ECE421S – Introduction to Machine Learning

Assignment 2

Neural Networks

Hard Copy Due: March 6, 2019 @ BA3014, 4:00-5:00 PM EST

Code Submission: ece421ta2019@gmail.com

March 6, 2019 @ 5:00 PM EST

General Notes:

- Attach this cover page to your hard copy submission
- For assignment related questions, please contact Matthew Wong (matthewck.wong@mail.utoronto.ca)
- For general questions regarding Python or Tensorflow, please contact Tianrui Xiao (tianrui.xiao@mail.utoronto.ca) or see him in person in his office hours, Tuesdays, 4:00-6:00 PM in BA-3128 (Robotics Lab)

Please circle section to which you would like the assignment returned

Tutorial Sections

001	002	003	**(004)**
005	006	007	Graduate

Group Members	
Names (Work Split)	StudentID
Michael Tang (50%)	1001389525
Khoi To (50%)	1002078477

1. Neural Network Using Numpy

1.1 Helper Functions:

1. ReLu:

2. Softmax:

```
def softmax(x):

    return np.exp(x)/sum(np.exp(x))
```

3. Compute:

4. AverageCE:

5. GradCE:

For a single data vector:
$L = -$

1.2 Backpropagation Derivation

1. (gradient of loss wrt outer layer weights)

$$\frac{\partial E^n}{\partial w_{ki}} = (y_k^n - t_k^n)x_i^n$$

\

1.3

Learning rate

2. Neural Networks in Tensorflow

2.1 Model implementation

Convolutional neural network is implemented in cnn.py (Appendix …)

```
import tensorflow as tf

class ConvolutionalNeuralNetwork(object):
    def build_model(self,
            seed=421,#tf seed
            alpha=10e-4, #learning rate for ADAM optimizer
            with_dropout=False, p=0.9, #dropout
            with_regularizers=False, beta=0.01 #regulizer
            ):
        #initialize
        tf.reset_default_graph()
        tf.set_random_seed(seed)


        # label
        self.y = tf.placeholder(tf.float32, [None, 10], 'y')

        # 1. input layer (dim: 28x28x1)
        self.x = tf.placeholder(tf.float32, [None, 28, 28], 'x')
        x_reshaped = tf.reshape (self.x, [-1, 28, 28, 1])

        # 2. 3 × 3 conv layer, 32 filters, vertical/horizontal strides of 1
        W_conv = tf.get_variable(
                'W_conv',
                shape=(3,3,1,32), #0,1:filter size, 2: channel, 3: filter numbers
                initializer=tf.contrib.layers.xavier_initializer() #Xavier scheme
                )
        b_conv = tf.get_variable(
                'b_conv',
                shape=(32),
                initializer=tf.contrib.layers.xavier_initializer()
                )
        conv_layer = tf.nn.conv2d(
            input=x_reshaped,
            filter=W_conv,
            strides=[1,1,1,1], #0: image number, 1,2:h/v stride, 3: # of channel
            padding='SAME',
```

```
    name='conv_layer'
    )
conv_layer = tf.nn.bias_add(conv_layer, b_conv) #output dim: 28x28x32

# 3. ReLU activation
relu_conv = tf.nn.relu (conv_layer)

# 4. A batch normalization layer
mean, variance = tf.nn.moments(relu_conv, axes=[0])

bnorm_layer = tf.nn.batch_normalization(
    relu_conv,
    mean=mean,
    variance=variance,
    offset=None, scale=None,
    variance_epsilon=1e-3
    )

# 5. A 2 × 2 max pooling layer (dim: 14x14x32)
maxpool2x2_layer = tf.nn.max_pool(
    bnorm_layer,
    ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
    padding='SAME'
    )

# 6. Flatten layer
flatten_layer = tf.reshape(maxpool2x2_layer, [-1, 14*14*32])

# 7. Fully connected layer (with 784 output units, i.e. corresponding to each pixel)
W_fcl_784 = tf.get_variable(
        'W_fcl_784',
        shape=(14*14*32, 784),
        initializer=tf.contrib.layers.xavier_initializer() #Xavier scheme
        )
b_fcl_784 = tf.get_variable(
        'b_fcl_784',
        shape=(784),
        initializer=tf.contrib.layers.xavier_initializer() #Xavier scheme
        )
fullconn784_layer = tf.add(tf.matmul(flatten_layer, W_fcl_784), b_fcl_784)

#drop out
if with_dropout:
    fullconn784_layer = tf.nn.dropout(fullconn784_layer, keep_prob=p)

# 8. ReLU activation
relu_fcl = tf.nn.relu (fullconn784_layer)
```

```
# 9. Fully connected layer (with 10 output units, i.e. corresponding to each class)
W_fcl_10 = tf.get_variable(
	'W_fcl_10',
	shape=(784, 10),
	initializer=tf.contrib.layers.xavier_initializer() #Xavier scheme
	)
b_fcl_10 = tf.get_variable(
	'b_fcl_10',
	shape=(10),
	initializer=tf.contrib.layers.xavier_initializer() #Xavier scheme
	)
fullconn10_layer = tf.add(tf.matmul(relu_fcl, W_fcl_10), b_fcl_10)

# 10. Softmax output
y_hat = tf.nn.softmax(fullconn10_layer)

# 11. Cross Entropy loss
# normal loss
self.loss = tf.reduce_mean (tf.nn.softmax_cross_entropy_with_logits_v2(logits=y_hat, labels=self.y))
# loss with l2 regularizers
if with_regularizers:
   regularizers = tf.nn.l2_loss(W_conv) + tf.nn.l2_loss(W_fcl_784) + tf.nn.l2_loss(W_fcl_10)
   self.loss = tf.reduce_mean(self.loss + beta*regularizers)

# optimizer
self.optimizer = tf.train.AdamOptimizer(learning_rate=alpha).minimize(self.loss)

# compute accuracy
correct_prediction = tf.equal(tf.argmax(y_hat, 1), tf.argmax(self.y, 1))
self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

2.2 Model Training

Stochastic Gradient Descent class is implemented in *sgd.py*. Epochs and batch size are set to 50 and 32 respectivly by default

```
from cnn import ConvolutionalNeuralNetwork
import tensorflow as tf

class StochasticGradientDescent(object):
    def __init__(self, data, recorder, cnn):
        self.dt  = data
        self.rc  = recorder
        self.cnn = cnn

    def build_trainer(self, epochs=50, batch_size=32):
        dt = self.dt
        cnn = self.cnn
        init = tf.global_variables_initializer()

        with tf.Session() as sess:
            sess.run(init)
            n = len(dt.y_train_oh)
            x = cnn.x
            y = cnn.y

            # SGD
            for i in range(epochs):
                #shuffle
                x_shuffled, y_shuffled = dt.shuffle(dt.x_train, dt.y_train_oh)
                #go through all batches
                for j in range(0, n, batch_size):
                    x_batch, y_batch = x_shuffled[j:j+batch_size], y_shuffled[j:j+batch_size]
                    # run optimizer
                    sess.run (cnn.optimizer, feed_dict = {x: x_batch, y: y_batch})

                loss_train, acc_train = sess.run ([cnn.loss, cnn.accuracy], feed_dict = {x: dt.x_train, y: dt.y_train_oh})
                loss_valid, acc_valid = sess.run ([cnn.loss, cnn.accuracy], feed_dict = {x: dt.x_valid, y: dt.y_valid_oh})
                loss_test, acc_test = sess.run ([cnn.loss, cnn.accuracy], feed_dict = {x: dt.x_test, y: dt.y_test_oh})
                print ("Iteration: ", i,
                       "Train: ", loss_train, acc_train, ' \ '
                       "Valid: ", loss_valid, acc_valid, ' \ '
                       "Test: ", loss_test, acc_test)

                self.rc.train = self.rc.train.append({'loss': loss_train, 'accuracy': acc_train}, ignore_index=True)
                self.rc.valid = self.rc.valid.append({'loss': loss_valid, 'accuracy': acc_valid}, ignore_index=True)
                self.rc.test = self.rc.test.append({'loss': loss_test, 'accuracy': acc_test}, ignore_index=True)
```

Basic test is conducted in *main.py* with default/required values (learning rate: 10^-4, epochs: 50, batch size: 32)

```
from cnn import ConvolutionalNeuralNetwork
from sgd import StochasticGradientDescent
from data import Data
from recorder import Recorder
from plotter import Plotter

import matplotlib.pyplot as plt

dt = Data()
rc = Recorder()
cnn = ConvolutionalNeuralNetwork()
sgd = StochasticGradientDescent(dt, rc, cnn)
plotter = Plotter(rc)

dt.load('notMNIST.npz')

#%% 2.1 + 2.2 Convolutional Neural Network +  Stochastic Gradient Descent

cnn.build_model(
    seed=421,#tf seed
    alpha=1e-4, #learning rate for ADAM optimizer
    with_dropout=False, p=0.9, #dropout
    with_regularizers=False, beta=0.01 #regulizer
)
sgd.build_trainer (
    epochs=50,
    batch_size=32
)

# plot loss & accuracy
plotter.plot_train_valid_test('img/basic')
```

Below figures are the results for train, valid, and test sets:

Figure 2.2.1 Train Loss & Accuracy over 50 epochs

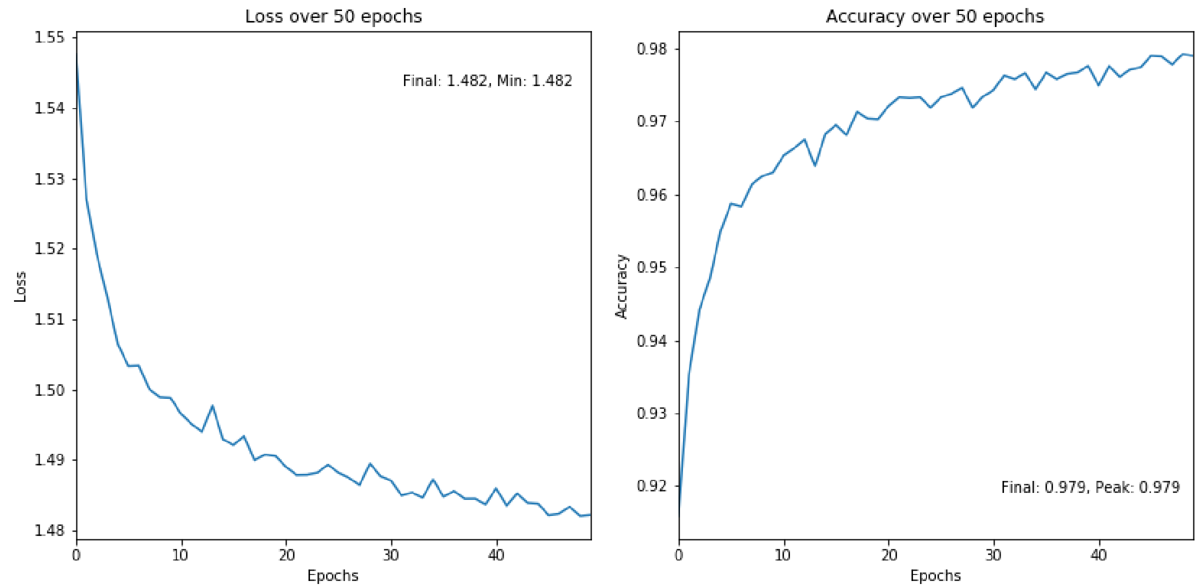


Figure 2.2.2 Valid Loss & Accuracy over 50 epochs

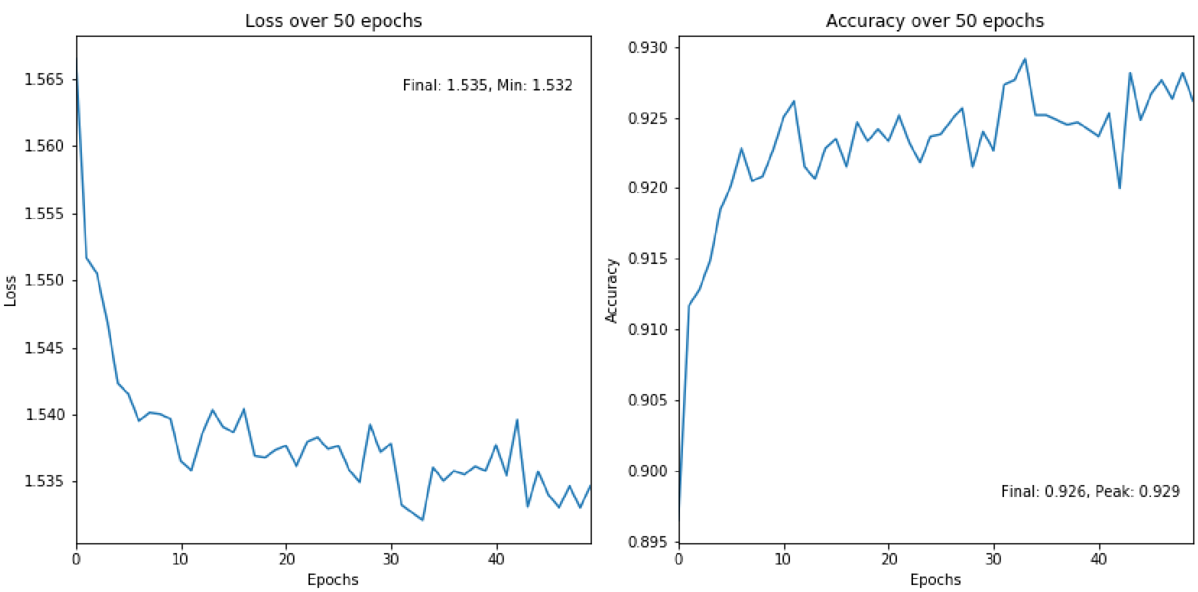


Figure 2.2.3 Test Loss & Accuracy over 50 epochs

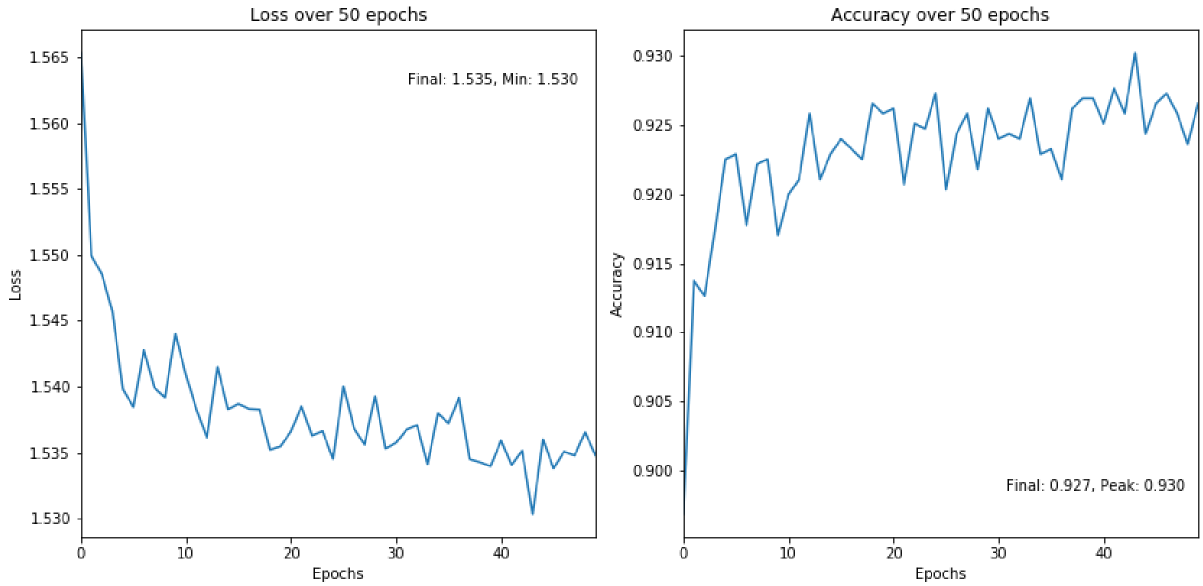


Table 2.2 Final training, validation and test accuracies

Set	Training	Validation	Test
Accuracy	**97.9%**	**92.6%**	**92.7%**

Observation:

- Loss and accuracy curves for validation and test sets seems to fluctutate strongly over the whole course of training. It means in every epoch, the model is trying to overfit to a batch of the training data. As a result, it is less fit to the validation and test data.
- At the end of training, training accuracy is able to reach 97.9%, meanwhile, valid and test accuracy only reach 92.6% & 92.7% respectively. The gap is approximately 5%. This means the model is overfitting towards the training set.
- However, validation and test accuracy curve converges very early at around 10 epochs.

2.3 Hyperparameter Investigation

1. L2 Normalization

1.1 Decay 0.01

Figure 2.3.1.1.1 Train Loss & Accuracy over 50 epochs

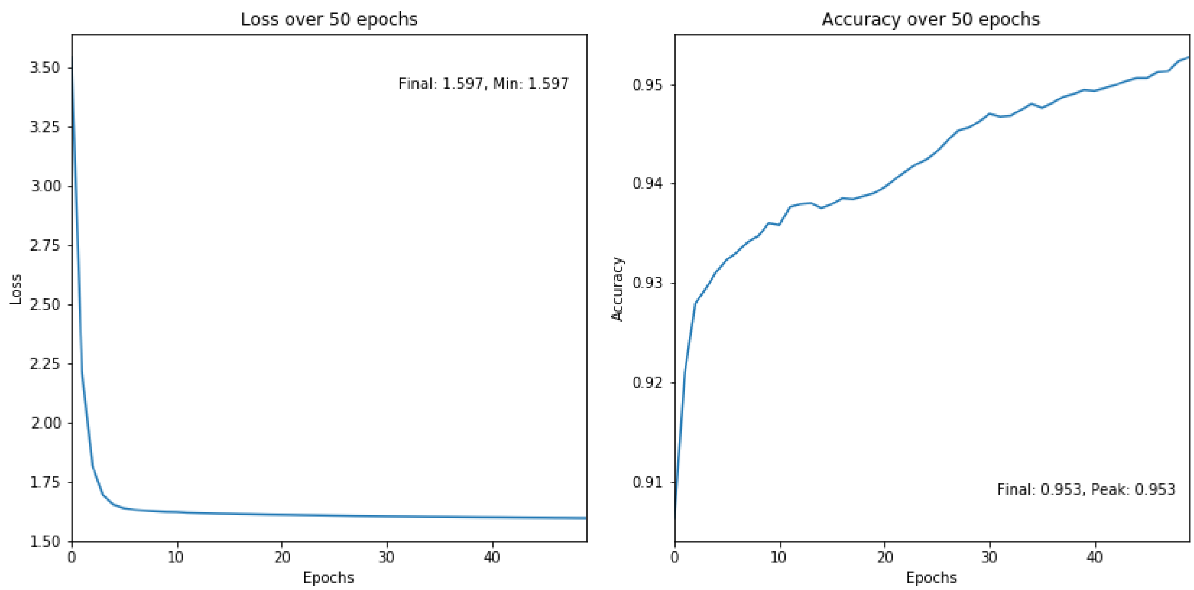


Figure 2.3.1.1.2 Valid Loss & Accuracy over 50 epochs

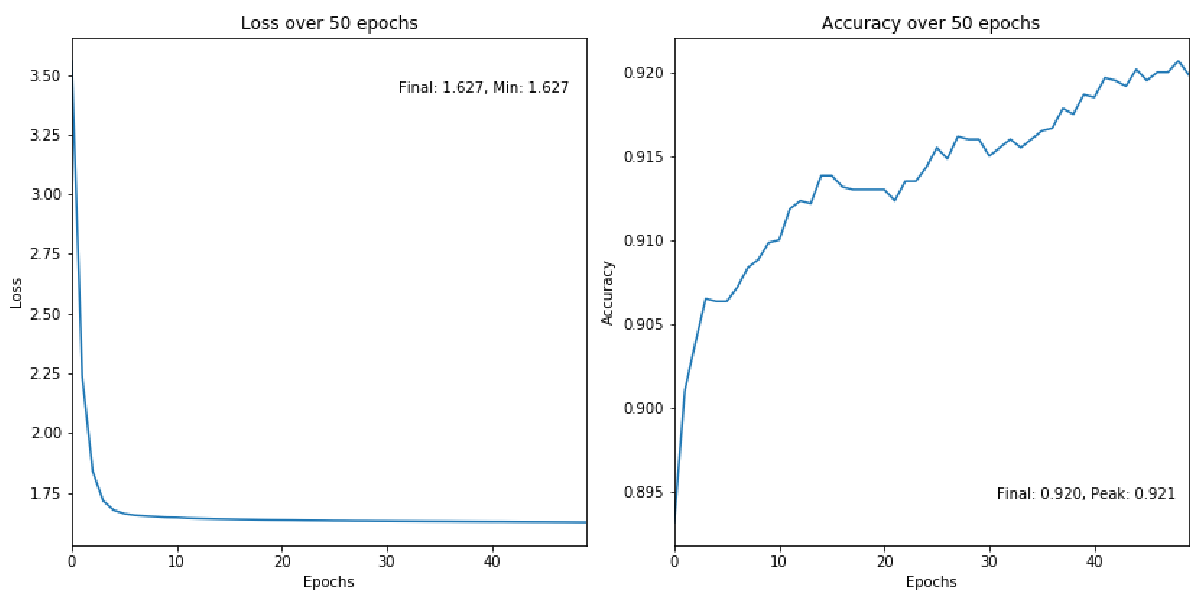


Figure 2.3.1.1.3 Test Loss & Accuracy over 50 epochs

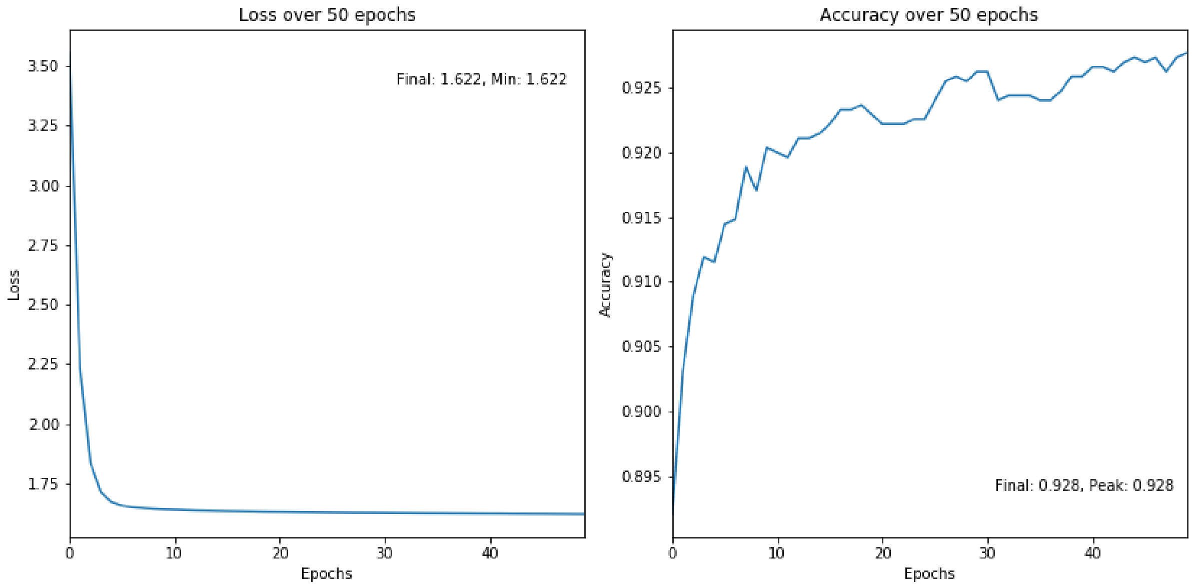


Table 2.2 Final training, validation and test accuracies

Set	Training	Validation	Test
Accuracy	**95.3%**	**92.0%**	**92.8%**

Observation:

- After L2 Normalization is applied, we can clearly see that loss and accuracy curves for training, validation, and test cases are smooth out. This means the model is less overfitting to any single batch of training data.
- Training accuracy is decreased by around 2.7% comparing to the very first training process, however, accuracy gap between training and validation/test cases becomes smaller at around 3%.
- The fact that training accuracy is lost by 2.7% proves that L2 regularization makes the model less overfitting to the training set.
- However, the 3 accuracy curves cannot converge as an effect of regularization (it probably requires more epochs).

1.2 Decay 0.1

Figure 2.3.1.2.1 Train Loss & Accuracy over 50 epochs

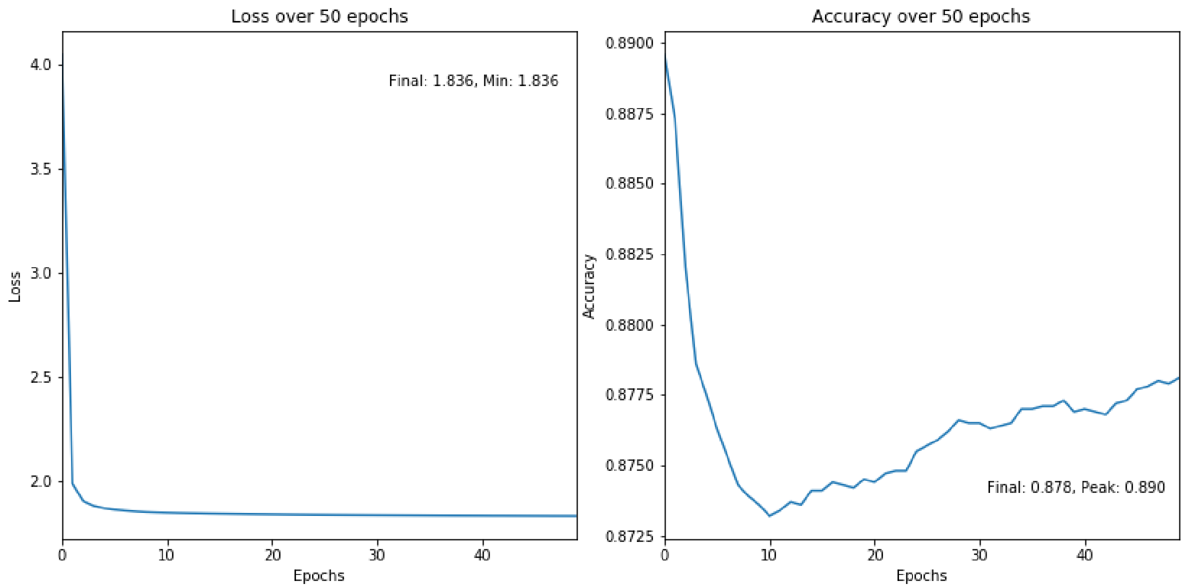


Figure 2.3.1.2.2 Valid Loss & Accuracy over 50 epochs

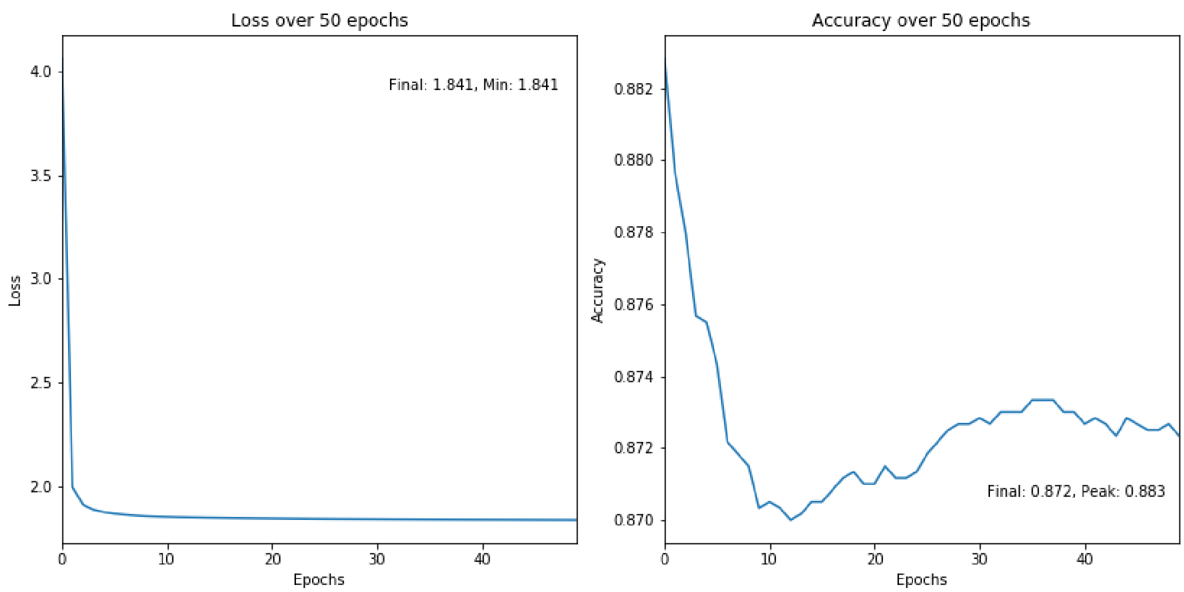


Figure 2.3.1.2.3 Test Loss & Accuracy over 50 epochs

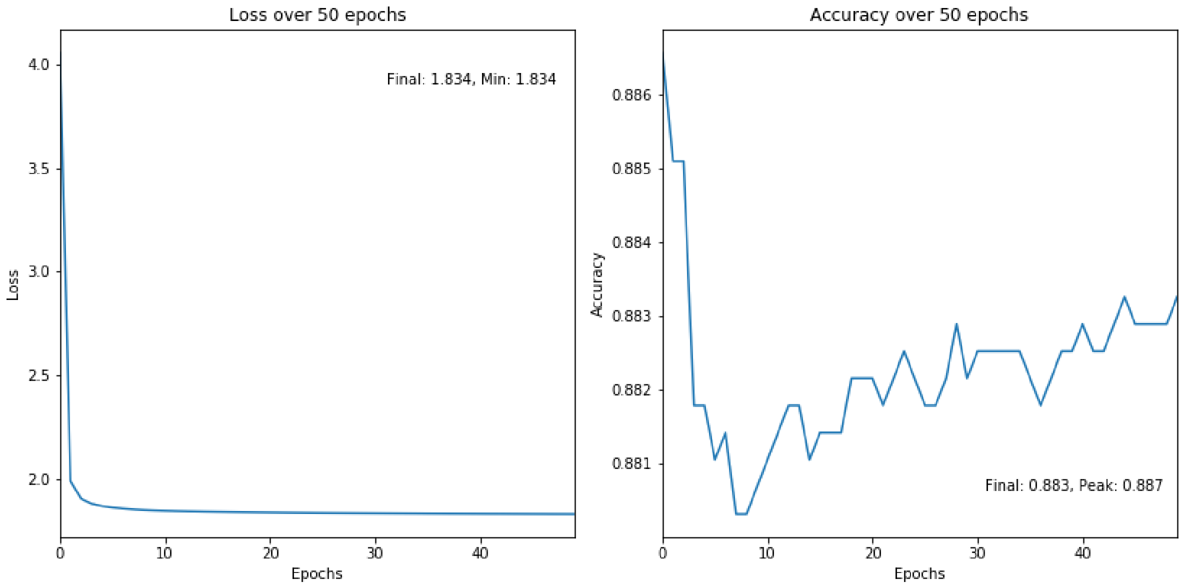


Table 2.2 Final training, validation and test accuracies

Set	Training	Validation	Test
Accuracy	**87.8%**	**87.2%**	**88.3%**

Observation:

- After increase the weight decay to 0.1, accuracy of 3 cases is suprisingly decreased for the first 10 epochs. For example, test accuracy decreases from 88.3% to 87.2%.
- The loss and accuracy curves seems to flutuate.
- Training, validation, and test accuracy is very close (difference is about 0.5%). This means the model almost solves the overfitting problem.
- However, comparing to the training without L2 regularization, the training, validation, and test accuracy drops by about 10.1%, 5.4%, and 4.4%. This means the model is starting to be underfitting.

1.3 Decay 0.5

Figure 2.3.1.3.1 Train Loss & Accuracy over 50 epochs

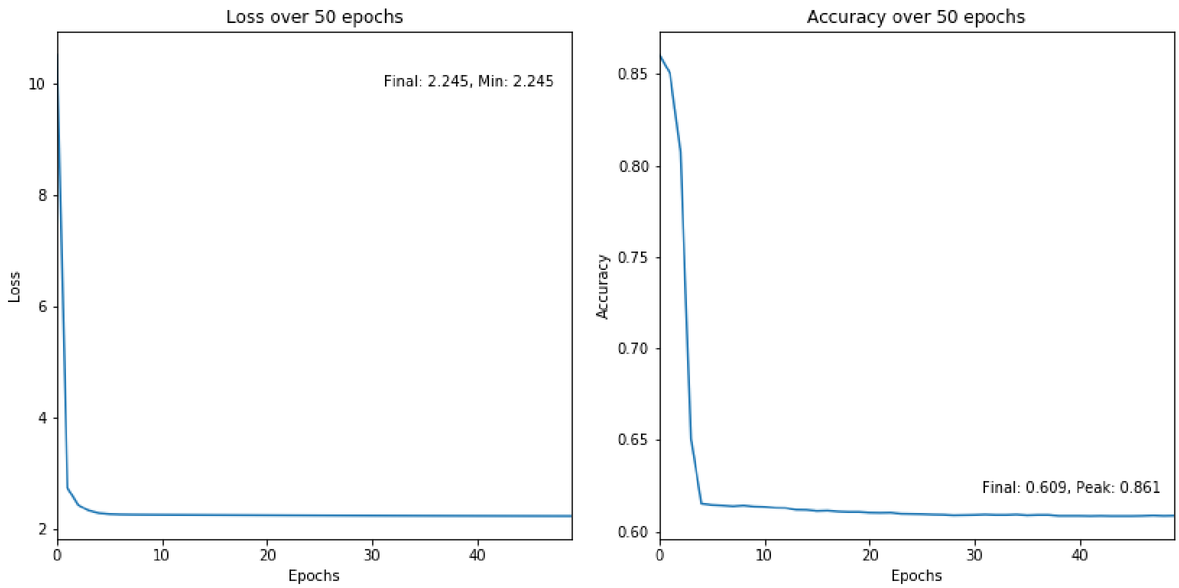


Figure 2.3.1.3.2 Valid Loss & Accuracy over 50 epochs

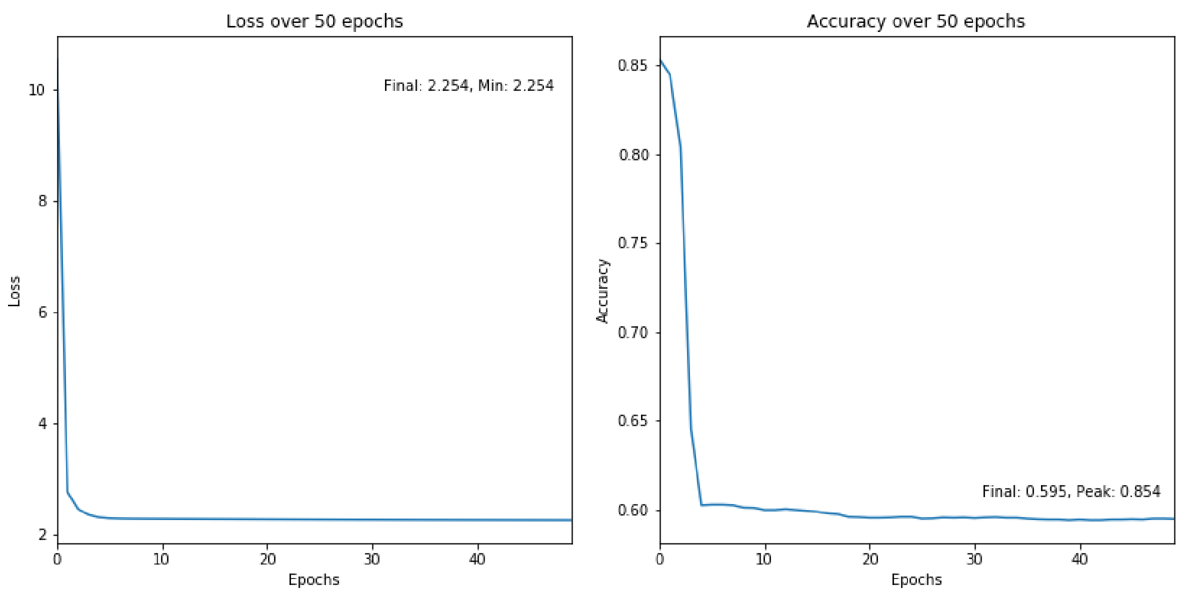


Figure 2.3.1.3.3 Test Loss & Accuracy over 50 epochs

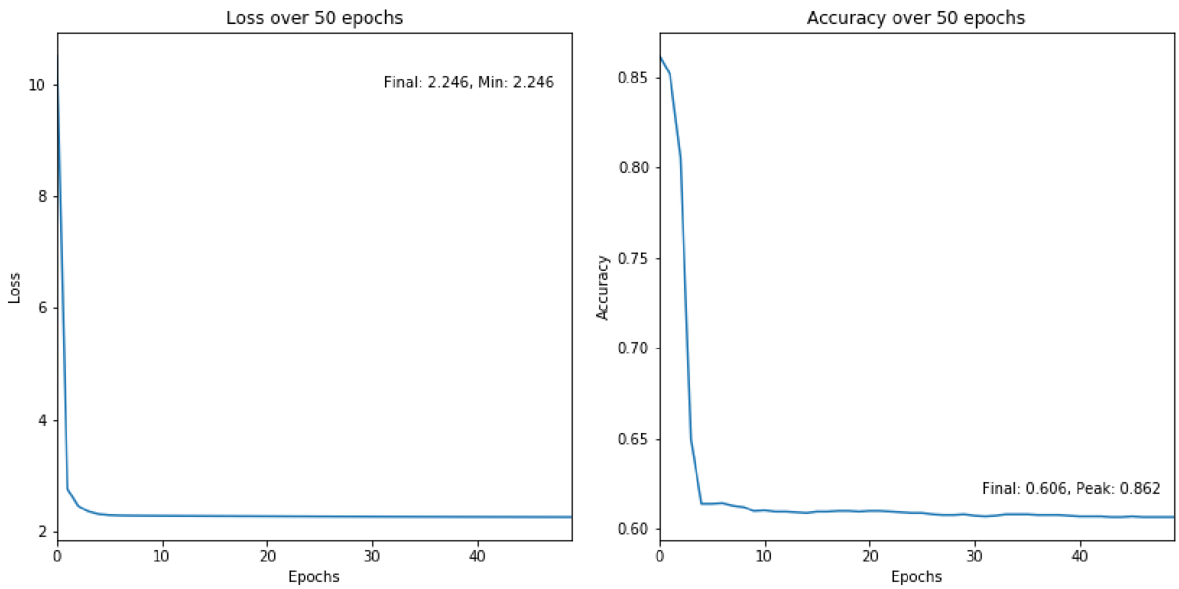


Table 2.2 Final training, validation and test accuracies

Set	Training	Validation	Test
Accuracy	**60.9%**	**59.5%**	**60.6%**

Observation:

- Accuracy curves of 3 cases completely drop from about 86% to 60% and converge after 5 epochs.
- Although accuracy gap is very small, accuracy losses by more than 30% for 3 sets comparing to the first experiment.
- We shall conclude that the model is already underfitting.

2. Dropout

2.1 P = 0.9

Figure 2.3.2.1.1 Training Loss & Accuracy over 50 epochs

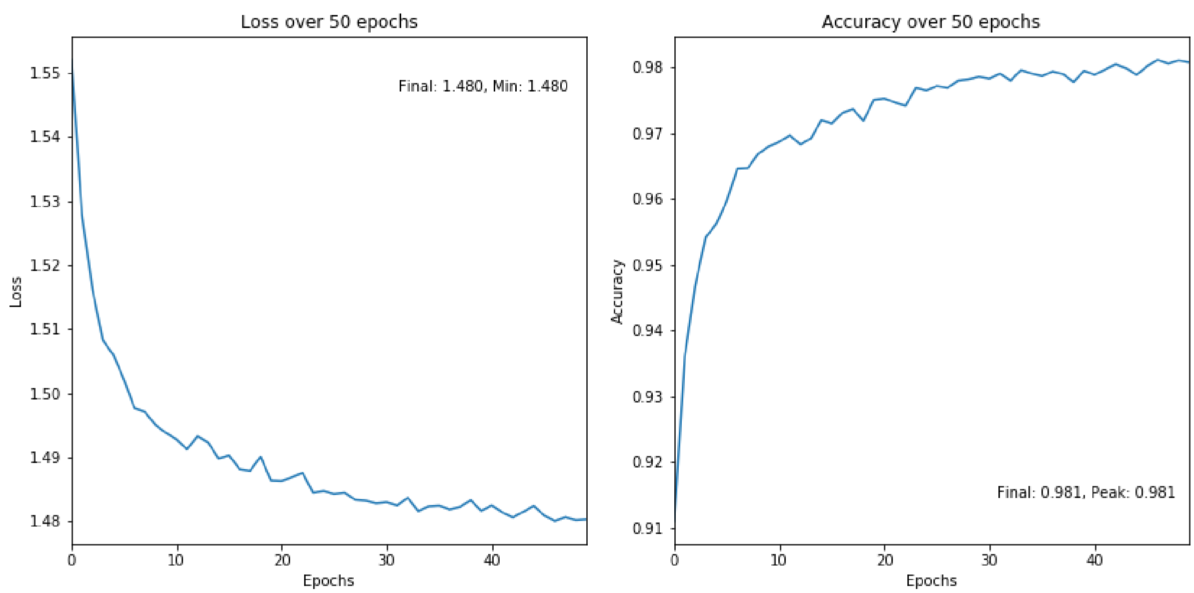


Figure 2.3.2.1.2 Validation Loss & Accuracy over 50 epochs

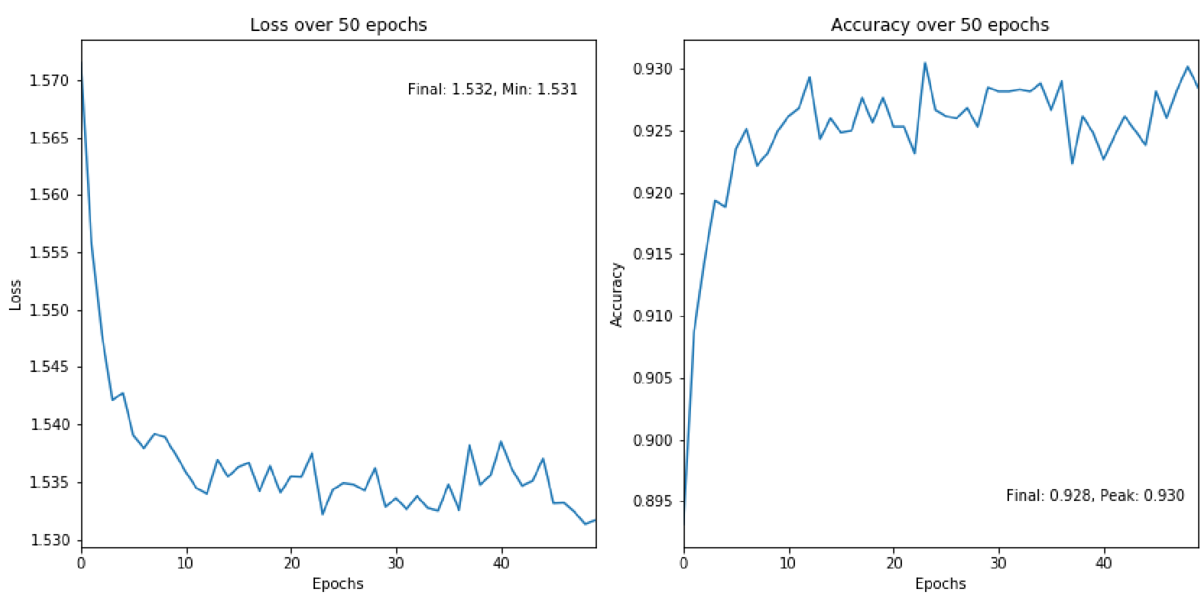


Figure 2.3.2.1.3 Test Loss & Accuracy over 50 epochs

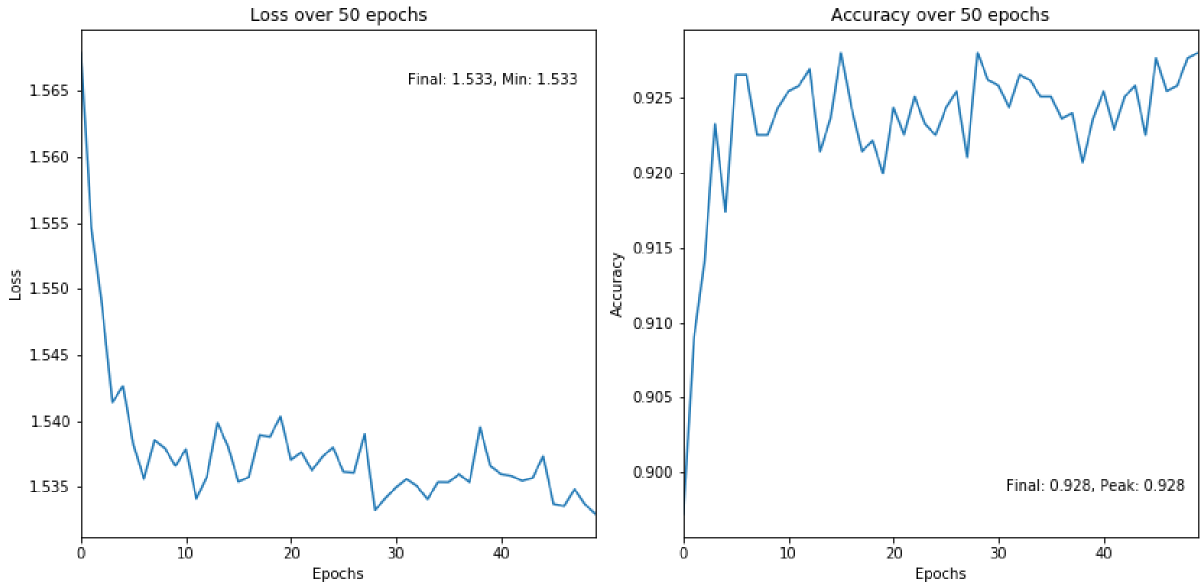


Table 2.3.1 Final training, validation and test accuracies

Set	Training	Validation	Test
Accuracy	**98.1%**	**92.8%**	**92.8%**

Observation:

- Dropout at p=0.9 seems to have small impact on the model by reducing the fluctuation of training accuracy curve.

2.2 P = 0.75

Figure 2.3.2.2.1 Training Loss & Accuracy over 50 epochs

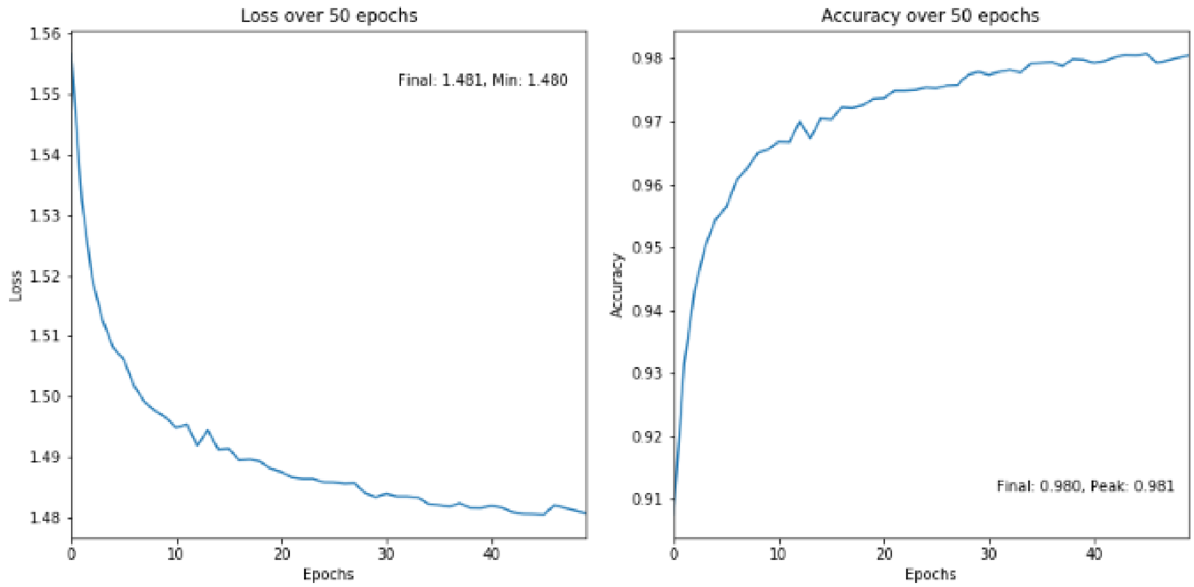


Figure 2.3.2.2.2 Validation Loss & Accuracy over 50 epochs

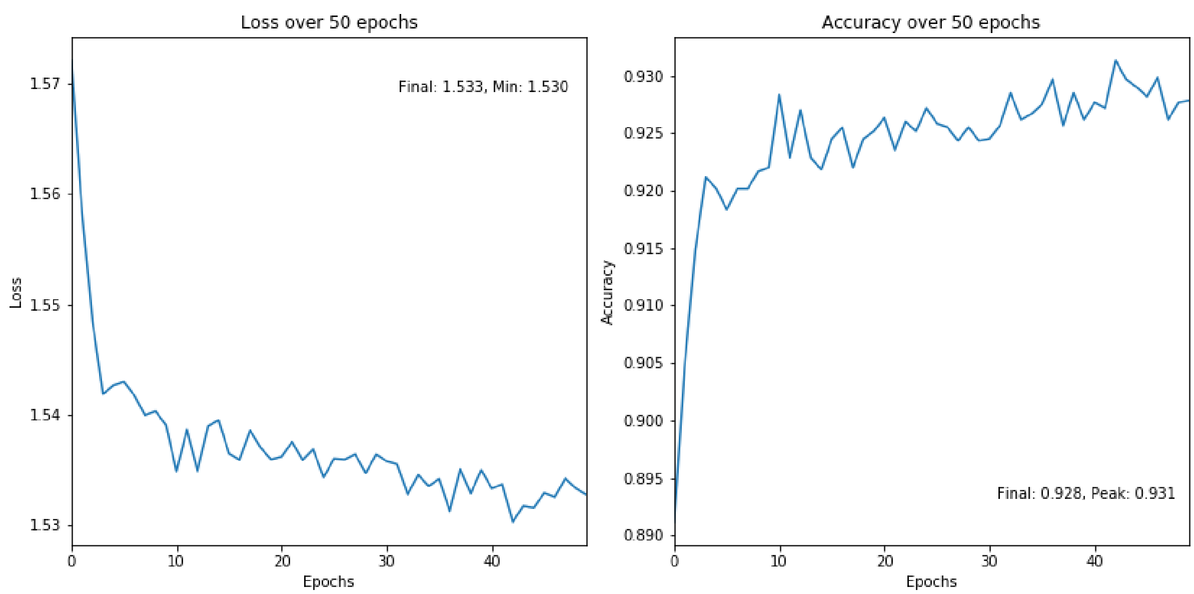


Figure 2.3.2.2.3 Test Loss & Accuracy over 50 epochs

2.3 P = 0.5

Figure 2.3.2.3.1 Training Loss & Accuracy over 50 epochs

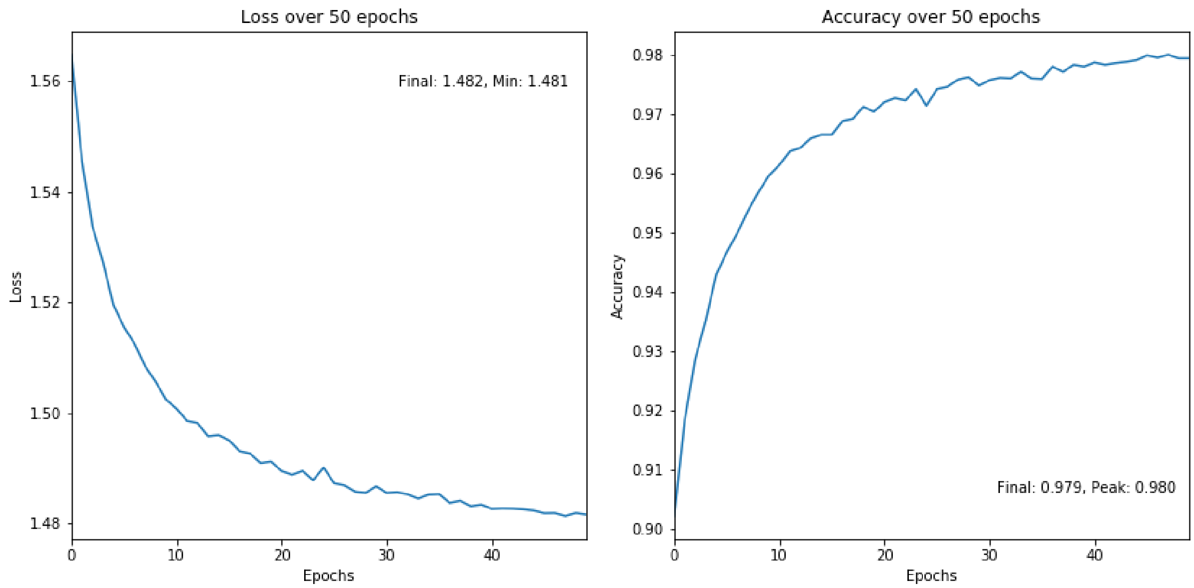


Figure 2.3.2.3.2 Validation Loss & Accuracy over 50 epochs

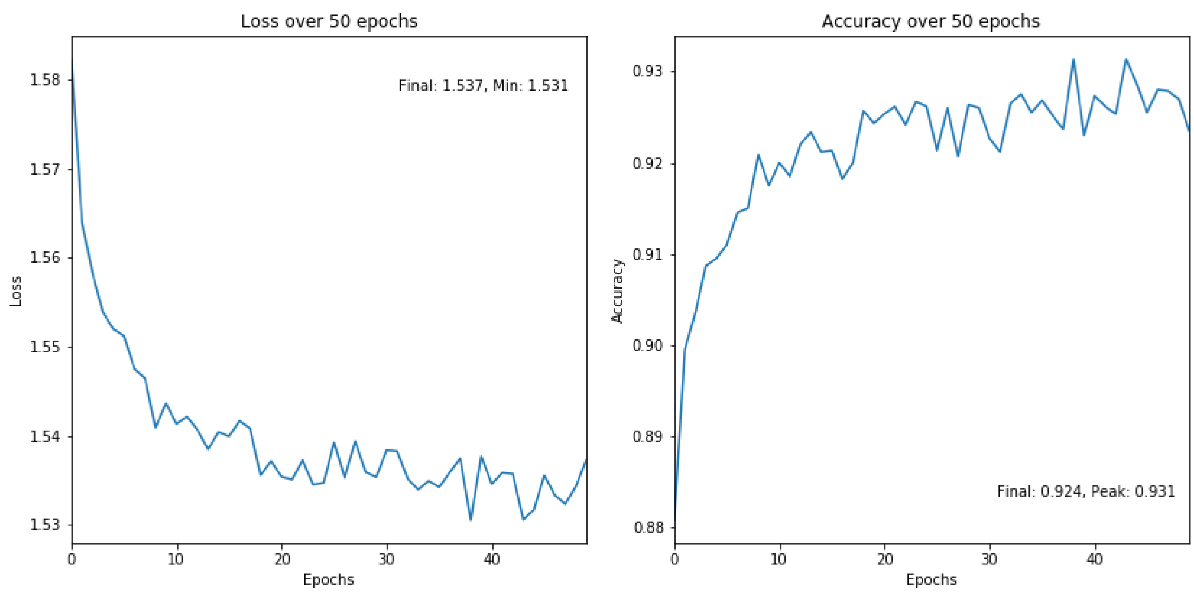
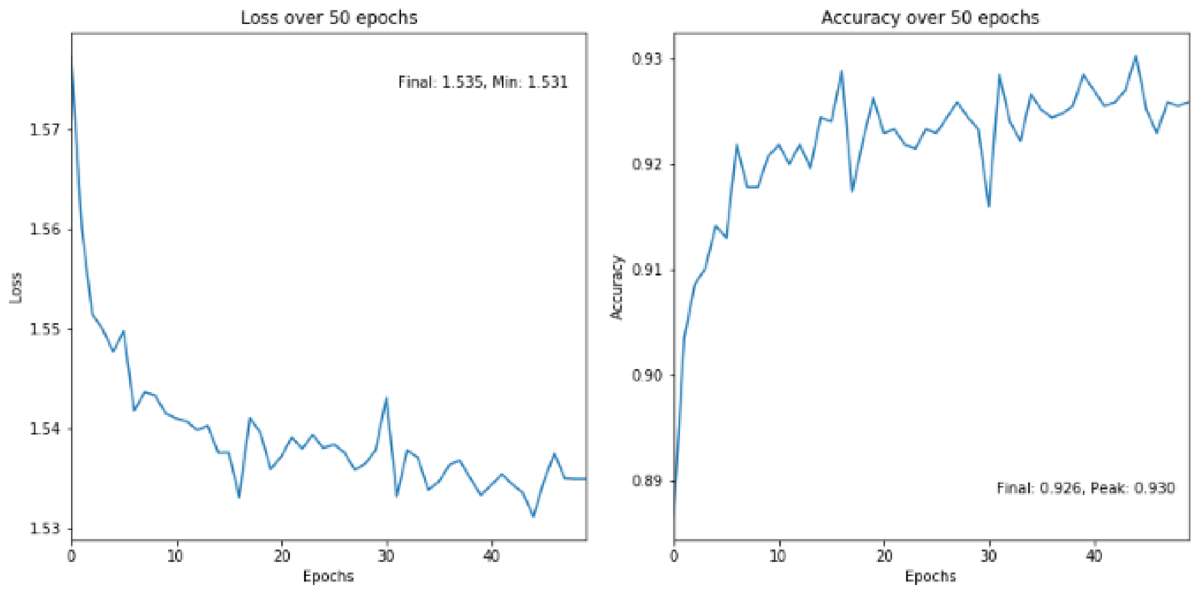


Figure 2.3.2.3.3 Test Loss & Accuracy over 50 epochs



Appendix

A.2.2 Model Training: Stochastic Gradient Descent

A.2.2.1 Basic Test: CNN + SGD

```
from cnn import ConvolutionalNeuralNetwork
from sgd import StochasticGradientDescent
from data import Data
from recorder import Recorder
from plotter import Plotter

import matplotlib.pyplot as plt

dt = Data()
rc = Recorder()
cnn = ConvolutionalNeuralNetwork()
sgd = StochasticGradientDescent(dt, rc, cnn)
plotter = Plotter(rc)

dt.load('notMNIST.npz')

#%% 2.1 + 2.2 Convolutional Neural Network +  Stochastic Gradient Descent

cnn.build_model(
    seed=421,#tf seed
    alpha=1e-4, #learning rate for ADAM optimizer
    with_dropout=False, p=0.9, #dropout
    with_regularizers=False, beta=0.01 #regulizer
)
sgd.build_trainer (
    epochs=50,
    batch_size=32
)

# plot loss & accuracy
plotter.plot_train_valid_test('img/basic')
```

A.2.3.1 L2 Normalization

A.2.3.1.1 Decay 0.01

```
#%% 2.3 L2 Decay 0.01

cnn.build_model(
    seed=421,#tf seed
    alpha=1e-4, #learning rate for ADAM optimizer
    with_dropout=False, p=0.9, #dropout
    with_regularizers=True, beta=0.01 #regulizer beta: weight decay
)
sgd.build_trainer (
    epochs=50,
    batch_size=32
)

# plot loss & accuracy
plotter.plot_train_valid_test('img/decay001')
```

A.2.3.1.2 Decay 0.1

```
#%% 2.3 L2 Decay 0.1

cnn.build_model(
    seed=421,#tf seed
    alpha=1e-4, #learning rate for ADAM optimizer
    with_dropout=False, p=0.9, #dropout
    with_regularizers=True, beta=0.1 #regulizer beta: weight decay
)
sgd.build_trainer (
    epochs=50,
    batch_size=32
)

# plot loss & accuracy
plotter.plot_train_valid_test('img/decay01')
```

A.2.3.1.2 Decay 0.5

```
#%% 2.3 L2 Decay 0.5

cnn.build_model(
    seed=421,#tf seed
    alpha=1e-4, #learning rate for ADAM optimizer
    with_dropout=False, p=0.9, #dropout
    with_regularizers=True, beta=0.5 #regulizer beta: weight decay
)
sgd.build_trainer (
    epochs=50,
    batch_size=32
)

# plot loss & accuracy
plotter.plot_train_valid_test('img/decay05')
```