Homework 5: Backpropagation

Thomas Wei
ThomasW219@gmail.com

***Abstract*—Neural networks are a powerful class of function approximator that have seen greater popularity with the increase in computation power over the last two decades. In addition, the vast amounts of data collected in modern times lends itself well to neural networks since they are often overparameterized and consequently require vasts amount of training data.**

In this paper we use fully connected neural networks to classify the MNIST dataset, a set of images of handwritten digits and their labels. We utilize fully connected neural networks and test their performance with different sized hidden layers, different numbers of hidden layers, and different activation functions.

I. Introduction

Neural networks are a powerful class of function approximator that have seen greater popularity with the increase in computation power over the last two decades. In addition, the vast amounts of data collected in modern times lends itself well to neural networks since they are often overparameterized and consequently require vasts amount of training data. Given the requisite training data, they can be used for a variety of tasks including regression, multi-class classification, multi-label classification and so on.

In this paper we use fully connected neural networks to classify the MNIST dataset, a set of images of handwritten digits and their labels. We utilize fully connected neural networks and test their performance with different sized hidden layers, different numbers of hidden layers, and different activation functions. Since the search space for neural network hyperparameters is so large, we only consider the effects of these changes in very specific conditions. For example, we hold the optimization algorithm and learning rate constant for all of our experiments and for the experiments where the number of hidden layers is varied, we keep the number of neurons in each hidden layer constant.

We found that in our experiments sigmoid activation functions on the hidden layers slightly outperformed ReLU activation functions (all other hyperparameters held constant) within 100 epochs of training. We also found that models with two hidden layers slightly outperformed models with one or three hidden layers within 300 epochs of training.

II. Method

A. Network Details

We first start by defining our loss function, the function that we're trying to optimize with backpropagation. Since we're working with the MNIST dataset and each image is one of ten digits, we're trying to solve a multi-class classification problem. As such, we use a softmax activation function and cross entropy loss between the output of the softmax layer and the desired vector. Let $\mathbf{p} = (p_1, \ldots, p_{10})^T$ be the output of the softmax layer when the network is given input $\mathbf{x}$. Then $\mathbf{y} = (y_1, \ldots, y_{10})^T$ is one-hot encoded vector of what digit the image $\mathbf{x}$ actually is. Our categorical cross entropy loss is given below.

$$L(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{10} y_i \log(p_i) \tag{1}$$

The softmax layer is just the final layer. The entire network will also include at least one hidden layer with either a sigmoid or ReLU activation function. The number of hidden layers and the size of each hidden layers are variable and will be used as a parameter in some of our experiments.

Each layer is fully connected to the previous and each connection bears a weight. The connections from a m node layer to a n node layer can be represented by a $n \times m$ matrix holding the weight of each connection. Given the values of the m node layer, the values of the following layer can be computed through multiplication with the weight matrix. To allow the neural net to model a larger set of functions, a non-linear function $g_k : \mathbb{R}^n \to \mathbb{R}^n$ is then applied to the product.

$$\mathbf{x}_{k+1} = g_k(\mathbf{W}_k \mathbf{x}_k) \tag{2}$$

In our case, we will unroll the 28 by 28 MNIST images into vectors of length 784. Doing so obfuscates any spacial relation in the data. Since our model is fully connected, the manner in which the pixels are mapped to entries in the vector would have no effect on the results of our model. Convolutional neural nets were developed to take advantage of spacial relationships in the data, however we will only focus on fully connected neural nets in this paper.

B. Backpropagation Derivation

Using the Euler-Lagrange formulation, we define the Hamiltonian to be:

$$H = L(\mathbf{x}^K, \mathbf{y}) + \sum_{k=0}^{K-1} (\boldsymbol{\lambda}^{k+1})^T [-\mathbf{x}^{k+1} + g_i(\mathbf{W}^k \mathbf{x}^k)] \tag{3}$$

Where $\mathbf{x}^K = \mathbf{p}$, $\mathbf{x}^0$ is the input layer, $\mathbf{W}^k$, $k = 0, 1, \ldots, K-1$ are transition matrices for the network, g_k, $k = 0, 1, K-2$ are either ReLU or sigmoid activations and g_{K-1} is our softmax activation. We then take the gradient with respect to each vector, giving us:

$$\nabla_{\mathbf{x}^K} H = \begin{bmatrix} -\frac{y_1}{x_1^2} \\ \vdots \\ -\frac{y_{10}}{x_{10}^2} \end{bmatrix} - \boldsymbol{\lambda}^K = 0 \tag{4}$$

for K and for any $k < K$ we have:

$$\nabla_{\mathbf{x}^k} H = -\boldsymbol{\lambda}^k + \nabla_{\mathbf{x}^k}[\boldsymbol{\lambda}^{k+1} g_k(\mathbf{W}^k \mathbf{x}^k)] \tag{5}$$

$$= -\boldsymbol{\lambda}^k + [(\mathbf{W}^k)^T \boldsymbol{\Lambda}^{k+1} g_k'(\mathbf{W}^k \mathbf{x}^k)] = 0 \tag{6}$$

Where $\boldsymbol{\Lambda}^k$ is a diagonal matrix with the elements of $\boldsymbol{\lambda}^k$ along the diagonal. We use these expressions to recursively calculate the values of the $\boldsymbol{\lambda}^k$ from the top down after finding the value of $\mathbf{x}^K$ from the bottom up. Note (6) only holds if the activation function can be interpreted as a vector of functions that take a single input and a single output. This does not apply to our final softmax activation function but we will discuss the derivation for the gradient for that in a later section.

To compute these expressions, we need to know the derivative of the activation functions. For now, let the inputs to the activations be written as z, g_0 be the ReLU function, and g_1 be the sigmoid function.

$$g_0'(z) = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases} \tag{7}$$

$$g_1'(z) = g_1(z)(1 - g_1(z)) \tag{8}$$

Now that we're able to fully compute the values of $\boldsymbol{\lambda}^k$, we can compute the derivative of the Hamiltonian with respect to individual weights.

$$\frac{\partial H}{\partial w_{ij}^k} = \lambda_i^{k+1} x_j^k g_k'(\mathbf{w}_i^k \cdot \mathbf{x}^k) \tag{9}$$

Since we want to minimize the Hamiltonian, we adjust the weight in the negative direction of the gradient.

C. Softmax Derivation

We wish to continue from (5) in the case where g_k cannot be expressed as a vector of scalar input, scalar output functions. Given a vector input $\mathbf{z} \in \mathbb{R}^m$, the $i \in \{1, \ldots, m\}$ component of the softmax output is given by:

$$g^i(\mathbf{z}) = \frac{\exp(z_i)}{\sum_{j=1}^m \exp(z_j)} \tag{10}$$

Because the output is normalized, we can interpret the values as probabilities. In order to continue from (5) we first consider the derivative of H with respect to a single element of $\mathbf{x}$:

$$\frac{\partial H}{\partial x_i^k} = (\boldsymbol{\lambda}^{k+1})^T \frac{\partial g(\mathbf{W}^k \mathbf{x}^k)}{\partial x_i^k} \tag{11}$$

We then invoke the chain rule to compute the desired partial derivative and repeat for each element of $\mathbf{x}$ to get the full gradient.

$$\frac{\partial H}{\partial x_i^k} = (\boldsymbol{\lambda}^{k+1})^T \frac{\partial g(\mathbf{W}^k \mathbf{x}^k)}{\partial \mathbf{W}^k \mathbf{x}^k} \frac{\partial \mathbf{W}^k \mathbf{x}^k}{\partial x_i^k} \tag{12}$$

$$= (\boldsymbol{\lambda}^{k+1})^T \begin{bmatrix} \frac{\partial g^1(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^1} & \cdots & \frac{\partial g^1(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^m} \\ \vdots & & \vdots \\ \frac{\partial g^m(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^1} & \cdots & \frac{\partial g^m(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^m} \end{bmatrix} \begin{bmatrix} w_{1i}^k \\ \vdots \\ w_{mi}^k \end{bmatrix} \tag{13}$$

Where the derivative of a softmax output with respect to the one of the elements of the input is given by the following.

$$\frac{\partial g^i(\mathbf{z})}{\partial z_j} = g_i(\mathbf{z})(\delta_{ij} - g_j(\mathbf{z})) \tag{14}$$

Here, $\delta_i j$ is the Kronecker delta. The derivative with respect to the weights is computed similarly except it can be simplified further to the following:

$$\frac{\partial H}{\partial w_{ij}^k} = x_k^j (\boldsymbol{\lambda}^{k+1})^T \begin{bmatrix} \frac{\partial g^1(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^i} \\ \vdots \\ \frac{\partial g^m(\mathbf{W}^k \mathbf{x}^k)}{\partial (\mathbf{W}^k \mathbf{x}^k)^i} \end{bmatrix} \tag{15}$$

III. Results

For all of these experiments, we used the Adam optimization algorithm which involves weighting of gradients from different steps and the concept of momentum. We also kept the learning rate and batch size constant throughout the experimentation as well. The averages were taken over 100 different random initializations of the neural net parameters.

A. Variation of hidden layer dimension

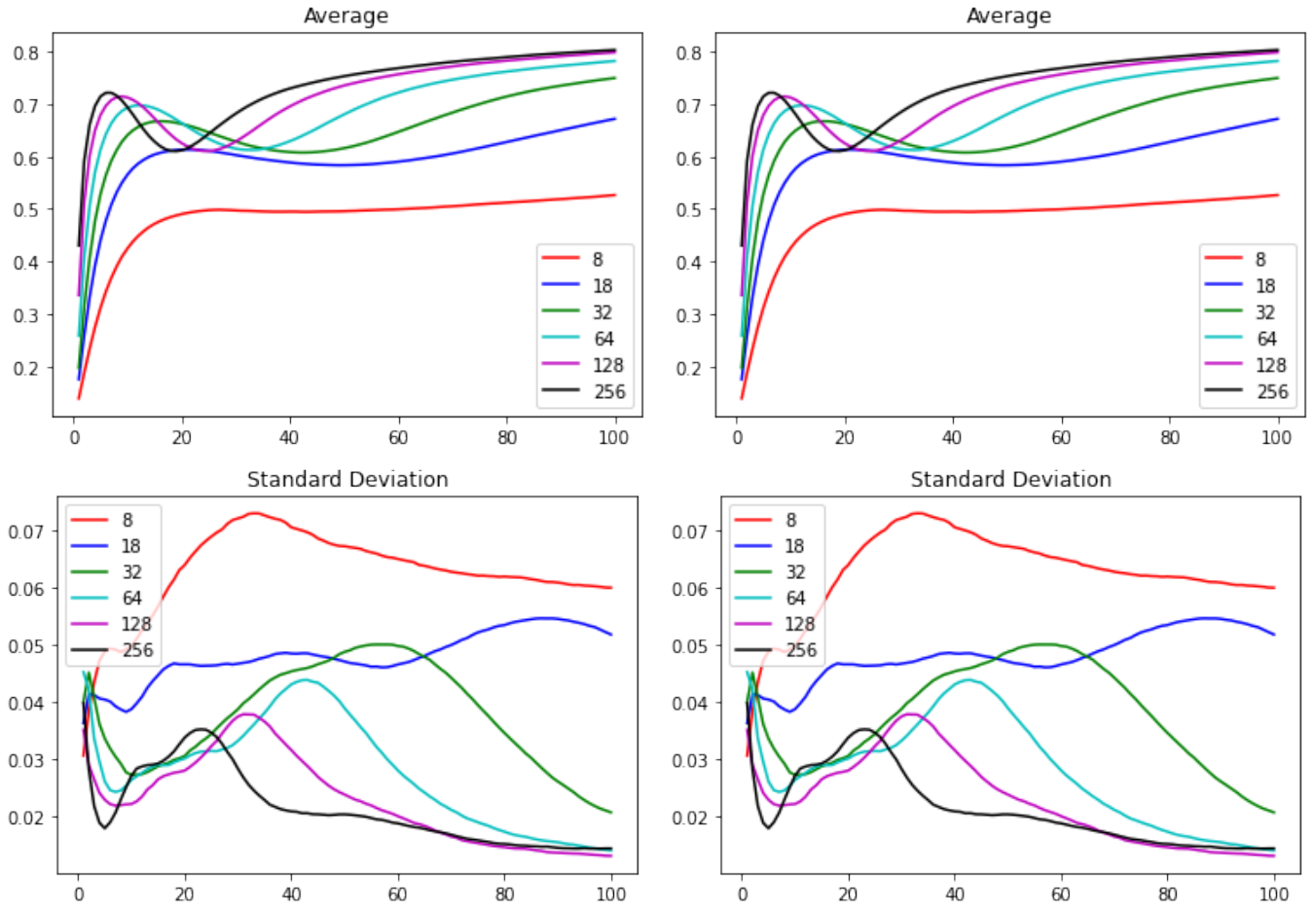
1) ReLU activation: We changed the number of neurons in the hidden layer of the neural network. For this set of experiments we used a ReLU activation function for the hidden layer. The results show that for the MNIST dataset, the largest number of neurons that we tried, 256, performed the best. In fact, performance increased with increasing number of neurons. The averages were taken over 100 different random initializations of the neural net parameters. In addition, we can see from the standard deviations that the performance is also more stable with respect to random initializations as the number of neurons increases. The data was also normalized so that each dimension of the input vectors had mean 0 and variance 1. The results can be seen in Fig. 1.

2) Sigmoid activation: Again, we changed the number of neurons in the hidden layer of the neural network, this time using a sigmoid activation function for the hidden layer rather than ReLU. The same trends hold in the sigmoid activation case: networks with more neurons in the hidden layer demonstrate better performance and are more stable with respect to random initialization. In addition, the average accuracies for all sigmoid networks were monotonically increasing, unlike those of the ReLU networks which had local minimums and maximums. From all our experiments, the results from computing test accuracies and training accuracies were extremely similar. This goes to show how representative the training data was of the test data. The results can be seen in Fig. 2.

B. Variation of number of hidden layers

1) ReLU activation: For this experiment we kept the number of neurons in the hidden layer(s) constant (at 128 neurons) but varied the number of hidden layers. We began with ReLU activations for all layers. What we found was that in both test and training accuracy, the model with two hidden layers very

Fig. 1. On the left are the average and standard deviations for the training accuracies for ReLU neural networks with different amounts of neurons in the hidden layer, on the right is the same metrics but for the test accuracies



slightly outperformed the models with one and three hidden layers. The standard deviations for the accuracies seemed to increase with the number of layers. The results can be seen in Fig. 3.

2) Sigmoid activation: For these experiments, all layers used sigmoid activation functions. The results are similar, with the two hidden layer model outperforming the other two models, however, in this case the difference in model performance was far clearer and the increase in performance seems to have leveled by 300 epochs. In the ReLU case the accuracies seemed to still be on the rise at the end of the scope of our testing. The results can be seen in Fig. 4.

IV. Summary

We demonstrated some of the effects of changing number of neurons in a single hidden layer, changing the number of hidden layers with constant amount of neurons in each, and changing activation functions in the hidden layers. We found that, at least up to 256 neurons in the hidden layer, performance seemed to scale with number of neurons and that sigmoid activation outperformed ReLU activation for the single hidden layer model. In the case of both ReLU and sigmoid activations, models with two hidden layers (128 neurons each) seemed to outperform models with one or three hidden layers.

Fig. 2. On the left are the average and standard deviations for the training accuracies for sigmoid neural networks with different amounts of neurons in the hidden layer, on the right is the same metrics but for the test accuracies

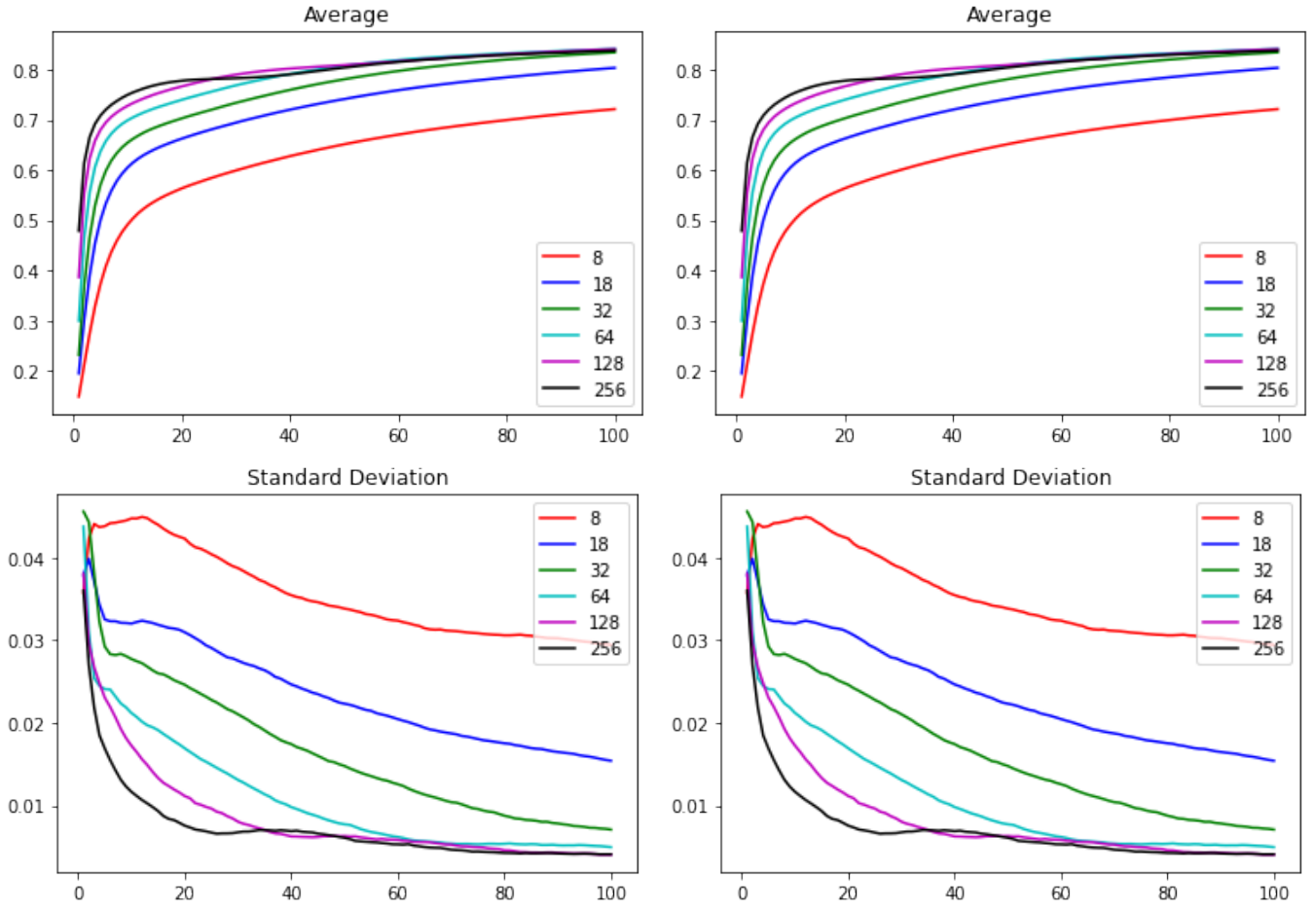


Fig. 3. On the left are the average and standard deviations for the training accuracies for ReLU neural networks with different numbers of hidden layers, on the right is the same metrics but for the test accuracies

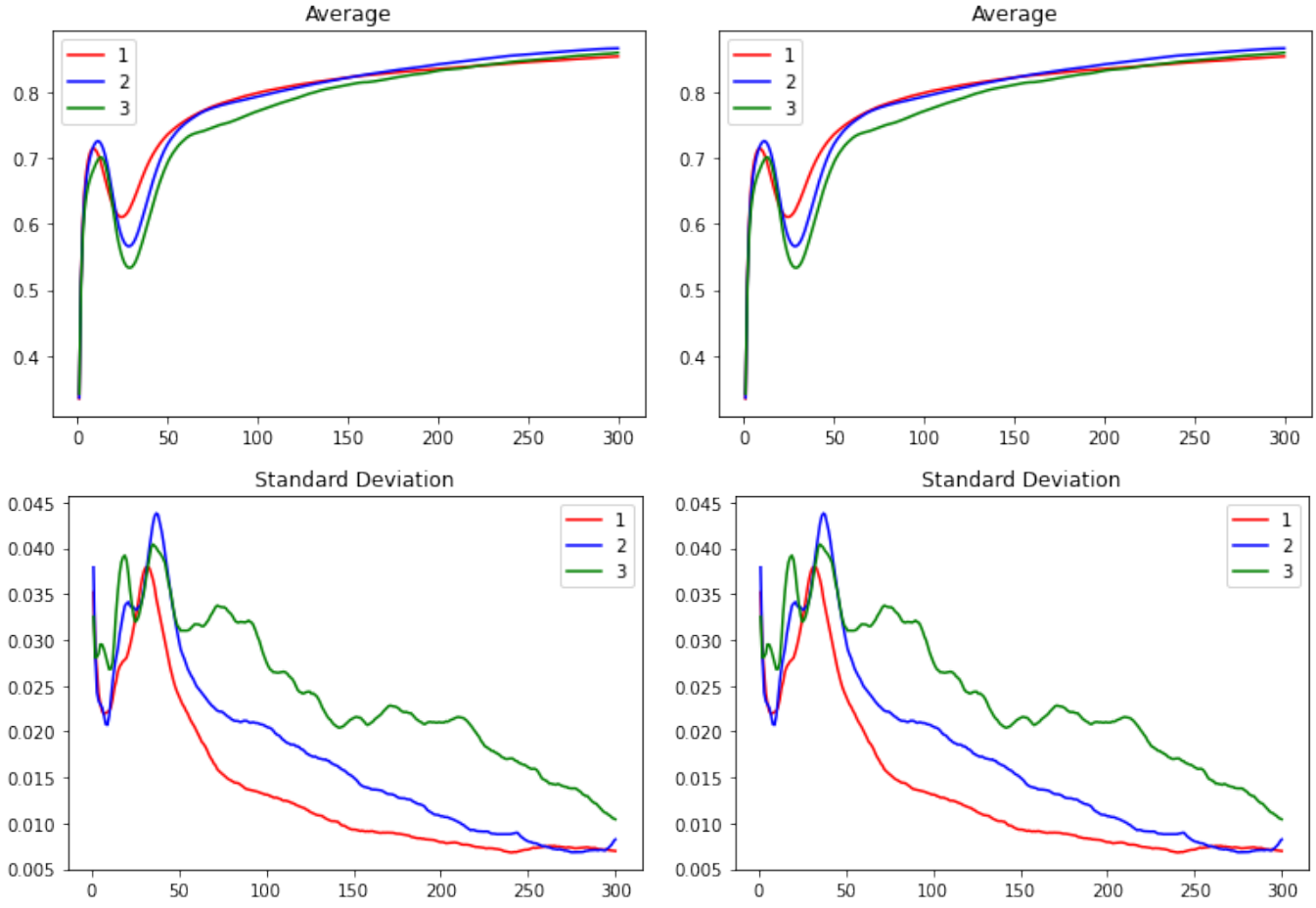


Fig. 4. On the left are the average and standard deviations for the training accuracies for sigmoid neural networks with different numbers of hidden layers, on the right is the same metrics but for the test accuracies