ECE521 A2

Baiwu Zhang 1001127540
Daiqing Li 1000795357

February 2017

1

1.1

1.1.1

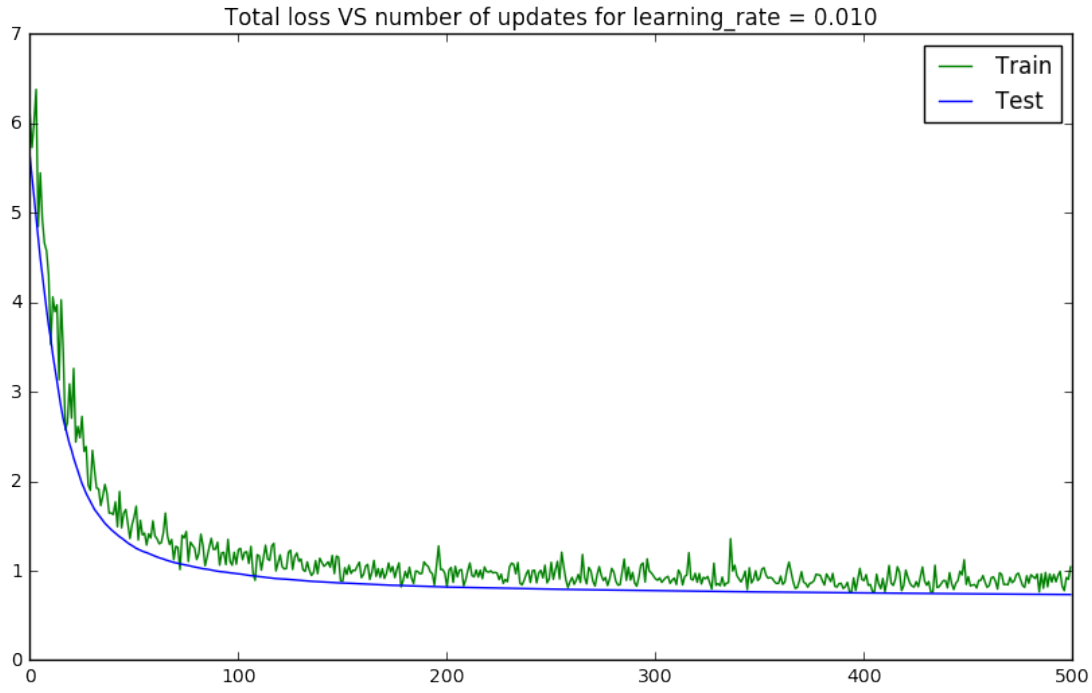

Figure 1: decay rate: 0.1, learning rate: 0.01, batch size: 100

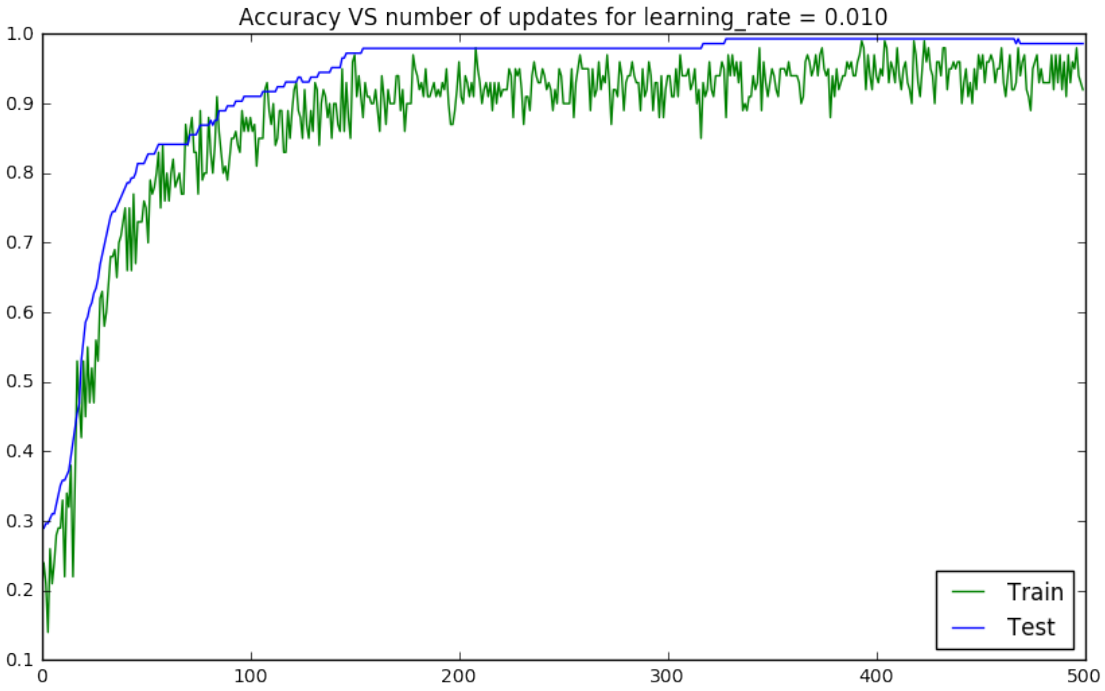

Figure 2: Best accuracy in test set is 0.99

1.1.2

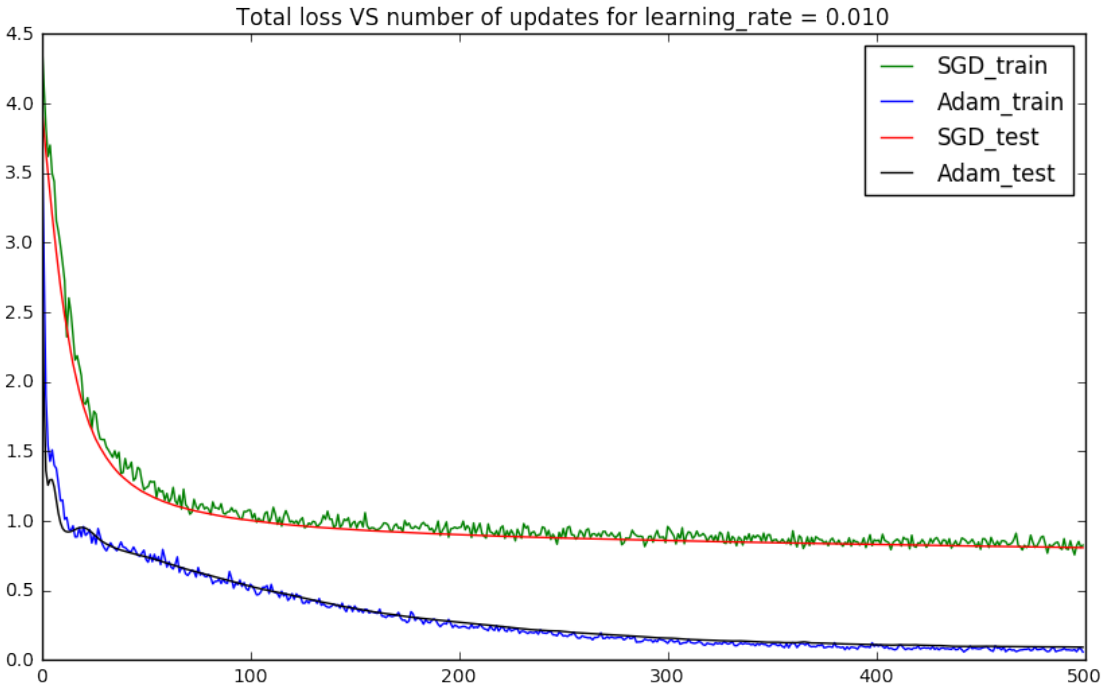

Figure 3: decay rate: 0.1, learning rate: 0.01, batch size: 500

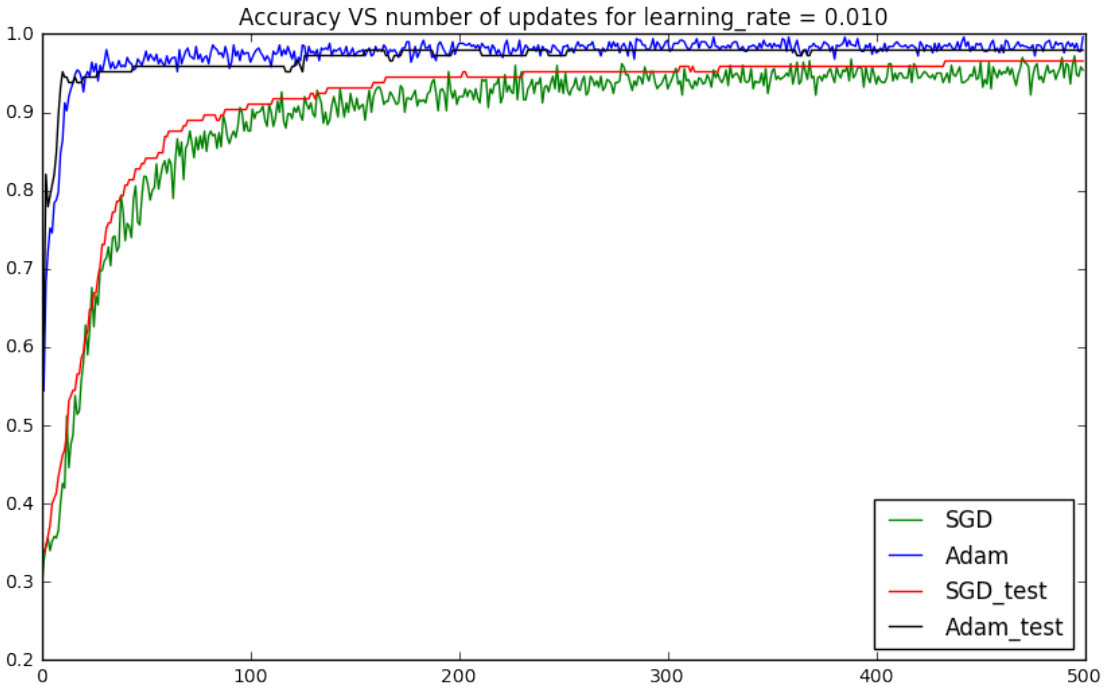

Figure 4: Best accuracy in test set is 0.99

Comparing to SGD, Adam optimizer has a faster convergence rate and it is able to find better optimums.

1.1.3

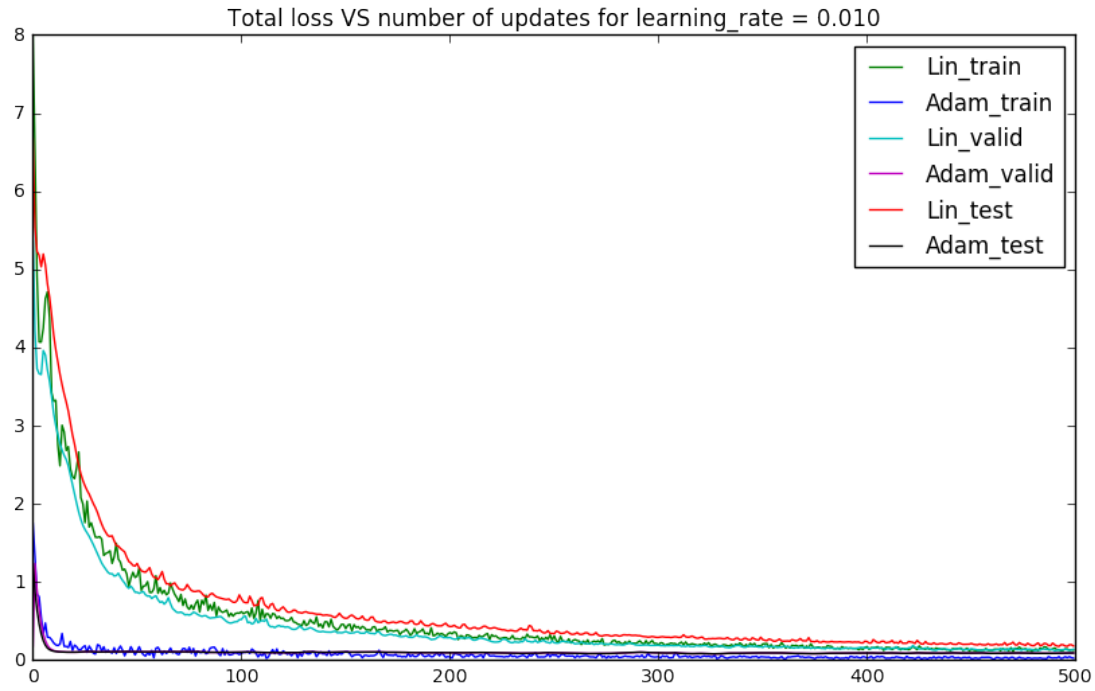

Figure 5: decay rate: 0.0, learning rate: 0.01, batch size: 500

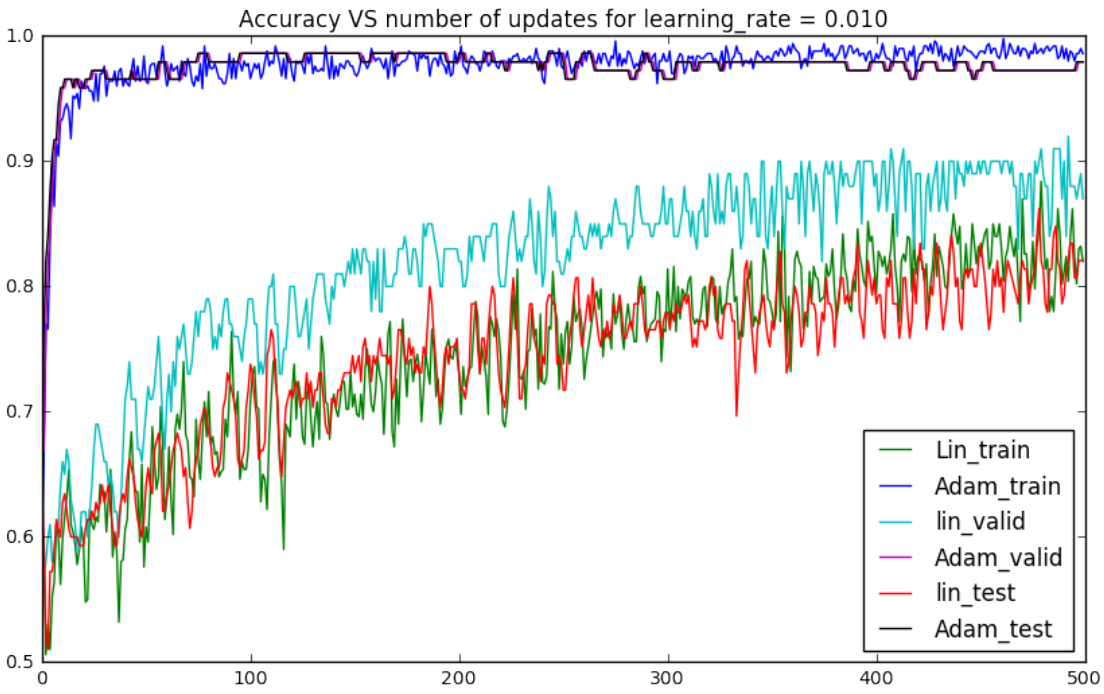

Figure 6: decay rate: 0.0, learning rate: 0.01, batch size: 500

Comparing to least square loss function, cross entropy loss has a faster convergence rate and its validation and test accuracy are higher and more stable. We can find that cross entropy loss has a better performance when we do not add any weight penalty. In other words, least square loss is easier to overfit training set and bad at dealing with outliers.

1.1.4

The log-likelihood of the training data under the Bernoulli assumption is

$$\begin{aligned} log(P(y|x,W)) &= log(\hat{y}(x)^y(1-\hat{y}(x)^{1-y})) && (1)\\ &= log(\hat{y}(x)^y) + log((1-\hat{y}(x))^{1-y}) && (2)\\ &= ylog(\hat{y}(x)) + (1-y)log((1-\hat{y}(x))) && (3) \end{aligned}$$

The above equation is also the negative of the cross-entropy loss. So minimizing the cross-entropy loss is equivalent to maximizing the log-likelihood of the training data under the Bernoulli assumption.

1.2

1.2.1

The expected loss is

$$E[L] = \sum_k \sum_j \int_{R_j} L_{kj} p(x, C_k) dx \tag{4}$$

In this case $L_{kj} = c$ if $k \neq j$ and 0 if $k = j$. In order to minimize the loss, we should assign the data x to the class with the largest posteriori probability $p(x, C_k)$. The loss equation will not contain $p(x, C_k)$ anymore. Choosing any other class will result in a larger loss simply because a smaller term than $p(x, C_k)$ is eliminated in the loss function.

1.2.2

We should assign x to class j which minimizes $\sum_k L_{kj} p(x, C_k)$. The idea is the same as the previous one, but the difference here is that loss penalty is not equal for each class, so we need to take into account all the penalties associated with one class, and choose the minimum.

1.2.3

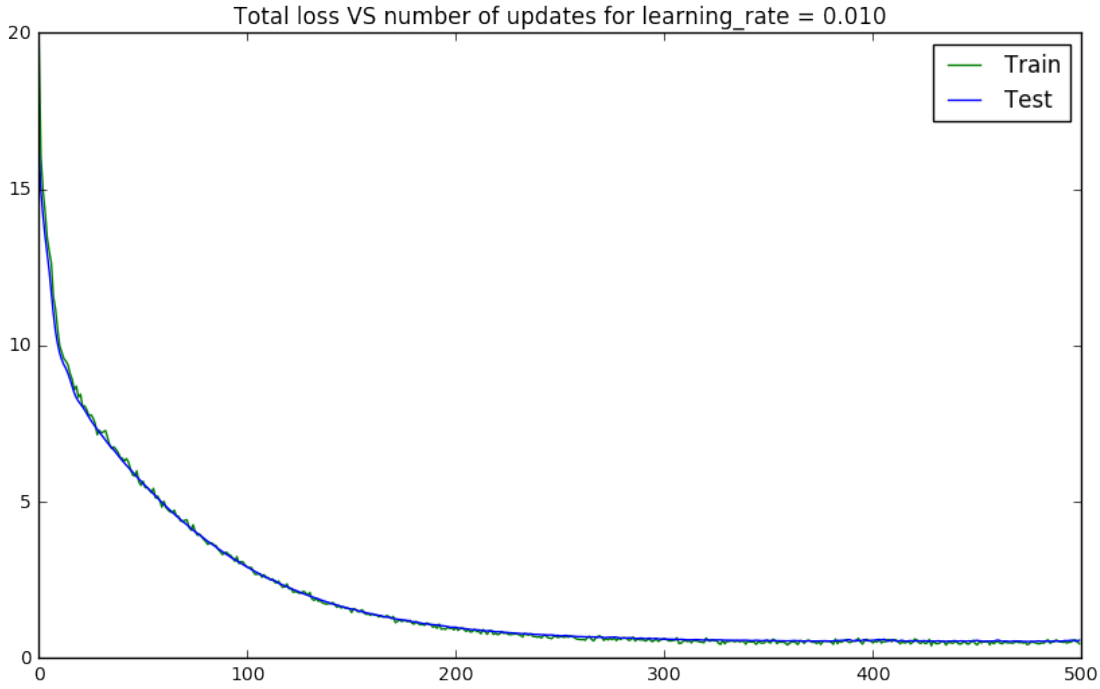

Figure 7: decay rate: 0.1, learning rate: 0.01, batch size: 500

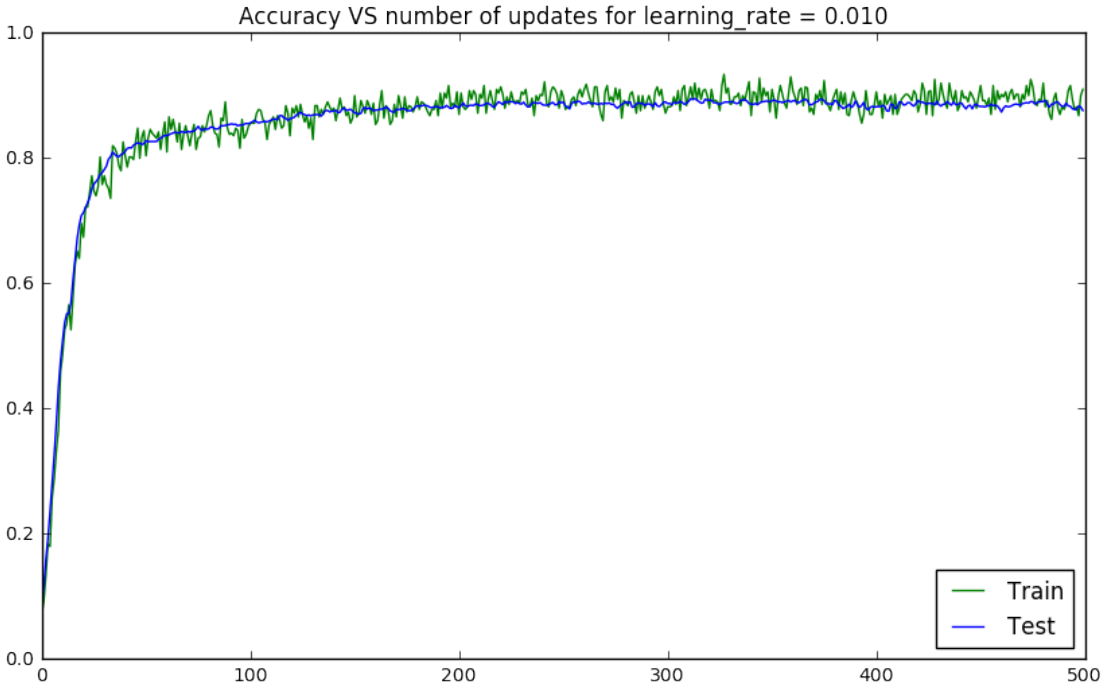

Figure 8: Best accuracy in test set is 0.90

The accuracy is lower than the binary classification problem. Because we need to classify 10 classes in this problem, it makes sense to get a lower accuracy using the same method as the last problem. In binary classification, we only need to find one hyperplane to separate the data, but here we need to find one for each class since we are using one-hot encoding. The problem becomes obviously harder.

2

2.1

2.1.1

The one sigmoid node neural network without weight decay is equivalent to a logistic regression model. And training the net is also equivalent to maximizing

$$\max_{W} \prod_{i=1}^{n} p(y_i|x_i, W) \tag{5}$$

Since the data is linearly separable, the maximum value of the above equation should be 1 where each y_i receives probability of 1. In this case, the sigmoid function becomes a step function, and W has infinite magnitude. Adding weight decay can avoid this situation.

2.1.2

This is the opposite case of 2.1.1. When the data is not linearly separable, maximizing the product of conditional probability will not result in a step function, thus the magnitude of the weights will be finite.

2.1.3

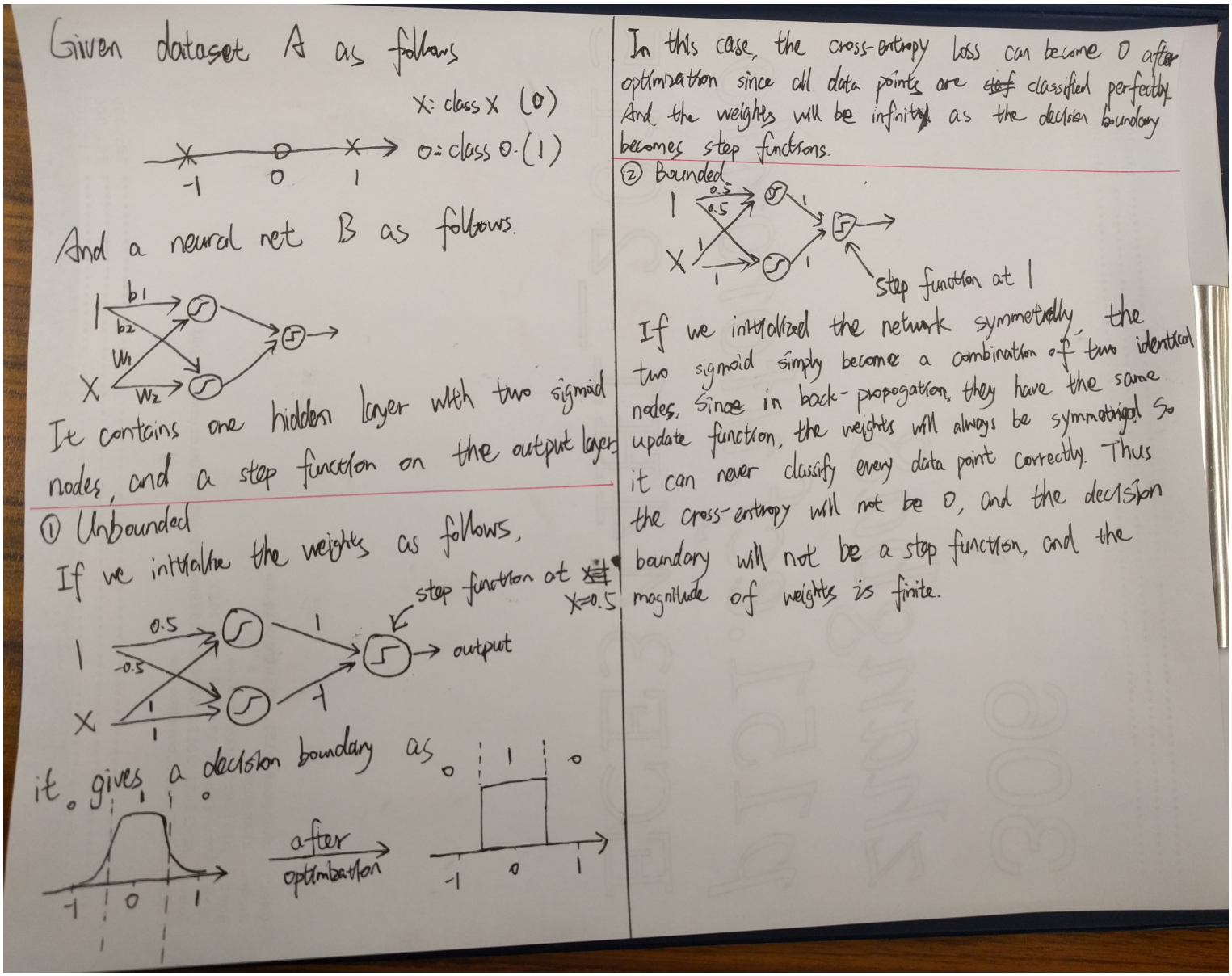

Figure 9: 2.1.3

2.2

2.2.1

```
def layerBlock(x, n_units, decay_rate):
    '''
    x:input tensor
    n_units:number of hidden units in this layer
    '''
    n_input = x.get_shape().as_list()[1]
    XavierSTD = np.sqrt(3.0 / (n_input + n_units))
    W = tf.Variable(tf.truncated_normal(shape=[n_input, n_units], stddev=XavierSTD)
    , name="weights")
```

```
    b = tf.Variable(tf.zeros([n_units]), name="biases")
    weight_decay = tf.reduce_sum(tf.scalar_mul(decay_rate, tf.nn.l2_loss(W)))
    tf.add_to_collection('losses', weight_decay)
    z = tf.add(tf.matmul(x, W), b)
    return z
```

Listing 1: 2.2.1

2.2.2

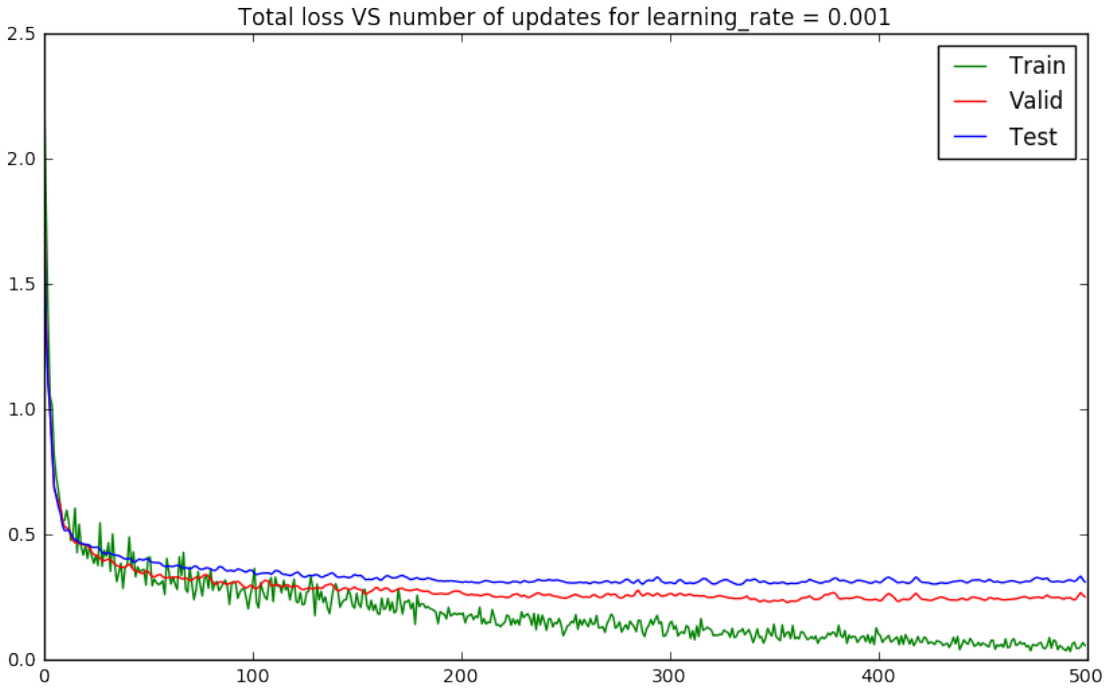

Figure 10: decay rate: 3e-4, learning rate: 0.001, batch size: 100

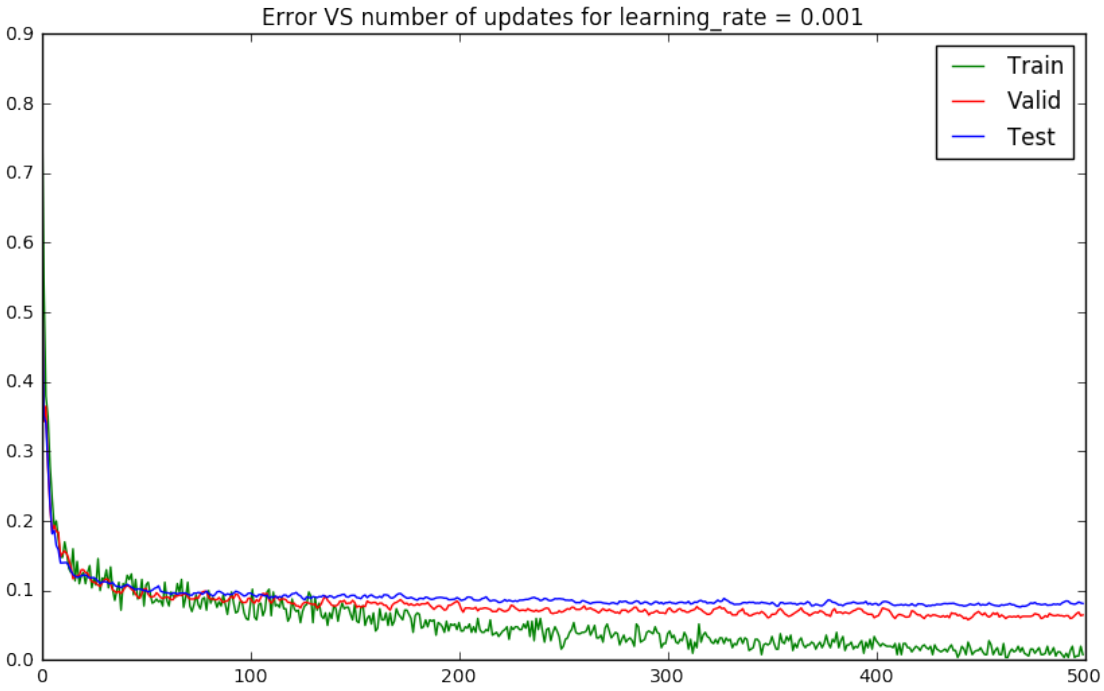

Figure 11: Best accuracy in test set is 0.92

From the loss graph, we can see the test loss and valid loss do not diverge even the training loss is almost reaching zero.

2.2.3

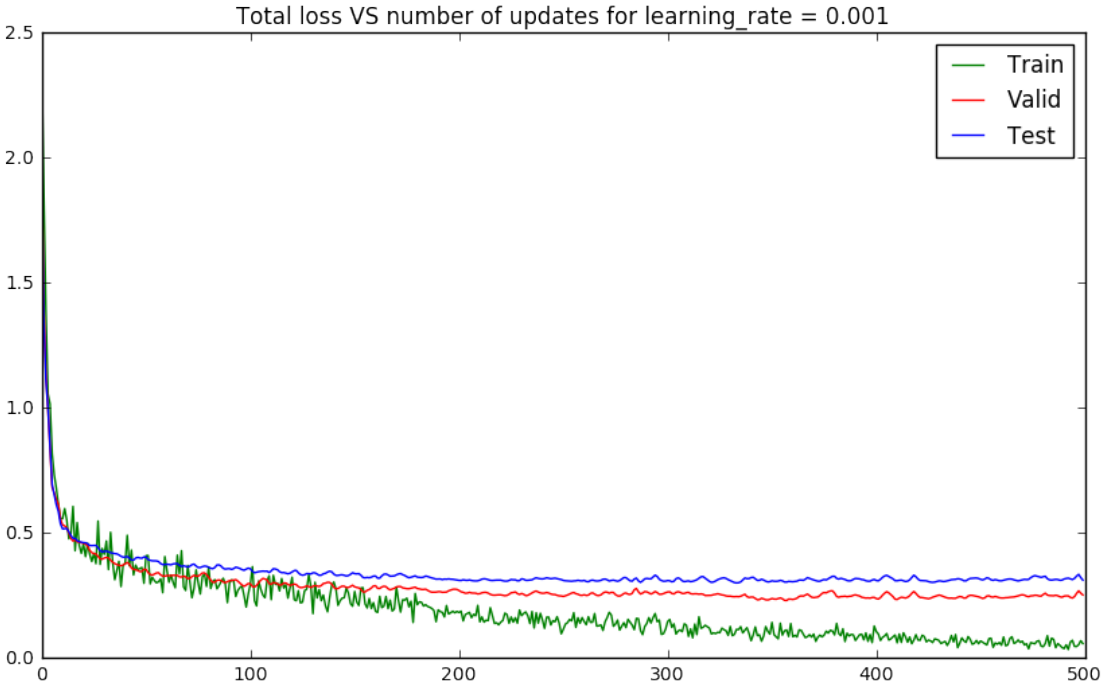

Figure 12: decay rate: 3e-4, learning rate: 0.001, batch size: 100

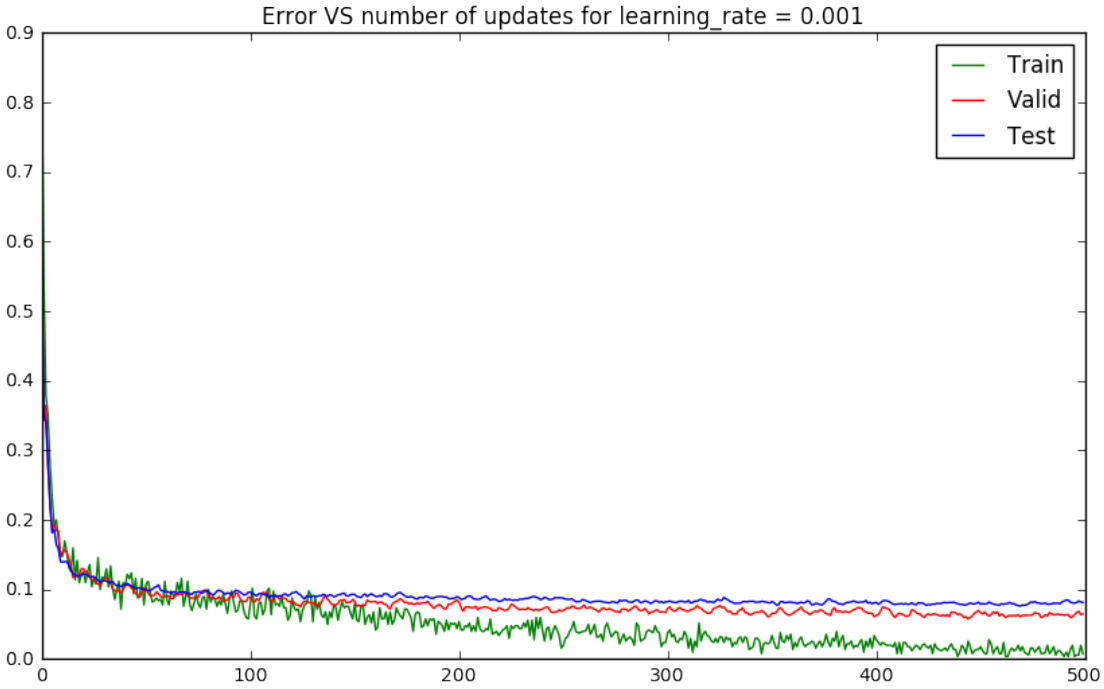

Figure 13: Best accuracy in test set is 0.92

From the loss graph we can see valid loss is almost flat after 200 iteration. We can set 200 as the early stop point. At this point, training error is 0.03, valid error and test error are 0.09 and 0.08 respectively. The early stop points are different in two graphs. The valid error becomes flat after 100 iteration in error plot. We should use loss graph to determine early stop point because we use validation loss to check if the training is overfit. We want to find the minimum validation loss point before training loss reaching minimum.

2.3

2.3.1

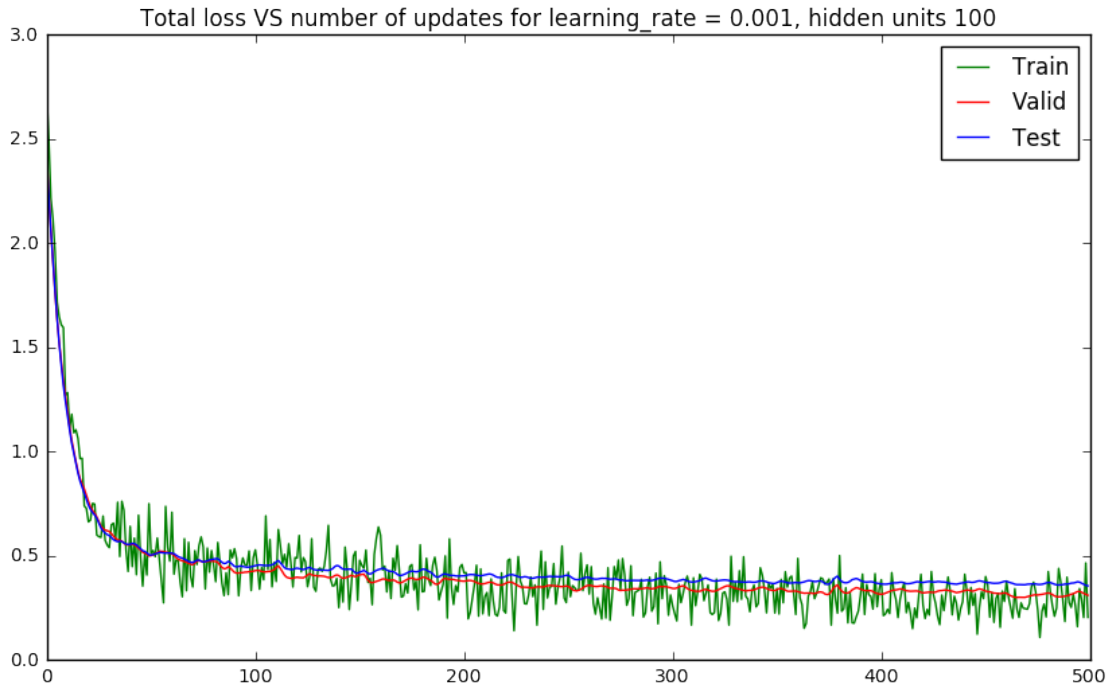

Figure 14: decay rate: 3e-4, learning rate: 0.001, batch size: 100

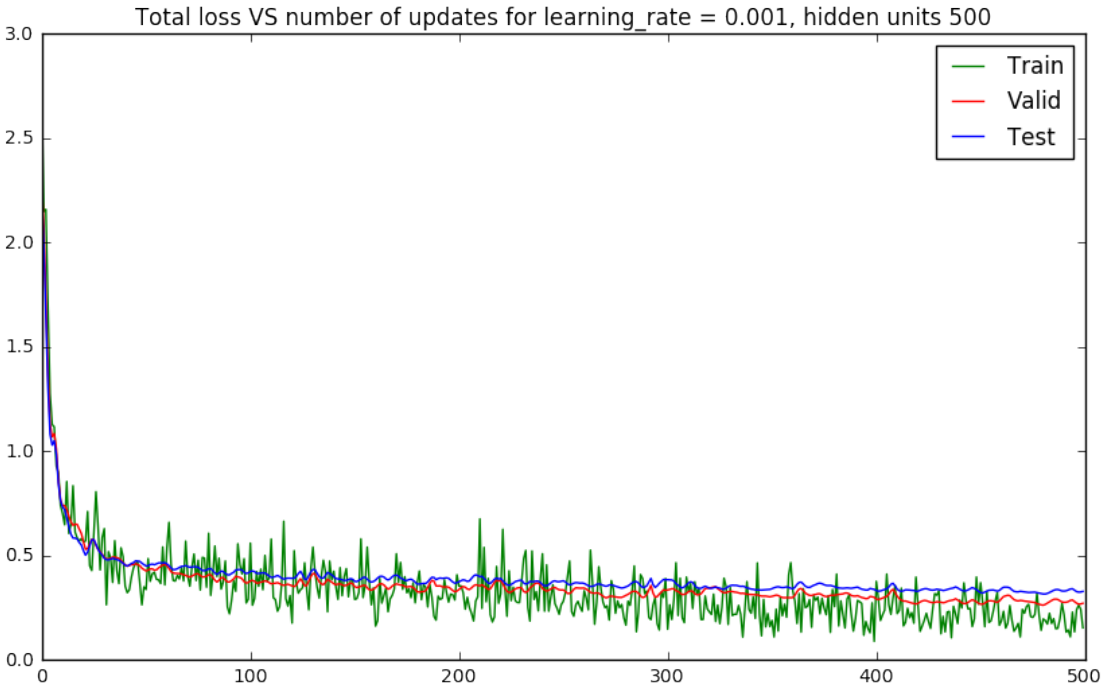

Figure 15: decay rate: 3e-4, learning rate: 0.001, batch size: 100

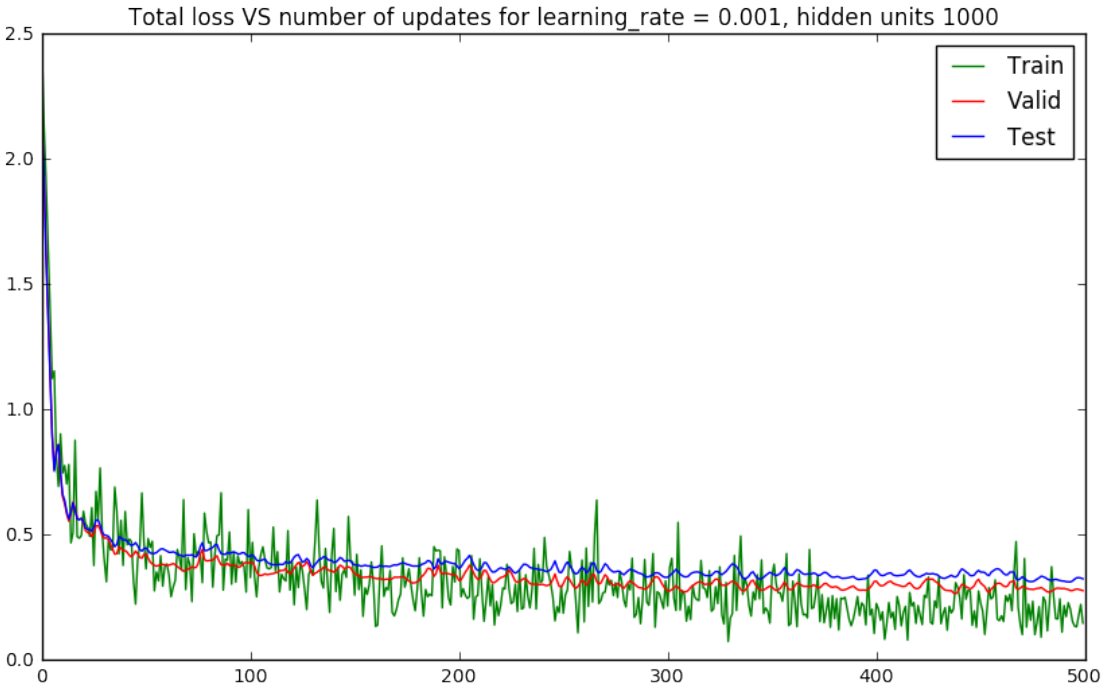

Figure 16: decay rate: 3e-4, learning rate: 0.001, batch size: 100

Number of hidden units affects the variance of validation loss and test loss. From the graph, we can barely see the difference. One of the reason we think is that the net capacity is not enough and our

training iteration is not large enough. A convolution neural net should have better performance.

2.3.2

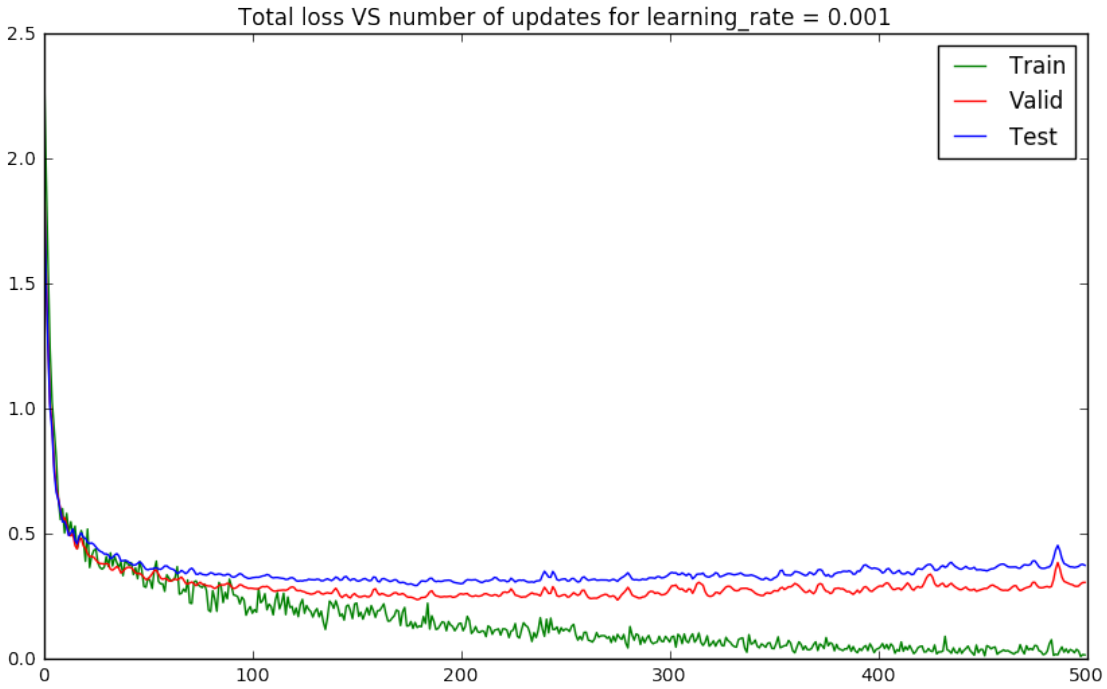

Figure 17: decay rate: 3e-4, learning rate: 0.001, batch size: 500

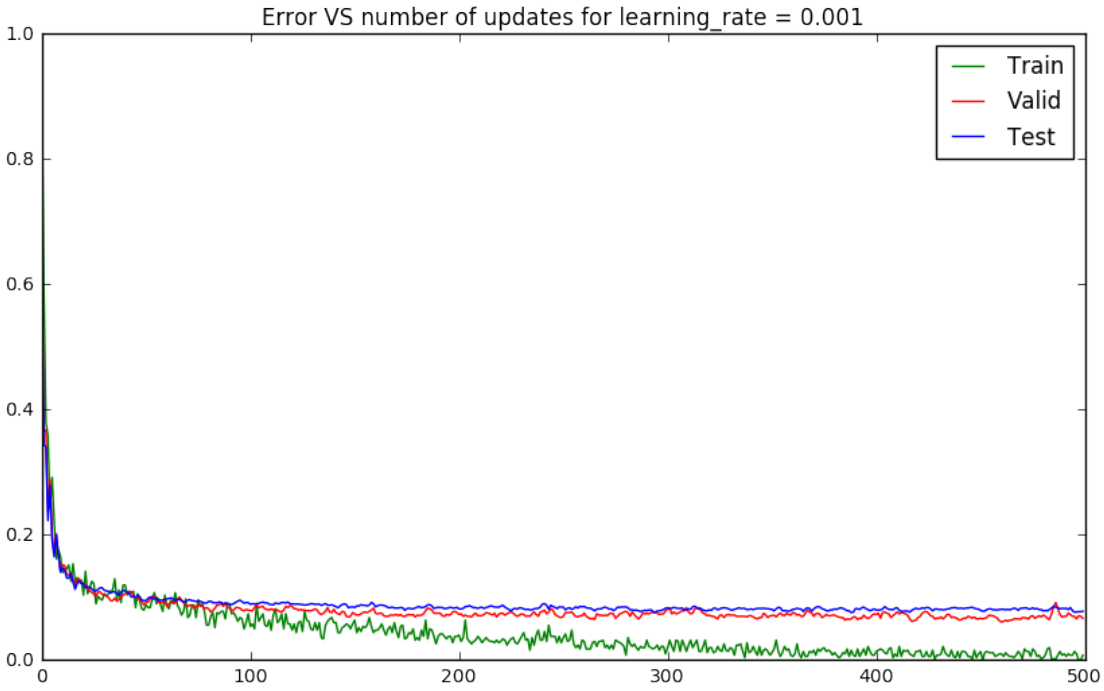

Figure 18: Best accuracy in test set is 0.92

The final validation error is 92.6% after training. Comparing to one layer structure, the validation and testing loss tend to reach minimum at a early point. We think it makes sense because two layers increase the degree of freedom when adjusting weights.

2.4

2.4.1

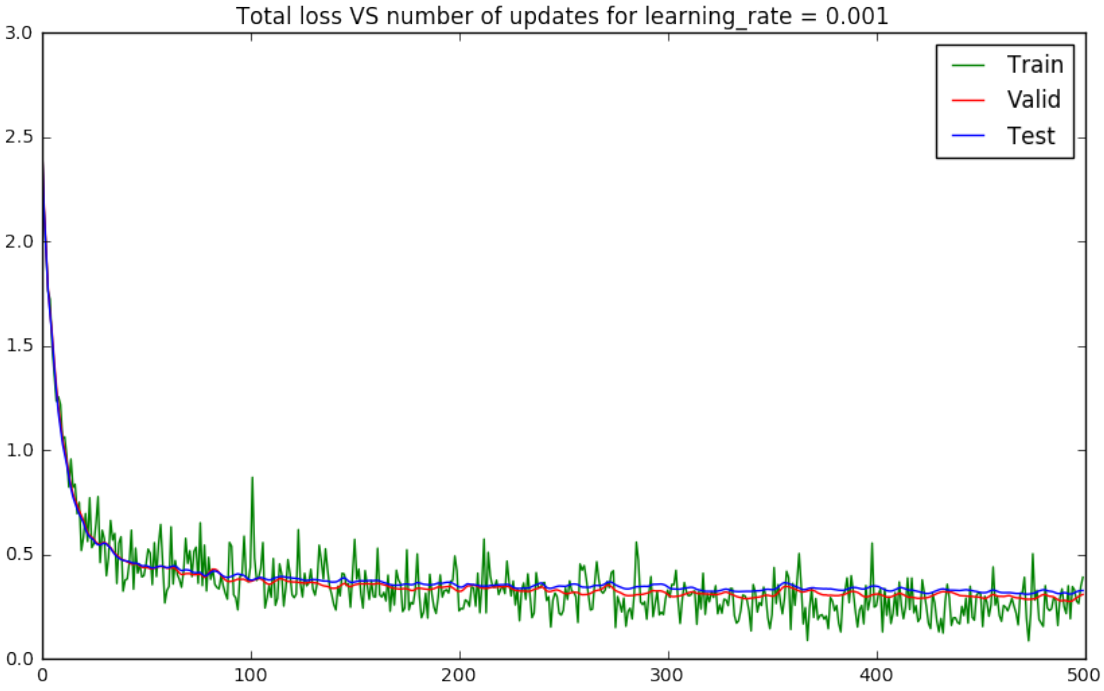

Figure 19: decay rate: 3e-4, learning rate: 0.001, batch size: 100

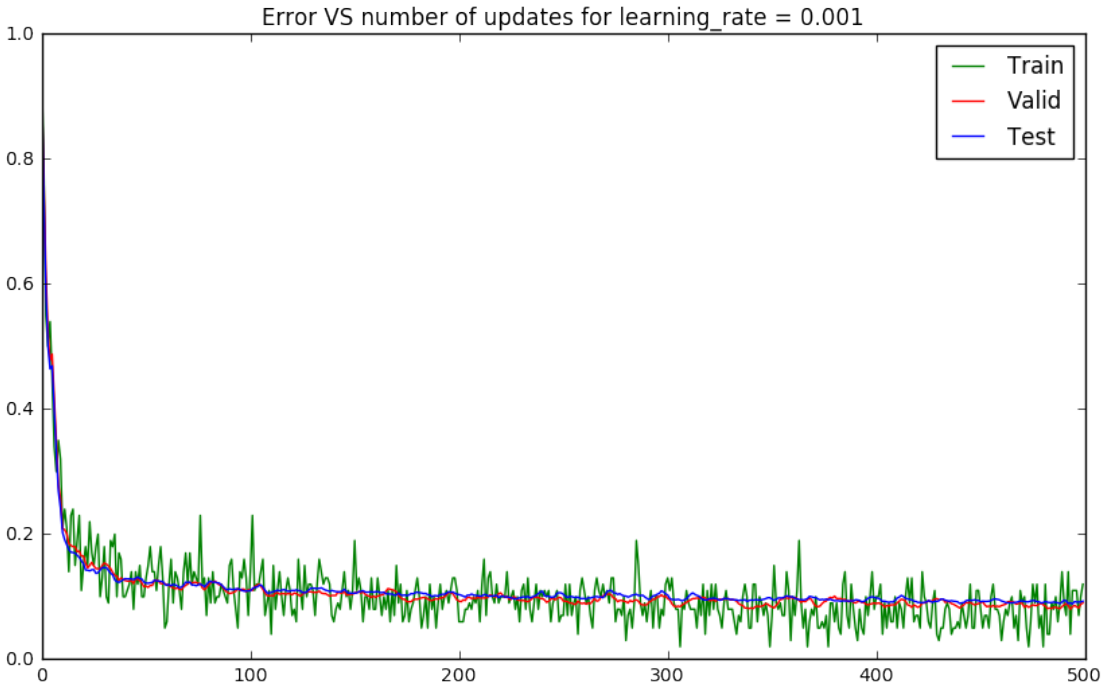

Figure 20: decay rate: 3e-4, learning rate: 0.001, batch size: 100

From graphs above, we find that dropout makes validation and test loss less varied and more stable. The overfit point in previous implementation is less obvious in dropout implementation.

2.4.2

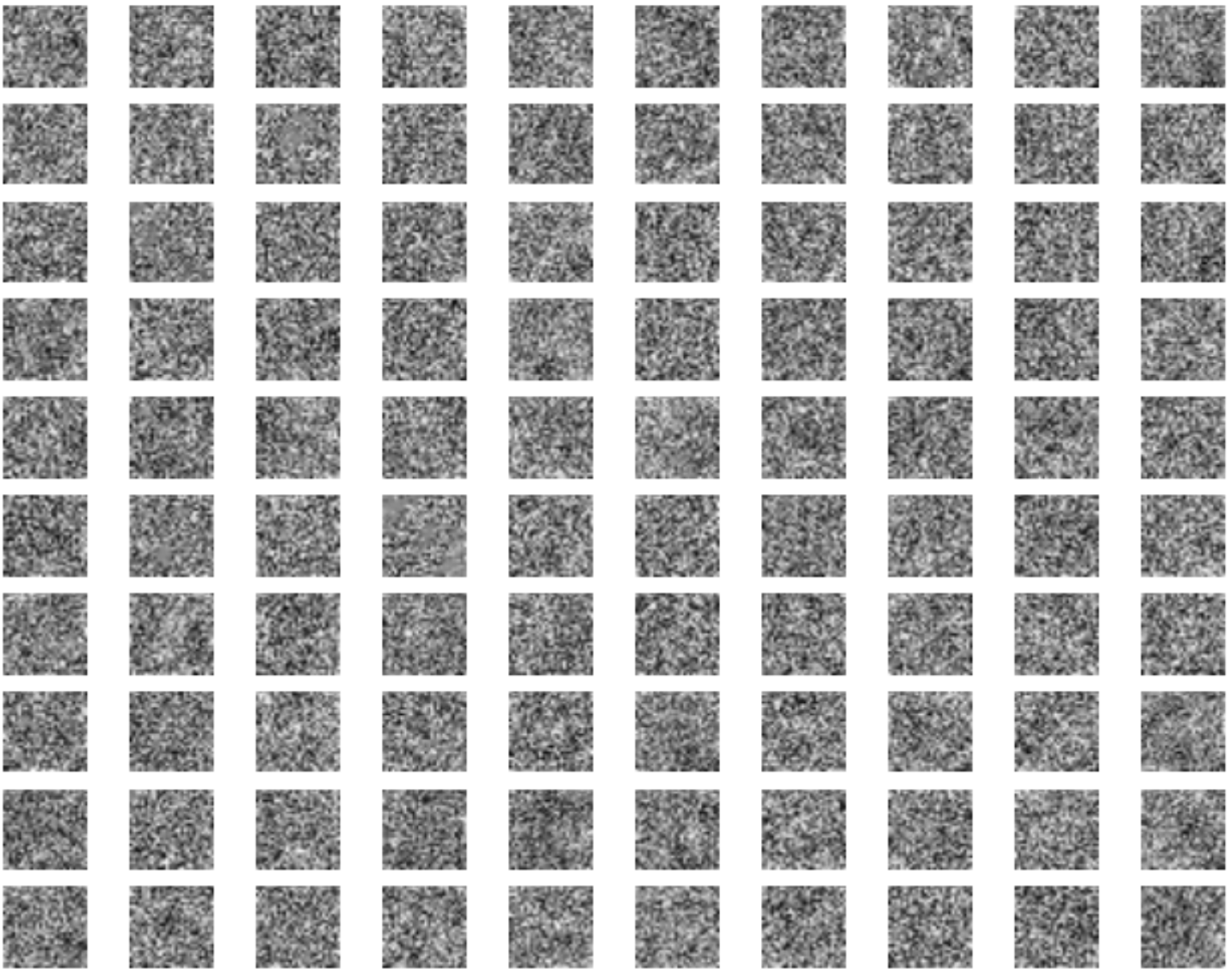

Figure 21: Weights at 25%

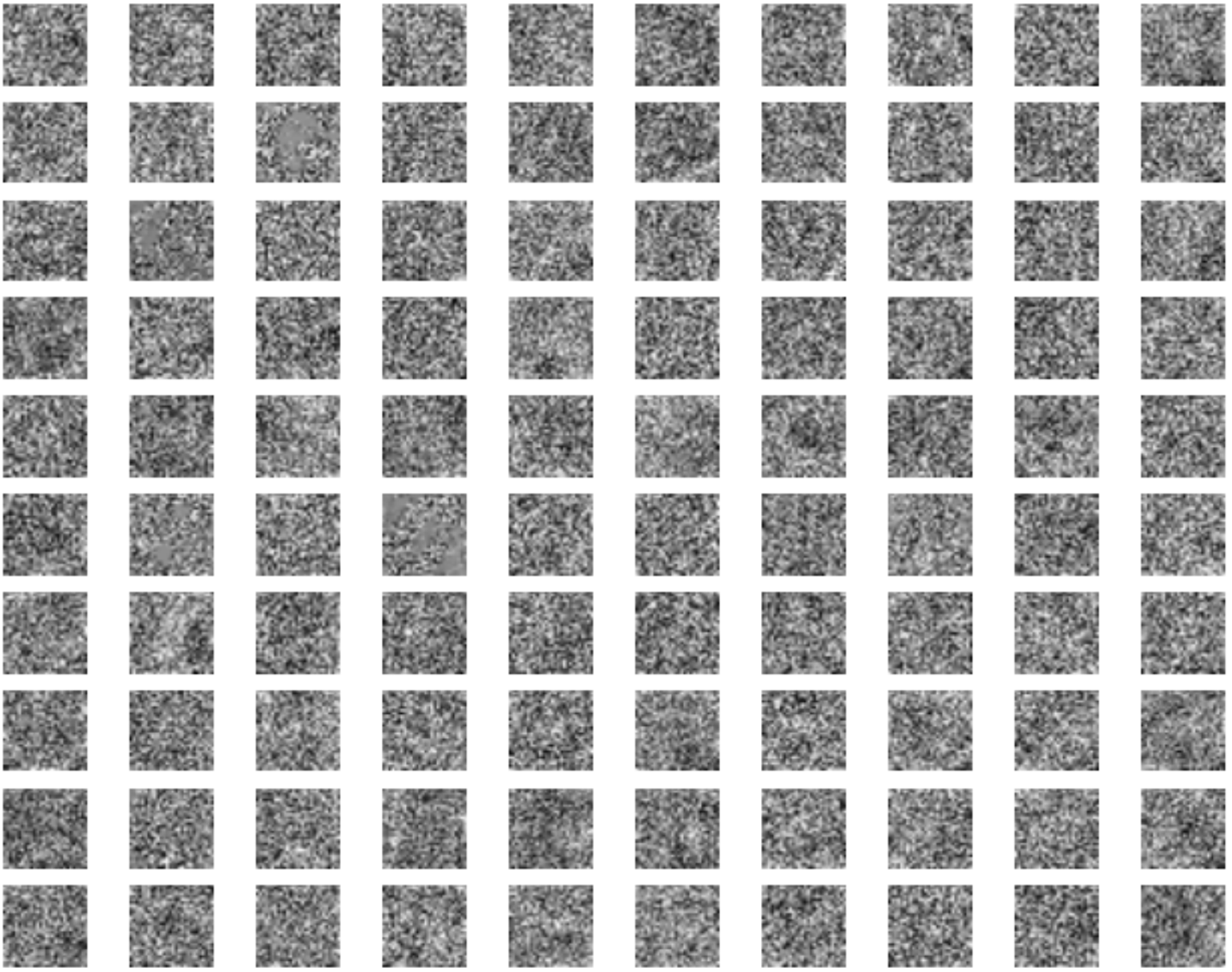

Figure 22: Weights at 50%

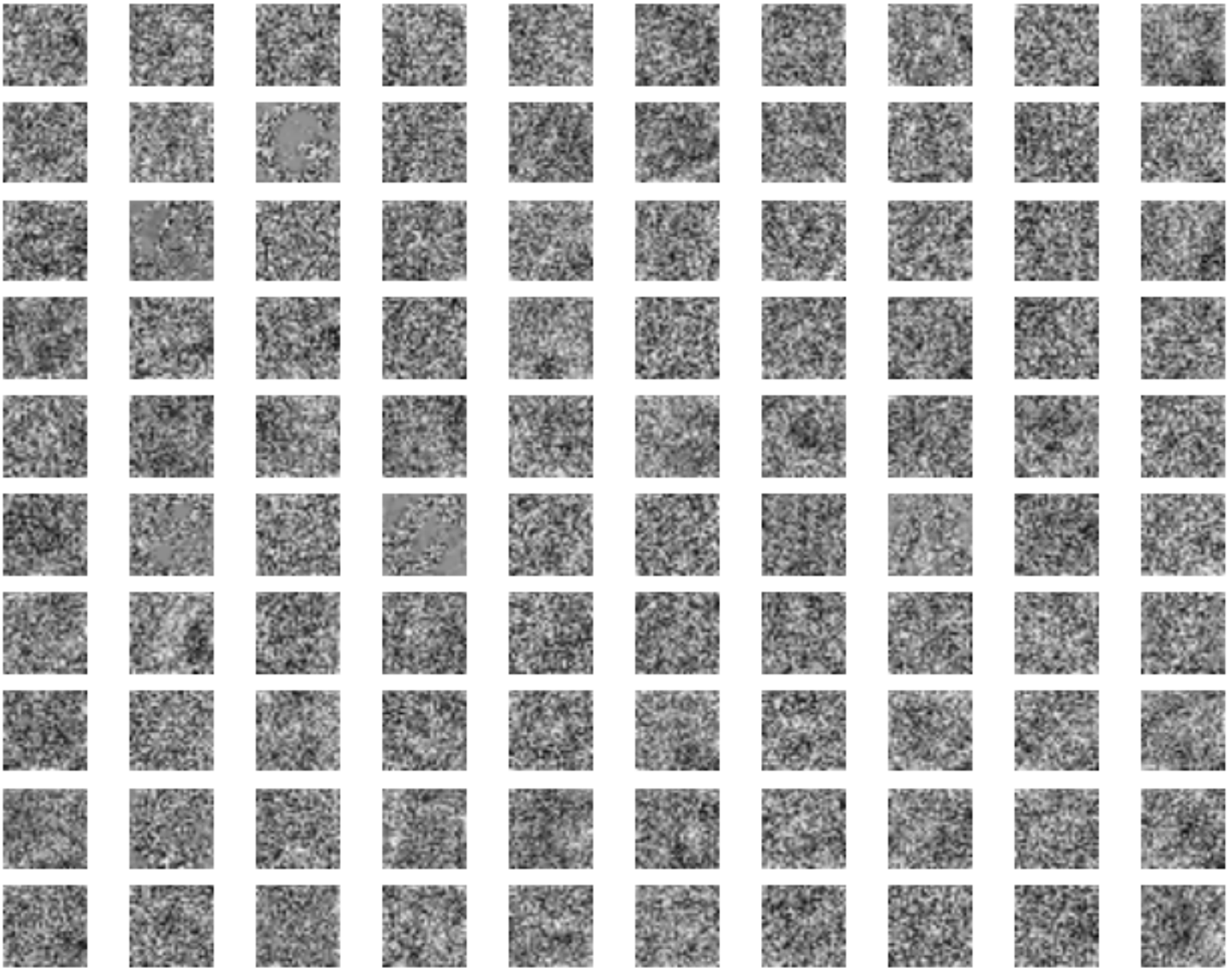

Figure 23: Weights at 75%

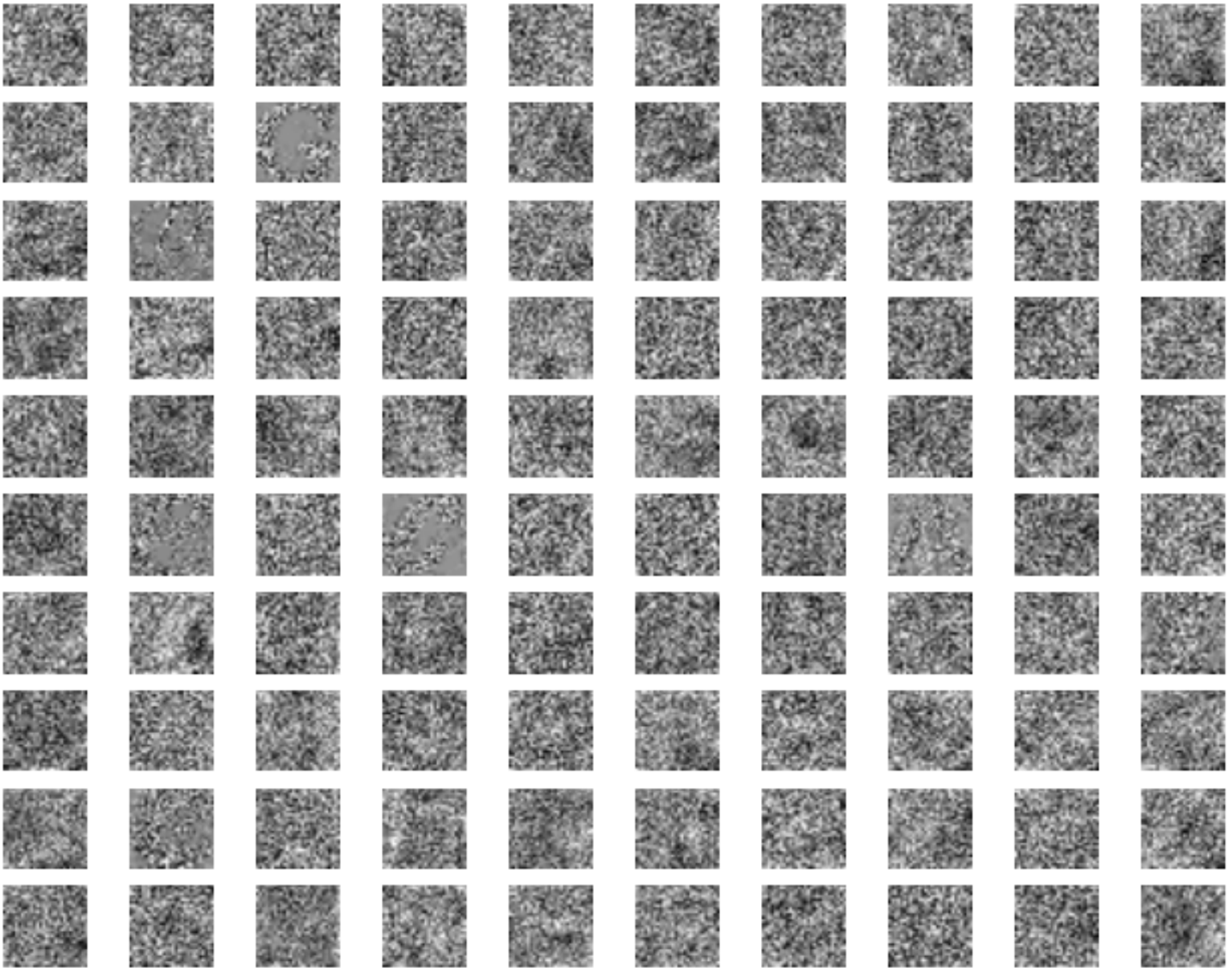

Figure 24: Weights at 100%

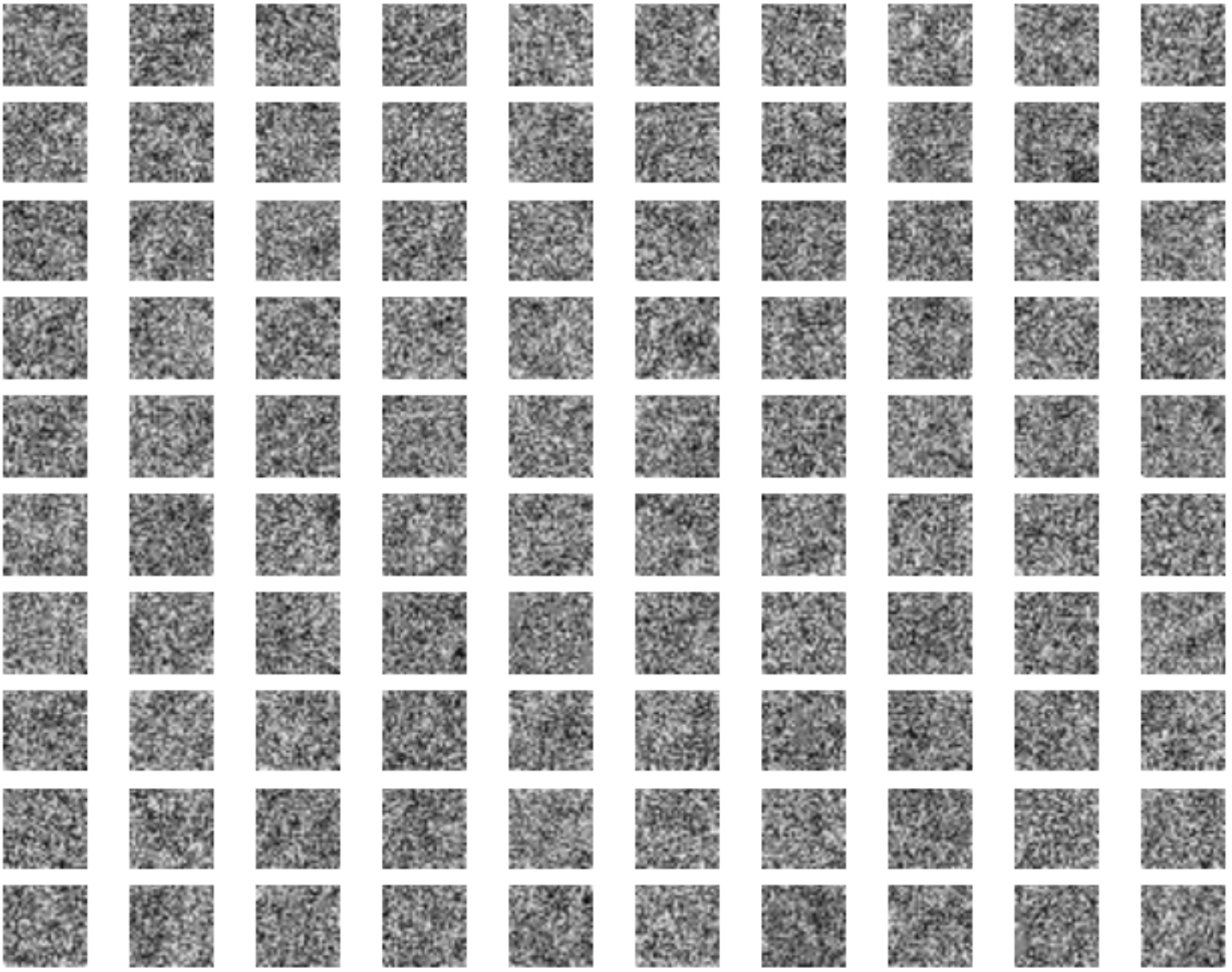

Figure 25: Weights at 25% with dropout

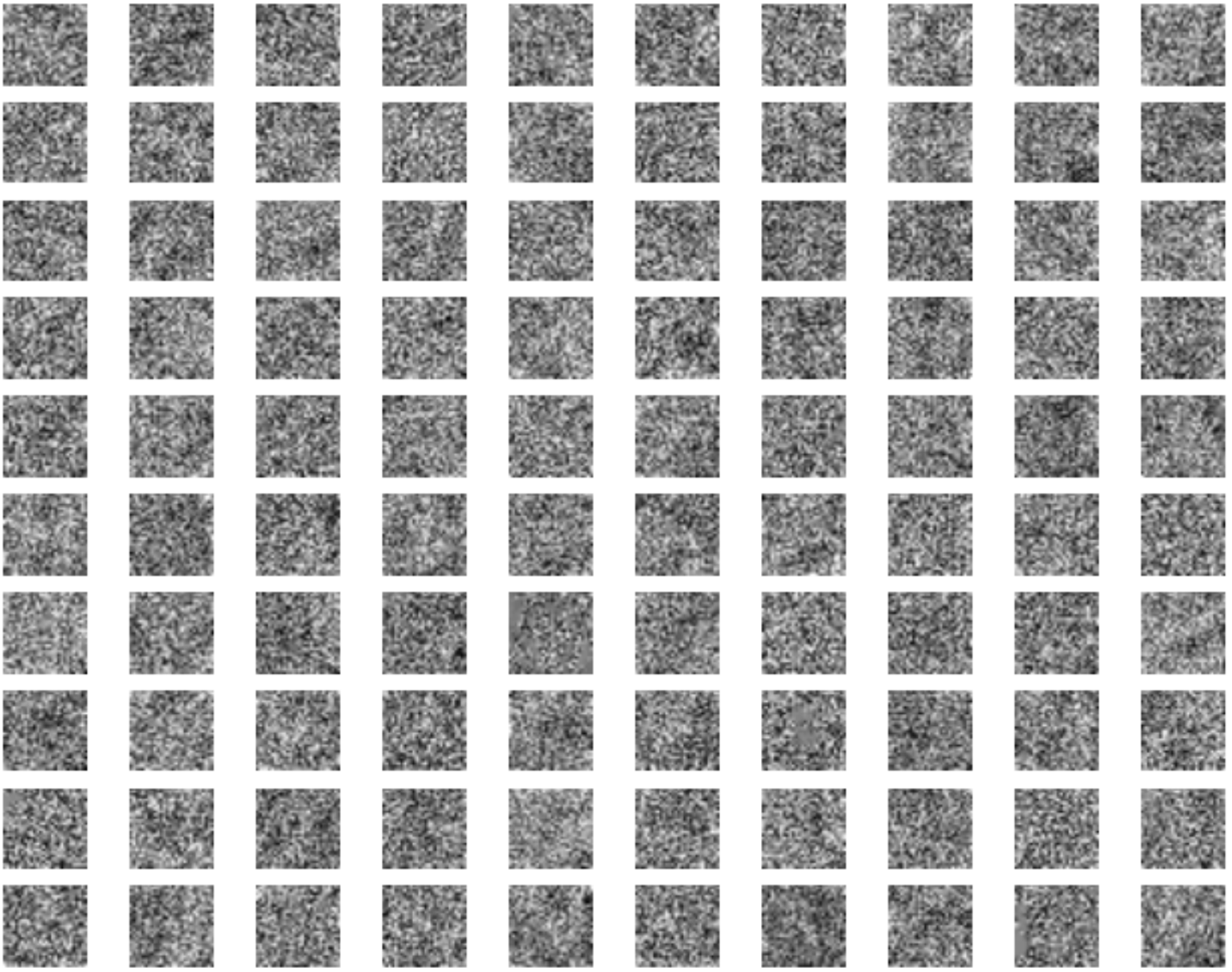

Figure 26: Weights at 50% with dropout

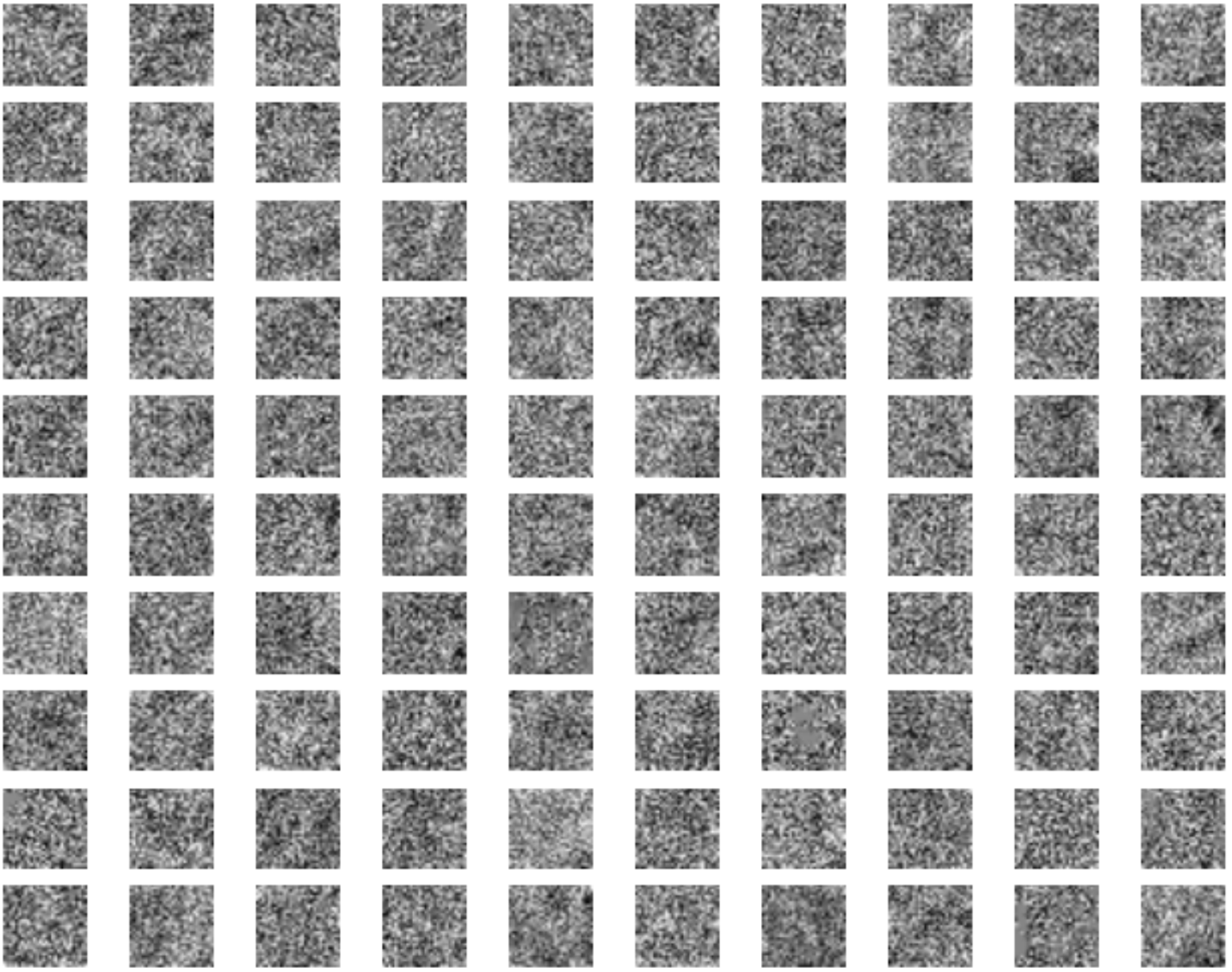

Figure 27: Weights at 75% with dropout

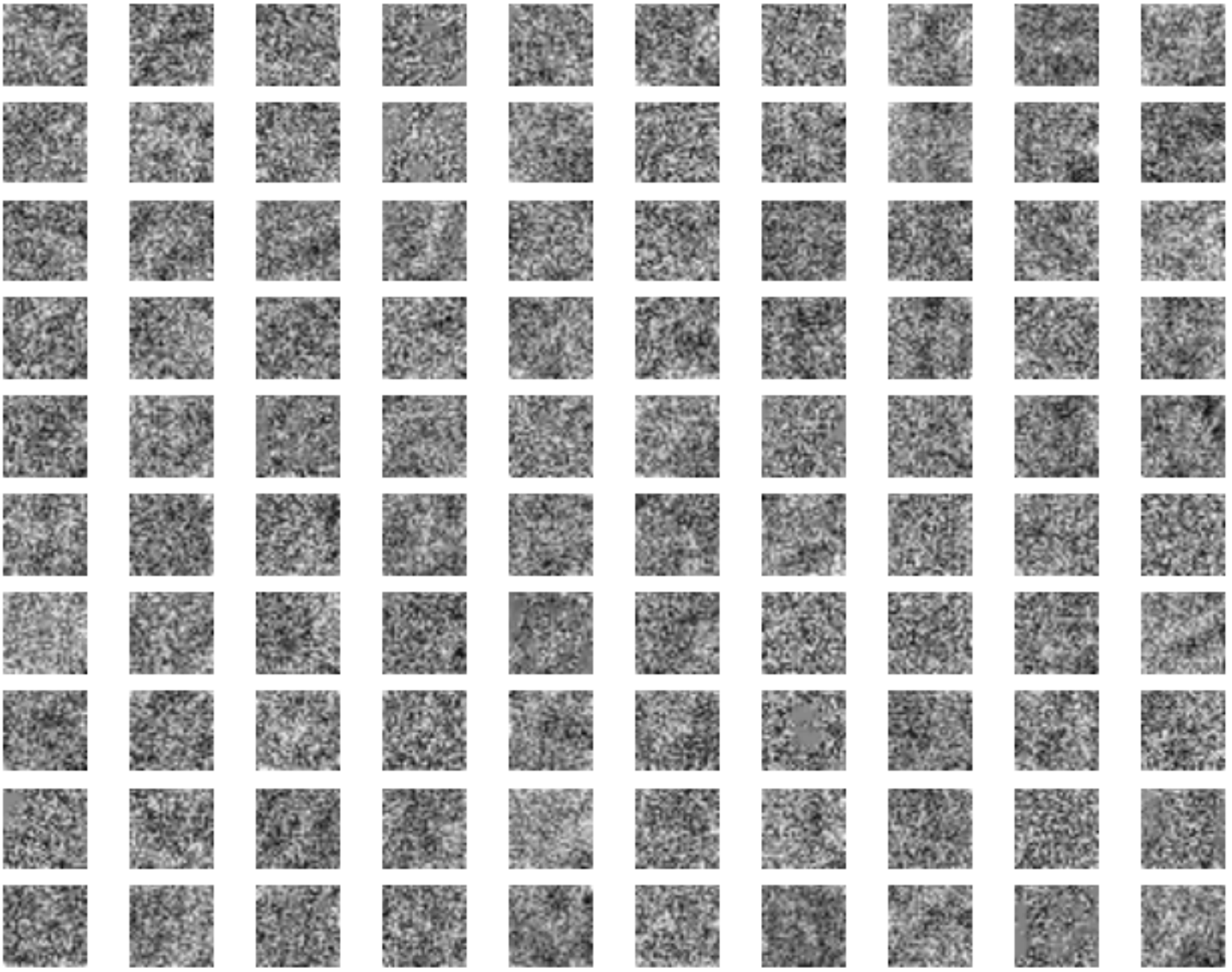

Figure 28: Weights at 100% with dropout

The weights change less as the training progress. It make sense since the gradient of error or the update term gets smaller as the algorithm gets closer to a local optimal. It looks like the weights generated from the dropout model have larger magnitude.

2.5

2.5.1

Table 1: Results

Layers	Learning Rate	Decay Rate	Hidden Units	Dropout	Validation Accuracy	Test Accuracy
2	0.0006824101	0.001315984	[245 176]	False	0.936	0.932
1	0.0029145386	0.0008541779	[448]	False	0.936	0.931
5	0.0040669442	0.0008015933	[277 484 479 439 365]	False	0.926	0.923
3	0.0005943762	0.0002017817	[212 360 129]	True	0.932	0.925
1	0.0013822459	0.0001547447	[115]	True	0.928	0.921

The best result is the first one, where we achieved 93.2% accuracy on test dataset.

2.5.2

By talking to other students, we found the following parameters give a slightly better result.

Table 2: Results from other students

Layers	Learning Rate	Decay Rate	Hidden Units	Dropout	Validation Accuracy	Test Accuracy
5	0.00292	0.00091	[496, 276, 486, 299, 376]	False	0.93	0.936

3 Code

```
def buildGraph_SGD(decay_rate, learning_rate):
    # Variable creation
    W = tf.Variable(tf.truncated_normal(shape=[784,1], stddev=0.5), name='weights')
    b = tf.Variable(0.0, name='biases')
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,1], name='target_y')

    # Graph definition
    y_predicted = tf.matmul(X,W) + b

    # Error definition
    error = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(targets =
    y_target, logits = y_predicted)) + \
            0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))

    acc = tf.reduce_mean(tf.cast(tf.equal(y_target, tf.cast((tf.sigmoid(y_predicted
    ) > 0.5), tf.float32)),tf.float32))
    # Training mechanism
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return W, b, X, y_target, y_predicted, error, train, acc

def getRandomBatch(trainData, trainTarget, size):
    idx = np.random.choice(trainData.shape[0], size, replace=False)
    return trainData[idx,:], trainTarget[idx,:]

decay_rate = 0.01
batch_size = 100
learning_rate = 0.01
#best learning rate 0.01
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_SGD(decay_rate,
    learning_rate)
```

```
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_acc_recorder = np.array([])
test_loss_recorder = np.array([])
test_acc_recorder = np.array([])
numIteration = 500
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_acc_recorder = np.append(train_acc_recorder, accuracy)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_acc_recorder = np.append(test_acc_recorder, test_acc)
    best_acc = max(best_acc, test_acc)
plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_acc_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), test_acc_recorder, 'b', label="Test")
plt.legend(loc="lower right")
#plt.axis([0,500, 0, 2])
plt.title("Accuracy VS number of updates for learning_rate = %0.3f"%(learning_rate)
    )
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 2: 1.1.1

```
def buildGraph_Adam(decay_rate, learning_rate):
    # Variable creation
    W = tf.Variable(tf.truncated_normal(shape=[784,1], stddev=0.5), name='weights')
    b = tf.Variable(0.0, name='biases')
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,1], name='target_y')

    # Graph definition
    y_predicted = tf.matmul(X,W) + b

    # Error definition
    error = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(targets =
    y_target, logits = y_predicted),
                                name='mean_squared_error') + \
            0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))

    acc = tf.reduce_mean(tf.cast(tf.equal(y_target, tf.cast((tf.sigmoid(y_predicted
    ) > 0.5), tf.float32)),tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return W, b, X, y_target, y_predicted, error, train, acc)

decay_rate = 0.01
batch_size = 500
learning_rate = 0.01
# training using SGD
```

```
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_SGD(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
sgd_train_loss_recorder = np.array([])
sgd_train_acc_recorder = np.array([])
sgd_test_loss_recorder = np.array([])
sgd_test_acc_recorder = np.array([])
numIteration = 500
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs, y_target: batch_ys})
    sgd_train_loss_recorder = np.append(sgd_train_loss_recorder, loss)
    sgd_train_acc_recorder = np.append(sgd_train_acc_recorder, accuracy)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target: testTarget})
    sgd_test_loss_recorder = np.append(sgd_test_loss_recorder, test_loss)
    sgd_test_acc_recorder = np.append(sgd_test_acc_recorder, test_acc)

# training using Adam
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
adam_train_loss_recorder = np.array([])
adam_train_acc_recorder = np.array([])
adam_test_loss_recorder = np.array([])
adam_test_acc_recorder = np.array([])
numIteration = 500
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs, y_target: batch_ys})
    adam_train_loss_recorder = np.append(adam_train_loss_recorder, loss)
    adam_train_acc_recorder = np.append(adam_train_acc_recorder, accuracy)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target: testTarget})
    adam_test_loss_recorder = np.append(adam_test_loss_recorder, test_loss)
    adam_test_acc_recorder = np.append(adam_test_acc_recorder, test_acc)
plt.plot(np.arange(numIteration), sgd_train_loss_recorder, 'g', label="SGD_train")
plt.plot(np.arange(numIteration), adam_train_loss_recorder, 'b', label="Adam_train")
plt.plot(np.arange(numIteration), sgd_test_loss_recorder, 'r', label="SGD_test")
plt.plot(np.arange(numIteration), adam_test_loss_recorder, 'k', label="Adam_test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()

plt.plot(np.arange(numIteration), sgd_train_acc_recorder, 'g', label="SGD")
plt.plot(np.arange(numIteration), adam_train_acc_recorder, 'b', label="Adam")
plt.plot(np.arange(numIteration), sgd_test_acc_recorder, 'r', label="SGD_test")
plt.plot(np.arange(numIteration), adam_test_acc_recorder, 'k', label="Adam_test")
plt.legend(loc="lower right")
#plt.axis([0,500, 0, 2])
plt.title("Accuracy VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()
```

Listing 3: 1.1.2

```
def buildGraph_linear(decay_rate, learning_rate):
    # Variable creation
    W = tf.Variable(tf.truncated_normal(shape=[784,1], stddev=0.5), name='weights')
```

```
    b = tf.Variable(0.0, name='biases')
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,1], name='target_y')

    # Graph definition
    y_predicted = tf.matmul(X,W) + b

    # Error definition
    error =  0.5 * tf.reduce_mean(tf.cast(tf.reduce_sum(tf.cast(tf.square(
    y_predicted - y_target), tf.float32),
                                                reduction_indices=1,
                                                name='squared_error'), tf.float32),
                                    name='mean_squared_error') + \
            0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))

    # y_predicted = tf.nn.l2_normalize(y_predicted, 0)
    acc = tf.reduce_mean(tf.cast(tf.equal(y_target, tf.cast(y_predicted > 0.5, tf.
    float32)),tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return W, b, X, y_target, y_predicted, error, train, acc
decay_rate = 0
batch_size = 500
learning_rate = 0.01
# training using SGD
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_linear(decay_rate,
    learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
lin_train_loss_recorder = np.array([])
lin_train_acc_recorder = np.array([])
lin_valid_loss_recorder = np.array([])
lin_valid_acc_recorder = np.array([])
lin_test_loss_recorder = np.array([])
lin_test_acc_recorder = np.array([])
numIteration = 500
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    lin_train_loss_recorder = np.append(lin_train_loss_recorder, loss)
    lin_train_acc_recorder = np.append(lin_train_acc_recorder, accuracy)
    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    lin_valid_loss_recorder = np.append(lin_valid_loss_recorder, valid_loss)
    lin_valid_acc_recorder = np.append(lin_valid_acc_recorder, valid_acc)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    lin_test_loss_recorder = np.append(lin_test_loss_recorder, test_loss)
    lin_test_acc_recorder = np.append(lin_test_acc_recorder, test_acc)

# training using Adam
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate,
    learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
adam_train_loss_recorder = np.array([])
adam_train_acc_recorder = np.array([])
adam_valid_loss_recorder = np.array([])
adam_valid_acc_recorder = np.array([])
adam_test_loss_recorder = np.array([])
adam_test_acc_recorder = np.array([])

for itr in range(numIteration):
```

```
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    adam_train_loss_recorder = np.append(adam_train_loss_recorder, loss)
    adam_train_acc_recorder = np.append(adam_train_acc_recorder, accuracy)
    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    adam_valid_loss_recorder = np.append(adam_valid_loss_recorder, test_loss)
    adam_valid_acc_recorder = np.append(adam_valid_acc_recorder, test_acc)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    adam_test_loss_recorder = np.append(adam_test_loss_recorder, test_loss)
    adam_test_acc_recorder = np.append(adam_test_acc_recorder, test_acc)
plt.plot(np.arange(numIteration), lin_train_loss_recorder, 'g', label="Lin_train")
plt.plot(np.arange(numIteration), adam_train_loss_recorder, 'b', label="Adam_train"
    )
plt.plot(np.arange(numIteration), lin_valid_loss_recorder, 'c', label="Lin_valid")
plt.plot(np.arange(numIteration), adam_valid_loss_recorder, 'm', label="Adam_valid"
    )
plt.plot(np.arange(numIteration), lin_test_loss_recorder, 'r', label="Lin_test")
plt.plot(np.arange(numIteration), adam_test_loss_recorder, 'k', label="Adam_test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), lin_train_acc_recorder, 'g', label="Lin_train")
plt.plot(np.arange(numIteration), adam_train_acc_recorder, 'b', label="Adam_train")
plt.plot(np.arange(numIteration), lin_valid_acc_recorder, 'c', label="Lin_valid")
plt.plot(np.arange(numIteration), adam_valid_acc_recorder, 'm', label="Adam_valid")
plt.plot(np.arange(numIteration), lin_test_acc_recorder, 'r', label="Lin_test")
plt.plot(np.arange(numIteration), adam_test_acc_recorder, 'k', label="Adam_test")
plt.legend(loc="lower right")
#plt.axis([0,500, 0, 2])
plt.title("Accuracy VS number of updates for learning_rate = %0.3f"%(learning_rate)
    )
plt.show()
```

Listing 4: 1.1.3

```
def dense_to_one_hot(labels_dense, num_classes=10):
    num_labels = labels_dense.shape[0]
    index_offset = np.arange(num_labels) * num_classes
    labels_one_hot = np.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot
def buildGraph_Adam(decay_rate, learning_rate):
    # Variable creation
    W = tf.Variable(tf.truncated_normal(shape=[784,10], stddev=0.5), name='weights'
    )
    b = tf.Variable(tf.zeros([10]), name='biases')
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,10], name='target_y')

    # Graph definition
    y_predicted = tf.matmul(X,W) + b

    # Error definition
    error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
    y_target, logits = y_predicted)) + \
            0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))

    correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
    acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
```

```
    return W, b, X, y_target, y_predicted, error, train, acc

decay_rate = 0.01
batch_size = 500
learning_rate = 0.01
#best learning rate 0.01
W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate,
    learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_acc_recorder = np.array([])
test_loss_recorder = np.array([])
test_acc_recorder = np.array([])
numIteration = 500
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_acc_recorder = np.append(train_acc_recorder, accuracy)
    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_acc_recorder = np.append(test_acc_recorder, test_acc)
    best_acc = max(best_acc, test_acc)
plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_acc_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), test_acc_recorder, 'b', label="Test")
plt.legend(loc="lower right")
#plt.axis([0,500, 0, 2])
plt.title("Accuracy VS number of updates for learning_rate = %0.3f"%(learning_rate)
    )
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 5: 1.2.3

```
# Parameters
learning_rate_list = [0.001, 0.01, 0.1]
batch_size = 100
display_step = 1
decay_rate = 3e-4
# Network Parameters
n_hidden_1 = 1000 # 1st layer number of features
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10

def buildGraph(decay_rate, learning_rate):
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,10], name='target_y')

    # Graph definition
    layer_1 = layerBlock(X, n_hidden_1, decay_rate)
    layer_1 = tf.nn.relu(layer_1)
    out_layer = layerBlock(layer_1, n_classes, decay_rate)
    y_predicted = out_layer

```

```
    # Error definition
    error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
    y_target, logits = y_predicted)) + \
                tf.add_n(tf.get_collection('losses'))

    correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
    acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return X, y_target, y_predicted, error, train, acc

decay_rate = 3e-4
batch_size = 100
learning_rate = 0.001
#best learning rate 0.001
tf.reset_default_graph()
X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_mis_recorder = np.array([])
valid_loss_recorder = np.array([])
valid_mis_recorder = np.array([])
test_loss_recorder = np.array([])
test_mis_recorder = np.array([])
numIteration = 500
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
    valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)

    best_acc = max(best_acc, test_acc)

plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 6: 2.2.2

```
decay_rate = 3e-4
batch_size = 100
learning_rate = 0.001
n_hidden_1 = 500
#best learning rate 0.001, number of hidden units 500
tf.reset_default_graph()
X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_mis_recorder = np.array([])
valid_loss_recorder = np.array([])
valid_mis_recorder = np.array([])
test_loss_recorder = np.array([])
test_mis_recorder = np.array([])
numIteration = 500
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
    valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)

    best_acc = max(best_acc, test_acc)

plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 7: 2.3.1

```
# Network Parameters
n_hidden_1 = 500
n_hidden_2 = 500
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10

def buildGraph(decay_rate, learning_rate):
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,10], name='target_y')

```

```
    # Graph definition
    layer_1 = layerBlock(X, n_hidden_1, decay_rate)
    layer_1 = tf.nn.relu(layer_1)
    layer_2 = layerBlock(layer_1, n_hidden_2, decay_rate)
    layer_2 = tf.nn.relu(layer_2)
    out_layer = layerBlock(layer_2, n_classes, decay_rate)
    y_predicted = out_layer

    # Error definition
    error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
    y_target, logits = y_predicted)) + \
                tf.add_n(tf.get_collection('losses'))

    correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
    acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return X, y_target, y_predicted, error, train, acc

decay_rate = 3e-4
batch_size = 100
learning_rate = 0.001
#best learning rate 0.001, number of hidden units 500
tf.reset_default_graph()
X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_mis_recorder = np.array([])
valid_loss_recorder = np.array([])
valid_mis_recorder = np.array([])
test_loss_recorder = np.array([])
test_mis_recorder = np.array([])
numIteration = 500
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
    valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)

    best_acc = max(best_acc, test_acc)

plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
```

```
plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 8: 2.3.2

```
# Network Parameters
n_hidden_1 = 500
n_hidden_2 = 500
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10
tf.reset_default_graph()
keep_prob = tf.placeholder(tf.float32) #dropout (keep probability)
def buildGraph(decay_rate, learning_rate):
    X = tf.placeholder(tf.float32, [None, 784], name='input_x')
    y_target = tf.placeholder(tf.float32, [None,10], name='target_y')

    # Graph definition
    layer_1 = layerBlock(X, n_hidden_1, decay_rate)
    layer_1 = tf.nn.relu(layer_1)
    dropout_1 = tf.nn.dropout(layer_1, keep_prob)

    layer_2 = layerBlock(dropout_1, n_hidden_2, decay_rate)
    layer_2 = tf.nn.relu(layer_2)
    dropout_2 = tf.nn.dropout(layer_2, keep_prob)

    out_layer = layerBlock(dropout_2, n_classes, decay_rate)
    y_predicted = out_layer

    # Error definition
    error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
    y_target, logits = y_predicted)) + \
                tf.add_n(tf.get_collection('losses'))

    correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
    acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # Training mechanism
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(loss=error)
    return X, y_target, y_predicted, error, train, acc

decay_rate = 3e-4
batch_size = 100
learning_rate = 0.001
#best learning rate 0.001, number of hidden units 500
X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_mis_recorder = np.array([])
valid_loss_recorder = np.array([])
valid_mis_recorder = np.array([])
test_loss_recorder = np.array([])
test_mis_recorder = np.array([])
numIteration = 20000
best_acc = 0
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss = sess.run([acc, error], feed_dict={X: batch_xs, y_target:
    batch_ys, keep_prob: 1.0})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

```

```
    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget, keep_prob: 1.0})
    valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
    valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget, keep_prob: 1.0})
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)

    if itr % 100 == 0:
        print("step %d, training accuracy %g, test set accuracy %g"%(itr, accuracy,
     test_acc))

    best_acc = max(best_acc, test_acc)
    sess.run(train, feed_dict={X: batch_xs, y_target: batch_ys, keep_prob: 0.5})


plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(
    learning_rate))
plt.show()

plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
plt.legend()
#plt.axis([0,500, 0, 2])
plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
plt.show()

print("Best accuracy in test set is %0.2f"%(best_acc))
```

Listing 9: 2.4.1

```
learning_rate = 0.001
X, y_target, y_predicted, error, train, acc, W1 = buildGraph(decay_rate,
    learning_rate)

init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
sess.run(init)
train_loss_recorder = np.array([])
train_mis_recorder = np.array([])
valid_loss_recorder = np.array([])
valid_mis_recorder = np.array([])
test_loss_recorder = np.array([])
test_mis_recorder = np.array([])

numIteration = 200
for itr in range(numIteration):
    batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
    accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys})
    train_loss_recorder = np.append(train_loss_recorder, loss)
    train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

    valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget})
    valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
    valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

    test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
    testTarget})
```

```
    test_loss_recorder = np.append(test_loss_recorder, test_loss)
    test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
    print("Iteration: %d, Train Acc: %0.3f, Valid Acc: %0.3f"%(itr, accuracy*100,
    valid_acc*100))

    if ((itr+1)%50 == 0):
      plt.figure()
      f, axarr = plt.subplots(40,25)
      W1_eval = W1.eval()
      for i in range(40):
    for j in range(25):
        axarr[i,j].imshow(W1_eval[:,i*25+j].reshape(28,28), cmap='gray')
        axarr[i,j].axis('off')
      plt.savefig("{}.png".format(itr));
```

Listing 10: 2.4.2

```
batch_size = 100
global_best_test_acc = 0

for numMod in range(100):

    learning_rate = np.exp(np.random.rand() * 3 - 7.5)
    numLayers = np.random.randint(5) + 1
    unitPerLayer = np.random.randint(100,500,numLayers)
    decay_rate = np.exp(np.random.rand() * 3 - 9)
    dropOut = np.random.randint(2)

    print("LR = %0.10f, NL = %d, DR = %0.10f, Dropout = %d" \
        %(learning_rate, numLayers, decay_rate, dropOut))
    print(unitPerLayer)

    #best learning rate 0.001, number of hidden units 500
   # tf.reset_default_graph()
    X, y_target, y_predicted, error, train, acc = buildGraph(numLayers,
    unitPerLayer, decay_rate, learning_rate, dropOut)

    init = tf.global_variables_initializer()

    sess = tf.InteractiveSession()
    sess.run(init)
    train_loss_recorder = np.array([])
    train_mis_recorder = np.array([])
    valid_loss_recorder = np.array([])
    valid_mis_recorder = np.array([])
    test_loss_recorder = np.array([])
    test_mis_recorder = np.array([])

    numIteration = 2000

    best_acc_valid = 0
    best_acc_test = 0

    for itr in range(numIteration):
        batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
        accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
    y_target: batch_ys, keep_prob: 0.8})
        train_loss_recorder = np.append(train_loss_recorder, loss)
        train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

        valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
    y_target: validTarget, keep_prob: 0.8})
        valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
        valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)

        test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData,
    y_target: testTarget, keep_prob: 0.8})
        test_loss_recorder = np.append(test_loss_recorder, test_loss)
```

```
48          test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
49          # print("Iteration: %d, Train Acc: %0.3f, Valid Acc: %0.3f"%(itr, accuracy
        *100, valid_acc*100))
50          best_acc_valid = max(best_acc_valid, valid_acc)
51          best_acc_test = max(best_acc_test, test_acc)
52
53      print("ValidAcc = %0.3f, testAcc = %0.3f\n" %(best_acc_valid, best_acc_test))
54
55      if(global_best_test_acc < best_acc_test):
56
57          global_best_test_acc = best_acc_test
58
59          b_lr = learning_rate
60          b_dr = decay_rate
61          b_do = dropOut
62          b_nl = numLayers
63          b_ul = unitPerLayer
64          b_va = best_acc_valid
65
66      print("BEST: LR = %0.10f, NL = %d, DR = %0.10f, Dropout = %d, ValidAcc = %0.3f,
          TestAcc = %0.3f"\
67      %(b_lr, b_nl, b_dr, b_do, b_va, global_best_test_acc))
68      print(b_ul)
```

Listing 11: 2.5.1