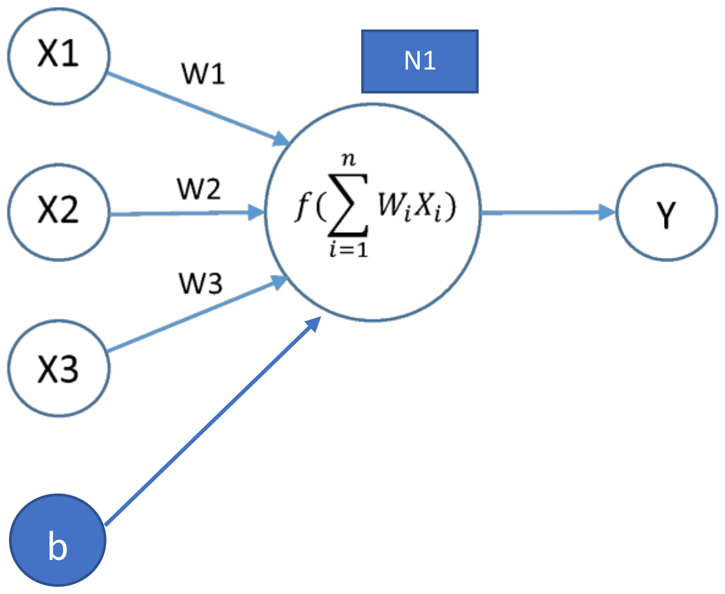
Neural Networks

Neural network imitates the human brain.
In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

Below is the simplest neural network model. It is similar to the logistic regression model.



Here, output Y is the predicted digit. This model represents a function line which is the linear combination of input features.

So, the algorithm looks like this-

- Node N1 will take linear combination of all input features i.e. (W1X1 + W2X2 + W3X3 + b)

- N1 will perform some activation function on input data. In this case, Sigmoid function. So, output from node N1 will be:

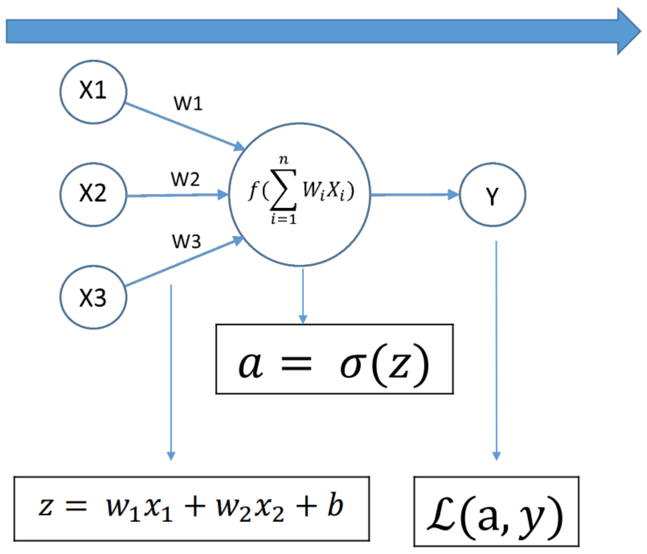
$$a = \sigma(w^T x + b)\ , where\ \sigma(z) = \frac{1}{1 + e^{-z}}$$

  and w and x are weights and inputs matrices.
- If output of Node1 is greater than 0.5, value of Y will be 1.
- If output of Node1 is smaller than or equal to 0.5, value of Y will be 0.
- Now, to determine cost function, we will use cross entropy error instead of root mean square error (because output is not continuous value, it is like a classification problem.

  $J(w, b) = \mathcal{L}(\text{a}, y) = -\left[y^{(i)}\log a^{(i)} + \left(1 - y^{(i)}\right)\log\left(1 - a^{(i)}\right)\right]$

- Optimization: Now, we have to minimize the cost function from the previous step. In short, we have to find value of ‘a’ that minimize the cost function. And as ‘a’ depends on w and b, now our main task is to find w and b which minimize cost function.
- Find derivative of $J(w, b)$, equate it to 0 and find value of w and b. It is known as Back Propagation.

Forward Propagation

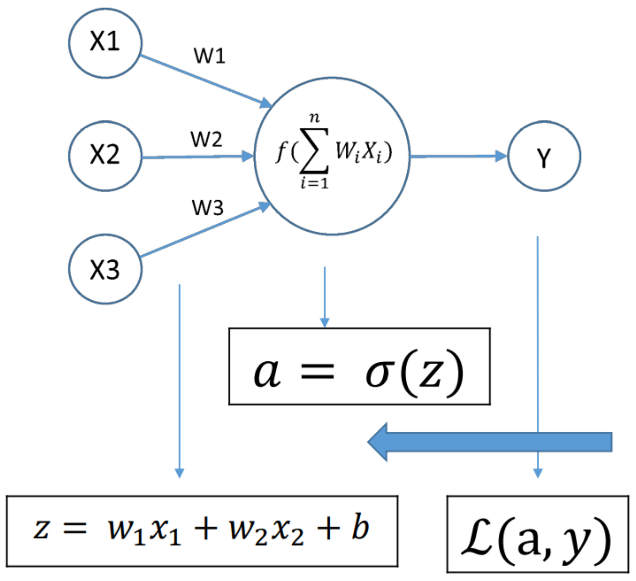


```
for i in range(0,N):
    z = w[i]*x[i]
z=+b
a = sigmoid(z)
L(a,y)= -(y*log(a) + (1-y)*log(1-a))
```

$z = w^T*X+b$ #w=[w1,w2] and X=[x1,x2]

One training example
{X,y} ➔ X = [x1,x2]

Backpropagation



Find w1,w2 and b that reduces $\mathcal{L}(a, y)$

Using derivatives

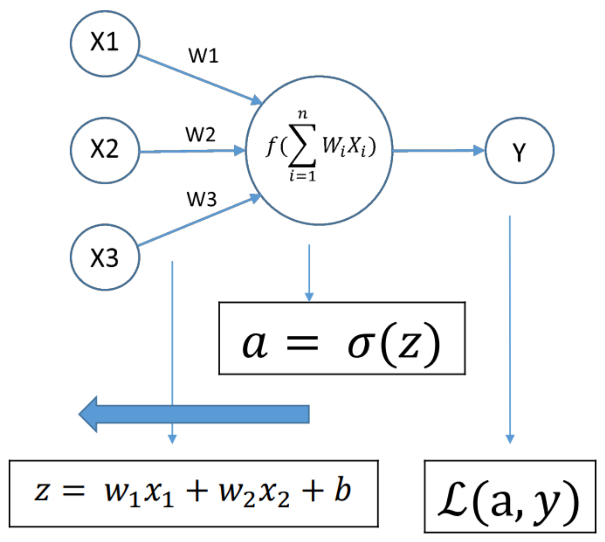
$$\frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{da}{dz} = a(1-a)$$

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

$$= a - y$$

Backpropagation


X1
X2
X3
W1
W2
W3
$f(\sum_{i=1}^{n} W_i X_i)$
Y
$a = \sigma(z)$
$z = w_1x_1 + w_2x_2 + b$
$\mathcal{L}(\mathrm{a}, y)$

$$\frac{d\mathcal{L}(\mathrm{a},y)}{dw1} = -\frac{d\mathcal{L}(\mathrm{a},y)}{dz} * \frac{dz}{dw1}$$
$$= \mathrm{x1}{*}\mathrm{d}\mathcal{L}\mathrm{z}$$

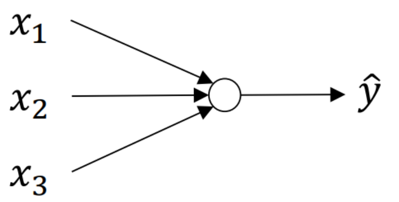
$$\frac{d\mathcal{L}(\mathrm{a},y)}{dw2} = -\frac{d\mathcal{L}(\mathrm{a},y)}{dz} * \frac{dz}{dw2}$$
$$= \mathrm{x2}{*}\mathrm{d}\mathcal{L}\mathrm{z}$$

$$\frac{d\mathcal{L}(\mathrm{a},y)}{db} = -\frac{d\mathcal{L}(\mathrm{a},y)}{dz} * \frac{dz}{db}$$
$$= \mathrm{d}\mathcal{L}\mathrm{z}$$

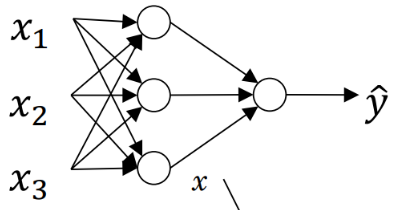
$$update:$$
$$w1 = w1 - \alpha * \frac{d\mathcal{L}(\mathrm{a},y)}{dw1}$$
$$w2 = w2 - \alpha * \frac{d\mathcal{L}(\mathrm{a},y)}{dw1}$$
$$b = -\alpha * \frac{d\mathcal{L}(\mathrm{a},y)}{db}$$

Similarly, it will work for deep neural network. In deep network, every node will act as logistic network individually with different activation function.

Sometimes, for hidden layers, we require different activation functions. To know more about different activation functions. Visit: [Activation functions]

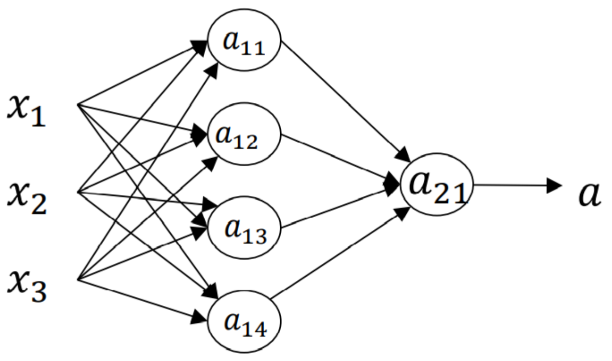


$$\begin{tikzcd}
x \arrow[dr] & & & \\
w \arrow[r] & z = w^T x + b \arrow[r] & a = \sigma(z) \arrow[r] & \mathcal{L}(a, y) \\
b \arrow[ur] & & &
\end{tikzcd}$$



$$\begin{tikzcd}
x \arrow[dr] & & & & & \\
W^{[1]} \arrow[r] & z^{[1]} = W^{[1]} x + b^{[1]} \arrow[r] & a^{[1]} = \sigma(z^{[1]}) \arrow[r] & z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \arrow[r] & a^{[2]} = \sigma(z^{[2]}) \arrow[r] & \mathcal{L}(a^{[2]}, y) \\
b^{[1]} \arrow[ur] & & W^{[2]} \arrow[ur] & b^{[2]} \arrow[u] & &
\end{tikzcd}$$

This model can be represented as:



x_1
x_2
x_3

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}
w_{41}	w_{42}	w_{43}

b_1
b_2
b_3
b_4

Layer 1

a_1
a_2
a_3
a_4

w_{11}	w_{12}	w_{13}	w_{14}

b_1

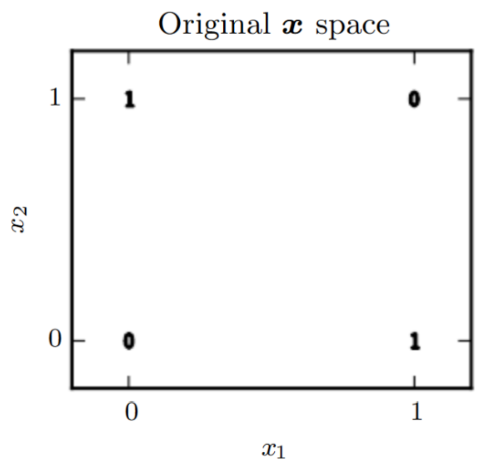
Layer 2

Simplest Multilayer Neural network:

- XOR function

X	
0	0
0	1
1	0
1	1

y
0
1
1
0

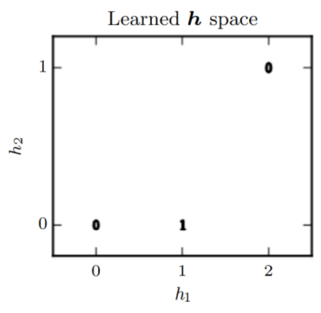


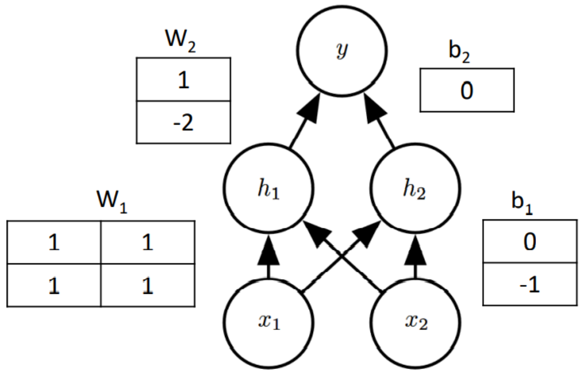
We can not use single layer network because it is impossible to separate 0 and 1 with linear line.

We need multilayer network here.

X	
0	0
0	1
1	0
1	1

y
0
1
1
0





By using another hidden layer, model combined two features and now, 0 and 1 can be easily separated by a linear line. (as shown in figure above)

Initialization of weights and bias:

If we initialize everything with zero, it will take a long time to converge. It is better to initialize weight with some small value and bias with 0.

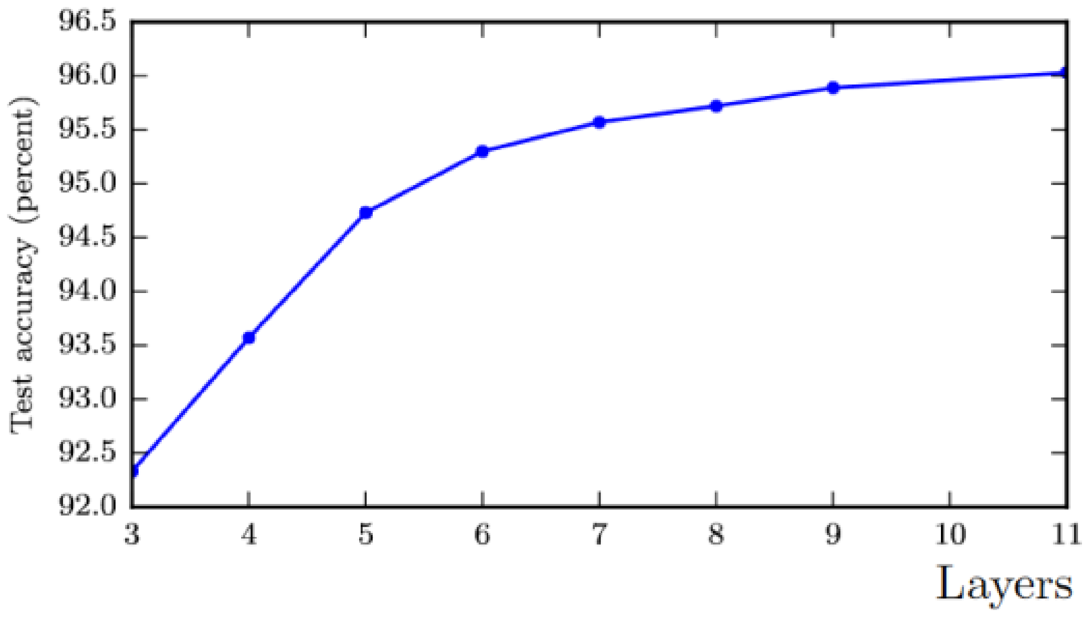
Universal Approximation Theorem:

“Universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of Rn, under mild assumptions on the activation function. The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.”

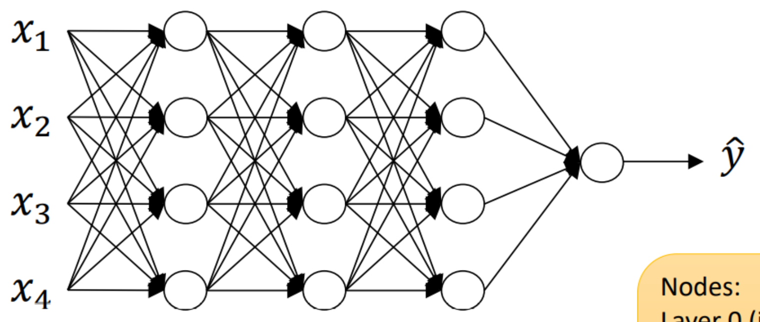
In short, a feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
A deep network with n layers will be perform same as shallow network with one layer with 2^n nodes.

Advantages of Deep Network:

- Latent feature extraction.
- Same functions can be implemented using a deep and shallow network
  - The shallow network implementation will require exponentially more nodes
  - More computational required.
- Performs better.



Network with L Layers



Layers: 4+1

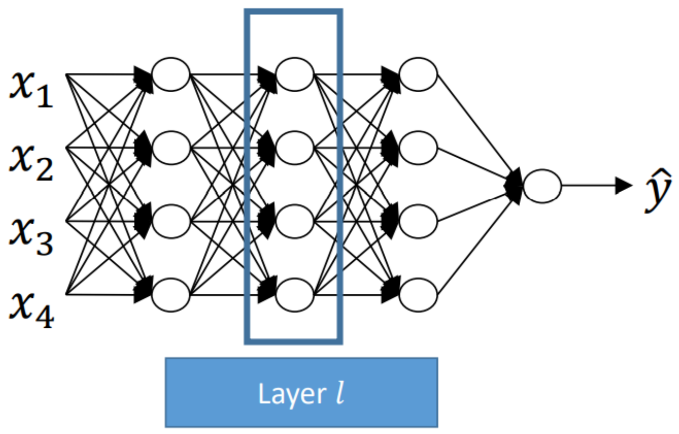
3 Hidden layers

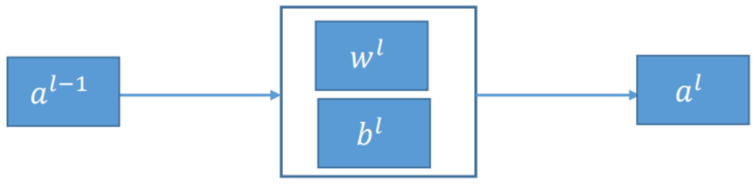
Nodes:
Layer 0 (input): 4
Layer 1 (hidden): 4
Layer 2 (hidden): 4
Layer 3 (hidden): 4
Layer 4 (output): 1

Total layers = 4 and 1 input layer

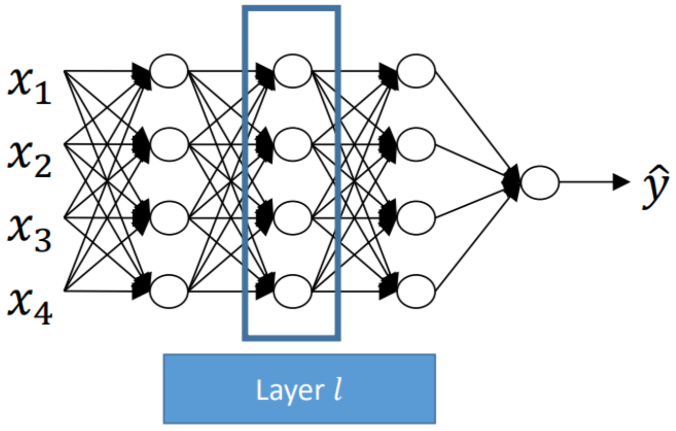
So, in deep network with L layers, the algorithm will be:

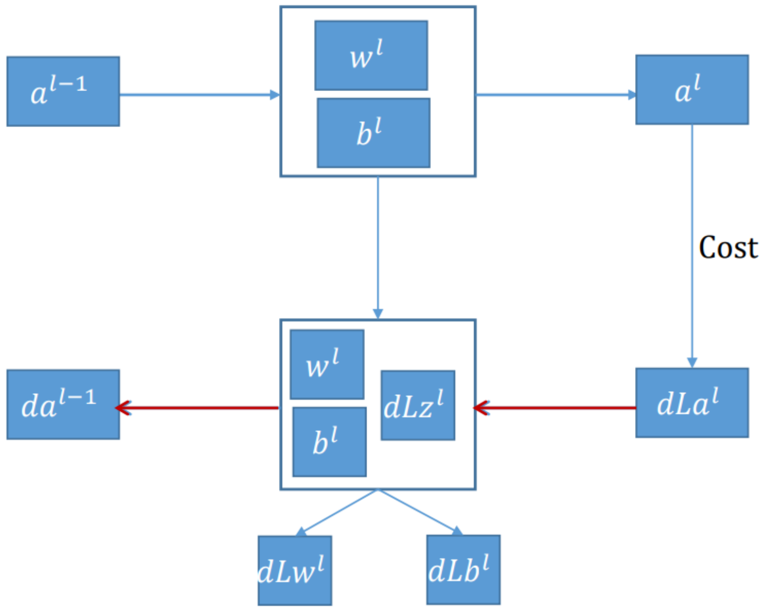
Forward Propagation in L-layer Networks



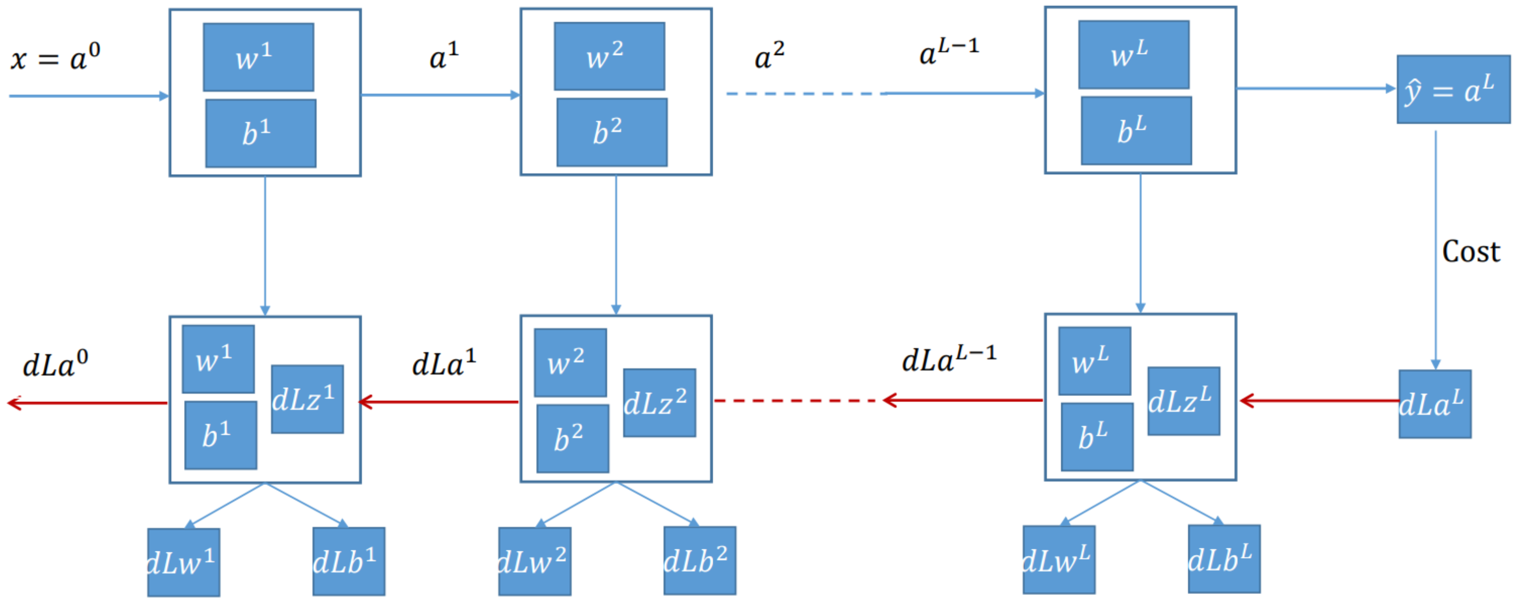


Backward Propagation in L-layer Networks





Complete progression



Algorithm for L layers:

Pseudo code

- For l^{th} layer
  - Forward propagation
    - Input: $a^{[l-1]}$
    - Output: $a^{[l]}$,
    - Function: g($z^{[l]}$) = g($w^{[l]}a^{[l-1]}$+$b^{[l]}$) [where g: sigmoid, tanh, relu]
  - Backward propagation
    - Input: $dLa^{[l]}$
    - Output $dLa^{[l-1]}, dLW^{[l]}, dLb^{[l]}$
    - Functions:
      - $dLa^{[l-1]} = w^{[l]} \times dLz^{[l]}$
      - $dLz^{[l]} = dLa^{[l]} \times g'^{[l]}(z^{[l]})$
      - $dLw^{[l]} = \frac{1}{m} dLz^{[l]} \times a^{[l-1]}$
      - $dLb^{[l]} = \frac{1}{m}\sum_m dLz^{[l]}$ [$Note$: $preserve\ the\ dimension$]

Repeat this for
l = L,L-1,L-2,…,1

Palak Jain