Artificial Neural Networks for Prediction

Rafael Martí
Universitat de València, Spain

INTRODUCTION

The design and implementation of intelligent systems with human capabilities is the starting point to design Artificial Neural Networks (ANNs). The original idea takes after neuroscience theory on how neurons in the human brain cooperate to learn from a set of input signals to produce an answer. Because the power of the brain comes from the number of neurons and the multiple connections between them, the basic idea is that connecting a large number of simple elements in a specific way can form an intelligent system.

Generally speaking, an ANN is a network of many simple processors called *units,* linked to certain neighbors with varying coefficients of connectivity (called *weights*) that represent the strength of these connections. The basic unit of ANNs, called an *artificial neuron,* simulates the basic functions of natural neurons: it receives inputs, processes them by simple combination and threshold operations, and outputs a final result.

ANNs often employ supervised learning in which training data (including both the input and the desired output) is provided. Learning basically refers to the process of adjusting the weights to optimize the network performance. ANNs belongs to machine-learning algorithms because the changing of a network's connection weights causes it to gain knowledge in order to solve the problem at hand.

Neural networks have been widely used for both classification and prediction. In this article, I focus on the prediction or estimation problem (although with some few changes, my comments and descriptions also apply to classification). Estimating and forecasting future conditions are involved in different business activities. Some examples include cost estimation, prediction of product demand, and financial planning. Moreover, the field of prediction also covers other activities, such as medical diagnosis or industrial process modeling.

In this short article I focus on the multilayer neural networks because they are the most common. I describe their architecture and some of the most popular training methods. Then I finish with some associated conclusions and the appropriate list of references to provide some pointers for further study.

BACKGROUND

From a technical point of view, ANNs offer a general framework for representing nonlinear mappings from several input variables to several output variables. They are built by tuning a set of parameters known as *weights* and can be considered as an extension of the many conventional mapping techniques. In classification or recognition problems, the net's outputs are categories, while in prediction or approximation problems, they are continuous variables. Although this article focuses on the prediction problem, most of the key issues in the net functionality are common to both.

In the process of training the net *(supervised learning),* the problem is to find the values of the weights *w* that minimize the error across a set of input/output pairs (patterns) called the training set *E*. For a single output and input vector x, the error measure is typically the root mean squared difference between the predicted output *p(x,w)* and the actual output value *f(x)* for all the elements *x* in *E* (RMSE); therefore, the training is an unconstrained nonlinear optimization problem, where the decision variables are the weights, and the objective is to reduce the training error. Ideally, the set *E* is a representative sample of points in the domain of the function *f* that you are approximating; however, in practice it is usually a set of points for which you know the *f*-value.

$$\underset{w}{Min}\ \ error(E,w)=\sqrt{\frac{\sum_{x\in E}(f(x)-p(x,w))^2}{|E|}} \qquad (1)$$

The main goal in the design of an ANN is to obtain a model that makes good predictions for new inputs (i.e., to provide good generalization). Therefore, the net must represent the systematic aspects of the training data rather than their specific details. The standard way to measure the generalization provided by the net consists of introducing a second set of points in the domain of *f* called the testing set, *T*. Assume that no point in *T* belongs to *E* and *f(x)* is known for all *x* in *T.* After the optimization has been performed and the weights have been set to minimize the error in *E (w=w*),* the error across the testing set *T* is computed (*error(T,w*)*). The net must exhibit a good fit between the target *f*-values and the output (prediction) in the training set and also in the testing set. If the RMSE in *T* is significantly higher than that one in *E*, you say that the net has *memorized* the data instead of *learning* them (i.e., the net has overfitted the training data).

The optimization of the function given in (1) is a hard problem by itself. Moreover, keep in mind that the final objective is to obtain a set of weights that provides low values of *error(T,w*)* for any set *T*. In the following sections I summarize some of the most popular and other not so popular but more efficient methods to train the net (i.e., to compute appropriate weight values).

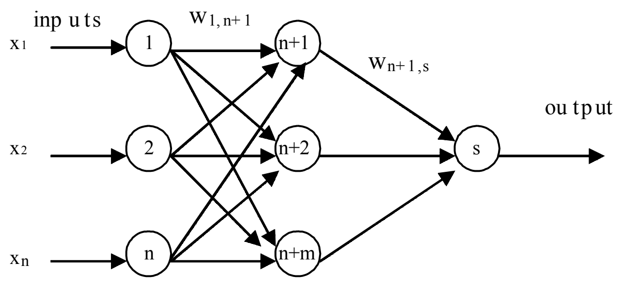
MAIN THRUST

Several models inspired by biological neural networks have been proposed throughout the years, beginning with the *perceptron* introduced by Rosenblatt (1962). He studied a simple architecture where the output of the net is a transformation of a linear combination of the input variables and the weights. Minskey and Papert (1969) showed that the perceptron can only solve linearly separable classification problems and is therefore of limited interest. A natural extension to overcome its limitations is given by the so-called multilayer-perceptron, or, simply, multilayer neural networks. I have considered this architecture with a single hidden layer. A schematic representation of the network appears in Figure 1.

Neural Network Architecture

Let *NN=(N, A)* be an ANN where *N* is the set of nodes and *A* is the set of arcs. *N* is partitioned into three subsets: N_I, input nodes, N_H, hidden nodes, and N_O, output nodes. I assume that *n* variables exist in the function that I want to predict or approximate, therefore $|N_I| = n$. The neural network has *m* hidden neurons ($|N_H| = m$) with a bias term in each hidden neuron and a single output neuron (we restrict our attention to real functions $f: \Re^n \to \Re$). Figure 1 shows a net where $N_I = \{1, 2, ..., n\}$, $N_H = \{n+1, n+2, ..., n+m\}$ and $N_O = \{s\}$.

Figure 1. Neural network diagram



Given an input pattern $x=(x_1,...,x_n)$, the neural network provides the user with an associated output *NN(x,w)*, which is a function of the weights *w*. Each node *i* in the input layer receives a signal of amount x_i that it sends through all its incident arcs to the nodes in the hidden layer. Each node *n+j* in the hidden layer receives a signal *input(n+j)* according to the expression

$$\text{Input}(n+j)=w_{n+j} + \sum_{i=1}^{n} x_i w_{i,n+j}$$

where w_{n+j} is the bias value for node *n+j,* and $w_{i,n+j}$ is the weight value on the arc from node *i* in the input layer to node *n+j* in the hidden layer. Each hidden node transforms its input by means of a nonlinear activation function: *output(j)=sig(input(j)).* The most popular choice for the activation function is the sigmoid function $sig(x)= 1/(1+e^{-x})$. Laguna and Martí (2002) test two activation functions for the hidden neurons and conclude that the sigmoid presents superior performance. Each hidden node *n+j* sends the amount of signal *output(n+j)* through the arc *(n+j,s)*. The node *s* in the output layer receives the weighted sum of the values coming from the hidden nodes. This sum, *NN(x,w),* is the net's output according to the expression:

$$NN(x,w) = w_s + \sum_{j=1}^{m} output(n+j)\, w_{n+j,s}$$

In the process of training the net (supervised learning), the problem is to find the values of the weights (including the bias factors) that minimize the error (RMSE) across the training set *E*. After the optimization has been performed and the weights have been set ($w=w^*$), the net is ready to produce the *output* for any *input* value. The testing error *Error(T,w*)* computes the Root Mean Squared Error across the elements in the testing set $T=\{y^1, y^2,..,y^s\}$, where no one belongs to the training set *E*:

$$Error(T,w^*) = \sqrt{\frac{\sum_{i=1}^{s} error(y^i, w^*)}{s}}.$$

Training Methods

Considering the supervised learning described in the previous section, many different training methods have been proposed. Here, I summarize some of the most relevant, starting with the well-known backpropagation