Jonathan Lee
Dr. Nelson
Neural Networks
2 February 2015

Multi-layer Perceptron

The multi-layer perceptron originated as a solution to limitations of the original single-layer perceptron. The single-layer perceptron model is a simple neural network consisting of only one neuron, or processing unit, that receives input and then produces a classifying output to solve specific tasks. These inputs are analogous to sensory stimuli from a particular environment. However, before reaching the neuron, the inputs are multiplied by weights which are values that can be modified to characterize the network, allowing it to produce unique solutions even with the same arrangement. Thus the value that the single neuron actually receives is a weighted sum of the inputs. Information in the model moves only in one direction across the network from inputs to the neuron to the output and thus the model is considered a feed-forward network. The weights and feed-forward nature make up the Rosenblatt perceptron's rule of propagation of information.

The neuron determines whether it is activated or deactivated by passing the weighted sum through an activation function which is typically a step function or a sigmoid function. The weighted sum is given by

$$u = \sum_{i=1}^{N} w_i a_i \,, \qquad \text{Equation 1}$$

where N is the number of inputs and w_i and a_i are the i^{th} weight and input. The output of the activation function determines the state of the node's activation. Since single-layer perceptrons only have one node or neuron, the output of the network is simply the output of the activation function.

As an example, a single-layer perceptron could have two inputs and an activation function that is a step function where any weighted sum less than 0 outputs -1 and is deactivated and any weighted sum greater than or equal to 0 outputs 1 and is activated as shown in *Figure 1*.

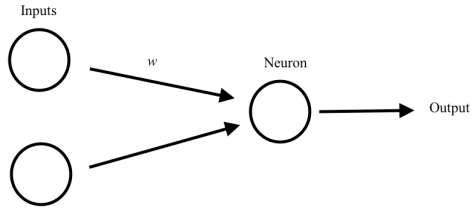

Figure 1: An example of a single-layer perceptron.

If this perceptron is to solve the AND task in which inputs are either 1 or 0, then the weighted sum should be greater than or equal to 0 only when both inputs are 1. To solve the OR task, the weighted sum should be greater than zero when at least one input is 1. Many different sets of weights can solve either task. It should be noted that activated and deactivated states can be separated by a line called a decision line when the weighted sum equals zero as shown by

$$w_1 a_1 + w_2 a_2 = 0 .$$ Equation 2

The line is called a decision line because points that fall on one side of it are deemed by the perceptron to be of a particular state while points on the other side are deemed to be of a different state. *Figure 2* visualizes this concept.

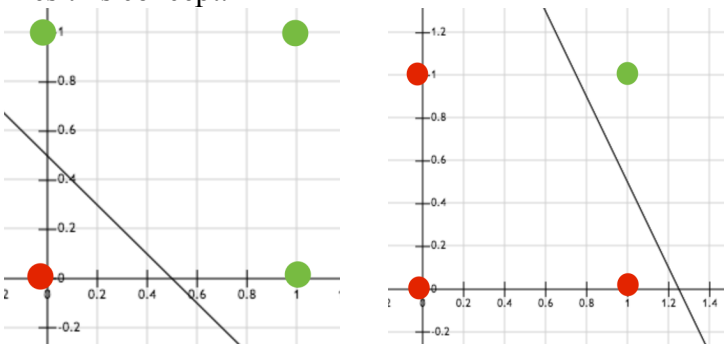

Figure 2: The left graph shows the division of classified outputs of the AND function while the right graph shows the division of classified outputs of the OR function. The x-axis is the value of the first input and the y-axis is the value

Because a line can divide the output classification, these tasks are considered linearly separable problems. However when confronted with a XOR task in which the weighted sum should be greater than or equal to zero when there is one and only one input of value 1, the limitations of single-layer perceptrons become apparent. Single-layer perceptrons are not able to solve more complex logical tasks such as XOR because the classifications of outputs are not linearly separable as shown in *Figure 3*.

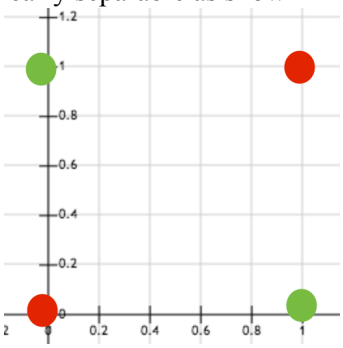

Figure 3: A line cannot be used to separate the outputs into their respective states.

This limitation is the motivation for developing multi-layer perceptrons. Multi-layer perceptrons are capable solving tasks such as XOR that require non-linear separation of outputs. The key concept behind the multi-layer perceptron, shown in *Figure 4* is to add more neurons in the form of hidden layers. Thus inputs will not go directly to the same node that determines the output of the entire network. In fact, nodes of a particular layer act as the input nodes of the next layer. These multi-layer perceptrons are still feed-forward networks and are not cyclic, so a particular layer will not pass information to a previous layer. In some networks, connections between layers may skip over other layers.

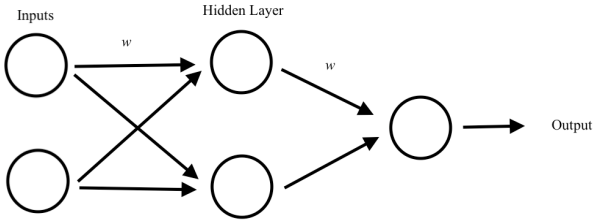

Figure 4: An example of a multi-layer perceptron.

Intermediate neurons will behave the same way as the single neuron of the single-layer perceptron. They take weighted sums as inputs, have activation functions and have defined states of activation. However, the addition of these neurons adds more complexity to the network by allowing various connections to be made between different nodes.

It should be noted that multi-layer perceptrons with linear activation functions can simply be replaced by single-layer perceptrons, according to a presentation by Ricardo Gutierrez-Osuna of Texas A&M University's Computer Science department. This limitation is the reason that sigmoid and other non-linear functions are necessary for multi-layer perceptrons.

The major issue that multi-layer perceptrons pose is that finding the set of weights that would cause the network to return desired outputs based on inputs would require significantly more computation because of the added nodes and connections. The concept of back-propagation gradient descent is a simple but not perfect solution to this limitation. This method looks for the minimum error between the desired outputs and the output that the network actually returns. The method finds the "direction" of weight modifications that would cause the network with an updated set of weights to have lower error. Thus the network learns by modifying the weights through enough iterations to ideally match the desired output. This "direction" is found by gradient descent where the partial derivatives of error with respect to each weight are calculated. Since the goal is to minimize the error and the gradient of the function gives the direction of steepest ascent, only the negative of each partial derivative is relevant and each partial derivative is thus considered the Δw for the respective weight. The Δw for a specific weight is given by

$$\Delta w = -\lambda \frac{\partial E}{\partial w} \qquad \text{Equation 3}$$

where λ is some constant used to scale the partial derivative. The Δw is then added to the respective weight to minimize the error between the desired output and the actual output. The updated weight is given by

$$w' = w + \Delta w \,. \qquad \text{Equation 4}$$

With enough iterations, back-propagation should theoretically train the network to achieve the desired values by minimizing the error.

Although back-propagation is a solution to the aforementioned limitation of multi-layer perceptrons, it has its own limitations. Firstly, when looking for a minimum of the error function of the network, back-propagation may lead the weights to reach a local minimum of the error function that is a suboptimal minimum. At this point, the gradient would become zero and thus descent would no longer produce nonzero Δw values. Additionally, if the error function has a steep curve, the Δw values may become relatively large causing the updated weight to overshoot the minimum. Different modifications could be made to amend these limitations such as decreasing λ to decrease the overall training rate of the weights but this might also increase iterations. For this reason, training networks is as much an art as it is a science.

Works Cited

"The Backpropagation Algorithm." Free University of Berlin. N.p., n.d. Web. 1 Feb. 2015. <http://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf>.

Beeman, Dave. "Multi-layer Perceptrons (Feed-forward Nets), Gradient Descent, and Back Propagation." University of Colorado. N.p., n.d. Web. 1 Feb. 2015. <http://ecee.colorado.edu/~ecen4831/lectures/NNet3.html>.

"Feed-forward Neural Networks." University of Nottingham. N.p., n.d. Web. 1 Feb. 2015. <http://www.cs.nott.ac.uk/~qiu/Teaching/G53MLE/ffnets-note.pdf>.

Gutierrez-Osuna, Ricardo. "L18: Multi-layer Perceptrons." Texas A&M University. N.p., n.d. Web. 1 Feb. 2015. <http://research.cs.tamu.edu/prism/lectures/pr/pr_l18.pdf>.

Haykin, Simon O. Neural Networks and Learning Machines. 3rd ed. Harlow: Prentice Hall, 2007. Print.

Park, Hyeyoung. "Multilayer Perceptron and Natural Gradient Learning." 2005. PDF file.

Riedmiller, Martin. "Machine Learning: Multi Layer Perceptrons." University of Freibrug. N.p., n.d. Web. 1 Feb. 2015. <http://ml.informatik.uni-freiburg.de/_media/teaching/ss10/05_mlps.printer.pdf>.