

ECE521 A2

Baiwu Zhang 1001127540
Daiqing Li 1000795357

February 2017

1

1.1

1.1.1

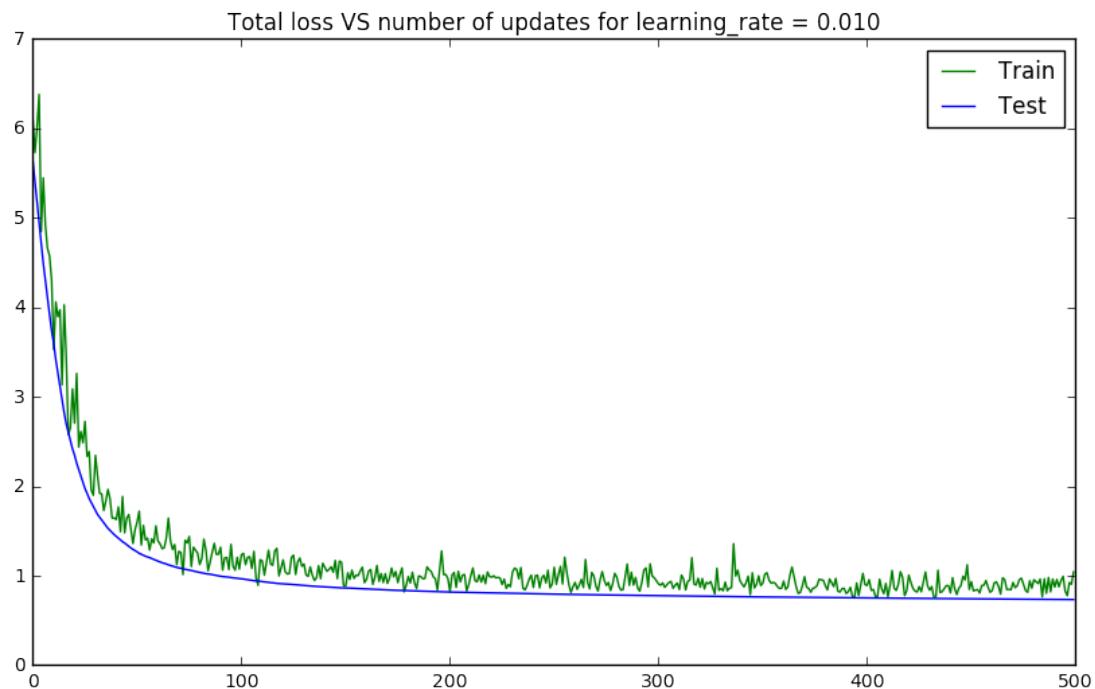


Figure 1: decay rate: 0.1, learning rate: 0.01, batch size: 100

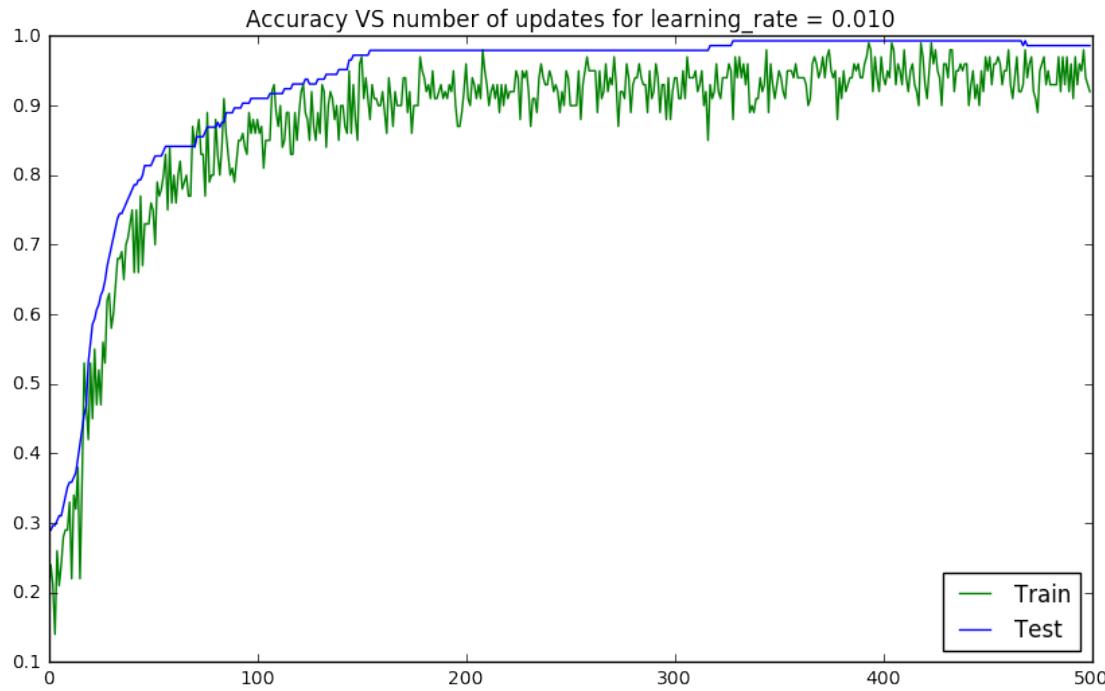


Figure 2: Best accuracy in test set is 0.99

1.1.2

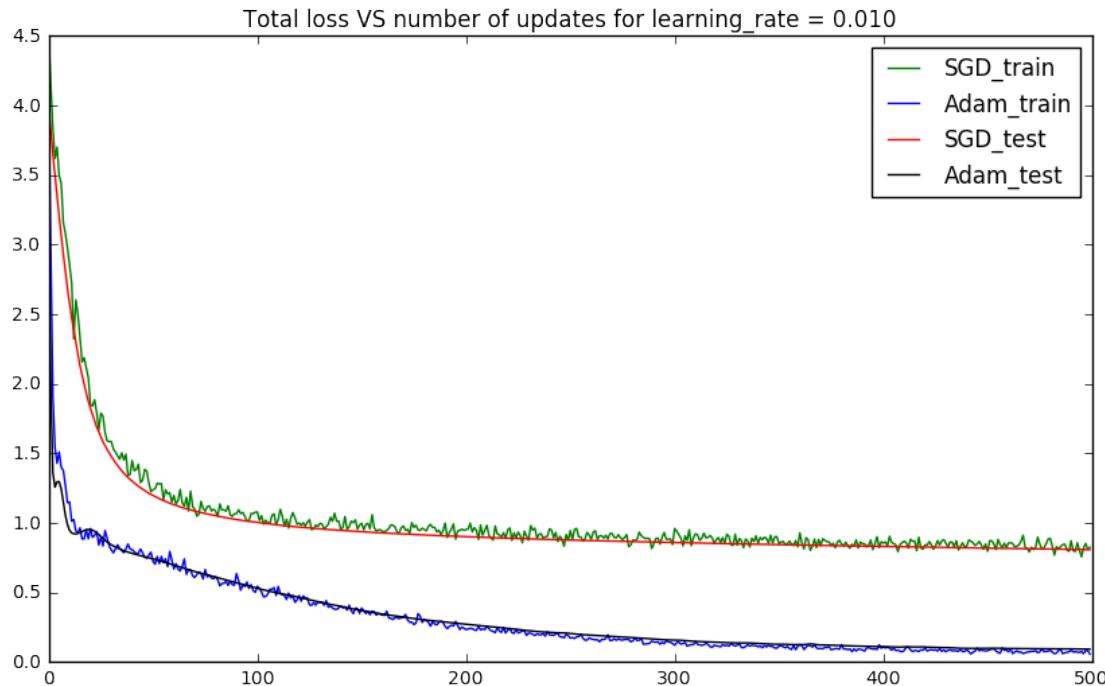


Figure 3: decay rate: 0.1, learning rate: 0.01, batch size: 500

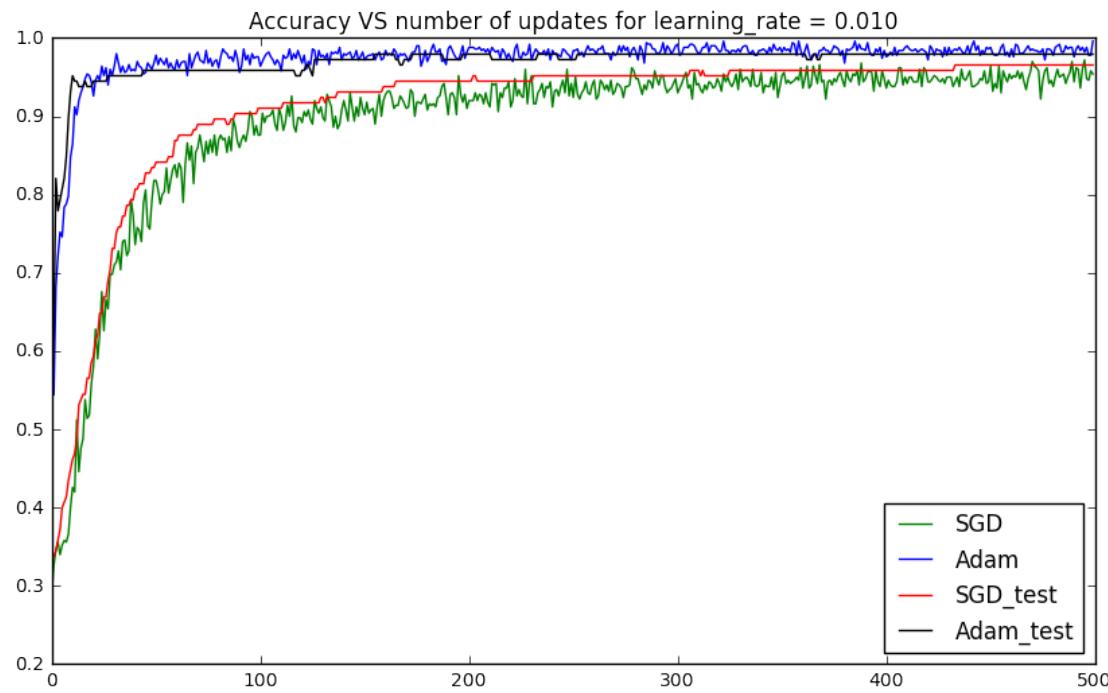


Figure 4: Best accuracy in test set is 0.99

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

1.1.3

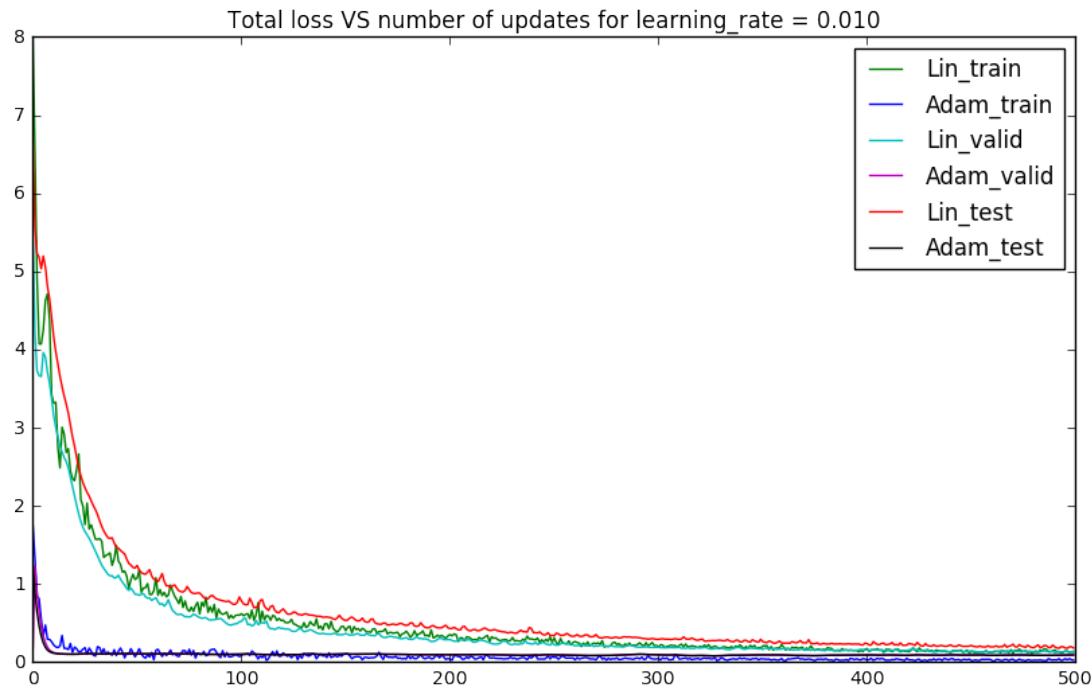


Figure 5: decay rate: 0.0, learning rate: 0.01, batch size: 500

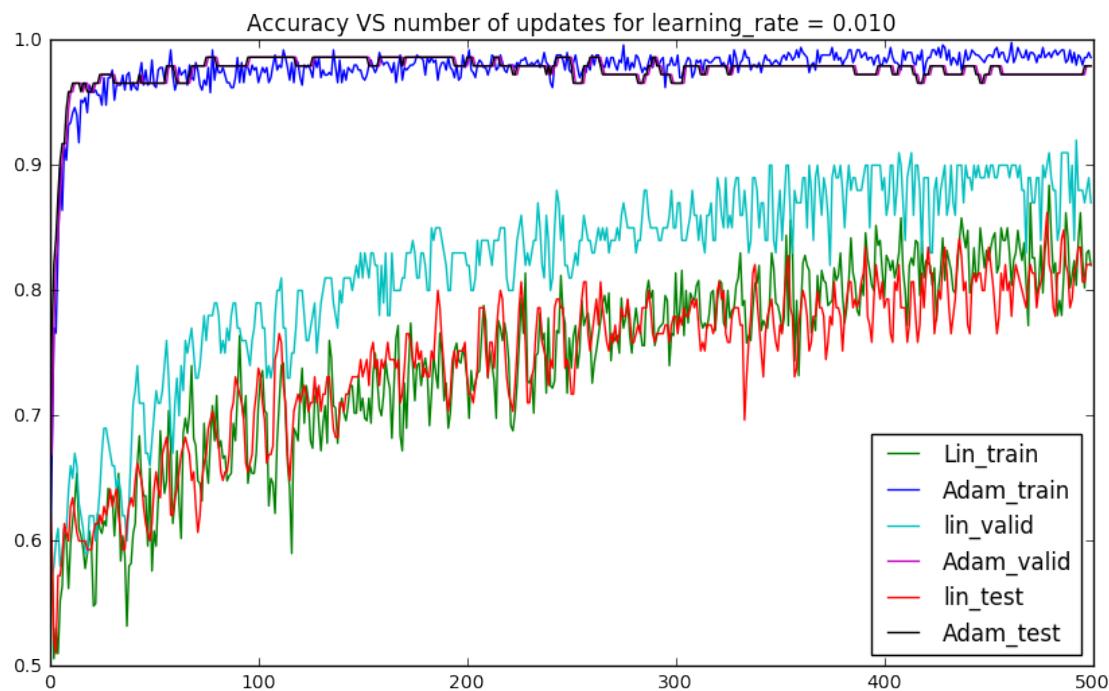


Figure 6: decay rate: 0.0, learning rate: 0.01, batch size: 500

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

$$\log(P(y|x, W)) = \log(\hat{y}(x)^y(1 - \hat{y}(x)^{1-y})) \quad (1)$$

$$= \log(\hat{y}(x)^y) + \log((1 - \hat{y}(x))^{1-y}) \quad (2)$$

$$= y\log(\hat{y}(x)) + (1 - y)\log((1 - \hat{y}(x))) \quad (3)$$

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

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

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

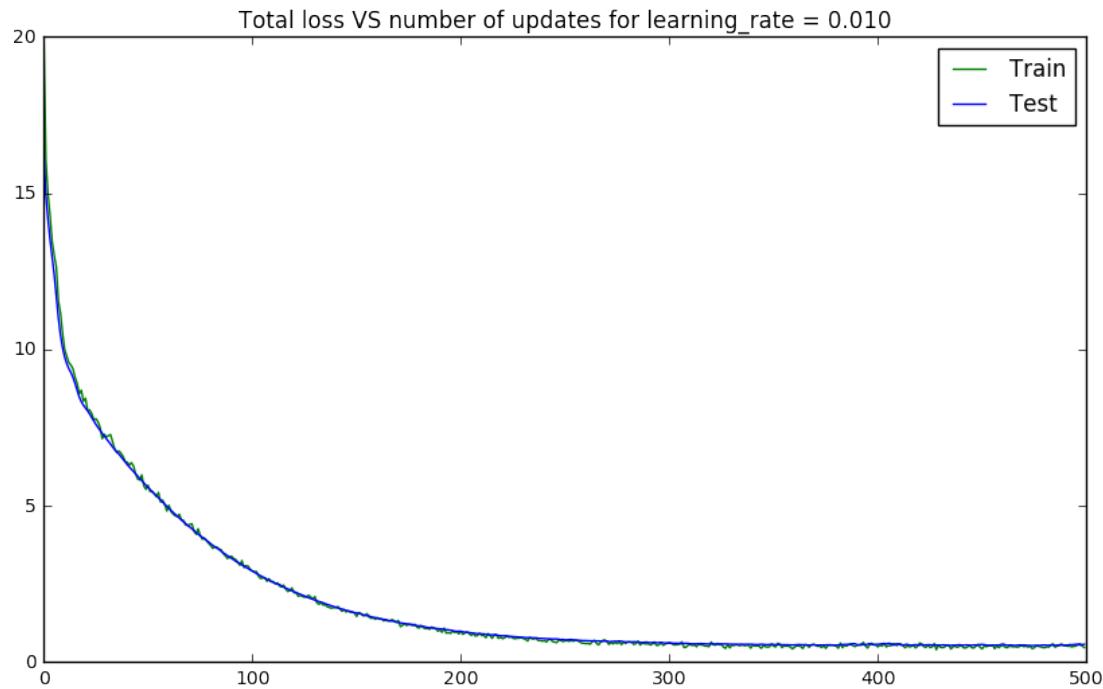


Figure 7: decay rate: 0.1, learning rate: 0.01, batch size: 500

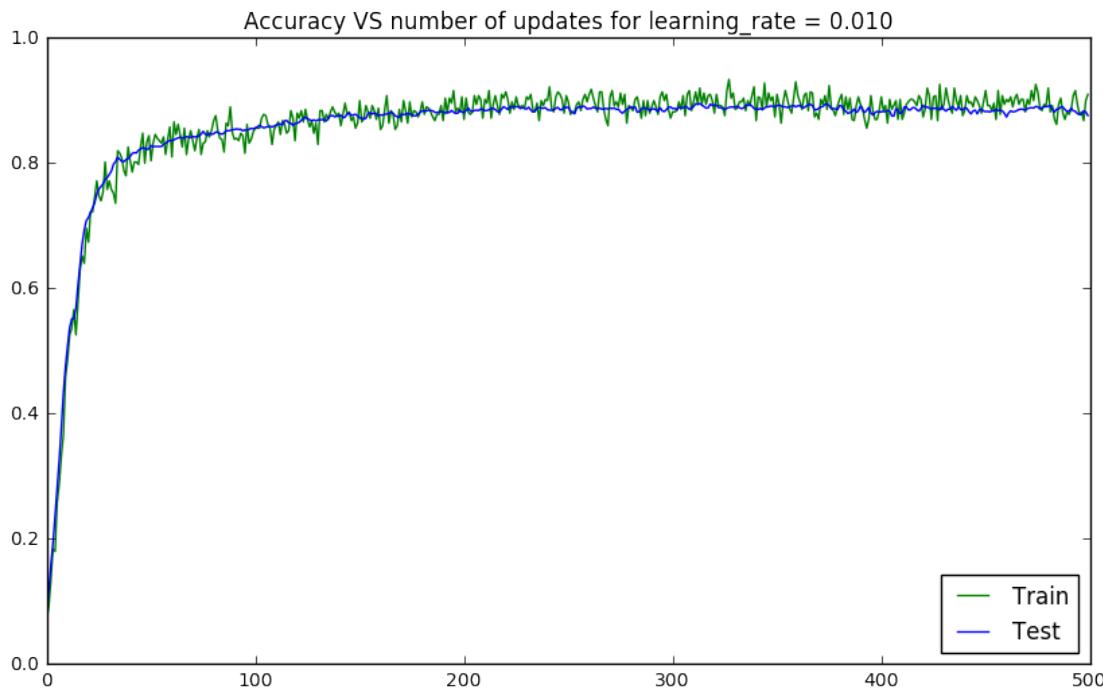


Figure 8: Best accuracy in test set is 0.90

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

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

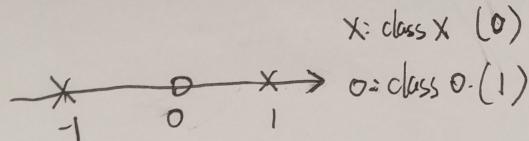
Since the data is linearly separable, the maximum value of the above equation should be 1 where each y_i receives probability of 1. In this case, the sigmoid function becomes a step function, and W has infinite magnitude. Adding weight decay can avoid this situation.

2.1.2

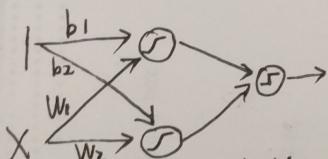
This is the opposite case of 2.1.1. When the data is not linearly separable, maximizing the product of conditional probability will not result in a step function, thus the magnitude of the weights will be finite.

2.1.3

Given dataset A as follows



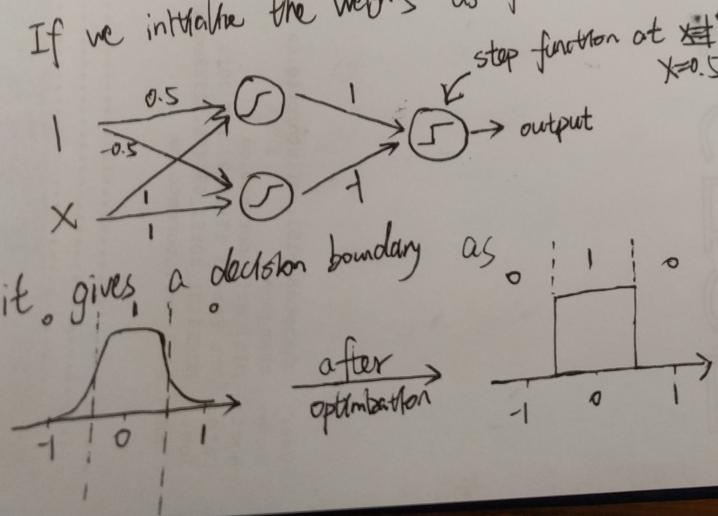
And a neural net B as follows.



It contains one hidden layer with two sigmoid nodes, and a step function on the output layer.

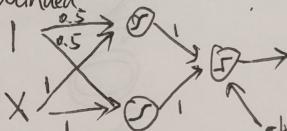
① Unbounded

If we initialize the weights as follows,



In this case, the cross-entropy loss can become 0 after optimization since all data points are classified perfectly. And the weights will be infinity as the decision boundary becomes step functions.

② Bounded



If we initialized the network symmetrically, the two sigmoid simply become a combination of two identical nodes. Since in back-propagation, they have the same update function, the weights will always be symmetrical. So it can never classify every data point correctly. Thus the cross-entropy will not be 0, and the decision boundary will not be a step function, and the magnitude of weights is finite.

Figure 9: 2.1.3

2.2

2.2.1

```

1 def layerBlock(x, n_units, decay_rate):
2     """
3         x: input tensor
4         n_units: number of hidden units in this layer
5     """
6     n_input = x.get_shape().as_list()[1]
7     XavierSTD = np.sqrt(3.0 / (n_input + n_units))
8     W = tf.Variable(tf.truncated_normal(shape=[n_input, n_units], stddev=XavierSTD),
9                     name="weights")

```

```

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

```

Listing 1: 2.2.1

2.2.2

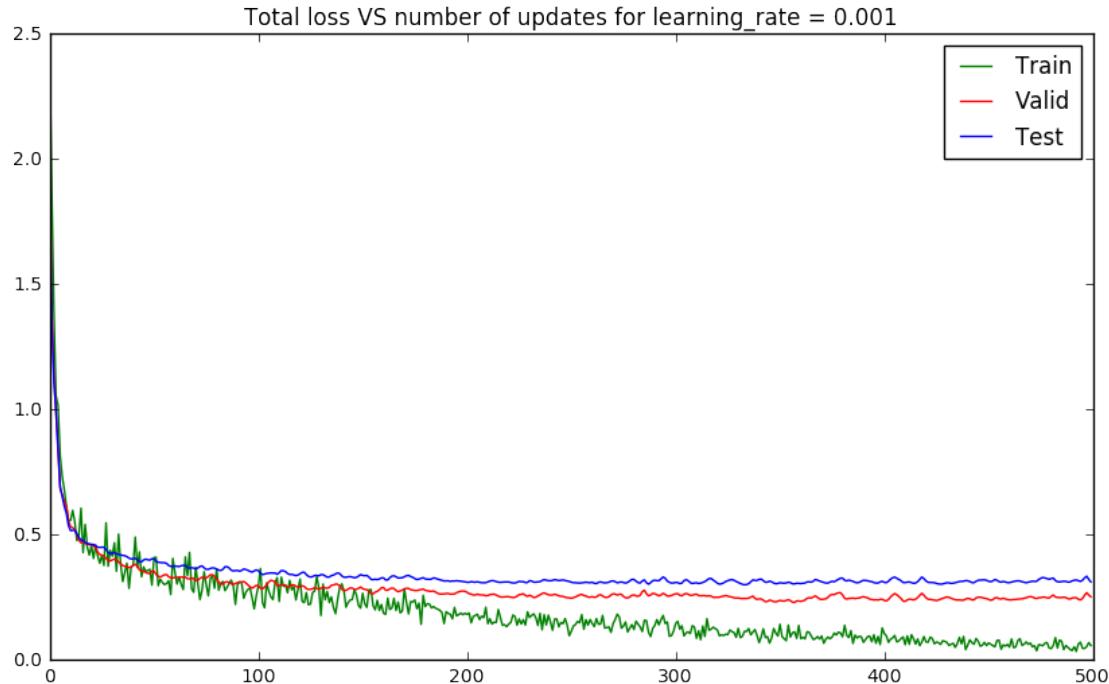


Figure 10: decay rate: 3e-4, learning rate: 0.001, batch size: 100

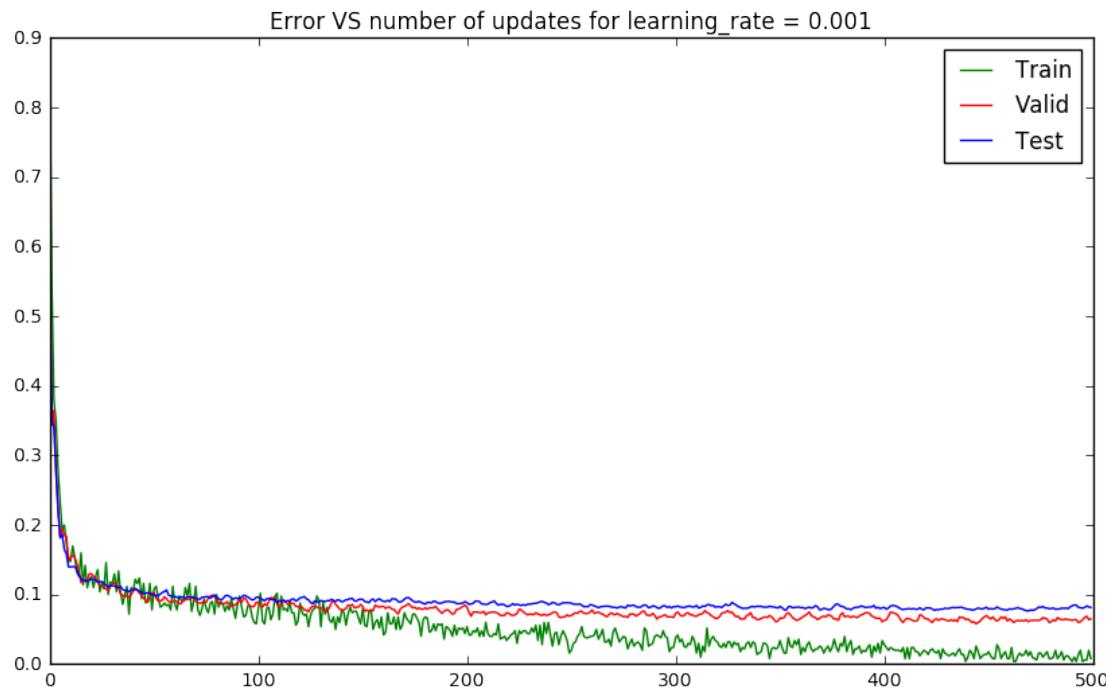


Figure 11: Best accuracy in test set is 0.92

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

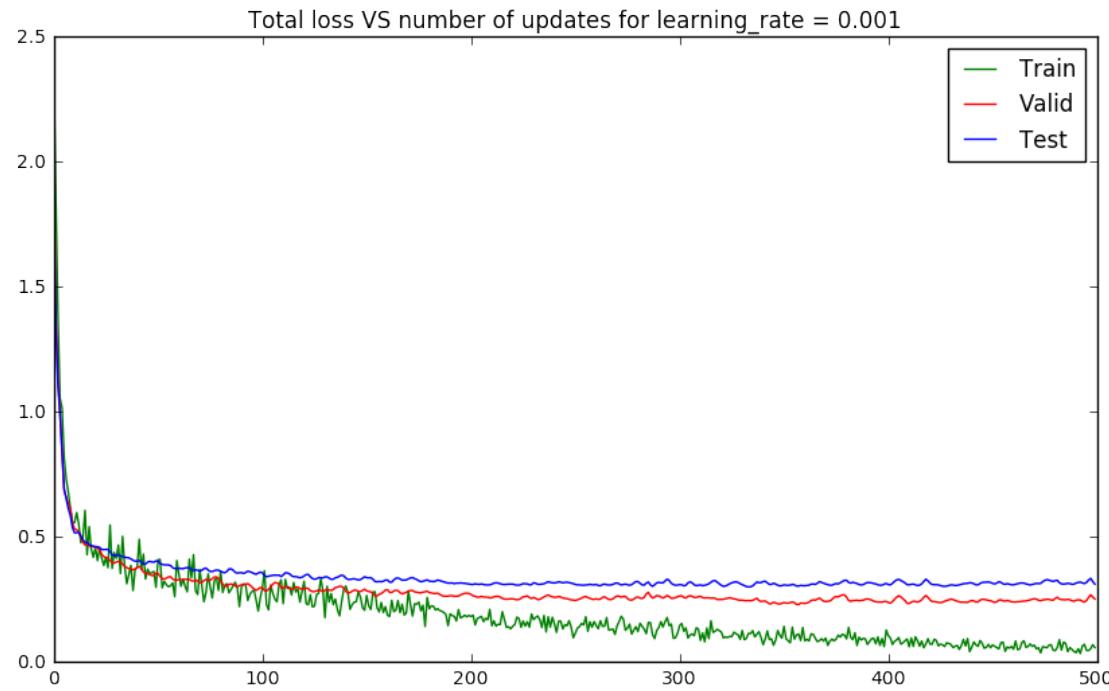


Figure 12: decay rate: 3e-4, learning rate: 0.001, batch size: 100

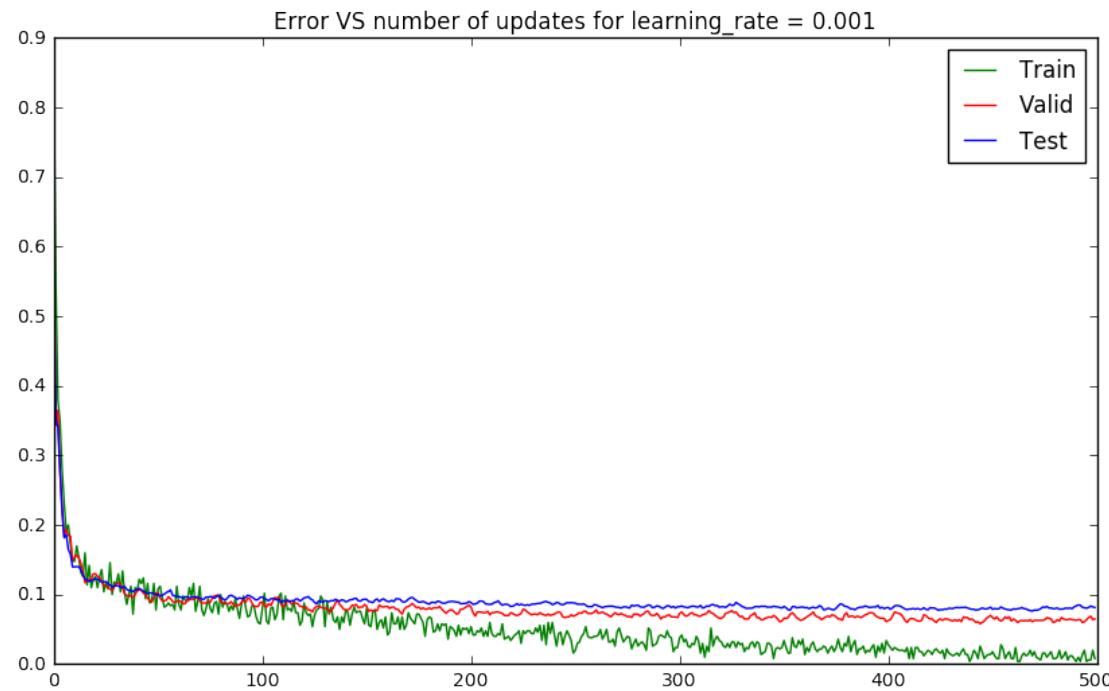


Figure 13: Best accuracy in test set is 0.92

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

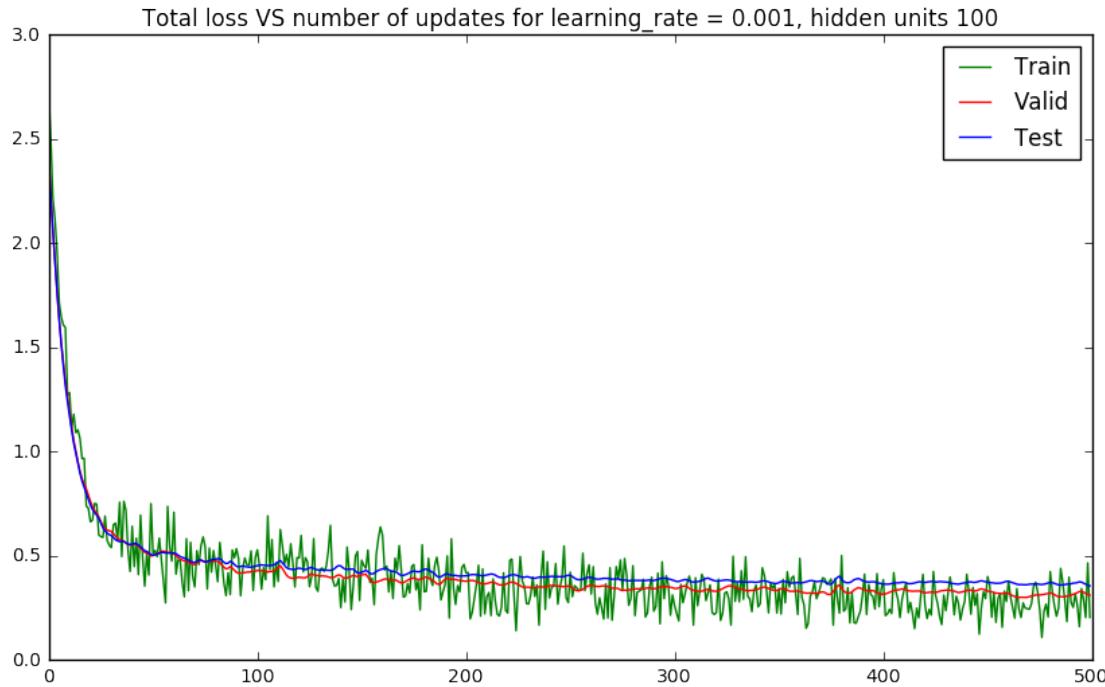


Figure 14: decay rate: 3e-4, learning rate: 0.001, batch size: 100

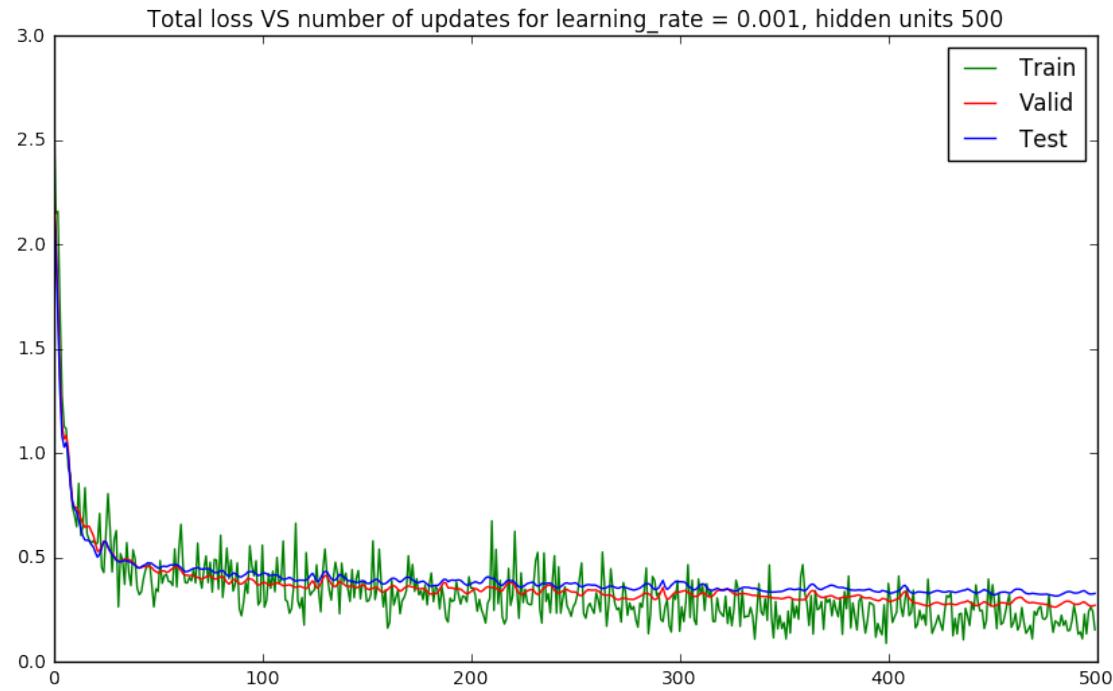


Figure 15: decay rate: 3e-4, learning rate: 0.001, batch size: 100

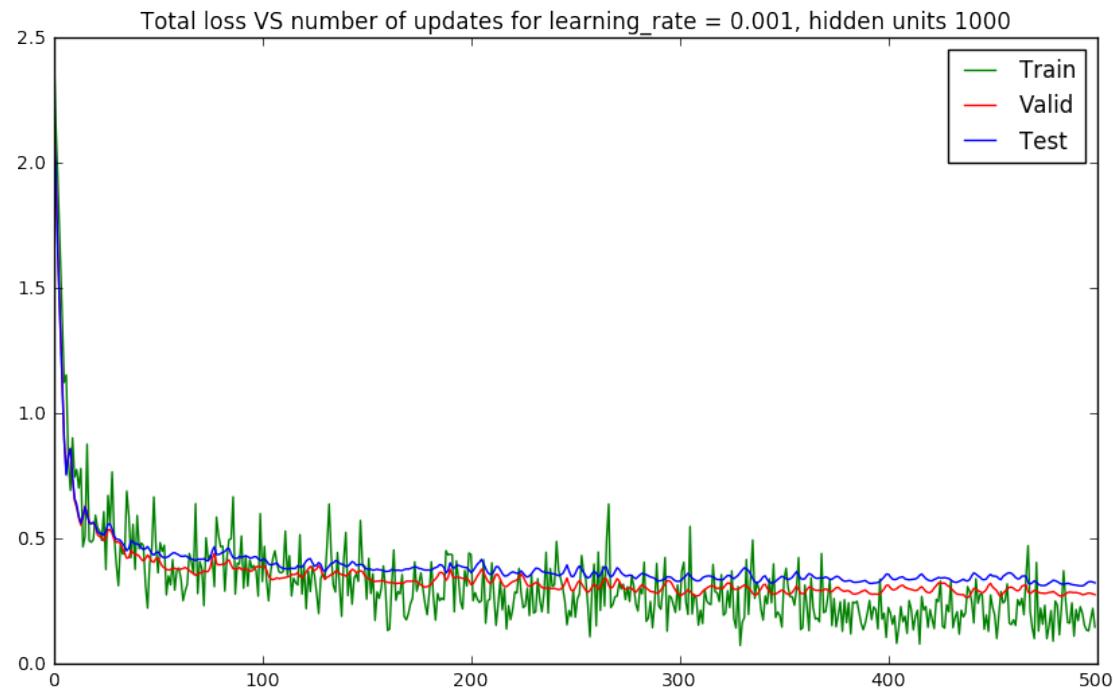


Figure 16: decay rate: 3e-4, learning rate: 0.001, batch size: 100

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

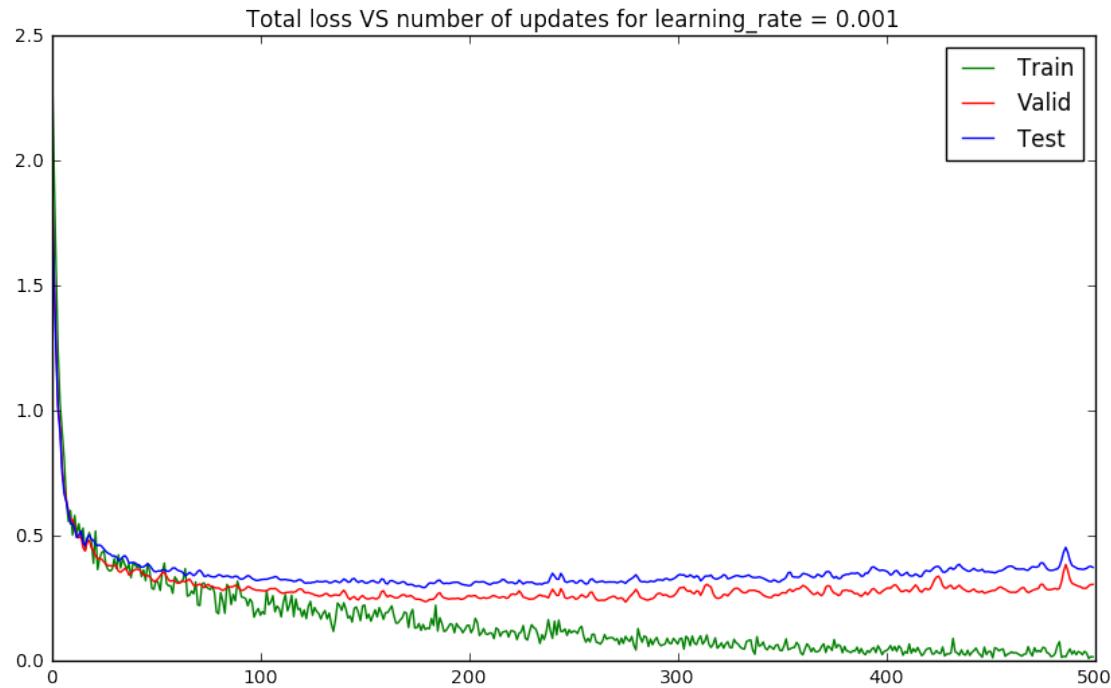


Figure 17: decay rate: 3e-4, learning rate: 0.001, batch size: 500

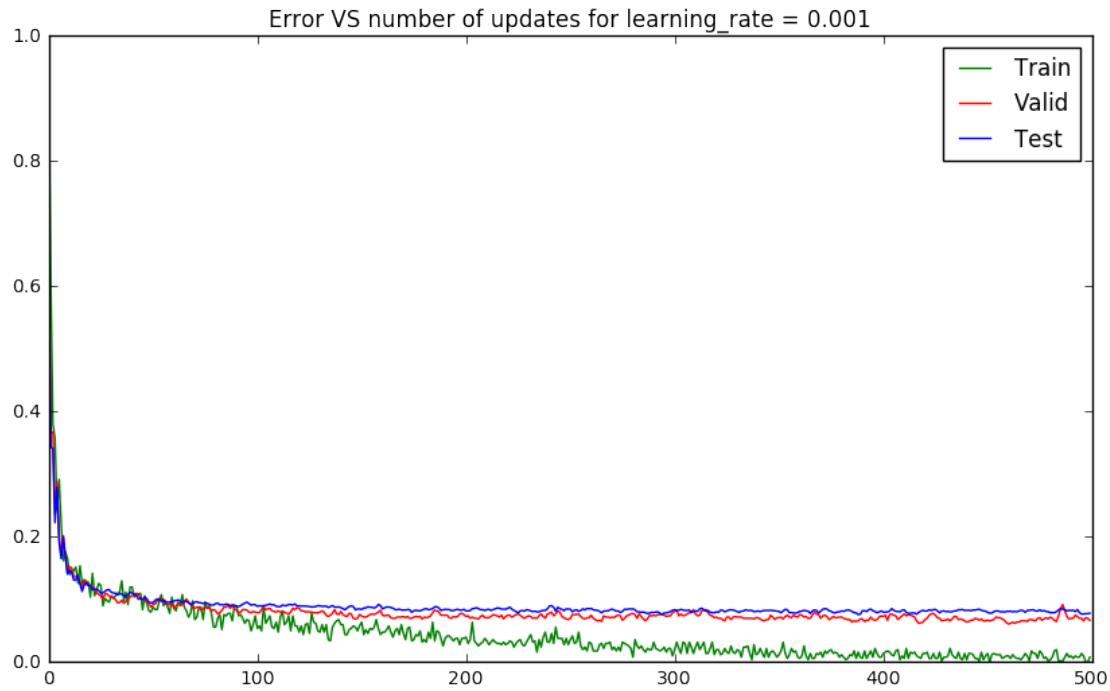


Figure 18: Best accuracy in test set is 0.92

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

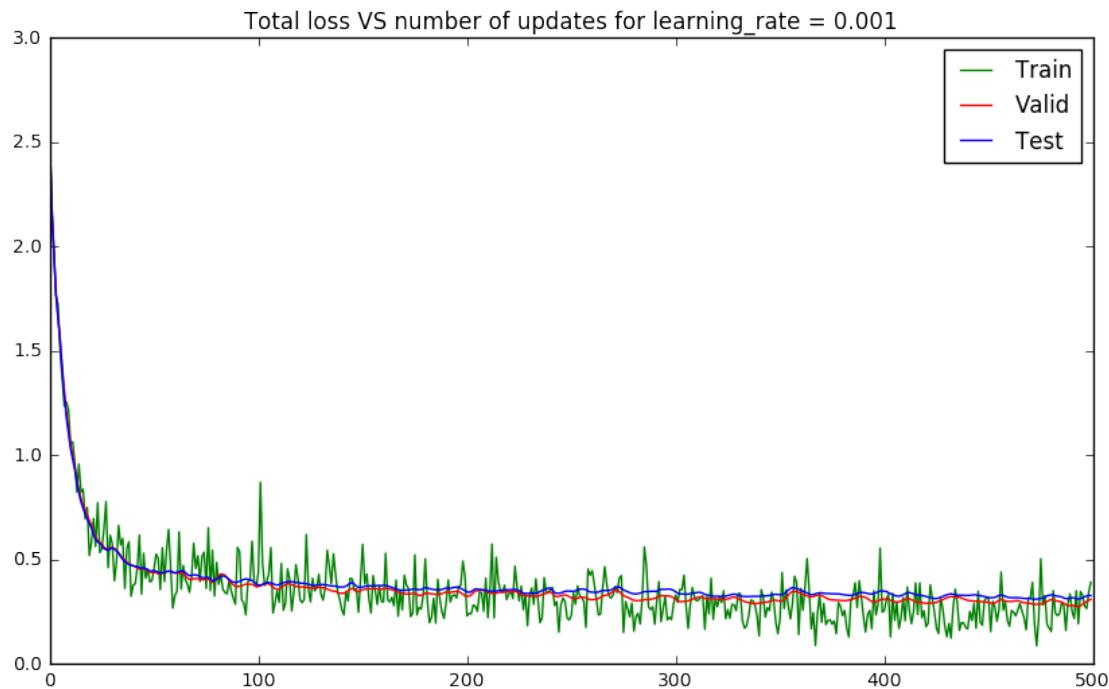


Figure 19: decay rate: 3e-4, learning rate: 0.001, batch size: 100

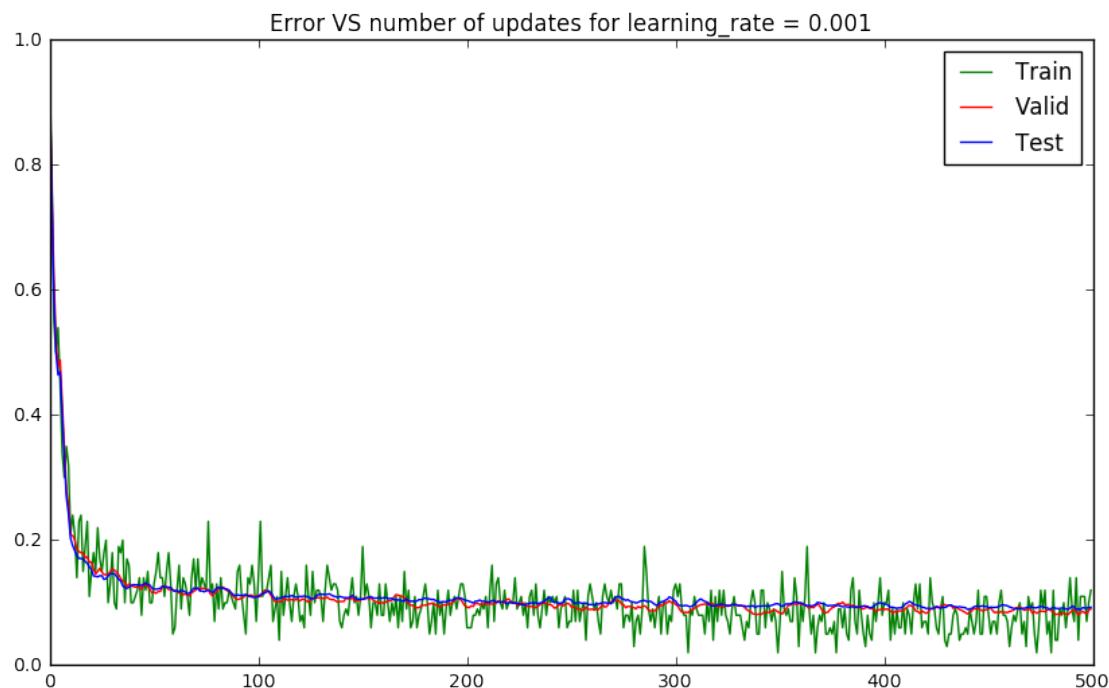


Figure 20: decay rate: 3e-4, learning rate: 0.001, batch size: 100

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

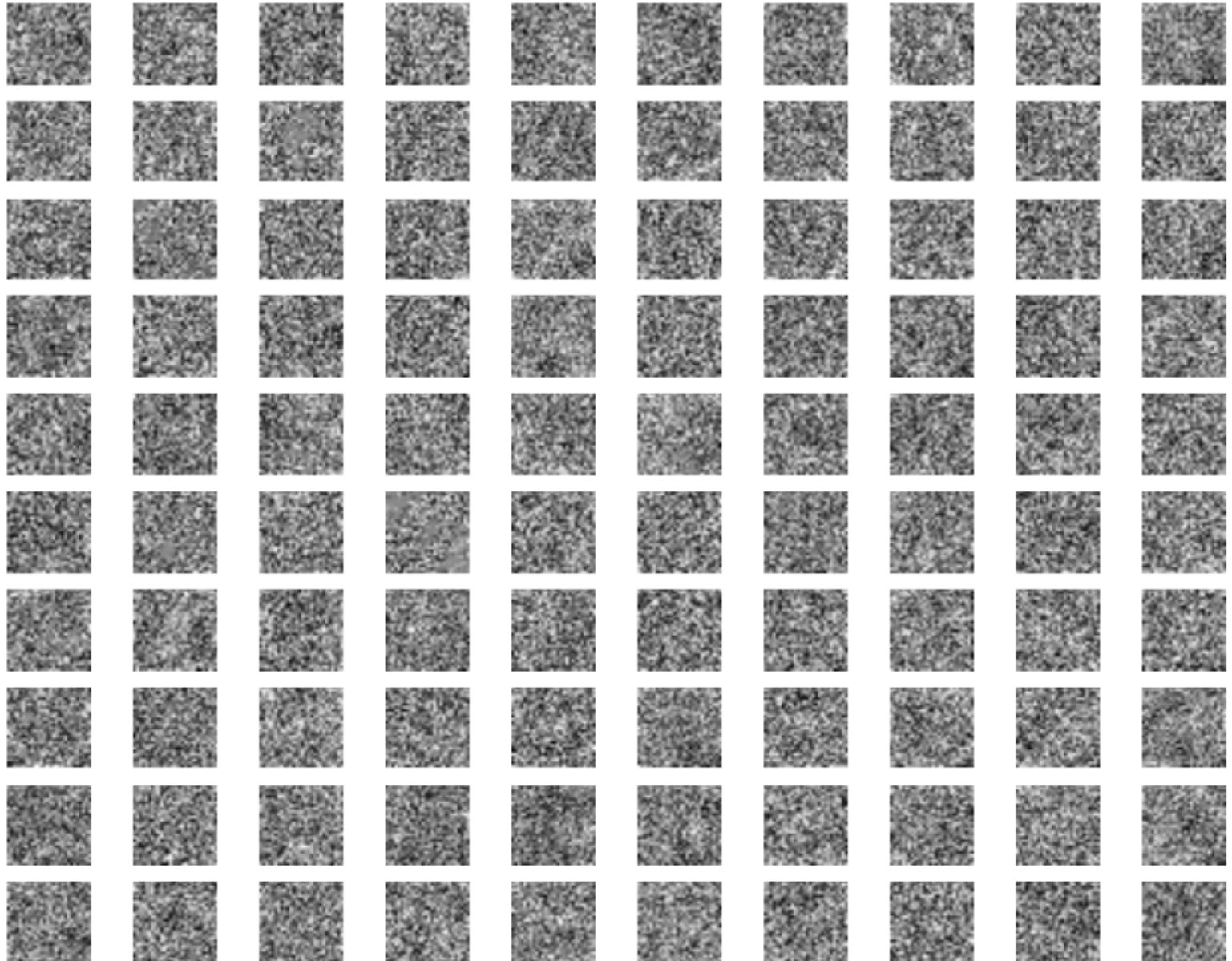


Figure 21: Weights at 25%

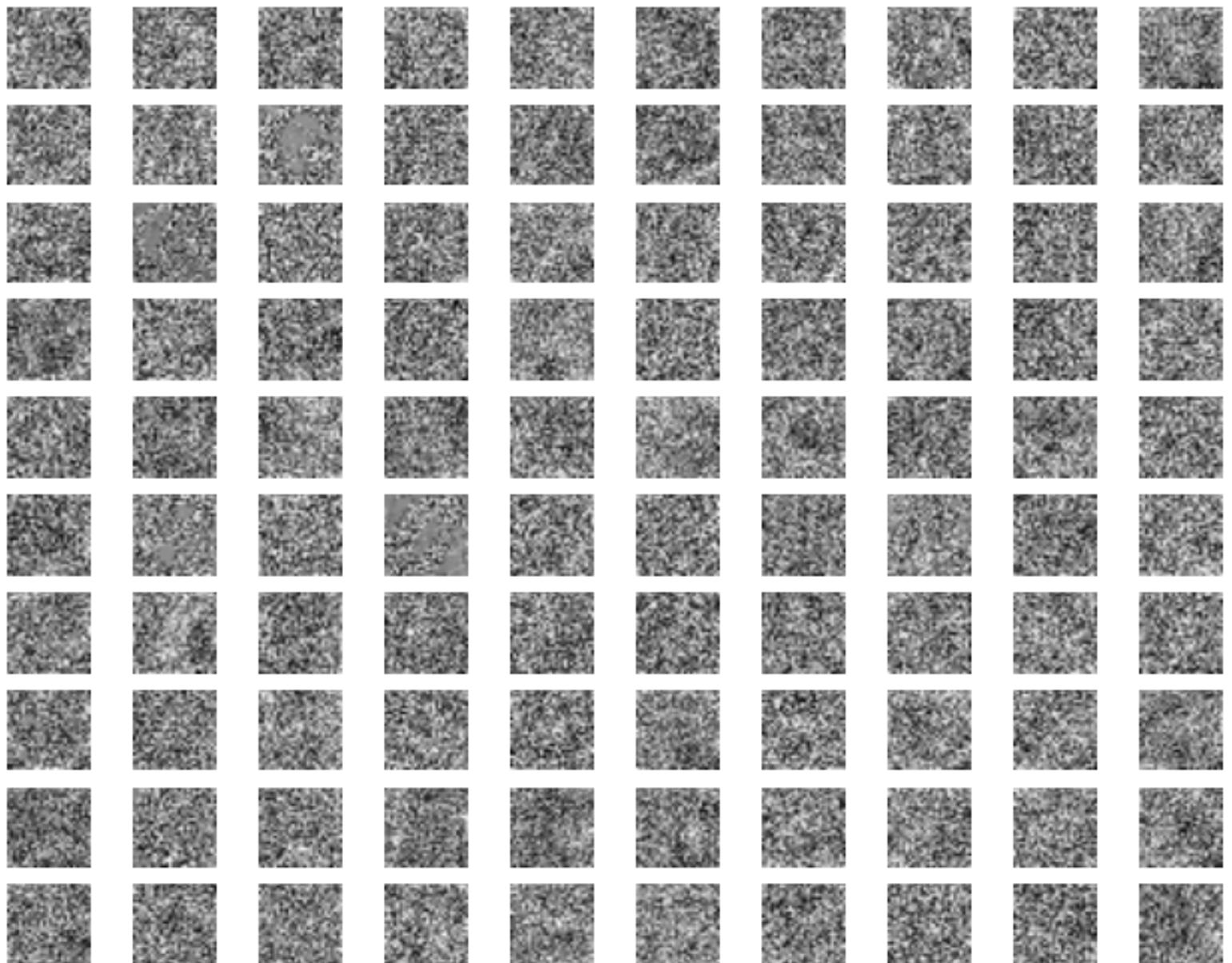


Figure 22: Weights at 50%

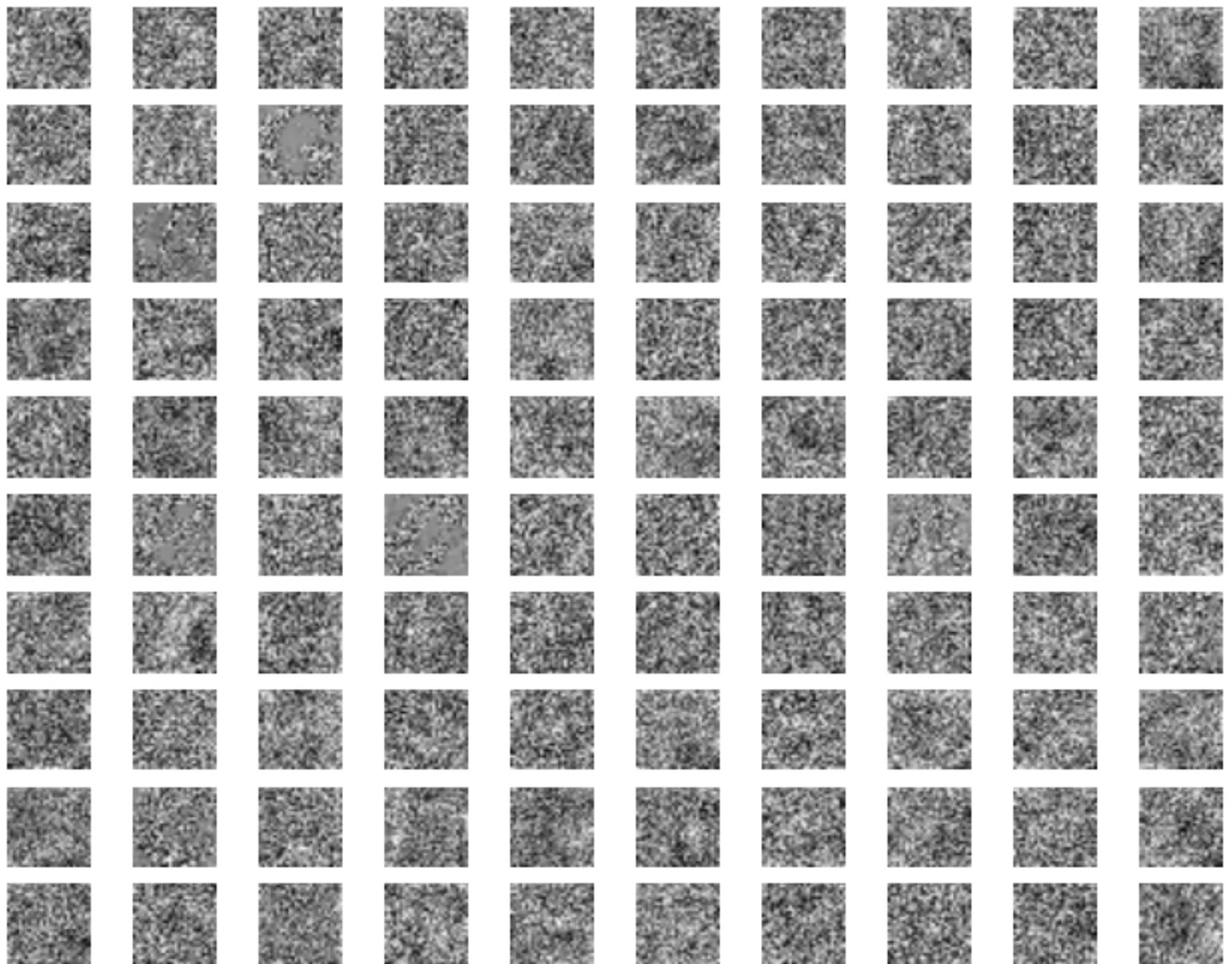


Figure 23: Weights at 75%

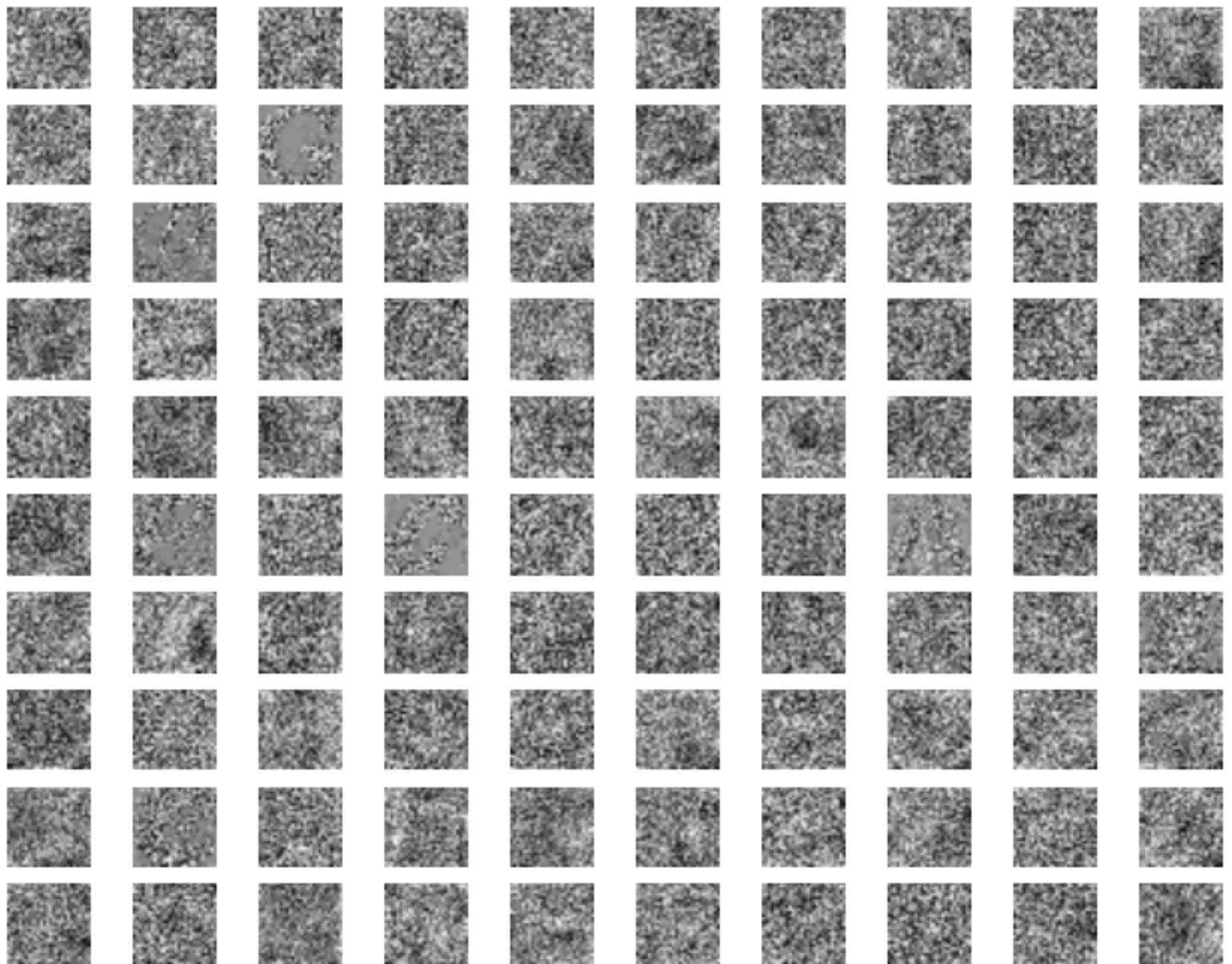


Figure 24: Weights at 100%

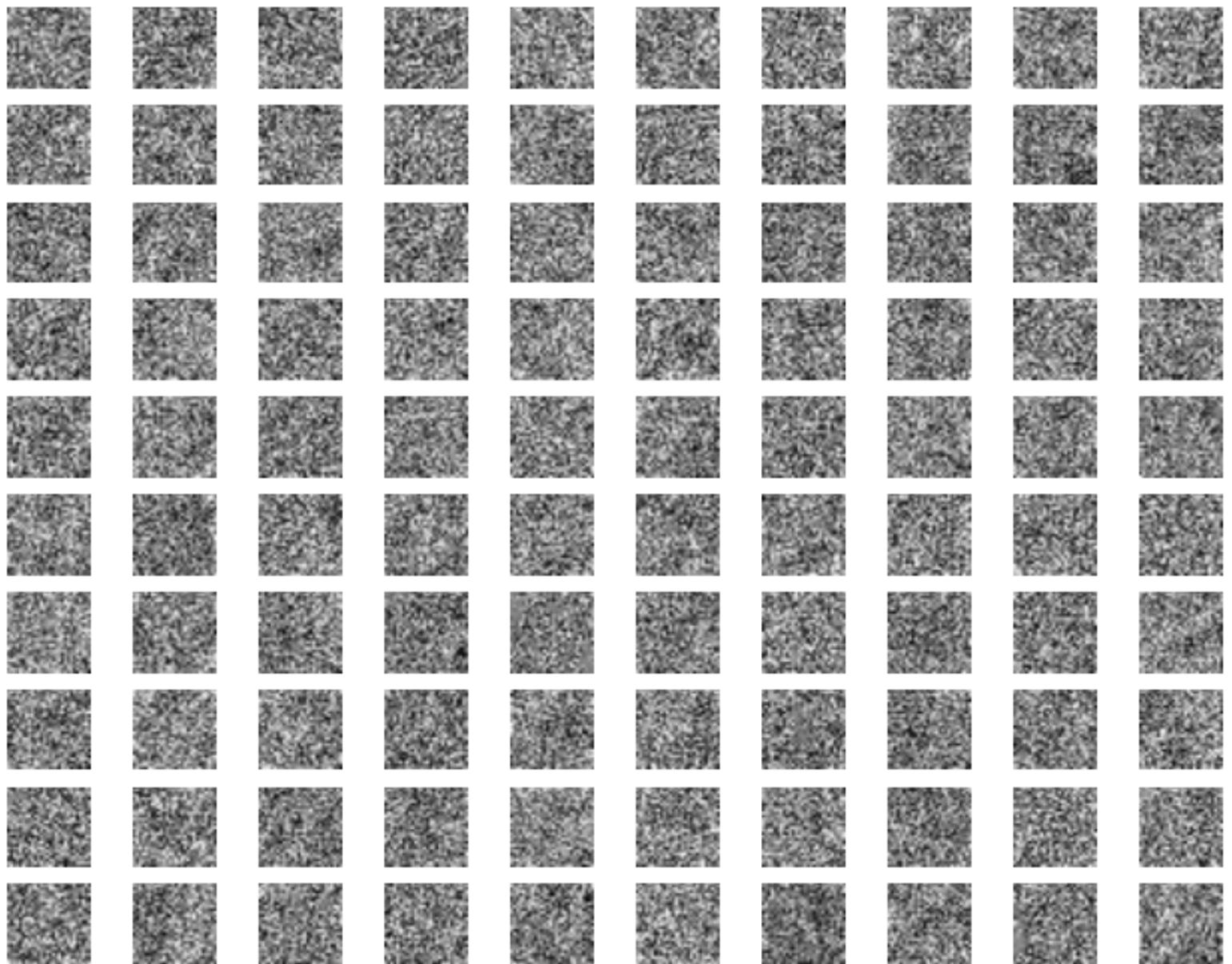


Figure 25: Weights at 25% with dropout

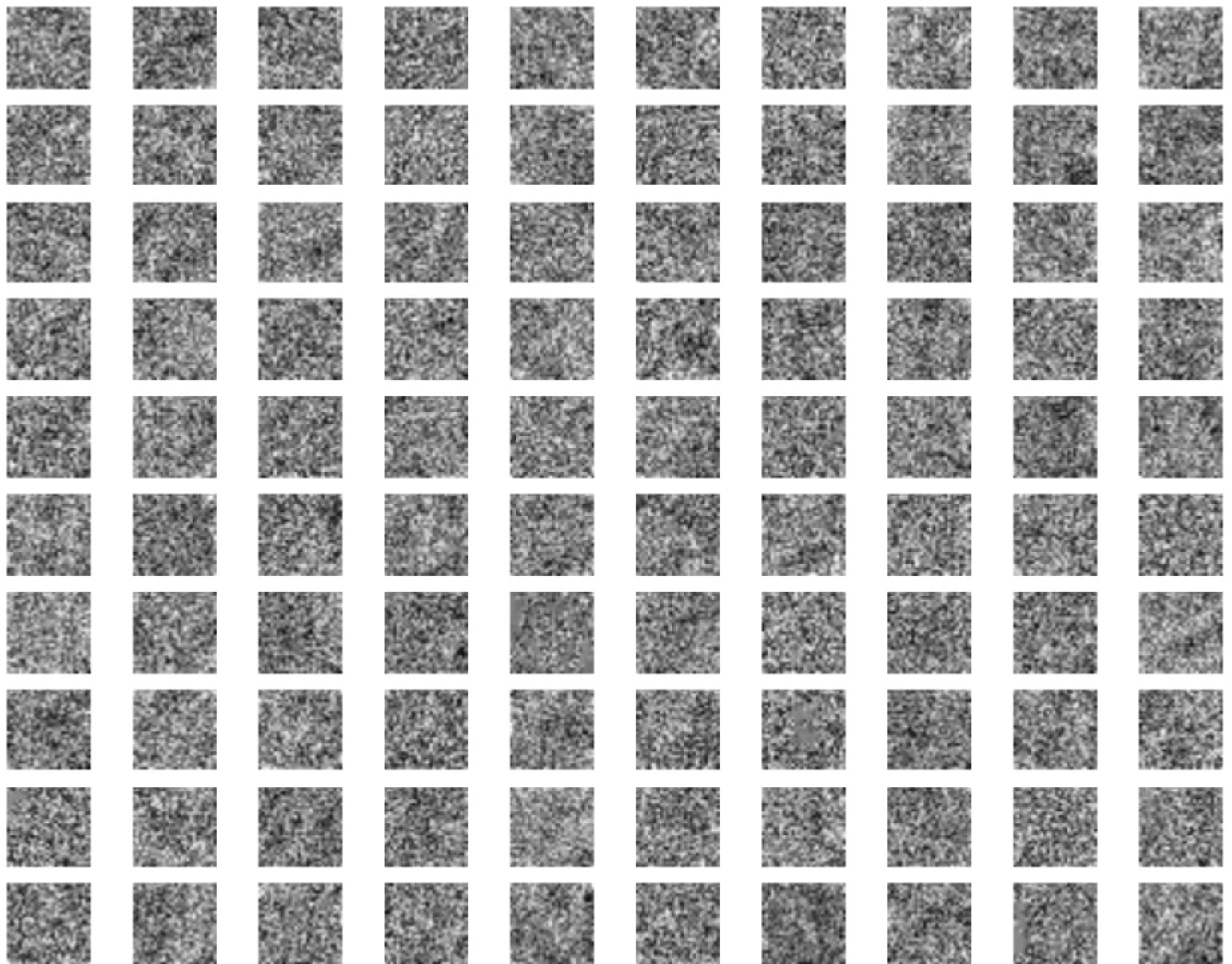


Figure 26: Weights at 50% with dropout

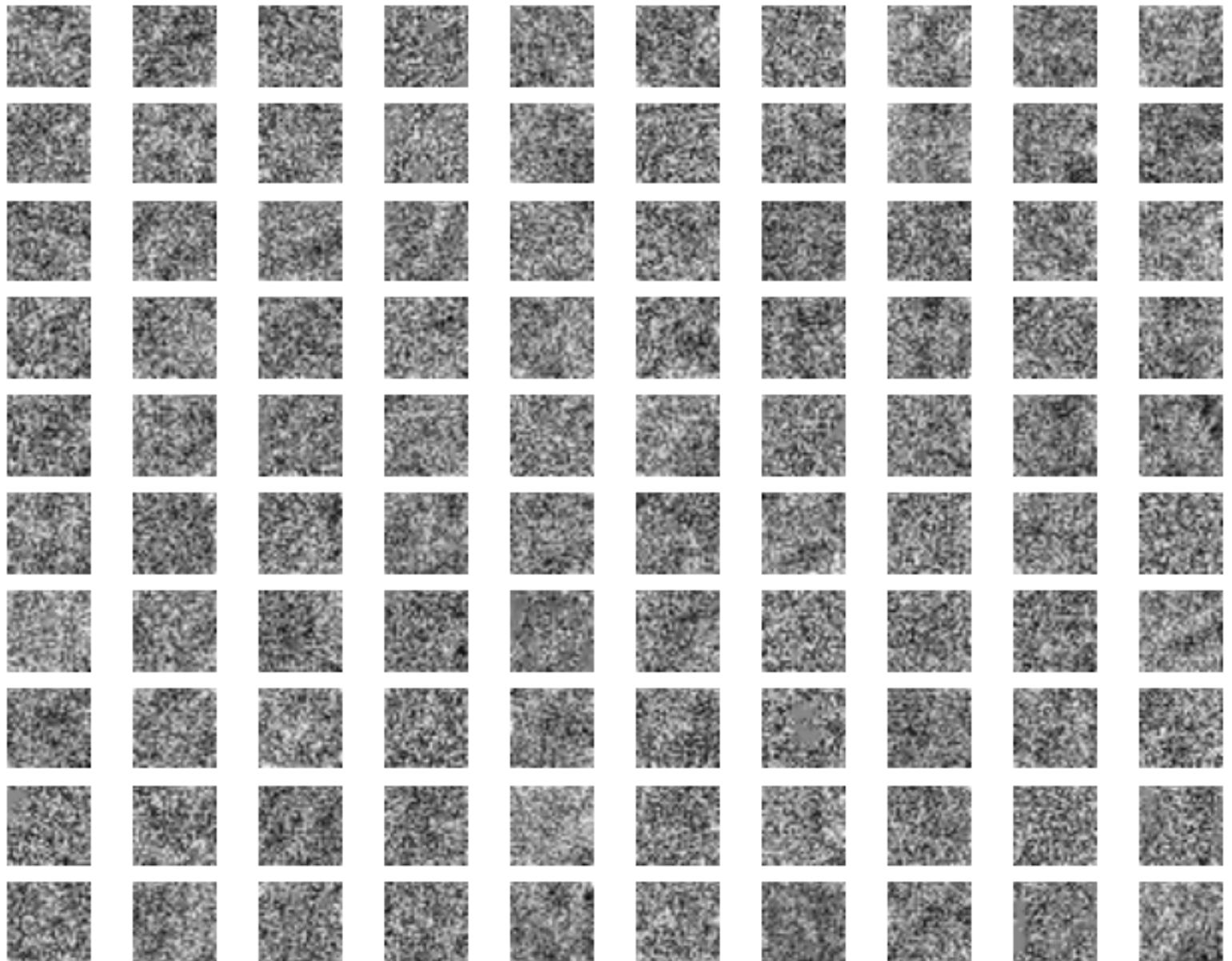


Figure 27: Weights at 75% with dropout

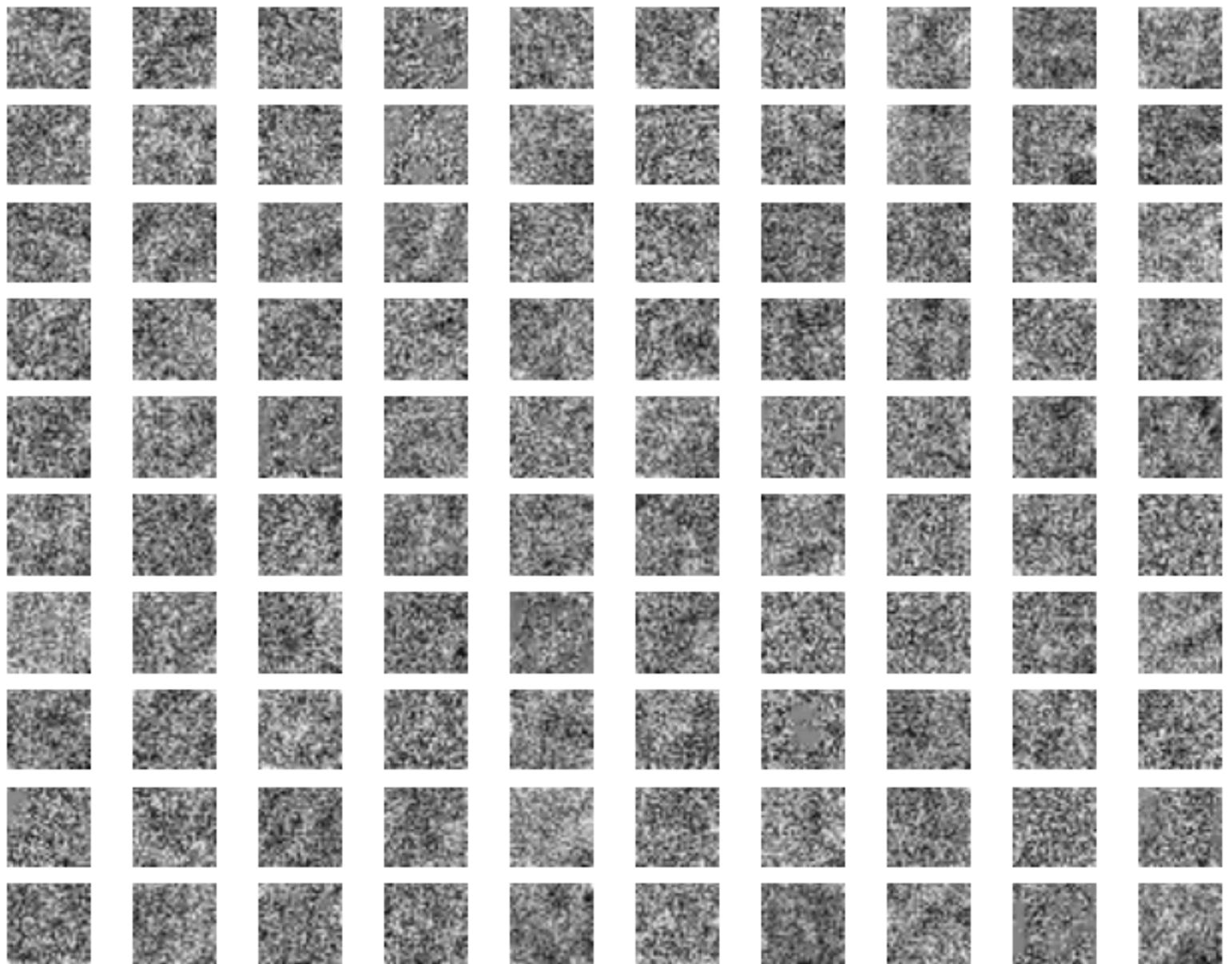


Figure 28: Weights at 100% with dropout

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

```

14 def buildGraph_SGD(decay_rate , learning_rate):
15     # Variable creation
16     W = tf.Variable(tf.truncated_normal(shape=[784,1] , stddev=0.5) , name='weights')
17     b = tf.Variable(0.0 , name='biases')
18     X = tf.placeholder(tf.float32 , [None, 784] , name='input_x')
19     y_target = tf.placeholder(tf.float32 , [None,1] , name='target_y')
20
21     # Graph definition
22     y_predicted = tf.matmul(X,W) + b
23
24     # Error definition
25     error = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(targets =
26         y_target , logits = y_predicted)) + \
27         0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W) , tf.float32))
28
29     acc = tf.reduce_mean(tf.cast(tf.equal(y_target , tf.cast((tf.sigmoid(y_predicted)
30         ) > 0.5) , tf.float32)) , tf.float32)
31     # Training mechanism
32     optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
33     train = optimizer.minimize(loss=error)
34     return W, b, X, y_target , y_predicted , error , train , acc
35
36 def getRandomBatch(trainData , trainTarget , size):
37     idx = np.random.choice(trainData.shape[0] , size , replace=False)
38     return trainData[idx,:] , trainTarget[idx,:]
39
40 decay_rate = 0.01
41 batch_size = 100
42 learning_rate = 0.01
43 #best learning rate 0.01
44 W, b, X, y_target , y_predicted , error , train , acc = buildGraph_SGD(decay_rate ,
45     learning_rate)
```

```

43 init = tf.global_variables_initializer()
44 sess = tf.InteractiveSession()
45 sess.run(init)
46 train_loss_recorder = np.array([])
47 train_acc_recorder = np.array([])
48 test_loss_recorder = np.array([])
49 test_acc_recorder = np.array([])
50 numIteration = 500
51 best_acc = 0
52 for itr in range(numIteration):
53     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
54     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
55                                 y_target: batch_ys})
56     train_loss_recorder = np.append(train_loss_recorder, loss)
57     train_acc_recorder = np.append(train_acc_recorder, accuracy)
58     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
59                                     testTarget})
59     test_loss_recorder = np.append(test_loss_recorder, test_loss)
60     test_acc_recorder = np.append(test_acc_recorder, test_acc)
61     best_acc = max(best_acc, test_acc)
62 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
63 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
64 plt.legend()
65 #plt.axis([0,500, 0, 2])
66 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
67 plt.show()
68 plt.plot(np.arange(numIteration), train_acc_recorder, 'g', label="Train")
69 plt.plot(np.arange(numIteration), test_acc_recorder, 'b', label="Test")
70 plt.legend(loc="lower right")
71 #plt.axis([0,500, 0, 2])
72 plt.title("Accuracy VS number of updates for learning_rate = %0.3f%(learning_rate")
73 plt.show()
74
75 print("Best accuracy in test set is %0.2f%(best_acc))
```

Listing 2: 1.1.1

```

76 def buildGraph_Adam(decay_rate, learning_rate):
77     # Variable creation
78     W = tf.Variable(tf.truncated_normal(shape=[784,1], stddev=0.5), name='weights')
79     b = tf.Variable(0.0, name='biases')
80     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
81     y_target = tf.placeholder(tf.float32, [None,1], name='target_y')
82
83     # Graph definition
84     y_predicted = tf.matmul(X,W) + b
85
86     # Error definition
87     error = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(targets =
88                         y_target, logits = y_predicted),
89                         name='mean_squared_error') + \
90                         0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))
91
92     acc = tf.reduce_mean(tf.cast(tf.equal(y_target, tf.cast((tf.sigmoid(y_predicted)
93                         ) > 0.5), tf.float32)),tf.float32))
94
95     # Training mechanism
96     optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
97     train = optimizer.minimize(loss=error)
98     return W, b, X, y_target, y_predicted, error, train, acc
99
100 # training using SGD
```

```

101 W, b, X, y_target, y_predicted, error, train, acc = buildGraph_SGD(decay_rate,
102     learning_rate)
103 init = tf.global_variables_initializer()
104 sess = tf.InteractiveSession()
105 sess.run(init)
106 sgd_train_loss_recorder = np.array([])
107 sgd_train_acc_recorder = np.array([])
108 sgd_test_loss_recorder = np.array([])
109 sgd_test_acc_recorder = np.array([])
110 numIteration = 500
111 for itr in range(numIteration):
112     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
113     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
114         y_target: batch_ys})
115     sgd_train_loss_recorder = np.append(sgd_train_loss_recorder, loss)
116     sgd_train_acc_recorder = np.append(sgd_train_acc_recorder, accuracy)
117     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
118         testTarget})
119     sgd_test_loss_recorder = np.append(sgd_test_loss_recorder, test_loss)
120     sgd_test_acc_recorder = np.append(sgd_test_acc_recorder, test_acc)
121
122 # training using Adam
123 W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate,
124     learning_rate)
125 init = tf.global_variables_initializer()
126 sess = tf.InteractiveSession()
127 sess.run(init)
128 adam_train_loss_recorder = np.array([])
129 adam_train_acc_recorder = np.array([])
130 adam_test_loss_recorder = np.array([])
131 adam_test_acc_recorder = np.array([])
132 numIteration = 500
133 for itr in range(numIteration):
134     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
135     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
136         y_target: batch_ys})
137     adam_train_loss_recorder = np.append(adam_train_loss_recorder, loss)
138     adam_train_acc_recorder = np.append(adam_train_acc_recorder, accuracy)
139     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
140         testTarget})
141     adam_test_loss_recorder = np.append(adam_test_loss_recorder, test_loss)
142     adam_test_acc_recorder = np.append(adam_test_acc_recorder, test_acc)
143 plt.plot(np.arange(numIteration), sgd_train_loss_recorder, 'g', label="SGD_train")
144 plt.plot(np.arange(numIteration), adam_train_loss_recorder, 'b', label="Adam_train")
145 plt.plot(np.arange(numIteration), sgd_test_loss_recorder, 'r', label="SGD_test")
146 plt.plot(np.arange(numIteration), adam_test_loss_recorder, 'k', label="Adam_test")
147 plt.legend()
148 #plt.axis([0,500, 0, 2])
149 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
150 plt.show()
151
152 plt.plot(np.arange(numIteration), sgd_train_acc_recorder, 'g', label="SGD")
153 plt.plot(np.arange(numIteration), adam_train_acc_recorder, 'b', label="Adam")
154 plt.plot(np.arange(numIteration), sgd_test_acc_recorder, 'r', label="SGD_test")
155 plt.plot(np.arange(numIteration), adam_test_acc_recorder, 'k', label="Adam_test")
156 plt.legend(loc="lower right")
157 #plt.axis([0,500, 0, 2])
158 plt.title("Accuracy VS number of updates for learning_rate = %0.3f%(learning_rate")
159 )
160 plt.show()

```

Listing 3: 1.1.2

```

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

```

```

157     b = tf.Variable(0.0, name='biases')
158     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
159     y_target = tf.placeholder(tf.float32, [None, 1], name='target_y')
160
161     # Graph definition
162     y_predicted = tf.matmul(X,W) + b
163
164     # Error definition
165     error = 0.5 * tf.reduce_mean(tf.cast(tf.reduce_sum(tf.cast(tf.square(
166         y_predicted - y_target), tf.float32),
167             reduction_indices=1,
168             name='squared_error'), tf.float32),
169             name='mean_squared_error') + \
170             0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32)))
171
172     # y_predicted = tf.nn.l2_normalize(y_predicted, 0)
173     acc = tf.reduce_mean(tf.cast(tf.equal(y_target, tf.cast(y_predicted > 0.5, tf.
174         float32)), tf.float32))
175
176     # Training mechanism
177     optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
178     train = optimizer.minimize(loss=error)
179
180     return W, b, X, y_target, y_predicted, error, train, acc
181
182 decay_rate = 0
183 batch_size = 500
184 learning_rate = 0.01
185 # training using SGD
186 W, b, X, y_target, y_predicted, error, train, acc = buildGraph_linear(decay_rate,
187     learning_rate)
188 init = tf.global_variables_initializer()
189 sess = tf.InteractiveSession()
190 sess.run(init)
191 lin_train_loss_recorder = np.array([])
192 lin_train_acc_recorder = np.array([])
193 lin_valid_loss_recorder = np.array([])
194 lin_valid_acc_recorder = np.array([])
195 lin_test_loss_recorder = np.array([])
196 lin_test_acc_recorder = np.array([])
197 numIteration = 500
198 for itr in range(numIteration):
199     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
200     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
201         y_target: batch_ys})
202     lin_train_loss_recorder = np.append(lin_train_loss_recorder, loss)
203     lin_train_acc_recorder = np.append(lin_train_acc_recorder, accuracy)
204     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
205         y_target: validTarget})
206     lin_valid_loss_recorder = np.append(lin_valid_loss_recorder, valid_loss)
207     lin_valid_acc_recorder = np.append(lin_valid_acc_recorder, valid_acc)
208     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
209         testTarget})
210     lin_test_loss_recorder = np.append(lin_test_loss_recorder, test_loss)
211     lin_test_acc_recorder = np.append(lin_test_acc_recorder, test_acc)
212
213 # training using Adam
214 W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate,
215     learning_rate)
216 init = tf.global_variables_initializer()
217 sess = tf.InteractiveSession()
218 sess.run(init)
219 adam_train_loss_recorder = np.array([])
220 adam_train_acc_recorder = np.array([])
221 adam_valid_loss_recorder = np.array([])
222 adam_valid_acc_recorder = np.array([])
223 adam_test_loss_recorder = np.array([])
224 adam_test_acc_recorder = np.array([])
225
226 for itr in range(numIteration):

```

```

217 batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batchSize)
218 accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
219 y_target: batch_ys})
220 adam_train_loss_recorder = np.append(adam_train_loss_recorder, loss)
221 adam_train_acc_recorder = np.append(adam_train_acc_recorder, accuracy)
222 valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: testData,
223 y_target: validTarget})
224 adam_valid_loss_recorder = np.append(adam_valid_loss_recorder, test_loss)
225 adam_valid_acc_recorder = np.append(adam_valid_acc_recorder, test_acc)
226 test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
227 testTarget})
228 adam_test_loss_recorder = np.append(adam_test_loss_recorder, test_loss)
229 adam_test_acc_recorder = np.append(adam_test_acc_recorder, test_acc)
230 plt.plot(np.arange(numIteration), lin_train_loss_recorder, 'g', label="Lin_train")
231 plt.plot(np.arange(numIteration), adam_train_loss_recorder, 'b', label="Adam_train")
232 plt.plot(np.arange(numIteration), lin_valid_loss_recorder, 'c', label="Lin_valid")
233 plt.plot(np.arange(numIteration), adam_valid_loss_recorder, 'm', label="Adam_valid")
234 plt.plot(np.arange(numIteration), lin_test_loss_recorder, 'r', label="Lin_test")
235 plt.plot(np.arange(numIteration), adam_test_loss_recorder, 'k', label="Adam_test")
236 plt.legend()
237 #plt.axis([0,500, 0, 2])
238 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
239 plt.show()
240
241 plt.plot(np.arange(numIteration), lin_train_acc_recorder, 'g', label="Lin_train")
242 plt.plot(np.arange(numIteration), adam_train_acc_recorder, 'b', label="Adam_train")
243 plt.plot(np.arange(numIteration), lin_valid_acc_recorder, 'c', label="Lin_valid")
244 plt.plot(np.arange(numIteration), adam_valid_acc_recorder, 'm', label="Adam_valid")
245 plt.plot(np.arange(numIteration), lin_test_acc_recorder, 'r', label="Lin_test")
246 plt.plot(np.arange(numIteration), adam_test_acc_recorder, 'k', label="Adam_test")
247 plt.legend(loc="lower right")
248 #plt.axis([0,500, 0, 2])
249 plt.title("Accuracy VS number of updates for learning_rate = %0.3f%(learning_rate)")
250 plt.show()

```

Listing 4: 1.1.3

```

248 def dense_to_one_hot(labels_dense, num_classes=10):
249     num_labels = labels_dense.shape[0]
250     index_offset = np.arange(num_labels) * num_classes
251     labels_one_hot = np.zeros((num_labels, num_classes))
252     labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
253     return labels_one_hot
254 def buildGraph_Adam(decay_rate, learning_rate):
255     # Variable creation
256     W = tf.Variable(tf.truncated_normal(shape=[784,10], stddev=0.5), name='weights')
257     b = tf.Variable(tf.zeros([10]), name='biases')
258     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
259     y_target = tf.placeholder(tf.float32, [None,10], name='target_y')
260
261     # Graph definition
262     y_predicted = tf.matmul(X,W) + b
263
264     # Error definition
265     error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
266                           y_target, logits = y_predicted)) + \
267                           0.5 * decay_rate * tf.reduce_sum(tf.cast(tf.square(W), tf.float32))
268
269     correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
270     acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
271
272     # Training mechanism
273     optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
274     train = optimizer.minimize(loss=error)

```

```

273     return W, b, X, y_target, y_predicted, error, train, acc
274
275 decay_rate = 0.01
276 batch_size = 500
277 learning_rate = 0.01
278 #best learning rate 0.01
279 W, b, X, y_target, y_predicted, error, train, acc = buildGraph_Adam(decay_rate,
280                                         learning_rate)
280 init = tf.global_variables_initializer()
281 sess = tf.InteractiveSession()
282 sess.run(init)
283 train_loss_recorder = np.array([])
284 train_acc_recorder = np.array([])
285 test_loss_recorder = np.array([])
286 test_acc_recorder = np.array([])
287 numIteration = 500
288 best_acc = 0
289 for itr in range(numIteration):
290     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
291     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
292                               y_target: batch_ys})
293     train_loss_recorder = np.append(train_loss_recorder, loss)
294     train.acc_recorder = np.append(train.acc_recorder, accuracy)
295     test.loss, test.acc = sess.run([error, acc], feed_dict={X: testData, y_target:
296                               testTarget})
297     test_loss_recorder = np.append(test_loss_recorder, test.loss)
298     test.acc_recorder = np.append(test.acc_recorder, test.acc)
299     best_acc = max(best_acc, test.acc)
300 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
301 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
302 plt.legend()
303 #plt.axis([0,500, 0, 2])
304 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
305 plt.show()
306
307 plt.plot(np.arange(numIteration), train_acc_recorder, 'g', label="Train")
308 plt.plot(np.arange(numIteration), test_acc_recorder, 'b', label="Test")
309 plt.legend(loc="lower right")
310 plt.title("Accuracy VS number of updates for learning_rate = %0.3f%(learning_rate")
311 )
312 plt.show()
313 print("Best accuracy in test set is %0.2f%"(best_acc))

```

Listing 5: 1.2.3

```

313 # Parameters
314 learning_rate_list = [0.001, 0.01, 0.1]
315 batch_size = 100
316 display_step = 1
317 decay_rate = 3e-4
318 # Network Parameters
319 n_hidden_1 = 1000 # 1st layer number of features
320 n_input = 784 # MNIST data input (img shape: 28*28)
321 n_classes = 10
322
323 def buildGraph(decay_rate, learning_rate):
324     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
325     y_target = tf.placeholder(tf.float32, [None, 10], name='target_y')
326
327     # Graph definition
328     layer_1 = layerBlock(X, n_hidden_1, decay_rate)
329     layer_1 = tf.nn.relu(layer_1)
330     out_layer = layerBlock(layer_1, n_classes, decay_rate)
331     y_predicted = out_layer
332

```

```

333 # Error definition
334 error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
335     y_target, logits = y_predicted)) + \
336         tf.add_n(tf.get_collection('losses'))
337
338 correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
339 acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
340 # Training mechanism
341 optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
342 train = optimizer.minimize(loss=error)
343 return X, y_target, y_predicted, error, train, acc
344
345 decay_rate = 3e-4
346 batch_size = 100
347 learning_rate = 0.001
348 #best learning rate 0.001
349 tf.reset_default_graph()
350 X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
351 init = tf.global_variables_initializer()
352 sess = tf.InteractiveSession()
353 sess.run(init)
354 train_loss_recorder = np.array([])
355 train_mis_recorder = np.array([])
356 valid_loss_recorder = np.array([])
357 valid_mis_recorder = np.array([])
358 test_loss_recorder = np.array([])
359 test_mis_recorder = np.array([])
360 numIteration = 500
361 best_acc = 0
362 for itr in range(numIteration):
363     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
364     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
365         y_target: batch_ys})
366     train_loss_recorder = np.append(train_loss_recorder, loss)
367     train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)
368
369     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
370         y_target: validTarget})
371     valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
372     valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)
373
374     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
375         testTarget})
376     test_loss_recorder = np.append(test_loss_recorder, test_loss)
377     test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
378
379     best_acc = max(best_acc, test_acc)
380
381 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
382 plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
383 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
384 plt.legend()
385 #plt.axis([0,500, 0, 2])
386 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
387 plt.show()
388
389 plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
390 plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
391 plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
392 plt.legend()
393 #plt.axis([0,500, 0, 2])
394 plt.title("Error VS number of updates for learning_rate = %0.3f%(learning_rate)")
395 plt.show()
396
397 print("Best accuracy in test set is %0.2f%%(best_acc))
```

Listing 6: 2.2.2

```

394 decay_rate = 3e-4
395 batch_size = 100
396 learning_rate = 0.001
397 n_hidden_1 = 500
398 #best learning rate 0.001, number of hidden units 500
399 tf.reset_default_graph()
400 X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
401 init = tf.global_variables_initializer()
402 sess = tf.InteractiveSession()
403 sess.run(init)
404 train_loss_recorder = np.array([])
405 train_mis_recorder = np.array([])
406 valid_loss_recorder = np.array([])
407 valid_mis_recorder = np.array([])
408 test_loss_recorder = np.array([])
409 test_mis_recorder = np.array([])
410 numIteration = 500
411 best_acc = 0
412 for itr in range(numIteration):
413     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
414     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
415                                 y_target: batch_ys})
416     train_loss_recorder = np.append(train_loss_recorder, loss)
417     train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)
418
419     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
420                                     y_target: validTarget})
421     valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
422     valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)
423
424     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
425                                    testTarget})
426     test_loss_recorder = np.append(test_loss_recorder, test_loss)
427     test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
428
429     best_acc = max(best_acc, test_acc)
430
431 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
432 plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
433 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
434 plt.legend()
435 #plt.axis([0,500, 0, 2])
436 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
437 plt.show()
438
439 plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
440 plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
441 plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
442 plt.legend()
443 #plt.axis([0,500, 0, 2])
444 plt.title("Error VS number of updates for learning_rate = %0.3f%(learning_rate)")
445 plt.show()
446 print("Best accuracy in test set is %0.2f%(%(best_acc))
```

Listing 7: 2.3.1

```

445 # Network Parameters
446 n_hidden_1 = 500
447 n_hidden_2 = 500
448 n_input = 784 # MNIST data input (img shape: 28*28)
449 n_classes = 10
450
451 def buildGraph(decay_rate, learning_rate):
452     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
453     y_target = tf.placeholder(tf.float32, [None, 10], name='target_y')
```

```

455 # Graph definition
456 layer_1 = layerBlock(X, n_hidden_1, decay_rate)
457 layer_1 = tf.nn.relu(layer_1)
458 layer_2 = layerBlock(layer_1, n_hidden_2, decay_rate)
459 layer_2 = tf.nn.relu(layer_2)
460 out_layer = layerBlock(layer_2, n_classes, decay_rate)
461 y_predicted = out_layer
462
463 # Error definition
464 error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
465 y_target, logits = y_predicted)) + \
466     tf.add_n(tf.get_collection('losses'))
467
468 correct_prediction = tf.equal(tf.argmax(y_predicted, 1), tf.argmax(y_target, 1))
469 acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
470 # Training mechanism
471 optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
472 train = optimizer.minimize(loss=error)
473 return X, y_target, y_predicted, error, train, acc
474
475 decay_rate = 3e-4
476 batch_size = 100
477 learning_rate = 0.001
478 #best learning rate 0.001, number of hidden units 500
479 tf.reset_default_graph()
480 X, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
481 init = tf.global_variables_initializer()
482 sess = tf.InteractiveSession()
483 sess.run(init)
484 train_loss_recorder = np.array([])
485 train_mis_recorder = np.array([])
486 valid_loss_recorder = np.array([])
487 valid_mis_recorder = np.array([])
488 test_loss_recorder = np.array([])
489 test_mis_recorder = np.array([])
490 numIteration = 500
491 best_acc = 0
492 for itr in range(numIteration):
493     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
494     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
495                               y_target: batch_ys})
496     train_loss_recorder = np.append(train_loss_recorder, loss)
497     train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)
498
499     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
500                                     y_target: validTarget})
501     valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
502     valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)
503
504     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData,
505                                    y_target: testTarget})
506     test_loss_recorder = np.append(test_loss_recorder, test_loss)
507     test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
508
509     best_acc = max(best_acc, test_acc)
510
511 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
512 plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
513 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
514 plt.legend()
515 #plt.axis([0,500, 0, 2])
516 plt.title("Total loss VS number of updates for learning_rate = %0.3f%(learning_rate)")
517 plt.show()
518
519 plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
520 plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")

```

```

517 plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
518 plt.legend()
519 #plt.axis([0,500, 0, 2])
520 plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
521 plt.show()
522
523 print("Best accuracy in test set is %0.2f"%(best_acc))

```

Listing 8: 2.3.2

```

524 # Network Parameters
525 n_hidden_1 = 500
526 n_hidden_2 = 500
527 n_input = 784 # MNIST data input (img shape: 28*28)
528 n_classes = 10
529 tf.reset_default_graph()
530 keep_prob = tf.placeholder(tf.float32) #dropout (keep probability)
531 def buildGraph(decay_rate, learning_rate):
532     X = tf.placeholder(tf.float32, [None, 784], name='input_x')
533     y_target = tf.placeholder(tf.float32, [None, 10], name='target_y')
534
535     # Graph definition
536     layer_1 = layerBlock(X, n_hidden_1, decay_rate)
537     layer_1 = tf.nn.relu(layer_1)
538     dropout_1 = tf.nn.dropout(layer_1, keep_prob)
539
540     layer_2 = layerBlock(dropout_1, n_hidden_2, decay_rate)
541     layer_2 = tf.nn.relu(layer_2)
542     dropout_2 = tf.nn.dropout(layer_2, keep_prob)
543
544     out_layer = layerBlock(dropout_2, n_classes, decay_rate)
545     y_predicted = out_layer
546
547     # Error definition
548     error = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
549                           y_target, logits = y_predicted)) + \
550                           tf.add_n(tf.get_collection('losses'))
551
552     correct_prediction = tf.equal(tf.argmax(y_predicted,1), tf.argmax(y_target,1))
553     acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
554
555     # Training mechanism
556     optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate)
557     train = optimizer.minimize(loss=error)
558
559     return X, y_target, y_predicted, error, train, acc
560
561 decay_rate = 3e-4
562 batch_size = 100
563 learning_rate = 0.001
564 #best learning rate 0.001, number of hidden units 500
565 X, y_target, y_predicted, error, train, acc = buildGraph(decay_rate, learning_rate)
566 init = tf.global_variables_initializer()
567 sess = tf.InteractiveSession()
568 sess.run(init)
569 train_loss_recorder = np.array([])
570 train_mis_recorder = np.array([])
571 valid_loss_recorder = np.array([])
572 valid_mis_recorder = np.array([])
573 test_loss_recorder = np.array([])
574 test_mis_recorder = np.array([])
575 numIteration = 20000
576 best_acc = 0
577 for itr in range(numIteration):
578     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
579     accuracy, loss = sess.run([acc, error], feed_dict={X: batch_xs, y_target:
580                           batch_ys, keep_prob: 1.0})
581     train_loss_recorder = np.append(train_loss_recorder, loss)
582     train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)

```

```

580     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
581                                         y_target: validTarget, keep_prob: 1.0})
582     valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
583     valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)
584
584     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
585                                         testTarget, keep_prob: 1.0})
586     test_loss_recorder = np.append(test_loss_recorder, test_loss)
587     test_mis_recorder = np.append(test_mis_recorder, 1.0 - test_acc)
588
588     if itr % 100 == 0:
589         print("step %d, training accuracy %g, test set accuracy %g"%(itr, accuracy,
590                                         test_acc))
591
591     best_acc = max(best_acc, test_acc)
592     sess.run(train, feed_dict={X: batch_xs, y_target: batch_ys, keep_prob: 0.5})
593
594
594 plt.plot(np.arange(numIteration), train_loss_recorder, 'g', label="Train")
595 plt.plot(np.arange(numIteration), valid_loss_recorder, 'r', label="Valid")
596 plt.plot(np.arange(numIteration), test_loss_recorder, 'b', label="Test")
597 plt.legend()
598 #plt.axis([0,500, 0, 2])
599 plt.title("Total loss VS number of updates for learning_rate = %0.3f"%(learning_rate))
600 plt.show()
601
602 plt.plot(np.arange(numIteration), train_mis_recorder, 'g', label="Train")
603 plt.plot(np.arange(numIteration), valid_mis_recorder, 'r', label="Valid")
604 plt.plot(np.arange(numIteration), test_mis_recorder, 'b', label="Test")
605 plt.legend()
606 #plt.axis([0,500, 0, 2])
607 plt.title("Error VS number of updates for learning_rate = %0.3f"%(learning_rate))
608 plt.show()
609
610 print("Best accuracy in test set is %0.2f"%(best_acc))

```

Listing 9: 2.4.1

```

1 learning_rate = 0.001
2 X, y_target, y_predicted, error, train, acc, W1 = buildGraph(decay_rate,
   learning_rate)
3
4 init = tf.global_variables_initializer()
5 sess = tf.InteractiveSession()
6 sess.run(init)
7 train_loss_recorder = np.array([])
8 train_mis_recorder = np.array([])
9 valid_loss_recorder = np.array([])
10 valid_mis_recorder = np.array([])
11 test_loss_recorder = np.array([])
12 test_mis_recorder = np.array([])
13
14 numIteration = 200
15 for itr in range(numIteration):
16     batch_xs, batch_ys = getRandomBatch(trainData, trainTarget, batch_size)
17     accuracy, loss, _ = sess.run([acc, error, train], feed_dict={X: batch_xs,
   y_target: batch_ys})
18     train_loss_recorder = np.append(train_loss_recorder, loss)
19     train_mis_recorder = np.append(train_mis_recorder, 1.0 - accuracy)
20
21     valid_loss, valid_acc = sess.run([error, acc], feed_dict={X: validData,
   y_target: validTarget})
22     valid_loss_recorder = np.append(valid_loss_recorder, valid_loss)
23     valid_mis_recorder = np.append(valid_mis_recorder, 1.0 - valid_acc)
24
25     test_loss, test_acc = sess.run([error, acc], feed_dict={X: testData, y_target:
   testTarget})

```

```

26     test_loss_recorder = np.append(test_loss_recorder , test_loss)
27     test_mis_recorder = np.append(test_mis_recorder , 1.0 - test_acc)
28     print("Iteration: %d, Train Acc: %0.3f, Valid Acc: %0.3f"%(itr , accuracy*100,
29           valid_acc*100))
30
31     if ((itr+1)%50 == 0):
32         plt.figure()
33         f, axarr = plt.subplots(40,25)
34         W1_eval = W1.eval()
35         for i in range(40):
36             for j in range(25):
37                 axarr[i,j].imshow(W1_eval[:, i*25+j].reshape(28,28), cmap='gray')
38                 axarr[i,j].axis('off')
39         plt.savefig("{}{}.png".format(itr));

```

Listing 10: 2.4.2

```

1 batch_size = 100
2 global_best_test_acc = 0
3
4 for numMod in range(100):
5
6     learning_rate = np.exp(np.random.rand() * 3 - 7.5)
7     numLayers = np.random.randint(5) + 1
8     unitPerLayer = np.random.randint(100,500,numLayers)
9     decay_rate = np.exp(np.random.rand() * 3 - 9)
10    dropOut = np.random.randint(2)
11
12    print("LR = %0.10f, NL = %d, DR = %0.10f, Dropout = %d" \
13          %(learning_rate, numLayers, decay_rate, dropOut))
14    print(unitPerLayer)
15
16    #best learning rate 0.001, number of hidden units 500
17    # tf.reset_default_graph()
18    X, y_target , y_predicted , error , train , acc = buildGraph(numLayers ,
19    unitPerLayer , decay_rate , learning_rate , dropOut)
20
21    init = tf.global_variables_initializer()
22
23    sess = tf.InteractiveSession()
24    sess.run(init)
25    train_loss_recorder = np.array([])
26    train_mis_recorder = np.array([])
27    valid_loss_recorder = np.array([])
28    valid_mis_recorder = np.array([])
29    test_loss_recorder = np.array([])
30    test_mis_recorder = np.array([])
31
32    numIteration = 2000
33
34    best_acc_valid = 0
35    best_acc_test = 0
36
37    for itr in range(numIteration):
38        batch_xs , batch_ys = getRandomBatch(trainData , trainTarget , batch_size)
39        accuracy , loss , _ = sess.run([acc , error , train] , feed_dict={X: batch_xs ,
40            y_target: batch_ys , keep_prob: 0.8})
41        train_loss_recorder = np.append(train_loss_recorder , loss)
42        train_mis_recorder = np.append(train_mis_recorder , 1.0 - accuracy)
43
44        valid_loss , valid_acc = sess.run([error , acc] , feed_dict={X: validData ,
45            y_target: validTarget , keep_prob: 0.8})
46        valid_loss_recorder = np.append(valid_loss_recorder , valid_loss)
47        valid_mis_recorder = np.append(valid_mis_recorder , 1.0 - valid_acc)
48
49        test_loss , test_acc = sess.run([error , acc] , feed_dict={X: testData ,
50            y_target: testTarget , keep_prob: 0.8})
51        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
50     *100, valid_acc*100))
51     best_acc_valid = max(best_acc_valid, valid_acc)
52     best_acc_test = max(best_acc_test, test_acc)
53
54     print("ValidAcc = %0.3f, testAcc = %0.3f\n" %(best_acc_valid, best_acc_test))
55
56     if(global_best_test_acc < best_acc_test):
57
58         global_best_test_acc = best_acc_test
59
60         b_lr = learning_rate
61         b_dr = decay_rate
62         b_do = dropOut
63         b_nl = numLayers
64         b_ul = unitPerLayer
65         b_va = best_acc_valid
66
67         print("BEST: LR = %0.10f, NL = %d, DR = %0.10f, Dropout = %d, ValidAcc = %0.3f,
68             TestAcc = %0.3f"\n
69             %(b_lr, b_nl, b_dr, b_do, b_va, global_best_test_acc))
70         print(b_ul)

```

Listing 11: 2.5.1