Neural Network Structure and Back-propagation

By: Joseph Carboni

Introduction

Artificial Neural Networks provide a useful way to approximate unknown functions given the inputs they receive and the outputs they produce. The fundamental usefulness associated with the use of a neural network lies in the ability to replace a 'black box', the unknown function, with a known system that is able to be trained to behave approximately the same way as the function. The two functions approximated in the original project were:

Problem 1 : $f(x,y) = \frac{1+\sin(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y)}{2},\ x \in (-2,2),\ y \in (-2,2)$

Problem 2 : $f(x,y,z) = \frac{3}{13}\left[\frac{x^2}{2}+\frac{y^2}{3}+\frac{z^2}{4}\right], x \in (-2,2),\ y \in (-2,2),\ z \in (-2,2)$

The data derived from these functions are separated into three separate sets: a training set, a testing set, and a validation set. The training set will be used to train the neural network, the testing set will be used to evaluate the network's performance after each training epoch, and the validation set will be used to assess the network's ability to generalize. I have provided the **.txt** files I used for the project.

Methods

Software

The programming language used for implementation was a statistical programming language called R. Since the implementation of a neural network requires a large number of iterative processes, this language offers an advantage by its ability to parallelize all of the operations occurring in a single layer of the network. However, R may have more trouble than some other languages when having to iterate through many layers in sequence, since these operations cannot be parallelized.

The primary function is called NeuralNet(), and it takes 7 arguments. The first three arguments take file paths for training, testing, and validation datasets respectively. The fourth argument allows the learning rate to be set, which uses a default rate of 0.1. The fifth argument allows the momentum coefficient to be set, which uses a default value of 0.5. The sixth argument allows the number of layers as well as the node content of each layer to be defined simultaneously by a single vector. Thus, the default vector, (2,3,1), defines 2 neurons at the input layer, 3 neurons in a single hidden layer, and 1 neuron at the output later. More hidden layers and their node contents can be defined by making the vector longer, adding more numbers to it. The seventh argument defines the number of epochs to run, which has been assigned a default value of 10,000. The eigth and final argument takes a simple TRUE/FALSE input for generating a plot of the MSE values across all epochs for the training and testing datasets. Supplying TRUE, the default option, will cause a plot to be generated once training is complete, and a value of FALSE will suppress the generation of such a graph.

Other than the potential output for a graph, the function prints the training and testing MSE's after each epoch while training. The function also returns several objects, including the validation MSE, the minimum testing MSE, and several weight and bias value matrices.

Network Structure and Function

Neural Networks are digital constructs inspired by the natural computational function of the brain, which uses many independent neurons (nodes) and synapses (connections) in order to extract information from inputs received from the natural world. In this sense, neurons and their respective weights are represented by arrays of weight values that also inherently hold neuron position and connection information by respective index positions in the arrays. Once the array dimensions have been initialized, representing node contents of successive layers, each weight is assigned a random number between -0.1 and 0.1. Also, a separate vector was used for bias weights associated with each neuron, which represent the neuron's willingness or reluctance to fire, often described as a threshold value for the neuron. Biases were also initialized randomly between -0.1 and 0.1.

For instance, a weight matrix representing the weights between two successive hidden layers, containing 3 and 4 neurons respectively, would be used to calculate the values in the next layer like this:

$$[\sigma_1^l \quad \sigma_2^l \quad \sigma_3^l] \begin{bmatrix} W_{11} & W_{21} & W_{31} & W_{41} \\ W_{12} & W_{22} & W_{32} & W_{42} \\ W_{13} & W_{23} & W_{33} & W_{43} \end{bmatrix} = [h_1^{l+1} \quad h_2^{l+1} \quad h_3^{l+1} \quad h_4^{l+1}]$$

Where σ is the output from each neuron in layer l as defined by the sigmoid function (see below), W_{ij} is the weight of the connection to neuron i in layer $l+1$ from neuron j in layer l, and h is the local field value for each neuron in layer $l+1$, signifying the sum of weighted inputs from the previous layer to that particular neuron.

Once this operation is done, calculating the sigmoid values to be used for the next weight matrix is achieved by applying each element, h_i^{l+1}, to the sigmoid function and keeping these values in a single vector (like above). The sigmoid function used for all of the neurons (except the input layer) is:

$$\sigma(h) = \frac{1}{1 + e^{(-\alpha h)}}$$

Where $\alpha = 1$, and h is the local field value of the neuron. This function is used in place of a hard, discontinuous threshold. It mimics the nature of a "switch", like the way a neuron would either fire or not-fire, but with a continuous function. This is important, because without out, backpropagation would not be possible. For more information about the sigmoid function, which is the logistic function in this case, read the Wikipedia article on it (Logistic function).

Once the outputs for the final neurons are calculated, output values are compared to the desired values of the problem function by subtraction and used for back-propagation. Back-propagation allows the neural network to assess and alter the contributions from neurons in hidden layers with respect to the error seen at the output layer. Back-propagation is achieved in this network structure by performing matrix multiplication between the delta values of previous layers to the transposed weight matrices associated with previous layers and then multiplying the product sums element wise to the sigmoid values of the current layer.

With the delta values of the current layer computed, weight change values for neurons in the next layer are computed by matrix multiplication between a single-column matrix of delta values from the current layer and a single row matrix of sigmoid values from the next layer. The learning rate, a coefficient which dictates the speed at which weights can be changed, is applied here. This matrix is then transposed and added to the weight matrix associated with the next layer.

For instance, using the same example as before between two hidden layers, back-propagation from 4 neurons (previous layer) to 3 neurons (current layer) is computed by:

$$\partial_i^l = \sigma_l(1-\sigma_l)\sum_{k=1}^{n}\partial_k^{l+1}\,W_k^{l+1}$$

$$\partial_i^l = \sigma_l(1-\sigma_l)\begin{bmatrix}\partial_1^{l+1} & \partial_2^{l+1} & \partial_3^{l+1} & \partial_4^{l+1}\end{bmatrix}\begin{bmatrix}W_{11}^{l+1} & W_{12}^{l+1} & W_{13}^{l+1}\\ W_{21}^{l+1} & W_{22}^{l+1} & W_{23}^{l+1}\\ W_{31}^{l+1} & W_{32}^{l+1} & W_{33}^{l+1}\\ W_{41}^{l+1} & W_{42}^{l+1} & W_{43}^{l+1}\end{bmatrix}$$

The two equations above are actually different representations of the same equation, the latter in matrix form. Also note that the weight matrix above is the transpose of the one used during forward propagation (see above). The transpose is used because now we are going in the opposite direction.

Now, the weight-changes associated with the neurons in the next later will be calculated using the delta values for the current layer and the sigmoid values received from the neurons in the next layer, incorporating the learning rate term. We'll assume the next layer, *l - 1*, also has 3 neurons, just like the current layer.

$$\Delta W_{ij}^{l-1} = \eta\left(\begin{bmatrix}\partial_1^l\\ \partial_2^l\\ \partial_3^l\end{bmatrix}\begin{bmatrix}\sigma_1^{l-1} & \sigma_2^{l-1} & \sigma_3^{l-1}\end{bmatrix}\right)^T$$

Where η is the learning rate, usually set to a value in the range (0,1). If you were to calculate the matrix operation inside of the parentheses, you would obtain a transposed matrix of the weight updates with respect to the weight matrix you want to update. Thus, the transpose is taken here so that corresponding elements of the matrix are correct when adding element-wise.

The back-propagation algorithm also employs a momentum coefficient. This allows the network to retain some information about prior weight changes that occurred in previous epochs, and use this information in current weight changes, adding it directly according to some coefficient, β:

$$W_{ij}^{t+1} = W_{ij}^{t} + \Delta W_{ij}^{t} + \beta\Delta W_{ij}^{t-1}$$

It is suggested that using momentum provides considerable learning performance, allowing the network to speed up on flat areas on the error surface and avoid getting stuck in local minima.