Using Neural Networks to Solve a Classification Problem

Introduction
Artificial Neural Networks are a biologically inspired mathematical model which uses “neurons” connected via “synapses” to move data through a network. Feed-forward artificial neural nets use the back-propagation algorithm to systematically change the synaptic weights while (input, output) vectors are presented to the network. During this process, the network is able to discover regularities in the data which map the input to the output (provided such a function exists, and the network is given sufficient time, neurons, and a small enough step size).

In this project, I programmed a feed-forward, artificial neural network and used it to classify letters from an adapted version of the “Artificial Character Database” from UCI Machine Learning Repository. This paper provides a brief explanation of the problem, shows the basic design of the neural network, and provides results of testing the problem set on a few different neural network architectures.

The Problem
This problem is a basic classification problem. A grid representation of the “Artificial Character Database” has been provided by Dr. Parker. In it, each character is represented as a 12x8 array of binary pixels, with 0 representing “off” and 1 representing “on”. 600 example images were used, and these were divided into three groups: 100 images were used as training data, 250 images were used as validation data after each epoch of training data, and finally 250 images were used to test the resulting network architecture. The images represented 10 capital letters, A, C, D, E, F, G, H, L, P, and R.

The Network
The artificial neural network used is a feed-forward ANN using back-propagation to update synaptic weights. There are 96 input neurons, each representing one of the 12x8 pixels in the image. There are 10 output neurons, each representing a different letter, of which one should output 1 (the correct letter) while all others output 0 (indicating that is not the correct letter).

The hidden neurons were activated using \( \tanh(x) \), as it allows the output to be between -1 and 1. Its derivative \( (sech(x))^2 \), was used for back propagation.

Soft-max was used to scale the output neurons to values between 0 and 1, making their error calculation easier and training more efficient.

The network was trained “online”; for each epoch, first a permutation of the input set was produced. This input set was then run, in its permuted order, through the network. After each input example,
weights were updated. Upon completion of the epoch, the sum of squared errors for the all in input examples was recorded. Additionally, the sum of squared errors for each output neuron was recorded after each epoch. The validation data was then presented to the network. No weight updates occurred during validation. After the validation data was presented, the sum of squared errors for the validation data was recorded.

Finally, after all epochs were performed, the network was tested using the testing data, and its total accuracy, accuracy for each letter, and confusion matrix are presented to the user.

Results

Given the sum of squared errors for each epoch of both the training and validation data, we are able to observe the training and validation errors for a particular network architecture to quickly determine a) if the network is capable of classifying the input data to our satisfaction, and b) at what point we should stop training the network, as the validation error is no longer decreasing.

Using 40 neurons, with an eta value of .1, I trained for several hundred epochs. Here are the error results for the first 50 epochs, showing that strong results are obtained after as few as 25.

![Error vs Epochs](image)

This network has an overall accuracy rating of .958, and achieves perfect classification for some letters. Its confusion matrix shows which letters look quite similar to the network:
As we can see, “A” is often misclassified as “H” and vice-versa. Similarly, the network gets confused attempting to distinguish “P” from “F”.

And here is the success rate for each letter:

<table>
<thead>
<tr>
<th>a</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>l</th>
<th>p</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.896</td>
<td>1</td>
<td>0.984</td>
<td>0.976</td>
<td>0.904</td>
<td>0.976</td>
<td>0.932</td>
<td>1</td>
<td>0.924</td>
<td>0.992</td>
</tr>
</tbody>
</table>

This network is provided with these files, and can be loaded into the network and tested if desired.

For fun, a terrible network design is provided. With only 4 neurons and an eta of .4, the network is only able to achieve a bit above 50% overall accuracy on the testing data:

Figure 2: A bad network