Classification of MNIST Digits Using a Feedforward Neural Network and Backpropagation

Daniel Alejandro Noble Hernandez
dan833
The University of Texas at Austin

---

1. Introduction

Feedforward neural networks, paired with backpropagation, are an efficient tool for machine learning applications, such as classification problems. A feedforward neural network, one of the simplest neural networks available, moves in only one direction and is an artificial neural network in which the nodes do not form connections in such a way that they form cycles. By moving forward through the neural network, one can come to an estimate for a prediction, and this can then be refined using the backpropagation algorithm. Backpropagation works by computing the gradient of the loss function with respect to the weights of the network, working backwards to refine the estimate for each of the weights at the different layers.

The purpose of this project was to implement a feedforward neural network on the MNIST dataset of hand-drawn digits to come up with a classification for each of them. As part of this process, the code would implement the backpropagation algorithm to refine the weights of each layer of the neural network, generating increasingly more accurate classification results over the course of the iterations.

---

2. Methods

2.1 Feedforward Neural Network

The dataset used consists of 60,00 training images and 10,000 testing images of hand-drawn digits 0-9 with their respective labels. Each image was represented by a greyscale 28x28 pixel array, for which each value was between 0 and 255. Each image was rolled out into a 1-dimensional array to make the data easier to work with.

The author began by implementing a class called NeuralNetwork that took the data and labels as inputs for the input layer and output layer respectively, and before instantiating it, one hot encoded the labels to provide a simple mapping system for the classification. The images' pixels were scaled to take on values between 0.01 and 1, and at this point, the NeuralNetwork class could be implemented using the modified dataset. Over a set number of epochs, the feed

forward algorithm was implemented, followed by the back propagation algorithm. The feed forward aspect was done as follows.

1. Calculate Z1 = Inputs * Weights + Bias1
2. Calculate A1 = sigmoid(Z1)
3. The above two steps were repeated for each layer of the neural network, so one more time in this case as the network had only two layers. For the outer layer, use the softmax function instead of the sigmoid function.

As seen above, one must use an activation function for each of the layers; in this case, the sigmoid function was used for convenience and efficiency. Notice that for the final layer, the author uses the softmax function rather than the sigmoid function as the activation function. Both of theses shrink the input to the (0, 1) interval but softmax further ensures that the outputs sum to 1, which makes sense for a probability distribution.

2.2 Backpropagation

Following this step, the back propagation algorithm could be implemented via the following steps:

1. Calculate the loss/error between the true value and the estimate attained through the feed forward algorithm.
2. Compute delta_a2 = cross_entropy(A2, Output)
3. Calculate Weight_2 -= learning_rate * A1.T * delta_a2
4. Calculate Bias2 -= learning_rate * sum(delta_a2)
5. Calculate delta_z2 = delta_a2 * Weight_2.T
6. Repeat step 2-5 for the initial layer

Through each iteration over the epochs, incorporated in the back propagation algorithm was a command to store the error in an array within the NeuralNetwork class; this could now be used to provide a plot of the error over the increasing iterations of training the network, the results of which are shown below.

---

3. Results

Figure 1 depicts the results of training the neural network – i.e. implementing the feedforward and backpropagation algorithms – over 1000 epochs. Using the cross-entropy function, rather than the standard mean squared error, allows for the computation of error between two probability distributions, which is ideal for the problem that this implementation deals with. One can see that although the resulting error starts out quite high, it drops off very quickly over the first few iterations, and reaches a relatively stable value by the 1000^{th} epoch.

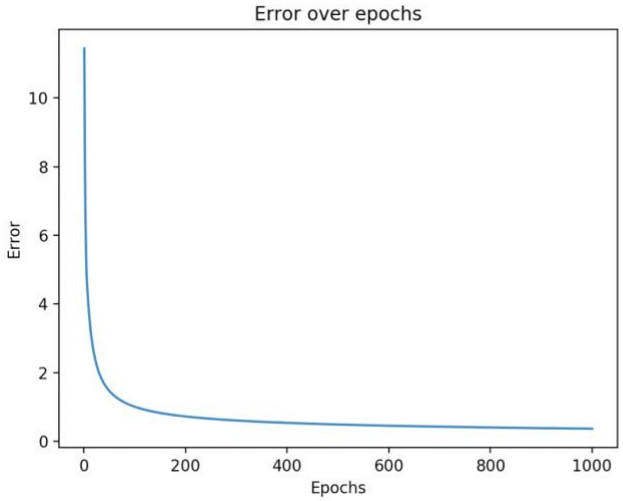

Figure 1: The graph shows that the loss generated by the feedforward network decreases significantly over training iterations, particularly so over the course of the first 100 iterations, and seems to reach a stabilized point by the 100th iteration.

Below, one can see the resulting accuracy of the neural network as a classifier for the MNIST dataset after training over 1000 epochs. Running the program with different numbers of epochs shows that there is little improvement in accuracy after around 1000 epochs, which is further supported by the flattened nature of the curve in Figure 1 towards the end.

```
Training accuracy :  89.435
Test accuracy :  89.02
```

Figure 2: The accuracy of the neural network on classifying the training and testing data after 1000 epochs hovers a little of 89%.

---

4. Conclusion

This implementation of a feedforward neural network with backpropagation to refine the weights of each layer yields a satisfactory classification algorithm for the MNIST dataset. This statement is made in light of the considerations of efficiency, accuracy, and volume of data used. The code can be compared with the earlier assignment of using a k-nearest neighbours algorithm to classify the data, with and without PCA. Although the KNN algorithm alone yielded an accuracy of 93% compared to the neural network's accuracy of 89%, it took significantly longer (6+ hours) on a fraction of the training images. On the other hand, the neural network came up with this accuracy within 10 minutes, on the full dataset, so is significantly more efficient and sacrifices a fairly small amount of accuracy. Comparing it to the KNN algorithm with PCA, the accuracy is higher in all cases. Even taking all 784 eigenvectors to maximize the accuracy to 86%, it was still lower than this implementation's accuracy, and took 2.5 times longer to execute. Compared to that earlier project then, the neural network is highly preferable as a classification algorithm for the MNIST dataset.