Implementation of an Artificial Neural Network Library in C

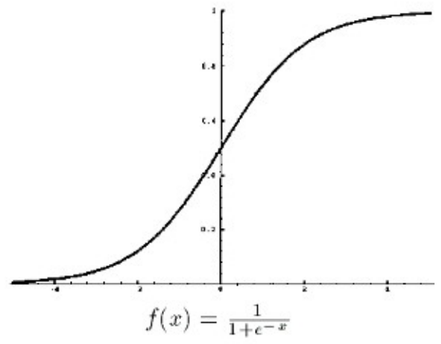
Jack Breese
TJHSST Computer Systems Lab
2007-2008

Abstract

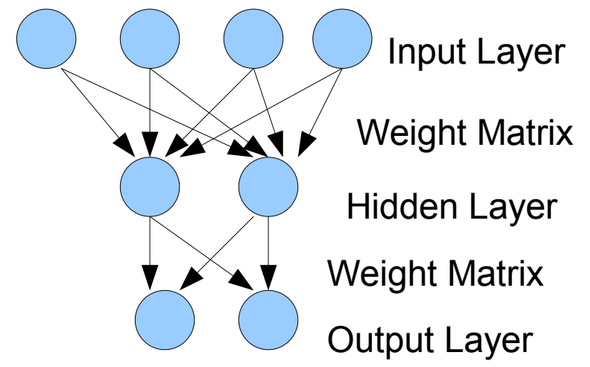
Over the course of the year, I implemented a general-use neural network library in the C programming language. It is capable of creating, managing, saving, and performing simple training on three-layer neural networks using a backpropagation heuristic.

What are Neural Networks?

An artificial neural network is a computational model which emulates the biological structure of the brain. It is a system of interconnected virtual neurons which are capable of modifying their connections, adapting their responses based on accuracy. This modeling of biological networks has widespread use in the field of pattern recognition and object classification, and is well suited to tasks such as optical character recognition and junk email filtering.



1.1 The Sigmoid Function



1.2 An Example Two-Layer Perceptron Neural Network

Image 1.2 is a diagram of the basic layout of a two-layer perceptron neural network. In a task such as face recognition, a neuron is created in the input layer for each discrete pixel value in the input image. The hidden layer is a layer of neurons used only in calculation. Each neuron adds up the values in the preceding layer, multiplied by the weight of each connection in the weight matrix. This process is then repeated for the output layer, where, after being processed by the sigmoid function, the output will be a value between 0 and 1, with 1 corresponding to 100% certainty that the input image matches the condition for that specific output neuron.

Training Neural Networks

My library trains neural networks using a backpropagation algorithm in order to adjust the weights contained within the weight matrices of the network. In training networks using the backpropagation model, they must first be first initialized with random weights. A test image is then read into the network, and each neuron multiplies the data in the layer before it by the connection's random weight. The program then runs the sigmoid function (1.1) on it, in order to simplify the data. A pseudocode algorithm for training with backpropagation and calculating neuron values can be found in the code section below.

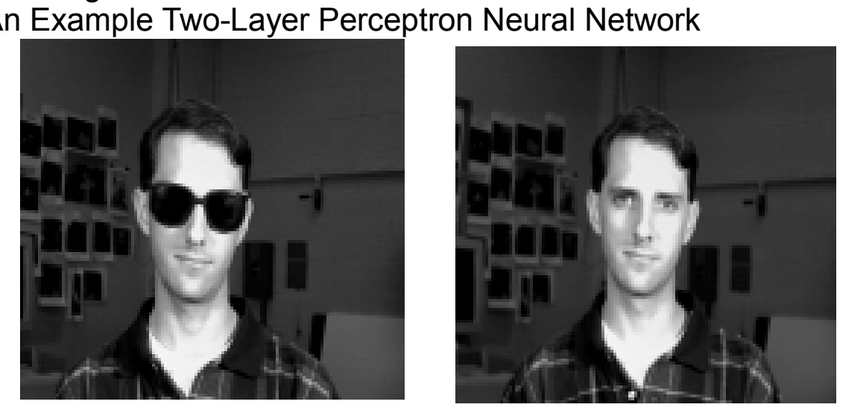
Implementation

The library for neural networks is implemented entirely in C. There are several structs (shown below) which allow for a simpler way of keeping track of the network and iterating through it, as well as simplified management of training data.

Methods exist in this library to initialize two-layer perceptron neural networks of arbitrary size, read data into them, adjust weights in training, propagate data through the network, save the network value, and calculate error.

Use and Results

My neural network library can be included in any C program as a header file, "NNetLib.h" The methods of my library have been tested and verified to work on a sample training data set released by Carnegie Mellon University (2.1) Though my library can read in images to the network, and propagate data through the network, adjust and save weights, due to limitations of the C programming language in the area of string processing, I have found it difficult to automate the training process, as the array of strings my library uses to store training image filenames does not correctly allocate or null-terminate. Ideally, my program would be capable of classifying images of faces similar to those in the sample set based on their defining characteristics, such as whether a person is wearing glasses, or looks angry. Though my library cannot fully complete the training necessary to achieve this, it still provides a useful set of tools for use in the further exploration of neural networks.



2.1 Example Faces from the CMU Sample Training Data

Some Program Methods and Data Structures

Methods

```
neuron* mkNeuron(int c)
     This method allocates memory for a neuron.  C is the number of
     neurons in the previous layer that will connect to it.

Neuron* initLayer(int size, int plsize)
     This method instantiates a layer of size size, with a weight m
     matrix stored for the layer before it, of size plsize.

void calcValue(neuron* n, int s)
     This method calculates the value for a neuron based on the
     weight matrix it stores and the values of the previous layer.
     It should only be called by the calcNet method, never on its
     own.

void calcNet(neuron* i, neuron* h, neuron* o)
     This method propagates the calculations through the entire
     neural network, calling the calcValue method on each neuron.

void readTrain(char* filename, trainInfo * tin, char* zval)
     This method reads training data from a structured file,
     filename, and reads it into tin.  Zval is used in calculating
     the expected outputs for a given training data, and is the
     type of input file that should produce an output of zero.

void readPGM(char* filename, pImg* pgm)
     This method reads in any valid pgm image, filename, to a pImg
     struct (which must first be allocated.)  It is used in reading
     in image data to train and test the neural network.

void backProp(neuron* i, neuron* h, neuron* o, double r, double e)
     This method performs backpropagation in order to adjust the
     weights on the neural network during the training process.
     R is the rate at which training should be performed, and e is
     the error from what is expected.

void saveWeights(neuron* i, neuron* h, neuron* o)
     This method saves the weight matrices of each layer in the
     neural network, so that it can be loaded and used again after
     training.

double act(double x)
     This method is the sigmoid activation function pictured above.
```

Data Structures

```
connection
     This struct holds a double weight, and a neuron pointer.  It
     each neuron holds an array of connections pointing to the
     layer preceding it, in order to store the weight matrix.
Neuron
     This struct holds a double, d, which stores the neuron's
     value,a double err, which stores the calculated error for
     training use, and an array of connections.
pImg
     This struct stores a valid pgm formatted image file, including
     all attributes, as well as raw ASCII formatted image data.
TrainInfo
     This struct stores an array of training image files, as well
     as an array of doubles which holds the expected values for
     each input file.
```

Algorithms

Calculating Neuron Values

```
     For each neuron in the previous layer:
               Sum += neuron_weight*neuron_value
     neuron_value = activation_function(sum)
```

Training

```
     For each neuron in the network:
               For each connection to the neuron:
                         weight = random_value()
     Until desired accuracy is reached:

               For each example in the training data:
                         actual_out = run_network(example)
                         exp_out = calculate_expected(example)
                         error = exp_out – actual out

                         For each neuron in the network:
                              calculate_delta_weights(error)
                              adjust_weights(neuron)
```