1 Foreword

Author : Anderson Chau

Disclaimer : The notes are written only for my understanding and memorization purpose after I have self-studied those online lecture notes.
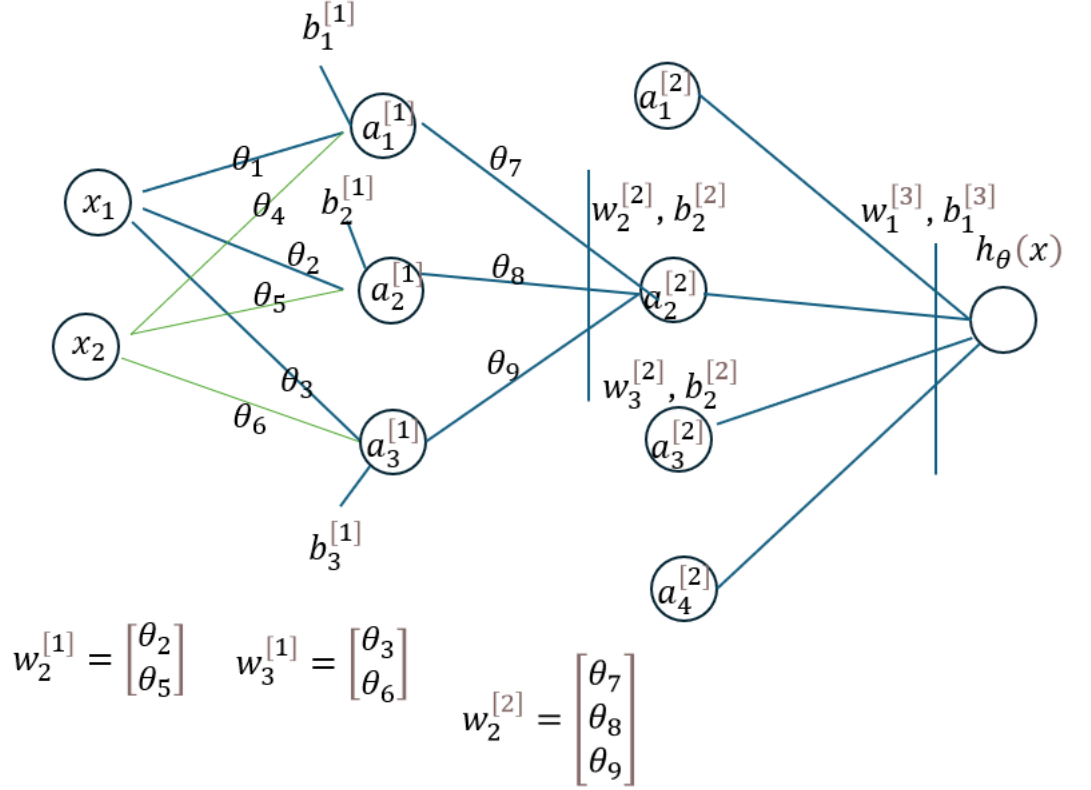
2 Main Idea

The old trick : (i) Forward feed (ii) Compute Loss (NN output vs training data) (iii) Backpropagation with Gradient Descent to tune parameters
2. Use activation functions to avoid the model to be a pure linear model, which is useless (just ax+b)
3. Examples of activation functions : Signmoid , (Leaky) ReLU, tanh etc.

3   ANN Structure

$$w_2^{[1]} = \begin{bmatrix} \theta_2 \\ \theta_5 \end{bmatrix} \quad w_3^{[1]} = \begin{bmatrix} \theta_3 \\ \theta_6 \end{bmatrix} \quad w_2^{[2]} = \begin{bmatrix} \theta_7 \\ \theta_8 \\ \theta_9 \end{bmatrix}$$

w's and b's are parameters : Totally ther are Hidden Layer 1 (layer 1) (2x3 + 3) + Hidden Layer 1 (layer 2) (3x4 + 4) + Output Layer (layer 3)(4x1 +1) number of parameters

4   Forward Feed (see above NN diagram)

$$a_1 = \mathrm{ReLU}(\theta_1 x_1 + \theta_4 x_2 + b_1^{[1]})$$

$$a_2 = \mathrm{ReLU}(\theta_2 x_1 + \theta_5 x_2 + b_2^{[1]})$$

Let i be i^{th} layer and j be j^{th} neuron (count vertifically) this layer of NN
Rewriting the notation : $w_j^{[i]}$ as **VECTOR** of input θ's for layer i and j^{th} neuron.
With the above, Layer 1 of NN can be expressed as:

For all $j \in [1, \ldots, m]$: (m=3, count vertically, is the number of number of

neuron layer 1)

$$z_j = \mathbf{w}_j^{[1]\top}\mathbf{x} + b_j^{[1]} \quad \text{where} \quad \mathbf{w}_j^{[1]} \in R^d, \ b_j^{[1]} \in R$$

(d = 2 , count vertically, is the number of number of previous layer , layer 0)

$$a_j = \text{ReLU}(z_j),$$

$$\mathbf{a} = [a_1, \ldots, a_m]^\top \in R^m$$

For Layer 3 :

$$\bar{h}_\theta(\mathbf{x}) = \mathbf{w}^{[3]\top}\mathbf{a} + b^{[3]} \quad \text{where} \quad \mathbf{w}^{[3]} \in R^n, \ b^{[3]} \in R$$

(n = 4, count vertically, is the number of number of neuron layer, layer 2)

5 Vectorization of Forward Feeding

Just by stacking the parameters . for layer 1 :

$$W^{[1]} = \begin{bmatrix} -w_1^{[1]T}- \\ -w_2^{[1]T}- \\ \vdots \\ -w_m^{[1]T}- \end{bmatrix} \in R^{m \times d}$$

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \in \mathbb{R}^{m \times 1} = \begin{bmatrix} -w_1^{[1]T}- \\ -w_2^{[1]T}- \\ \vdots \\ -w_m^{[1]T}- \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}$$

Hence we get:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = ReLU(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = ReLU(z^{[2]})$$
$$h = W^{[3]}a^{[2]} + b^{[3]}$$

In General :

$$a^{[1]} = \text{ReLU}(W^{[1]}x + b^{[1]})$$
$$a^{[2]} = \text{ReLU}(W^{[2]}a^{[1]} + b^{[2]})$$
$$\vdots$$
$$a^{[r-1]} = \text{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]})$$
$$h(x) = W^{[r]}a^{[r-1]} + b^{[r]}$$

6   Vectorized Backpropagation

It is just matrix calculus + chain rule + gradient descent combined.