Mathematics and Artificial Neural Network

Giorgi Kakulashvili

May 5, 2018

Contents

1 Artificial Neural Network ... 2
1.1 Forward Propagation ... 2
1.2 Backward Propagation ... 3

1. Artificial Neural Network

Suppose we have two finite sets of vectors in X and Y

$$X = \left\{ \left(x_1^j, x_2^j, \dots, x_n^j \right) : 1 \le j \le N_x,\ x_i^j \in [0,1] \right\}$$

$$Y = \left\{ \left(y_1^j, y_2^j, \dots, y_m^j \right) : 1 \le j \le N_y,\ y_i^j \in [0,1] \right\}$$

and there exists surjective mapping

$$f : X \to Y$$

such that for all $x \in X$ we know corresponding $f(x) = y \in Y$, then we can construct $\hat{f}$ function, which will approximate f function.

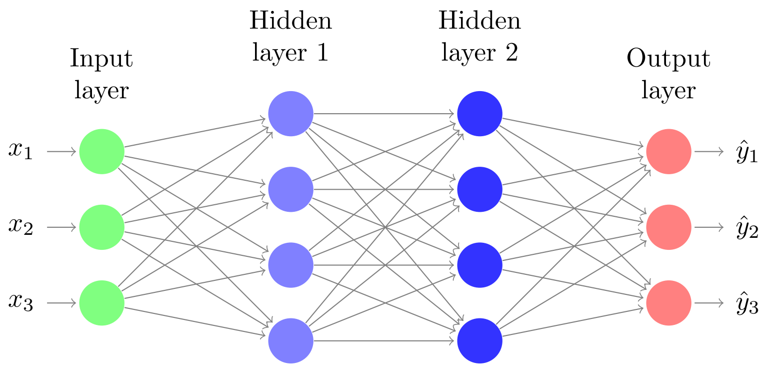
For simplicity we'll be using X and Y as matrices

$$X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^{N_x} \\ x_2^1 & x_2^2 & \cdots & x_2^{N_x} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & \cdots & x_n^{N_x} \end{bmatrix} \quad and \quad Y = \begin{bmatrix} y_1^1 & y_1^2 & \cdots & y_1^{N_y} \\ y_2^1 & y_2^2 & \cdots & y_2^{N_y} \\ \vdots & \vdots & \ddots & \vdots \\ y_m^1 & y_m^2 & \cdots & y_m^{N_y} \end{bmatrix}$$

Let X_j and Y_j be corresponding columns, then for all i, there exists j, such that

$$f(X_i) = Y_j$$

For constructing $\hat{f}$ function we are using *Neural Network*, which is commonly visualized like



Here first and last layers are inputs and output and number of neurons depend on vector size, while *hidden layers* are chosen. Suppose we have $L+1$ layers, with n_l neurons in l layer.

For given X_i vector calculation of $\hat{f}(X_i)$ is *forward propagation*

1.1. Forward Propagation

All nodes are connected with each other. Each connection has weight and each neuron, expect inputs, have biases. To get values of neurons in layer $l+1$, we need to do following opertaions

$$z^{(l)} = \begin{bmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_{n_l}^{(l)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{bmatrix} \times \begin{bmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{n_{l-1}}^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{bmatrix} = w^{(l)} a^{(l-1)} + b^{(l)}$$

and by activation function σ, we have

$$a^{(l)} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{bmatrix} = \begin{bmatrix} \sigma\left(z_1^{(l)}\right) \\ \sigma\left(z_2^{(l)}\right) \\ \vdots \\ \sigma\left(z_{n_l}^{(l)}\right) \end{bmatrix} = \sigma\left(z^{(l)}\right), \quad a^{(0)} = X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_0} \end{bmatrix}, \quad a^{(L)} = \hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n_L} \end{bmatrix}$$

Here we take activation function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

which maps $\mathbb{R}$ to $(0,1)$, and its derivitive is

$$\sigma'(x) = \sigma(x)\left(1-\sigma(x)\right)$$

1.2. Backward Propagation

As we can see from *Forward Propagation*, artificial neural network is continuous multidimensional function

$$f : X \to Y, \quad \hat{f} : X \to \hat{Y}, \quad \hat{f} \approx f$$

Output of the function depends on weights and biases, so we need to adjust them. Since our function is divided into layers, we have cost function on each of them

$$C_0 = \frac{1}{2}\|\hat{y} - y\|^2 = \frac{1}{2}\sum_j \left(\hat{y}_j - y_j\right)^2, \quad C_k = \frac{1}{2}\left\|a^{(L-k)} - \tilde{a}^{(L-k)}\right\|^2 = \frac{1}{2}\left\|\delta^{(L-k)}\right\|^2, \quad C = \sum_{k=0}^{L-1} C_k$$

C function depends on $w_{jk}^{(l)}$ and $b_j^{(l)}$ parameters and by tilde symbol we denote that it is corrected version. Our goal is to minimize cost function, and for this we use *gradient descent* method. For this we need to calculate partial derivatives.

In output layer we have following equations

$$a_j^{(L)} = \sigma\left(z_j^{(L)}\right) \tag{1.1}$$

$$z_j^{(L)} = \sum_{k=1}^{n_{L-1}} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)} \tag{1.2}$$

$$C_0 = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^{(L)} - y_j\right)^2 \tag{1.3}$$

And by chain rule we get following

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C_0}{\partial a_j^{(L)}} = a_k^{(L-1)}\sigma'\left(z_j^{(L)}\right)\left(a_j^{(L)} - y_j\right) = a_k^{(L-1)}\sigma'\left(z_j^{(L)}\right)\delta_j^{(L)}$$

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C_0}{\partial a_j^{(L)}} = \sigma'\left(z_j^{(L)}\right)\delta_j^{(L)}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=1}^{n_L}\frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C_0}{\partial a_j^{(L)}} = \sum_{j=1}^{n_L} w_{jk}^{(L)}\sigma'\left(z_j^{(L)}\right)\delta_j^{(L)}$$

If we rewrite it in matrix form, we get

$$\frac{\partial C_0}{\partial w^{(L)}} = \left(\frac{\partial C_0}{\partial w_{jk}^{(L)}}\right)_{n_L \times n_{L-1}} = \left(a_k^{(L-1)}\sigma'\left(z_j^{(L)}\right)\delta_j^{(L)}\right)_{n_L \times n_{L-1}} = \sigma'\left(z^{(L)}\right)\delta^{(L)}\left(a^{(L-1)}\right)^T$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \left(\frac{\partial C_0}{\partial b_j^{(L)}}\right)_{n_L \times 1} = \sigma'\left(z^{(L)}\right)\delta^{(L)}$$

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \left(\frac{\partial C_0}{\partial a_k^{(L-1)}}\right)_{n_{L-1} \times 1} = \left(\sum_{j=1}^{n_L} w_{jk}^{(L)}\sigma'\left(z_j^{(L)}\right)\delta_j^{(L)}\right)_{n_{L-1} \times 1} = \left(w^{(L)}\right)^T \sigma'\left(z^{(L)}\right)\delta^{(L)}$$

where $\sigma'(z^{(L)})$ is diagonal matrix

$$\sigma'(z^{(l)}) = \begin{bmatrix} \sigma'\left(z_1^{(l)}\right) & 0 & \cdots & 0 \\ 0 & \sigma'\left(z_2^{(l)}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'\left(z_{n_l}^{(l)}\right) \end{bmatrix}$$

Now we have formulas for backward propagation, and since $\tilde{a}^{(L)} = Y$ and $a^{(L)} = \hat{Y}$, we can simply generalize is for l layer

$$\delta^{(l)} = a^{(l)} - \tilde{a}^{(l)} \tag{1.4}$$

$$\frac{\partial C_{L-l}}{\partial w^{(l)}} = \sigma'\left(z^{(l)}\right)\delta^{(l)}\left(a^{(l-1)}\right)^T \tag{1.5}$$

$$\frac{\partial C_{L-l}}{\partial b^{(l)}} = \sigma'\left(z^{(l)}\right)\delta^{(l)} \tag{1.6}$$

$$\frac{\partial C_{L-l}}{\partial a^{(l-1)}} = \left(w^{(l)}\right)^T \sigma'\left(z^{(l)}\right)\delta^{(l)} \tag{1.7}$$

From here we can choose algorithms for finding local minimum. Let's take small $0 < \gamma \leq 1$ and write

$$\tilde{w}^{(l)} = w^{(l)} - \gamma\frac{\partial C_{L-l}}{\partial w^{(l)}}$$

$$\tilde{b}^{(l)} = b^{(l)} - \gamma\frac{\partial C_{L-l}}{\partial b^{(l)}}$$

$$\tilde{a}^{(l-1)} = a^{(l-1)} - \gamma\frac{\partial C_{L-l}}{\partial a^{(l-1)}}$$

So, we have $a^{(0)} = X$, $\delta^{(L)} = a^{(L)} - Y$ and

$$\delta b^{(l)} = \gamma\sigma'\left(z^{(l)}\right)\delta^{(l)}$$

$$\delta^{(l-1)} = \left(w^{(l)}\right)^T \delta b^{(l)}$$

$$\tilde{b}^{(l)} = b^{(l)} - \delta b^{(l)}$$

$$\tilde{w}^{(l)} = w^{(l)} - \delta b^{(l)}\left(a^{(l-1)}\right)^T$$