Notes on a 2-Layer Feed Forward Neural Network for Regression Tasks

Jonathan Navarrete

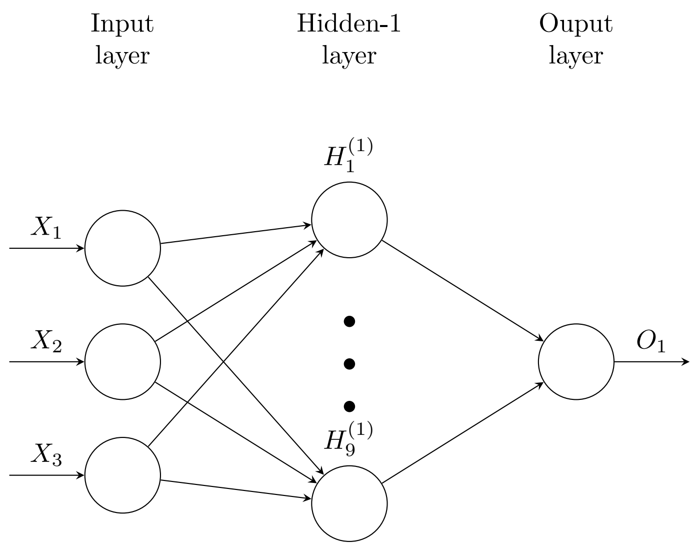
2019
January

1 Introduction

The following notes relate to a simple feed forward neural network trained for regression tasks. The neural network has an input layer (the inputs) and two hidden layers. The first hidden layer recieves the input data and transforms it into a nonlinear space. The data is feed forward into the second hidden layer for the output. The neural network is trained using vanilla stochastic gradient descent (SGD).

The neural network is implemented in a Python script (nnet.py).

2 Architecture



2.1 Data

The neural network takes 3 inputs, where the design matrix $\mathbf{X}$ is $N \times 3$; however, this can be generalized to any number of M inputs for a $N \times M$ design matrix. The neural network is then fed one observation at a time, $\mathbf{x}_i^\intercal$ for $i = 1, 2, ..., N$

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ & \cdots & \\ x_{N1} & x_{N2} & x_{N3} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^\intercal \\ \mathbf{x}_2^\intercal \\ \cdots \\ \mathbf{x}_N^\intercal \end{bmatrix}$$

There is one output target

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}$$

2.2 First Hidden Layer

In the first hidden layer there are 9 hidden units $H_1^{(1)}, H_2^{(1)}, ..., H_9^{(1)}$. Each hidden unit sees each of the three 3 inputs, aggregates the inputs with weights and passes that hidden output to an activation function. The activation function used in the first hidden layer is logistic,

$$H_k^{(1)} = f(z) = \frac{1}{1+e^{-z}}$$

where $z = w_1^{(1)} x_i 1 + w_2^{(1)} x_i 2 + w_3^{(1)} x_i 3$.

For each hidden unit, we have three weights $\mathbf{w}^{(1)}$

2.3 Second Hidden Layer

In the second hidden layer there is 1 hidden units (the output unit). Because there is only one output, y_i, there is only the need for one hidden unit.

The second hidden layer expects 9 inputs from the first hidden layer. The activation function is simply the identity function $g(z) = z$, where $z = w_1^{(2)} x_i 1 + w_2^{(2)} x_i 2 + w_9^{(2)} x_i 3$.

3 Feed Forward

3.1 Input to First Hidden Layer

For each observation i, we take input $\mathbf{x}_i^\intercal$ and pass it to the first layer to each hidden unit j

$$h_j = f(z) = \frac{1}{1+e^{-z_j}}$$

where

$$z_j = \mathbf{x}_i^\intercal \mathbf{w}_j^{(1)} = \begin{bmatrix} x_{i1} & x_{i2} & x_{i3} \end{bmatrix} \begin{bmatrix} w_1^{(1)} \\ w_2^{(1)} \\ w_3^{(1)} \end{bmatrix} = w_1^{(1)} x_1 + w_2^{(1)} x_2 + w_3^{(1)} x_3$$

This can be more efficiently computed by using a 3×9 weights matrix

$$\mathbf{W}^{(1)} = \begin{bmatrix} \mathbf{w}_1^{(1)} & \mathbf{w}_2^{(1)} & \cdots & \mathbf{w}_9^{(1)} \end{bmatrix}$$

Take the input row vector $\mathbf{x}_i^\intercal$ and multiply it with $W^{(1)}$ to obtain $\mathbf{z}^\intercal$

$$\mathbf{z}^\intercal = \mathbf{x}_i^\intercal \mathbf{W}^{(1)} = \begin{bmatrix} z_1 & z_2 & \cdots & z_9 \end{bmatrix}$$

Afterwards, we pass the hidden layer outputs through the activation function,

$$\mathbf{h}^\intercal = f(\mathbf{z}^\intercal) = \begin{bmatrix} h_1 & h_2 & \cdots & h_9 \end{bmatrix}$$

3.2 Hidden to Output Layer

After the first hidden layer is computed, the hidden outputs $\mathbf{h}^\intercal$ is fed to the output layer. The ouptut layer has 1 output and expects 9 inputs. Thus, we have a 9×1 weights matrix

$$\mathbf{W}^{(2)} = \mathbf{w}^{(2)} = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_9 \end{bmatrix}$$

$$o_i = g(\mathbf{h}^\intercal \mathbf{w}^{(2)}) = \mathbf{h}^\intercal \mathbf{w}^{(2)}$$

4 Backpropogation

> The back-propagation algorithm (Rumelhart et al., 1986a), often simply called backprop, allows the information from the cost to then ow backward throughthe network in order to compute the gradient. [...] The term back-propagation is often misunderstood as meaning the whole learning algorithm for multilayer neural networks. Actually, back-propagation refers only to the method for computing the gradient, while another algorithm,such as stochastic gradient descent, is used to perform learning using this gradient. Furthermore, back-propagation is often misunderstood as being specic to multi-layer neural networks, but in principle it can compute derivatives of any function (for some functions, the correct response is to report that the derivative of the function is undened).[1]

4.1 Calculation Errors

For this regression task we will use the mean squared error (MSE) as the statistic to measure performance. We'd like to create a neural network that has a small as possible MSE. Thus we are trying to minimize MSE with respect to the weights in the neural network.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (o_i - y_i)^2 = \frac{1}{N} \sum_{i=1}^{N} e_i^2$$

[1]https://www.deeplearningbook.org/contents/mlp.html

To train the neural network, it'll be necessary to differentiate this error response with respect to each weight. This will be accomplished using the backpropogation algorithm.

4.2 Gradient for Output Layer

Using the backprop algorithm we will take the derivative of MSE with respect to (wrt) the weights in the output layer. Let $E = MSE(w^{(2)})$, then by the chain rule we obtain

$$\frac{dE}{dw_k^{(2)}} = \frac{dE}{do_i}\frac{do_i}{dw_k^{(2)}}$$

The derivative of E wrt o_i is

$$\frac{dE}{do_i} = \frac{2}{N}\sum_i^N (y_i - o_i)$$

and the derivative of o_i wrt $w_k^{(2)}$ is

$$\frac{do_i}{dw_k^{(2)}} = \frac{d}{dw_k^{(2)}}(dw_1^{(2)}h_1^{(1)} + \cdots + w_k^{(2)}h_k^{(1)} + \cdots + +w_9^{(2)}h_9^{(1)}) = h_k^{(1)}$$

Thus, the partial derivate is

$$\frac{2}{N}\sum_i^N (y_i - o_i) \times h_k^{(1)}$$

However, this process will need to be continued for all weights $w_1^{(2)}, \cdots, w_9^{(2)}$. Taking partial derivatives for all weights gives us the gradient.

$$\frac{dE}{dw_k^{(2)}} = \frac{dE}{do_i}\nabla o_i$$

where

$$\nabla o_i = \begin{bmatrix} \frac{do_i}{dw_1^{(2)}} \\ \cdots \\ \frac{do_i}{dw_9^{(2)}} \end{bmatrix} = \begin{bmatrix} h_1^{(1)} \\ \cdots \\ h_9^{(1)} \end{bmatrix}$$

4.3 Gradient for Hidden Layer

Just as was done for the second hidden layer, the output layer, we have to take the derivative wrt the first hidden layer's weights $w^{(1)}$.

$$\frac{dE}{dw_k^{(1)}} = \frac{dE}{do_i}\frac{do_i}{dh_k^{(1)}}\frac{dh_k}{dw_j^{(1)}}$$

Taking the derivative of E wrt o_i remains the same. However, the second partial derivative is a bit different

$$\frac{do_i}{dh_k^{(1)}} = \frac{d}{dh_k^{(1)}}(dw_1^{(2)}h_1^{(1)} + \cdots + w_k^{(2)}h_k^{(1)} + \cdots + +w_9^{(2)}h_9^{(1)}) = w_k^{(2)}$$

Repeating this process of taking partial derivatives wrt each hidden layer input we obtain the gradient

$$\nabla o_i = \begin{bmatrix} \frac{do_i}{dh_1^{(1)}} \\ \cdots \\ \frac{do_i}{dh_9^{(1)}} \end{bmatrix} = \begin{bmatrix} w_1^{(2)} \\ \cdots \\ w_9^{(2)} \end{bmatrix}$$

The last remaining step is to calculate the derivative of h_k wrt $w_j^{(1)}$

$$\frac{dh_k}{dw_j^{(1)}} = \frac{dh_k}{dz_k} \times \frac{dz_k}{dw_j^{(1)}}$$

Since $h_k = f(z_k) = \textbf{logistic}(z_k)$, uses the logistic activation function, we'll need to take the derivative of the logistic function.

$$f'(z_k) = f(z_k) \times (1 - f(z_k))$$

with the gradient being

$$\nabla \mathbf{h} = f(\mathbf{z})(1 - f(\mathbf{z}))$$

and

$$\frac{dz_k}{dw_j^{(1)}} = \frac{d}{dw_j^{(1)}}(w_k^{(1)}x_k) = x_k$$

with the gradient being

$$\nabla \mathbf{z_k^\intercal} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

and for k = 1, ..., 9

$$\nabla \mathbf{z} = \begin{bmatrix} \mathbf{z_1}^\intercal \\ \cdots \\ \mathbf{z_9}^\intercal \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ & \cdots & \\ x_1 & x_2 & x_3 \end{bmatrix}$$

Finally, the gradient of E becomes

$$\nabla E \ = \frac{dE}{do_i} \times \nabla o_i \times \nabla \mathbf{h} \nabla \mathbf{z}^\intercal = \frac{2}{N} \sum_i^N \left((y_i - o_i) \times \begin{bmatrix} w_1^{(2)} \\ \cdots \\ w_9^{(2)} \end{bmatrix} \times f(\mathbf{z})(1 - f(\mathbf{z})) \right) \begin{bmatrix} x_1 & x_2 & x_3 \\ & \cdots & \\ x_1 & x_2 & x_3 \end{bmatrix}$$