Neural Networks and Deep Learning

Week-4

After our 1st 3 weeks spent in building a solid foundation of how a Neural Networks, we are now ready to take a deep dive to see the functioning of Deep Neural Networks.

We start with a basic nomenclature to understand which layer and unit we are referring to:

L-> # Layers
$n^{[l]}$-> # units in Layer L
$n^{[0]}$ -> n_x (Input)

We have derived the generalised formulas for Forward Propagation and Back Propagation:

Forward Propagation:

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

Input: $A^{[l-1]}$
Output: $A^{[l]}$, Cache: $Z^{[l]}$

Backward Propagation:

$$dZ^{[l]} = dA^{[l]} * g'^{[l]}(Z^{[l]}) \text{ ->①}$$

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

$$db^{[l]} = \frac{1}{m} * np.sum(dZ^{[l]})$$

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

$$dA^{[l]} = W^{[l]T} * dZ^{[l+1]} \text{ ->②}$$

From ① & ②, we get,

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

we get

Input: $dA^{[l]}$

Cache: $dZ^{[l]}$

Output:

- $dA^{[l-1]}$
- $dW^{[l]}$
- $db^{[l-1]}$

In Machine Learning, lots of complexity comes from data and not from code i.e. the reason you get magical results even with few lines of code.

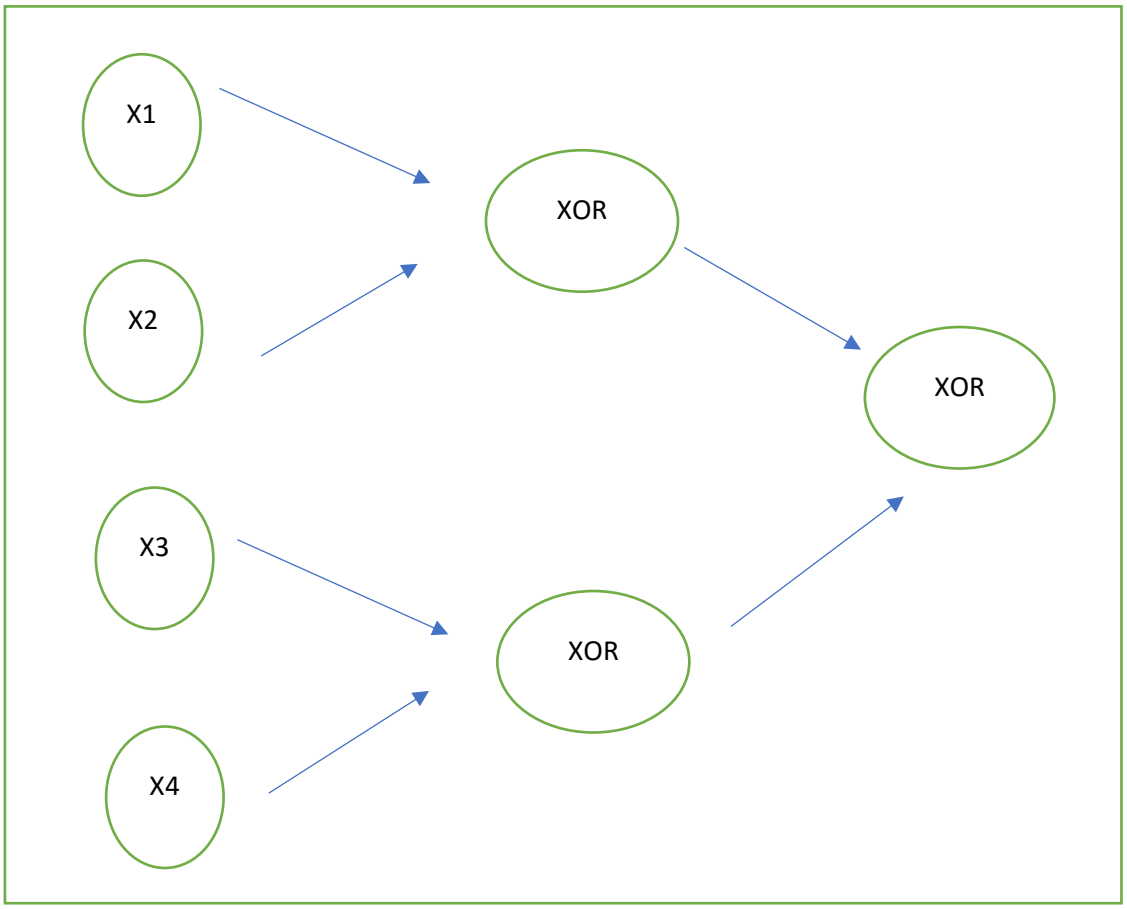
Why do we require deep representations?

To answer this, we shall take example of Face detection wherein in the initial hidden layers we just detect the edges and as we go around we slowly start detecting eyes, ears and nose and then finally, thus signifying the importance of a deep network.

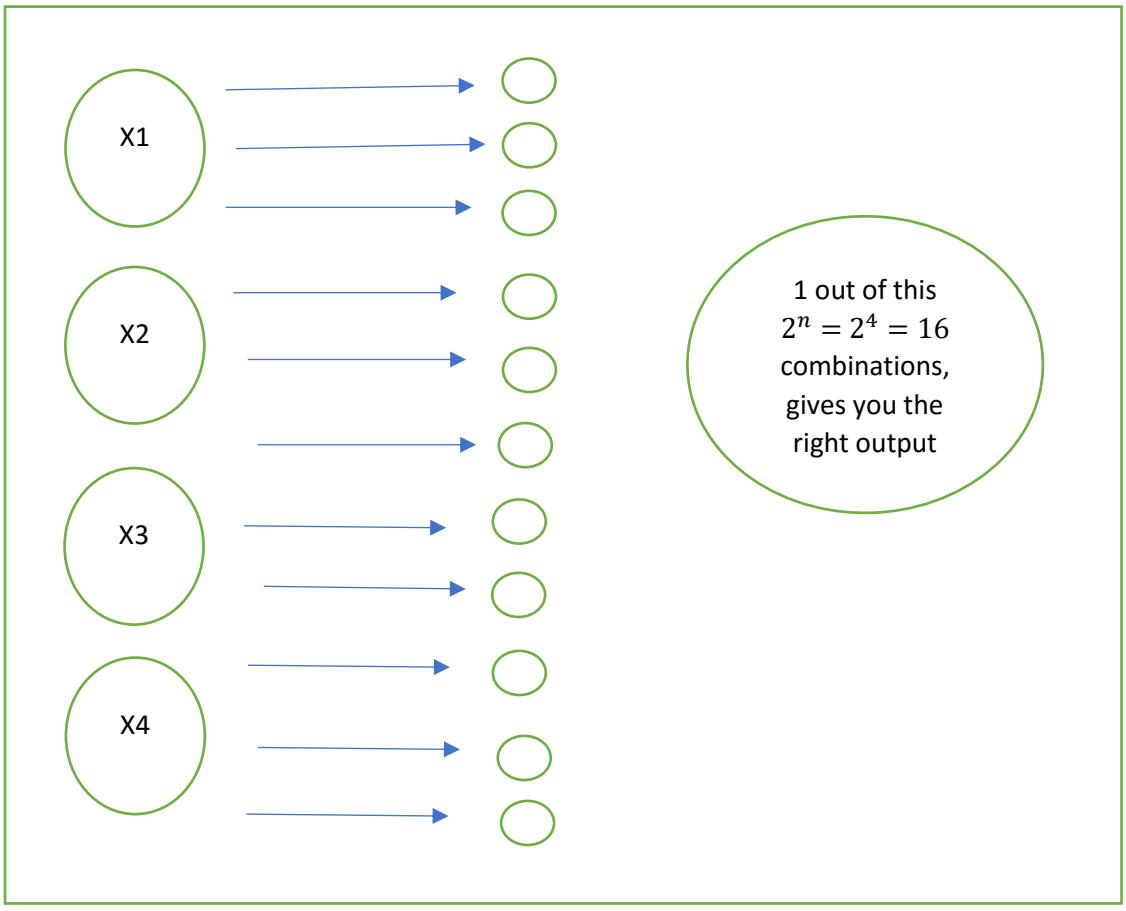
There is a famous correlation between Circuit Theory and Deep Learning:

“There are functions you can compute with a “small” L-Layer deep neural network that shallower Neural Network require exponentially more hidden units to compute.

The following example illustrates the same:



2-Layer Neural Network to perform a XOR Operation



1-Layer Neural Network to perform a XOR Operation

(We had to compute $2^n = 2^4 = 16$ computations to see what the XOR output was

Finally, we see the difference between Parameters and Hyper Parameters:

Parameters:

- $W^{[1]}, b^{[1]}$
- $W^{[2]}, b^{[2]}$

  ……
  ……
  ……
  ……
- $W^{[l]}, b^{[l]}$

Hyper Parameters:

- Learning Rate α
- # Iterations
- # Hidden Units L
- Hidden units $n^{[1]}, n^{[2]}$……$n^{[l]}$
- Choice of Activation Function:
  - Tanh
  - Relu (or) Leaky Relu
  - Sigmoid

https://www.quora.com/What-are-hyperparameters-in-machine-learning

Some of them draw an analogy between Neural Network and functioning of the brain, but this is not accurate as:

- We still don't know what a single neuron in a brain does?
- We don't know how a single neuron in brain learns?

Programming Assignment:

In the programming exercise, we will implement all the building blocks of a neural network and use these building blocks in the next assignment to build a neural network of any architecture we want. After completing this assignment, we:

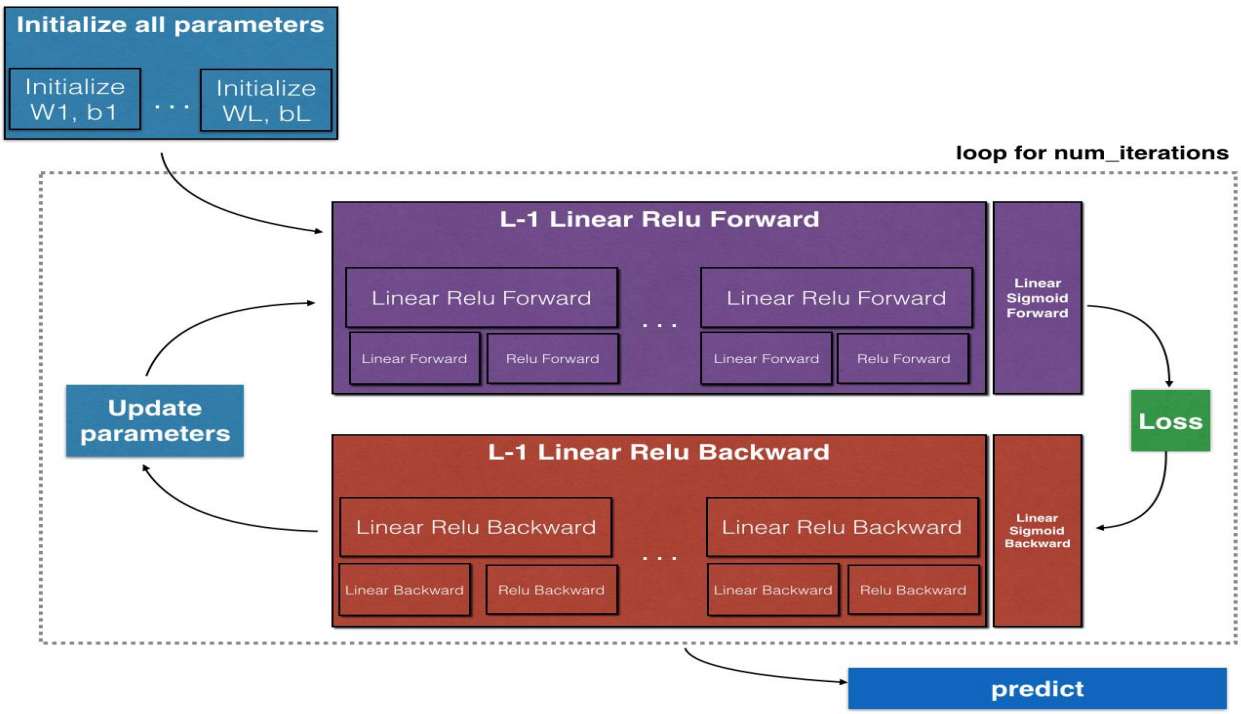
- Develop an intuition of the overall structure of a neural network.
- Write functions (e.g. forward propagation, backward propagation, logistic loss, etc...) that would help you decompose your code and ease the process of building a neural network.
- Initialize/update parameters according to your desired structure.

Outline of the Assignment

To build our neural network, we will be implementing several "helper functions". These helper functions will be used in the next assignment to build a two-layer neural network and an L-layer neural network. Here is an outline of this assignment, we will:

- Initialize the parameters for a two-layer network and for an L-layer neural network.
- Implement the forward propagation module (shown in purple in the figure below).
  - LINEAR part of a layer's forward propagation step (resulting in $Z^{[l]}$).
  - ACTIVATION function (relu/sigmoid).
  - Combine the previous two steps into a new [LINEAR->ACTIVATION] forward function.
  - Stack the [LINEAR->RELU] forward function L-1 time (for layers 1 through L-1) and add a [LINEAR->SIGMOID] at the end (for the final layer L). This gives you a new L_model_forward function.
- Compute the loss.
- Implement the backward propagation module (denoted in red in the figure below).
  - LINEAR part of a layer's backward propagation step.
  - The gradient of the ACTIVATE function (relu_backward/sigmoid_backward)

    - Combine the previous two steps into a new [LINEAR->ACTIVATION] backward function.
    - Stack [LINEAR->RELU] backward L-1 times and add [LINEAR->SIGMOID] backward in a new L_model_backward function
- Finally update the parameters.



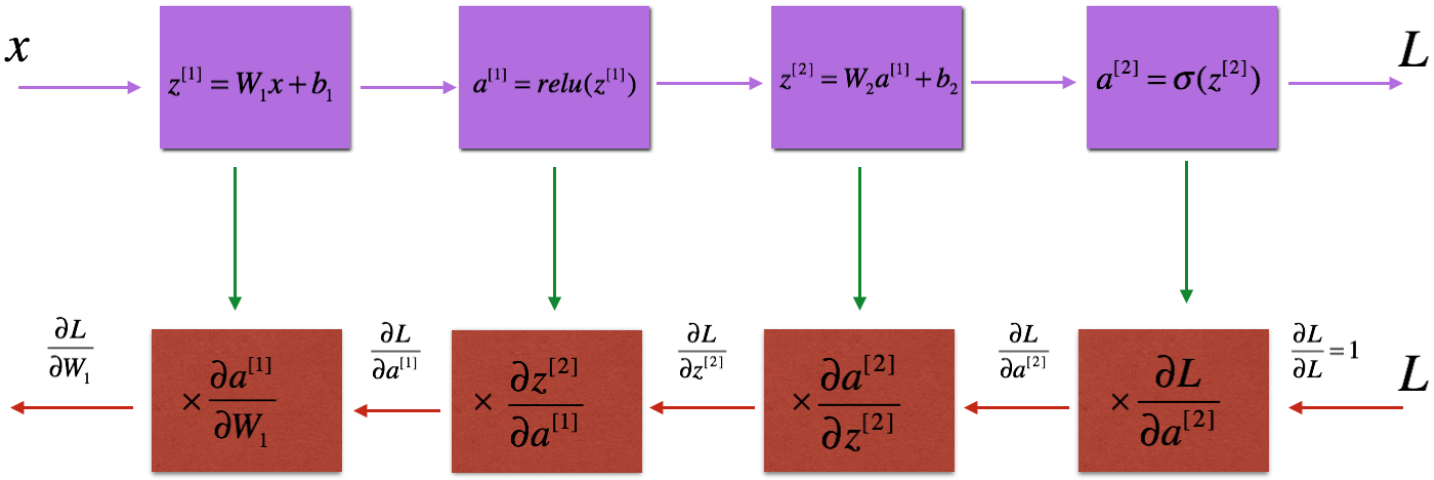
The initialization for a deeper L-layer neural network is more complicated because there are many more weight matrices and bias vectors.

When completing the initialize_parameters_deep, we should make sure that your dimensions match between each layer.
Recall that $n^{[l]}$ is the number of units in layer l.
Thus for example if the size of our input X is (12288,209) (with m=209m=209 examples) then:

	Shape of W	Shape of b	Activation	Shape of Activation
Layer 1	$(n^{[1]}, 12288)$	$(n^{[1]}, 1)$	$Z^{[1]} = W^{[1]} X + b^{[1]}$	$(n^{[1]}, 209)$
Layer 2	$(n^{[2]}, n^{[1]})$	$(n^{[2]}, 1)$	$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$	$(n^{[2]}, 209)$
⋮	⋮	⋮	⋮	⋮
Layer L-1	$(n^{[L-1]}, n^{[L-2]})$	$(n^{[L-1]}, 1)$	$Z^{[L-1]} = W^{[L-1]} A^{[L-2]} + b^{[L-1]}$	$(n^{[L-1]}, 209)$
Layer L	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 209)$

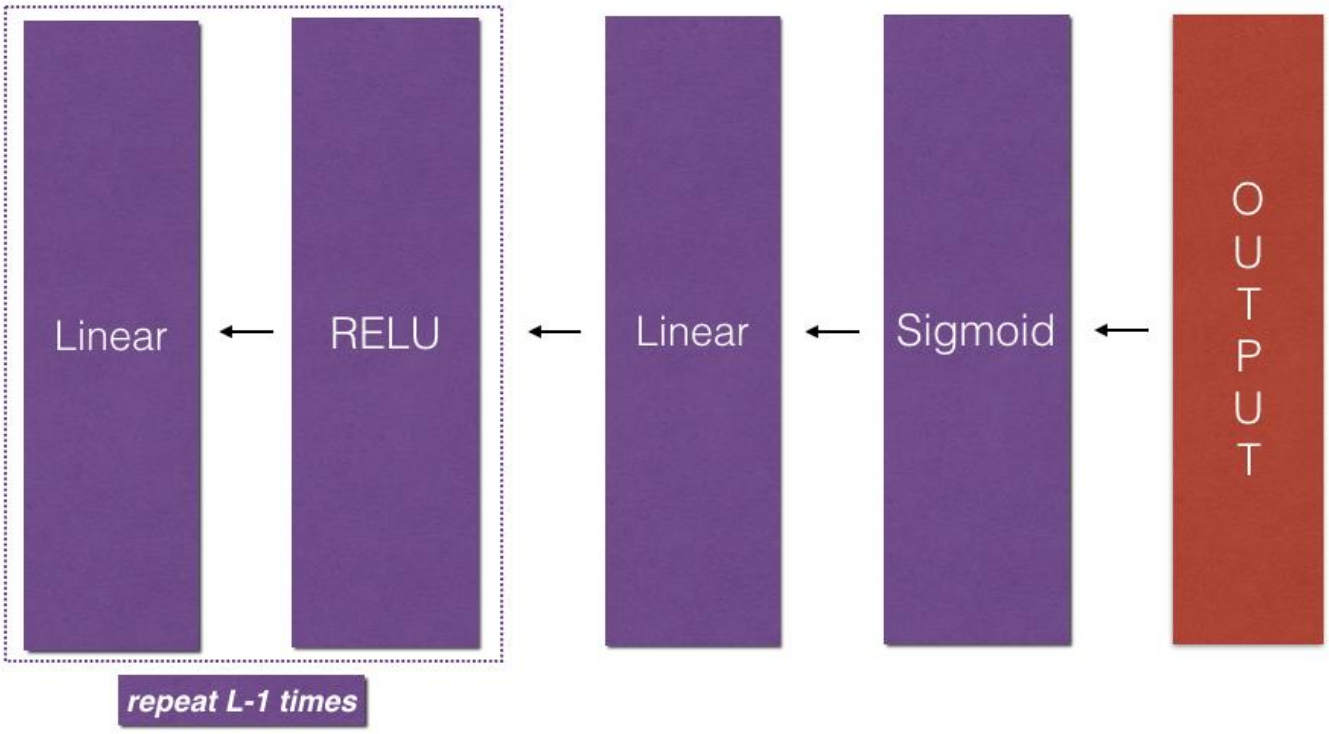
After initialising all the parameters, you implement the 1st half of the algorithm: forward-propagation, wherein you use “Relu” as the activation function for the first L-1 Layers and “Sigmoid” for the last Layer.

Then we compute the cost and start with the remaining half, i.e. the backward-propagation to reduce the cost of the Neural Network.



$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W_1}$$

In, Back-propagation, our primary task is to reduce cost, we do this by updating the parameters by calculating their derivatives. Here we start with the last Layer, where we left in the forward-propagation, and calculate $dA^{[L]}$, then from that we get $dZ^{[L]}$ from that we get $dA^{[L-1]}$, $dW^{[L]}$, $db^{[L]}$…and so on till $dW^{[1]}$, $db^{[1]}$.



Now, we have all the helper functions ready, so we simply merge them all.

Result:

We now have understood how to build a Deep -L Layer Network (in this case 5) to our Cat Classifier and have seen that we have reached an accuracy of about 80% on the test set.