2.1 Introduction to Neural Networks

In the context of this package, a neural network is nothing more than a function with adjustable or tunable parameters. Let the input to a neural network be denoted by x, a real-valued (row) vector of arbitrary dimensionality or length. As such, x is typically referred to as *input*, *input vector*, *regressor* and sometimes, *pattern vector*. Typically, the length of vector x is said to be the *number of inputs* to the network. Let the network output be denoted by $\hat{y}$, an approximation of the desired output y, also a real-valued vector having one or more components, and the *number of outputs* from the network. Often data sets contain many input-output pairs. Then x and y denote matrices with one input and one output vector on each row.

Generally, a neural network is a structure involving weighted interconnections among *neurons*, or *units*, which are most often nonlinear scalar transformations, but which can also be linear. Figure 2.1 shows an example of a one-hidden-layer neural network with three inputs, $x = \{1, x_2, x_3\}$ that, along with a *unity bias* input, feed each of the two neurons comprising the *hidden layer*. The two outputs from this layer and a unity bias are then fed into the single output layer neuron, yielding the scalar output, $\hat{y}$. The layer of neurons is called hidden since its outputs are not directly seen in the data. This particular type of neural network is described in detail in Section 2.5, Feedforward and Radial Basis Function Networks. Here, this network will be used to explain common notation and nomenclature used in the package.

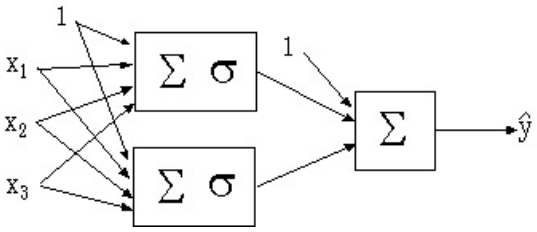

Figure 2.1. A feedforward neural network with three inputs, two hidden neurons, and one output neuron.

Each arrow in Figure 2.1 corresponds to a real-valued *parameter*, or a weight, of the network. The values of these parameters are tuned in the network training.

Generally, a neuron is structured to process multiple inputs, including the unity bias, in a nonlinear way, producing a single output. Specifically, all inputs to a neuron are first augmented by multiplicative weights. These weighted inputs are summed and then transformed via a nonlinear *activation function*, σ. As indicated in Figure 2.1, the neurons in the first layer of the network are nonlinear. The single output neuron is linear, since no activation function is used.

By inspection of Figure 2.1, the output of the network is given by

$$\hat{y} = w_1^2 + w_2^2 \sigma(w_1^1 + w_2^1 x_2 + w_3^1 x_3) + w_3^2 \sigma(w_4^1 + w_5^1 x_2 + w_6^1 x_3) \qquad (1)$$

involving the various parameters of the network, the weights $\{w_1^2, w_2^2, w_3^2, w_4^1, w_5^1, w_6^1\}$. The weights are sometimes referred to as *synaptic strengths*.

Eq. (2.0) is a nonlinear mapping, $x \to \hat{y}$, specifically representing the neural network in Figure 2.1. In general, this mapping is given in more compact form by

$$\hat{y} = g(\theta, x) \qquad (2)$$

where the θ is a real-valued vector whose components are the parameters of the network, namely, the weights.

When algorithmic aspects, independent of the exact structure of the neural network, are discussed, then this compact form becomes more convenient to use than an explicit one, such as that of Eq. (2.1).

This package supports several types of neural networks from which a user can choose. Upon assigning design parameters to a chosen network, thus specifying its structure $g(\cdot,\cdot)$, the user can begin to train it. The goal of training is to find values of the parameters θ so that, for any input x, the network output $\hat{y}$ is a good approximation of the desired output y. Training is carried out via suitable algorithms that tune the parameters θ so that input training data map well to corresponding desired outputs. These algorithms are iterative in nature, starting at some initial value for the parameter vector θ and incrementally updating it to improve the performance of the network.

Before the trained network is accepted, it should be validated. Roughly, this means running a number of tests to determine whether the network model meets certain requirements. Probably the simplest way, and often the best, is to test the neural network on a data set that was not used for training, but which was generated under similar conditions. Trained neural networks often fail this validation test, in which case the user will have to choose a better model. Sometimes, however, it might be enough to just repeat the training, starting from different initial parameters θ. Once the neural network is validated, it is ready to be used on new data.

The general purpose of the *Neural Networks* package can be described to be function approximation. However, depending on the origin of the data, and the intended use of the obtained neural network model, the function approximation problem may be subdivided into several types of problems. Different types of function approximation problems are described in Section 2.1.1. Section 1.1, Features of This Package includes a table giving an overview of the supported neural networks and the particular types of problems they are intended to address.

◂ Previous | Next ▸