A17 – Neural Networks

Introduction

By interconnecting many artificial neurons (Figure 1) to other neurons we form a neural network. A typical network would consist of an input layer, a hidden layer, and an output layer (Figure 2).

The input layer consisting of input neurons receive a set of signals which may be features. These signals are then fed forward to the hidden layer. Note that the outputs of neurons in preceeding layers are connected to every neuron in the succeeding layer. The hidden layer then acts on these inputs and passes the results onto the output layer. The network is trained by adjusting the weights according to a weight change rule which is derived from the optimization of some cost function.

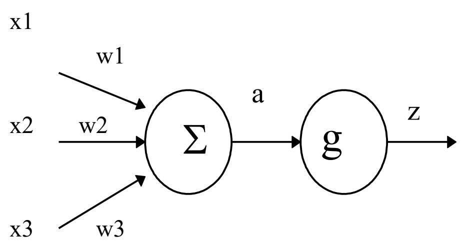

Figure 1.Artificial Neuron

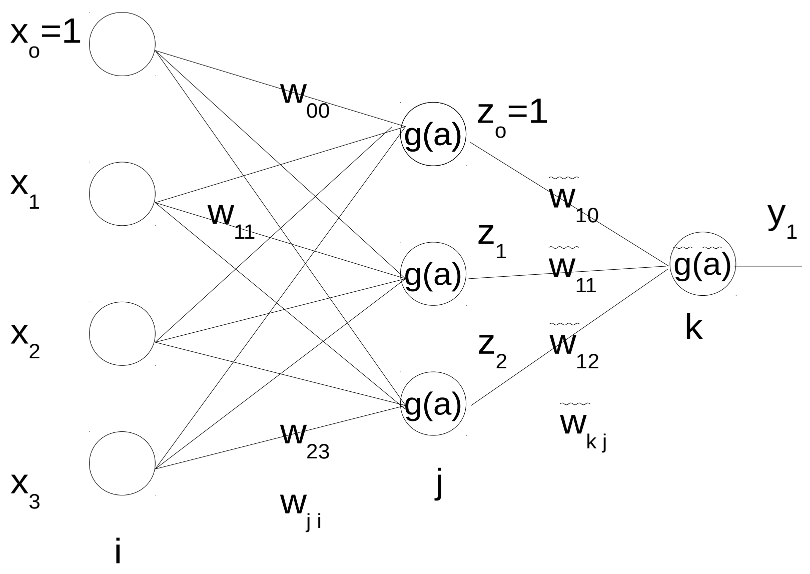

Figure 2. A neural network.

Error Backpropagation

An elegant training algorithm for multilayer neural networks is the error backpropagation. A detailed derivation of the algorithm appears in a review article by C. Bishop [1] which we replicate below. In supervised learning, a desired output is presented to the network. The difference or error between the network's output and the desired output, together with the activation functions in each layer is then used to compute the error derivatives with respect to the weights.

Let z_j be the output of the jth hidden node with activation function g. Then from the perceptron z is given by

$$z_j = g(a_j), a_j = \sum_{i=0}^{d} w_{ji} x_i \quad . \tag{1}$$

Where x_i is the ith input vector component where the input vector **x** has dimension d. Let y_k be the kth output of each output layer node with activation function $\widetilde{g}$. Then y_k is given by

$$y_k = \widetilde{g}(\widetilde{a}_k), \widetilde{a}_k = \sum_{j=0}^{c} \widetilde{w}_{kj} z_j \tag{2}$$

The total sum of squares error E is given by

$$E=\sum_{q=1}^{n} E^q, E^q=\frac{1}{2}\sum_{k=1}^{c}\{y_k(\boldsymbol{x}^q;\boldsymbol{w})-t_k^q\}^2 \tag{3}$$

Where n is the number of data and t_k^q is the desired output for the qth input at the kth output node. Note that there are c output nodes.

Using chain rule we can compute for the derivatives of E^q with respect to the final layer weights w_{kj} as given below

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}}=\frac{\partial E^q}{\partial \tilde{a}_k}\frac{\partial \tilde{a}_k}{\partial \tilde{w}_{kj}} \quad . \tag{4}$$

We introduce the definition

$$\tilde{\delta}_k \equiv \frac{\partial E^q}{\partial \tilde{a}_k} \tag{5}$$

And from Equation (2) we can write (4) as

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}}=\tilde{\delta}_k z_j \quad . \tag{6}$$

From (2) and (3) we can express $\tilde{\delta}_k$ as

$$\tilde{\delta}_k=\tilde{g}'(a_k)\{y_k-t_k\} \quad . \tag{7}$$

Thus, so long as the derivative of the activation function $\tilde{g}$ is known, we have an expression for (7).

Taking the derivatives of E^q with respect to the weights w_{ji} from the input to the hidden layer, we get

$$\frac{\partial E^q}{\partial w_{ji}}=\frac{\partial E^q}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \quad . \tag{8}$$

From equation (1) note that $\partial a_j/\partial w_{ji}=x_i$. If now we define

$$\delta_j \equiv \frac{\partial E^q}{\partial a_j} \tag{9}$$

we can then write equation (8) as

$$\frac{\partial E^q}{\partial w_{ji}}=\delta_j x_j \tag{10}$$

What is left is to find an expression for δ_j . Since we can write (9) as

$$\delta_j=\frac{\partial E^q}{\partial a_j}=\sum_{k=1}^{c}\frac{\partial E^q}{\partial \tilde{a}_k}\frac{\partial \tilde{a}_k}{\partial a_j} \quad , \tag{11}$$

then by using equations (5), (1) and (2) we have

$$\delta_j = g'(a_j) \sum_{k=1}^{c} \tilde{w}_{kj} \tilde{\delta}_k \quad . \tag{12}$$

And again if the activation function between the input and hidden layer has defined derivatives, we can get an expression for (12).

To train a multilayer neural network using error backpropagation, here are the steps:

1. Per pattern q in the data set evaluate the activations of the hidden unit using equation (1) and the activation for the output units using (2).

2. Compute the error of each output unit using

$$\tilde{\delta}_k = \tilde{g}'(a_k)\{y_k - t_k\} \quad .$$

3. Compute the errors for the hidden units using

$$\delta_j = g'(a_j) \sum_{k=1}^{c} \tilde{w}_{kj} \tilde{\delta}_k \quad .$$

4. Compute the error derivatives of this pattern using

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \tilde{\delta}_k z_j$$

and

$$\frac{\partial E^q}{\partial w_{ji}} = \delta_j x_i \quad .$$

5. Repeat steps 1 to 4 for each pattern. Sum the error derivatives to get the derivative of the complete error function.

An epoch is defined as that time when all the patterns in the data set have been presented to the network. After one epoch each weight is then modified using $w_{ji}^{new} = w_{ji}^{old} + \Delta w_{ji}$ where

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \quad . \tag{13}$$

where j refers to the preceeding unit, i refers to the target unit, η is the learning rate. The learning rate may be set to an appropriate value (less than 1.0) by trial and error.

The choice for the activation function g should be any function that is differentiable or even piecewise differentiable. Some example of activation function for g and their derivatives are as follows:

Hyperbolic tangent

$$\begin{gathered} g(a) = \tanh a = \frac{e^a - e^{-a}}{e^a + e^{-a}} \\ g'(a) = 1 + g(a)^2 \end{gathered} \quad . \tag{14}$$

Sigmoid function

$$\begin{gathered} g(a) = \frac{1}{1+e^{-a}} \\ g'(a) = g(a)(1-g(a)) \end{gathered} \quad . \tag{15}$$

Linear Function

$$g(a) = a g'(a) = 1 \quad . \tag{16}$$

Rectified Linear

$$\begin{gathered} g(a) = \begin{Bmatrix} 0 & if\, z<0 \\ a & if\, z>a \end{Bmatrix} \\ g'(a) = \begin{Bmatrix} 0 & if\, z<0 \\ 1 & if\, z>a \end{Bmatrix} \end{gathered} \quad . \tag{17}$$

Neural Network as a Curve Fitting Tool

Besides classification, another problem that can be solved by ta neural network is polynomial fitting. Any function f(x) can be expressed as a series expansion:

$$f(x) = w_o + w_1 x + w_2 x^2 + w_3 x^3 + \ldots \quad . \tag{18}$$

If we let the different powers of x be our input vectors,

$$\begin{gathered} x_o = 1 \\ x_1 = x \\ x_2 = x^2 \\ x_3 = x^3 \\ \vdots \end{gathered} \tag{19}$$

then we can once again use the perceptron input,

$$a = x_1 w_1 + x_2 w_2 + x_3 w_{3.} \ldots = \sum x_i w_i = \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{w} \quad .$$

The network architecture can be a single layer or more. The inputs are different powers of x and the output is a single value equal to the value of the function given x. A neural network with a single layer and a linear activation is the same as a perceptron. In such a case, the learned weights are the polynomial coefficients.

Monitoring learning

When the network has been stimulated with all of the data, the network is said to have gone through one “epoch”. The network may need to be stimulated through several epochs until some cost function reaches a minimum. A common cost function in network training is the sum of squares (SSE) error given by

$$E = \frac{1}{2} \sum_{i=1}^{N} \left(d^i - z^i \right)^2 \quad . \qquad (20)$$

This is the same error as Equation 3. Plotting E vs. epoch allows us to see if the network is converging to a solution. If we want to compare network performance versus different data sizes it is better to use the root-mean-square (RMS) error defined by,

$$E_{RMS} = \sqrt{2E/N} \qquad (21)$$

for a fairer comparison.

Procedure

1. Neural network for regression - Program a neural network to learn a sine function. Alternatively, you may train the network to learn any function.
2. Neural network for classification - Program another neural network to classify your fruit data in Activity 13 (Perceptron). Set aside half of the data from each class as the training set and the remaining half as the test set. Using the test set determine the accuracy of your network.

Reference

1. C. Bishop. *Neural Networks and Their Applications*. Rev. Sci. Instrum. 65(6), June 1994.