Modifying Activations in a Multilayer Perceptron

Landon Buell

29 March 2020

1 Abstract

2 Introduction

3 Multilayer Perceptron Behavior

A multilayer perceptron is one of the simplest and oldest neural network architectures [1]. It was developed in 1957 by Frank Rosenblatt and is loosely based off of biological neuron behavior [4]. It is based off of creating layers of artificial neurons, connected through repeated linear transformations. In practice, a layer of neurons is modeled by a column vector. Each entry in that vector corresponds to the numerical value contained within that particular neuron. The value in this entry is called the *activation* of that neuron.

3.1 Forward Propagration

Layers of neurons are connected through transformation operators in the form of standard matrices. This means that each activation is a formed from a linear combination of the activations in the previous layer of the network. Often, a *bias function* is added to each neuron as well. Finally, each layer is then subject to an *activation function* which modifies the entries in each neuron. In general, we can model the i-th activation in the l-th layer of a network by the function:

$$x_i^{(l)} = f\Big[\sum_j W_{ij}^{(l-1)} x_j^{(l-1)} + \beta^{(l-1)}\Big] \tag{1}$$

Where:

- $W_{ij}^{(l-1)}$ is the element in the i-th row and j-th column of the weighting $\hat{W}^{(l-1)}$. The number of rows in this matrix is given by the number of neurons in the $(l-1)$-th layer, and the number of columns in this matrix is given by the number of neurons in the (l)-th layer.

- $x_j^{(l-1)}$ is the j-th element of the vector $\vec{x}^{(l-1)}$. This is also the activation value for the j-th neuron in the $(l-1)$-th layer of the network model.
- $\beta^{(l-1)}$ is the *bias function* for the $(l-1)$-th layer of the network
- $f[]$ is the *activation function* for each layer of the network model.

Written more explicitly: Let layer $\vec{x}^{(l)}$ be represented by the vector $\vec{y}$ and contain a neurons, and let $\vec{x}^{(l)}$ be represented by the vector $\vec{x}$ and contain b neurons. Additionally, et the linear transformation $\hat{W}^{(l-1)}$ be represented by the matrix $\hat{W}$ and the bias function $\vec{\beta}^{(l-1)}$ be represented by the vector $\vec{\beta}$.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{a-2} \\ y_{a-1} \end{bmatrix} = f\left(\begin{bmatrix} W_{0,0} & W_{0,1} & \dots & W_{0,(b-2)} & W_{0,(b-1)} \\ W_{1,0} & W_{1,1} & \dots & W_{1,(b-2)} & W_{1,(b-1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W_{(a-2),0} & W_{(a-2),1} & \dots & W_{(a-2),(b-2)} & W_{(a-2),(b-1)} \\ W_{(a-1),0} & W_{a-1),1} & \dots & W_{a-1),(b-2)} & W_{a-1),(b-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{b-2} \\ x_{b-1} \end{bmatrix} + \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{a-2} \\ \beta_{a-1} \end{bmatrix} \right) \quad (2)$$

In a Multilayer Perceptron Neural Network with L layers, this operation is repeated $L-1$ times. The activation in the final layer of the network - $\vec{x}^{(L-1)}$ - represents the decision of the network based on the activation of the input layer - $\vec{x}^{(0)}$. The act of taking a vector of input features and passing it through the layers of the network via transformation, biases, and activation functions to attain an output decision is called *forward propagation* [2].

3.2 Backward Propagation

In order to *train* a neural network, the algorithm must have a objective to attain. This objective is to minimize the value of a *loss function* [3] also called an object function or cost function. Each time a sample of data is forward-propagated through the model, the network's final output activations correspond to a decision made. This prediction is then compared to the known target for that particular sample. The difference between this predicted value and expected value is used to compute the value for the loss function for that sample. The larger the output of the function, the more the model deviated from the true value and the smaller the value, the closer the model is to attaining the correct value [2].

The value of this decision function is used to repeatedly work backwards and adjust the entries in the weighing matrices. This is generally done through a procedure called *stochastic gradient descent* [1]. This algorithm uses element-wise differences to compute discrete gradients as a function of all of the entries in the weighting vectors. Each training sample is then used to systematically modify values in the linear combinations produced to reduce the value of the loss function.

4 Methodology

In this experiment, we have used the open-source Python module Scikit-Learn to create a Multilayer Perceptron Classifier object with the following Hyper-parameters set in manually.

5 Results

6 Conclusions

References

[1] Géron Aurélien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2017.

[2] Goodfellow, Ian, et al.*Deep Learning*. MIT Press, 2017.

[3] James, Gareth, et al. An Introduction to Statistical Learning with Applications in R. Springer, 2017.

[4] McCulloch, Warren S., and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, 1943, pp. 115–133.