Neural Network for Recognition of Handwritten Digits

Upayan Mathkari
upayan@cs.utexas.edu

The University of Texas at Austin
Dept. of Mechanical Engineering

1. Introduction

Artificial neural networks (ANN) are computing systems inspired by the biological neural networks that constitute animal brains. The ANN consists of series of interconnected nodes, called neurons, aggregated in series of layers where the input layer consists of raw data fed into the network and the output represents the network's prediction or conclusion regarding the data. Connections between neurons of different layers are analogous to synapses where an input signal can be processed by one neuron than passed on to other neurons it is connected to. The overall strength of the signal passed from one neuron to the next is a function of the weight and bias of the connection. With this rough framework, an ANN can be trained to "learn" by adjusting fine-tuning its weights and biases until a desirable accuracy of performing its desired function is achieved.

In this project, we were tasked with training an ANN to accurately classify handwritten digits from the MNIST database. The network was trained using backpropagation methods to determine the optimal weights and biases for the training dataset and then tested using the test dataset also available from MNIST. Due to the sheer size of the dataset, stochastic gradient descent was utilized to speed up the training time.

2. Methods

Structuring the Network
In the effort of maximizing accuracy, we tried a variety of different network structures however the size of the input and output layer were kept consistent. The input layer was sized with 784 neurons to represent the 784 input pixels while the output layer was sized with 10 neurons to represent the 10 different digits that could be predicted for each input.

Preprocessing
To make the computation of the cost function easier, one-hot encoding was utilized to transform the labels or target values in the dataset into a 10-dimensional column vector with magnitude 1 where the n-th row corresponding to the digit *n* in the label was assigned a value of 1 while the other rows were assigned values of zeros. This made computing the cost function easier as it became as easy as finding the vector difference between the 10-dimensional output of the ANN and the 10-dimensional target value.

Backpropagation Algorithm
This algorithm is what does the "learning" in the neural network. The goal of the algorithm is to determine the combination of weights and biases for every connection in the network that yields the minimal value of the cost function, which essentially serves a measure of the network's inaccuracy. In this example, we use the quadratic cost function defined as:

$$C_x \equiv \frac{\|y(x)-a\|^2}{2}$$

In minimizing the cost function, we minimize it average value of over all the training examples tested. To reduce the computational time of testing each and every training example, we utilize stochastic gradient descent which groups the examples into random mini-batches representative of the entire dataset and computes the cost for these mini-batches instead of the whole training set. The gradient itself is comprised of the cost function’s partial derivatives with respect to each of the weights and biases in the network. Iterating with respect to the gradient is as simple as adding the gradient times a learning rate, *nu*¸ to the input parameter and adjusting as shown:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$
$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

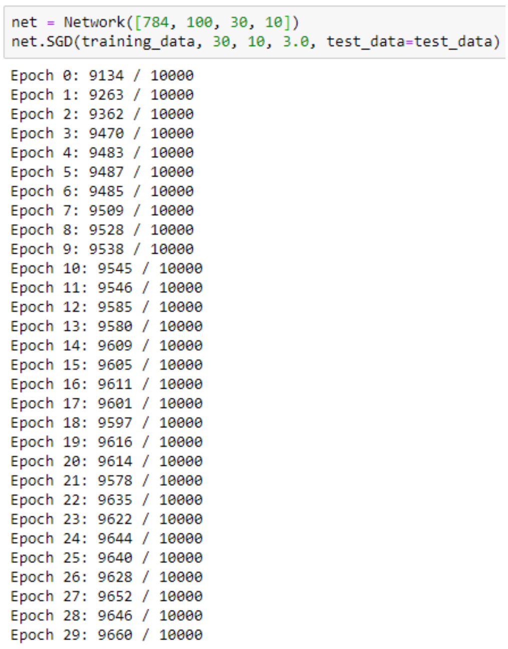
However, the actual computation of this gradient vector is quite challenging in a neural network with thousands of input weights and biases. Backpropagation is an algorithm for an efficient calculation of this gradient. It utilizes the chain rule to determine how a small change in one weight can be propagated across layers into the output as shown:

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_{mnp\ldots q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \ldots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}}.$$

This equation represents how the partial derivative of the cost function with respect to one particular weight can be viewed as the sum of all the derivate paths through which the weight can affect the cost function.

3. Results

With trial and error, the optimal learning rate was determined to be 3.0. Different structures were experimented with, however, a simple 4-layer structure with two hidden layers consisting of 100 and 30 neurons respectively, proved to be most effective with an average accuracy of 96.6%. Experimentation with different structures was challenging owing to the excessive computational time required to train the network.



```
net = Network([784, 100, 30, 10])
net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

```
Epoch 0: 9134 / 10000
Epoch 1: 9263 / 10000
Epoch 2: 9362 / 10000
Epoch 3: 9470 / 10000
Epoch 4: 9483 / 10000
Epoch 5: 9487 / 10000
Epoch 6: 9485 / 10000
Epoch 7: 9509 / 10000
Epoch 8: 9528 / 10000
Epoch 9: 9538 / 10000
Epoch 10: 9545 / 10000
Epoch 11: 9546 / 10000
Epoch 12: 9585 / 10000
Epoch 13: 9580 / 10000
Epoch 14: 9609 / 10000
Epoch 15: 9605 / 10000
Epoch 16: 9611 / 10000
Epoch 17: 9601 / 10000
Epoch 18: 9597 / 10000
Epoch 19: 9616 / 10000
Epoch 20: 9614 / 10000
Epoch 21: 9578 / 10000
Epoch 22: 9635 / 10000
Epoch 23: 9622 / 10000
Epoch 24: 9644 / 10000
Epoch 25: 9640 / 10000
Epoch 26: 9628 / 10000
Epoch 27: 9652 / 10000
Epoch 28: 9646 / 10000
Epoch 29: 9660 / 10000
```

4. Summary

This project demonstrates that Artificial Neural Networks provide an effective means for image recognition. One of the challenges associated with construction of the ANN is the sheer computational time required for training the network making experimentation challenging. Rather, developing an efficient neural network requires a better understanding of the concepts.