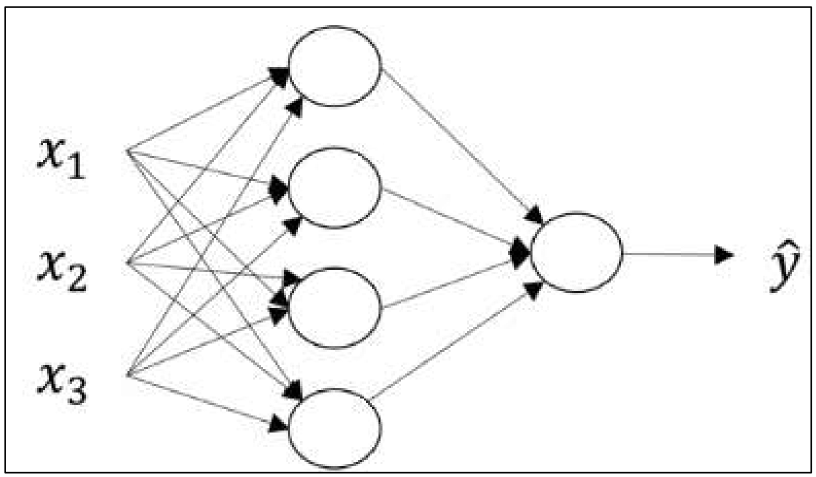
Deep Learning-II

(Shallow Neural Networks)

DR. JASMEET SINGH
ASSISTANT PROFESSOR,
CSED, TIET

Shallow Neural Networks

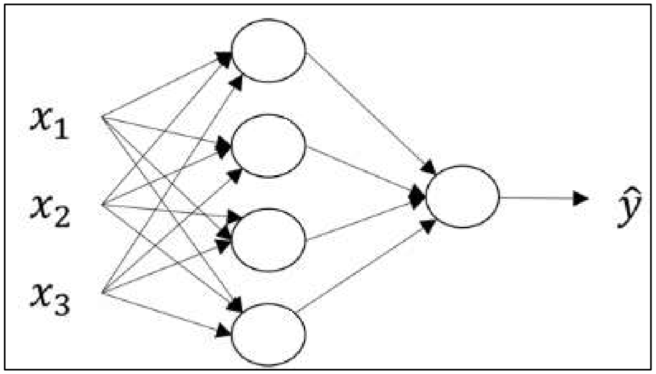
- Shallow neural networks consist of only 1 or 2 hidden layers.
- The figure below shows a shallow neural network with 1 hidden layer, 1 input layer and 1 output layer.



Notations

In general, following notations are used for the shallow neural network (shown in the figure):

1. L- number of layers.
2. A_L-Activations at L^{th} layer (produced by applying activation function on weighted sum of inputs); A_0 are the activations of input layer (which are the input feature values).
3. W_L is the weight matrix at the L^{th} layer (L>0).
4. b_L is the bias matrix at the L^{th} layer (L>0)



In the given diagram, if n0, n1, and n2 nodes are present in each layer, then the dimensions of different matrices (for one example) are:

- a_0: n_0 X 1, a_1:n_1 X 1, a_2: n_2 X 1
- W_1:n_1Xn_0, W_2:n_2Xn_1
- b_1: n_1X1, b_2: n_2X1

For n examples,

A_0: n_0 X n, A_1:n_1 X n, A_2: n_2 X n

Forward Propagation

- In the forward propagation, the predicted values (y^=A_2) and the cost function is computed.
- For one example, these are computed as:

$$z_1 = W_1 a_0 + b_1$$
$$a_1 = g_1(z_1)$$
$$z_2 = W_2 a_1 + b_2$$
$$a_2 = g_2(z_2)$$

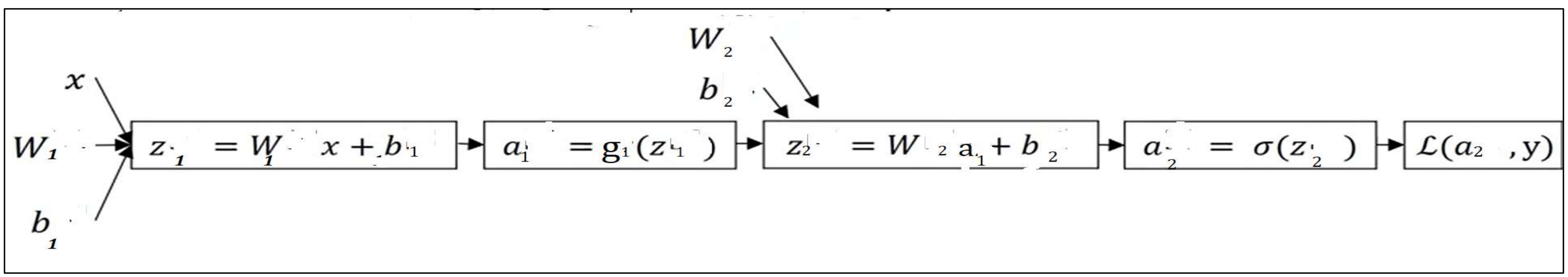
Where g_1 and g_2 are the activation functions applied at the first and second layer.

- For n examples, the activations at layer 1 (A_1) and layer 2 (A_2) are computed as:

$$Z_1 = W_1 A_0 + b_1$$
$$A_1 = g_1(Z_1)$$
$$Z_2 = W_2 A_1 + b_2$$
$$A_2 = g_2(Z_2)$$

Backpropagation

- In the backpropagation mode, gradients of cost function J w.r.t W1, W2, b1, and b2 needs to be computed i.e. $\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial b_2}$.
- In order to understand, how these gradients are computed, consider the shallow neural network drawn in Slide 1 for one training example.
- The first two rectangles shows hidden layer with g1 activation function; and next two rectangle shows output layer with sigmoid activation function (for binary classification); and L denotes binary cross entropy function.



Backpropagation (Contd……)

$$\frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{a_2}} = \frac{\partial - (y log a_2 + (1-y)\log(1-a_2)}{\partial a_2} = \frac{-y}{a_2} + \frac{1-y}{1-a_2} = \frac{\boldsymbol{a_2} - \boldsymbol{y}}{\boldsymbol{a_2}(\boldsymbol{1} - \boldsymbol{a_2})}$$

Now, $\frac{\partial L(a_2,y)}{\partial a_2}$ will be used to compute $\frac{\partial L(a_2,y)}{\partial z_2}$

$$\frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{z_2}} = \frac{\partial L(a_2,y)}{\partial a_2} . \frac{\partial a_2}{\partial z_2} = \frac{a_2 - y}{a_2(1-a_2)} . \frac{e^{-z_2}}{(1+e^{-z_2})^2} = \frac{a_2 - y}{a_2(1-a_2)} . \frac{1}{(1+e^{-z_2})}\left(1 - \frac{1}{(1+e^{-z_2})}\right)$$

$$= \frac{a_2 - y}{a_2(1-a_2)} . a_2(1-a_2) = \boldsymbol{a_2} - \boldsymbol{y}$$

Now, $\frac{\partial L(a_2,y)}{\partial z_2}$ will be used to compute $\frac{\partial L(a_2,y)}{\partial W_2}$, $\frac{\partial L(a_2,y)}{\partial b_2}$, $\frac{\partial L(a_2,y)}{\partial a_1}$

$$\frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{W_2}} = \frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{z_2}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{W_2}} = (\boldsymbol{a_2} - \boldsymbol{y})\boldsymbol{a_1^T}$$

$$\frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{b_2}} = \frac{\partial \boldsymbol{L}(\boldsymbol{a_2},\boldsymbol{y})}{\partial \boldsymbol{z_2}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{b_2}} = (\boldsymbol{a_2} - \boldsymbol{y})$$

Backpropagation (Contd……)

$$\frac{\partial L(a_2,y)}{\partial a_1} = \frac{\partial L(a_2,y)}{\partial z_2} . \frac{\partial z_2}{\partial a_1} = \boldsymbol{W_2^T(a_2 - y)}$$

Now, $\frac{\partial L(a_2,y)}{\partial a_1}$ will be used to compute $\frac{\partial L(a_2,y)}{\partial z_1}$

$$\frac{\partial L(a_2, y)}{\partial z_1} = \frac{\partial L(a_2, y)}{\partial a_1} . \frac{\partial a_1}{\partial z_1} = \boldsymbol{W_2^T(a_2 - y) * g_1'(z_1)}$$

[* denotes element-wise multiplication]

Now, $\frac{\partial L(a_2,y)}{\partial z_1}$ will be used to compute $\frac{\partial L(a_2,y)}{\partial W_1}, \frac{\partial L(a_2,y)}{\partial b_1}$

$$\boldsymbol{\frac{\partial L(a_2,y)}{\partial W_1} = \frac{\partial L(a_2,y)}{\partial z_1} . \frac{\partial z_1}{\partial W_1} = W_2^T(a_2 - y) * g_1'(z_1)x^T}$$

$$\boldsymbol{\frac{\partial L(a_2,y)}{\partial b_1} = \frac{\partial L(a_2,y)}{\partial z_1} . \frac{\partial z_1}{\partial b_1} = W_2^T(a_2 - y) * g_1'\ (z_1)}$$

Backpropagation Summary

For one example,	For n examples,
$\frac{\partial L(a_2,y)}{\partial z_2} = a_2 - y$	$\frac{\partial J}{\partial z_2} = dz_2 = A_2 - Y$
$\frac{\partial L(a_2,y)}{\partial W_2} = (a_2 - y)a_1^T$	$\frac{\partial J}{\partial W_2} = dW_2 = \frac{1}{n} dz_2 . A_1^T$
$\frac{\partial L(a_2,y)}{\partial b_2} = (a_2 - y)$	$\frac{\partial J}{\partial b_2} = db_2 = \frac{1}{n} np.sum(dz_2, axis = 1, keepdims = True)$
$\frac{\partial L(a_2,y)}{\partial z_1} = W_2^T(a_2 - y) * g_1'\ (z_1)$	$\frac{\partial J}{\partial z_1} = dz_1 = W_2^T(A_2 - Y) * g_1'(Z_1)$
$\frac{\partial L(a_2,y)}{\partial W_1} = W_2^T(a_2 - y) * g_1'\ (z_1)x^T$	$\frac{\partial J}{\partial W_1} = dw_1 = \frac{1}{n} dz_1 . X^T$
$\frac{\partial L(a_2,y)}{\partial b_1} = W_2^T(a_2 - y) * g_1'\ (z_1)$	$\frac{\partial J}{\partial b_1} = db_1 = \frac{1}{n} np.sum(dz_1, axis = 1, keepdims = True)$

Random Initialization

- For logistic regression, it was okay to initialize the weights to zero. But for a neural network of initialize the weights to parameters to all zero and then applied gradient descent, it won't work.
- Let ***W1***, the weight matrix of layer 1 and ***W2***, the weight matrix of layer 2 be initialized with 0.
- Now, if the weight matrices are the same, the activations of neurons in the hidden layer would be the same. Moreover, the derivatives of the activations would be the same.
- Therefore, the neurons in that hidden layer would be modifying the weights in a similar fashion i.e. there would be no significance of having more than 1 neuron in a particular hidden layer.
- But, we don't want this. Instead, we want that each neuron in the hidden layer to be unique, have different weight and work as a unique function. Therefore, we initialize the weights randomly.

Random Initialization (Contd....)

The best method of initialization is ***Xavier's Initialization***. Mathematically it is defined as:

$$W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}\right)$$

$$b^{[l]} = 0$$