Week #4:- Deep Neural Network

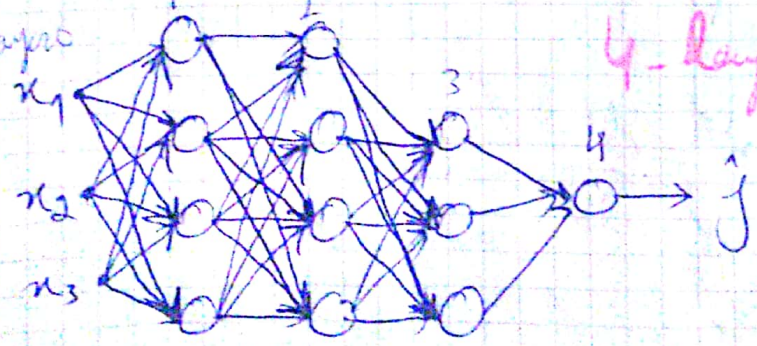
Deep L-layer Neural Network:-

Previously we saw 1 layered network where we implemented the logistic regression and later in week 3 a 2-layered network

→ Remember ⇒ no of layers = total hidden layer plus output layer.

Let's work out some notation that we'll be using here.

e.g here we have a network.



4-layered N.N

$L = 4 \Rightarrow$ (No. of layers).

$n^{[l]}$ = no. of units in layer.

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

$a^{[l]}$ = activations in layer l $\quad n^{[0]} = n_x = 3$

$a^{[l]} = g^{[l]}(z^{[l]})$, $w^{[l]}$ = weights for $z^{[l]}$

$b^{[l]} \Rightarrow$ bias for $z^{[l]}$.

Forward Propagation in Deep network :-

Considering the last example of network we designed. Let's go over how forward propagation will look like for one training example.

given $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ for first layer we can compute

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

the activations for the same layer is given by :-

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

how about layer 2 :-

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

$$a^{[2]} = g^{[2]}(z^{[2]}) \quad \text{and so on.}$$

Until we get to output layer

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

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

Now we have seen it for one training example let's go through it for m training examples via Vectorization.

Vectorization for m training examples:-

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

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

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

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

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

1st training example; mth training ex.

In this process we end up with

$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

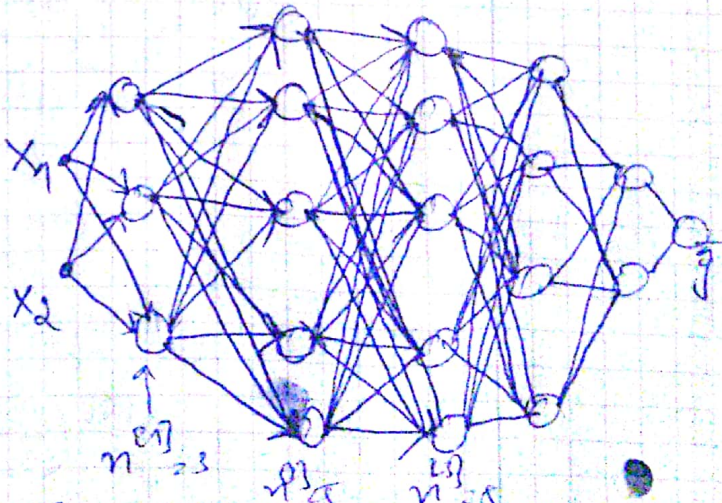
Getting your matrix dimensions right:-

getting the matrix dimensions right is extremely important when implementing your neural network. So working that out on a piece of paper can really save you a lot of headache. Let's see how to do that.

→ Let's say we have a 5-layered neural network

$L = 5$

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



Let's forget the bias term for a while.

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

$(3,1) \quad (3,2)\ (2,1) \quad (3,1)$

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

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

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

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

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

$(5,1) = (5,3)(3,1) \cdot (5,1)$

$w^{[3]} \in \mathbb{R}^{5 \times 5}$

$z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$

$(5,1) \quad (5,5), (5,1) \quad (5,1)$

$w^{[4]} \in \mathbb{R}^{4 \times 5}$

$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$

$(4,1) = (4,5)(5,1) \quad (4,1)$

$w^{[5]} \in \mathbb{R}^{2 \times 4}$

$z^{[5]} = w^{[5]} a^{[4]} + b^{[5]}$

$(2,1) \quad (2,4)\ (4,1) \quad (2,1)$

$w^{[6]} \in \mathbb{R}^{1 \times 2}$

$z^{[6]} = w^{[6]} a^{[5]} + b^{[6]}$

$(1,1) = (1,2)(2,1) \quad (1,1)$

In backward propagation $dw^{[l]}$'s dimension = w's

db's " = b's

In Vectorized version:

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

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

⇒ this is still $(n^{[1]}, 1)$, but it expands into $(n^{[1]}, m)$ by python broadcasting and added into the $W^{[1]}X$ matrix element wise.

$Z^{[1]}, A^{[1]} : (R^{n^{[1]}}, 1)$.

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

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

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

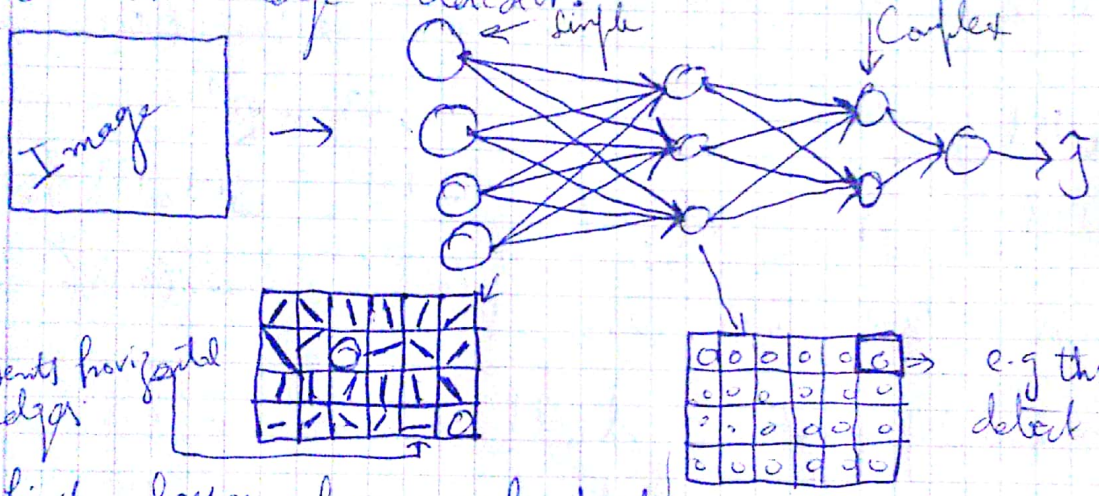
Why deep neural networks work better?

→ Let's go through a couple of examples and try to develop an intuition. Why they might work well?

So, first. What is a deep network computing?

→ If you are building a system of face recognition or face detection, here is what a deep neural network could be doing:-

⇒ You input a face's image and then the first layer could be thought of as may be a feature detector or an edge detector.



Represents horizontal edges

→ e.g this part detect the eye

→ First layer of a neural network may be thought of as edge detector. In this example I am computing what the first 20 hidden units try to compute on this image.

So this hidden unit is trying to figure out where are the edges of that orientation in the Image.

Now we have the detected edges Now the next layer can take the detected edges and form those parts of face Now by taking together those computed parts of face you can take those to detect different types of faces.

→ So, you can think that first layer is being used to compute simple functions like detecting features and later layers are taying to compute more complex functions by taking the previously outputted simple functions.

Main Intuitive takeaway:-

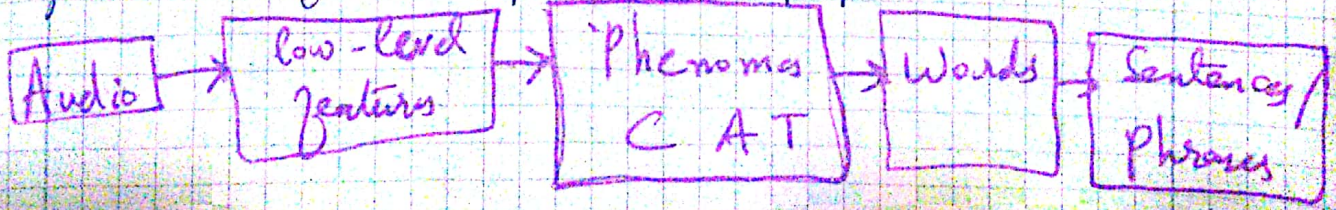
Finding simple things like edges then building them up, composing them together to detect more complex things.

→ This type of simple to complex heirachical representation applies in other types of data rather then just imagery data as well.

e.g. If you want to build speech recognition system

→ If you input an audio clip then may be first level of neural network is being used to compute low level audio features like audio waveform, like is it going up or down or is it white noise or pitch.

→ Then by those low level features you learn to develop some high level things like sounds. In linguistics they are called Phonemes. Composing these sounds together to recognize words and in turn composing words together to form complete sentences / phrases



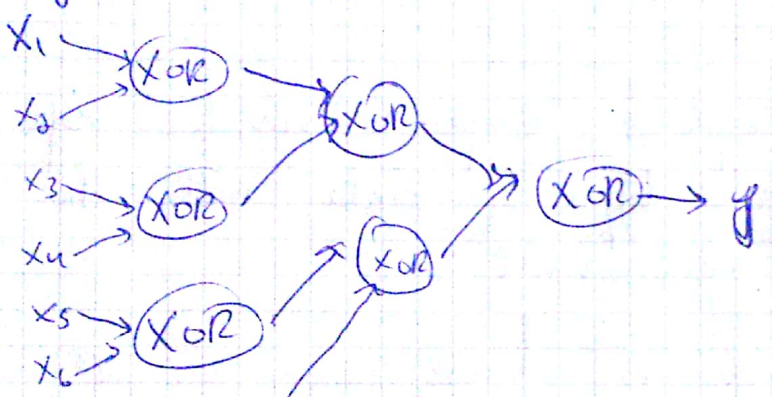
Circuit Theory and Deep Learning:-

Informally:- There are functions you can compute with a "small" L layer deep neural network that shallower networks require exponentially more hidden units to compute.

Let's motivate this by an example:-

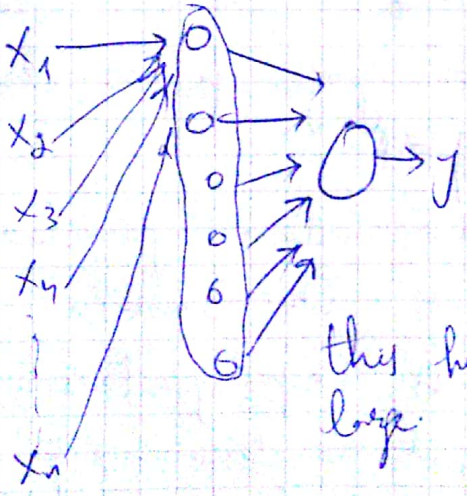
$$x_1 \text{ XOR } x_2 \text{ XOR } x_3 \text{ XOR } \cdots \text{ XOR } x_n$$

One way to do that is to compute it like this:-



So, to compute all this, the depth of the network will be of the order $O(\log(n))$

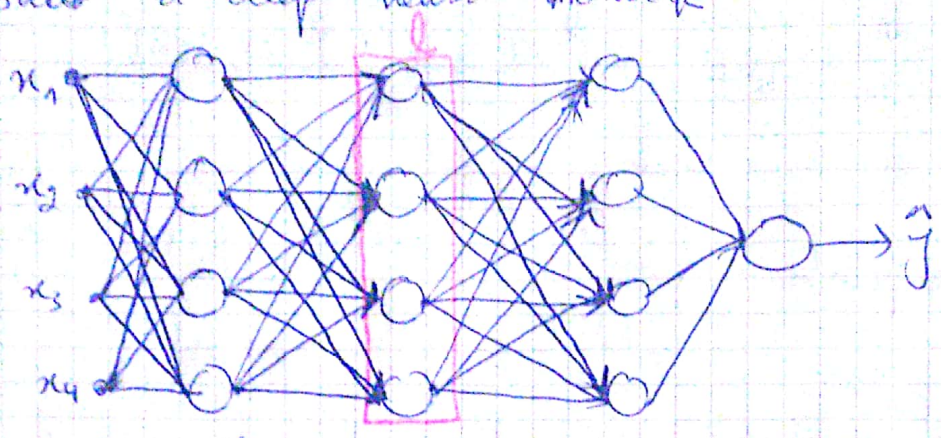
Now with shallower network if we do it like this:-



this hidden layer will be exponentially large. $2^{n-1} \Rightarrow O(2^n)$

Building blocks of deep neural Networks:-

- We have already seen basic building blocks for forward propagation and backward propagation. Let's see how you can take all those building blocks together and build a deep neural network.



- Let's consider the above drawn neural network and focus on rectangled layer

Layer l : $W^{[l]}$, $b^{[l]}$

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

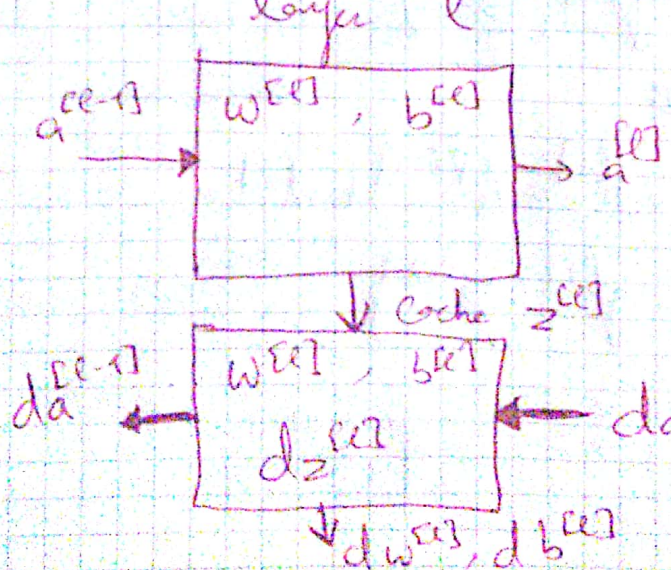
o/p $Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ Cache $Z^{[l]}$

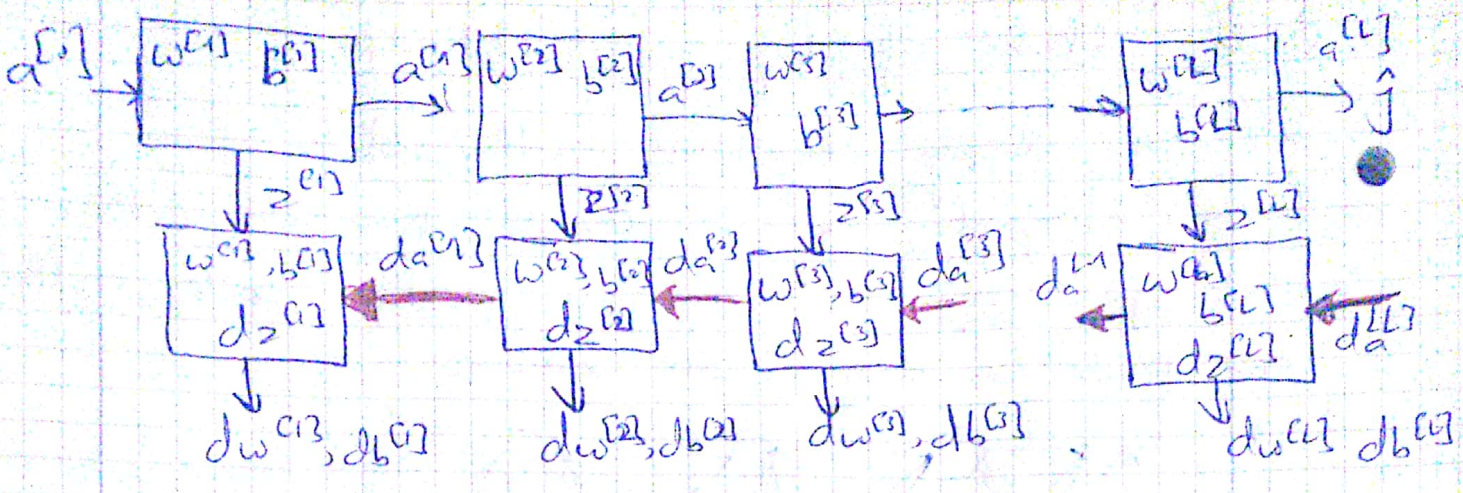
$a^{[l]} = g^{[l]}(Z^{[l]})$

- Backward propagation: I/p $da^{[l]}$, Cache ($Z^{[l]}$)

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

To summarize





Red denotes backpropagation.

$$\text{update} \begin{cases} w^{[l]} = w^{[l]} - \alpha\, dw^{[l]} \\ b^{[l]} = b^{[l]} - \alpha\, db^{[l]} \end{cases}$$

Forward propagation step and backward propagation step:-

Let's see how we can develop code to implement these algorithms. Let's start with forward propagation step.

→ Forward propagation for layer l :-

→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache $(z^{[l]})$ ← $w^{[l]}, b^{[l]}$

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

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

Vectorized Version:

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

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

→ Backward propagation for layer l:-

→ Input $da^{[l]}$

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

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

Let's write the vectorized version:-

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

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

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

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

- Parameters Vs Hyperparameters:-

Being a good neural network developer means that, you not only manage your parameters well, but also your hyperparameters.

So, what are hyperparameters? Let's discuss that.

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

- Hyperparameters: Learning rate α

iterations

of hidden layers

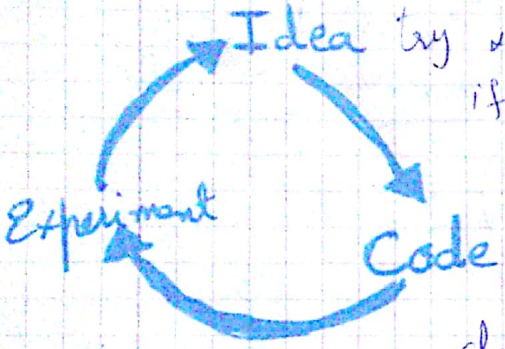
hidden units $n^{[1]}, n^{[2]}, \ldots$

Choice of activation function.

→ as all these things control the values of your parameters W, b so we call them hyperparameters

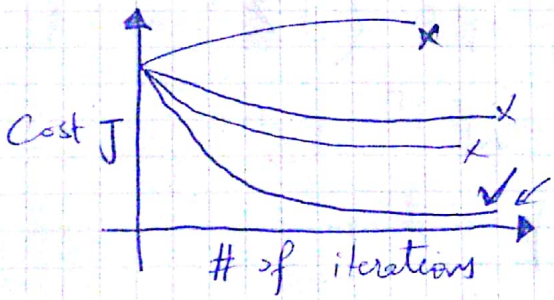
→ later in the course we will see other hyperparameters like momentum, mini batch, size, regularization.

Applied Deep Learning is very empirical process.



try $\alpha = 0.01$ and train the node, later if that doesn't work well, try $\alpha = 0.05$ and other values

One thing you can do is plot different α's Cost against # of iterations.



→ figuring out hyperparameter values at the start of training is very difficult, so you should try different values and go around the above blue cycle like hit and trial and see what works.

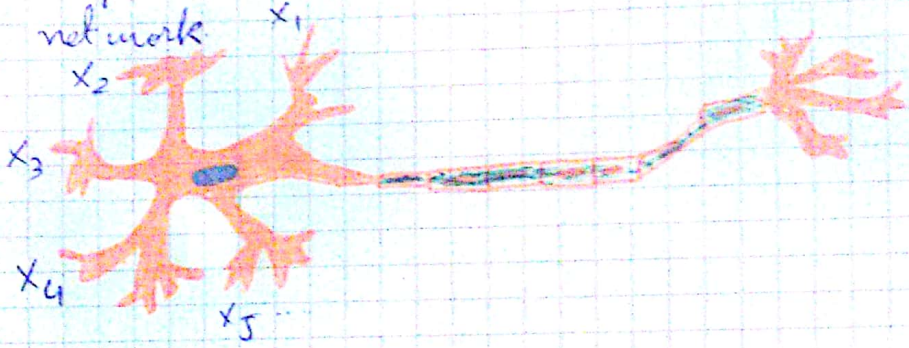
What does Deep Learning have to do with human brain?

In short ⇒ not a whole lot ;)

- When you implement a neural network you implement forward propagation and backpropagation

  the equations gives very less intuition about the whole things relation with the brain

  → If we look at the neuronal cell of human brain structure of that is more or less the same of neural network



- There is very little information today that how one neuron operate or if they use an algorithm like backprop. and forward prop to operate or if there is totally different learning algorithm.