CMSC 25400 Homework 5

Ruolin Zheng

March 1, 2019

The submission zip file include:
weights.npy - a numpy array of weights corresponding to each layer of the Neural Net
neural_net.py - the Neural Net class and helper functions
driver.py - driver code for performance study and plots
TestDigitYPred.txt - the prediction result on *TestDigitX.csv*
TestDigitY2Pred.txt - the prediction result on *TestDigitX2.csv*
hw5.pdf

1 Program Description

1.1 Input, Output

The input dataset consists of 50000 images each shaped as a vector of length 784. The output predictions are integer values in the range 0 to 9, corresponding to each digit.

1.2 Data Structure

Neural Net is a class which is associated with the following adjustable parameters: a learning rate η, a batch size b, total number of layers (input + hidden + output) L, number of neurons per hidden layer N, and number of epochs. **NeuralNet.weights** is a list which stores the weight numpy array for each layer. The weights from layer l to layer $l+1$ is stored in the **NeuralNet.weights[l+1]** and thus the first entry is always $None$. We also add a bias unit to the input layer which then becomes a vector of length 785.
For example, on a Neural Net with 2 hidden layers and N_l neurons each, the dimension of each weight array is as follows,
NeuralNet.weights = [None, N_l * 785, N_l * N_l, N_l * N_l, 10 * N_l].

1.3 Training

I reserved $\frac{1}{5}$ of the training data as a holdout set for validation during training, and trained the remaining 40000 data in batches. The training algorithm is as follows:

```
for each epoch do
    shuffle the dataset;
    for each batch in the dataset do
        params = forward(data);
        computeLoss(params);
        avgGradients = backward(params);
        updateWeights(avgGradients);
    end
end
```

For class and function definition, please also refer to the docstrings in *neural_net.py.*

1.3.1 Randomization and Initialization

The weights are drawn from a Gaussian distribution. I also shuffled the dataset at the beginning of each epoch such that the Neural Net would not see the same batches every time.

1.3.2 Forward Propagation

NeuralNet.forward() computes the a and z values on a given batch and returns them in the form of lists $alist, zlist$. $alist[0]$ is initialized as the input vector plus a bias unit of 1 for each datapoint in the batch. Subsequently for each layer l, the corresponding entries are computed as

$$\begin{aligned} zlist[l] &= np.dot(alist[l-1], self.weights[l].T) \\ alist[l] &= sigmoid(zlist[l]) \end{aligned}$$

Activation for the output layer, $a[L]$, is computed with **softmax()** instead of **sigmoid()** which gives

$$alist[-1] = softmax(zlist[-1])$$

For a Neural Net with 2 hidden layers, N_l neurons each, setting the batch size to b, the list of a and z's is shaped as follows,
$alist = [b * 785, b * N_l, b * N_l, b * 10], zlist = [None, b * N_l, b * N_l, b * 10]$.

1.3.3 Computing the Loss

The loss is computed from the activations of the last layer, or, equivalently, we refer to them as **y probabilities**, $yprobs = alist[-1]$.

$$loss = np.sum(-np.log10(yprobs[np.arange(yprobs.shape[0]), batch\_ys]))$$

1.3.4 Backward Propagation

We initializes the list of *delta* by computing the derivatives of **softmax**, the result of the derivation for the loss at the i-th example L_i with respect to the activation of the activation of the k-th neuron in the output layer a_k is as follows,

$$\frac{\partial L_i}{\partial a_k} = \begin{cases} p_k - 1 \text{ if } k = y_i \\ p_k \text{ if } k \neq y_i \end{cases} \quad \text{where } p_k \text{ is the } \textbf{predicted y probability}$$

In code, we simply subtract 1 from the predicted probability that corresponds to the true y label.

$$dlist[-1][np.arange(batch\_ys.shape[0]), batch\_ys] - = 1$$

The gradients for the last layer is thus

$$gradients[-1] = np.dot(dlist[-1].T, alist[-2])$$

We then compute δ and the gradients for each layer l except for the input layer, where the derivative of sigmoid, **d_sigmoid(z)** is defined as **sigmoid(z) * (1 - sigmoid(z))**.

$$\begin{aligned} dlist[l] &= np.dot(dlist[l+1], self.weights[l+1]) * d\_sigmoid(zlist[l]) \\ gradients[l] &= np.dot(dlist[l].T, alist[l-1]) \end{aligned}$$

Finally, we average the gradients over the batch size and return $\frac{\nabla}{b}$.

1.3.5 Updating the Weights

We update the weights as in Gradient Descent, subtracting the product of the learning rate and averaged gradient from our current weights. This ensures that the weights are decreasing along the direction of greatest magnitude.

$$weights - = learning\_rate * gradients$$

1.4 Validating

At the end of each epoch, we validate our Neural Net against the holdout set by calculating the number of wrong predictions it makes over the size of the holdout set. This corresponds to the function **NeuralNet.validate()**. We breaks out of the loop if the error of current iteration is more than 0.02 above that of the previous, so as to prevent overfitting.

1.5 Testing

With a trained Neural Net, we checks against the test set data and labels *TestDigitX.csv, TestDigitY.csv* with **NeuralNet.validate()**. We then report the error rate.

1.6 Predicting

NeuralNet.predict() takes a dataset and runs **NeuralNet.forward()** to obtain the activation values at the last layer. Each activation value is shaped as a vector of length 10, containing the predicted y probability corresponding to each digit from 0 to 9. The single-digit prediction value is thus given by the entry with the highest probability,

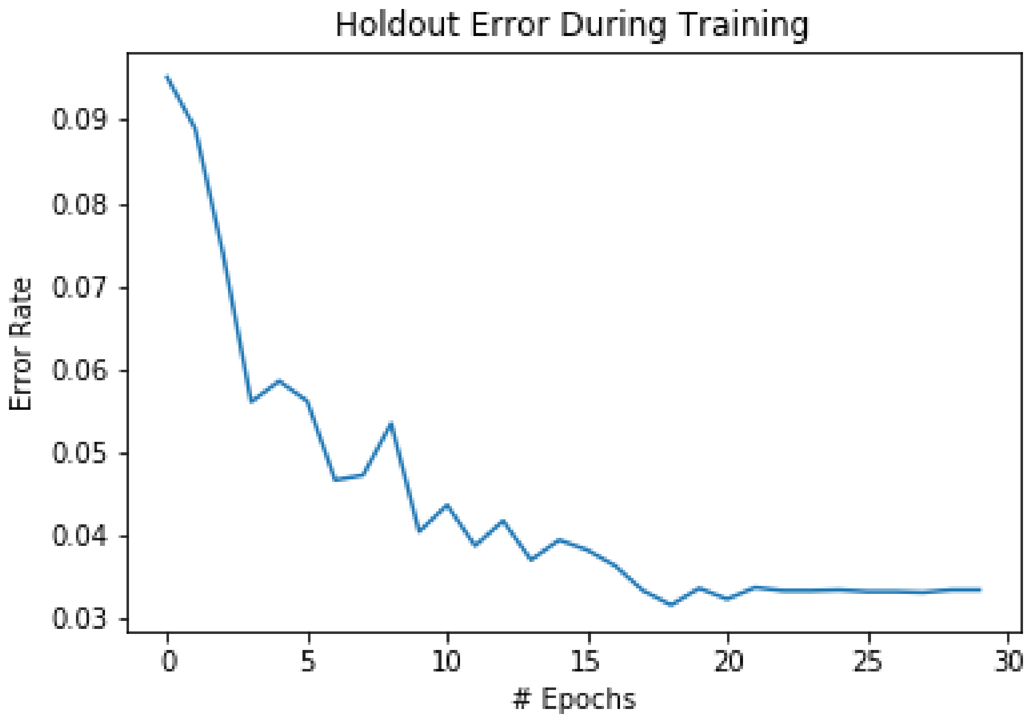
$$yhats = np.argmax(yprobs, axis = 1)$$

2 Performance Study

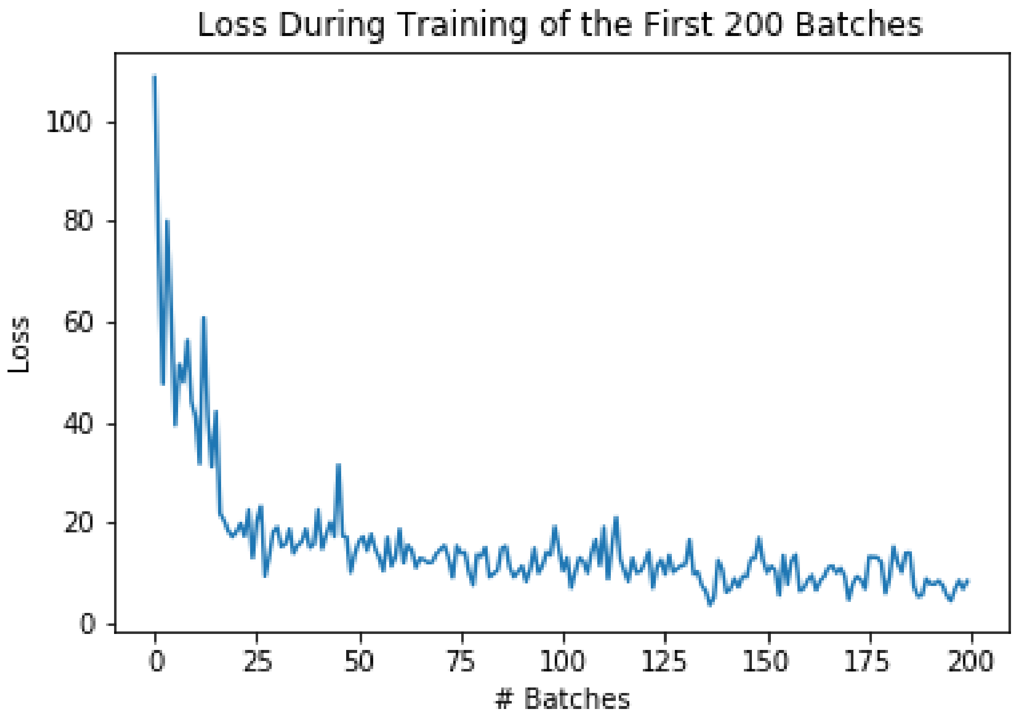
After experimenting with a range of learning rate, batch size, number of hidden layers, number of neurons per hidden layer and number of epochs, the following combination of parameters produced the lowest testing error of **0.0279**.
Neural Net Parameters: learning rate 0.5, batch size 4, 3 layers (1 hidden), 256 hidden units, 20 epochs.
(I chose a batch size as a multiple of 4 to make optimal use of memory cache.) Justification for the choices of parameters is as follows.

2.1 Holdout Error and Loss During Training

Neural Net Parameters: learning rate 0.8, batch size 16, 5 layers (3 hidden), 400 neurons per hidden layer, and 30 epochs.
Below is a plot of holdout error as a function of number of epochs. As we see from the plot, the error rate decreases sharply during the first five epochs and then gradually over the course of the total 30 epochs. The curve of decrease is not very smooth and is an expected behavoir of minibatch gradient descent. Although there is a slight increase in holdout error starting from epoch 18, it is unlikely due to overfitting since the increase is very small. As shown in the plot, the curve smooths out and the error rate remains mostly constant from epoch 21 onwards, and thus we stop at epoch 30.



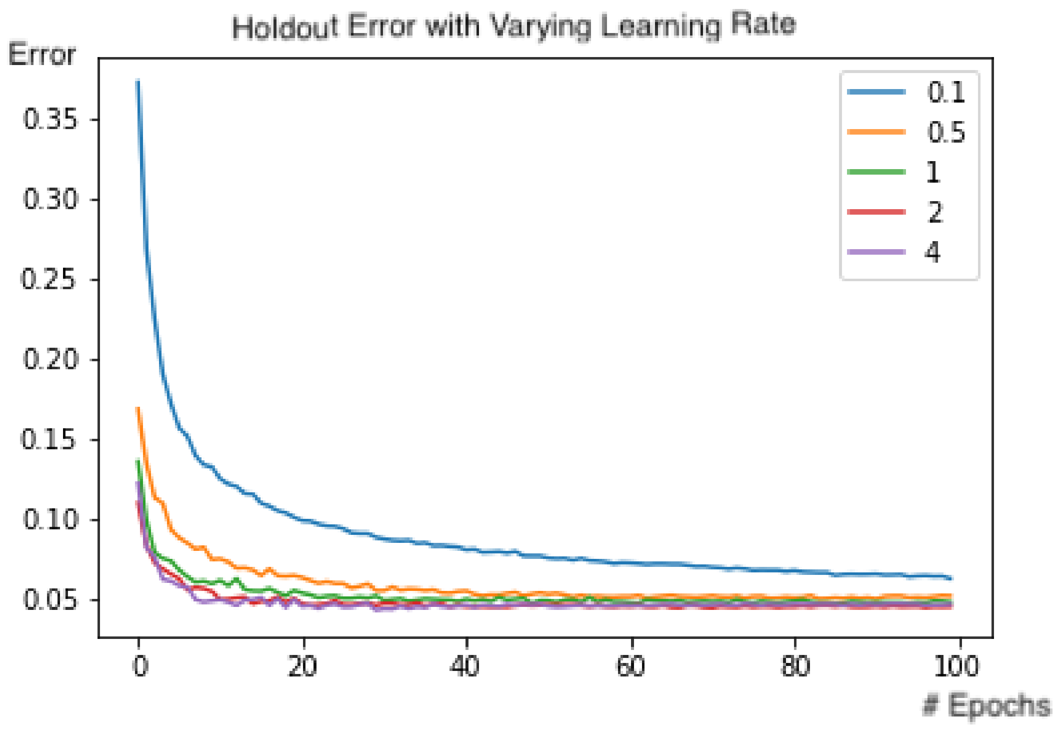
Below is a plot of the softmax loss over the first 200 batches. (In each epoch, the dataset is divided into $\frac{40000}{16} = 2500$ batches.) The softmax loss is calculated for each batch by comparing the true label with the result from the output layer. The softmax loss is positively correlated with the holdout error, and we observe a similar jagged, decreasing curve.

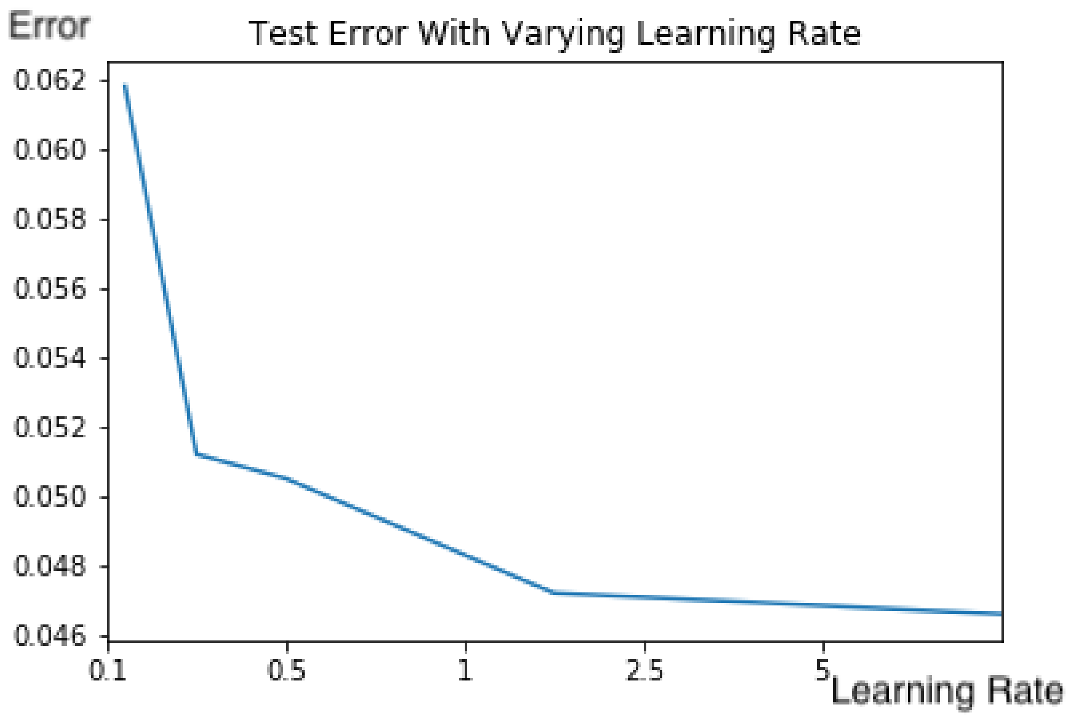


2.2 Learning Rate

Neural Net Parameters: varying learning rate, batch size 100, 4 layers (2 hidden), 128 neurons per hidden layer, and 100 epochs.
As shown in the table and figures below, a higher learning rate results in a lower testing error. This makes sense as the batch size is relatively large and we may benefit from more aggressive gradient descent. However, a thing to note is that the performance does not change too much for learning rate > 0.5, and we might want to choose a learning rate closer to 1 to avoid missing the local minimum.

Learning rate	Testing Error
0.1	0.0618
0.5	0.0512
1	0.0505
2	0.0472
4	0.0466

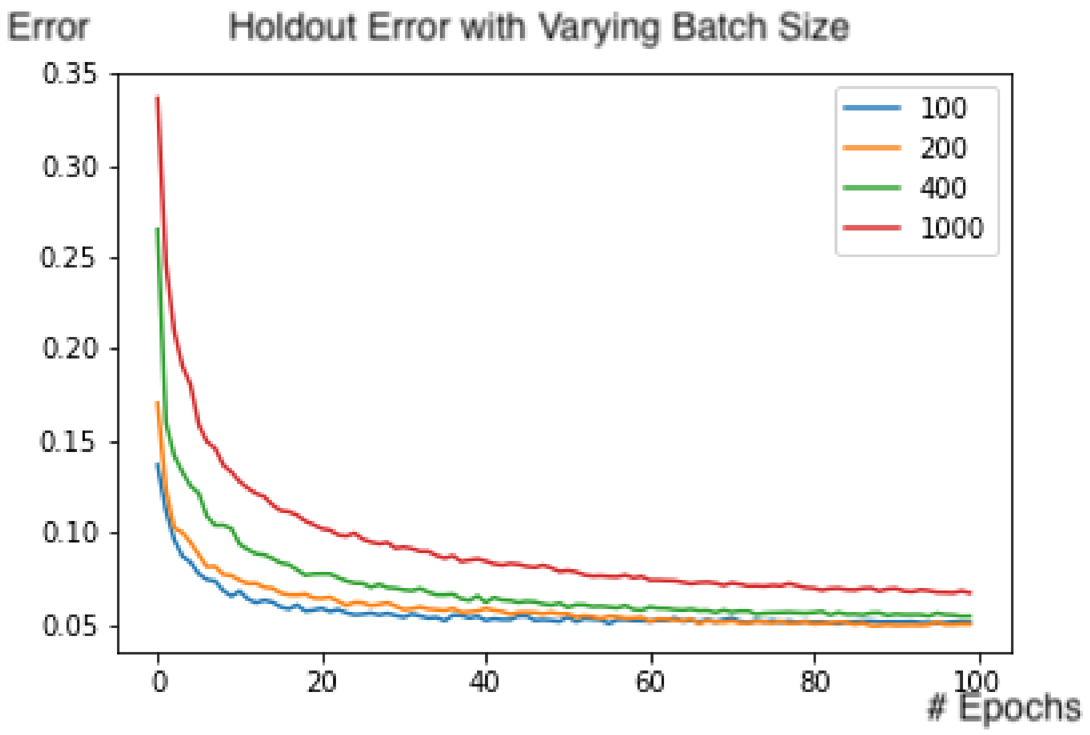

Holdout Error with Varying Learning Rate
Error
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100
# Epochs
0.1
0.5
1
2
4

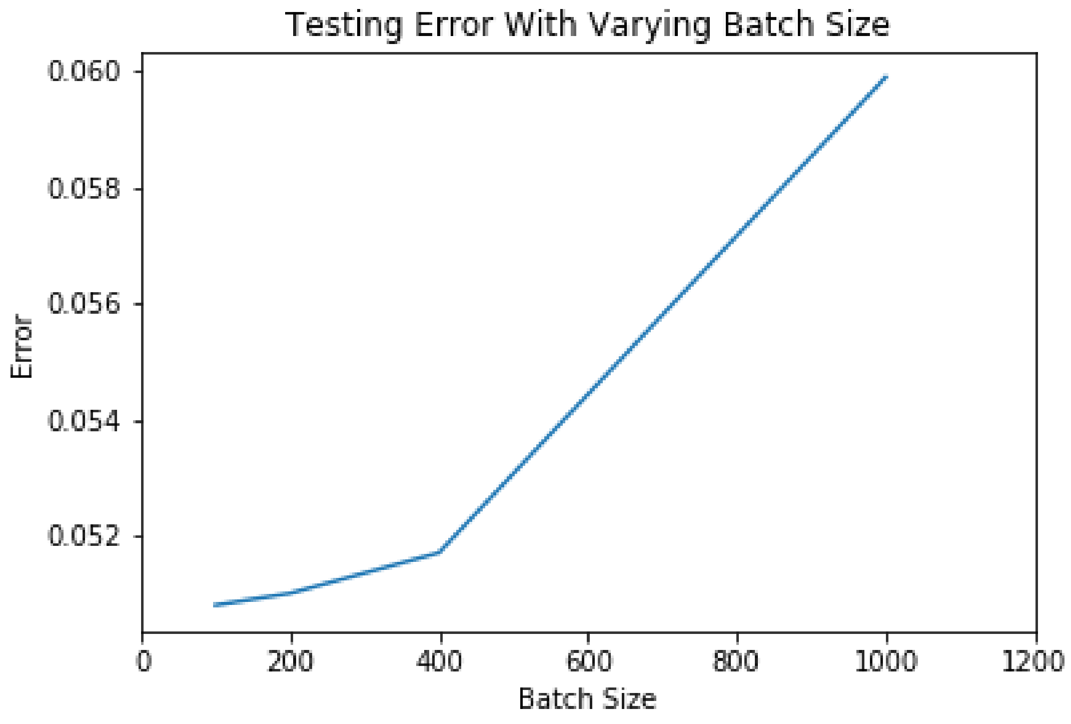

Test Error With Varying Learning Rate
Error
0.062
0.060
0.058
0.056
0.054
0.052
0.050
0.048
0.046
0.1
0.5
1
2.5
5
Learning Rate

2.3 Batch Size

Neural Net Parameters: learning rate 1, varying batch size, 4 layers (2 hidden), 128 neurons per hidden layer, and 100 epochs.
As shown in the table and figures below, a smaller batch size results in a lower testing error. This is reasonable since we may be missing out important updates on specific datapoints when we average the gradients over the batch size.

Batch Size	Testing Error
100	0.0508
200	0.051
400	0.0517
1000	0.0599

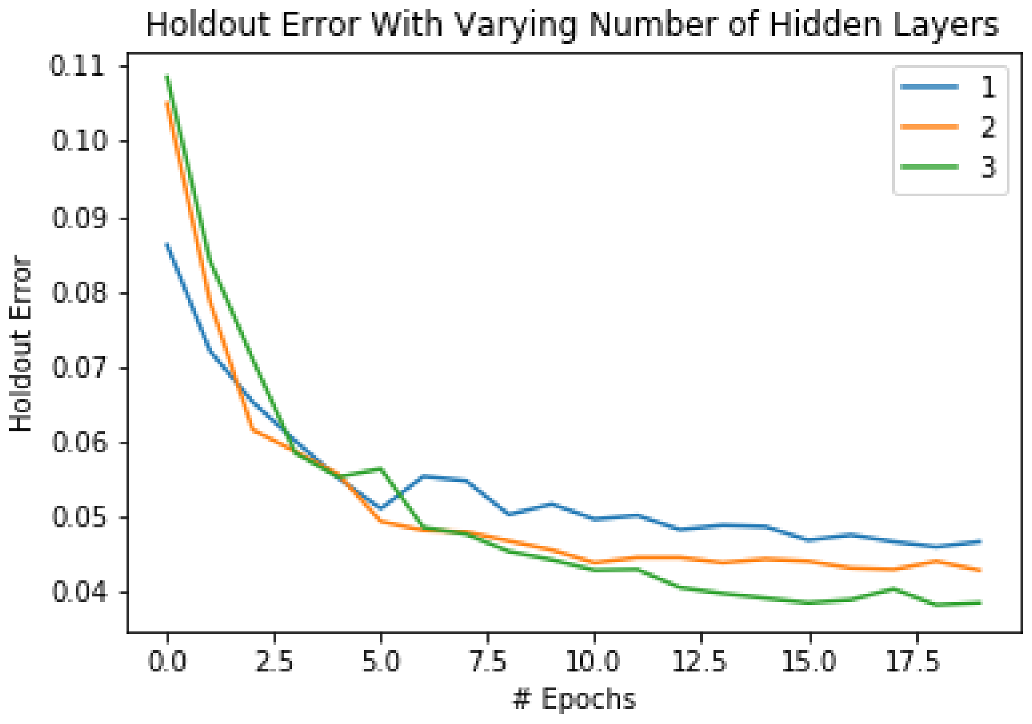


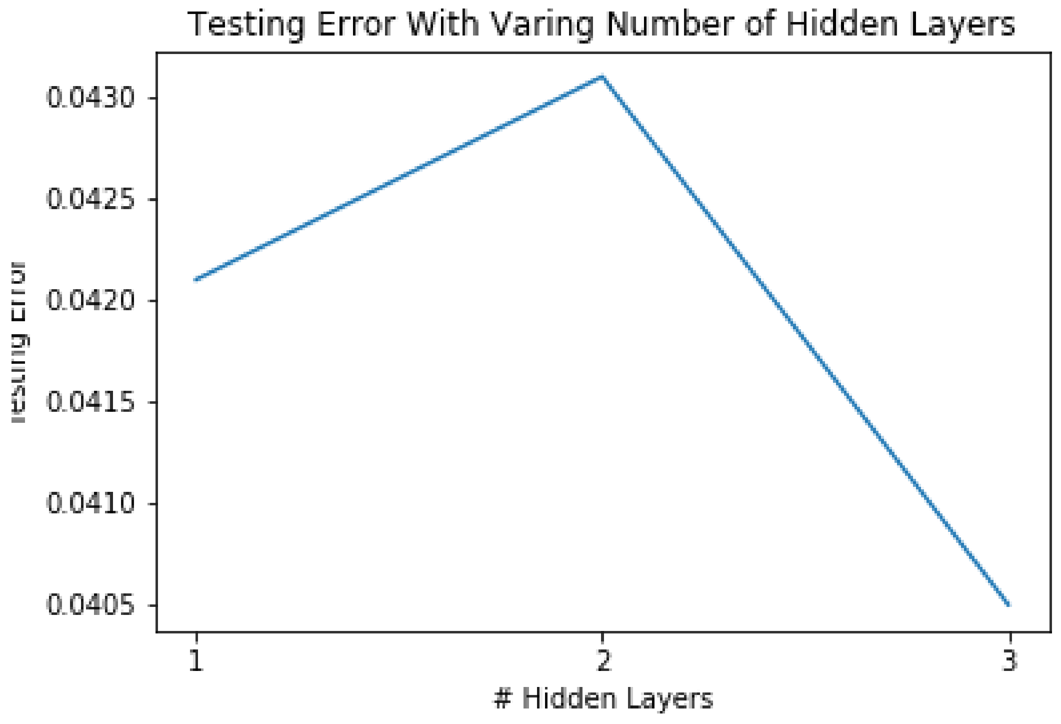


2.4 Number of Layers

Neural Net Parameters: learning rate 0.5, batch size 16, varying number of layers, 300 neurons per hidden layer, and 20 epochs.
Our random initialization of the weights from a Gaussian distribution may determine the local minimum we converge to. This may help explain the somewhat irregular trend we observe as the number of layer increases. As shown in the table and figures below, a greater number of hidden layer generally results in a lower error rate; the scaling of the second plot may be a little misleading and in fact there is not that much of a difference between using the one or two hidden layers. So far, using three hidden layers gives the optimal performance. (However, as our inputs are only vectors of length 784, using too deep a Neural Net may easily results in overfitting.)

Number of Hidden Layers	**Testing Error**
1	0.0421
2	0.0431
3	0.0405


Holdout Error With Varying Number of Hidden Layers
Holdout Error
# Epochs
1
2
3

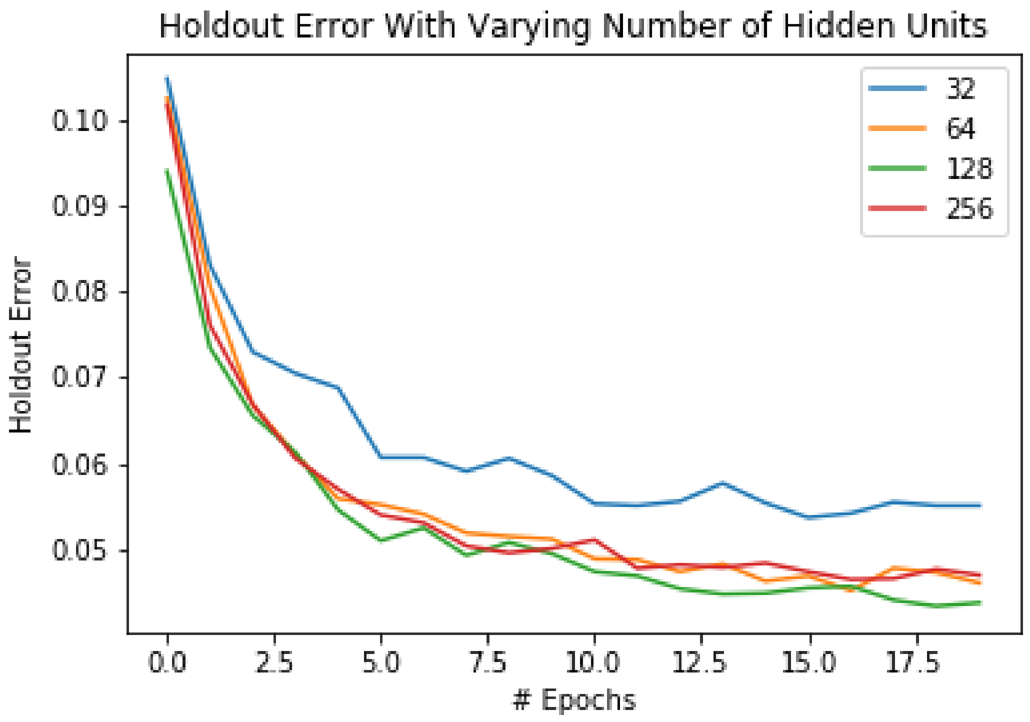

Testing Error With Varing Number of Hidden Layers
Testing Error
# Hidden Layers

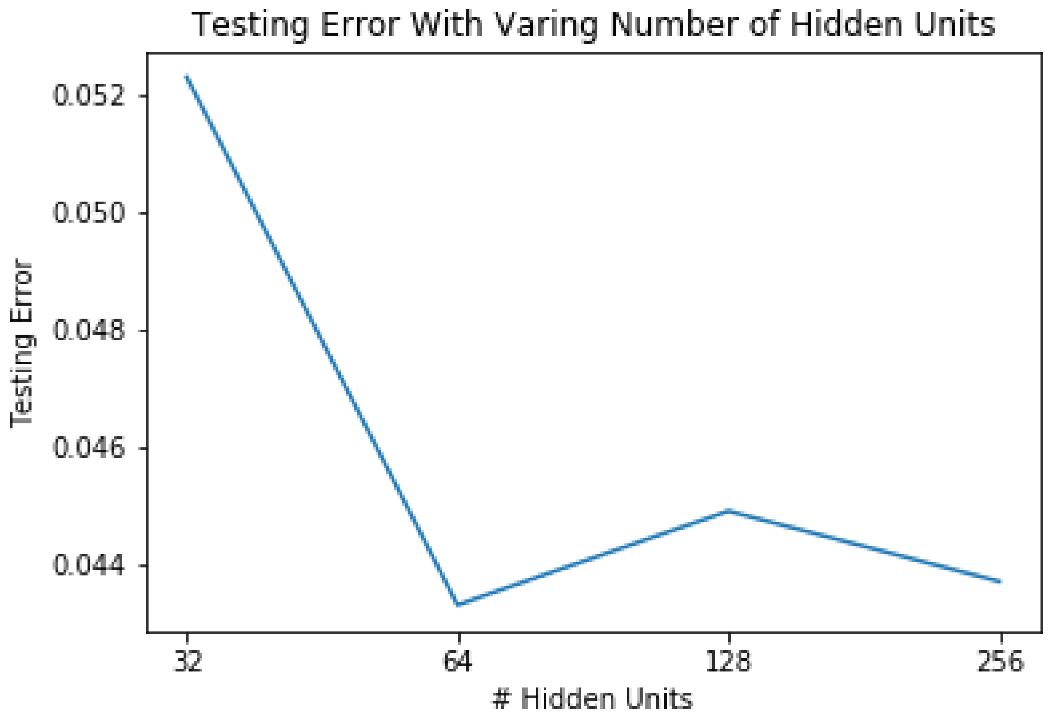
2.5 Number of Neurons per Hidden Layer

Neural Net Parameters: learning rate 0.5, batch size 16, 4 layers (2 hidden), varying number of hidden units, and 20 epochs.

Similar to the case of varying the number of hidden layers, our random initialization of the weights from a Gaussian distribution may determine the local minimum we converge to. This may also help explain the somewhat irregular trend we observe as the number of hidden units increases. As shown in the table and figures below, more hidden units generally result in a lower testing error. This is reasonable since more hidden units correspond to more features the Neural Net can detect, which helps increase the strength of prediction.

Number of Hidden Units	Testing Error
32	0.0523
64	0.0433
128	0.0449
256	0.0437

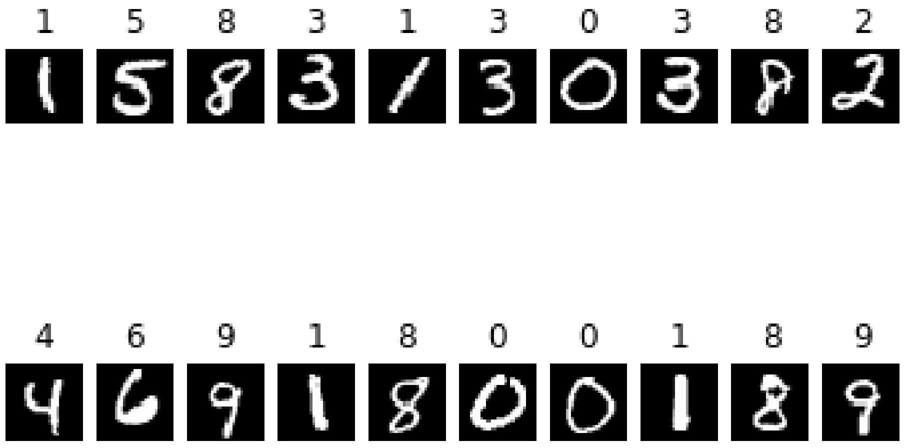




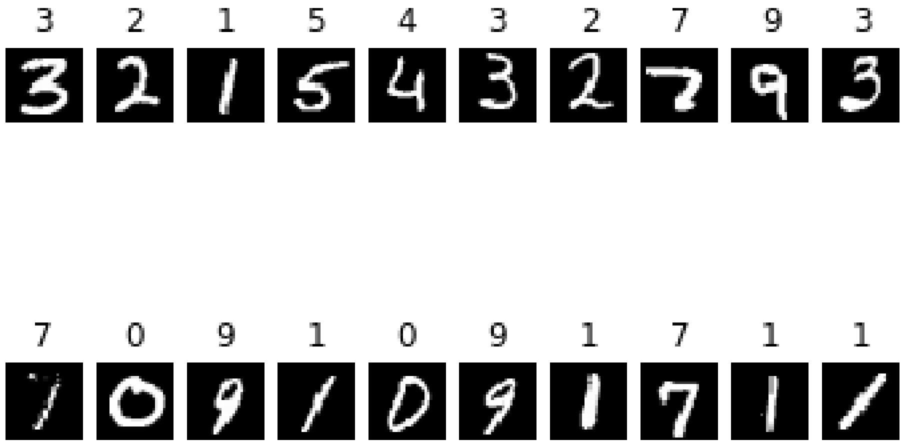
3 Comment

3.1 Result

The Neural Net I used to predict *TestDigitX.csv* and *TestDigitX2.csv* has the following parameters:
learning rate 0.5, batch size 4, 3 layers (1 hidden), 256 hidden units, 20 epochs.
The final testing error on *TestDigitX.csv* is **0.0279 = 2.79%**.
The figures below show the prediction results of 20 randomly drawn datapoints from *TestDigitX.csv* and *TestDigitX2.csv*. As the figure depicts, the result is pretty decent.



Predictions on 20 random draws from X



Predictions on 20 random draws from X2

3.2 Design and Vectorization of the Code

The **softmax()**, **sigmoid()**, and **d_sigmoid()** function has been vectorized to operate on any 2D matrices, which allows for efficient computation to be done for each batch of inputs shaped as $b * 785$. Because of the vectorization technique, the code runs efficiently even with very small batch size (about 300 seconds for the aforementioned configuration.)

3.3 Running the Program

Please run *driver.py* in Python 3.