Back-Propagation

Bharath Masetty

bmasetty@utexas.edu — bm37498

1 Introduction

Back propagation is a fundamental method used for building machine learning models involving artificial neural networks(ANNs). It is a gradient based method used to update the connecting network weights at each layer of the ANN based on the loss function. In this work, we demonstrate the implementation of this fundamental method to classify the MNIST dateset. Throughout the assignment, we analyse multiple variants of a simple three layer neural network (input layer, hidden layer and output layer) with different choices of activation functions and hidden layer size. In the end, we analyse the performance of the model on the sigmoid and the rectified linear(ReLU) activation function on networks with different number of hidden layer nodes.

2 Methods

We are building a classifier to identify the images from MNIST hand-written digit dataset which consists of 60000 training images and 10000 testing images along with corresponding labels. Each image consists of 28 x 28 pixels (784 total pixel values). We use a simple three layer fully connected neural network to build the classification model. A scaled version of the network architecture is shown in Figure 1. The size of the input layer and the output layer are fixed at 784 and 10 respectively. The size of hidden layer is varied from 50 to 650 in steps of 100 units to analyse the variation in classification performance. The input layer does not have any activation function and softmax function is used to calculate the prediction probabilities at the output layer.Two choices if activation functions were used for the hidden layer, sigmoid and ReLU, as shown in [1] and [2] respectively.

$$z_j = 1/1 + e^{-\Sigma x_i w_{ij}} \tag{1}$$

$$z_j = \Sigma x_i w_{ij} * \chi(\Sigma x_i w_{ij}) \tag{2}$$

where $\chi(x)$is an indicator function. The shapes of the two activation functions along the real axis is shown in Figure 2.

Let the weights connecting input and hidden layer be represented by w_{ij} and the weights connecting the hidden and the output the layer be w_{jk}. Let E be the loss represented as a function of weights as shown in eq.(3). x_i, y_i are the input and output vectors of the nodes in i^{th} layer respectively.

$$E(\mathbf{w}) = \Sigma_{n=1}^{N} E_n(w) \tag{3}$$

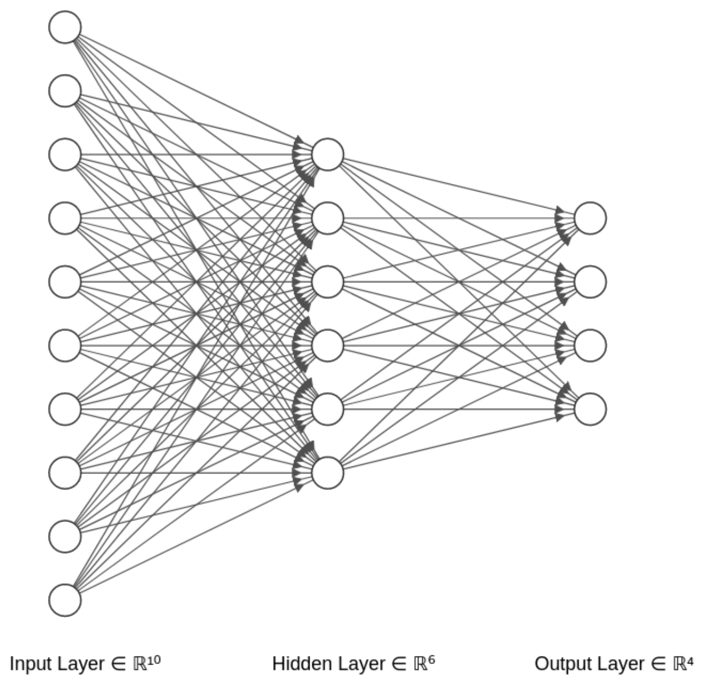

Figure 1: Scaled version of the network architecture.

$$E_n(w) = 0.5\Sigma_k(y_{nk} - t_{nk}) \tag{4}$$

N is the number of nodes in the output layer. The equation to evaluate the gradients with respect to the weights are given by the following equation.

$$\frac{\partial E_n}{\partial w_{ij}} = \delta_j y_i \tag{5}$$

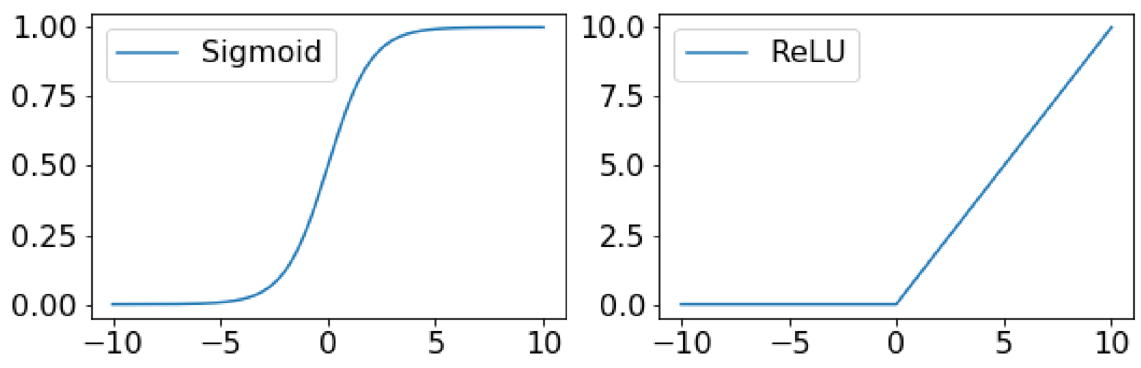

Figure 2: Activation functions.

Algorithm for back-propagation:
1. Initialize the network weights using a uniform distribution between 0 and 0.0001.

```
For each training batch:
    For each training image:
        2. Forward propagate the input to get the softmax activation at the output layer.
        3. Calculate the $\delta_k = y_k - t_k$ using the responses from output layer and target values.
        4. Calculate the $\delta_j = h'(a_j)\Sigma_k w_{jk}\delta_k$ for the hidden layer.
        5. Evaluate the gradients using the following equations using equation (5).
        6. Update the weights $w = w - \eta\Delta w$, where $\eta$ is the learning rate.
    7. Calculate the mean training loss, mean testing loss and testing accuracy.
Repeat steps 2-7 until termination.
```

3 Results

Figure 3 shows the variation of training loss, testing loss and testing accuracy of a sample architecture with 650 hidden layer nodes. The metrics are presented for both ReLU and Sigmoid activation functions.

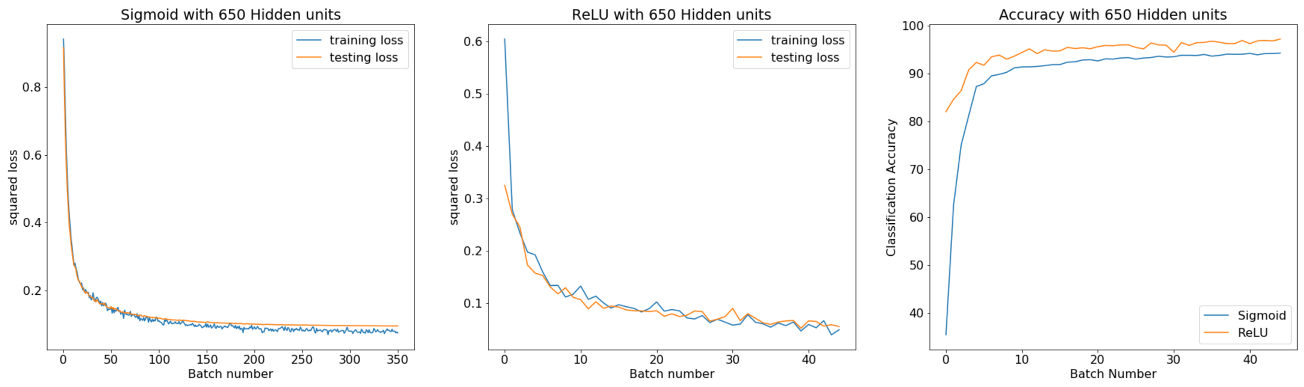

Figure 3: Performance of network with 650 hidden layer units.

It can be clearly observed that the ReLU activation functions results in convergence to asymptotic performance of 97 percent testing accuracy, much faster than the sigmoid activation function. The first reason for the faster convergence of ReLU over sigmoid is the reduced likelihood of ***vanishing gradients***. As the absolute value of the input at a given node increases, the slope of the sigmoid function decreases which could result in close to zero gradients of the loss function. On the other hand, in ReLU, the slope of the activation function is constant if the input is positive. This reduces the chance of resulting in vanishing gradients.

The second reason is the ***sparsity*** of ReLU when ever the input to the node is negative. More such inputs, greater is the sparsity of the representation. On the other hand, sigmoid always gives a non-zero output for any given input value. Having a certain degree of sparsity in the activation function has been observed to be useful for better representation.

Figure 4 shows the analysis similar to the one shown in Figure 3 for 7 different network architectures with increasing number of hidden layer nodes. When we augment the network architecture in this way, it is expected to perform better because, by adding more nodes, we are increasing the complexity of the function to better represent our model.

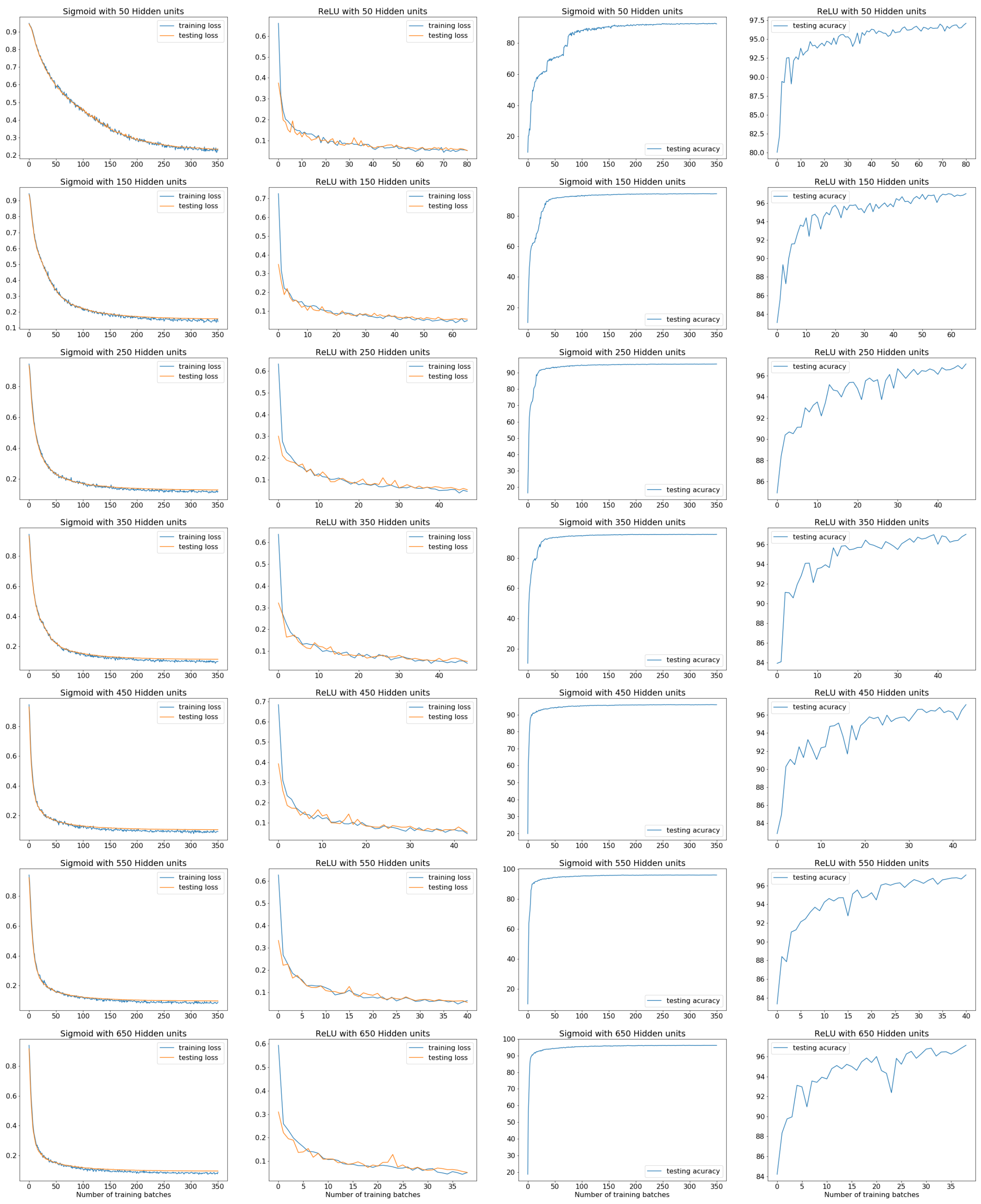

Figure 4: Variation of the training loss, testing loss and classification accuracy with increasing number of training batches on different network architectures.

As expected, the training and testing losses decrease more rapidly for the networks with larger number of hidden layer nodes. The effect is more pronounced for sigmoid activation function than ReLU. This aspect can be more clearly observed in Figure 5 which shows the variation of testing accuracy with increasing number of hidden layer nodes for both activation functions. The performance of sigmoid activation substantially increases with number of nodes as opposed to ReLU which is almost invariant.

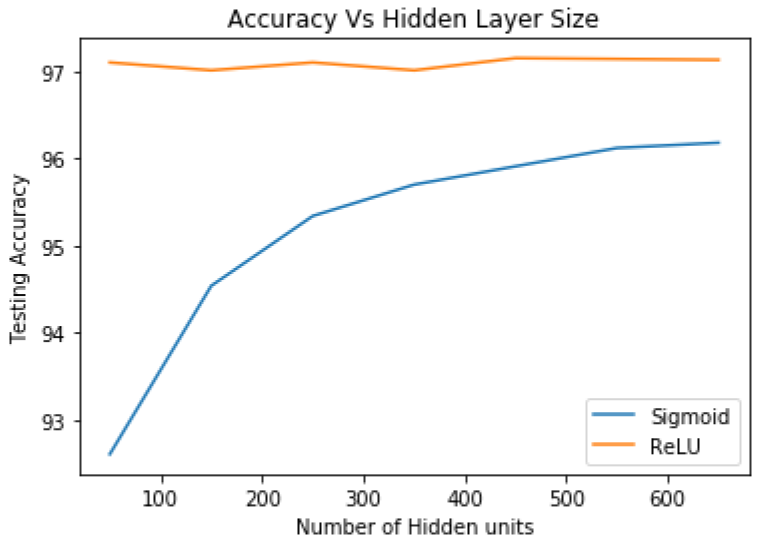

Figure 5: Variation of the testing accuracy with increasing number of hidden units for sigmoid and ReLU activation functions.

The reason for this difference in response to hidden nodes can also be owed to the properties of ReLU and Sigmoid. Intuitively, we can explain this by looking at the sparsity of the activation functions. Sigmoid is less sparse than ReLU, which means, more information is represented in less space. By increasing the number of nodes, we are providing the network with a way to relax the density of the representation in the network. Hence, more nodes results in faster learning. Since, ReLU is not a bounded activation function and sparser than sigmoid, not much utility is added to the network via the extra hidden nodes. For this reason, sigmoid is more sensitive to the variation in network architecture than ReLU.

4 Conclusion

In this assignment, we have implemented the back-propagation algorithm for training an artificial neural network to classify the MNIST Handwritten digits dataset. A comparative analysis has been presented to understand the performance of ReLU and Sigmoid activation functions. The overall analysis shows that ReLU provides better performance as compared to Sigmoid in a relatively lesser number of training routines. We also observed that the networks with sigmoid activation function are more sensitive to increasing the number of hidden nodes than ReLU. Both these difference were explained using the concept of sparsity and vanishing gradients and their effect on representing information into the model.