Implementation of a simple neural network

Neural networks are a set of algorithms designed to recognize patterns. I decided to implement my own[2] in order to understand how they work. The result is a configurable, multidimensional simple neural network. I followed the 3Blue1Brown video [1] on backpropagation. In this document I will try to summarize the math parts.

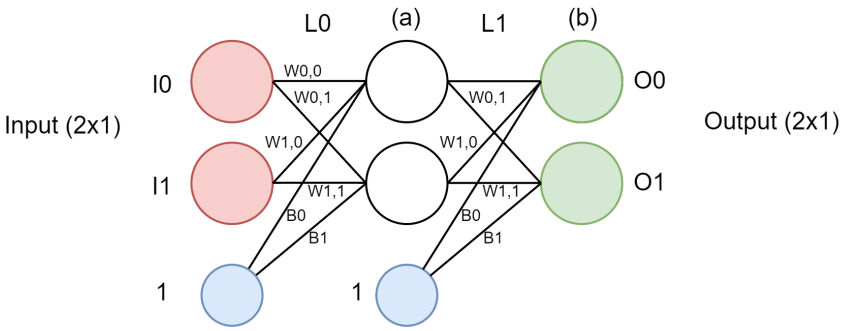

Figure 1: A sample architecture (1 hidden, 1 output)

Matrix representation

It takes any input shape, but it is flattened using the numpy command ravel() to a single array. The following structures represent the parameters of a single layer:

$$X_{1,n} = \begin{bmatrix} x_0 & \cdots & x_n \end{bmatrix} \qquad \rightarrow \qquad X_{1,n+1} = \begin{bmatrix} x_0 & \cdots & x_n & 1 \end{bmatrix} \tag{1}$$

$$W_{n,m} = \begin{bmatrix} w_{0,0} & \cdots & w_{0,m} \\ \vdots & \ddots & \vdots \\ w_{n,0} & \cdots & w_{n,m} \end{bmatrix} \tag{2}$$

$$Y_{1,m} = \begin{bmatrix} y_0 & \cdots & y_m \end{bmatrix} \tag{3}$$

Forward propagation

This step is pretty straightforward. First, append a one to the input data. This serves as the input for the bias. Next, take the extended input and dot multiply by its weights. That output is fed to the activation function.

$$X_{m+1,n} = InputData_{m,n} + 1_{1,1} \tag{4}$$

$$V_{m,n} = X_{m+1,1} \cdot W_{n,m+1} \tag{5}$$

$$Y_{m,n} = act(V_{m,n}) \tag{6}$$

Backwards propagation

This segment requires to know the chain rule. This step starts from the last layer and keeps running until getting to the beggining.

This is the error calculation in the last layer:

$$E = error(Y) = error(act(V)) = error(act(X \cdot W)) \tag{7}$$

To update the weights we need the derivative of the error respect to the weights.

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial V} * \frac{\partial V}{\partial W} \tag{8}$$

$$\frac{\partial E}{\partial Y}_{m,n} = error'(Y_{m,n}) \qquad \frac{\partial Y}{\partial V}_{m,n} = act'(V_{m,n}) \qquad \frac{\partial V}{\partial X}_{m,n} = V \qquad \frac{\partial V}{\partial W}_{m,n} = X \tag{9}$$

Now we can apply stochastic gradient descent for this layer:

$$W = W - \frac{\partial E}{\partial W} * LR \tag{10}$$

Realize that $\frac{\partial E}{\partial X}$ from L is $\frac{\partial E}{\partial Y}$ for L_{-1}. Only in the last (first) layer $\frac{\partial E}{\partial Y}$ is the derivation of the error.

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} * \frac{\partial Y}{\partial V} * \frac{\partial V}{\partial X} \tag{11}$$

References

[1] 3Blue1Brown. *Backpropagation calculus — Deep learning, chapter 4*. URL: `https://www.youtube.com/watch?v=tIeHLnjs5U8`.

[2] Jaume Colom. *Simple net implementation*. URL: `https://github.com/jaumecolomhernandez/simple-net`.