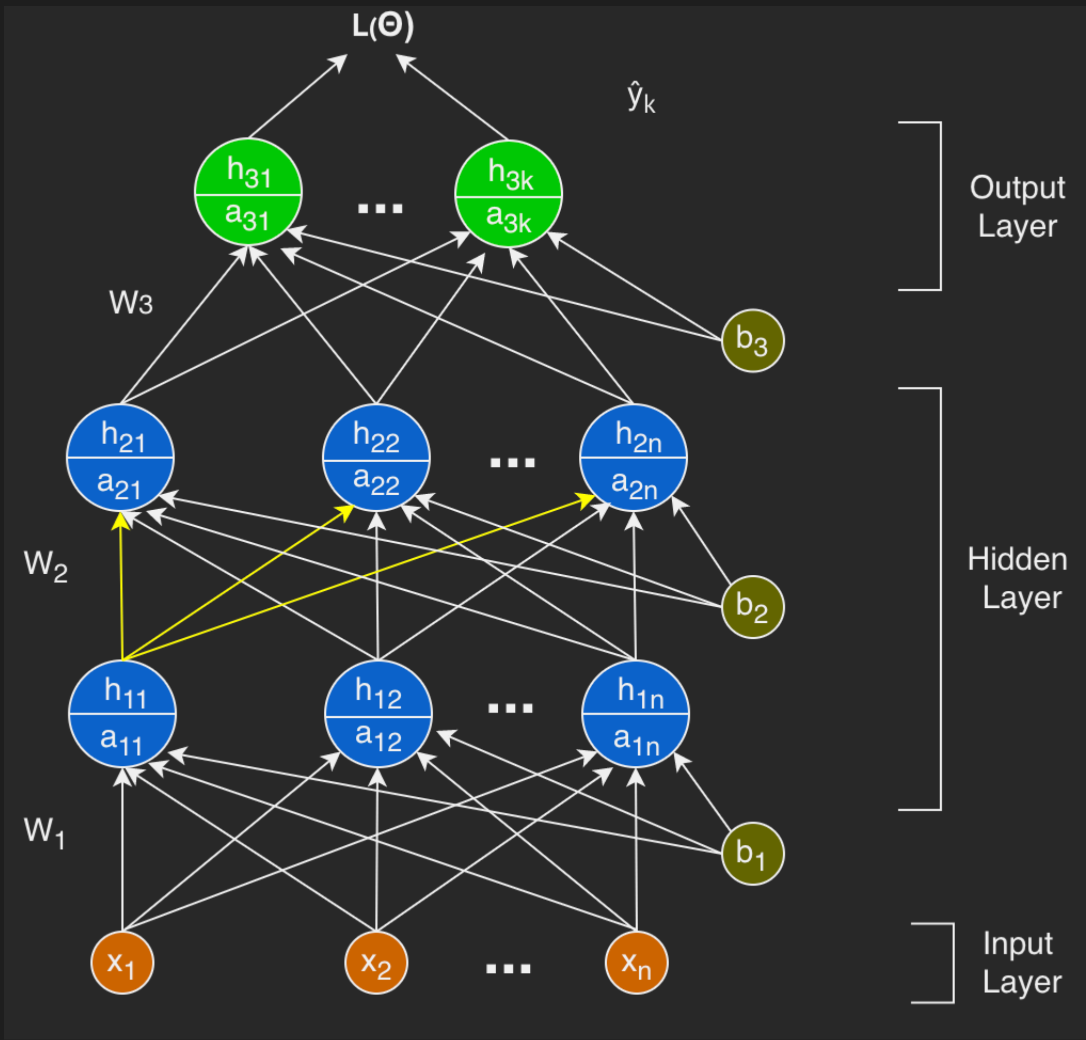
Understanding a Simple Neural Networks Learning for Multi-Class Classification -Maths Version



We have learned through a simple Neural Networks for Binary Classification in a separate blogs. Now we will try to understand the working of Neural Networks for Multi-Class Classification. For the consistency of the paramters names across all layers we have used 'h' at the outer layer too.

Summarizing Feed Forward parameters for Multi-Class Classification:

Hidden Layer 1

Input/Output Parameters	Parameter Expressions	Parameter Description

X_1	$\begin{bmatrix} x_1 \\ x_2 \\ . \\ x_n \end{bmatrix}$	Input vector (different features)
W_1	$\begin{bmatrix} w_{111} & w_{112} & \dots & w_{11n} \\ w_{121} & w_{122} & \dots & w_{12n} \\ \dots & \dots & \dots & \dots \\ w_{1m1} & w_{1m} & \dots & w_{1mn} \end{bmatrix}$	W_1 matrix size is m x n, where, m = number of inputs and n = number of neurons
b_1	$\begin{bmatrix} b_{11} \\ b_{12} \\ \dots \\ b_{1n} \end{bmatrix}$	b_1 is a vector with size as n, number of neuron in layer 1
a_{1n}	$\begin{bmatrix} a_{11} \\ a_{12} \\ \dots \\ a_{1n} \end{bmatrix} = W_1 * X_1 + b_1$	A pre-activation function vector of size n, number of neurons in layer 1. The Matrix multiplication size will be, [n * m] * [m * 1] + [n * 1] = [n*1]
h_{1n}	$\begin{bmatrix} h_{11} \\ h_{12} \\ \dots \\ h_{1n} \end{bmatrix} = \begin{bmatrix} g(a_{11}) \\ g(a_{12}) \\ \dots \\ g(a_{1n}) \end{bmatrix}$	A activation function vector of size n, number of neurons in layer 1. The activation can be any non-linear continous function like Sigmoid, tanh etc.

If the function to be considered is Sigmoid then,

$g(a_{11}) = \frac{1}{1+e^{-a_{11}}}$ and so on.

Hidden Layer 2

Input/Output Parameters	Parameter Expressions	Parameter Description
W_2	$\begin{bmatrix} w_{211} & w_{212} & \dots & w_{21n} \\ w_{221} & w_{222} & \dots & w_{22n} \\ \dots & \dots & \dots & \dots \\ w_{2m1} & w_{2m2} & \dots & w_{2mn} \end{bmatrix}$	W_2 matrix size is m x n, where, m = number of previous layer neurons and n = number of neurons in the current layer

b_2	$\begin{bmatrix} b_{21} \\ b_{22} \\ \dots \\ b_{2n} \end{bmatrix}$	b_2 is a vector with size as n, number of neuron in layer 2
a_{2n}	$\begin{bmatrix} a_{21} \\ a_{22} \\ \dots \\ a_{2n} \end{bmatrix} = W_2 * h_1 + b_2$	A pre-activation function vector of size n, number of neurons in layer 2. The Matrix multiplication size will be [n * m] * [m * 1] + [n*1] = [n * 1]
h_{2n}	$\begin{bmatrix} h_{21} \\ h_{22} \\ \dots \\ h_{2n} \end{bmatrix} = \begin{bmatrix} g(a_{21}) \\ g(a_{22}) \\ \dots \\ g(a_{2n}) \end{bmatrix}$	A activation function vector of size n, number of neurons in layer 2. The activation can be any non-linear continous function like Sigmoid, tanh etc.

If the function to be considered is Sigmoid then,

$g^{'}(a_{21}) = \frac{1}{1+e^{-a_{21}}}$ and so on.

Output Layer

Input/Output Parameters	Parameter Expressions	Parameter Description
W_3	$\begin{bmatrix} w_{311} & w_{312} & \dots & w_{31n} \\ w_{321} & w_{322} & \dots & w_{32n} \\ \dots & \dots & \dots & \dots \\ w_{3m1} & w_{3m2} & \dots & w_{3mn} \end{bmatrix}$	W_3 matrix size is m x n, where, m = number of previous layer neurons and n = number of neurons in the current layer(output layer).
b_3	$\begin{bmatrix} b_{31} \\ b_{32} \\ \dots \\ b_{3n} \end{bmatrix}$	b_2 is a vector with size as n, number of neuron in layer 2.
a_{3n}	$\begin{bmatrix} a_{31} \\ a_{32} \\ \dots \\ a_{3n} \end{bmatrix} = W_3 * h_2 + b_3$	A pre-activation function vector of size n, number of neurons in output layer. The Matrix multiplication size will be [n * m] * [m * 1] + [n*1] = [n * 1].
		A activation function vector of

h_{3n}	$\begin{bmatrix} h_{31} \\ h_{32} \\ \dots \\ h_{3n} \end{bmatrix} = \begin{bmatrix} softmax(a_{31}) \\ softmax(a_{32}) \\ \dots \\ softmax(a_{3n}) \end{bmatrix}$	size n, number of neurons in output layer. The activation can be any non-linear continous function like Sigmoid, tanh etc. Here we will use softmax function.

For softmax,

$softmax(a_{31}) = \frac{e^{a_{31}}}{\sum e^{a_i}}$ where *i*, ranges from 1 to n, n being the number of neurons at the output layer.

So if you combine the above layer equations it will result into below:

$\hat{y} = f(x) = O(W_3 g(W_2 g(W_1 X_1 + b_1) + b_2) + b_3)$, where O can be a softmax or any other non-linear continous function.

CROSS ENTROPY LOSS FUNCTION for Multi-Class Classification is given by:

$L(\Theta) = -[(1-y)log(1-\hat{y}) + ylog\hat{y}]$