April 6, 2018

ECE521: Assignment 3

Linear and Logistic Regression

Younghoo Cho (1000594270), Deng Pan (1000548874), Keon Young Park (1004686253)
Equal contribution from each member.

1 Neural Networks

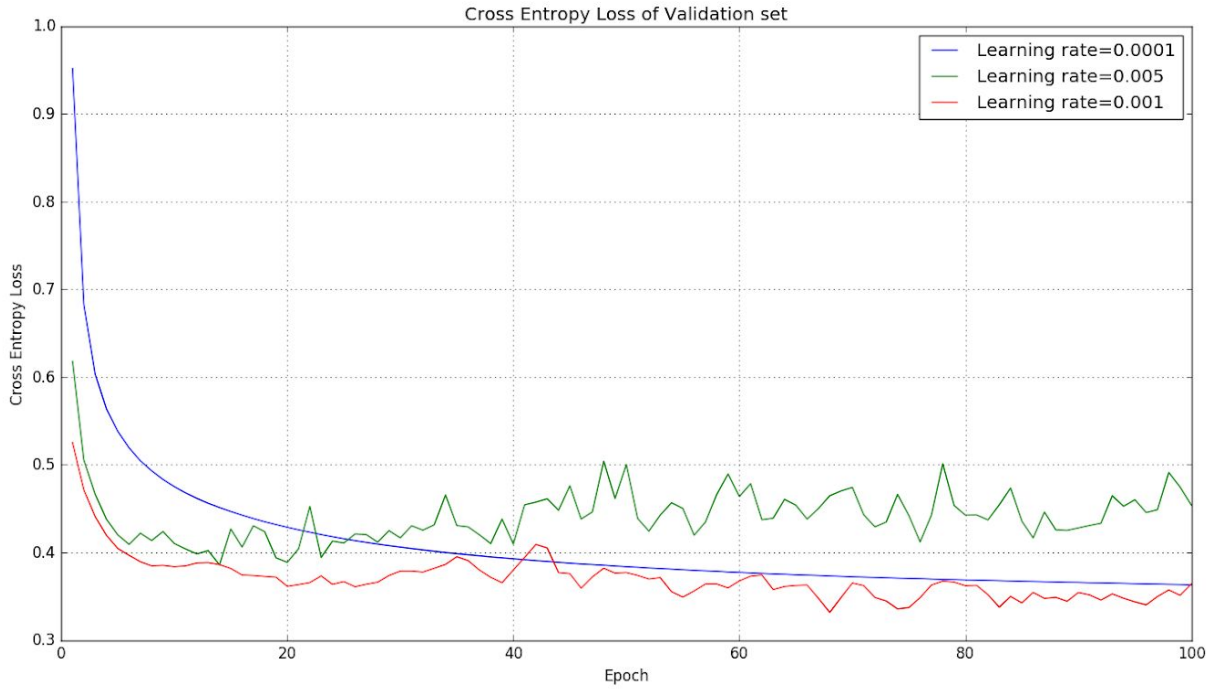
1.1 Feedforward fully connected neural networks

1.1.1 layer-wise building block

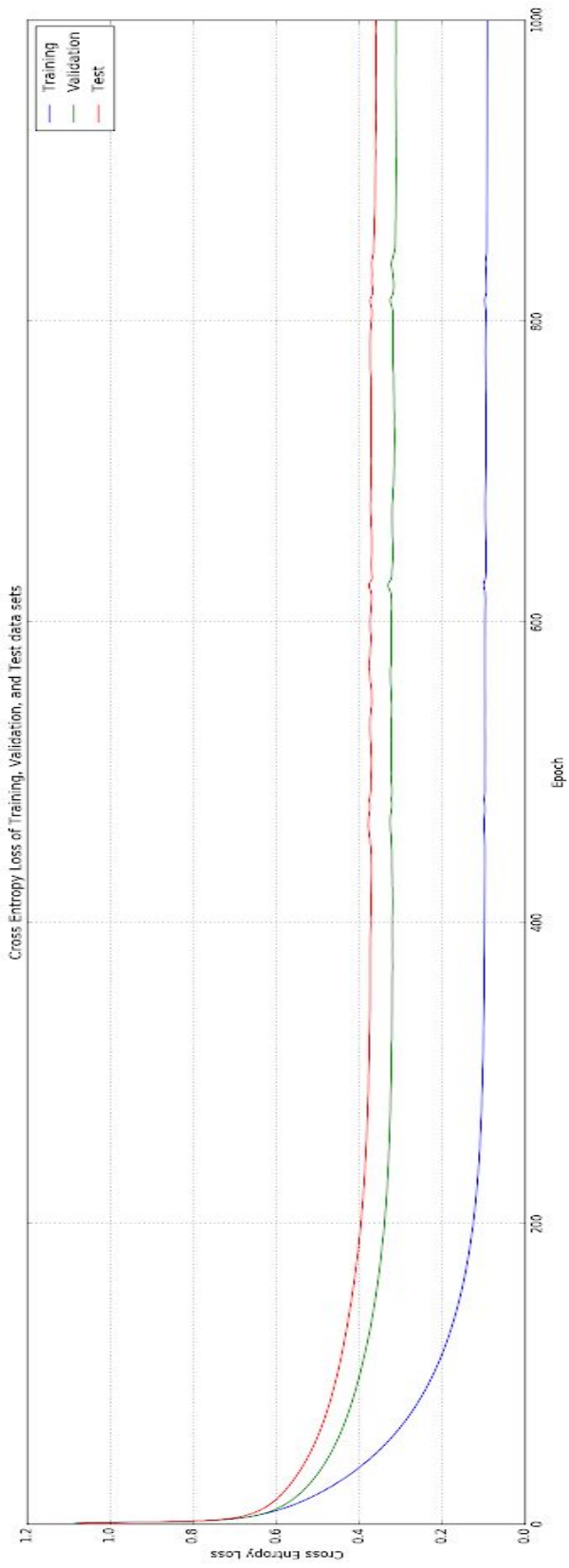
```
37 # Part 1: layer-wise building block
38 def calculate_s(data, num_hidden_units):
39     # data is N x d
40     # w is d x M, where d = dim(input vector), M = number of hidden units
41     w = tf.get_variable("weights", shape = [data.shape[1], num_hidden_units],
42                         initializer = tf.contrib.layers.xavier_initializer())
43     # b is 1 x M, where M = number of hidden units
44     b = tf.get_variable("bias", shape=[1, num_hidden_units],
45                         initializer = tf.zeros_initializer)
46     # s is N x M
47     s = tf.add(tf.matmul(data, w), b)
48     return s
```

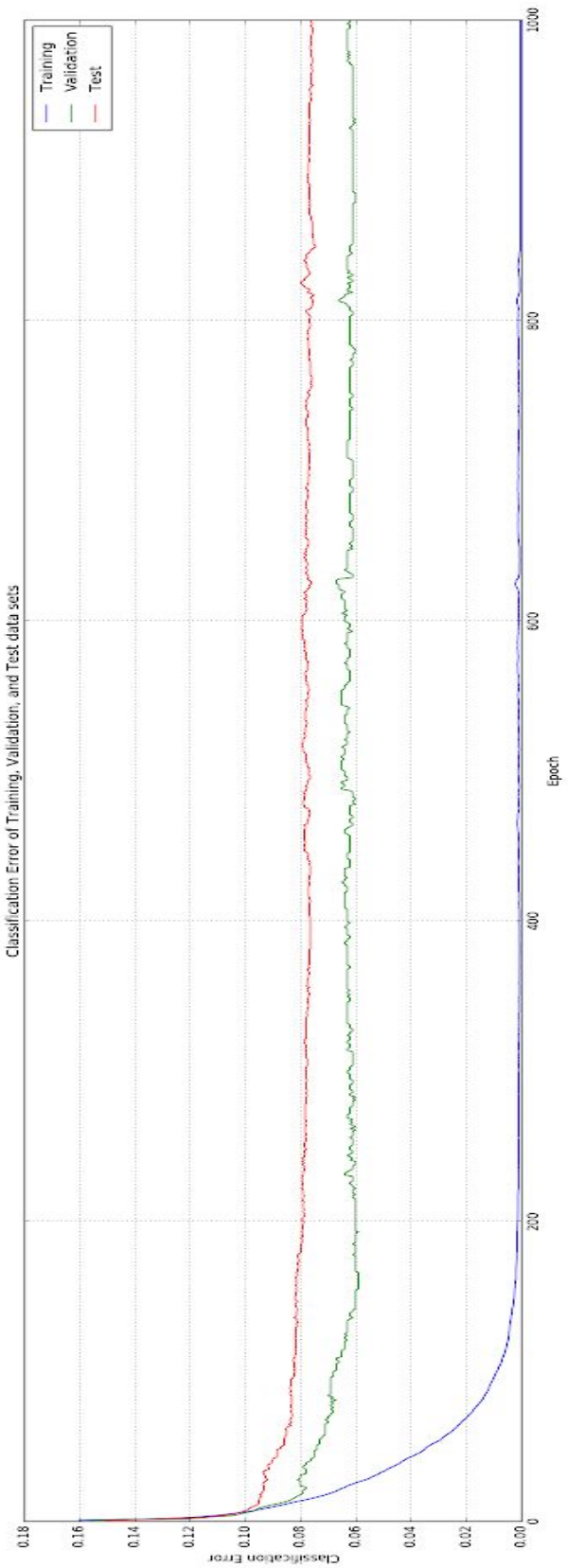
The parameter ‘data’ is an N x d matrix, where N is the number of input vectors, and d is the dimension of the vector (without bias). This function creates and initializes the weight and the bias. Note that the weight uses the Xavier initializer to initialize its weights. The output of this function is an N x M matrix, where N is the number of input vectors, and M is the number of units in the next layer (either hidden or output). Note that because this function takes in a N x d matrix, it is able to compute the weighted sum of the next layer for all input vectors in the given mini-batch.

1.1.2 Learning



The above is a graph showing the cross-entropy loss using different learning rates of 0.001, 0.005, 0.0001, which are the same selection of learning rates as A2. As can be seen above, the learning rate of 0.005 is definitely worse than the other two, as can be seen by the higher loss value as epoch is increased. When choosing between 0.001 and 0.0001, although 0.001 is lower for the earlier epochs, it is quite erratic, which signals that the learning rate is too high. A learning rate of 0.0001 however is quite smooth in its progress, as epochs increase. This smoothness is generally preferred as it is quite clear when a minimum has been reached, and when an overfitted situation occurs (if it occurs at all). Additionally, at epoch 100, they both reach the same loss. Therefore, 0.0001 is chosen as the best learning rate, and is used for all experiments hereafter.


Cross Entropy Loss of Training, Validation, and Test data sets
Training
Validation
Test
Cross Entropy Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
200
400
600
800
1000


Classification Error of Training, Validation, and Test data sets
Training
Validation
Test
Classification Error
Epoch
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0
200
400
600
800
1000

The above is a plot of cross entropy error and classification error as epochs are increased, for the training set, validation set, and test set. As can be seen above, the test cross entropy loss and classification error peters out around epoch 200. After epoch 200, we see minimal improvements and therefore questions if training the model for longer epochs than 200 is worth it, as you gain minimal accuracy improvements over significant increase in training time.

1.1.3 Early stopping

An epoch of 200 seems to be a good place to stop. From there, we see minimal improvements with significant increase in training time. It can be seen in the plot that near epoch 200, both classification error and cross-entropy loss seems to converge. Therefore, the early stopping point is the same for both plots. In general, cross-entropy loss should be used for determining early stopping. This is because there may be a case where you get high cross-entropy loss but low classification error. This occurs if the probability distribution (ie. the prediction from the neural network) has near uniform distribution (ie. the network doesn't really know which is the correct prediction) but still the maximum probability among that near uniform distribution still happens to align with the target label. For example, if the correct label is 3, and the predicted probability distribution is [20, 20, 21, 19, 20], the neural network would predict 3 correctly (since it has the highest probability of 21), but it isn't very sure. This would lower the classification error while increasing the loss. This kind of situation should be avoided. On the other hand, there cannot be the case where there is low loss but high classification error. Therefore, the cross-entropy loss should be used to determine the early stopping point, because a low loss directly translates to low classification error.

The following is the training, validation, and test classification error:

Data Set	Train	Validation	Test
Test Classification Error	0.000733316	0.062	0.0792952

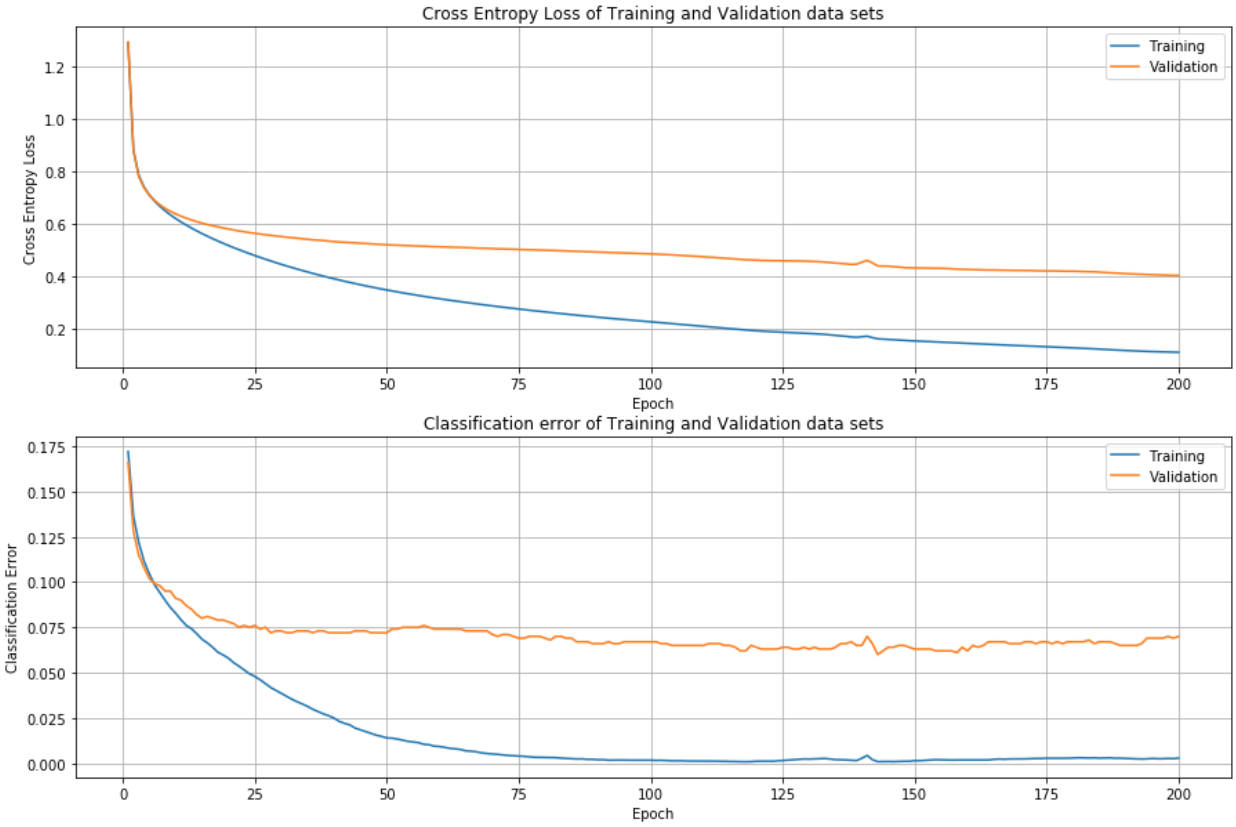
1.2 Effect of hyperparameters

1.2.1 Number of hidden units

No. of hidden units	100	500	1000
Best CE Loss of Valid. Set	0.31808728	0.33253926	0.31639642
Test Classification Error	0.08773863	0.08039647	0.07782674

Shown above is the best validation cross-entropy (CE) loss and the subsequent test classification error for 100, 500, and 1000 hidden units in the hidden layer of our neural network. **Summary**: Increasing the number of hidden units doesn't affect cross-entropy loss much, but it does decrease your test classification error.

1.2.2 Number of layers



Shown above is the training and validation cross-entropy loss and classification error for the 2-hidden-layer neural network.

- Final Validation set CE loss: 0.40248
- Final Validation set classification error: 0.07

The following is the final validation validation error and the validation classification error.

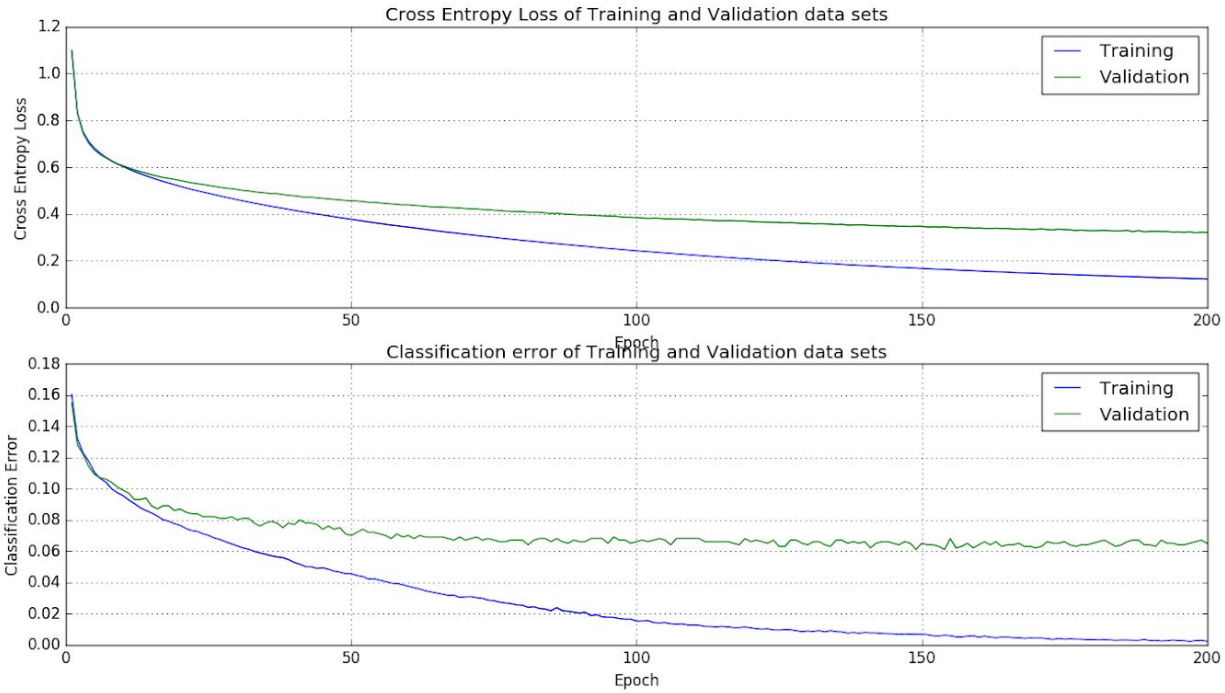
Architecture	1 layer with 1000 hidden units	2 layers with 500 hidden units each
CE Loss of Test Set	0.31639642	0.487554
Test Classification Error	0.07782674	0.0829662

As shown above, final Cross Entropy loss and Classification error for the test dataset suggest that the model with 1 hidden layer of 1000 units performs better than the double layered neural network. From the outcome, we learned that adding more layers into neural network generally gives better accuracy on training data set but not necessarily on the test data set, which suggests the overfitting of the model.

We also concluded that increasing complexity of neural network is likely to overfit and reduce performance on test data without proper size of train data. Therefore, increasing the number of hidden layers may not be the best option for enhancing the performance of your model.

1.3 Regularization and visualization

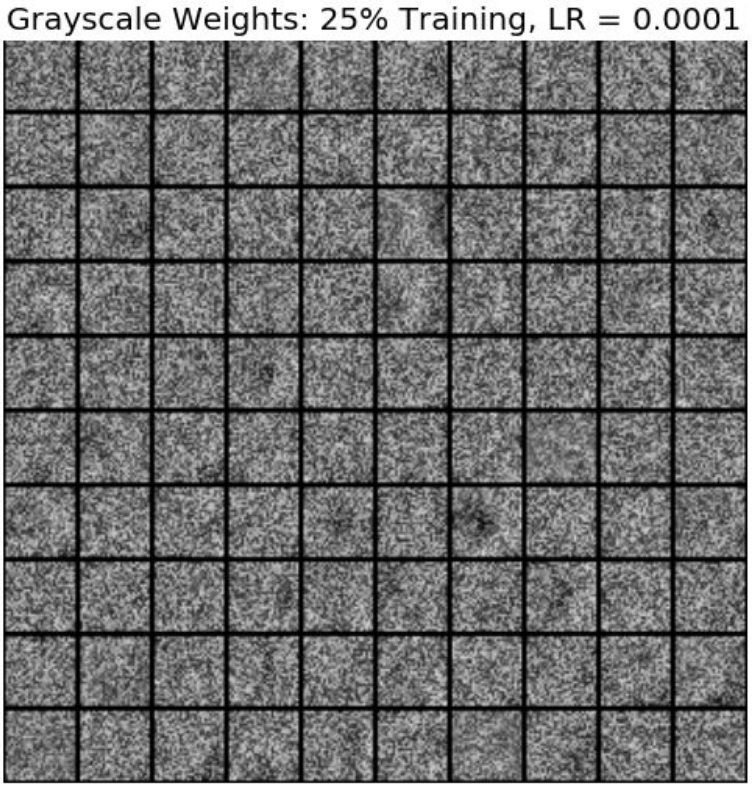
1.3.1 Dropout



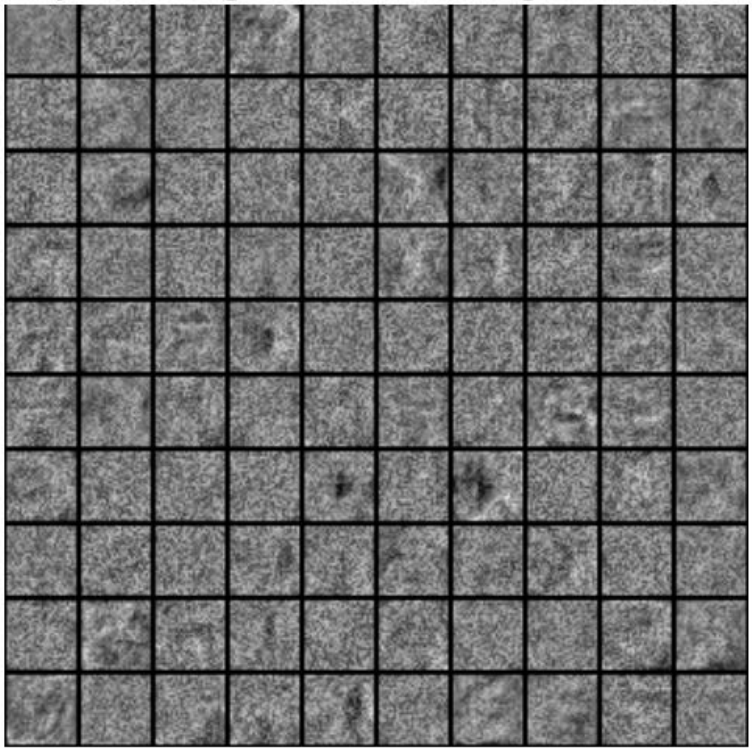
Validation loss at epoch 200: 0.32053983
Validation classification error at epoch 200: 0.065

Compared to the result in 1.1.2 and 1.1.3, the model without dropout has a slightly better validation classification error, although it is quite marginal. We learned in class that dropout does help with overfitting, but we see here that our results does not show this behaviour. Our hypothesis for this is that the 1-hidden layer neural network is too small/shallow to see the benefits of dropout.

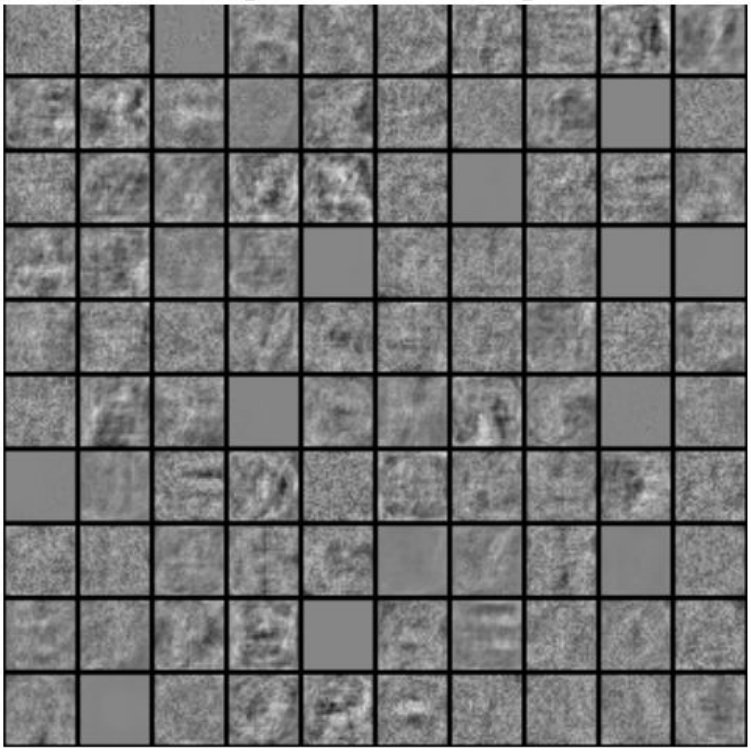
1.3.2 Visualization:



Grayscale Weights: 100% Training, LR = 0.0001



Grayscale Weights: 25% Training, LR = 0.001



Grayscale Weights: 100% Training, LR = 0.001

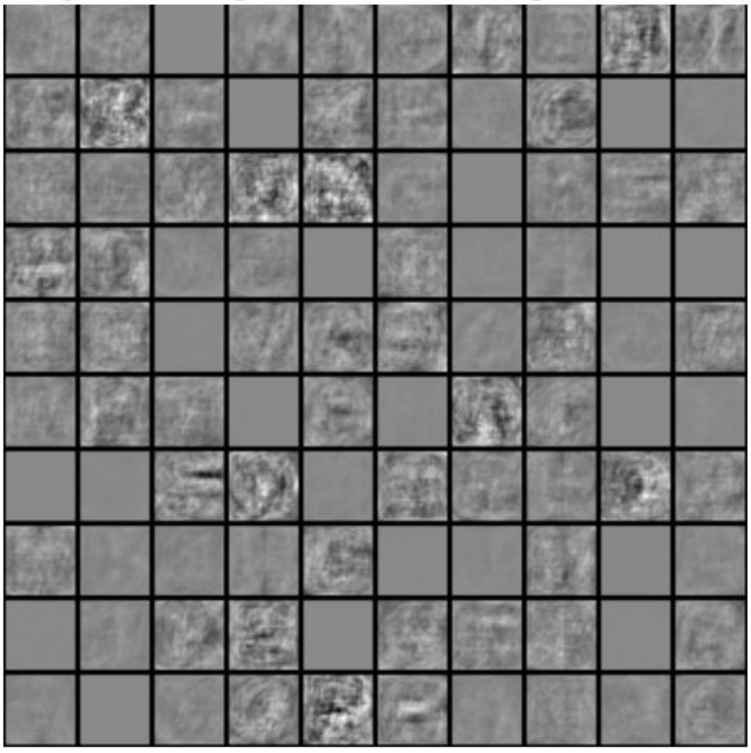


Image Function Description

Using config_image_h1(), the input weight matrix for hidden layer 1 will be reformatted such that the weights connected each hidden unit (in this case, 28x28) will be plotted in grayscale. The program does this for the first 100 hidden units to visualize how the weights images change as the progression of learning nears completion. Such images are then presented in a 10 x 10 grid and are trained accordingly for learning rates of 1e-4 over 100 epochs.

```
146 # Part 3: Resize the weight matrix into a grayscale image
147 def config_image_h1(wt):
148     # configuring the first 100 hidden units as images for display
149     wtMat_temp = wt[0:100,:]
150 
151     # find padding constant for each hidden unit image
152     pad_index = tf.argmin(tf.reshape(wtMat_temp, [100*784]), axis = 0)
153     pad_const = tf.reshape(wtMat_temp, [100*784])[pad_index]
154 
155     # adding in the padding: output is 100 x 30 x 30 tensor
156     wtMat = tf.reshape(wtMat_temp, [100, 28, 28])
157     wtMat = tf.pad(wtMat,[[0,0],[1,1],[1,1]],"CONSTANT",constant_values=pad_const)
158 
159     # reformat to acquire a 300 x 300 tensor for grayscale image
160     wtMat = tf.unstack(wtMat, num = 100, axis = 0, name = 'unstack')
161     wtMat = tf.concat(wtMat, 0)
162     wtMat = tf.split(wtMat, 10, 0)
163     wtMat = tf.concat(wtMat, 1)
164 
165     return wtMat
```

Grayscale Mapped Weights: Observations

Based on the grayscale weights, visual characteristics resembling that of alphabetic characters slowly emerges. Hidden units that doesn't possess this trend are seen to have its randomly initialized weights cleared to certain uniform values as witnessed for the learning rate = 1e-3 and 100% training case (ie. fully grey boxes). All graphs are trained for 100 epochs.

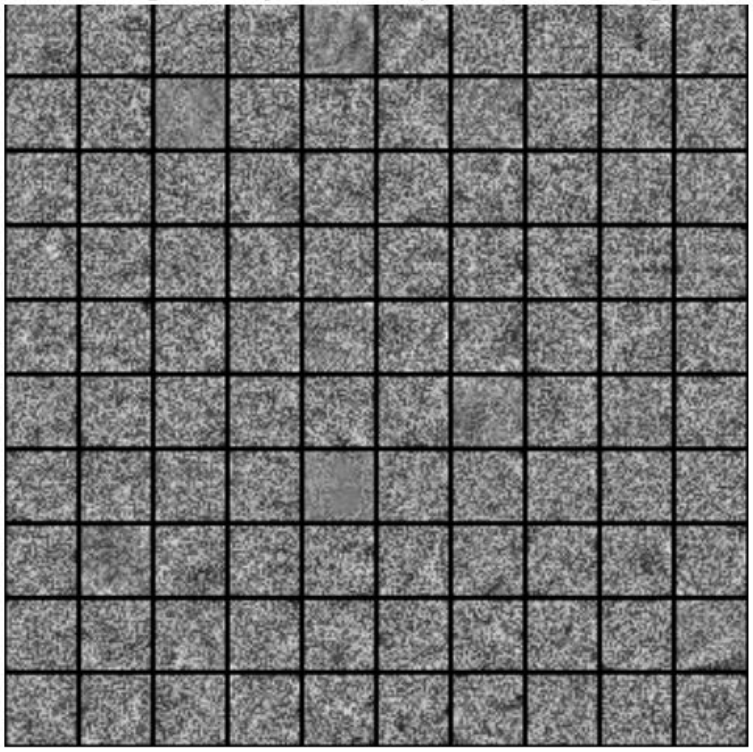
A possible conceptualization of what the neural network is performing is that the hidden units in the first hidden layer contains different configurations of letters being written down (e.g. the letter could be slanted, upside-down, bolded, or written in some others ways that differs from each other). These configurations are stored as weights connected to the units, which is then matrix-element-wise multiplied with the incoming image (28x28 in size) to compute the total sum that represents a degree of correlation between that image and those trained weights. As the weights become more trained, the configurations stored will be more realistic in its resemblance of the actual images in real life, as shown in the visualization figures. The output layers can then take these correlations to finally output the probability of each letter occurring through the ***quality and quantity*** of different letter configurations occuring.

As the number of hidden-layers increases, a similar line of reasoning may be applied. Thus, for neural networks with higher complexity, this process of correlating between different configurations and the input data image becomes more abstract.

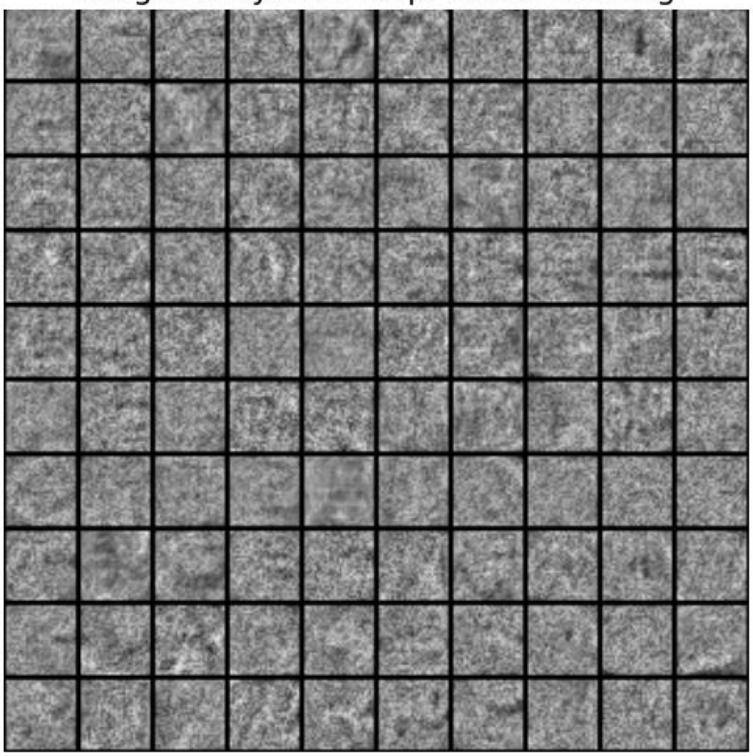
Using a dropout rate of 50% and learning rate of 1e-4, the model showed a slower learning behaviour than the one without dropout as the characteristic configurations appeared dimmer. These grayscale images are displayed below.

Note: Both images uses a dropout rate of 50%

Weight Grayscale Map: 25% Training



Weight Grayscale Map: 100% Training



1.4 Exhaustive search for the best set of hyperparameters

1.4.1 Random search

Model	Learning Rate	#hidden layers	#hidden units	Weight decay	Dropout?	Validation Classification Error	Test Classification Error
1	4.033e-4	2	277	4.295e-4	No	0.0580	0.0789
2	4.391e-4	2	129	3.455e-4	No	0.0750	0.0881
3	5.968e-3	1	468	5.459e-5	No	0.0800	0.0855
4	5.296e-4	5	476	1.157e-4	Yes	0.0720	0.0921
5	9.019e-4	2	346	1.829e-3	No	0.0720	0.0808

1.4.2 Exchange ideas among the groups

From Piazza, this is the best model so far, with the lowest test error of 6.94%:

Model	Learning Rate	#hidden layers	#hidden units	Weight decay	Dropout?	Validation Classification Error	Test Classification Error
1	1.200e-3	5	294	1.836e-4	Yes	0.0620	0.0694