ECE421 – Introduction to Machine Learning

Assignment 2

Neural Networks

Shiyi Zhang	Jiani Li	Tianyi Wang
Contribution: 33%	Contribution: 33%	Contribution: 33%

Part I. Neural Networks using Numpy

1. Helper Functions

a. ReLU()

```
def relu(x):
    relu_x = np.maximum(x, 0)
    return relu_x
```

b. softmax()

```
def softmax(x):
    x = x - np.max(x, axis=1, keepdims=True)
    sm_x = np.exp(x) / np.sum(np.exp(x), axis=1, keepdims=True)
    return sm_x
```

c. compute()

```
def computeLayer(X, W, b):
    return np.dot(X, W) + b
```

d. averageCE()

```
def CE(target, prediction):
    ce = (-1) * np.mean(np.multiply(target, np.log(prediction)))
    return ce
```

e. gradCE()

```
def gradCE(target, prediction):
    return (prediction - target)
```

- *let s be the value after softmax function*

$dL/do = dL/ds * ds/do$

- The average entropy is : $1/N \sum_{1}^{N} \sum_{k=1}^{10} y_k log(s_k) \Rightarrow dL/ds = -1/N \sum_{k=1}^{10} \frac{y_k}{s_k}$
- $since\ s(o_j) = exp(o_j)/\sum_{k=1}^{10} exp(o_k)$

  There are 2 conditions depending on the value of i and j:

  $ds_j/do_i = s_j\ (1 - o_j),\ if\ i = j$;

  $ds_j/do_i = -s_j\ s_i,\ if\ i \neq j$;

- Apply the chain rule:

$$dL/do_i = -1/N \sum_{k=1}^{10} \frac{y_k}{s_k} * ds_j/do_i$$

$$= -1/N * [y_i(1 - S_i) + \sum_{k \neq i}^{10} \frac{y_k}{s_k}(-s_k s_i)]$$

$$= 1/N * [-y_i + y_i S_i + \sum_{k \neq i}^{10} y_k s_i]$$

$$= 1/N * [-y_i + s_i \sum_{k=1}^{10} y_k s_i] = 1/N\ (s_i - y_k)$$

=> Thus, the total gradient of the cross entropy loss of the inputs to the softmax function:

dL/do = prediction - target

2. Backpropagation Derivation

a. $dL/dWo = dL/do * do/dWo = hiddenlayer_{output}.Transpose\ .\ gradCE(target,\ prediction)$

Shape: (num_hidden_units, 10000) dot product (10000, 10) = (num_hidden_units, 10)

b. $dL/dBo = dL/do * do/dBo = vector\ of\ ones.Transpose\ .\ gradCE(target,\ prediction)$

Shape: (1, 10000) dot product (10000, 10) = (1, 10)

c. $dL/dWh = input\ data.Transpose\ .\ derivative\ of\ relu * gradCE(target,\ prediction)\ .\ Wo.Transpose$

derivative of relu is 0 *if value before relu is negative, and derivative is* 1 *otherwise.*

Shape: (784, 10000) dot product ((10000, num_hidden_units) by element multiply ((10000, 10) dot product (10, num_hidden_units)))

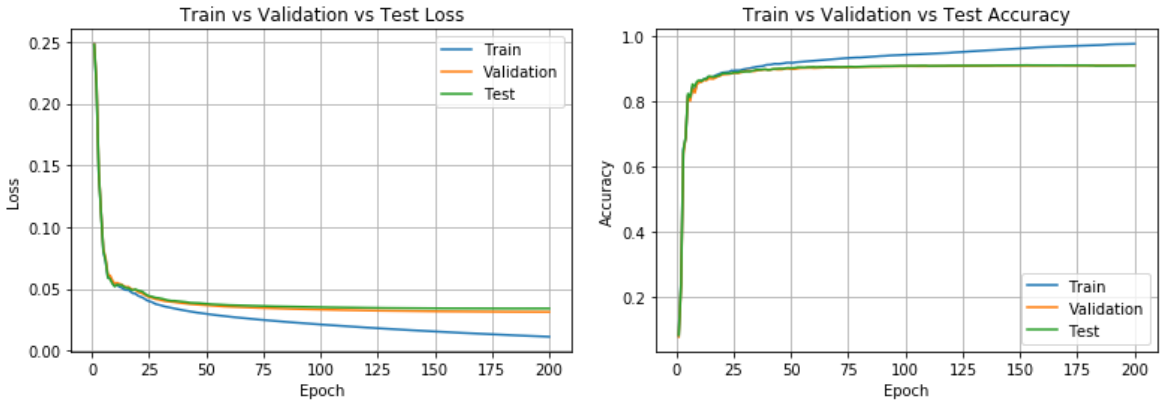
d. $dL/dBh = vector\ of\ ones.\ derivative\ of\ relu * gradCE(target,\ prediction)\ .\ Wo.Transpose$

Shape: (1, 10000) dot product ((10000, num_hidden_units) by element multiply ((10000, 10) dot product (10, num_hidden_units)))

3. Learning

Accuracy and loss plot for training, validation and testing dataset.

Where number of hidden units H = 1000, number of iterations = 200, $\gamma = 0.9$, $\alpha = 1e\text{-}5$



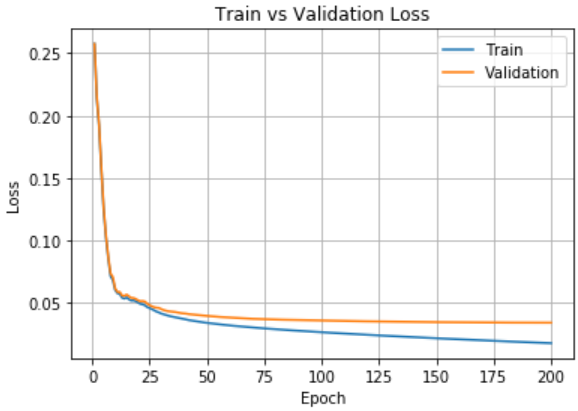
Final test accuracy is: 0.908957415565345

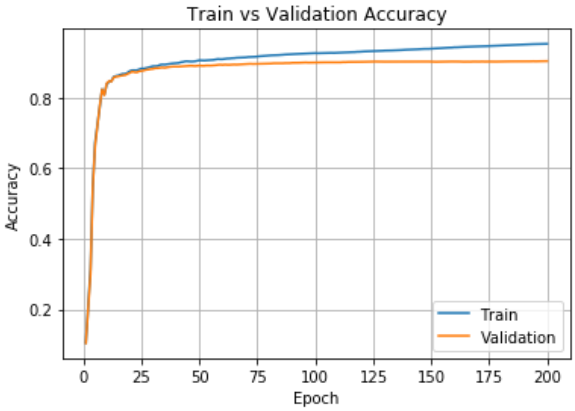
4. Hyperparameter investigation

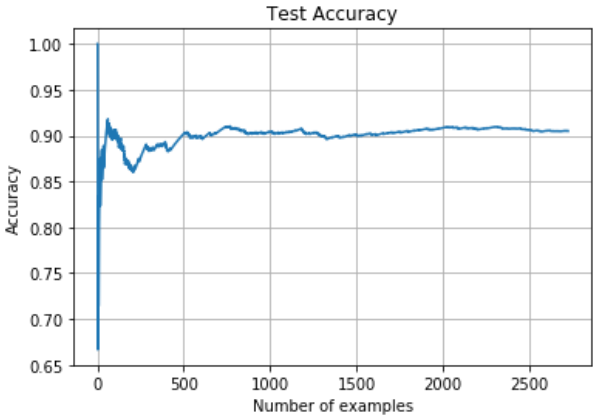
4.1 Number of hidden units

a. Accuracy and loss plot for training, validation datasets, and accuracy plot for testing dataset.

Where number of hidden units H = 100, number of iterations = 200, $\gamma = 0.9$, $\alpha = 1e\text{-}5$

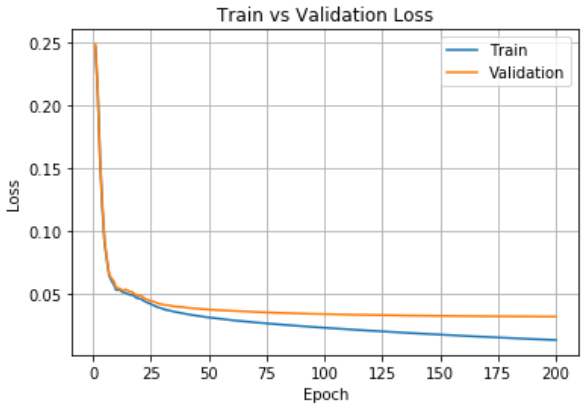


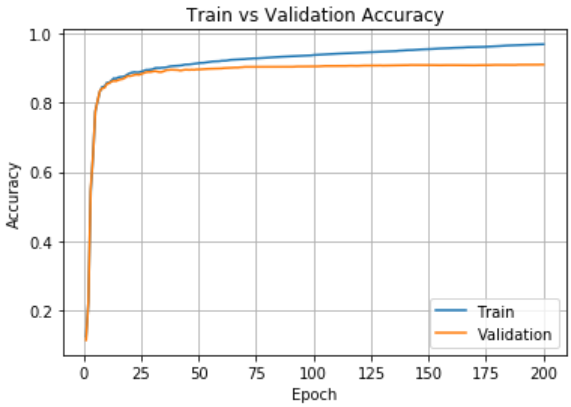


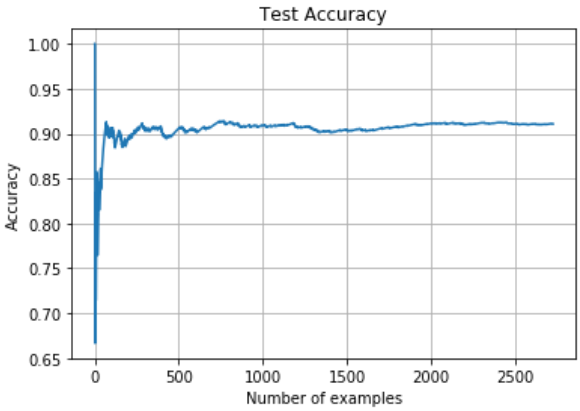


Final test accuracy is: 0.9049192364170338

b. Accuracy and loss plot for training, validation datasets, and accuracy plot for testing dataset. Where number of hidden units H = 500, number of iterations = 200, γ = 0.9, α = 1e-5

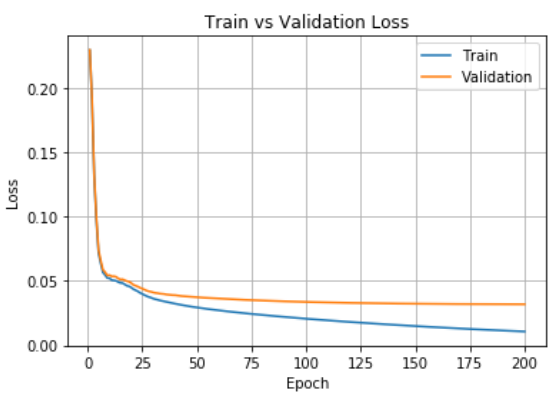


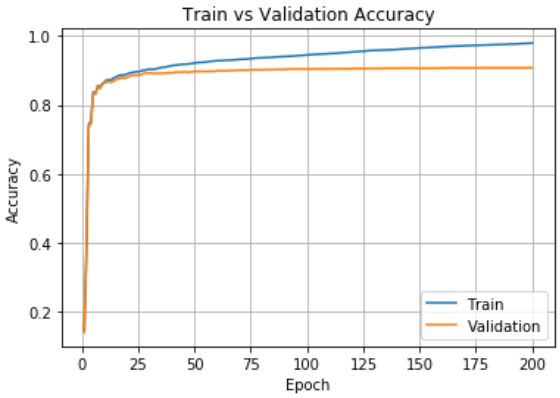


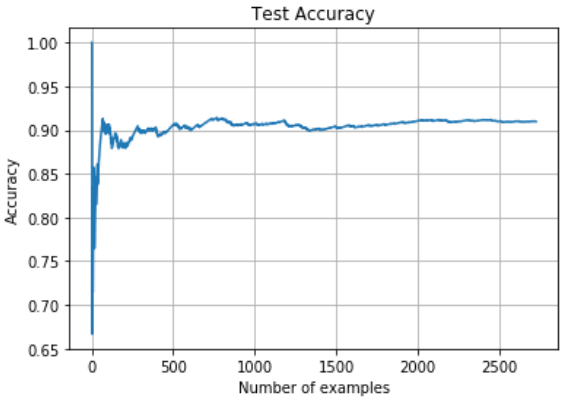


Final test accuracy is: 0.9107929515418502

c. Accuracy and loss plot for training, validation datasets, and accuracy plot for testing dataset. Where number of hidden units H = 2000, number of iterations = 200, $\gamma = 0.9$, $\alpha = 1e\text{-}5$







Final test accuracy is: 0.9096916299559471

Summary: the final test accuracy of the 3 number of hidden units models are all high and close to each other. 500 hidden units works best for our problem, 2000 the next and 100 the third. It is because 2000 is so large that the model is too complicated and overfits the training set, thus it does not work well on the test set. And 100 is too small that our ANN is too simple to do the classification.

4.2 Early stopping

As shown in the plot, when the epoch number reached 50, the loss of validation set hardly changes, therefore, if we stop at 50 iterations, we would get a similar test accuracy and a lot of computation time will be saved. At epoch = 50, training accuracy = 0.92, validation and test accuracy = 0.90.

Part II. Neural Networks in Tensorflow

1. Model implementation

```
def CNN(x, y, learning_rate, beta1, beta2, epsilon):
    tf.set_random_seed(421)
    x = tf.placeholder("float", [None, 28,28,1])
    y = tf.placeholder("float", [None, 10])
    weights = {
        'w1': tf.get_variable('w1', shape=(3,3,1,32), initializer=tf.contrib.layers.xavier_initializer()),
        'w2': tf.get_variable('w2', shape=(14*14*32,784), initializer=tf.contrib.layers.xavier_initializer()),
        'w3': tf.get_variable('w3', shape=(784,10), initializer=tf.contrib.layers.xavier_initializer()),
    }
    biases = {
        'b1': tf.get_variable('b1', shape=(32), initializer=tf.contrib.layers.xavier_initializer()),
        'b2': tf.get_variable('b2', shape=(784), initializer=tf.contrib.layers.xavier_initializer()),
        'b3': tf.get_variable('b3', shape=(10), initializer=tf.contrib.layers.xavier_initializer()),
    }
    train_data = tf.cast(x, 'float32')
    conv = tf.nn.conv2d(train_data, weights['w1'], strides=[1, 1, 1, 1], use_cudnn_on_gpu=True, padding='SAME')
    conv = tf.nn.bias_add(conv, biases['b1'])
    activation1 = tf.nn.relu(conv)

    mean, var = tf.nn.moments(activation1, [0, 1, 2])
    offset = tf.Variable(tf.zeros([32]))
    scale = tf.Variable(tf.ones([32]))
    batch_norm = tf.nn.batch_normalization(activation1, mean, var, offset, scale, epsilon)
```

```
max_pool = tf.nn.max_pool2d(batch_norm, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
flatten = tf.reshape(max_pool, [-1, weights['w2'].get_shape().as_list()[0]])
fc1 = tf.add(tf.matmul(flatten, weights['w2']), biases['b2'])
activation2 = tf.nn.relu(fc1)
fc2 = tf.add(tf.matmul(activation2, weights['w3']), biases['b3'])

#prediction = tf.nn.softmax(fc2)
CEloss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(y, fc2))

num_correct = tf.equal(tf.argmax(fc2, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(num_correct, tf.float32))


optimizer = tf.train.AdamOptimizer(learning_rate, beta1, beta2, epsilon).minimize(CEloss)

return fc2, CEloss, accuracy, optimizer, weights, biases, x, y
```

2. Model Training

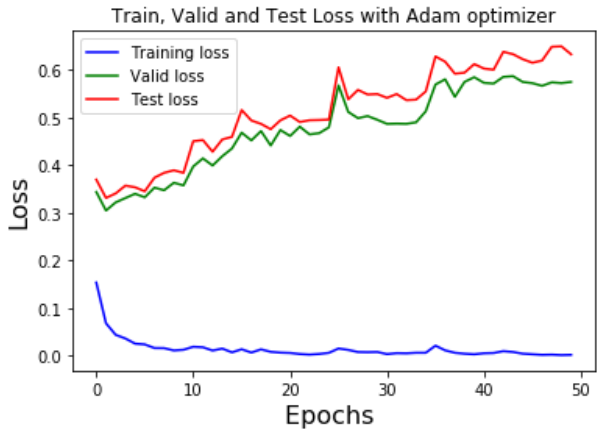
Model training using SGD for a batch size of 32, for 50 epochs and the Adam optimizer for learning rate of $\alpha = 1e-4$.

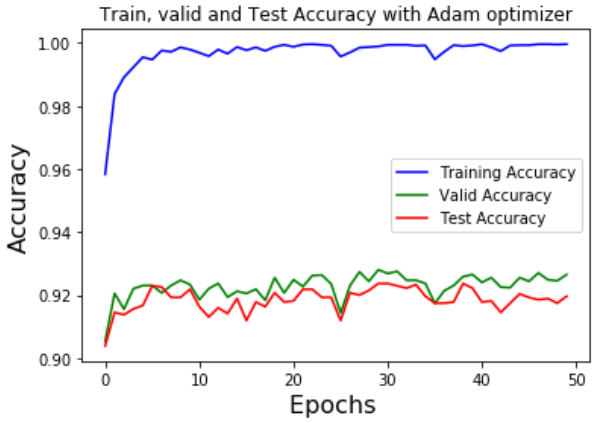
```
SGD(trainData, validData, testData, 32, 1e-4, 50, 1e-8, 0.9, 0.999, train_target, valid_target, test_target)
```

```
def SGD(trainData, validData, testData, batch_size, learning_rate, epoch, epsilon, beta1, beta2, train_target, valid_target, test_target):
  trainData = trainData.reshape(-1, 28, 28, 1)
  validData = validData.reshape(-1, 28, 28, 1)
  testData = testData.reshape(-1, 28, 28, 1)
  tf.reset_default_graph()
  [prediction, CEloss, accuracy, optimizer, weights, biases, x, y] = CNN(trainData, train_target, learning_rate, beta1, beta2, epsilon)
  init_op = tf.global_variables_initializer()

  num_train_batch = trainData.shape[0] // batch_size

  train_loss = np.zeros(epoch)
  train_accuracy = np.zeros(epoch)
  valid_loss = np.zeros(epoch)
  valid_accuracy = np.zeros(epoch)
  test_loss = np.zeros(epoch)
  test_accuracy = np.zeros(epoch)

  sess = tf.InteractiveSession()
  sess.run(init_op)

  for i in range(epoch):
    train_idx = np.arange(trainData.shape[0])
    np.random.shuffle(train_idx)
    trainData = trainData[train_idx]
    train_target = train_target[train_idx]
```

```
    for j in range(num_train_batch):
      train_x_batch = trainData[j*batch_size:(j+1)*batch_size, :]
      train_y_batch = train_target[j*batch_size:(j+1)*batch_size, :]
      opt = sess.run(optimizer, feed_dict={x: train_x_batch, y: train_y_batch})
    train_loss_current, train_accuracy_current = sess.run([CEloss, accuracy], feed_dict={x: trainData, y: train_target})
    train_loss[i] = train_loss_current
    train_accuracy[i] = train_accuracy_current

    valid_loss_current, valid_accuracy_current = sess.run([CEloss, accuracy], feed_dict={x:validData, y:valid_target})
    valid_loss[i] = valid_loss_current
    valid_accuracy[i] = valid_accuracy_current

    test_loss_current, test_accuracy_current = sess.run([CEloss, accuracy], feed_dict={x:testData, y:test_target})
    test_loss[i] = test_loss_current
    test_accuracy[i] = test_accuracy_current

    print("epoch =", i+1, "train loss: ", train_loss[i], "valid loss: ", valid_loss[i])
    print("epoch =", i+1, "train accuracy: ", train_accuracy[i], "valid accuracy: ", valid_accuracy[i])
    print("epoch =", i+1, "test accuracy: ", test_accuracy[i])
  sess.close()
```

```
plt.plot(range(len(train_loss)), train_loss, 'b', label='Training loss')
plt.plot(range(len(train_loss)), valid_loss, 'g', label='Valid loss')
plt.plot(range(len(train_loss)), test_loss, 'r', label='Test loss')
plt.title('Train, Valid and Test Loss with Adam optimizer')
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.legend()
plt.figure()
plt.show()

plt.plot(range(len(train_loss)), train_accuracy, 'b', label='Training Accuracy')
plt.plot(range(len(train_loss)), valid_accuracy, 'g', label='Valid Accuracy')
plt.plot(range(len(train_loss)), test_accuracy, 'r', label='Test Accuracy')
plt.title('Train, valid and Test Accuracy with Adam optimizer')
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.legend()
plt.figure()
plt.show()
```

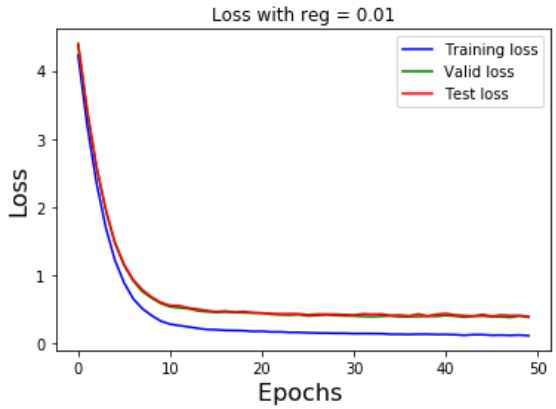


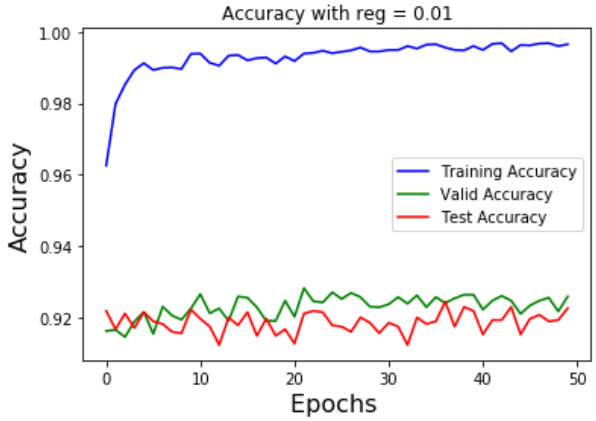


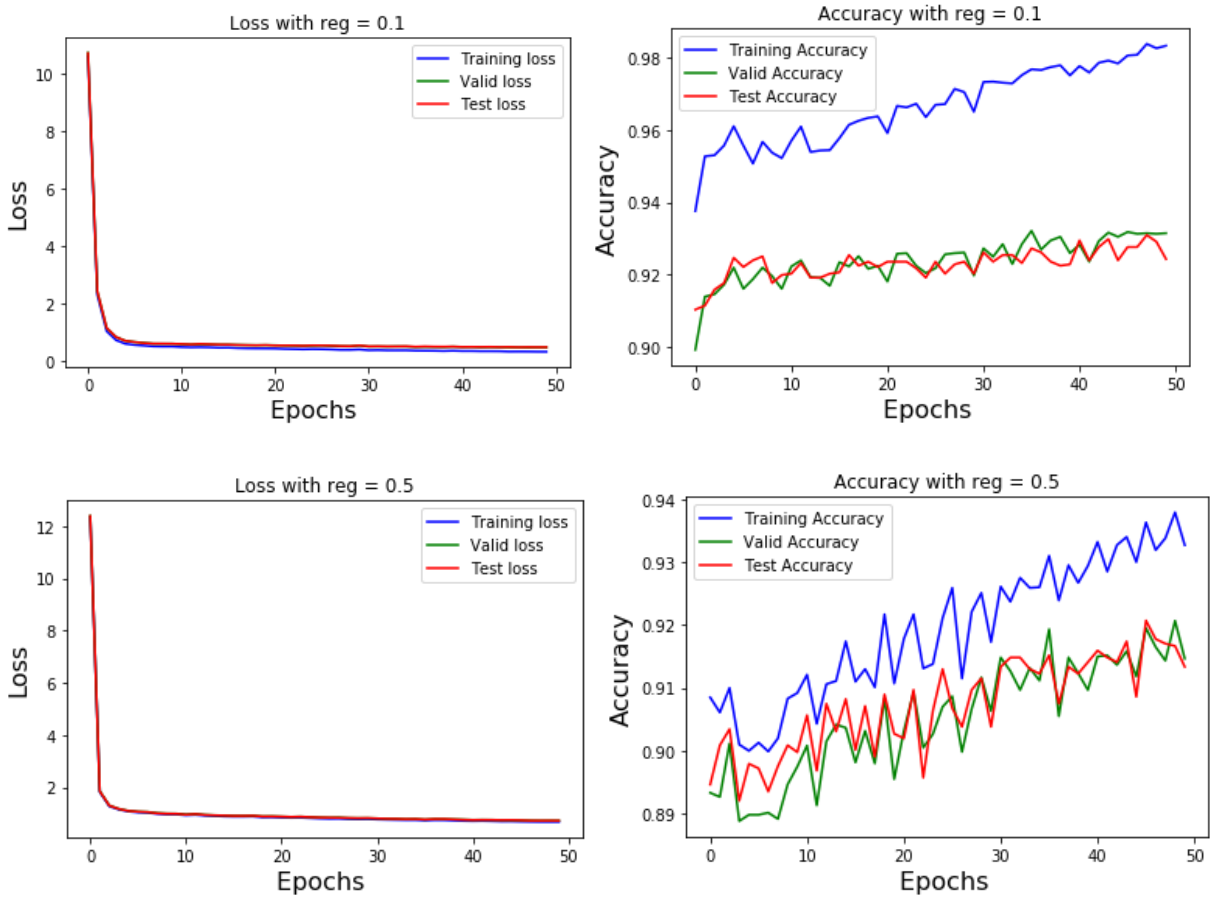
3. Hyperparameter Investigation

3.1 L2 Regularization

λ	Training Accuracy	Validation Accuracy	Test Accuracy
0.01	0.9965000152587891	0.9258333444595337	0.9225403666496277
0.1	0.9832000136375427	0.9315000176429749	0.9243758916854858
0.5	0.932699978351593	0.9146666526794434	0.9133626818656921



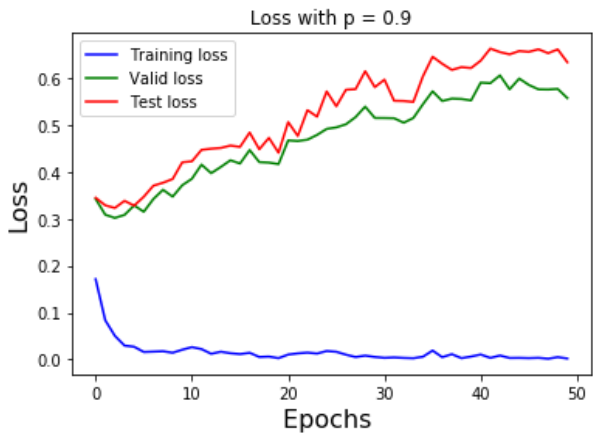


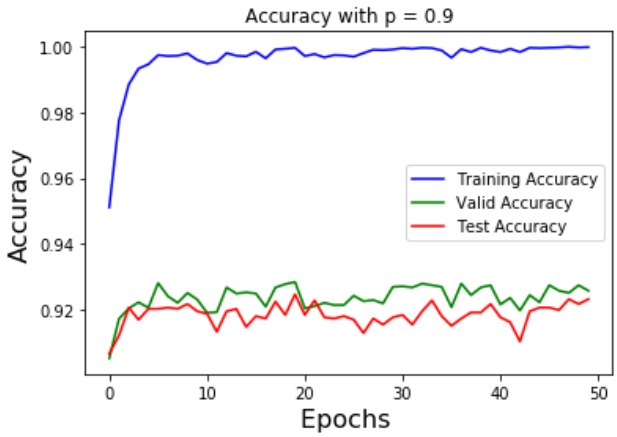


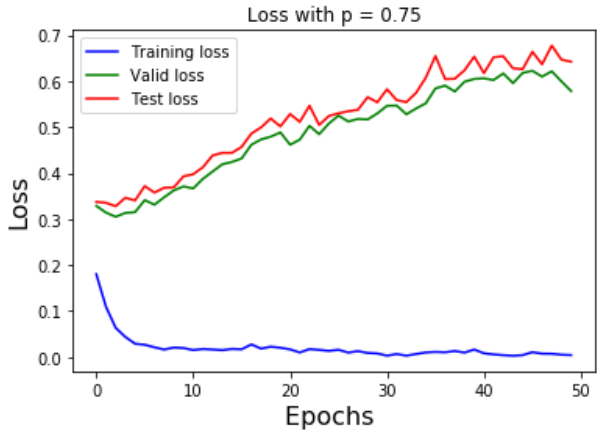
We add L2 regularization to our model to prevent overfitting. Regularization will penalize neurons with heavy weights. λ is the regularization parameter. As λ increases, the penalty term also increases, which is equivalent to adding more noise to the model and can lead to lower accuracy for training, validation and testing.

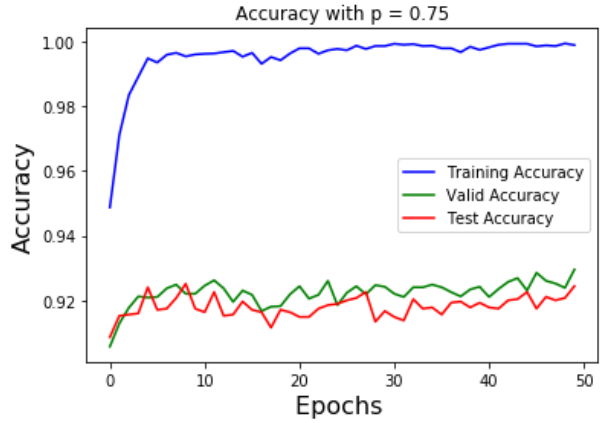
3.2 Dropout

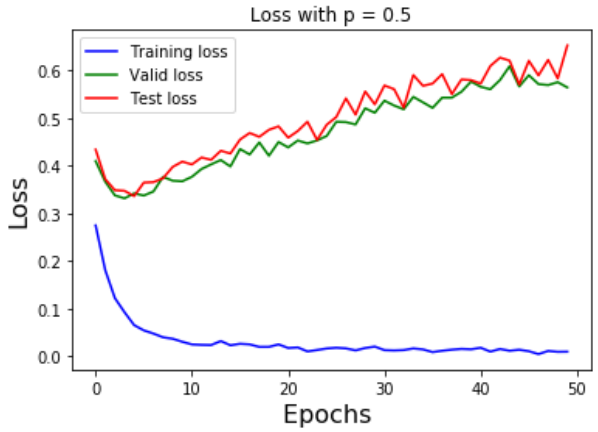
p	Training Accuracy	Validation Accuracy	Test Accuracy
0.9	0.9997000098228455	0.9258333444595337	0.923274576663971
0.75	0.9988999962806702	0.9294999837875366	0.9243758916854858
0.5	0.9983000159263611	0.9261666536331177	0.9170337915420532

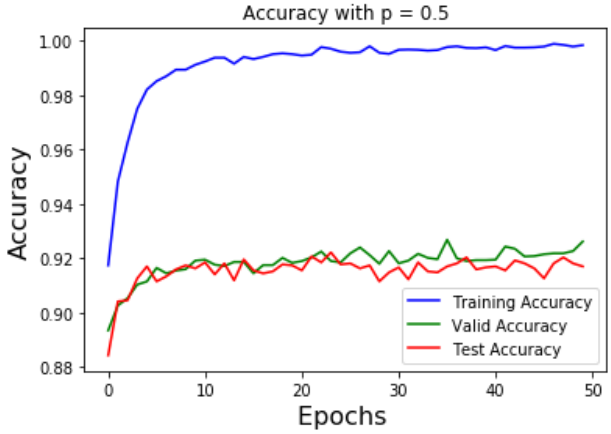












Dropout means zeroing out some hidden units in a certain layer, which is applied to prevent overfitting. p is the keeping probability, indicating the probability of each hidden unit that will not be zeroed out. As the keeping probability decreases, more units will be set to zero, which is equivalent to adding more noise to the model and can lead to lower accuracy for training, validation and testing.