1)	10	841

(9) WHY CONVOLUTIONS?

Advantages

(1) Parameter sharing
- A feature detector (such as vertical edge detector) that's useful in one part of the images is probably useful in another part of the image.

(2) Sparsity of connections
- In each layer, each output value depends only on a small number of inputs which makes it translation invariance

Putting it all together:

WEEK 2

① WHY LOOK AT CASE STUDIES?

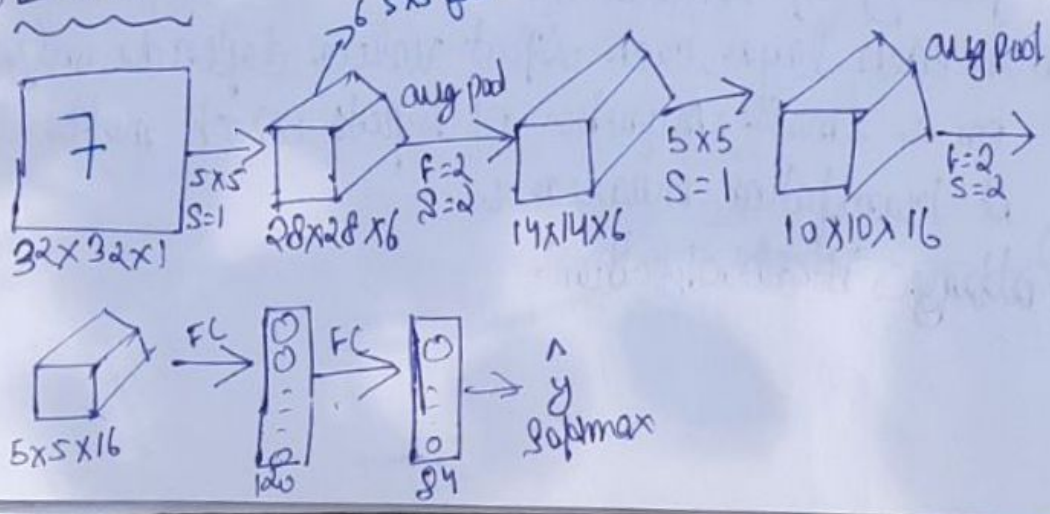
- Some NN architecture that works well in some tasks can also work well in other tasks.
- Here are some classical CNN network:
  - LeNet-5
  - AlexNet
  - VGG
- The best CNN architecture that won last ImageNet competition is called ResNet & it has 152 layers

② CLASSIC NETWORKS

An ans some of the classic neural network are:
1. LeNet-5
2. Alex Net
3. VGG

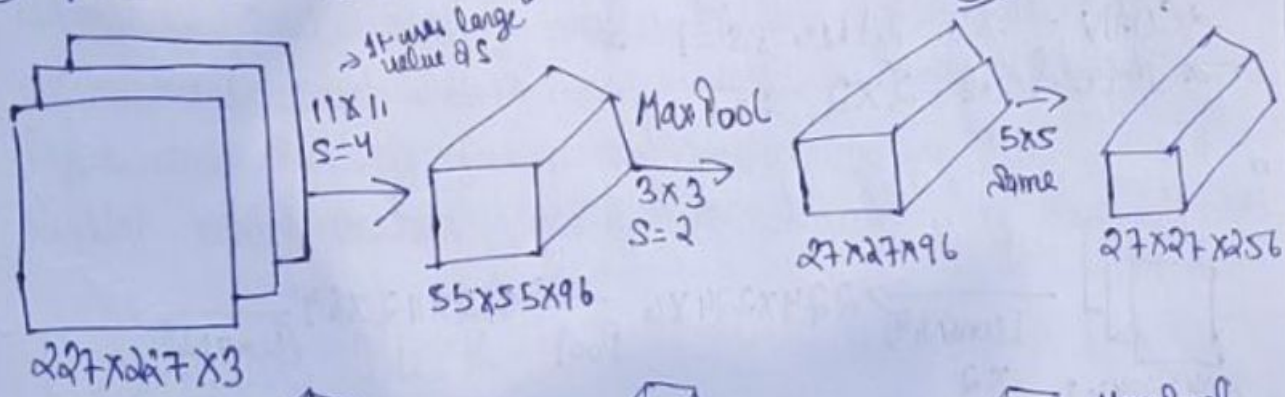
① LeNet-5

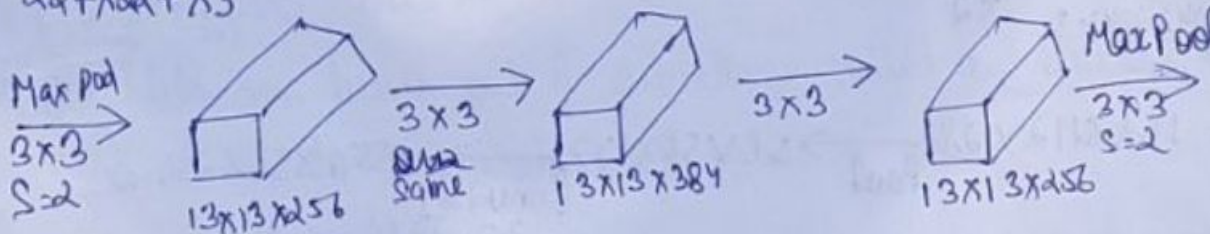
- it is trained on greyscale image

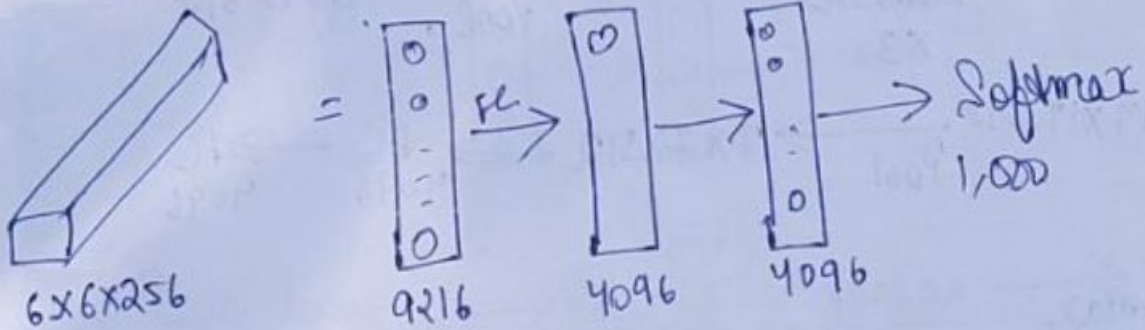


- This model was published in 1998. The last layer wasn't using Softmax Back then
- It Has 60K Parameters
- Dimensions of the images ↓ as no. of channels ↑
- Conv ⇒ Pool ⇒ Conv ⇒ Pool ⇒ FC ⇒ FC ⇒ Softmax this type of arrangement is quite common
- The activation function, used in the paper was Sigmoid & Tanh

② AlexNet (named after Alex Krushevsky)





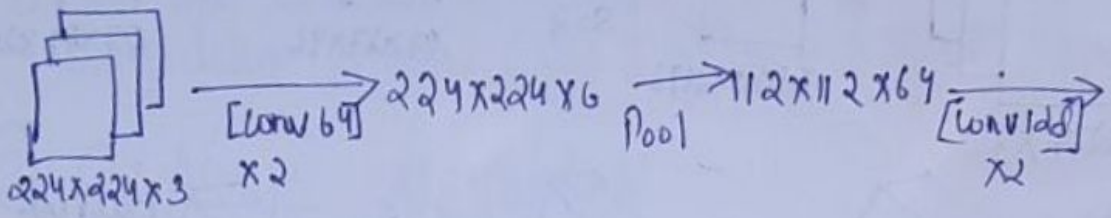


- Similar to LeNet-5 but Has 60 million parameters compared to 60K parameter of LeNet-5
- Has Relu activation function

- Multiple GPUs were used because the GPUs were not so fast back then

(3) VGG-16

- Instead of having so many hyperparameters it use a much simpler network where you focus on just having Conv-layers that are just 3×3 filters with a stride of one and always use same padding.
- It focus on having only these blocks:
  - CONV = 3×3 filter, s=1 &
  - Max-Pool = 2×2, s=2



→ 112×112×128 → Pool → 56×56×128 → [Conv 256] ×3 → 56×56×256 →

Pool → 28×28×256 → [Conv 512] ×3 → 28×28×512 → Pool → 14×14×512

[Conv 512] ×3 → 14×14×512 → Pool → 7×7×512 → FC 4096 → FC 4096

→ Softmax 1000

- This network is large even by modern standards. It has around 138 million parameters.

- At has a total memory of 96MB per image for only forward propagation
- Number of filters increase from 64 to 128 to 256 to 512. 512 was made twice

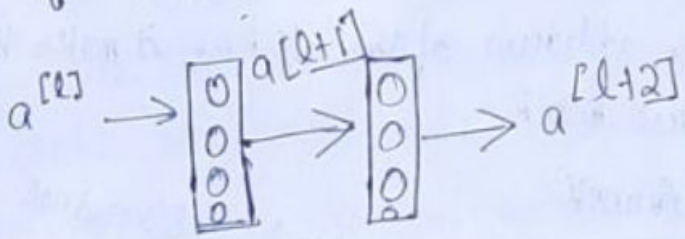
3) RESNETS

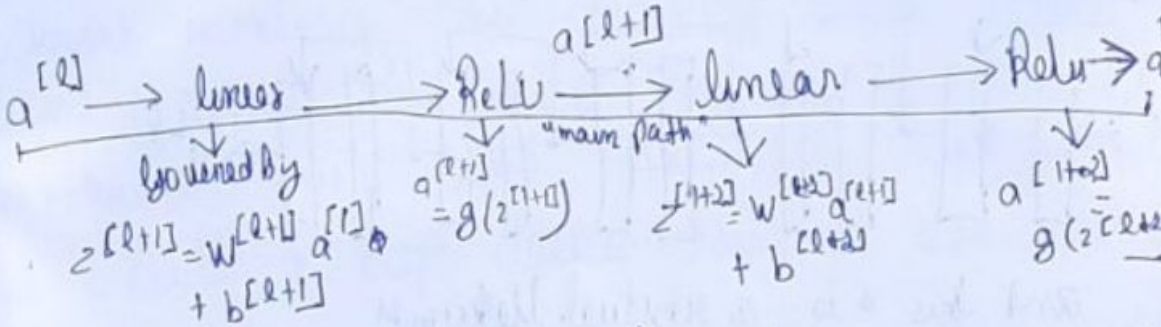
Very, very deep NN are difficult to train because of vanishing and exploding gradient. We'll learn about skip connections which allows you to take the activation from one layer and suddenly feed it to another layer. You'll build ResNet which enables you to train very, very deep networks.

RESIDUAL BLOCK

ResNets are built out of something called a residual block let's first describe what that is -

- 2 layer of a NN

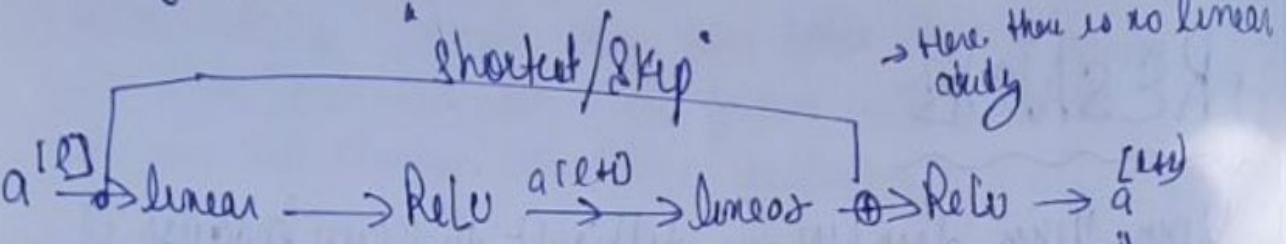




This is called main path. and we are going to change this

we're going to going make a change to this.

we're going to take a[l] and just forward it, and just add a[l] before applying to non-linearity, the Relu non-linearity and we'll call this a shortcut



So, rather than needing to follow the main path the information from a[l] can now follow a shortcut to go much deeper into the N.N. also this means that last equation goes away and we instead have that the output $a^{[l+2]}$ is the Relu non-linearity g applied to $z^{[l+2]}$ as before but now plus a[l]

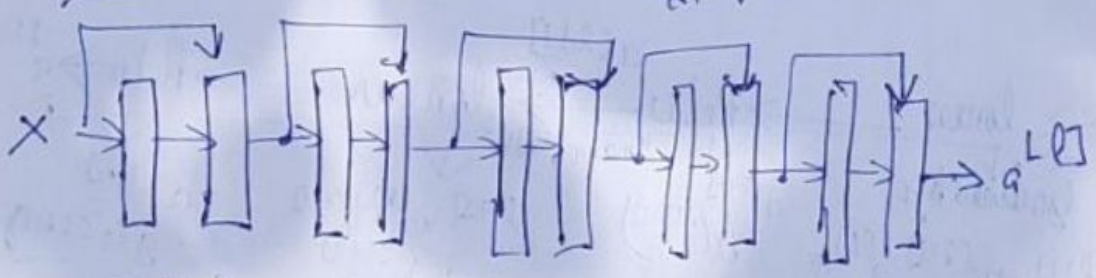
→ Formula for this change to g = non-linearity

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

So, the addition of this a[l] here, it makes this a residual block

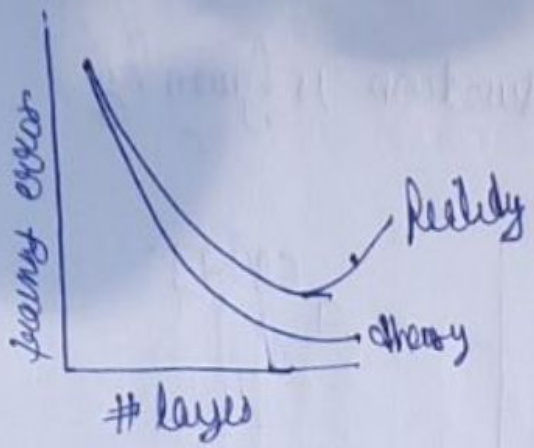
Residual Nn Network

It has 5 residual block

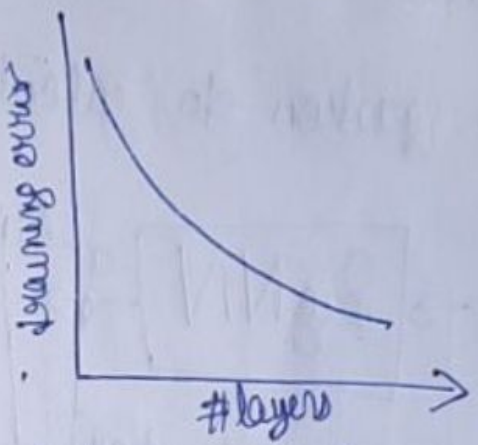


And this is a residual Network

Plain



ResNet



- performance of ResNet increases as the network goes deeper

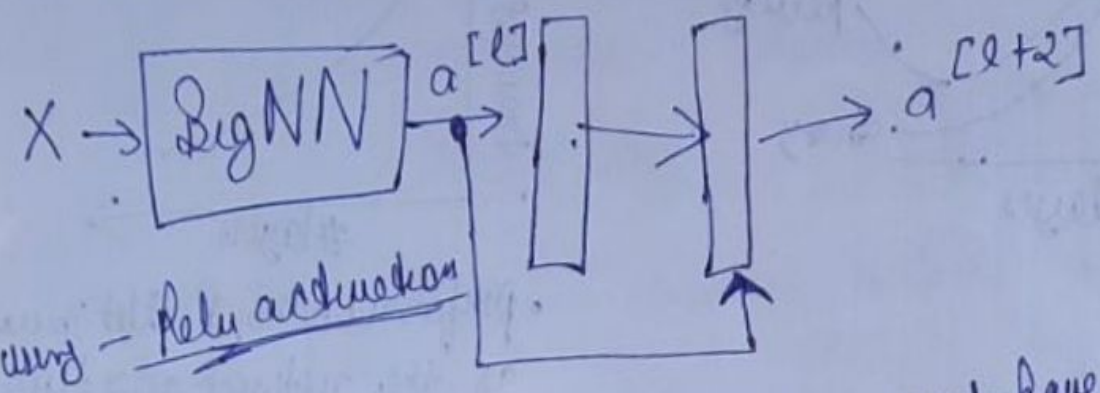
- In some cases going deeper won't effect the performance and that depends on the problem on your hand.
- Some people are trying to train 1,000 layer now which isn't used in practice

(4) WHY RESNETS WORK

In order to make a good model, we first have to make sure that it's performance on the training data is good. We have seen earlier that training deeper deeper networks using a plain network increases the training error after a point of time. But while training a residual ~~an~~ network, this isn't the case. Even when we build a ~~dep~~ deeper residual network, the training error generally does not

(increase.

The equation to calculate activation is given by



using - Relu activation

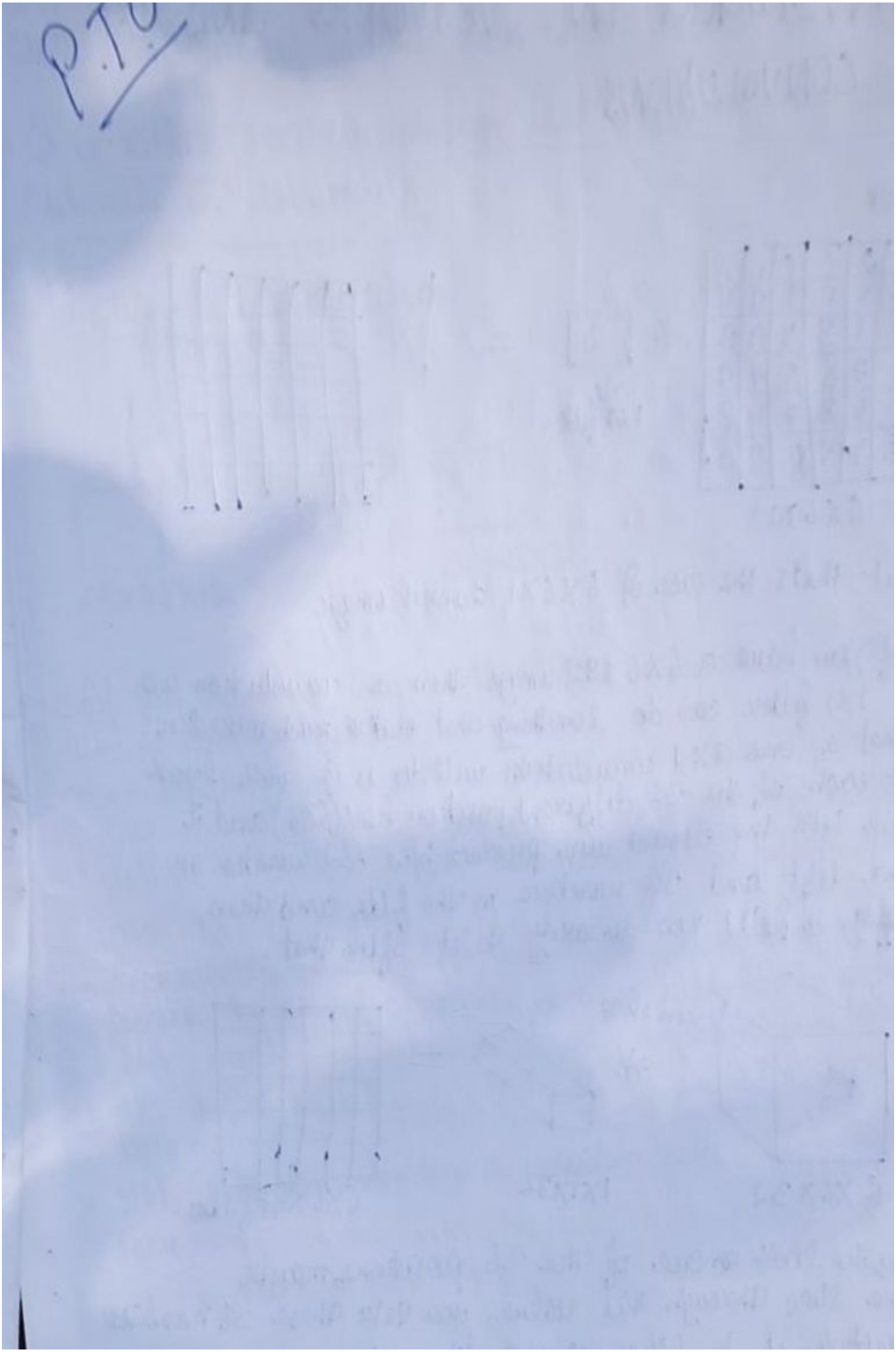
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) \rightarrow \text{they must have same dimensions}$$

$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

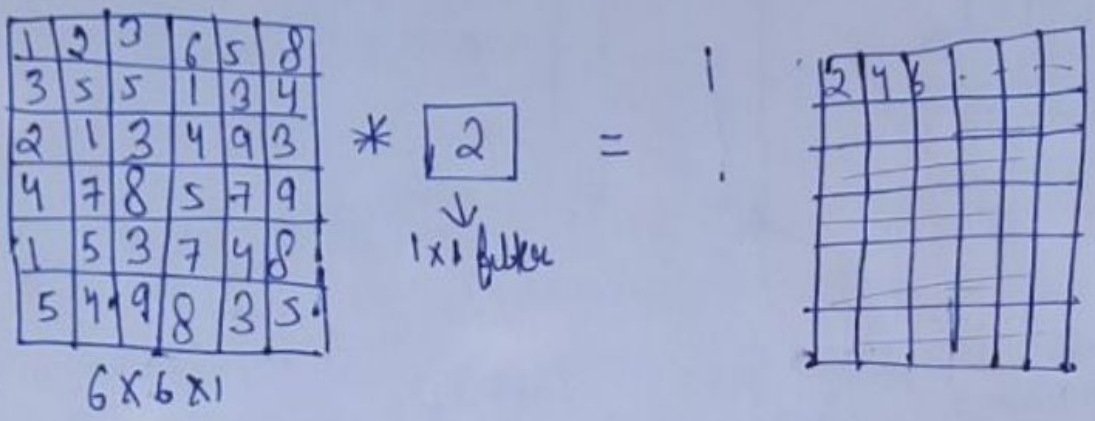
Now, lets say $w^{[l+2]} = 0$ and the bias $b^{[l+2]}$ is also 0

then $a^{[l+2]} = g(a^{[l]})$

It is fairly easy to calculate $a^{[l+2]}$ knowing just the value of $a^{[l]}$.

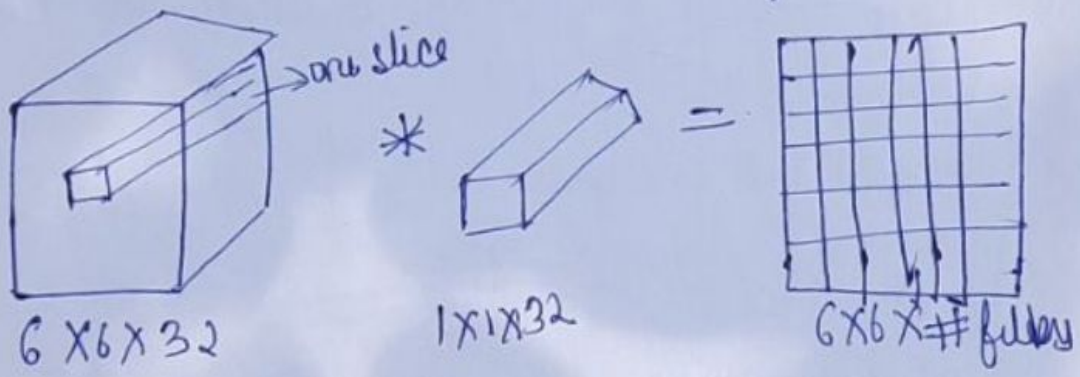


(5) NETWORKS IN NETWORKS AND 1X1 CONVOLUTIONS



But that's the case of 6X6X1 channel image

If you have a 6X6X32 image then a convolution with a 1X1 filter can do something that makes much more sense what a ~~one~~ 1X1 convolution will do is it will look at each of the 36 different position here (6X6) and it will take the element wise product b/w 32 numbers on the left and 32 numbers in the filter and then apply a ReLU non-linearity to it after that.

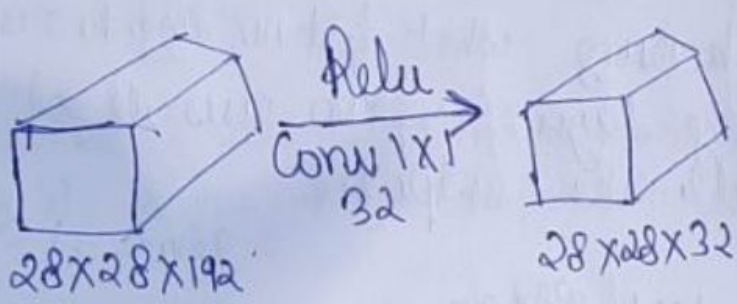


So, to look on one of the 36 positions, maybe one slice through this value, you take these 36 numbers multiply it by / slice through the volume like that,

and we end up with a single number which then gets plotted in one of outputs like that.

- It is called 1×1 convolution but sometimes also called Networks in Networks.

Using 1×1 convolution

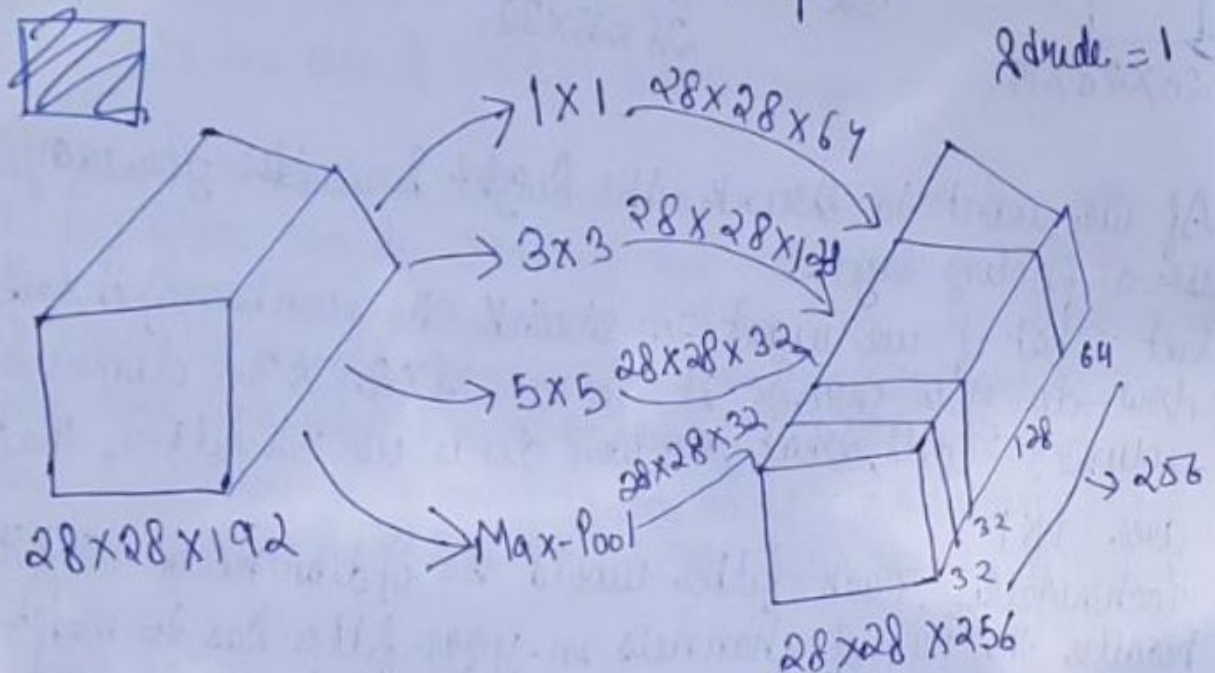


- If we want to shrink the height & width you can use a pooling layer
- But what if we want to shrink the number of channels how do you shrink it to a 28×28×32 dimension volume? Well, what you can do is use 32 filters that are 1×1
- Technically, each filter would be of dimension 1×1×192 because the no. of channels in your filter has to match the number of channels in your input volume and the output of this process will be a 28×28×32 volume
- This help in Building a inception network (next topic)

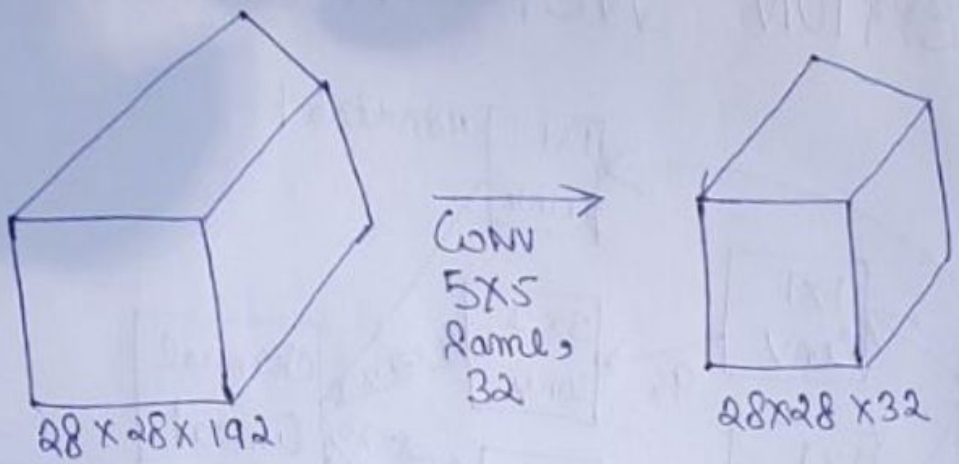
⑥ INCEPTION NETWORK MOTIVATION

- When you design a CNN you have to decide all the layers yourself. Will you pick a 3×3 conv or 5×5 conv or maybe a max pooling layer. You have so many choices

Suppose we have 28×28×192 input volume. Instead of ~~using~~ or choosing what filter size to use or conv layer or pooling layer Inception uses all of them and stacks all the outputs

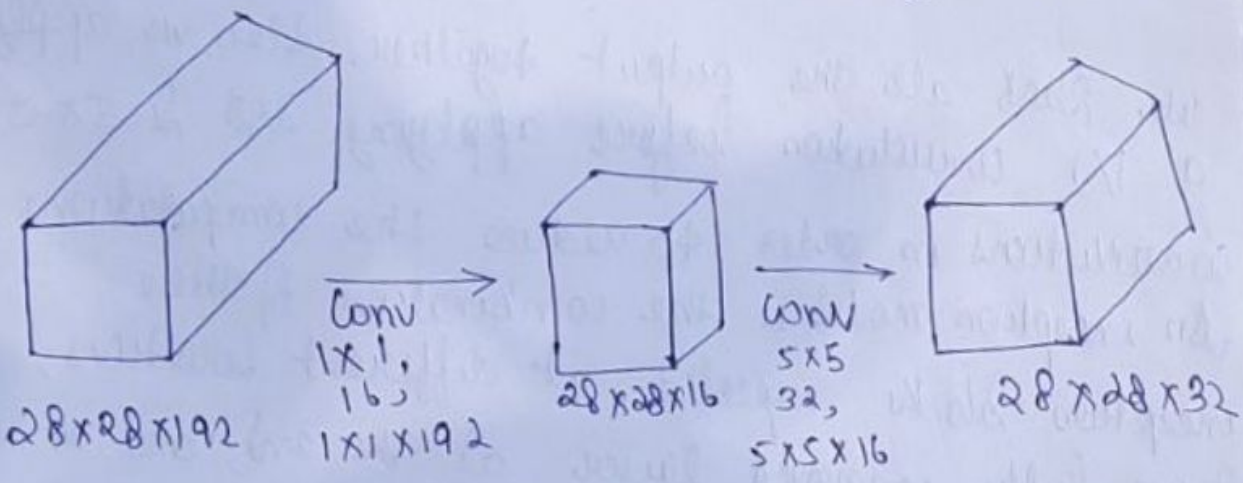


A good question to ask here - why are we using all these filters instead of using just a single filter size 5×5? let's look at how many computations would arise if we would have used only a 5×5 filter on our input.



Number of parameters = 28x28x 32x5 x5 x192 = 120million
So it is has huge computation cost.

Now let's look at the computations at 1x1 convolution and then a 5x5 convolution will give us:

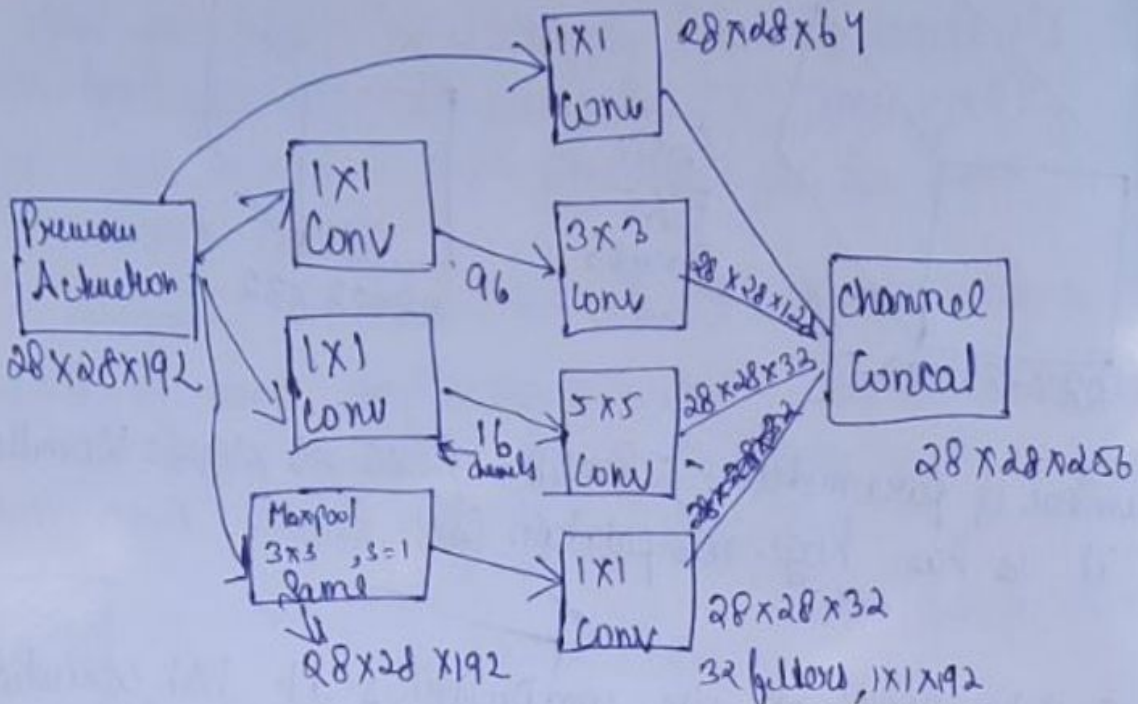


No. of parameter in first convolution = 28x28x16x1x1x192
= 2.4 million

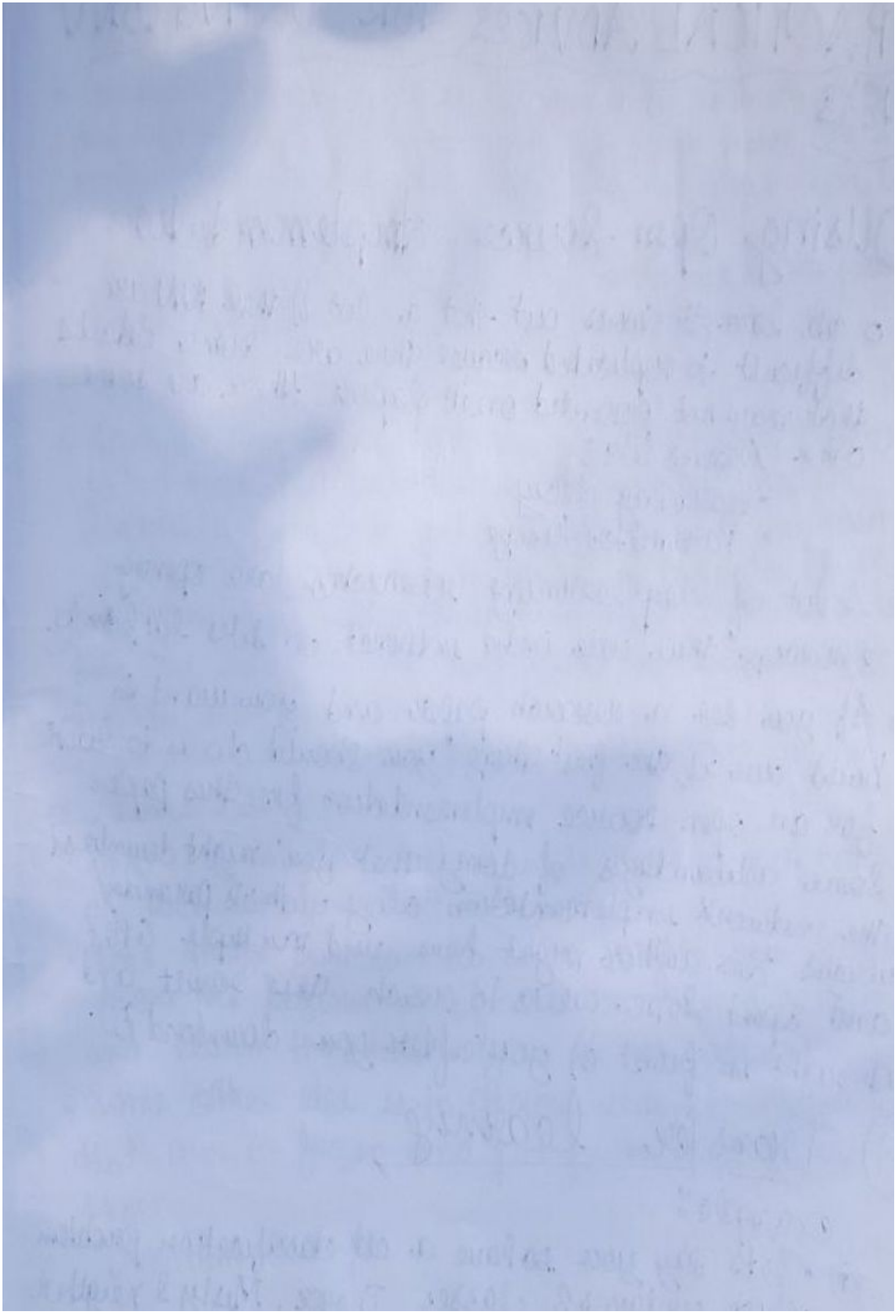
No. of parameter in 2nd " = 28 * 28 x 32 x 5 x 5 x 16
= 10 million

Here we have seen a significant reduction. This is the key idea behind inception

(7) INCEPTION NETWORK



we stack all the output together. Also, we apply a 1X1 convolution before applying 3X3 & 5X5 convolutions in order to reduce the computations. An inception model is the combination of these inception blocks repeated at different locations, some fully connected layer at the end, and a softmax classifier to output the classes.



PRACTICAL ADVICES FOR USING CONVNETS

(1) Using Open-Source Implementation

- It turns out that a lot of these NN are difficult to replicated because there are some details that may not presented on its papers. There are some other reasons like:
  - learning decay
  - Parameter tuning
- A lot of deep learning researchers are opening sourcing their code into internet or sites like Github.
- If you see a research paper and you want to build over it, the first thing you should do is to look for an open source implementation for this paper.
- Some advantages of doing that you might download the network implementation along with its parameter/weights. The author might have used multiple GPUs and spent some weeks to reach this result and it might be front of you after you download it.

(2) Transfer learning

example:

(A) • lets say you have a cat classification problem which contains 3 classes Tigger, Misty & neither
→ cat names

- You don't have much a lot of data to train a NN on those image
- Go online and download a good NN with its weights, the softmax function layer and put your own one & make the network learn only the new layer while other layer weight are frozen
- Frameworks have options to ~~make~~ make the parameters frozen in some layers using trainable=0 or freeze=0
- One of the tricks that can speed up your training is to run the pretrained NN without final Softmax layer and get an intermediate representation of your images and save them to disk. And then use that representation to a shallow NN this can save you the time needed to run an image through all the layers.
  - It's like converting your images into vectors
- Another example
  - what if in the last example you have a lot of pictures for your cats
  - One thing you can do is to freeze few layers from the beginning of the pretrained network and learn the other weights in the network.
  - Some other idea is to throw away the layers that aren't frozen and put your own layers there
- Another example:

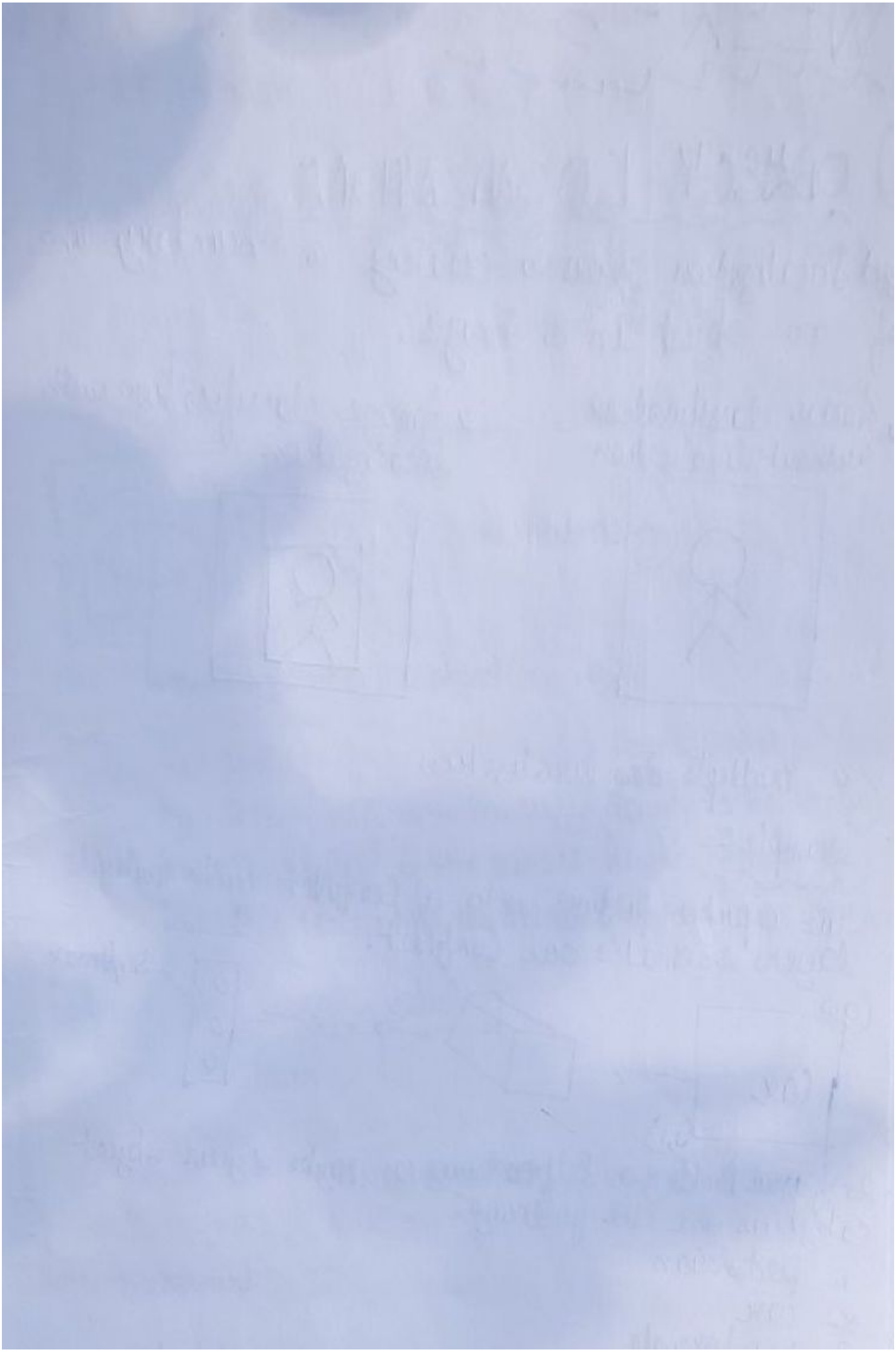
- If you have enough data, you can fine tune all the layers in your pretrained network but don't random initialize the parameter, leave the learned parameters as it is and learn from them

(3) DATA AUGMENTATION

#Same

(4) STATE OF COMPUTER VISION

from online Notes



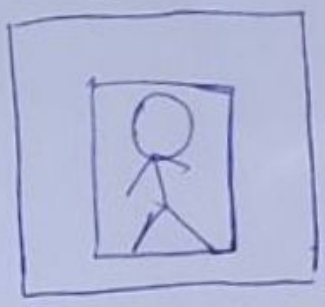
WEEK - 3

① OBJECT LOCALIZATION

- Object localization refers to creating a boundary line of an object in a image.

* Image classification without localization

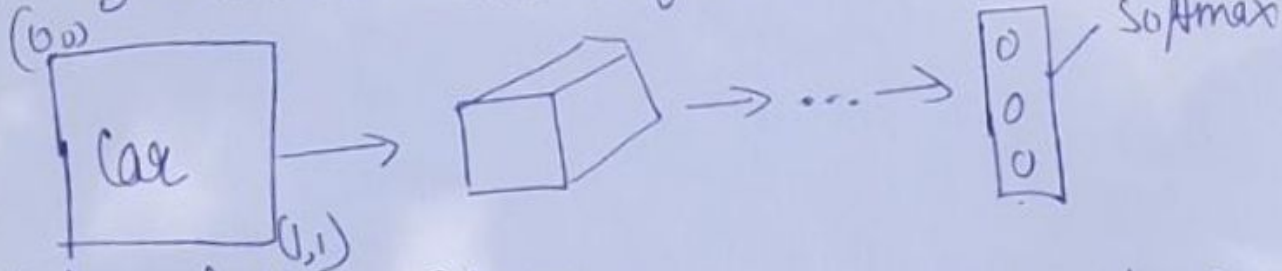
* Image classification with localization



* multiple ~~class~~ localization

- Example :-

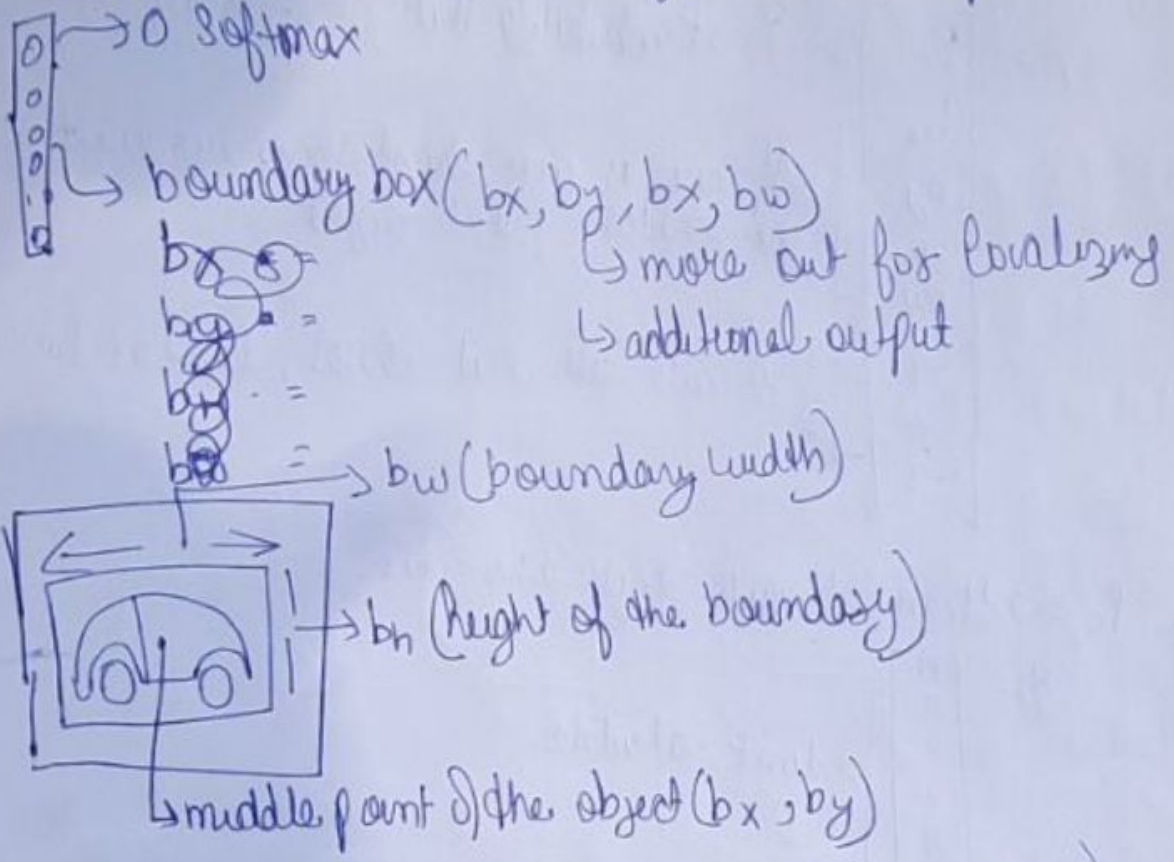
we input a picture into a ConfNet with multiple layers so that's our ConfNet.



So if you building a self driving car maybe your object categories are the followings:

1. Pedestrian
2. car
3. motorcycle
4. background ⇒ when there is nothing to detect

Now for for localizing the object we have to change our N.N. we need to have few more output units



$b_x = 0.5$ (halfway to the right to the image)
$b_y = 0.7$ (70% to the way down to the image)
$b_h = 0.3$ (30% of the overall height of the image)
$b_w = 0.4$ (0.4 of the overall width of the entire image)

Defining the target label y

These are 4 classes

1. Pedestrian
2. Car
3. Motorcycle
4. background

} Need to output b_x, b_y, b_h, b_w class label (1-4)

So, now let's define the target variable Y

$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c - p_c = Probability that there is an any object

b_x, b_y, b_h, b_w → If there is an object then we want the coordinates location of that object

c_1, c_2, c_3 → It will tell is it class1, class2 or class3

If $p_c = 0$ then Y will look like this

$$Y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$ → don't calculate

Let's describe the loss function

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ \cdots + (\hat{y}_8 - y_8)^2 \end{cases}$$ There are 8 output in y

(2) LANDMARK DETECTION

example:

If you are working in a face recognition problem you might want some points on the face like corners of the eyes, corners of the mouth, and corners of the

nose and so on. This can help in a lot of application like detecting the pose of the face.

- Suppose we need 64 landmarks then Y will have 65 output

$$Y = \begin{bmatrix} p_c \\ y_1 \\ y_2 \\ \vdots \\ y_{64} \end{bmatrix} \rightarrow \text{Is there is face If yes then} \qquad Y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \rightarrow \text{No face}$$

(3) OBJECT DETECTION

- To solve the object detection problem we use a technique called sliding window detection
- We move this sliding window on the image to detect the image and moving that sliding ~~win~~ window Box so on ~~diff~~ image then we zoom the image little bit then we move it again then again same process.
- Disadvantage of sliding window is computation time.
- To solve this problem, we can implement the sliding window with a convolutional approach.

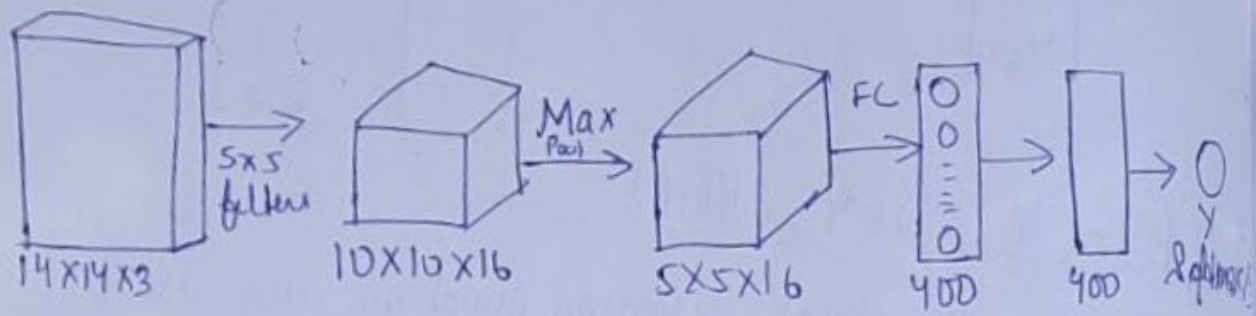
(4) CONVOLUTIONAL IMPLEMENTATION OF SLIDING WINDOWS

→

So, 'to Another' we'll learn how do implement object detection algo convolutionally to make this algo of work fast

Turning FC layer into Convolutional layers

So, let's say that your object detection algorithm inputs 14X14X3 images



Now we'll see how these layers turned into convolutional layer

From Video and Notes

(5) BOUNDING BOX PREDICTIONS

In this we'll learn how do make your predictions to be more accurate

Sometimes None of the Sliding ~~window~~ window boxes really match up perfectly with the position of the car

A good way to get ~~this~~ output more accurate bounding boxes is with the YOLO algorithm.

- Yolo stands for You only look once
- Lets say we have a image of 100 x 100
- We use 3 x 3 grid on the images this will devide our image in a 9 parts like this

- Then we apply • localization algorithm ~~on~~ ~~discussed~~ to each section of the grid

~~Point~~ 01

An Yolo algorithm we'll have problem if we have found more than one object in one grid. Yolo ~~is~~ algorithm has a great speed and a conv net implementation. Yolo uses a single CNN network for both classification & localizing the object using bounding boxes

(6) INTERSECTION OVER UNION

It computes the intersection over union of these two bounding boxes

Intersection over Union (IOU)

$$= \frac{\text{Size of } \square}{\text{Size of } \square}$$

"Correct" if IOU ≥ 0.5

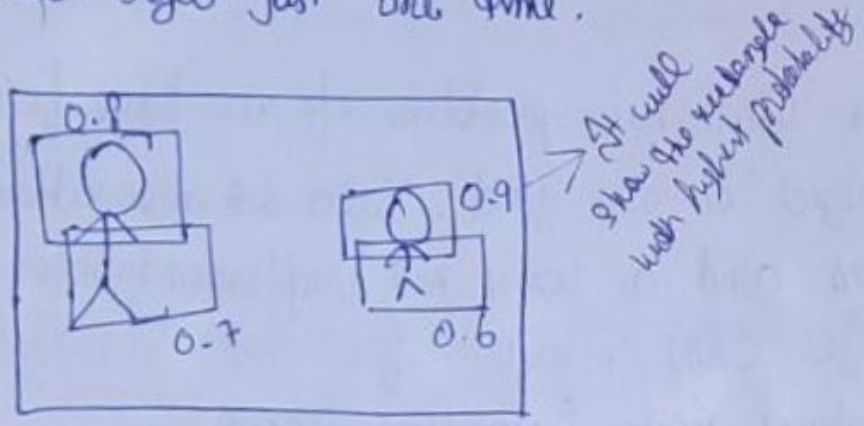
There can change this to 0.6 or 0.7 or any value.

IOU measure of the overlap B/w two Bounding Boxes

- Higher the IOU the better is the accuracy

(7) NON-MAX SUPPRESSION

- One of the problem we have with Yolo algorithm is the it can detect an object multiple times
- Non-max Suppression is used so that Yolo algorithm detect object just one time.



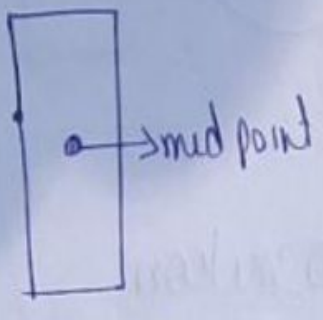
- Non-max Sup. algo.
  - → Y shape should be $[P_c, b_x, b_y, b_h, b_w]$
  - → discard all boxes with $P_c < 0.6$ ↳ or 0.5 any value
  - → Pick the box with largest P_c
  - → discard any remaining box with IOU > 0.5 with the

(8) ANCHOR BOXES → watch video

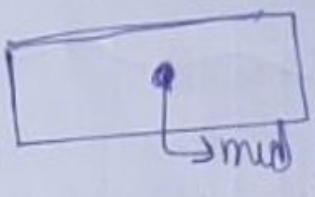
An yolo algorithm, a grid only detects one object. what if a grid cell wants to detect multiple ~~box~~ object?

In this situation we use Anchor Box

Anchor Box 1



Anchor Box 2



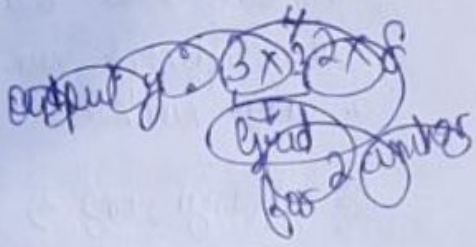
we Generally have 4 or 5 anchor Boxes

with 2 anchor Box, Each object in training image is assigned to grid cell that contains object's midpoint & anchor Box for the grid cell with highest IOU. You have to check where your object should be based on its rectangle closest to which anchor Box.

Example :-

$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ — anchor Box 1; Anchor Box 1

- Anchor Boxes allows your algorithm to specialize, means in our case to easily detect wider images or taller ones.



with video

⑨ YOLO ALGORITHM

WEEK-4

(1) What is face recognition

Working of face

(2) ONE SHORT LEARNING

- It refers to when a recognition system is able to recognize a person face from just from one image of a person

Learning a "similarity" function

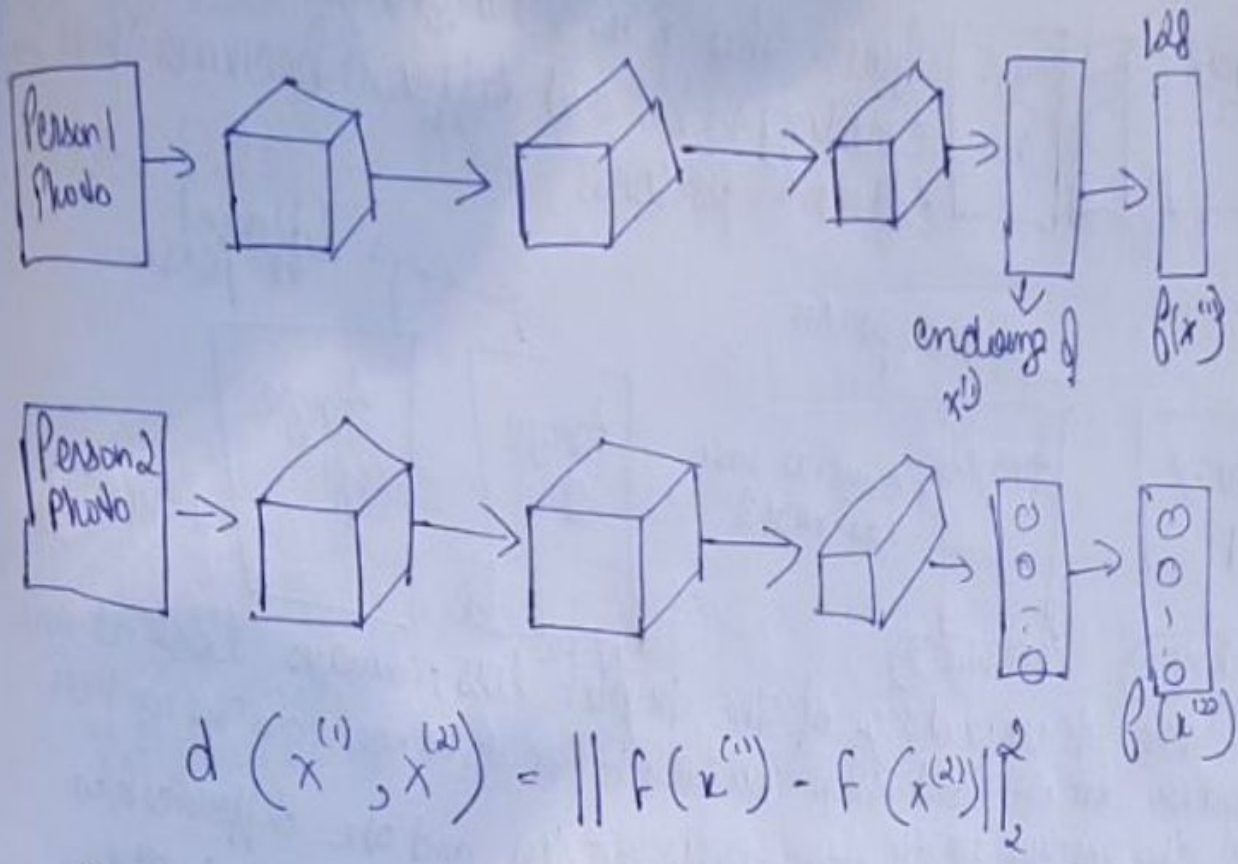
d(img1, img2) = degree of difference b/w images

So if the two images are of the same person, you want this output to be a small number and if 2 images are of two different people then you want it to output a large number.

If $d(img1, img2) \leq \tau$ → threshold called tau (same person)

$d(img1, img2) > \tau$ (different person)

(3) SIAMESE NETWORK



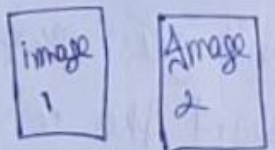
$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

So this idea of running two identical, convolutional NN works on two different inputs and then comparing them, it is called Siamese NN. architecture
These 2 NN ~~has~~ has same parameter.

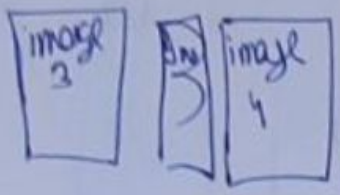
(4) TRIPLET LOSS

To apply the triplet loss, you need to compare pairs of images

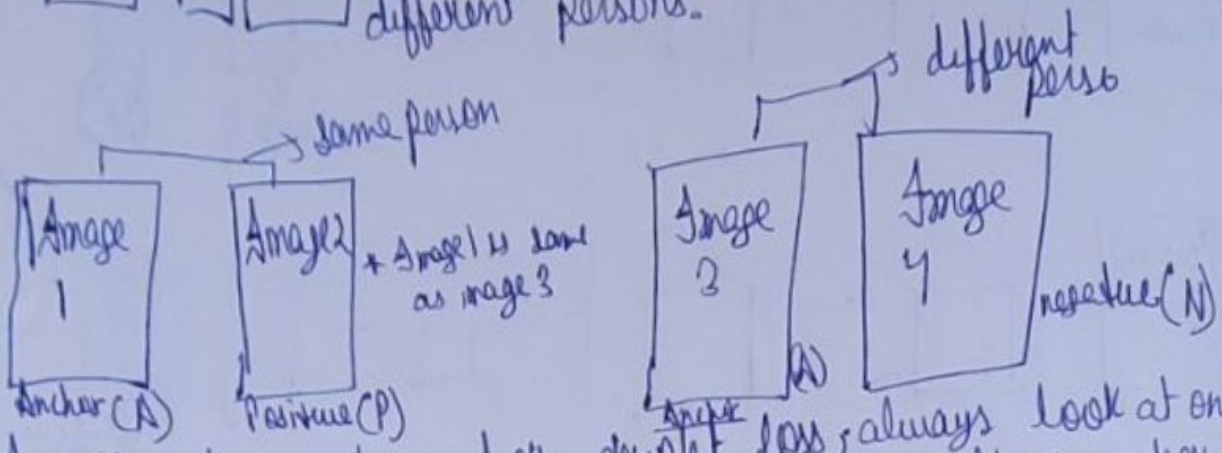
Triplet loss is one of the loss functions we can use to loss functions we can use to solve the similarity distance in a Siamese network.



Given this pair of images, you want their encoding to be similar because these are the same person.



Given this pair of images you want their encoding to be quit different because these are different persons.



In the terminology of the triplet loss, always look at one anchor image and then you want to distance b/w the anchor and the positive image

* the difference b/w img1 I2 should be less and the difference b/w image3 and 4 should be more this is what gives rise to the term triplet loss which means you'll always looking at three images at a time (anchor, positive and negative img at time)

So to formulize this what you want is for the parameters of your N.N. of you encoding to have following property which is

$$\underbrace{\|F(A) - f(P)\|^2}_{\text{which is } d(A,P)} \leq \underbrace{\|F(A) - F(N)\|^2}_{\text{which is } d(A,N)}$$

Now if we move the equation

$$\|F(A) - F(P)\|^2 - \|F(N) - F(P)\|^2 \leq 0$$

But we are going to change the equation a little which is one usual way to make sure this is satisfied, is everything equals zero

So, to make sure that the NN doesn't just output zero for all the encoding. So to prevent this what we are going todo is modify (It needs to be quit a bit smaller than zero)

So we add margin (α) here.

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0$$

α → alpha or margin

Triplet loss function

Given 3 picture A, P, N

$$L(A, P, N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

Triplet Cost function

$$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$$

~~Suppose you have a training set of 10,000 pictures with 1,000 different persons~~

#Notice that in order to define this dataset of triplets you do need some pairs of A & P. Pairs of pictures of the same person.

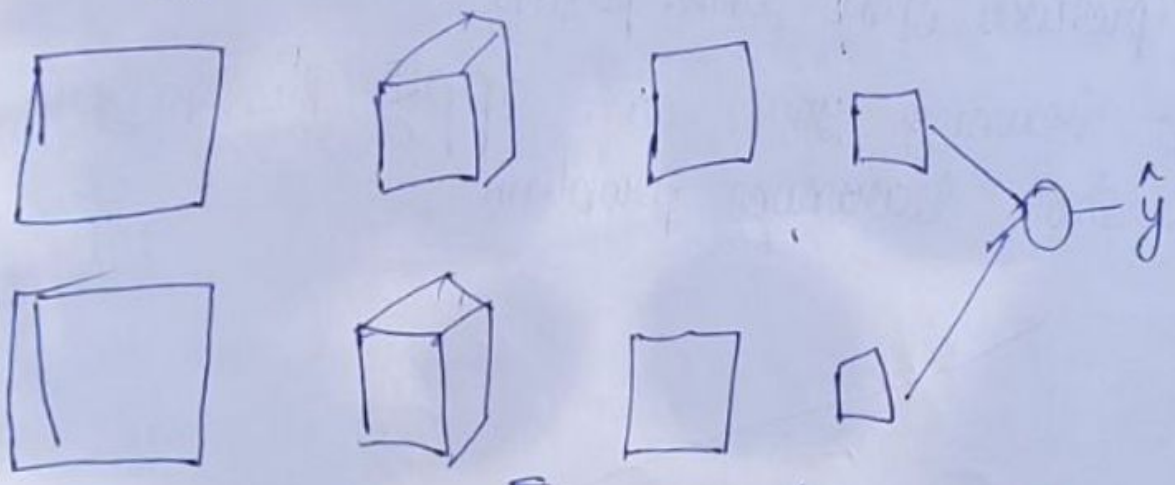
After training you can apply this to your one shot learning problem

⑥ FACE VERIFICATION AND BINARY

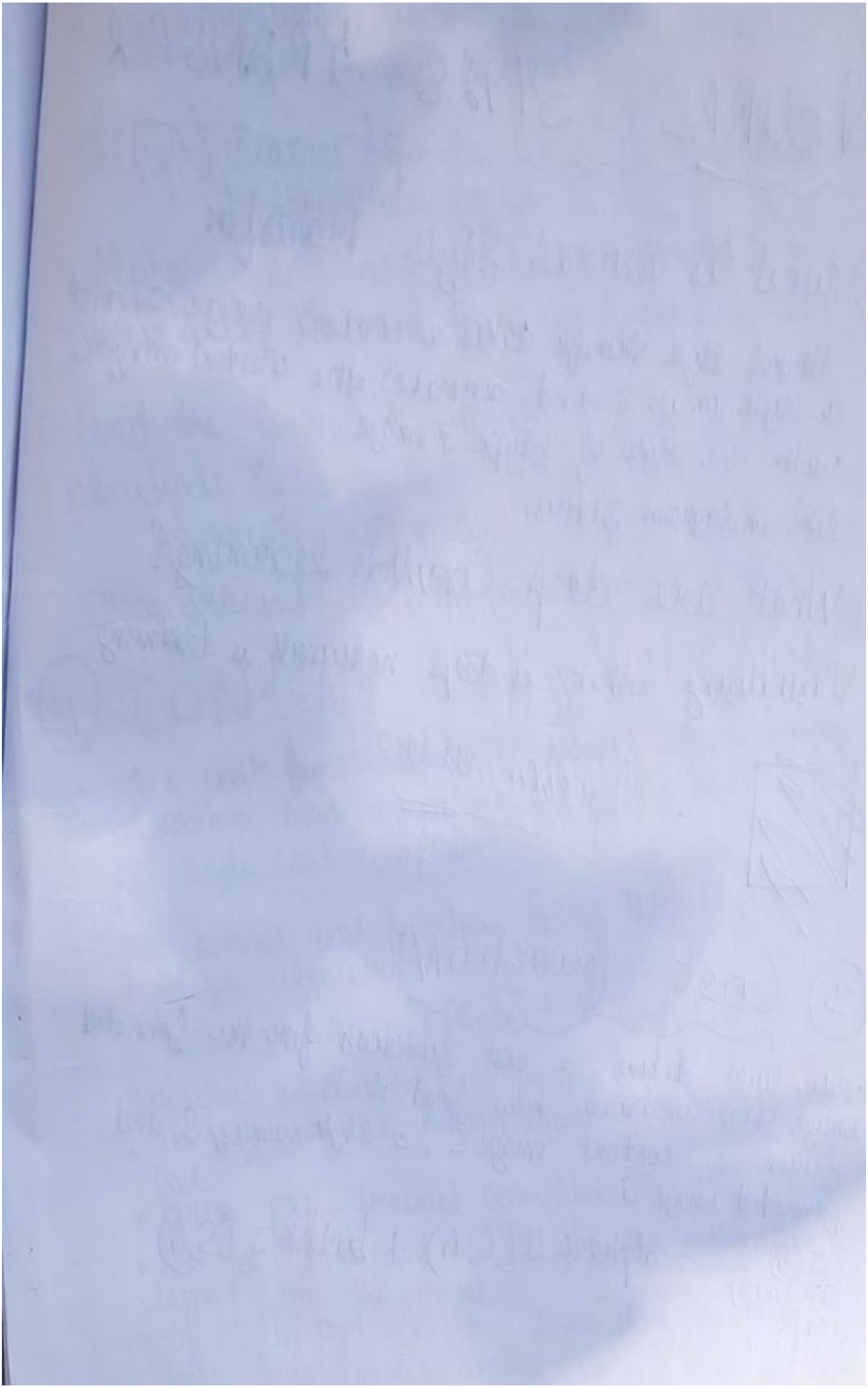
~~face recognition can also be posed as a straight binary classification problem~~

The triplet loss is one good way to learn the parameters of convnet for face recognition. There another way to learn these parameter let me show you how face recognition can also be posed as a straight binary classification problem

Another way to train a NN is to take pair of NN (Siamese Network) and have them compute the embeddings



From nodes online



NEURAL STYLE TRANSFER

(1) What is neural style transfer

Neural style transfer takes a content image C and a style image S and generates the content image G with the style of style image

like instagram filters.

(2) What are deep ConvNets learning?

Visualizing what a deep network is learning

from index

(3) Cost function

- We will define a cost function for the generated image that measures how good it is.
- Given a content image C, a style image S and generated image G.

$J(G) = \text{alpha} * J(C, G) + \text{beta} * J(S, G)$

J(G) cost function

$J(C, G)$ measures how similar is the generated image to the content image

$J(S, G)$ measures how similar is the generated image to the style image

Alpha & Beta are relative weighting to the similarity to and these are hyperparameter.

- Find the generated image G
  - i) Initiate G randomly
    - example : 100 X 100 X 3
  - ii) Use gradient descent to minimize $J(G)$

(4) CONTENT COST FUNCTION

The cost function of the Neural style transfer algorithm has a content cost component and a style cost component.

The overall cost function of the Neural style transfer algorithm

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

So what is content cost function?

- Lets say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet (Eg VGG network)
- let $a^{(C)[l]}$ and $a^{(G)[l]}$ be the activation of layer l on the images
- If $a^{(C)[l]}$ and $a^{(G)[l]}$ are similar then they will have same content

~~a J(i,j) at a layer l = 1/2~~

$J_{context}(i,j) = \frac{1}{2} \| a^{[l][i]} - a^{[l][i]} - a^{[l][j]} \|^2$

(5) SIMPLE COSH FUNCTION

#5 SEQUENCE MODELS

WEEK - 1

① WHY SEQUENCE MODELS

examples of sequence models are :
- ↳ Speech recognition (from audio to text)
- ↳ Music generation
- ↳ Sentiment classification
- ↳ DNA Sequence analysis
- ↳ Machine translation (one language to other)
- ↳ Video activity recognition
- ↳ Name entity recognition (find person name from sentence)

② NOTATION

X: Harry Potter and Hermione Granger invented a new spell

lets say we want to find the names in this sentence (name entity recognition)

Now we need a model to output Y which has one output per input word

y : 1 1 0 1 1 0 0 0 0 (sequence of 9 words)

Now, the input is the sequence of nine words. So, eventually we're going to have some set of features to represent these 9 words.

X : Harry Potter and Hermione Granger invented a new spell
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ - - - - $x^{<t>}$ - $x^{<9>}$.

We use $x^{<t>}$ to index into the positions in the sequence.
$T_x = 9$ (length of input)

Y : 1 1 0 1 1 0 0 0 0
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<9>}$

$T_y = 9$

Note: T_x & T_y can be different

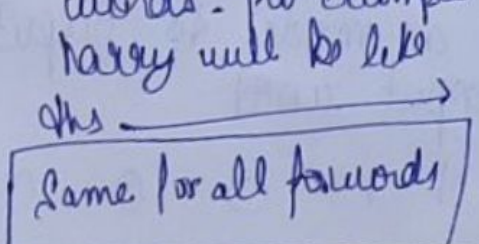
$x^{(i)}$ = i training example
$x^{(i)<t>}$ = is the sequence of training example.
$T_x^{(i)}$ = input sequence length for training example i
Similarly $y^{(i)}$, $y^{(i)<t>}$, $T_y^{(i)}$.

To represent a word in the sentence the first thing you do is come up with a vocabulary (called dictionary)

Vocabulary

a	1
aaron	2
and	367
-	-
Harry	4075
Potter	6830
-	-
Zulu	10,000 or any no.

What you can do is then use one hot representations to represent each of these words. For example the representation for Harry will be like this →



Same for all words

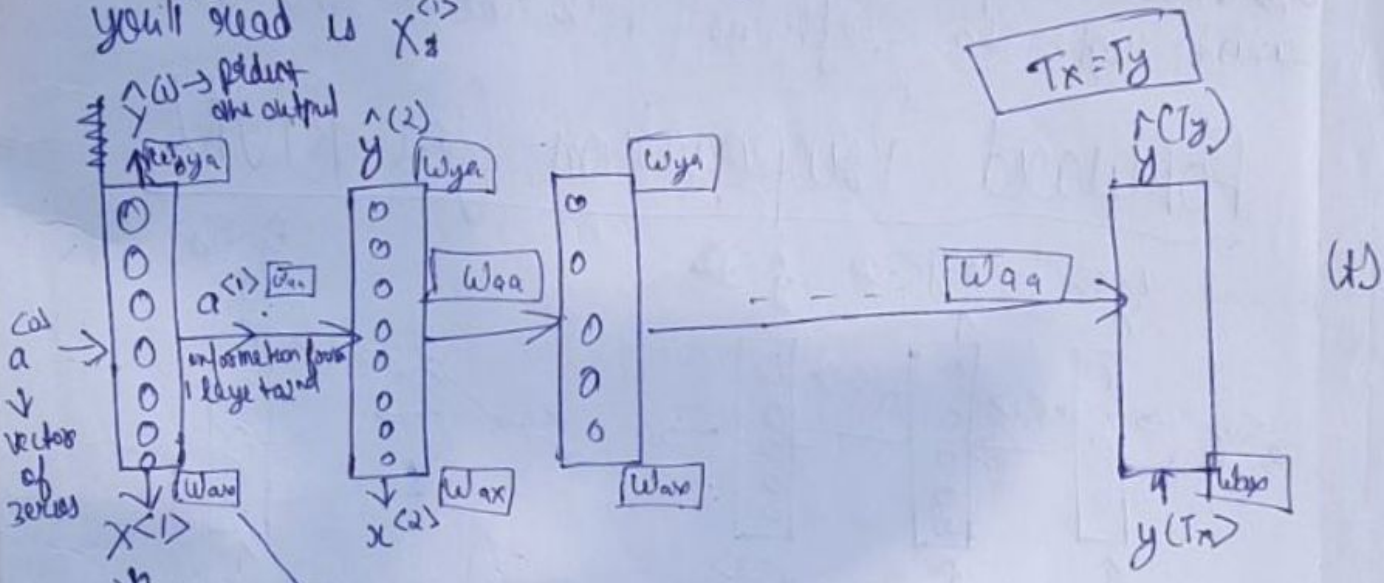
(3) RECURRENT NEURAL NETWORK

One thing you could do is try to use a standard NN for this task. But it will have problem then:

→ Input, outputs can be different lengths in different examples
- Doesn't share features learned across different position of text.

So RNN don't have these advantages

→ If you are reading from left to right then the first you'll read is $X_1^{\langle 1 \rangle}$



There are three weight matrices here

① W_{ax}: (No of hidden neurons, n_x)

② W_{aa}: (No of hidden neurons, No of hidden neurons) ← Horizontal connection

③ W_{ya}: (No of hidden neurons, n_y) → output predictions.

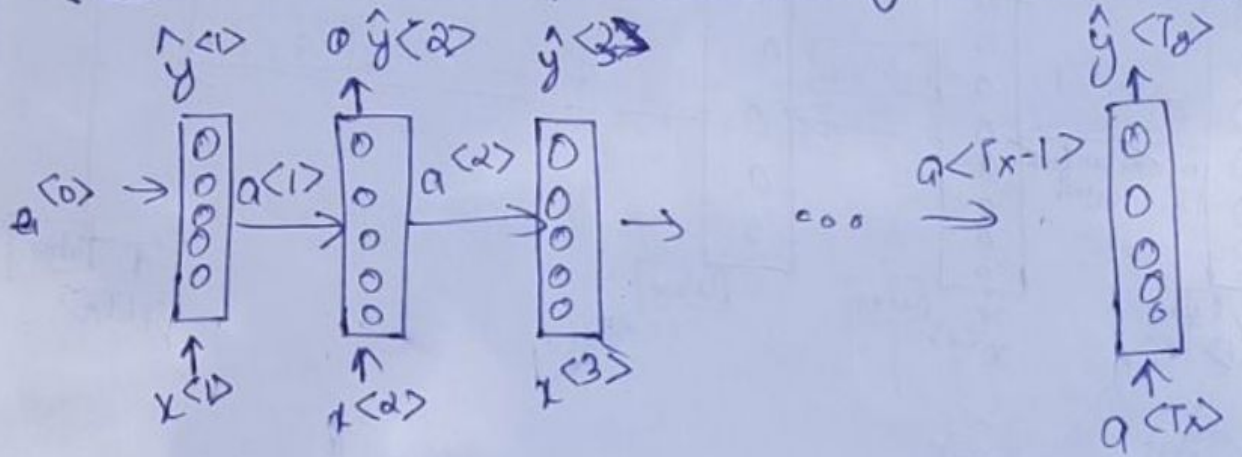
For prediction of $\hat{y}^{<3>}$ it gets information from x^1, x^2 and x^3

One weakness of RNN is that it only uses the information that is earlier in the sequence to make a prediction (left to right). for our sentence. It can ~~can~~ detect Harry is a name then it will output at x, but full name is Harry potter so thats the problem

X: Harry Potter & Hermione Granger invented a new spell.

~~do a note example~~ So we also need to go from ~~and~~ right to left we'll talk about this later

Forward Propagation for RNN



$$a^{<1>} = g(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$$

$$\hat{y}^{<1>} = g(W_{ya}a^{<1>} + b_y)$$

~~W~~ activation function are often tanh or Relu

$$a^{<t>} = g(W_{aa}a^{(t-1)} + W_{ax}X^{(t)} + b_a)$$

$$\hat{y}^{(t)} = g(W_{ya}a^{<t>} + b_y)$$

$$a^{\langle t \rangle} = g(W_a [a^{(t-1)}, x^{(t)}] + b_a)$$

W_a = 100 $\left[W_{aa} \mid W_{ax} \right]$ ← 10,000 →

← 100 → taking together

If $W_{aa} = (100, 000)$

$W_{ax} = (100, 10,000)$

then $W_a = (100, 10100)$

and what does mean $[a^{t-1}, x^{(t)}]$?

just take 2 vectors & stack them together

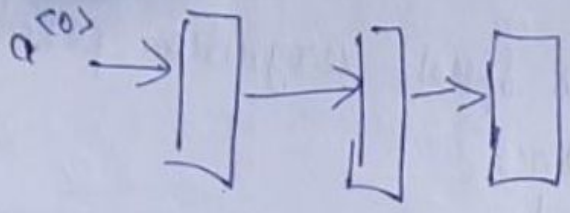
$$[a^{(t-1)}, x^{\langle t \rangle}] = \begin{bmatrix} a^{(t-1)} \\ \hline x^{(t)} \end{bmatrix} \begin{matrix} \uparrow 100 \\ \downarrow \\ \uparrow \\ \downarrow 10000 \end{matrix} \quad \uparrow 10100 \downarrow$$

$$[W_{aa} \mid W_{ax}] \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = W_{aa} a^{(t-1)} + W_{ax} x^{(t)}$$

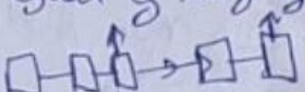
4) BACKPROPAGATION THROUGH TIME

(5) DIFFERENT TYPES OF RNNs

① Many to Many



* In this no. of input = no. of output
But sometimes they are not equal eg language translation



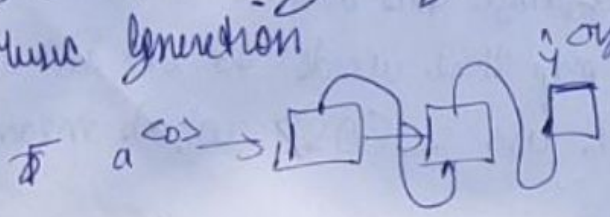
This architecture we have discussed before

② Many to one
Used in Sentimental Analysis

The output will be 1-5 like stars in Reviews

③ One to one
This is like standard N.N

④ One to many
Music generation



6) LANGUAGE MODEL AND SEQUENCE GENERATION

What is language model

Let's say we are doing a speech recognition problem and someone say a sentence

- The apple and pair salad
- The apple and pear salad.

Here pear & pair sounds exactly the same so how would a speech recognition application choose from the two. That's where the language model comes in let's see how.

P(The apple & pair salad) = 3.2×10^{-13}
P(The apple & pear salad) = 5.7×10^{-10}

> with these probabilities the 2nd sentence is much more likely by over a factor of 10 to the 3 compared to the sentence.

What the language model does is it estimates the probability of that particular sequence of words.

$$P(y^{<1>}, y^{<2>}, \ldots y^{<T_y>})$$

To train RNN we need training set.

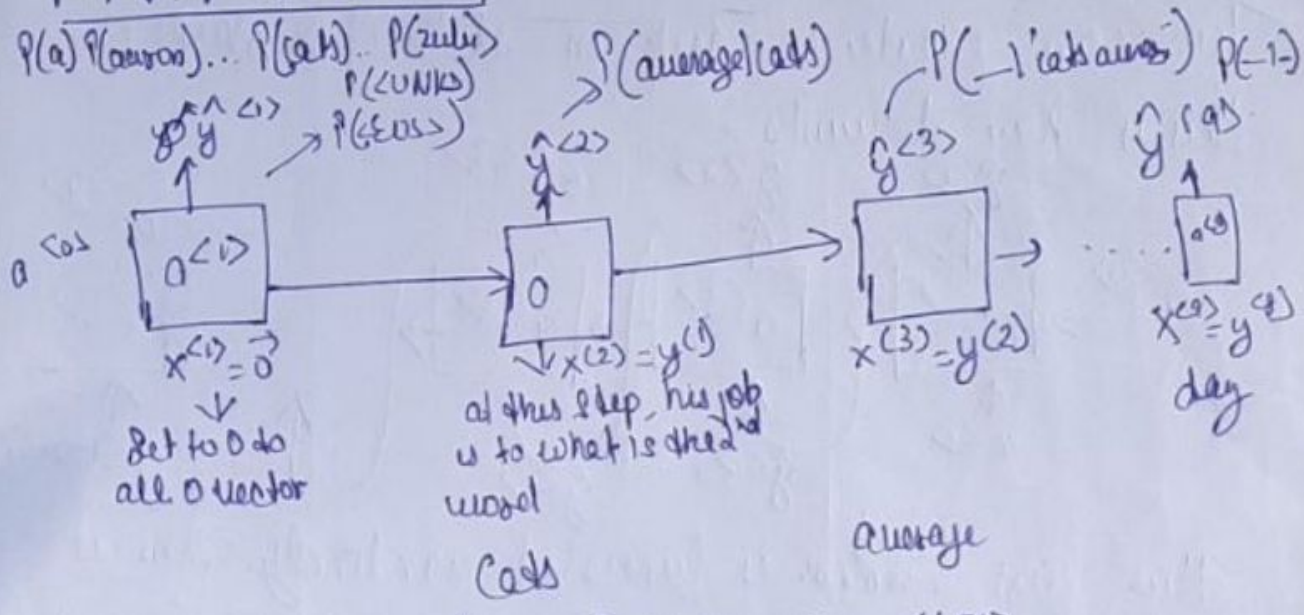
Training set : large corpus of english text

Suppose we have a sentance → Cats average 15 hour of sleep a day.

The first thing we'll do is tokenize this sentence which means we'll form a vocabulary and map these words to one hot vector. At the end of the sentence we use <EOS>. which means end of sentance

So what if our sentence has a word which is not in our ~~vocabulary~~ vocabulary then we assign those word a unique token UNK (Unknown words)

RNN MODEL



→ cats average is Raws sleep a day. <EOS>

and so on thus RNN learn to predict one word at a time going from left to right.

The loss function is defined by cross-entropy loss:

$$\mathcal{L}(y^{<t>}, \hat{y}^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

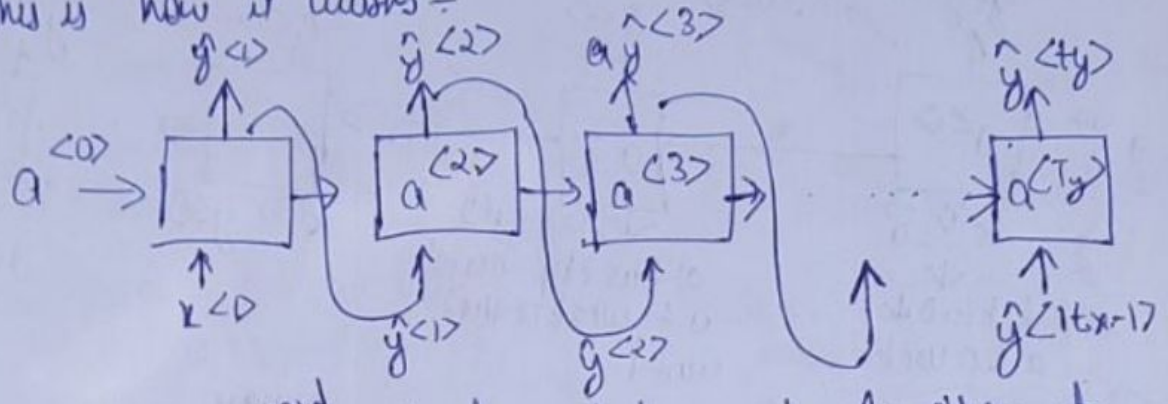
i is for all elements in the corpus.

t - for all timesteps.

(7) SAMPLING NOVEL SEQUENCES

It refers to getting to sample words from a trained model of RNN. like you trained a model of Shakespeare text and you want text out of this model. You can create a text in style of Shakespeare.

This is how it works :-



The first word is generated randomly then it find the probability of next word the word with maximum probability is selected (from the vocabulary) this is how it is done.

It will keep going until we get the <EOS> token. we can reject any <UNK> token.

(8) VANISHING GRADIENTS WITH RNN.


2018
6

) GATED RECURRENT UNIT (GRU)

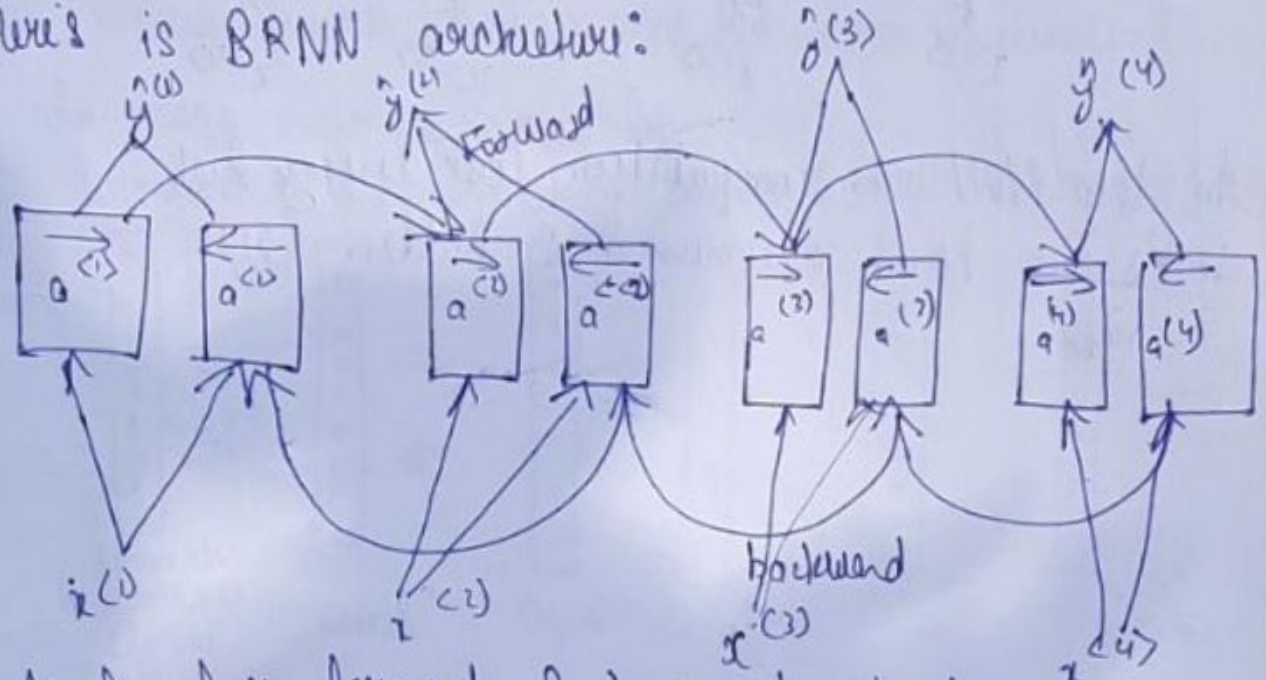
(10) LONG SHORT-TERM MEMORY (LSTM)

(11) BIDIRECTIONAL RNN

~~[illegible]~~

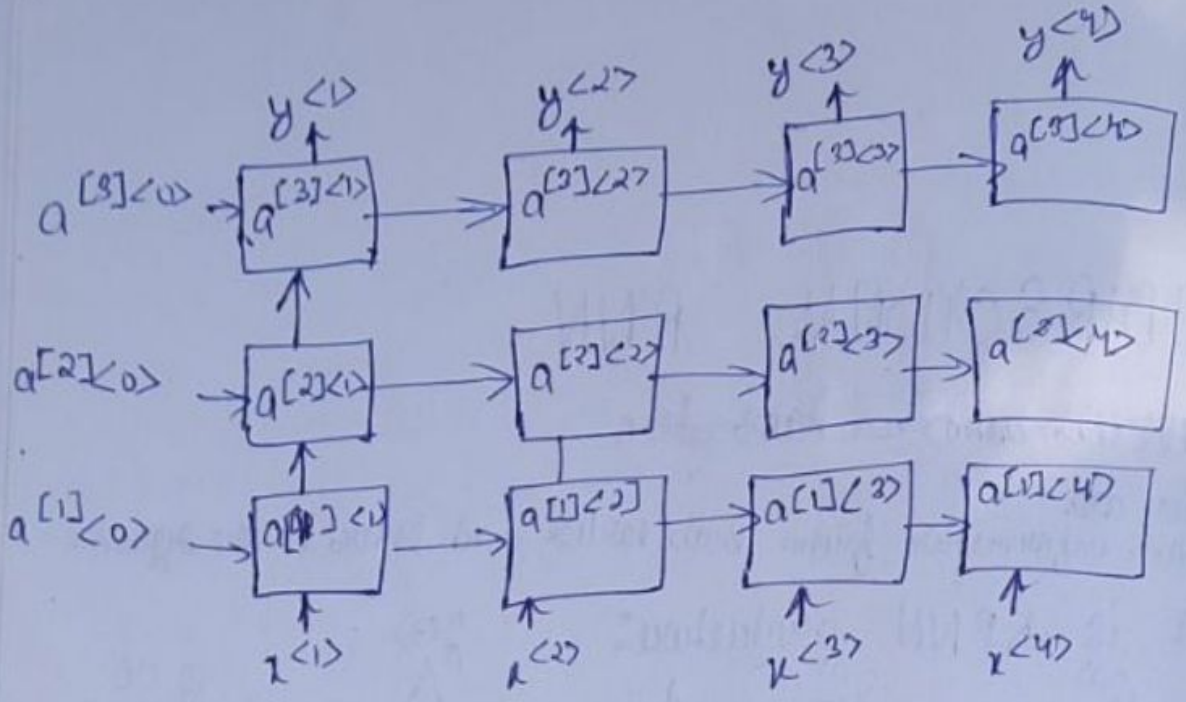
At take information from both earlier and later in the sequence

Here's is BRNN architecture:



- At has both forward & backward info. going
- Note, that BIRNN is an cyclic graph.
- The block here can be RNN, LSTM or GRU
- An live speech recognition if you use BIRNNs you will need to wait for the person who speaks to stop to take the entire sequence. and then make you predictions

⑫ DEEP RNNs



In deep RNN the computation cost is very high that's we have maximum of 3 hidden layer

WEEK - 2

[NLP is revolutionized by DL]

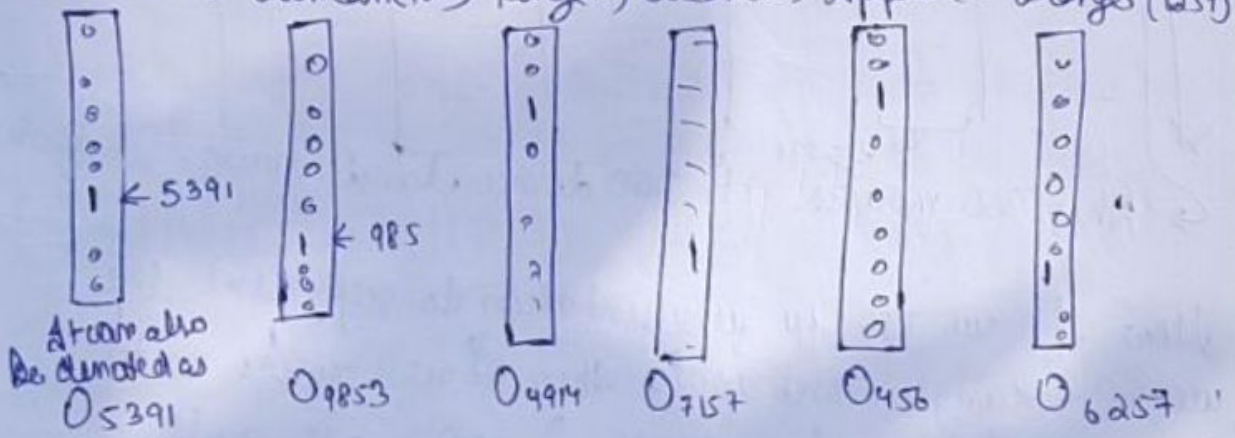
① WORD REPRESENTATION

Word embeddings which is a way of representing words. In this our algorithms automatically understand analogies like : man is to woman, as King is to Queen

we represent words using a one hot vector as discussed.

An image example would be.

Man (5391) Woman (9853) King (4914) Queen (7157) Apple (456) Orange (6257)



One of the weakness of this representation is that it treat a word as a thing that itself and it doesn't allow an algorithm to generalize across words

- Example : "I want a glass of orange ____". (a model will predict that next word is juice

But what if we change "orange" to "Apple"? then a model can't predict a ~~word~~ next word "juice" because it was trained orange not apple.

So, now it doesn't know that somehow apple and orange are much similar than king & orange or green or orange

So, instead of one hot presentation, we can use featurized representation.

WORD EMBEDDING

	(5391) Man	(9853) Woman	(4914) King	(7157) Queen	(456) Apple	(6257) Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	0.02
food	0.04	0.001	0.002	0.01	0.95	0.97
⋮	⋮	⋮	⋮	⋮	⋮	⋮

300

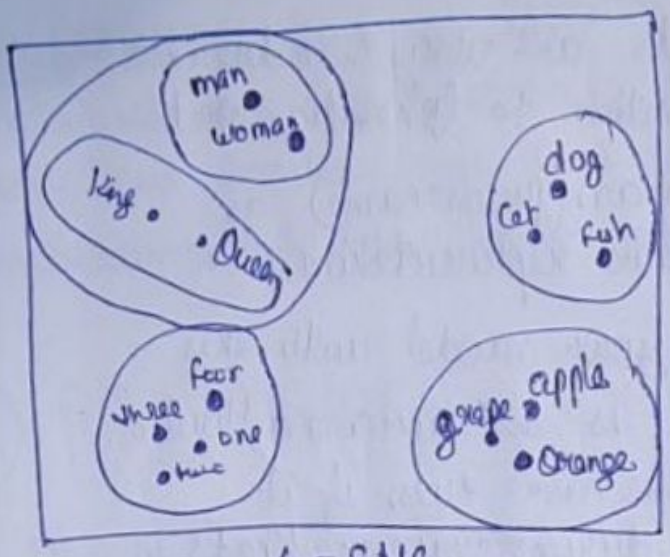
e 5391

This 300 means it is 300 dimensional (having 300 features)

Now if we use the representation to represent the words orangle and apple then we'll notice that the representations for orange & apple are quite similar

Visualizing word embeddings

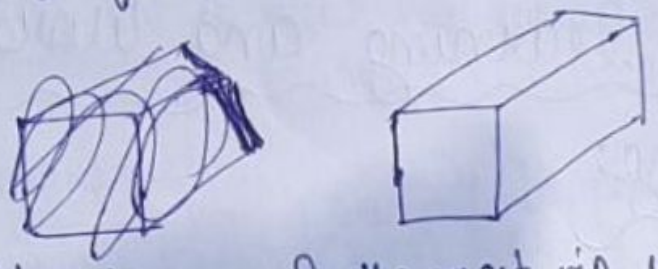
If we are able to learn a 300D feature vector or 300D embedding for each words, one of the popular things to do is to take 300D data and embed it to 2D space (so that we can visualize) one common algorithm for this is t-SNE



t-SNE

man and woman are group together
King & Queen
& man, woman, king, queen group together.

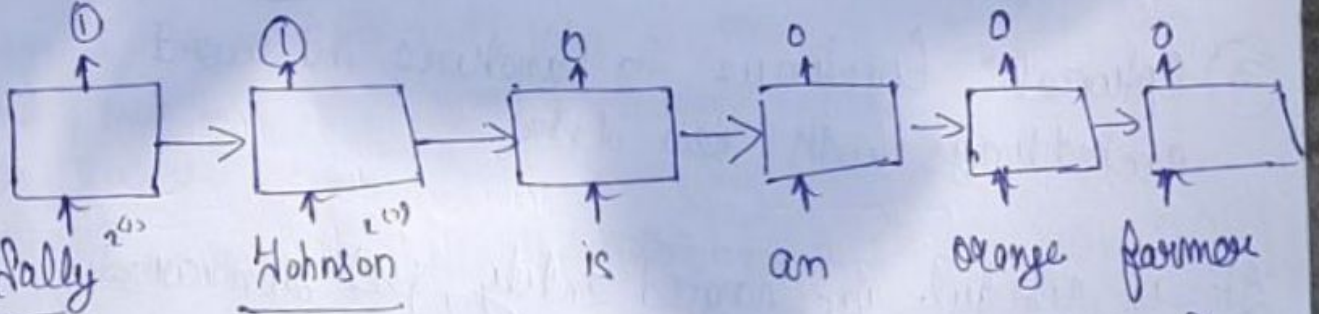
Word embedding for this example is 300D space like this



word embedding is one of the most imp. ideas in NLP

(2) USING WORD EMBEDDING

Given example (for name entity recognition problem)



If we now see a new input (On our trained model above)

Robert lin is an apple farmer

knowing that orange and apple are very similar will make it easier for you algorithm to generalize that Robert Lin is also a human (person's name) as apple and orange have near representation.

Now if you have tested your model with this sentence "Mahmoud Badry is a durian cultivator" the network should learn the name even if it hasn't seen the word durian during training. That's the power of word representation

(c)

Transfer learning and word embeddings

(1) learn word embeddings from large text corpus. (or download pre-trained embedding online)

(2) Transfer embedding to new task with smaller training set. (either than using a 10,000 dimensional vector you can now use 300D vector)

(3) Optional: Continue to finetune the word embeddings with new data.

It is useful for named entity, text summarization, etc.

In face recognition we learned about Siamese network architecture in which we encode each face into a vector and then check how similar are those vectors here encoding of embeddings have similar meaning.

(3) PROPERTIES OF WORD EMBEDDINGS

One of the most fascinating properties of word embeddings is that they can also help with analogy (similar) reasoning. While analogy reasoning may not by itself the most important NLP application, but it might help convey a sense of what these word embeddings can do.

word embedding table:

	Man (5391)	woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
age	0.03	0.02	0.70	0.69	0.03	-0.02
food	0.09	0.01	0.02	0.01	0.95	0.97

e_{man} e_{woman} e_{king}

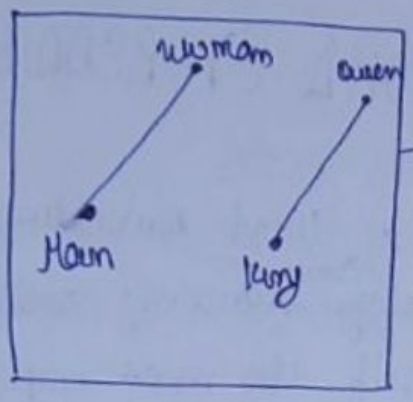
Man is to woman and king is to ____ ? we can find out in the word embedding table.
lets subtract e_{man} from e_{woman} (vectors)
will get $e_{man} - e_{woman} = [-2 \ 0.01 \ 0.01 \ 0.08]$
(we can also write this in words)

Similar for King & Queen

$e_{king} - e_{woman} = [-2 \ 0 \ 0 \ 0]$ (this is approx)
(we can also write this in words)

To find analogy we have to compare man, woman with king & queen

$e_{man} - e_{woman} \approx e_{king} - e_{queen}$? (≈ this mean close to)



- This vector represents the gender difference
- difference is about the gender in both
- ~~2D~~
- This 2D of 4D vector using t-SNE

To find out we use this formula

- $e_{man} - e_{woman} \approx e_{king} - e$??

For this we have to find word w(woman) that maximizes the similarity b/w e_w compare to $e_{king} - e_{man} + e_{woman}$

$$= \underbrace{Sim}_{\text{Similarity}}(e_w, e_{king} - e_{man} + e_{woman})$$

It turns out that e_{queen} is the best solution here solution here that gets the similar vector.

The most similarity function is called Cosine Similarity

$$Sim(U, V) = \frac{U^T V}{\|U\|_2 \|V\|_2}$$

T = transpose

$U = e_w$

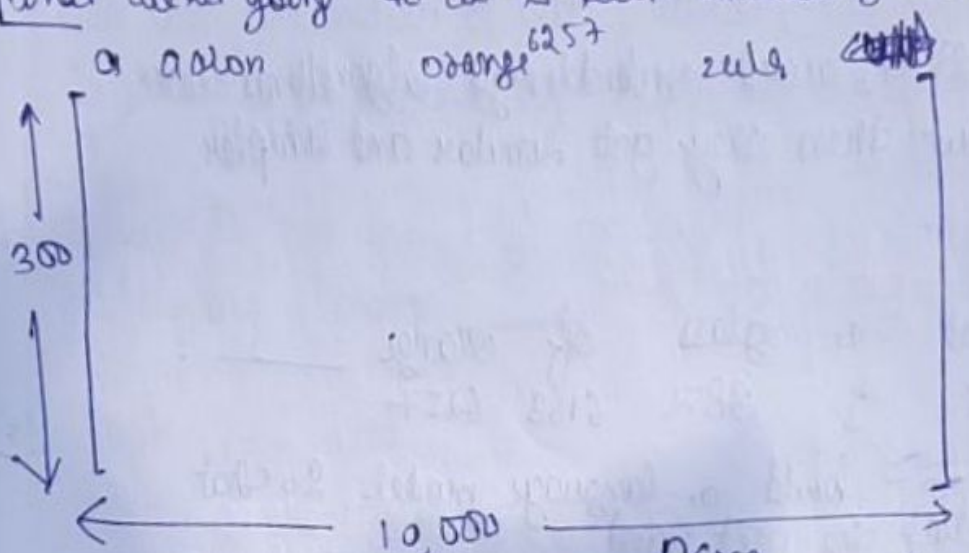
$V = e_{king} - e_{man} + e_{woman}$

(4) EMBEDDING MATRIX

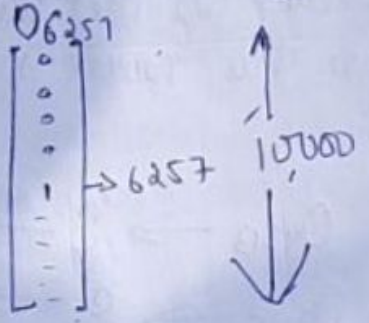
When you implement an algorithm to learn a word embedding

Let's say we are using 10,000-word vocabulary

What we're going to do is learn embedding matrix E,



Orange will be represented as



Embed matrix

$E \cdot O_{6257} = [\ ]$ then this will be are 300 D vector

$E = (300, 10{,}000)$

$O = (10{,}000, 1)$

$(300, 1)$

$= e_{6257}$

Our goal is to learn our embedding matrix E. We initialize E randomly and we use gradient descent to learn all the parameter of this 300 by 10,000 dimensional matrix

LEARNING WORD EMBEDDINGS: WORD2VEC & GLOVE

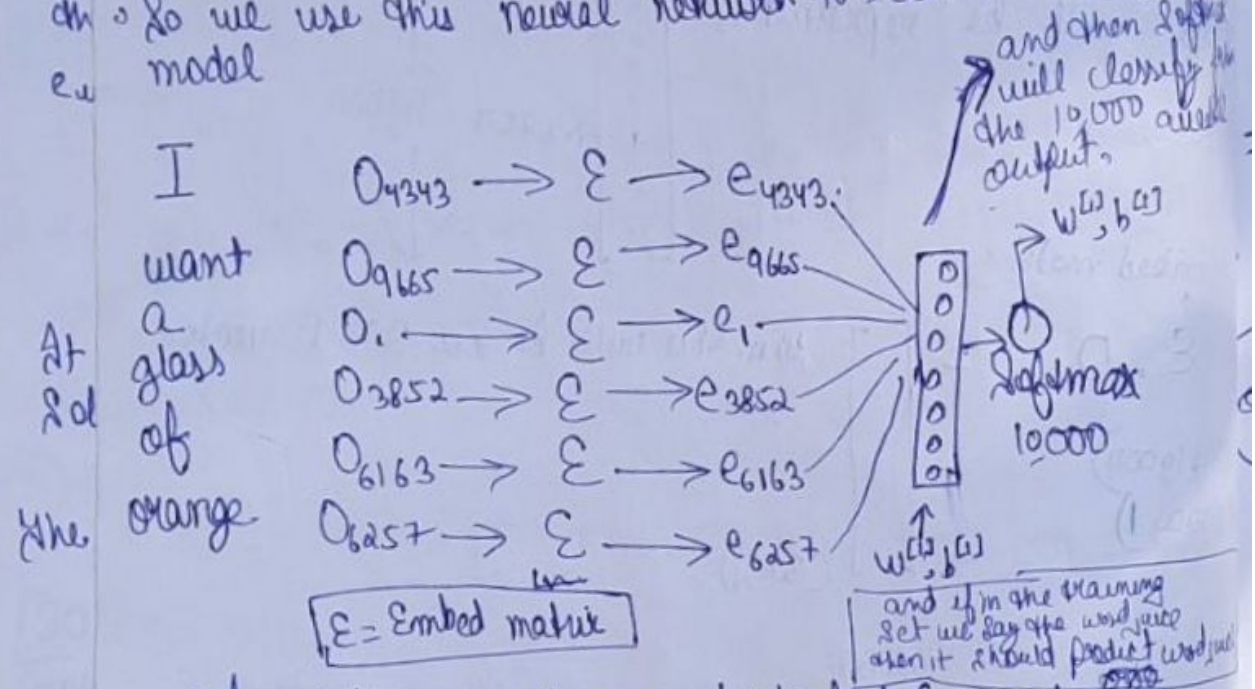
① LEARNING WORD EMBEDDING

At the start, word embeddings algorithms were complex but then they got similar and simpler.

example:-

I want a glass of orange ——.
4343 9665 1 3852 6163 6257

- we want to build a language model so that we can predict the next word.
- So we use this neural network to learn the language model



- In this example we took last 6 words that falls before the word that we want to predict.

Suppose we have a example: "I want a glass of orange juice to go along with my cereal"

To suppose we want to predict word "juice" here we can do this by many ways.

a) last 4 words
we use a window of last 4 words (4 is hyperparameter)
"a glass of orange" and try to predict the next word from it.

b) 4 words on the left or right
a glass of orange ____ to go along with

c) last one word
which is 'orange'

To summarize, the language modeling problem poses as machine learning problem where you input the context ~~like it~~ like the last four ~~to~~ words and predict some target words. And posing that problem allows you to learn good word embeddings.

(2) WORD2VEC

lets talk about Skip-grams first-

for example:
our sentence → "I want a glass of ~~one~~ orange juice to go along with my cereal".

An skip-gram we choose context and target.

An Skip gram, what we are going to do is come up with few context to target errors to create our Supervised learning ~~algorithm~~ problem. So rather than having the context be always the last four words or last end words immediately before the target word, what we'll do is pick a word to be context word.

The target is chosen randomly based on a window with a specific size

Context	Target	How far
orange	juice	+1
orange	glass	-2
orange	my	+6

we want to use this learning problem to learn good word embeddings

* This is not an easy learning because learning within +10 or -10 words which is hard

Word2Vec model: → It has 2 algorithm in it ① CBOW ② Skip gram

lets learn basics about word

→ ~~It~~ Here ~~was~~ model learns c to t (Context to target)

* we get e_c by $E.O_c$

$$P(t|c) = \hat{y}$$

~~In~~ this algorithm use cross entropy loss function

$$P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

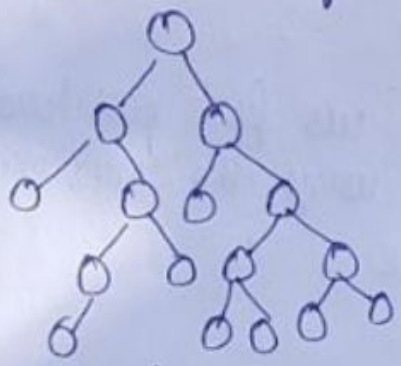
θ_t = parameter associate with output t

loss function for Softmax will be used which is

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

y as a one hot vector

- Here we are summing 10,000 number which correspond to the number of words in our vocabulary.
- If the no. of our vocabulary is very large say 1 million the computation will be become very slow. But we can solve the computation problem by using "Hierarchial Softmax classifier" which is tree classifier.



But in practice the hierarchial softmax classifier doesn't use a balanced tree like the drawn one. Common words are at top which are used commonly and less common are at bottom

How to Sample the context ?

- choose random from corpus
- There are some words common words which dominate other words.

- At uses 2 ideads of learning word embedding
  - ① Skip gram model
  - ② CBOW (Continuous Bag-of-words)

③ NEGATIVE SAMPLING

Negative sampling allows you to do something similar to the skip-gram model, but with a much more efficient learning algorithm. We will create a different learning problem.

- Ex:

Our Sentence: "I want a glass of orange juice to go with my cereal"

Our Sampling

Context	word	target
orange	juice	1
Orange	king	0
Orange	book	0
Orange	the	0
Orange	of	0

We get positive example by using the same skip-gram technique.

To generate a negative example, we pick a word randomly from the vocabulary

Notice that, we got word "of" as negative example although it appeared in the same sentence

(4) GLOVE WORD VECTORS

It refers to Global vectors for word representation

Previously, we were sampling pairs of words context and target words by picking 2 words that appear in close proximity to each other in our text corpus.

So what the Glove algorithm does is, it starts off just by making that explicit.

X_{ij} = # times i appears in the context of j

i → t

j → c

$X_{ij} = X_{ji}$ if we choose a window pair, but they will not equal if we choose the previous words for for example. In Glove they use

a window which means they are equal

X_{ij} is a count that capture. how often do words i and j appear with each other,

So what glove model does it optimize the following

Model

minimize

APPLICATIONS USING WORD EMBEDDINGS

① SENTIMENT CLASSIFICATION

example

X	Y
The dessert is excellent.	★ ●●●●○
Service was quite slow.	●●○○○

One of the problem with the sentiment classification is you might not have a huge labed data set.

RNN is used for solving problem of sentimental Analysis

Sentence: "Completely lacking in good taste, good service, and good ambience"

Here if we do a simple classification for this sentence it won't work. So, we use RNN for this

② DEBIASING WORD EMBEDDING

In this we'll learn the ideas for diminishing or eliminating these forms of bias in word embeddings. Here, Bias doesn't mean the bias variants. it means gender ethnicity, sexual orientation bias.

The problem of bias in word embeddings:

- Man : Woman as King : Queen ✓
- Man : Computer_Programmer as Woman : Homemaker ✗
- Father : Doctor as Mother : Nurse ✗

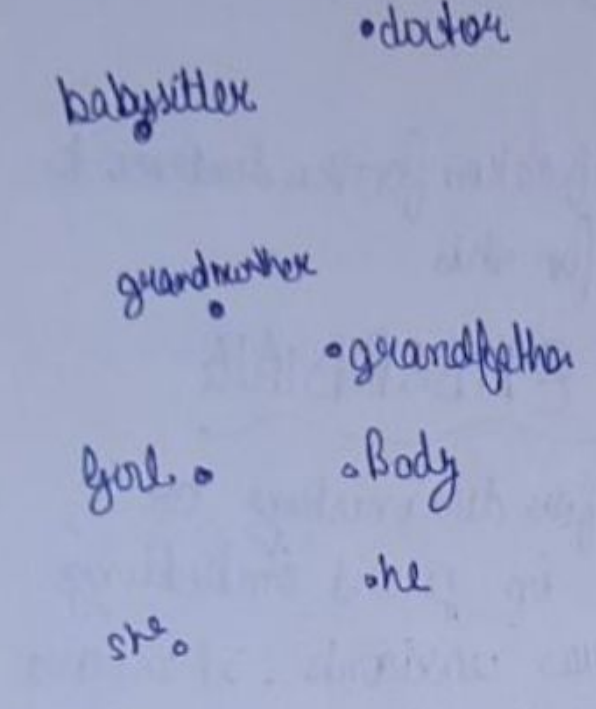
word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text to train the model.

Addressing Bias in word embeddings

So Here's the idea. Lets say we've already learned a word embedding like this -

→

Given these learned word embeddings:

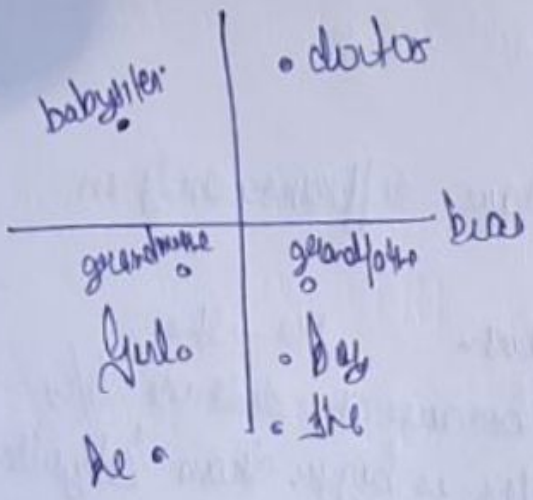


So, first thing we are going to do it is identify the direction corresponding to a particular bias we want to reduce or eliminate

So how do you identify the direction corresponding to the bias? for the case of gender what we can do is take the embedding vector for he and and subtract the embedding vector for she, because the differs by gender

Here are the steps:

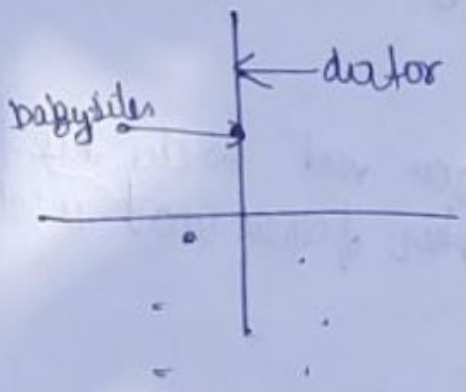
1. Identify the direction.
   - e_he - e_she
   - e_male - e_female
   - =
   - choose some k differences & Average them
   - this will help you to find this



By that we found the bias direction which is 1 Vector and the non-bias vector which is 299 Vector.

b] Neutralize: For every word that not definitional, project to get rid of bias

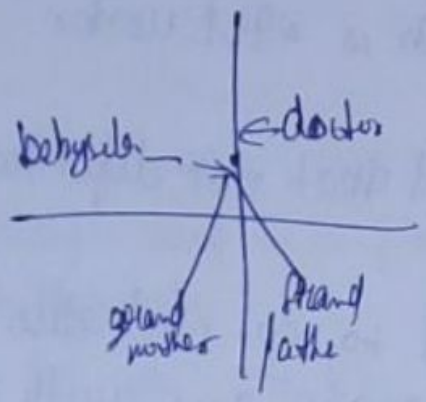
- Babysitter & doctor need to be neutralize. so we project them on non-bias axis with direction of the bias



after that they will be equal in the term of Gender. To do this the authors of the paper trained a classifier to tell the words that needs to be neutralized or not.

9 Equalise parts

- We want each pair to have difference only in gender like
  - Grandfather - Grandmother He - She
- We want to do this because the distance b/w Grandfather & Babysitter is bigger than Babysitter & grandmother



- To do that we move grandfather & grandmother to a point where they will be in the middle of the non-bias axis

- There are some words you need to do this for in your steps. My Number of their word is relatively small