Multilayer perceptron

From Wikipedia, the free encyclopedia

A **multilayer perceptron** (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network.[1][2] MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.[3]

Contents

Theory

Activation function

If a multilayer perceptron has a linear activation function in all neurons, that is, a linear function that maps the weighted inputs to the output of each neuron, then it is easily proved with linear algebra that any number of layers can be reduced to the standard two-layer input-output model (see perceptron). What makes a multilayer perceptron different is that some neurons use a *nonlinear* activation function which was developed to model the frequency of action potentials, or firing, of biological neurons in the brain. This function is modeled in several ways.

The two main activation functions used in current applications are both sigmoids, and are described by

$$y(v_i) = \tanh(v_i) \quad \text{and} \quad y(v_i) = (1 + e^{-v_i})^{-1},$$

in which the former function is a hyperbolic tangent which ranges from -1 to 1, and the latter, the logistic function, is similar in shape but ranges from 0 to 1. Here y_i is the output of the ith node (neuron) and v_i is the weighted sum of the input synapses. Alternative activation functions have been proposed, including the rectifier and softplus functions. More specialized activation functions include radial basis functions which are used in another class of supervised neural network models.

Layers

The multilayer perceptron consists of three or more layers (an input and an output layer with one or more *hidden layers*) of nonlinearly-activating nodes and is thus considered a deep neural network. Each node in one layer connects with a certain weight w_{ij} to every node in the following layer. Some people

do not include the input layer when counting the number of layers and there is disagreement about whether w_{ij} should be interpreted as the weight from i to j or the other way around.

Learning through backpropagation

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

We represent the error in output node j in the nth data point (training example) by $e_j(n) = d_j(n) - y_j(n)$, where d is the target value and y is the value produced by the perceptron. We then make corrections to the weights of the nodes based on those corrections which minimize the error in the entire output, given by

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

Using gradient descent, we find our change in each weight to be

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the *learning rate*, which is carefully selected to ensure that the weights converge to a response fast enough, without producing oscillations. In programming applications, this parameter typically ranges from 0.2 to 0.8.

The derivative to be calculated depends on the induced local field v_j, which itself varies. It is easy to prove that for an output node this derivative can be simplified to

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

where ϕ' is the derivative of the activation function described above, which itself does not vary. The analysis is more difficult for the change in weights to a hidden node, but it can be shown that the relevant derivative is

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n))\sum_k -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n).$$

This depends on the change in weights of the kth nodes, which represent the output layer. So to change the hidden layer weights, we must first change the output layer weights according to the derivative of the activation function, and so this algorithm represents a *backpropagation of the activation function*. [4]

Terminology

The term "multilayer perceptron" often causes confusion. It is argued the model is not a single perceptron that has multiple layers. Rather, it contains many perceptrons that are organised into layers, leading some to believe that a more fitting term might therefore be "multilayer perceptron network". Moreover, these "perceptrons" are not really perceptrons in the strictest possible sense, as true perceptrons are a special case of artificial neurons that use a threshold activation function such as the Heaviside step function, whereas the artificial neurons in a multilayer perceptron are free to take on any arbitrary activation function. Consequently, whereas a true perceptron performs binary classification, a neuron in a multilayer perceptron is free to either perform classification or regression, depending upon its activation function.

The two arguments raised above can be reconciled with the name "multilayer perceptron" if "perceptron" is simply interpreted to mean a binary classifier, independent of the specific mechanistic implementation of a classical perceptron. In this case, the entire network can indeed be considered to be a binary classifier with multiple layers. Furthermore, the term "multilayer perceptron" now does not specify the nature of the layers; the layers are free to be composed of general artificial neurons, and not perceptrons specifically. This interpretation of the term "multilayer perceptron" avoids the loosening of the definition of "perceptron" to mean an artificial neuron in general.

Applications

Multilayer perceptrons using a backpropagation algorithm are the standard algorithm for any supervised learning pattern recognition process and the subject of ongoing research in computational neuroscience and parallel distributed processing. They are useful in research in terms of their ability to solve problems stochastically, which often allows one to get approximate solutions for extremely complex problems like fitness approximation.

MLPs were a popular machine learning solution in the 1980s, finding applications in diverse fields such as speech recognition, image recognition, and machine translation software,[5] but have since the 1990s faced strong competition from the much simpler (and related[6]) support vector machines. More recently, there has been some renewed interest in backpropagation networks due to the successes of deep learning.

References

1. Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961
2. Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. “Learning Internal Representations by Error Propagation”. David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations. MIT Press, 1986.
3. Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function *Mathematics of Control, Signals, and Systems*, 2(4), 303–314.
4. Haykin, Simon (1998). *Neural Networks: A Comprehensive Foundation* (2 ed.). Prentice Hall. ISBN 0-13-273350-1.
5. Neural networks. II. What are they and why is everybody so interested in them now?; Wasserman, P.D.; Schwartz, T.; Page(s): 10-15; IEEE Expert, 1988, Volume 3, Issue 1
6. R. Collobert and S. Bengio (2004). Links between Perceptrons, MLPs and SVMs. Proc. Int'l Conf. on Machine Learning (ICML).

External links

- A Gentle Introduction to Backpropagation - An intuitive tutorial by Shashi Sathyanarayana

(http://numericinsight.com/uploads/A_Gentle_Introduction_to_Backpropagation.pdf) This is an updated PDF version of a blog article that was previously linked here. This article contains pseudocode ("Training Wheels for Training Neural Networks") for implementing the algorithm.
- Weka: Open source data mining software with multilayer perceptron implementation (http://www.cs.waikato.ac.nz/ml/weka/).

- Border pairs method description (http://www.springerlink.com/index/200651K318731758.pdf).

Categories: Classification algorithms | Artificial neural networks

---