

Neural Networks and Deep Learning

Complete Course Review

This is my first step in Deep Learning after a brief introduction of Neural Networks in Andrew Ng's Machine Learning Course by Stanford University.

The course begins with the phrase "AI is the new Electricity", similar to Electricity which transformed all the industries 100 years back and now AI would bring a similar transformation and Deep Learning is driving these developments.

So, this pretty much answers why I am doing this specialization- I want to be a part of these change rather than seeing the change.

We start the course by giving a very simple definition for Deep Learning as follows:
"Training neural networks or sometimes very large neural networks".

In the 1st week, Andrew Ng starts off with his classic housing prediction example to show how features (parameters) are selected in a Neural Network (NN), evaluated the reason why Deep Learning is taking off so well.

I also understood that there is no hard rule to know the values of hyper-parameters (learning rate, # iterations, # Layers, etc), we need to test them with different set of values and compare the results. The general rule of thumb is "Applied Deep Learning is a very empirical process". We generally follow:

Idea -> Code -> Experiment

Set Values -> Implement -> Analyse Result

In the process, I understood the importance of Vectorising rather than using loops as using loops significantly increases the time complexity as the data size increases and # iterations increases

In the 2nd week, we have built a cat recognizer initially with logistic regression with a success rate of 70%. Later, we added few hidden layers (~1), to see what was the result and then later increased the depth (more number of hidden layers) thereby getting a first-hand experience of how deep learning works.

In week 2, we take the 1st formal step into neural network by understanding how logistic regression works, we see it as a neural network with '0' hidden layers to build a cat classifier with the help of sigmoid activation function to determine whether a given image is cat or non-cat, we got a training accuracy of 99% and Test accuracy of 70% indicating that we are over-fitting the data.

In Week 3, we continue from where we left in week 2 by defining logistic regression as a neural network with '0' hidden units to shallow neural network with '1' hidden layer.

We understood, the intuition behind the back-propagation algorithm, the forward propagation computes the final output and computes the cost(loss) and in the back propagation we try to optimize the parameters, in such a way to reduce the cost very similar to how we try to do optimize something to reduce error and make it more efficient, we try to correct it next time when we perform.

We have seen the importance of random-initialisation instead of initialising them to zeros because if you initialise all of them to zeros all your hidden units of a given layer would compute the same value, which is redundant.

After our 1st 3 weeks spent in building a solid foundation of how a Neural Networks, we took a deep dive to see the functioning of Deep Neural Networks, understood the complete flow of Forward-Propagation and Backward-Propagation.

I think Andrew did a great job especially in giving intuitions e.g. how he used computing graphs to explain the concept of derivatives making it easy to understand even for a non-programmer.

One drawback I felt with the Stanford Machine Learning course was we were asked to program in Octave/Matlab, but I think that was heard by the course instructors and we are using Python 3 in this course.

The programming assignments were really helpful as they helped me to feel of the algorithm better, the way we built logistic regression as a simple neural network with no hidden layer and by the end of the course with a deep neural network.

We have seen different activation functions namely: sigmoid, tanh and Relu (Rectified Linear Unit) and also understood the pros and cons of each one of them. The why part really helped me to get a better insight while implementing these.

I have learnt a very important debugging tool: matrix dimensions -> this reduced 90% of all my errors, checking whether my matrix dimensions are right, if yes go to next step else fix that.

The analogy from circuit theory helps you to understand why deep layers are needed, more informally, there are functions you can compute with a "small" L-layer deep neural network (NN) that shallow NN require exponentially more hidden units to compute. Intuitions from what different layers of a deep network does in Image recognition, Speech recognition helped me get a better understanding of how deep learning works.

In Machine Learning/Deep 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.

At the end of this course, I can confidently say how a simple deep network, understood the power of a simple yet extremely efficient back-propagation algorithm. I feel I have the foundation set to dive deep into deep learning. I am excited for the 2nd course....

Neural Networks and Deep Learning

Week-1




This is my first step in Deep Learning after a brief introduction of Neural Networks in Andrew Ng's Machine Learning Course by Stanford University.

The course begins with the phrase "AI is the new Electricity", similar to how Electricity has transformed all the industries 100 years back and now AI would bring a similar transformation and Deep Learning is driving these developments.

So, this pretty much answers why I am doing this specialization- I want to be a part of this change rather than seeing the change.

In the 1st week, Andrew Ng starts off with his classic housing prediction example to show how features (parameters) are selected in a Neural Network (NN).

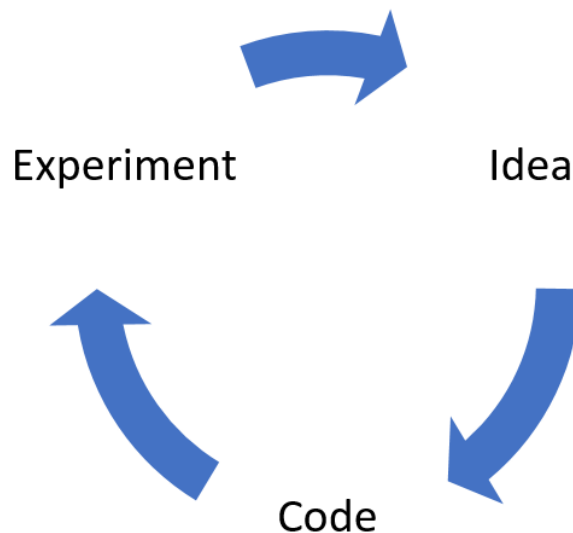
We discuss about different types of Neural Networks and also see what to apply when:

- | | | |
|-----------------------|---|----------------|
| 1. Price Prediction |  | Standard |
| Online Advertising | | Neural Network |
| 2. Photo |  | Convolutional |
| Tagging | | Neural Network |
| 3. Speech Recognition |  | Recurring |
| Machine Translation | | Neural Network |

We try to understand the fundamental difference between structured and unstructured data and also evaluate the reasons why Deep Learning is suddenly taking off due to 2 primary reasons:

1. More data
 - Size of the Neural Network
 - Training data
2. More computational power

We understand the importance of Vectorisation and how it helps us to remove loops which helps us to iterate through the below diagram quickly:



Neural Networks and Deep Learning

Week-2

In week 1, we have given a very basic definition of what a neural network is. Now, we take the 1st formal step into neural network by understanding how logistic regression works, we see it as a neural network with '0' hidden layers to build a cat classifier.

We understand how to use sigmoid activation function, to determine whether a given image is cat or non-cat. We get the intuition behind cost function, which in the simplest terms mean penalising the algorithm heavily for every wrong decision it makes, to better fit the training data.

We understood how vectorization significantly improves the computation time instead of using loops.

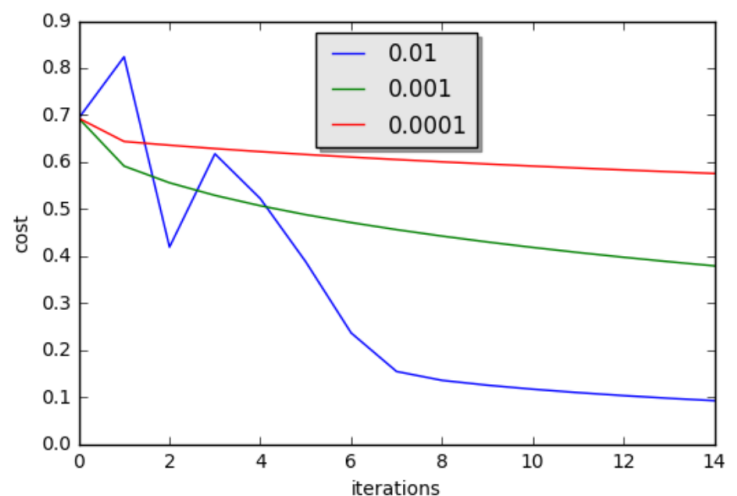
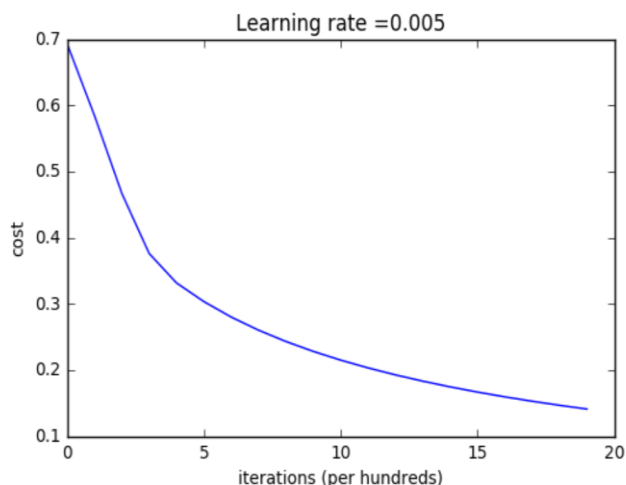
We have seen how Broadcasting works in python, which is helpful in most of the situations but at times it can introduce subtle undesired bugs.

After running logistic regression on the cat classifier, we got a training accuracy of 99% and Test accuracy of 70% indicating that we are over-fitting the data.

We have seen, how the results vary if we change the value of the parameter α (learning-rate).

Result:

Learning Rate	Train Accuracy	Test Accuracy
0.05	99.04 %	70 %
0.01	99.51 %	68 %
0.001	88.99 %	64 %
0.0001	68.42 %	36 %



Both the graphs are on Test Set only.

Conclusion:

1. The graph on the left (Cost Vs Number of Iterations) confirms that our algorithms work well, as cost is decreasing with number of iterations.
2. The graph on the right (Cost Vs Number of Iterations (for different learning rates))
 - Different learning rates give different costs and thus different predictions results
 - If the learning rate is too large (0.01), the cost may oscillate up and down. It may even diverge (though in this example, using 0.01 still eventually ends up at a good value for the cost).
 - A lower cost doesn't mean a better model. You have to check if there is possibly overfitting. It happens when the training accuracy is a lot higher than the test accuracy.
 - In deep learning, it is usually recommended that you:
 - Choose the learning rate that better minimizes the cost function.
 - If your model overfits, use other techniques to reduce overfitting.

Neural Networks and Deep Learning

Week-3

In Week 3, we continue from where we left in week 2 by defining logistic regression as a neural network with '0' hidden units to shallow neural network with '1' hidden layer.

We understood, the intuition behind the back-propagation algorithm, the forward propagation computes the final output and computes the cost(loss) and in the back propagation we try to optimize the parameters, in such a way to reduce the cost very similar to how we try to do optimize something to reduce error and make it more efficient, we try to correct it next time when we perform.

We learnt a very simple technique of how Matrix Dimensions helps us to check if the flow of our algorithm is right, which is extremely helpful while debugging.

We now started using new activation functions Tanh, Relu and Leaky Relu function in addition to sigmoid function because sigmoid function is almost linear between in certain range and if we use sigmoid function as the activation function throughout the neural network, we will simply compute an output, which is some linear combination of your input, but if we use sigmoid function in the final output layer, as it performs well as classifier.

We have seen the importance of random-initialisation instead of initialising them to zeros because if you initialise all of them to zeros all your hidden units of a given layer would compute the same value, which is redundant.

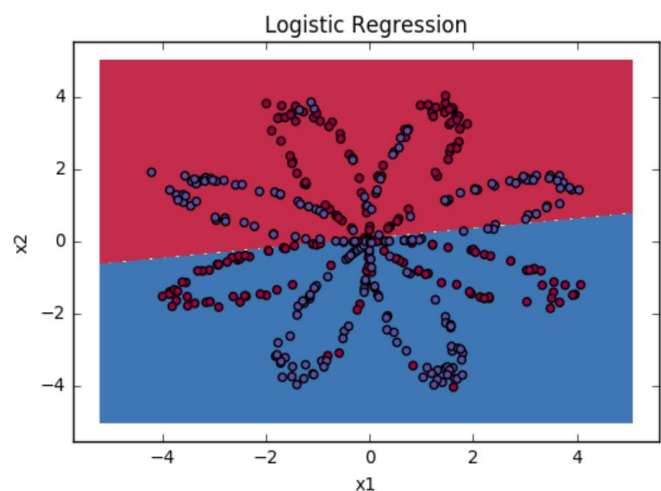
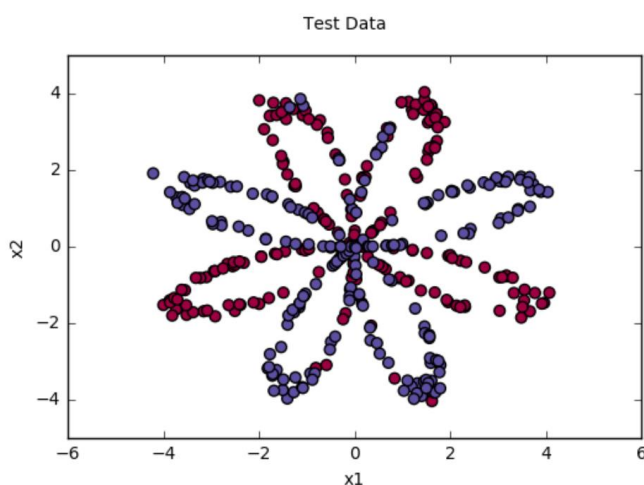
In the Programming Assignment, we generate red and blue points to form a flower. We will then fit a neural network to correctly classify the points.

To understand the importance of neural network even more, we implemented the same problem with logistic regression and as we can see the image on the right, we get a linear boundary line where the dataset is not linearly separable.

The decision boundary of a logistic regression is linear as long as the input features are linear.

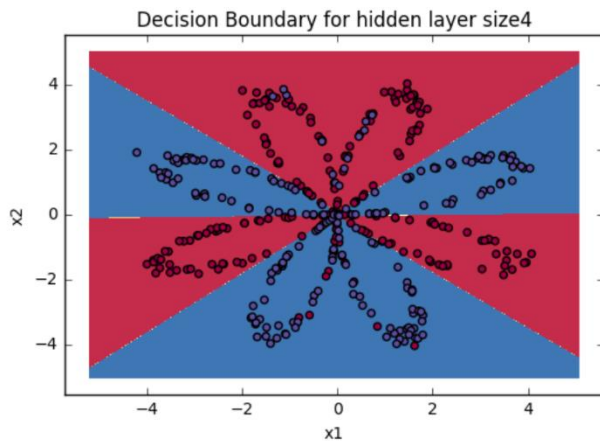
$Z = w^T X + b$, $Y = \sigma(Z)$ -> Linear Decision Boundary as Input is Linear

Say $X = f^2$, $Z = w^T X + b$, $Y = \sigma(Z)$ -> Non- Linear Decision Boundary as Input is Non-linear



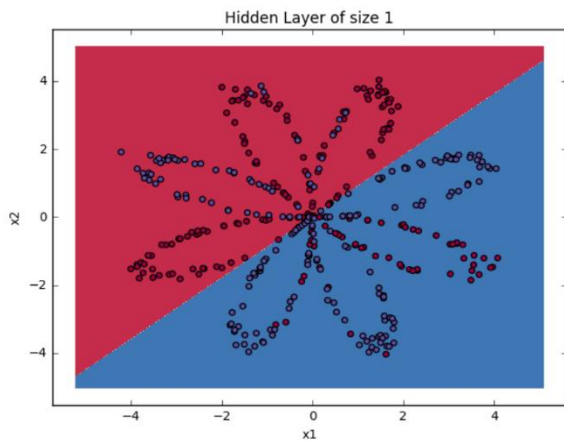
We built a Neural Network with
1 Input layer (2 features x_1 , x_2)
1 hidden layer (4 units)
1 Output Layer.

Result:

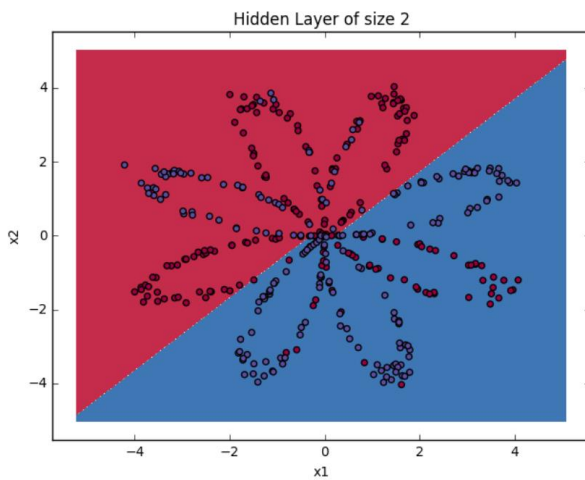


Accuracy for 4 hidden units: 90.5%

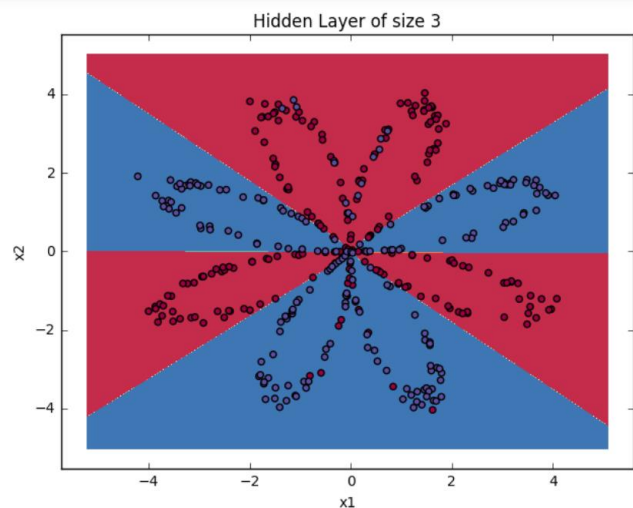
Hidden Layers of Different Values:



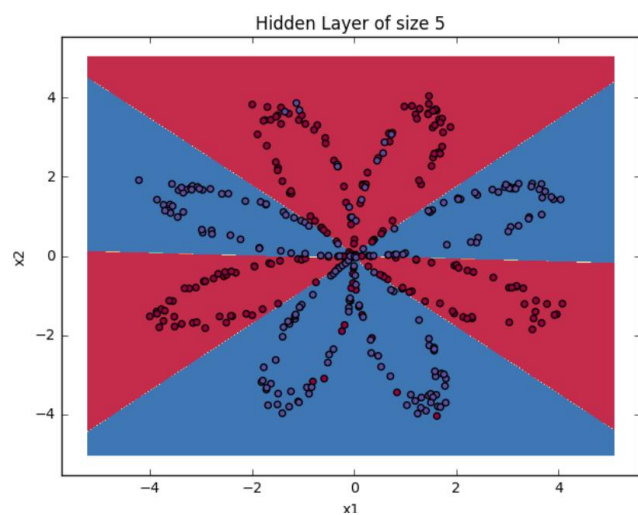
Accuracy for 1 hidden unit: 67.5%



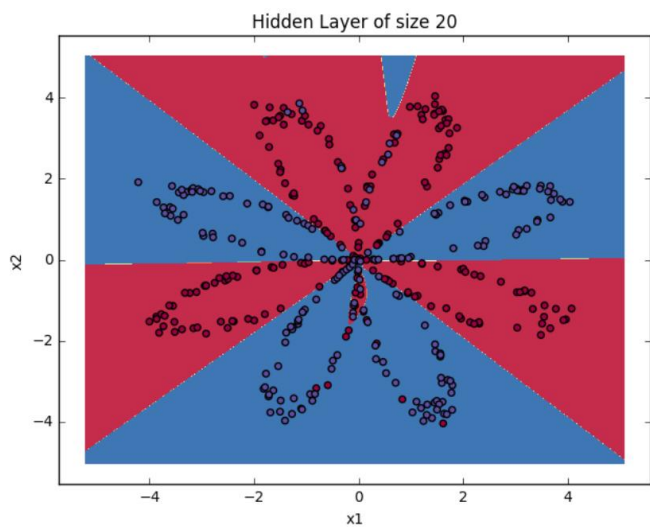
Accuracy for 2 hidden units: 67.25%



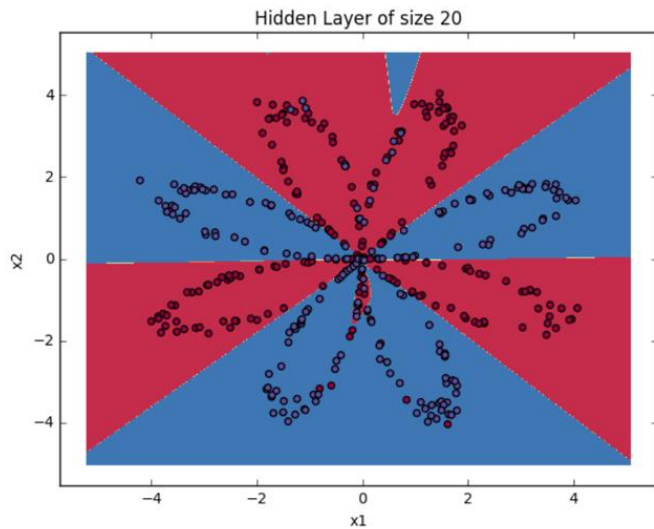
Accuracy for 3 hidden units: 90.75%



Accuracy for 5 hidden units: 91.25%



Accuracy for 20 hidden units: 90.0%



Accuracy for 50 hidden units: 90.25%

Conclusion:

- The larger models (with more hidden units) are able to fit the training set better, until eventually the largest models overfit the data.
- The best hidden layer size seems to be around $n_h = 5$. Indeed, a value around here seems to fit the data well without also incurring noticeable overfitting.

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:

$$dZ^{[l]} = dA^{[l]} * g'^{[l]}(Z^{[l]}) \rightarrow \textcircled{1}$$

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

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

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

$$dA^{[l]} = W^{[l]T} * dZ^{[l+1]} \rightarrow \textcircled{2}$$

From $\textcircled{1}$ & $\textcircled{2}$, 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.

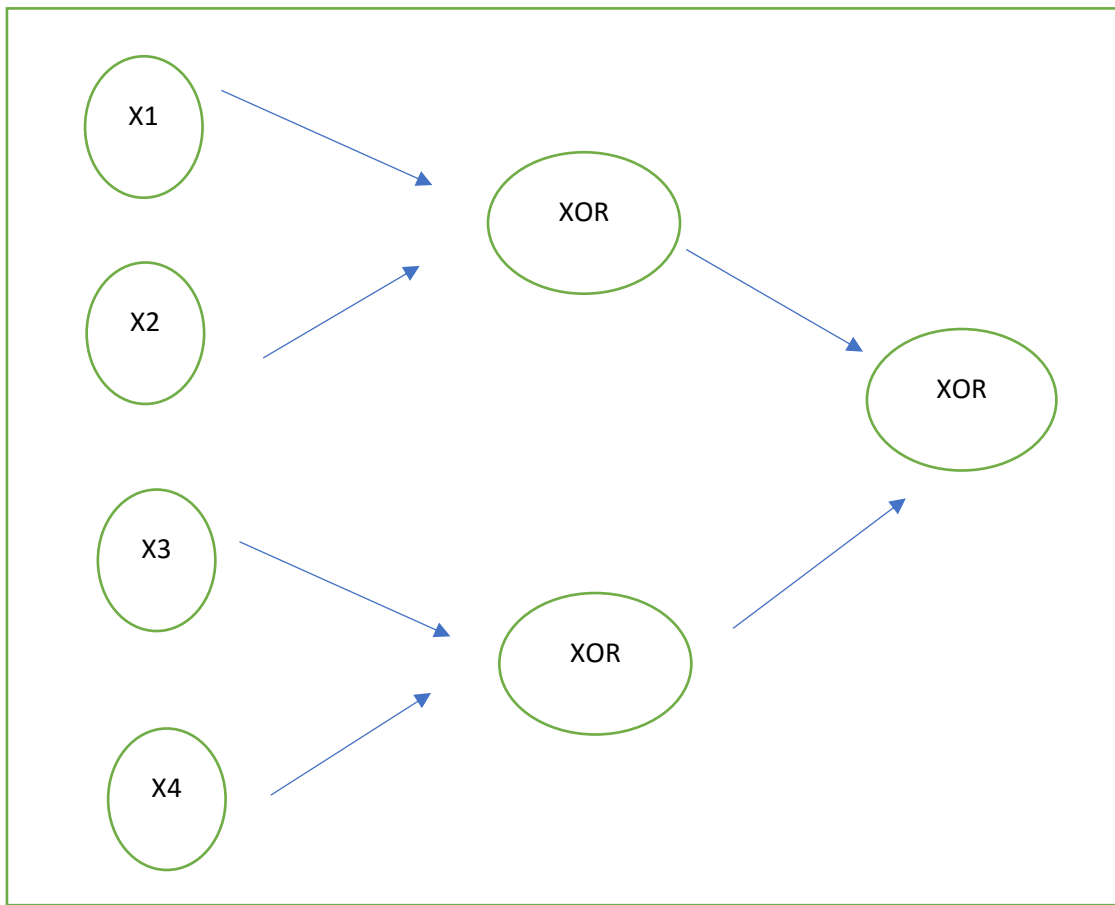
Why do we require deep representations?

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.

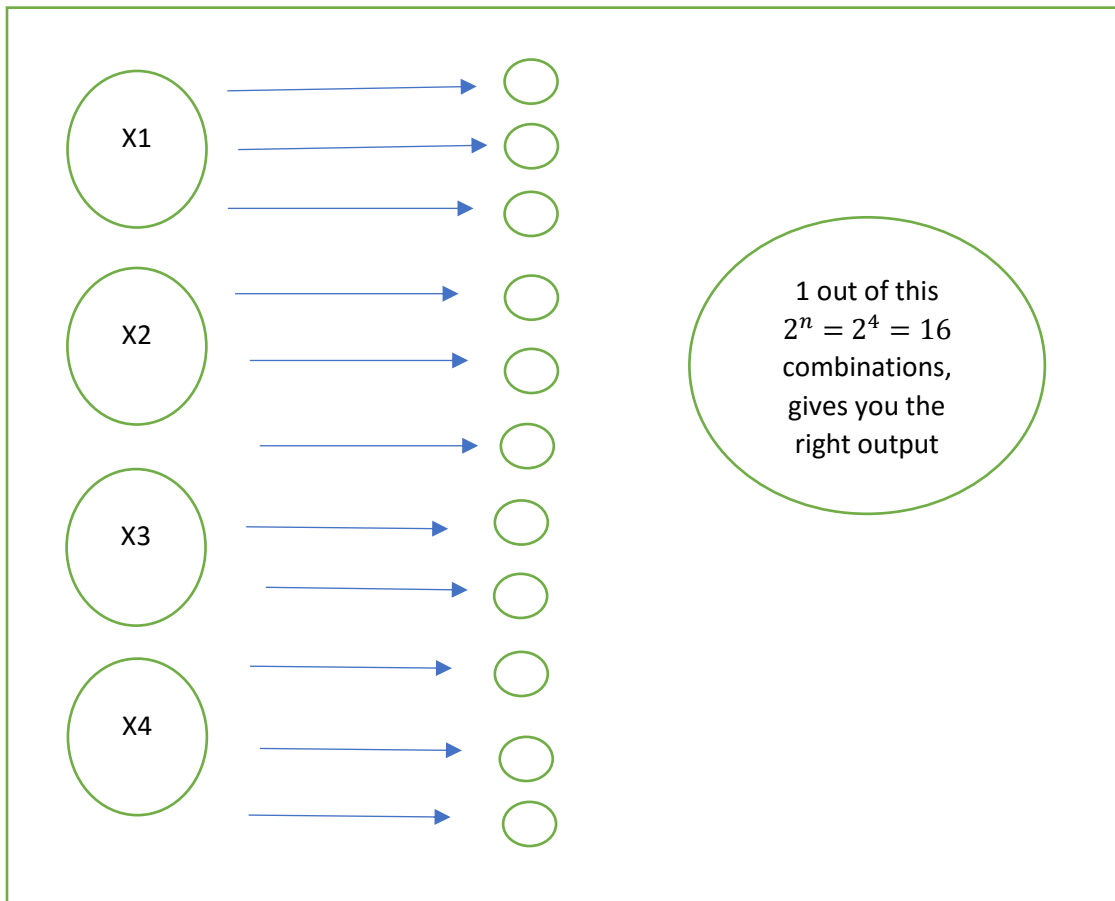
There is a famous correlation between Circuit Theory and Deep Learning:

“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 $n^{[1]}, n^{[2]}, \dots, n^{[L]}$
- 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:

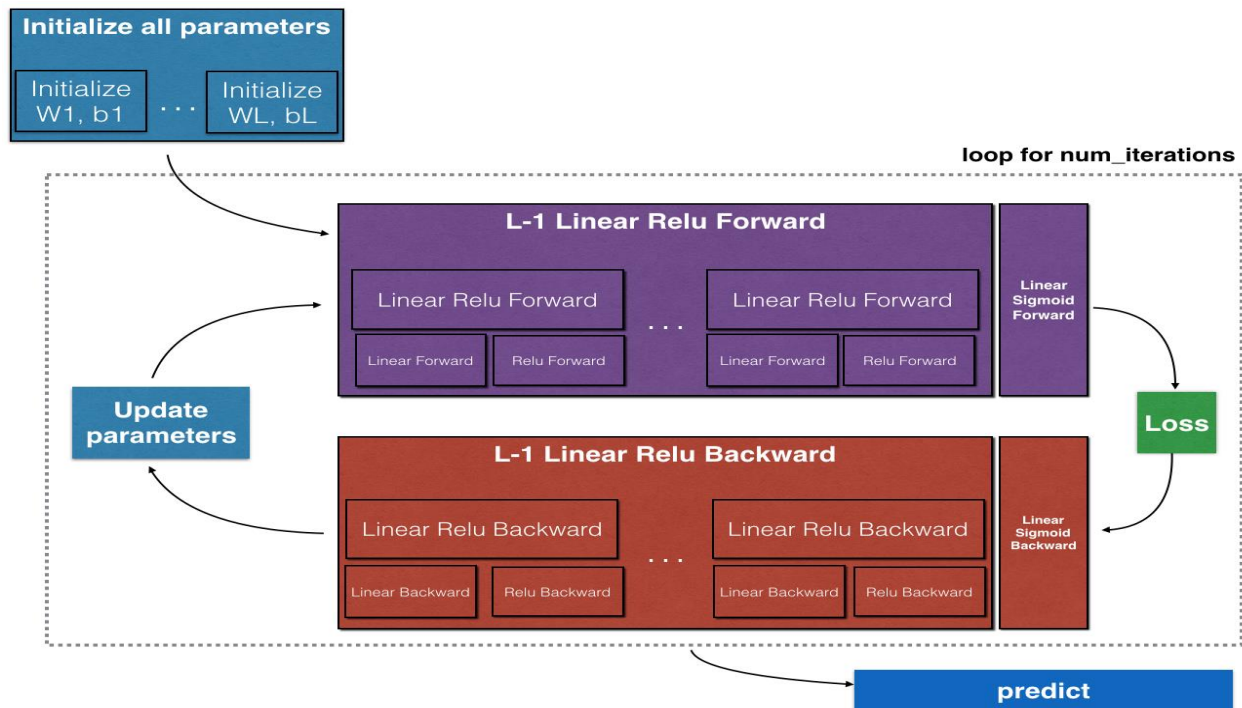
- Develop an intuition of the overall structure of a neural network.
- Write functions (e.g. forward propagation, backward propagation, logistic loss, etc...) that would help you decompose your code and ease the process of building a neural network.
- Initialize/update parameters according to your desired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

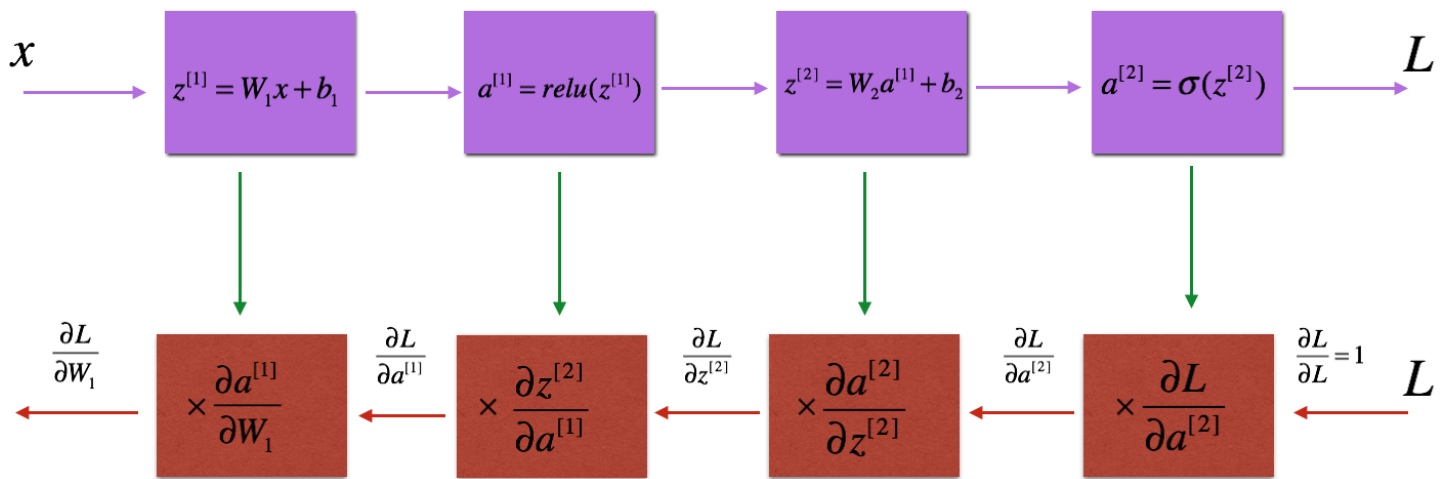
Recall that $n^{[l]}$ is the number of units in layer l.

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	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 209)$

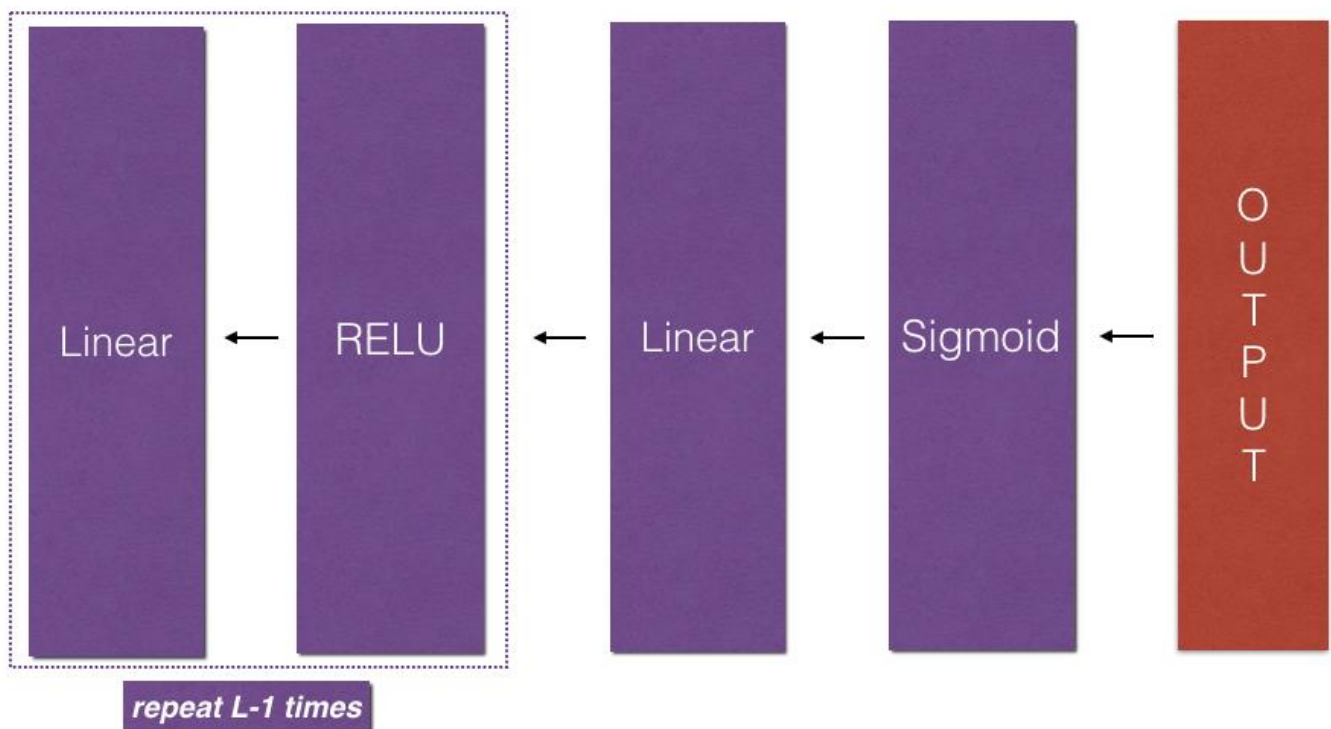
After initialising all the parameters, you implement the 1st half of the algorithm: forward-propagation, wherein you use “Relu” as the activation function for the first L-1 Layers and “Sigmoid” for the last Layer.

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



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

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 $dW^{[1]}$, $db^{[1]}$.

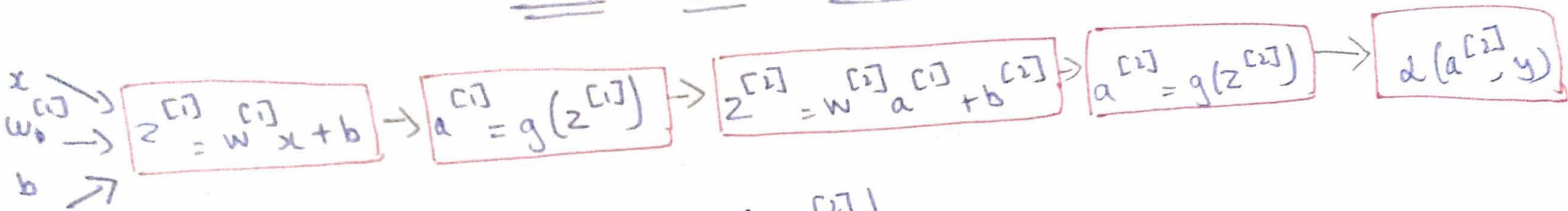


Now, we have all the helper functions ready, so we simply merge them all.

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.

2-Layer Neural Network



$$d(a^{[2]}, y) = -(y \log a^{[2]} + (1-y) \log (1-a^{[2]}))$$

$$\textcircled{1} \frac{dL}{da^{[2]}} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}} \rightarrow \boxed{da^{[2]} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}}$$

$$\textcircled{2} \frac{dL}{dz^{[2]}} = \frac{dL}{da^{[2]}} \cdot \frac{da^{[2]}}{dz^{[2]}}$$

$$\frac{da^{[2]}}{dz^{[2]}} = \frac{d}{dz} \left(\frac{1}{1+e^{-z}} \right) = \frac{(1+e^{-z})^{-1}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \times \left(1 - \frac{1}{1+e^{-z}} \right) = a \times (1-a)$$

(Assuming Sigmoid)

$$\frac{dL}{dz^{[2]}} = \left(-\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}} \right) \times (a^{[2]} \times (1-a^{[2]}))$$

$$= -y(1-a^{[2]}) + (1-y)a^{[2]} \rightarrow \boxed{\frac{dL}{dz^{[2]}} = a^{[2]} - y}$$

$$\textcircled{3} \frac{dL}{dw^{[2]}} = \frac{dL}{dz^{[2]}} \times \frac{dz^{[2]}}{dw^{[2]}}$$

$$\frac{dz^{[2]}}{dw^{[2]}} = \frac{d}{dw^{[2]}} (w^{[2]} a^{[1]} + b^{[2]}) = a^{[1]}$$

$$\frac{dL}{dw^{[2]}} = (a^{[2]} - y) \times a^{[1]} \rightarrow$$

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

* $\frac{1}{m}$ indicates Average.

$$\textcircled{4} \frac{dL}{db^{[2]}} = \frac{dL}{dz^{[2]}} \times \frac{dz^{[2]}}{db^{[2]}}$$

$$\frac{dz^{[2]}}{db^{[2]}} = \frac{d}{db^{[2]}} (w^{[2]} a^{[1]} + b^{[2]}) = 1$$

$$\frac{dL}{db^{[2]}} = (a^{[2]} - y) \times 1 \rightarrow$$

$$db^{[2]} = \frac{1}{m} dz^{[2]}$$

* Sum of all the elements

⑤ Similarly, $\frac{dL}{dz^{[2]}} = \frac{dL}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}}$

$$\frac{dz^{[2]}}{da^{[1]}} = \frac{d}{da^{[1]}} (w^{[2]} a^{[1]} + b^{[2]}) = w^{[2]}$$

$$\frac{da^{[1]}}{dz^{[1]}} = \frac{d}{dz^{[1]}} \left(\frac{1}{1+e^{-x}} \right) = a^{[1]} * (1-a^{[1]}) \Rightarrow g^{[1]'}(z^{[1]})$$

$$\frac{dL}{dz^{[2]}} = dz^{[2]} * w^{[2]} * g^{[1]'}(z^{[1]}) \Rightarrow$$

$$dz^{[1]} = w^{[2]T} dz^{[2]} \cdot g^{[1]'}(z^{[1]})$$

Element-wise
multiplication

$$dw^{[2]} = \frac{1}{m} dz^{[2]} x^T$$

~~dL/dw~~ ⑥ $\frac{dL}{dw^{[2]}} = \frac{dL}{dz^{[2]}} \frac{dz^{[2]}}{dw^{[2]}} = dz^{[2]} x$

⑦ $\frac{dL}{db^{[2]}} = \frac{1}{m} * dz^{[2]}$

Dimensional Analysis! - Forward Propagation

Input $x \rightarrow n^{[0]}$

Hidden layer $\rightarrow n^{[1]}$

Output layer $\rightarrow n^{[2]} = 1$

} no. of features.

$$W^{[1]} = \begin{pmatrix} n^{[1]} & n^{[0]} \end{pmatrix}$$

$$Z^{[1]} = W^{[1]} x + b^{[1]} \rightarrow \begin{matrix} \text{single element} \\ \text{Broadcasts into} \end{matrix} \begin{pmatrix} n^{[1]} & 1 \end{pmatrix}$$

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

$$W^{[2]} = \begin{pmatrix} n^{[2]} & n^{[1]} \end{pmatrix}$$

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

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

Dimensional Analysis :-

Backward Propagation :-

$$dz^{[2]}_{(n^{[2]}, 1)} = A^{[2]}_{(n^{[2]}, 1)} - y_{(n^{[2]}, 1)}$$

$$dw^{[2]}_{(n^{[2]}, 1)} = \frac{1}{m} dz^{[2]}_{(n^{[2]}, 1)} \cdot A^{[1]T}_{(1, n^{[1]})}$$

$$db^{[2]}_{(1, 1)} = \frac{1}{m} \text{np.sum}(dz^{[2]}_{(1, 1)})$$

$$dz^{[1]}_{(n^{[1]}, 1)} = w^{[2]T}_{(n^{[2]}, 1)} * dz^{[2]}_{(n^{[2]}, 1)} * g'^{[1]}(z^{[1]}_{(n^{[1]}, 1)})$$

$$db^{[1]}_{(1, 1)} = \frac{1}{m} \text{np.sum}(dz^{[1]}_{(1, 1)})$$

Backward Propagation :-

Input: $da^{[L]}$

Output: $da^{[L-1]}$, $dw^{[L]}$, $db^{[L]}$

$$dz^{[L]} = \frac{dL}{dz^{[L]}} = \frac{dL}{da^{[L]}} \frac{da^{[L]}}{dz^{[L]}}$$

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

$$dz^{[L]} = da^{[L]} * g^{[L]'}(z^{[L]}) \quad \text{--- (1)}$$

$$dw^{[L]} = dz^{[L]} * A^{[L-1]}$$

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

This is the idea.
Actual formula might slightly vary. --- (2)

$$\frac{dL}{da^{[L-1]}} = \frac{dL}{dz^{[L]}} \frac{dz^{[L]}}{da^{[L-1]}}$$

$$da^{[L-1]} = \cancel{dz^{[L]}} w^{[L]T} dz^{[L]}$$

$$da^{[L]} = w^{[L+1]T} dz^{[L+1]} \quad \text{--- (3)}$$

$$\therefore dz^{[L]} = w^{[L+1]T} dz^{[L+1]} * g^{[L]'}(z^{[L]})$$