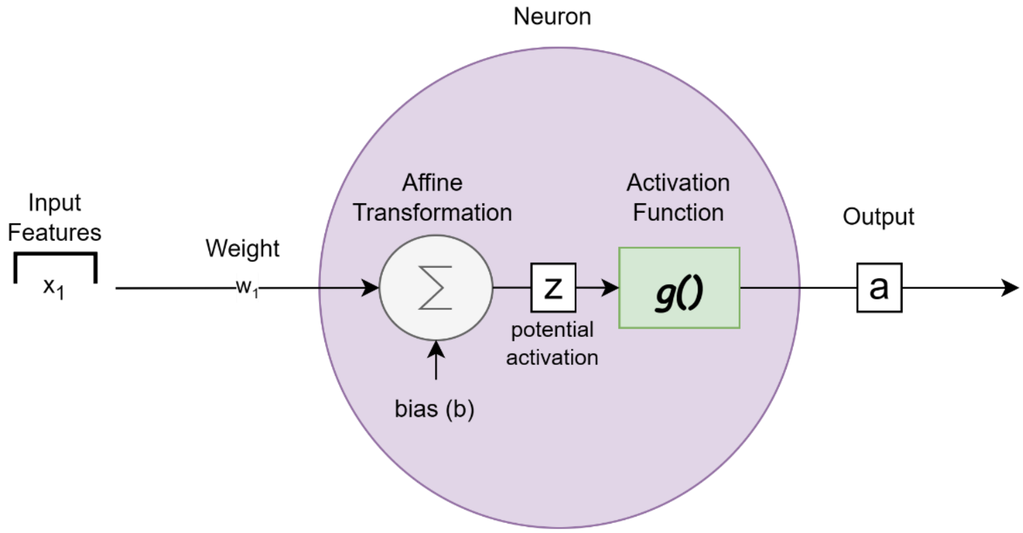
Prepared by: Rajan Adhikari

Artificial Neural Network

1 MATHEMATICAL MODEL OF NEURON



Consider a neuron with only one input features. Let w_1 be the weight of the input. Here, x_1 refers the input and b is the bias to the network.

$$z = w_1 . x_1 + b$$

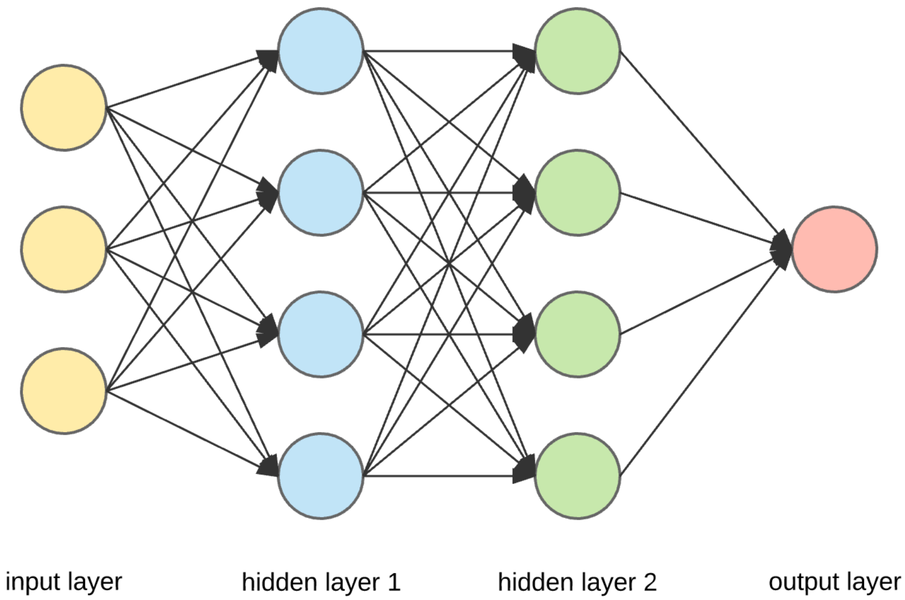
Where z is the activation potential. Now, the activation potential goes through the activation function

$$a = g(z)$$

Where, $g()$ is the activation function and a is the activation value which is also the output of the neuron.

2 NEURAL NETWORK

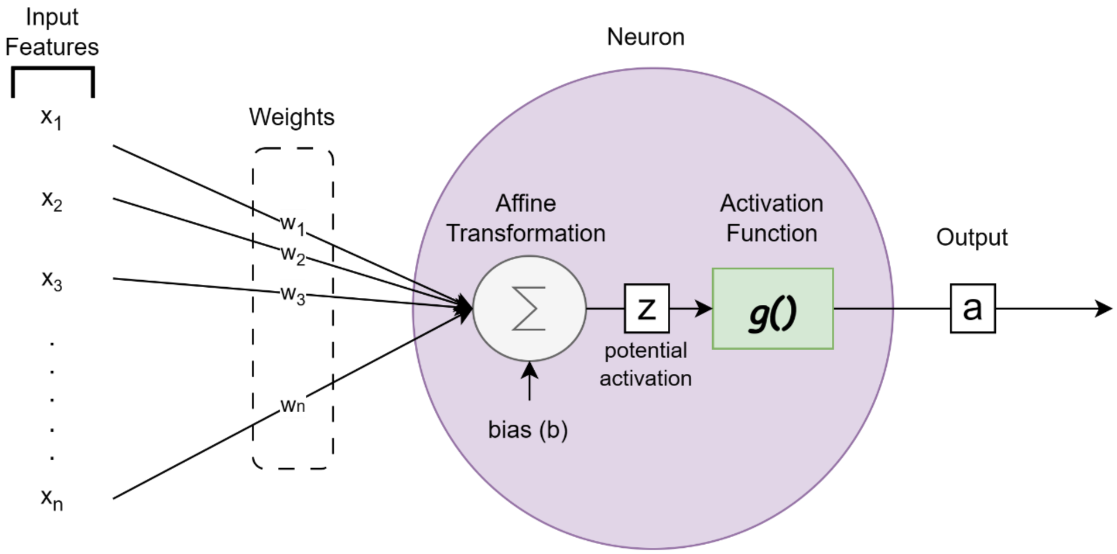
Neural Network is the network of neurons arranged in a structure such that there are multiple neurons in each layer of network. There is a connection from neuron of one layer of network to another layer of network.



3 FORWARD PROPAGATION

Forward propagation is the computational execution of neural network from input to the output. We now construct a neural network step by step increasing the complexity in the network. Let's get started with single neuron but with multiple inputs.

Step 1: Single layered, multiple inputs



Consider a single neuron that has multiple input features $x_1, x_2, x_3, \ldots x_n$ then,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Notation:* x *is a scaler and* ***bolded $\boldsymbol{x}$ is a vector.*

And the weight associated with each input features is $w_1, w_2, w_3, \ldots, w_n$ then, we have weight vectors as:

$$w = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}$$

Later we have to take dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$, so for our ease we take w as row vector. If you are comfortable with $\boldsymbol{w}^{\boldsymbol{T}}.\boldsymbol{x}$, feel free to use was column vector so.

**Just remember, throughout this document, we will be using* $\boldsymbol{w}.\boldsymbol{x}$ *for the dot product and* $\boldsymbol{W}.\boldsymbol{X}$ *for matrix multiplication avoiding the use of transpose.*

Now, for the computation that occurs on neuron, we first compute activation potential through affine transformation as:

$$z = \boldsymbol{w}.\boldsymbol{x} + b$$

Here, **w** and $\boldsymbol{x}$ are the vectors. But b is a bias occurring in a neuron and has no relation with number of input so its always scalar. And, the final output z would be a scalar (How? Check yourself).

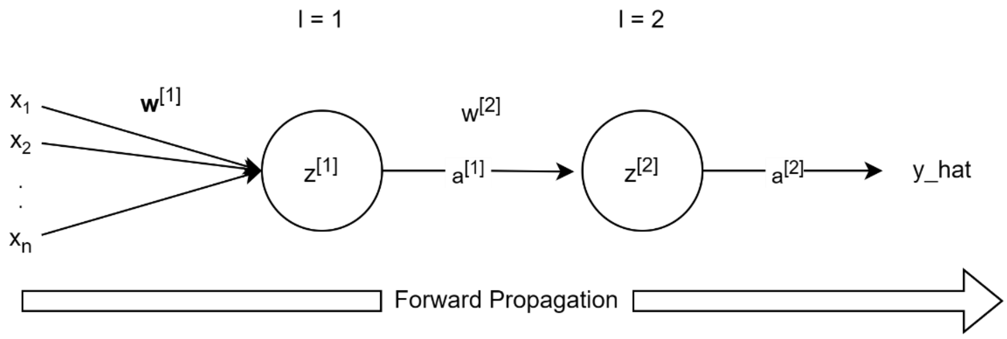
Now, applying activation function on activation potential, we get activation value:

$$a = g(z)$$

Here, a is the activation value and is scalar again, inherited form z.

Step 2: Single neuron in each layer for two-Layered network

Now, let's construct a neural network such that there are multiple layers where each layer contains only one neuron:



Then the forward computation in the first layer $l \ = 1$ would be:

$$z^{[1]} = \boldsymbol{w}^{[\mathbf{1}]}.\boldsymbol{x} + b^{[1]}$$

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

**Remember, Superscript [1] refers to the layer we are computing in but not the power raised.*

Here, $\boldsymbol{x}$ and $\boldsymbol{w}$ are vectors whereas b is scalar. Also, z and a are scalars. Each layer in the neural network is likely to use different activation function thus we are using $g^{[1]}$. Thus,

$\boldsymbol{x}$ is the input vector

$\boldsymbol{w}^{[\mathbf{1}]}$ is the weight vector of layer 1

$b^{[1]}$ is the bias in the neuron of layer 1

$z^{[1]}$ is the activation potential in the neuron of layer 1

$a^{[1]}$ is the activation potential in the neuron of layer 1

And, $g^{[1]}$ is the activation function in layer 1

Now for computation in 2nd layer, we now use the activation value from layer 1 as input and associate new weight to this input. Thus, in the layer l = 2 computation would be:

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

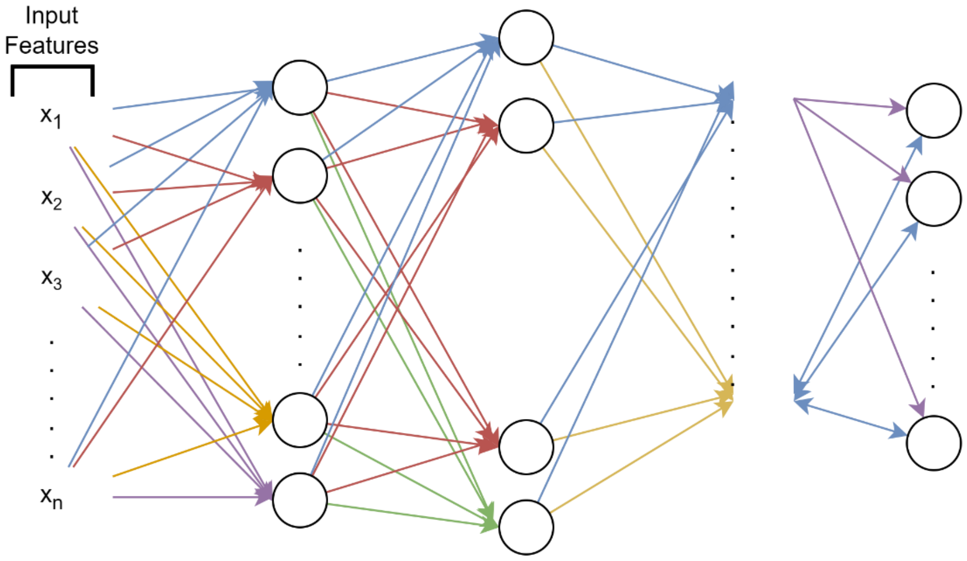
$$a^{[2]} = g^{[2]}(z^{[2]})$$

The activation value of second layer is the output of the neural network (*hence, we can only use Sigmoid or SoftMax activation function in the output layer if the task is classification otherwise, we can use any activation function including linear function if the task is regression*). Then,

$$\hat{y} = a^{[2]}$$

Step 3: Multiple neurons in each layer for two-Layered network

Now, let's construct a neural network such that there are multiple layers and each layer contains multiple neurons:



L-layered Network

Then the forward computation in the first layer $l\ = 1$ would be:

For each neuron $j = 1 \dots n^{[l]}$,

$$\boldsymbol{z}_j^{[1]} = \boldsymbol{w}_j^{[\mathbf{1}]}.\boldsymbol{x} + b_j^{[1]}$$

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

Similarly, for the layer $l\ =\ 2$, for each neuron $j\ =\ 1 \dots n^{[l]}$,

$$\boldsymbol{z}_j^{[1]} = \boldsymbol{w}_j^{[1]}.\boldsymbol{a}^{[1]} + b_j^{[1]}$$

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

And so on for all the layers.

Forward Propagation

Thus, the forward propagation is of the form:

```
$For\ l\ = 1 \dots L\ \{$
    $For\ each\ neuron\ j\ =\ 1 \dots n^{[l]}\ \{$
        $z_j^{[l]} = \boldsymbol{w}_j^{[l]}.\boldsymbol{a}^{[l]} + b_j^{[l]}$
        $a_j^{[l]} = g^{[l]}(z_j^{[l]})$
    $\}$
$\}$
```

Step 4: Vectorization

We form a matrix and vector from weights and biases of neuron of a layer. And, instead of operating individually on each component, we operate directly in the form of vector and matrix. This way, we are able to perform all the computation at once.

For example,

For layer $l\ =\ 1$, the weight input in each neuron is already a vector. So, for $n^{[1]}$ neuron in the first layer of network, we will have the same number of Weight vectors incoming from all the inputs features i.e. each weight vector has a size of $1 \times n$, where n is the size of input features. And if we bring together all $n^{[1]}$ number of vectors into a matrix, we get a size of $n^{[1]} \times n$.

Similarly, each neuron has a single bias but if we stack the biases of all neurons in the first layer, we get a bias vector $n^{[1]} \times 1$.

$$\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}.\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

Here,

input $\boldsymbol{x}$ is a vector of size $n\ \times\ 1$,

$\boldsymbol{W}$ is a weight matrix of size $n^{[1]} \times n$ and

$\boldsymbol{b}$ is a bias vector of size $n^{[1]} \times 1$

Thus $\boldsymbol{W}.\boldsymbol{x}$ produces a vector of size $n^{[1]} \times 1$ and addition with bias b produces the activation potential vector $\boldsymbol{z}$ of size $n^{[1]} \times 1$. We then obtain activation vector $\boldsymbol{a}$ of size $n^{[1]} \times 1$.

The reason $\boldsymbol{b}$, $\boldsymbol{z}$ and $\boldsymbol{a}$ are vectors and **W** a matrix now is we are taking all the neurons of layer at once in the operation to avoid iterative computation. This let us to compute parallelly for all the neurons in layer.

Vectorized Forward Propagation

Now, the vectorized form of forward propagation is:

```
$For\ l\ = 1 \ldots L\ \{$
        $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[\boldsymbol{l}]}.\boldsymbol{a}^{[\boldsymbol{l}-\mathbf{1}]} + \boldsymbol{b}^{[l]}$
        $\boldsymbol{a}^{[l]} = g^{[l]}(\boldsymbol{z}^{[l]})$
$\}$
```

Step 5: Multiple training examples,

We were taking single training examples with n features as of now. Let's consider we have m training examples in our dataset and we take all training examples at once. Then our input would be a matrix of size $n \times m$ and represented by $\boldsymbol{X}$. And as earlier, for layer 1

$\boldsymbol{W}$ is a weight matrix of size $n^{[1]} \times n$ and

$\boldsymbol{b}$ is a bias vector of size $n^{[1]} \times 1$

Then, $\boldsymbol{W}.\boldsymbol{X}$ produces a vector of size $n^{[1]} \times m$ and addition with bias $\boldsymbol{b}$ produces the activation potential vector $\boldsymbol{Z}$ of size $n^{[1]} \times m$. We then obtain activation vector $\boldsymbol{A}$ of size $n^{[1]} \times m$. Now, we can proceed in the same till the last layer.

Vectorized Forward Propagation

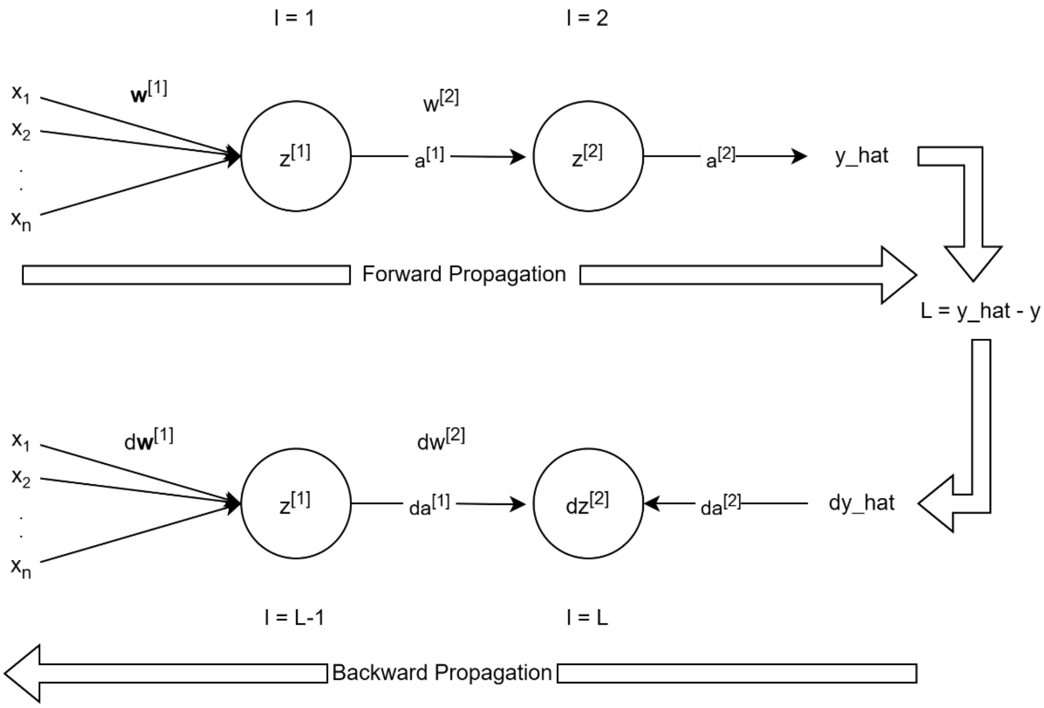
Now, the vectorized form of forward propagation becomes:

```
$For\ l\ = 1 \ldots L\ \{$
        $\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[\boldsymbol{l}]}.\boldsymbol{A}^{[\boldsymbol{l}-\mathbf{1}]} + \boldsymbol{b}^{[l]}$
        $A^{[l]} = g^{[l]}(Z^{[l]})$
}
```

This concludes the forward propagation step by step derivation.

4 BACKWARD PROPAGATION

To understand and derive the basic form of back propagation, consider a neural network as below where each layer in the neural network contains only one neuron and we are taking only two layers in the network:



To make our derivation easy, we consider 2nd layer as L layer and 1st layer as L-1 layer.

Before proceeding to the backward propagation, let's devise the form of forward propagation in each layer:

In layer L-1,

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

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

**Note: We use small case notation for z, w and a which means we are computing for the single neuron in the particular layer.*

Similarly, in layer L,

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

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

Since L-th layer is the final layer in our network,

$$\hat{y} = a^{[L]}$$

And, at the end of forward computation we compute the final output of the neural network as above. Then we compute error for each training example, and, taking all training examples we generally compute an average of errors known as **cost function**.

For regression, cost function can be

$$C = \frac{\sum_{i=1}^{m}(\widehat{y_i} - y_i)}{m}$$

And for classification, cost function can be

$$C = \frac{1}{m}\sum_{i=1}^{m}[-y_i\ log\widehat{y_i} - (1 - y_i)\log(1 - \widehat{y_i})]$$

Where, C is the cost function.

Depending on the nature of problem, we may take any form of cost function.

We can now start the backpropagation on neural network. Through backpropagation we propagate the error devised from the forward propagation in the network by comparing actual value (y) with the predicated value ($\hat{y}$) .

We propagate the error backward in the network so as to update the weight and biases which were input in the neuron. And using gradient descent, we can obtain the new weight and bias for each neuron.

So, let's get started with the backpropagation step by step:

Step 1: First compute the derivate of Error with respect to the $\hat{y}$ because error is computed using y and $\hat{y}$ and y is always the same thus constant for lifetime.

$$\frac{\partial C}{\partial \hat{y}}$$

Derivation of this is totally based on the form of the cost function we use to evaluate the problem. Thus, we won't compute any further derivative of it.

But, to ease on writing notation we will write,

$$d\hat{y} = \frac{\partial C}{\partial \hat{y}} \dots \dots \dots (1)$$

Similarly, for any future computation, if derivative of Cost is computed w.r.t. say p variable, we denote by dp.

$$dp = \frac{\partial C}{\partial p}$$

Now, we compute the derivative in Layer L as below:

Step 2: We compute the derivative of cost with respect to the $a^{[L]}$,

$$da^{[L]} = \frac{\partial C}{\partial a^{[L]}} = \frac{\partial C}{\partial \hat{y}} = d\hat{y} \dots \dots \dots (2)$$

Step 3: Compute the derivative of cost with respect to the $z^{[L]}$,

$$dz^{[L]} = \frac{\partial C}{\partial z^{[L]}} = \frac{\partial C}{\partial a^{[L]}} . \frac{\partial a^{[L]}}{\partial z^{[L]}} = da^{[L]} . a'^{[L]} \dots \dots \dots (3)$$

Here,

$\frac{\partial a^{[L]}}{\partial z^{[L]}} = a'^{[L]}$ because, a is activation value obtained through applying activation function on z. And, depending on activation function, the derivative value will be different. Thus, we denote this by $a'^{[L]}$ in general.

For example, for sigmoid function

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

And, for RELU

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

<u>Step 4</u>: Compute the derivative of cost w.r.t. $w^{[L]}$.

$$dw^{[L]} = \frac{\partial C}{\partial w^{[L]}} = \frac{\partial C}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial w^{[L]}} = dz^{[L]}.\, a^{[L-1]} \qquad [\because We\ obtained\ this\ form\ (3)]\ \ldots\ldots\ldots (4)$$

<u>Step 5</u>: Compute the derivative of cost w.r.t. $b^{[L]}$

$$db^{[L]} = \frac{\partial C}{\partial b^{[L]}} = \frac{\partial C}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial b^{[L]}} = dz^{[L]}.\, 1 = dz^{[L]} \qquad [\because z = wx + b\ and\ from\ (3)]\ \ldots\ldots\ldots (5)$$

<u>Now, we compute the derivative in Layer L-1 as below:</u>

<u>Step 6</u>: Compute the derivative of cost w.r.t. $a^{[L-1]}$.

$$da^{[L-1]} = \frac{\partial C}{\partial a^{[L-1]}} = \frac{\partial C}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial a^{[L-1]}} = dz^{[L]}.\, w^{[L]} \qquad [\because from\ 3]\ \ldots\ldots\ldots (6)$$

Remember,

The sequence of derivatives we performed is based on the forward propagation form in L-th layer:

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

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

This is because backpropagation uses a chain method for computing derivative.

<u>Step 7</u>: Next, compute the derivative of cost w.r.t. $z^{[L-1]}$

$$dz^{[L-1]} = \frac{\partial C}{\partial z^{[L-1]}} = \frac{\partial C}{\partial a^{[L-1]}} \cdot \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} = da^{[L-1]}.\, a'^{[L-1]} \quad [\because from\ (6)]\ \ldots\ldots\ldots (7)$$

<u>Step 8</u>: Again, we compute the derivative of cost w.r.t. $w^{[L-1]}$ and $b^{[L-1]}$ similar as above:

$$dw^{[L-1]} = \frac{\partial C}{\partial w^{[L-1]}} = \frac{\partial C}{\partial z^{[L-1]}} \cdot \frac{\partial z^{[L-1]}}{\partial w^{[L-1]}} = dz^{[L-1]}.\, a^{[L-2]} \quad \ldots\ldots\ldots (8)$$

And,

$$db^{[L-1]} = \frac{\partial C}{\partial b^{[L-1]}} = \frac{\partial C}{\partial z^{[L-1]}} \cdot \frac{\partial z^{[L-1]}}{\partial b^{[L-1]}} = dz^{[L-1]}.1 = dz^{[L-1]}$$

$$= a^{[L]}.a'^{[L]}.w^{[L]}.a'^{[L-1]} \quad \dots\dots\dots(9)$$

And so on for other layers like L-2, L-3… to 1 if they exist.

Now, if we compare the form of backpropagation derivative in weight (w) and bias (b) of L layer and L-1 layer, we notice a pattern.

From above derivations, we can write:

For Layer L

$$dz^{[L]} = da^{[L]}.a'^{[L]} = d\hat{y}.a'^{[L]}$$

$$dw^{[L]} = dz^{[L]}.a^{[L-1]}$$

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

For layer L-1

$$da^{[L-1]} = dz^{[L]}.w^{[L]}$$

$$dz^{[L-1]} = da^{[L-1]}.a'^{[L-1]} = dz^{[L]}.w^{[L]}.a'^{[L-1]}$$

$$dw^{[L-1]} = dz^{[L-1]}.a^{[L-2]}$$

$$db^{[L-1]} = dz^{[L-1]}$$

And, if layer L-2 exists, we can write,

$$dz^{[L-2]} = dz^{[L-1]}.W^{[L-1]}.a'^{[L-2]}$$

$$dw^{[L-2]} = dz^{[L-2]}.a^{[L-3]}$$

$$db^{[L-2]} = dz^{[L-2]}$$

Back Propagation

So, in easy term, we can write

For $l \,=\, L$ i.e. last layer

$$dz^{[L]} = d\hat{y}.a'^{[L]}$$

$$dw^{[L]} = dz^{[L]}.a^{[L-1]}$$

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

for $l = L - 1\ to\ 1$ i.e. except last layer

$$dz^{[l]} = dz^{[l+1]}.w^{[l+1]}.a'^{[l]}$$

$$dw^{[l]} = dz^{[l]}.a^{[l-1]}$$

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

But, in general neural network, we won't have single neuron in each layer. Instead, only the L layer depicts the form as above, and from L-1 to 1st layer depicts form similar to below:

Following two case arises from the above:

1. From Lth layer, the error is back propagated to all the neuron L-1 layer and from L-1 layer to L-2 layer and so on. The back propagated error from l^{th} is passed as a whole to each neuron of $(l - 1)$ layer.
2. Also, each neuron in L-1 layer receives back propagated error from each neuron in L layer as shown in above diagram i.e. each neuron in L-1 layer accumulates error from all the neuron in Lth layer. So, we can say,

   For neuron i, in layer $l = L - 1 \dots 1$,

$$\sum_{j=1}^{n^{[l]}} dz_j^{[l+1]}.w_{ij}^{[l+1]}$$

   is the accumulated error through the back propagation. This changes the form of equations we derived above except the last layer as it remains same. Here $n^{[l]}$ is the number of neurons in that particular layer.

Backpropagation Formula (*Memorize*)

Finally, we get our required form for back propagation in iterative form as:

For $l \,=\, L$ i.e. last layer, for each neuron j

$$dz^{[L]} = d\hat{y}.a'^{[L]}$$

$$dw^{[L]} = dz^{[L]}.a^{[L-1]}$$

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

for $l = L - 1\ to\ 1$ i.e. except last layer, for each neuron j

$$dz^{[l]} = a'^{[l]} \sum_{j=1}^{n^{[l]}} dz_{ij}^{[l+1]}.w_{ij}^{[l+1]}$$

$$dw^{[l]} = dz^{[l]}.a^{[l-1]}$$

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

Now, we can apply weight update rule using gradient descent.

$$w = w - \alpha\frac{\partial C}{\partial w} = w - \alpha.dw$$

$$b = b - \alpha\frac{\partial C}{\partial b} = b = \alpha.db$$

Back Propagation Algorithm:

1. Randomly initialize the weight and bias of all neurons in the network.
2. Repeat until convergence:

   for each training tuple **x** in X,

   a. forward propagation

      For each layer l in the network for each neuron j, compute

      $$z = \boldsymbol{w}^{[l]}\boldsymbol{x} + b$$

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

   b. backward propagation

      For $l = L$ i.e. last layer, for each neuron j

      $$dz^{[L]} = d\hat{y}.a'^{[L]}$$

      $$dw^{[L]} = dz^{[L]}.a^{[L-1]}$$

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

      for $l = L - 1\ to\ 1$ i.e. except last layer, for each neuron j

$$dz^{[l]} = a'^{[l]} \sum_{j=1}^{n^{[l]}} dz_{ij}^{[l+1]} . w_{ij}^{[l+1]}$$

$$dw^{[l]} = dz^{[l]} . a^{[l-1]}$$

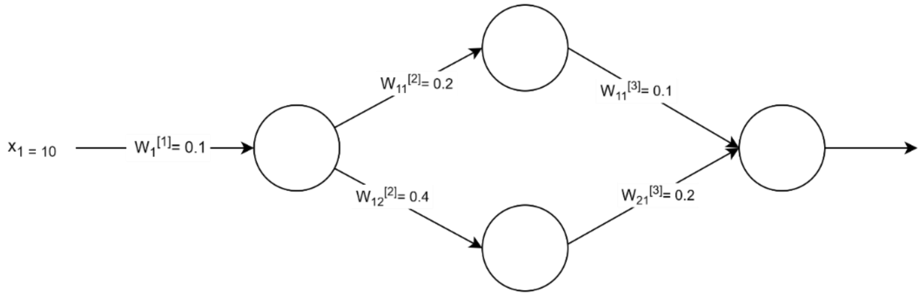
$$db^{[l]} = dz^{[l]}$$

3. Update weight and biases of each neuron for all layer in the network:

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

$$b^{[l]} = b^{[l]} - \alpha . db^{[l]}$$

i) **Numerical Problem 1:** Given a neural network, perform forward propagation considering a classification problem and Compute the loss. Consider a sigmoid activation function and learning rate 0.1.



Where (x, y) = (1, 0).

Solution:

There are three layers in the network. And, at first let's execute forward propagation. Since there is only one training example, we use a vector form and for our ease on calculation we ignore bias. Thus,

In layer 1:

w is a scaler as we have only one input thus, $w = 0.1$. And $x = a^{[0]}$.

$$a^{[1]} = sigmoid\left(wa^{[0]}\right) = 1 * 0.2 = sigmoid(0.2) = 0.55$$

In layer 2:

$\boldsymbol{w}$ is a vector as we have two neurons in layer 2. Thus,

$$\boldsymbol{w} = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$\boldsymbol{a}^{[\mathbf{2}]} = sigmoid\left(\boldsymbol{w}a^{[1]}\right) = \ sigmoid\left(\boldsymbol{w}.a^{[\mathbf{1}]}\right) = sigmoid\left(\left[\begin{bmatrix}0.2\\0.4\end{bmatrix}.(0.55)\right]\right)$$

$$= sigmoid\left(\begin{bmatrix}0.2*0.55\\0.4*0.55\end{bmatrix}\right) = sigmoid\left(\begin{bmatrix}0.11\\0.22\end{bmatrix}\right) = \begin{bmatrix}0.53\\0.55\end{bmatrix}$$

<u>Now, in layer 3:</u>

w is again a vector as two inputs are there for a single neuron. Thus

$$w = [0.1 \quad 0.2]$$

$$a^{[3]} = sigmoid\left(\boldsymbol{w}.\boldsymbol{a}^{[\mathbf{2}]}\right) = sigmoid([0.1 \quad 0.2].\begin{bmatrix}0.53\\0.55\end{bmatrix})$$

$$= sigmoid(0.1*0.53+0.2*0.55) = sigmoid(0.163) = 0.54$$

The final output of the network is 0.54.

Now, computing the loss using log loss function

$$C = -ylog\hat{y} - (1-y)\log(1-\hat{y})$$

$$= \ -(0*\log(0.54)) - (1-0)*\ log*(1-0.54) = \ 0.337242168$$