ECE521 Assignment 2 Report

Submission Date:

27th Feb, 2017

Written by:

Winston Wong (1001614853)

(Source code and data for each sub-question is attached to email)

1.1.1.

```
def build_graph():


    #Hyperparameter
    regularization = 0.01

    t = tf.placeholder(tf.float32, [None,1], name ="label")
    X = tf.placeholder(tf.float32, [None,784], name ="x")


    W = tf. Variable(tf.truncated_normal(shape =[784,1], stddev = 0.5), [784,
    1], name = "weight")
    b = tf.Variable(tf.truncated_normal(shape =[1,], stddev = 0.5), name =
    "bias")
    z = tf.matmul(X,W) + b
    t_predicted = tf.sigmoid(z)

    loss_CE = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(z, t,
    name="loss_d"))
    loss_w = tf.reduce_sum(tf.square(W)) * regularization
    total_loss = loss_CE + loss_w

    optimizer = tf.train.GradientDescentOptimizer(learning_rate = 8E-4)
    train = optimizer.minimize(loss=total_loss)

    result = tf.cast(tf.greater(t_predicted, 0.5),tf.int32)
    accuracy = tf.reduce_mean(tf.cast(tf.equal(result, tf.cast(t,tf.int32)),
    tf.float32))

    return train, t, X, W, b, total_loss, loss_CE, accuracy
```

The graphs shown below in Figure 1 and Figure 2 demonstrates the relationship between training/testing accuracy vs number of epochs and cross-entropy loss vs number of epochs respectively. The max test accuracy is 0.986207.

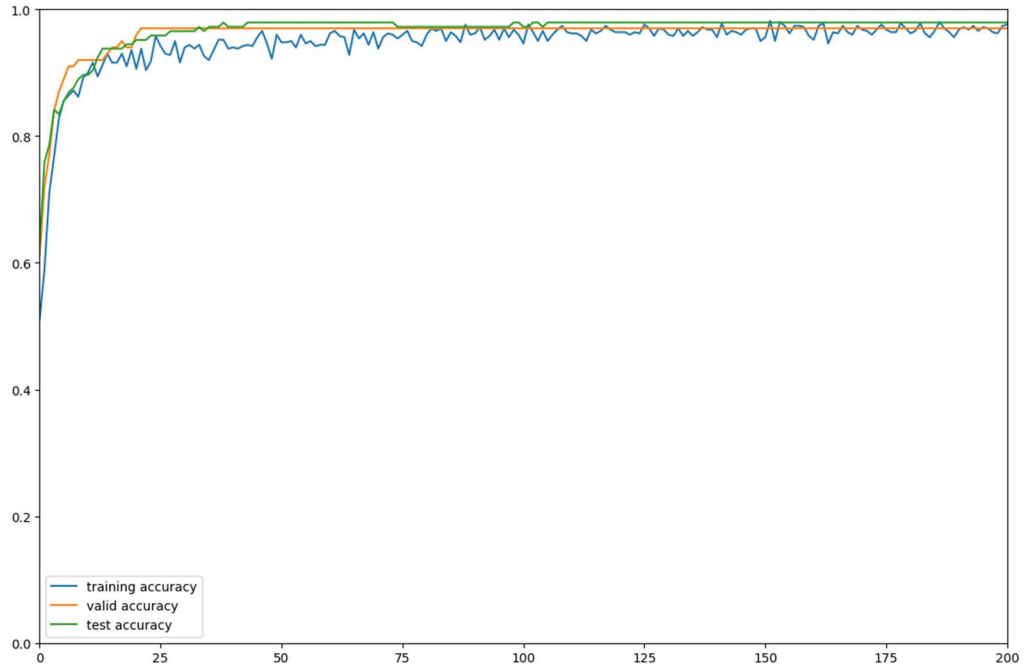


Figure 1 Accuracy vs Number of Epochs (Training, Cross-validation, Testing)

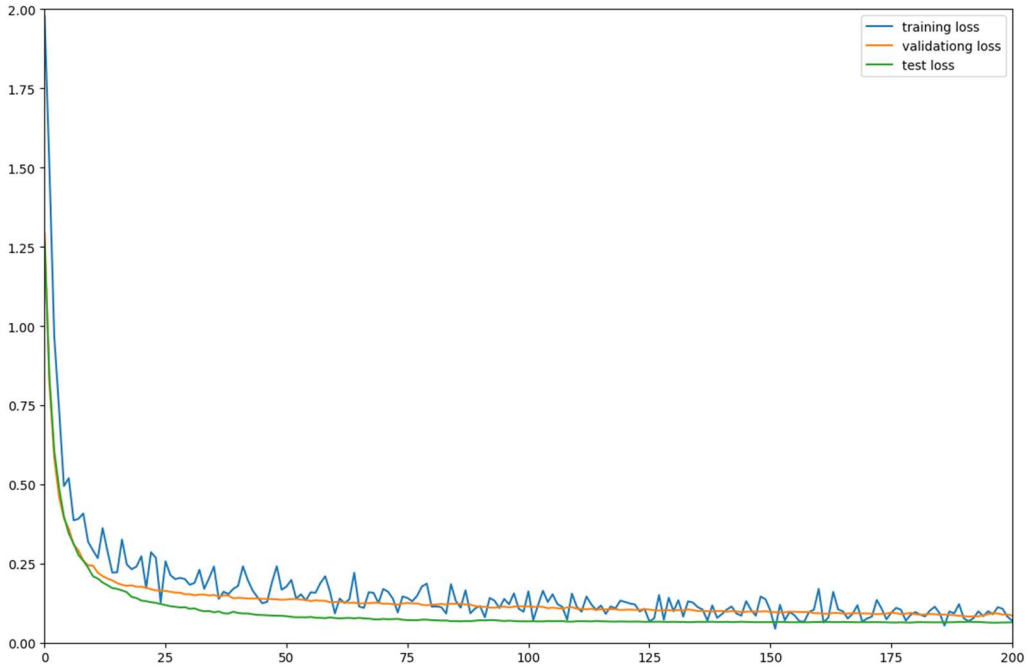


Figure 2 Cross Entropy Loss vs Number of Epochs (Training, Cross-validation and Testing)

1.1.2

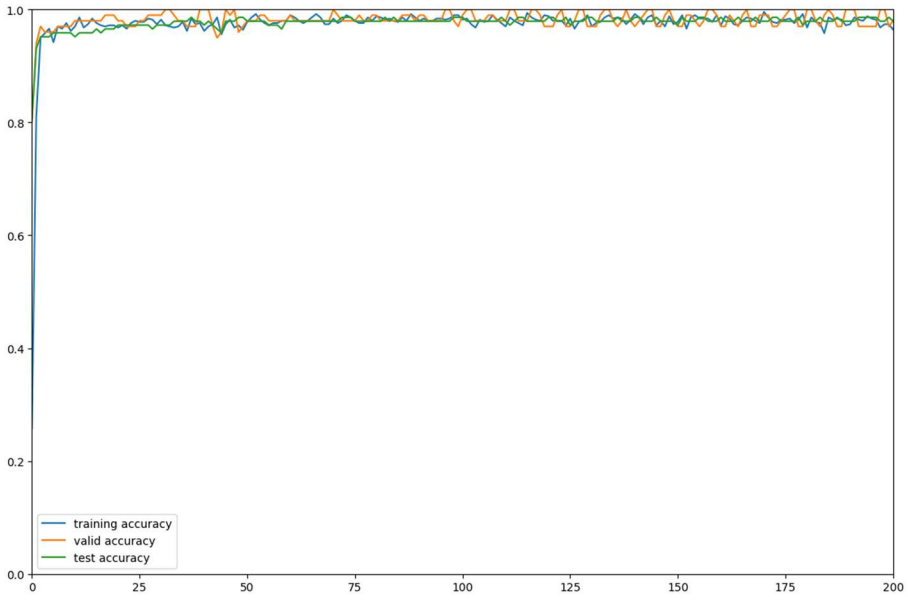


Figure 3 Accuracy vs Number of Epochs (Training, Cross-validation, Testing)

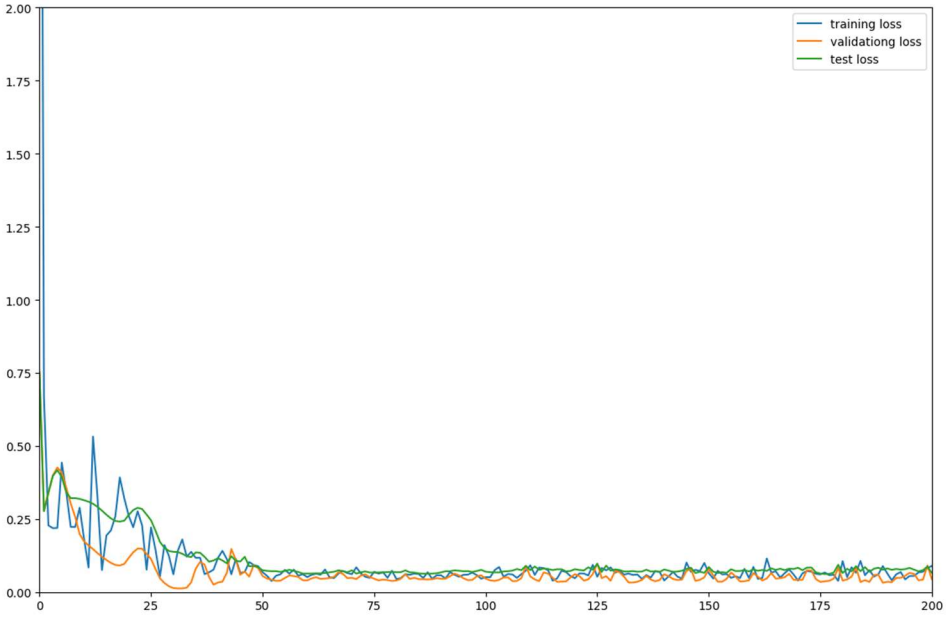


Figure 4 Cross-entropy loss vs Number of Epochs (Training, Cross-validation, and Testing)

The Adam optimizer quickly reduces the training loss at the first few iteration to value under 0.25, where as in stochastic gradient descent it takes approximately 25 iterations. However, SGD has a lower tendency to overshoot than the Adam optimizer during training. Nevertheless, it can be observed that the loss converges at around 50th iteration with ADAM optimizer, yet it takes almost 200 iteration for SGD. Therefore, Adam optimizer converges faster in this problem and should be used.

1.1.3.

The classification accuracy is reported as follows:

	Logistic Regression (Cross-Entropy)	Linear Regression (Square)
Training Accuracy	0.99971431	0.993143
Validation Accuracy	0.99000001	0.97
Test Accuracy	0.97241378	0.951724

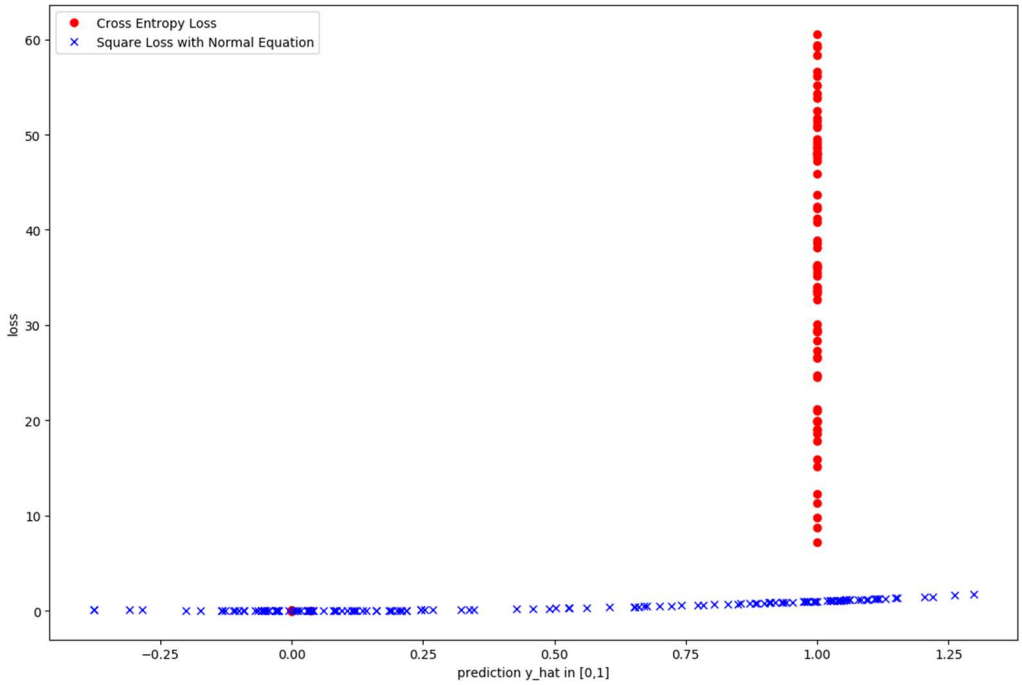


Figure 5 Loss (Cross-entropy and l2) vs Prediction [0,1}

From the plot, the following can be observed:

1. With logistic regression, the predictions made using logistic regression cluster at 0 or 1, but the prediction made using linear regression is spread out.
2. Incorrect prediction is heavily penalized in the case of cross-entropy loss. In square loss, the penalty remains low for all incorrect predictions.
3. Thus, using cross entropy loss, the loss function is more sensitive to incorrect data, especially in the restricted range [0, 1].

1.1.4

For $k \in \{0,1\}$, the Bernoulli distribution of a single observation is given by

$$P(Y = k|x, W) = p^k(1-p)^{1-k}$$

Thus log-likelihood is

$$\log P(Y = k|x, W) = k \log \hat{y} + (1-k)\log(1-\hat{y})$$

For n different independently and identically distributed rvs that are Bernoulli distributed, we have that

$$P(Y_1 = k_1, Y_2 = k_2 \dots Y_n = k_n|X, W) = \prod_{i=1}^{n} P(Y_i = k_i|x, W)$$

Thus,

$$logP(Y_1 = k_1, Y_2 = k_2 \dots Y_n = k_n|X, W) = \sum_{i=1}^{n} logP(Y_i = k_i|x, W)$$

$$= \sum_{i=1}^{n} k_n \log p + (1-k_n)\log(1-p)$$

Thus, the MLE estimation

$$\hat{y} = \underset{p}{\operatorname{argmax}}\, logP(Y_1 = k_1, Y_2 = k_2 \dots Y_n = k_n|X, W) = \underset{p}{\operatorname{argmax}} \sum_{i=1}^{n} k_n \log p + (1-k_n)\log(1-p)$$

The maximum of the above expression is equivalent to the minimum of the expression multiplied by -1. Therefore, the MLE for $\hat{y}$

$$\hat{y} = \underset{p}{\operatorname{argmin}} \sum_{i=1}^{n} -k_n \log p - (1-k_n)\log(1-p)$$

The cross entropy, by definition for $k \in \{0,1\}$ is given by

$$CE = \sum_{i=1}^{n} -k_n \log \hat{y} - (1-k_n)\log(1-\hat{y})$$

Thus,

$$\min_{\hat{y}}(CE)$$

$$= \min\left(\sum_{i=1}^{n} -k_n \log \hat{y} - (1-k_n)\log(1-\hat{y})\right)$$

$$= max\big(logP(Y_1 = k_1, Y_2 = k_2 \dots Y_n = k_n|X, W)\big)$$

As required.

1.2.1

We chose R_k such that a posteriori is maximized, thus

$$R_K = \underset{R_j, j=1\ldots n}{\operatorname{argmax}} P\big(y = C_k, x \in R_j \big| W\big) = \underset{R_j, j=1\ldots n}{\operatorname{argmax}} P\big(y = C_k \big| x \in R_j, W\big)$$

The expected loss

$$E[L] = \sum_{k=1}^{n} \sum_{j=1}^{n} L_{kj} P(x \in R_j, y = C_k)$$

If incorrect predictions all share equal penalty and we are NOT penalizing correct predictions, then

$$E[L] = \alpha \sum_{k=1}^{n} \sum_{j=1}^{n} P(x \in R_j, y = C_k) - \sum_{k=1}^{n} P(x \in R_j, y = C_k)$$

Thus

$$\min(\, E[L]) = \min\left(\sum_{k=1}^{n} \sum_{j=1}^{n} P\big(x \in R_j, y = C_k\big)\right) - \max\left(\sum_{k=1}^{n} P(x \in R_k, y = C_k)\right)$$

By the sum rule of probability, we also know that

$$\sum_{j=1}^{n} P(x \in R_j, y = C_k) = 1$$

Which implies

$$\alpha \sum_{k=1}^{n} \sum_{j=1}^{n} P\big(x \in R_j, y = C_k\big) = \alpha k$$

Which is just a constant. Thus the minimum of the first term is NOT dependent on how we choose R_k.

Therefore, we have shown that

$$\min(\, E[L]) = \ \ \alpha k - \max\left(\sum_{k=1}^{n} P(x \in R_j, y = C_k)\right)$$

$$\min(\, E[L]) = \ \ \alpha k - \sum_{k=1}^{n} \max(P(x \in R_k, y = C_k))$$

Thus, since we have chosen R_k to be such that a posteriori probability is maximized, we have that

$$\min(\ E[L]) = \ \ \alpha k - \sum_{k=1}^{n} P(x \in R_k, y = C_k)$$

Thus the classifier represents a minimum-loss system.

1.2.2

$$E[L] = \sum_{k=1}^{n} \sum_{j=1}^{n} L_{kj} P(x \in R_j, y = C_k)$$

$$\min(\ E[L]) = \min\left(\sum_{k=1}^{n} \sum_{j=1}^{n} L_{kj} P\big(x \in R_j, y = C_k\big) \right)$$

We want to choose the best R_k such that if $x \in R_k$ then classify it to class C_k.

Again, in the case that no information was given on L_{kj}, if we assume that the penalty on the correct prediction is 0, then x should be assigned to the class C_k **if**

$$\sum_{\substack{j=1 \\ j \neq k}}^{n} L_{kj} P\big(x \in R_j, y = C_k\big)$$

Is the smallest.

1.2.3

The best test classification accuracy is 0.894273. The accuracy is significantly worse than that of the binary case. Intuitively, this is true because in the case of binary class, if a prediction were to be made randomly, there is at least 50% that we will get it right. However, in the case where there 10 classes that we need to classify, the probability of make a random correct prediction is only 10%.

Moreover, intuitively it is easier to identify the difference between 2 classes than it is to distinguish the difference between 10 classes. For instance, some classes in the 10 classes might share common features which the model needs to recognize. But for binary case, the difference is straightforward.

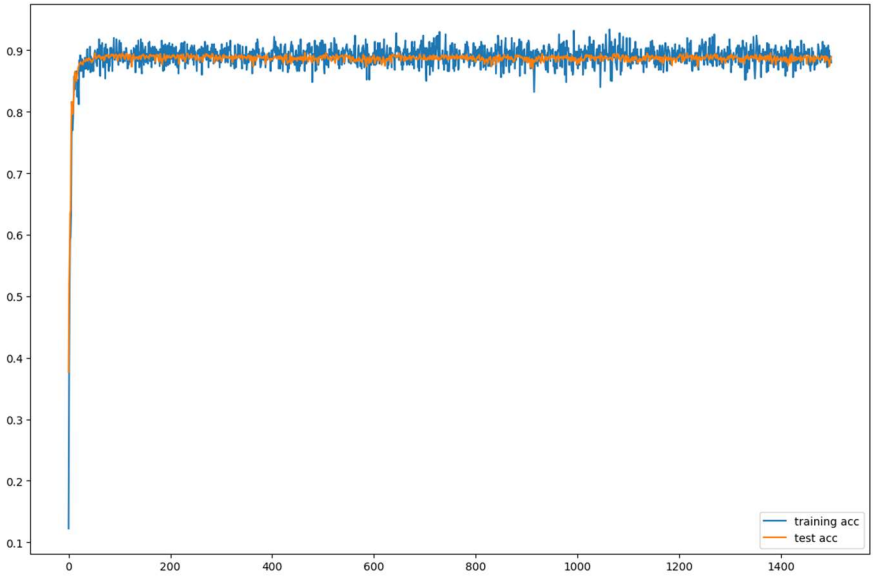


Figure 6 Accuracy vs Number of Epochs (Training and Test)

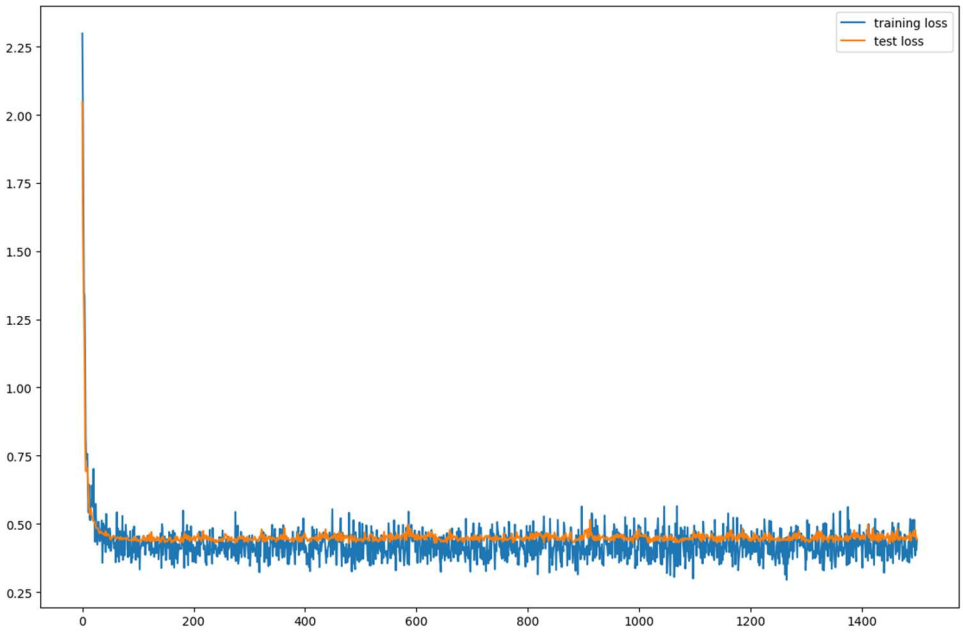


Figure 7 Cross-entropy loss vs Number of Epochs (Training and Test)

2.1.1

We note that the sigmoid function

$$f(m) = \frac{1}{1 + e^{-m}}$$

Becomes a step function centered at 0 as m→ ∞

For a linearly separable data set, we have that

$$\begin{cases} P(Y = 1) = 1, & if\ t > \theta^T x \\ P(Y = 0) = 0, & if\ t \leq \theta^T x \end{cases}$$

Thus

$$\begin{cases} P(Y = 1) = 1, & if\ x < \theta^{T^{-1}} t \\ P(Y = 0) = 0, & if\ x \geq \theta^{T^{-1}} t \end{cases}$$

To illustrate this, we take the simple case when the data is 1D, we have

$$\begin{cases} P(Y = 1) = 1, & if\ x < \frac{t}{\theta} \\ P(Y = 0) = 0, & if\ x \geq \frac{t}{\theta} \end{cases}$$

Which is a step function shifted to $\frac{t}{\theta}$.

To construct a step function above with sigmoid,

$$f(x) = \frac{1}{1 + e^{-k(x - \frac{t}{\theta})}}$$

The weight in this case is k, which has to approach infinity for $f(x)$ to behave as a step function. So the weight $w = k$ has to be such that $\left|\left|w\right|\right|_2^2 = \infty$

2.1.2

If data is linearly inseparable, then we know that there exists some region α in X such that

$$0 < P(Y = 1 | -\alpha < X < \alpha) < 1$$

Same holds for $P(Y = 0 | -\alpha < X < \alpha)$. The probability is strictly greater than 0 and strictly less than 1.

From this, we know that k must be bounded. If not, then the sigmoid function will become a step function, as illustrated in 1.2.1.

2.1.3

Using the properties proven in 2.1.1 and 2.2.2, we consider the 2 cases:

1. Raw data is linearly inseparable, but after passing through the first hidden layer, data becomes separable.
2. Raw data is linearly inseparable at every stage.

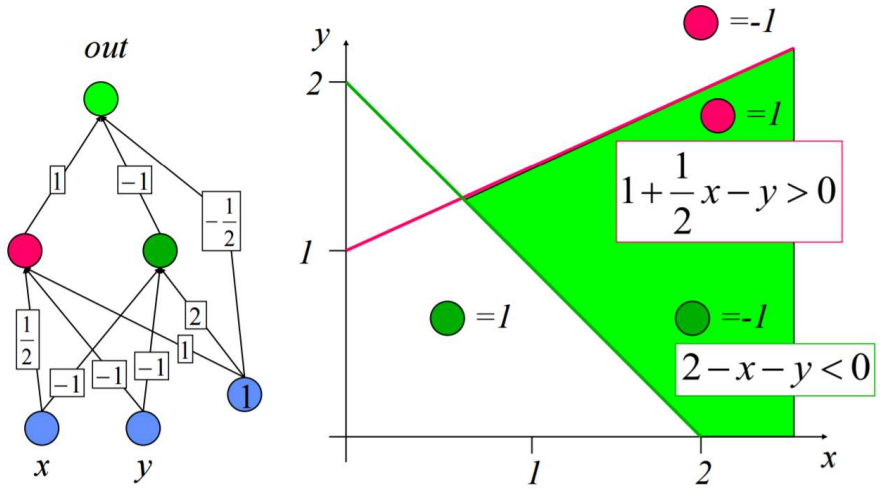


Figure 8 Source: University of Washington CSE573 [1]

Concretely, for case 1, we consider the data in Figure 8 above. Suppose that the data in the cyan region is classified as 1, and 0 otherwise. The original neural network has the optimum weight as shown. The red and green line in the figure represents the output of the red and green neurons respectively. Clearly, the data is linearly inseparable. **However, the input that the out neuron takes in is linearly separable.**

Now, modify the neural network such that the "out" neuron has a sigmoid activation function as shown below in Figure 9.

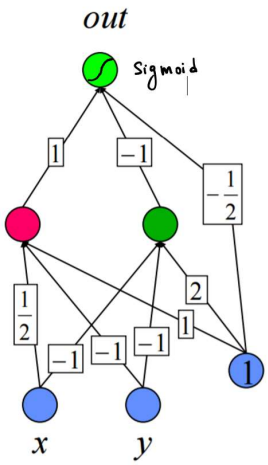


Figure 9 Modified Neural Network

Since we know the out neuron takes in linearly separable data, the weight matrix connecting the hidden layer and output layer is unbounded. It follows that the magnitude of concatenated vector of matrices is also unbounded. That is $\left|\left|vec\{w\}\right|\right|_2^2 = \infty$

For case 2, we consider the data set consisting only 2 examples of different class but lies at the same location (i.e. same input but different class). It is clear that in all stages the inputs that every neuron takes in remain linearly inseparable because two points are inseparable. Thus, the optimal weights are always bounded.

2.2.1

```
def compute_weighted_sum(previous_layer, n_units):

    #xavier initialization
    dev = tf.sqrt(3/(n_units[0] + n_units[1]))
    initW = tf.truncated_normal(shape=[n_units[0],n_units[1]], stddev = dev )

    #weight matrix
    weight_matrix = tf.Variable(initW, [n_units[0], n_units[1]], name = ↵
↳   "weight_matrix")

    #bias (broadcasted)
    bias = tf.Variable(tf.truncated_normal(shape=[1,n_units[1]],stddev=dev↵
↳   ), [1,n_units[1]], name="bias")

    #weighted sum
    Z = tf.matmul(previous_layer,weight_matrix) + bias

    return Z, weight_matrix
```

2.2.2

When regularization of 3E-4 is applied, the plots below were obtained. The second figure is a plot of the cross-entropy loss vs number of epochs. The first figure is the plot of error vs number of epochs. We observe that the cross validation loss reaches a minimum at iteration 23, and the validation error reaches a minimum at iteration 47. The training accuracy at the end reaches 0.998 (error is 0.002). The validation accuracy reaches 0.939 (error is 0.061), and the test accuracy reaches 0.924 (error is 0.076). We observed that the training loss continues decreasing until it and almost reaches 0 throughout training, but the cross-validation loss and test loss cease to decrease after some epochs.

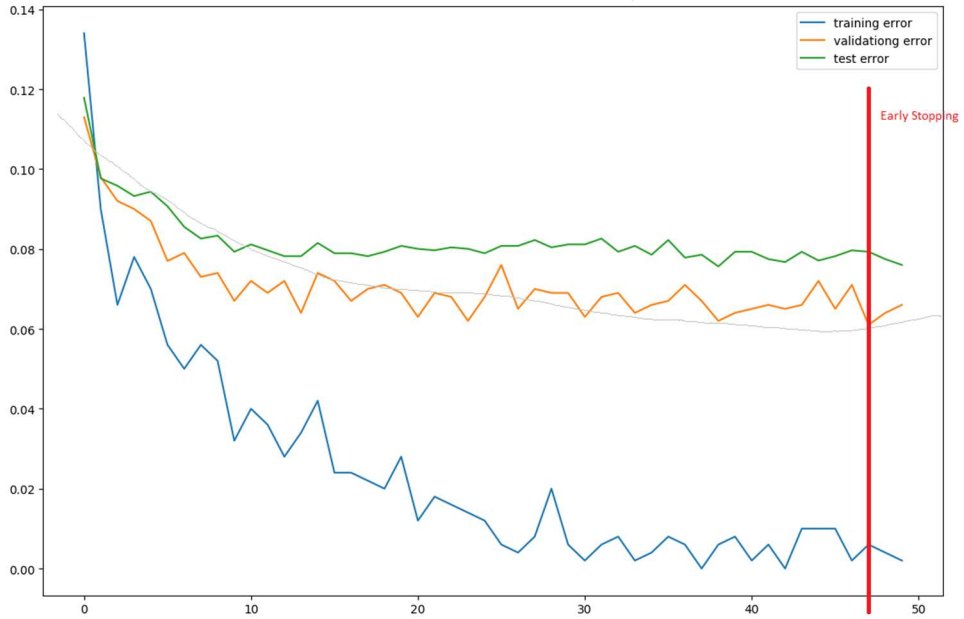


Figure 10 Error vs Number of Epochs (Training, validation and Test)

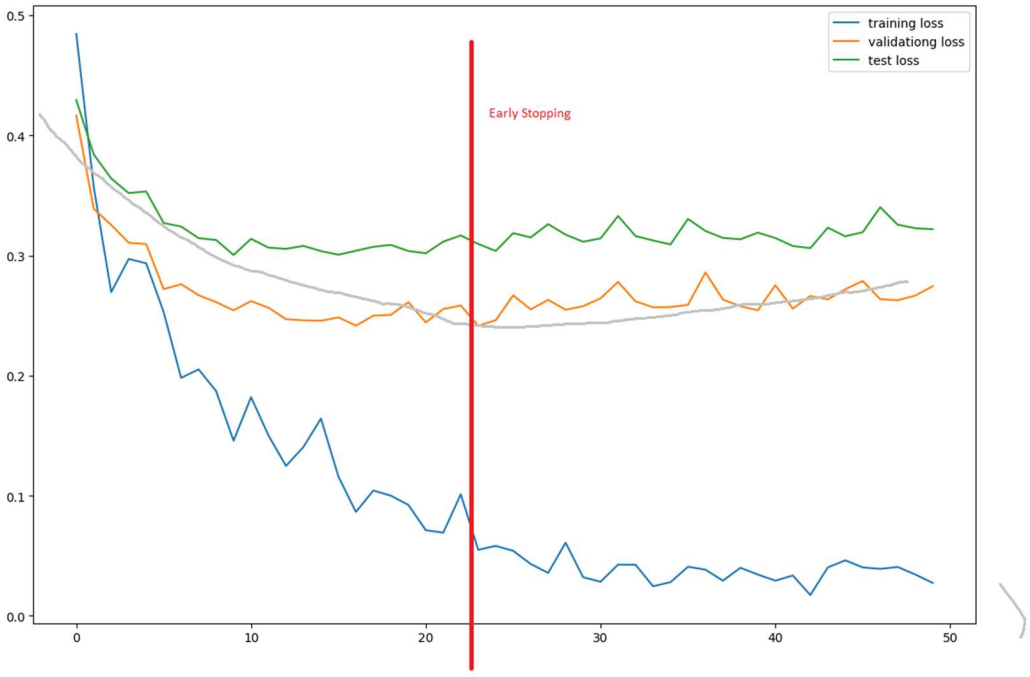


Figure 11 Cross-entropy loss vs Number of Epochs (Training, validation and Test)

2.2.3.

Using the plots from 2.2.2 with regularization equals 3E-4, the early stopping points happens around 23th epoch for cross-validation loss, and at 47th epoch for the plot of classification error, as illustrated in Figure 10 and Figure 11. At epoch = 47, the validation accuracy has achieved its maximum. And, at epoch = 23, the validation loss has achieved its minimum and is equal to 0.241468. Notice that the training error and training loss is still decreasing after the early stopping point.

The two plots is different. This is because the loss function is a continuous function that measures how similar prediction is to the target for each training example. Whereas the classification error discretizes the similarity between prediction and target to either correct or incorrect. Thus the loss function gives a more accurate depiction and should be used.

2.3.1

	Best Validation Error	Best Test Error
Hidden units = 100	0.934	0.914
Hidden units = 500	0.94	0.923
Hidden units = 1000	0.94	0.924
Hidden units = 3000	0.945	0.926

After extensive testing, we can conclude that the more hidden units the higher the accuracy of the final result for the notMNIST dataset **with regularization 3E-4.**

2.3.2

The graph above shows the validation and test error vs number of epochs. The maximum validation error is 0.944 (error = 0.056), at epoch 22. When the training is complete, overfitting starts to kick in and the validation accuracy decreases slightly to 0.937 (error = 0.063). Comparing the test error of the 2-layer neural net with the 1-layer 1000 units neural network, the 2-layer neural network performance are very similar. The table below summarizes the error.

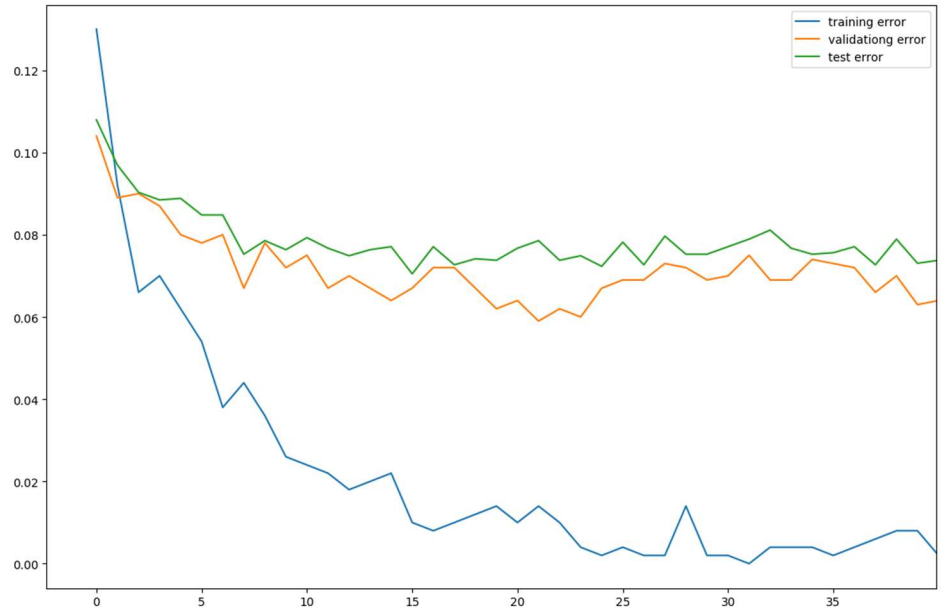


Figure 12 Error vs Number of Epochs (Training, validation and Test)

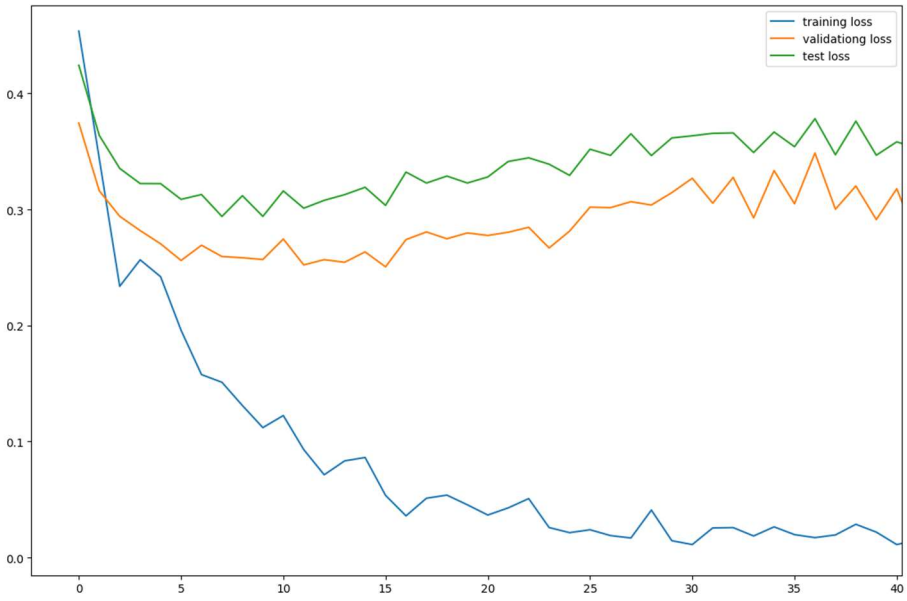


Figure 13 Cross-entropy Loss vs Number of Epochs (Training, validation and Test)

	Validation Error (Best)	Test Error (Best)
1-layer 1000 units	0.94	0.924
2-layer 500 units	0.94	0.927

Comparing the test set of the two architectures, we found that overfitting kicks in slightly earlier in the 2 layer case than in the 1 layer case. From the table, we can see that the test error is slightly better in the two-layer case.

2.4.1

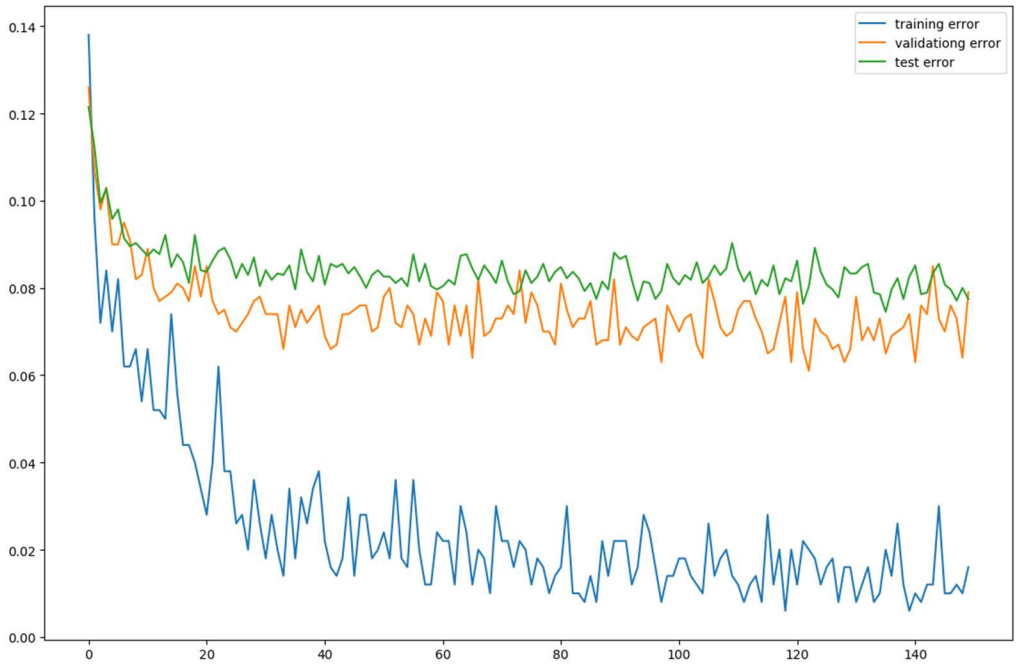


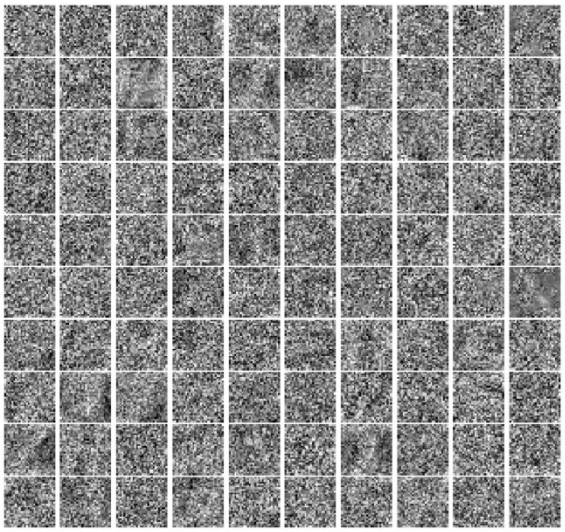
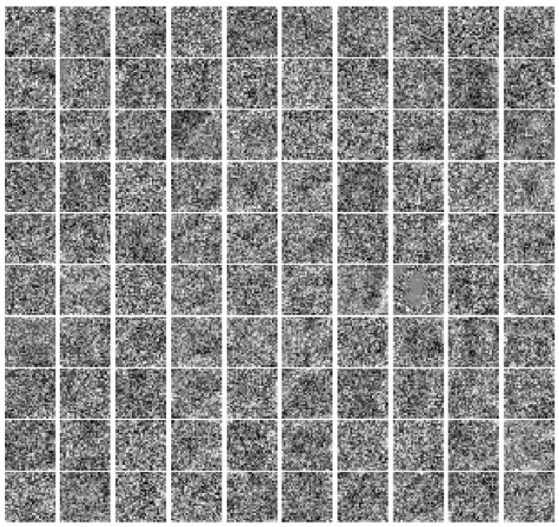
Figure 14 Error vs Number of Epochs (Training, validation and Test)

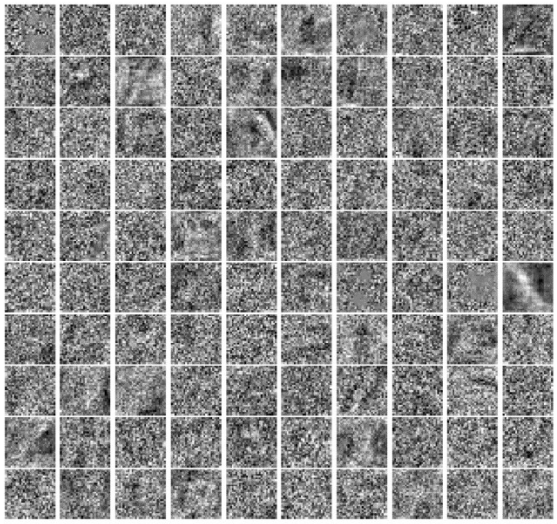
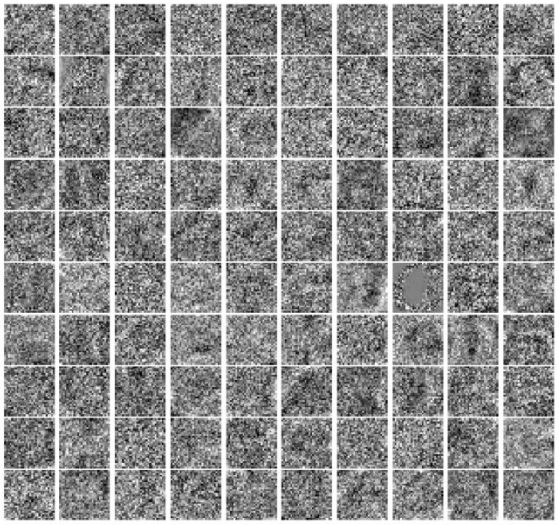
	Training Accuracy	Validation Accuracy	Test Accuracy
Without dropout (from 2.2)	0.998	0.94	0.924
With dropout	0.992	0.936	0.919

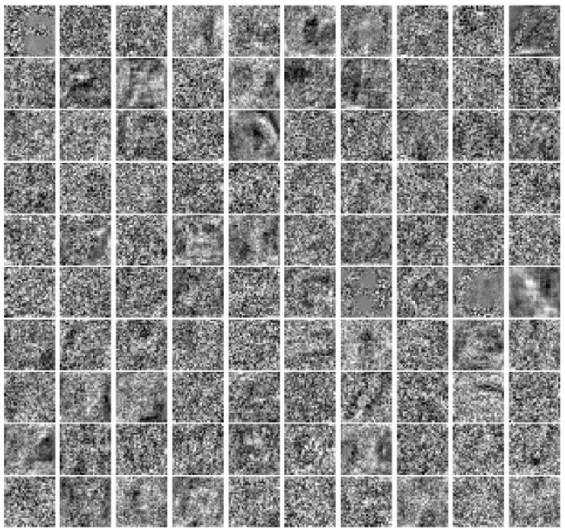
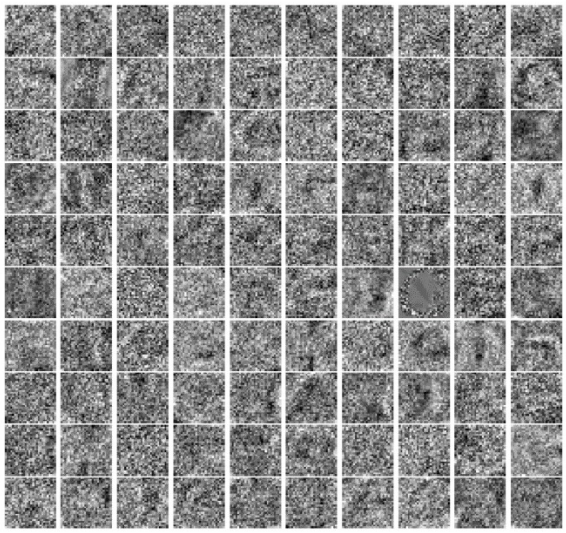
The training accuracy with dropout is lower than the accuracy without dropout. We can also see that the validation accuracy and test accuracy of the neural network with dropout is slightly lower than that without dropout. Hence, the neural network without dropout performs better in this classification.

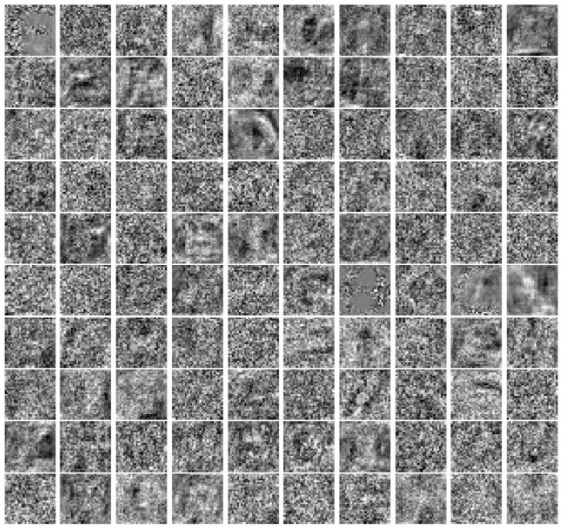
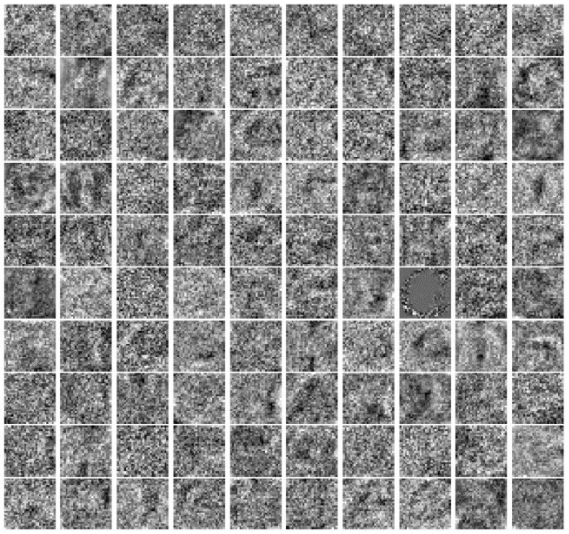
2.4.2

Shown below is a side by side comparison of the visualizations between training with dropout and without dropout.

	
25% , No Dropout	25% , With Dropout

	
50% , No Dropout	50% , With Dropout

	
75%, No Dropout	75%, With Dropout

	
100%, No Dropout	100%, With Dropout

As expected, as epoch increases towards 100% of the early stopping point, the visualization becomes clearer. Even though the 100 visualizations of weights sampled above is a not a full representation of the 1000 neuron units in the neural network, we can still make the following generalizations. First, we notice that in the case of no dropout, patterns can be seen earlier in the training than that with dropout. Also the weights in the neural network with no dropout shows a sharper contrast within each image. (Which implies the magnitude of weights to each neuron with no dropout has a greater variance). Lastly, among the 100 visualizations presented for the case with/without dropout, we can see that there number of noise-free images in the case with no dropout is higher than that with dropout. Nevertheless, the performance between the two is very close and the difference is subtle.

2.5.1

By running a random search on the hyper-parameters, we have the data as follows:

	Model 1	Model 2	Model 3	Model 4	Model 5
Number of Hidden Layers	4	5	5	5	2
Number of Hidden Units	[128 489 497 499]	[386 278 290 235 422]	[464 320 290 379 218]	[327 187 295 315 251]	[187 457]
Learning Rate	0.0047320	0 .00085392	0.0030956	0.00911438	0.00157194
Weight Decay Coefficient	0.00041186	0.00038108	0.00026335	0.00071773	0.000156636
Keep Probability (Dropout)	0.5	0.5	0.5	1	0.5
Best Validation Error	13.7%	8.2%	10%	7.7%	8.2%
Best Test Error	16.2%	9.3%	11%	8.8%	9.5%

2.5.2

Using the specifications given by Maxim and Helen,

	Specification
Number of Hidden Layers	5
Number of Hidden Units	[221 302 341 367 457]
Learning Rate	Exp(-7.238)
Weight Decay Coefficient	Exp(-7.7944)
Keep Probability (Dropout)	0.5

The best validation error is 5.8%. The best test error obtained is 7.2%.

Works Cited

[1] University of Washington, 2011. [Online]. Available: https://courses.cs.washington.edu/courses/csep573/11wi/lectures/22-nns.pdf.