Mario Ruiz

mjruiz@syr.edu | ist-718

IST-718: Lab #3

Professor Fox

**Introduction**

The MNIST dataset is a collection of images created by the National Institute of Standards and Technology (NIST). Each handwritten image in the dataset consists of 28 x 28 pixels. With the provided 60,000 images for training, and 10,000 for testing, users have typically found between 93 to near 100% accuracy. The successful implementation of varying models has been both a triumph, and problem for computer vision. Specifically, since the optimal solution has been attained, predicting handwritten images is no longer one of the defacto challenges.

To further expand on the earlier handwritten problem, the fashion MNIST dataset has emerged. Like the original MNIST dataset, the fashion variant consists of the same number of train and test images, each being the same dimension. However, each image in the collection is a series of ten possible clothing type.

In this study, the fashion MNIST dataset will be compared using neural network, as well as support vector machines (svm). The analysis will largely focus on comparing accuracy when predicting with either algorithms. In the future, additional algorithms can be tested, as well as overall benchmarking between each algorithm and prediction.

**Data Preparation**

The dataset used for this study was obtained directly from the fashion-mnist[[1]](#footnote-1) repository. Then each corresponding file were committed into a dedicated code crepository[[2]](#footnote-2):

* t10k-images-idx3-ubyte.gz
* t10k-images-idx1-ubyte.gz
* train-images-idx3-ubyte.gz
* train-labels-idx3-ubyte.gz

Since this study has been a simplified computer vision problem, no additional data scrubbing was performed. However, future studies could possibly improve the analysis by including additional images that are not the target fashion groups.

Since the datasets were downloaded locally, two different functions were used to load the datasets into python. First, the input\_data.read\_data\_sets function was used, and needed by the corresponding neural network. Specifically, the tensorflow implementation requires the input object to be the base.Dataset type[[3]](#footnote-3). Next the load\_mnist[[4]](#footnote-4) was used for the svm modelling. This function simply returns both images and labels. Since the fashion-mnist repository[[5]](#footnote-5) was not cloned, the latter function was copied locally into the svm.py[[6]](#footnote-6).

**Results**

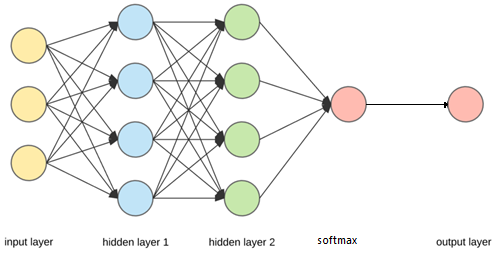
Once the required dataset was loaded into python, a brief exploratory verified that the dataset is the expected fashion MNIST:



**Figure 1**. Visual of first 25 fashion MNIST images. The code used to generate this image can be reviewed in Appendix A below.

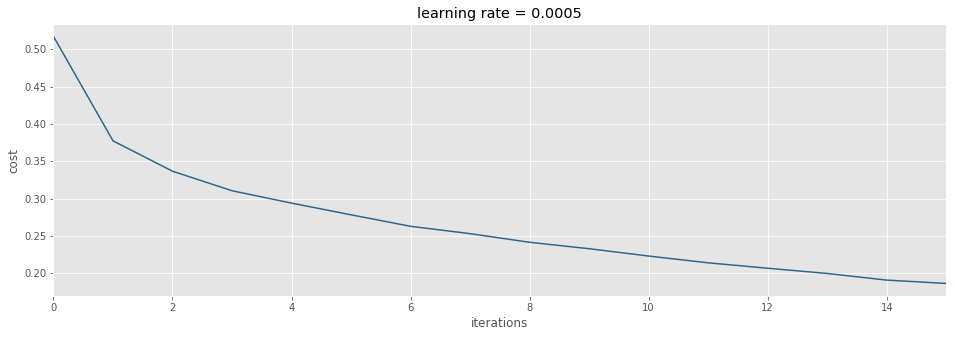
A neural network was trained using two hidden layers each implementing a linear function, followed by a ReLU function. Finally, a softmax function creates an output for each fashion target class.

An initialize\_parameters function was defined for the input and output of each neural network layer. The aggregation of each of these layer attributes was used to define the forward propagation behavior of the overall neural network:



**Figure 2**. Representation of two hidden layers, followed by the softmax and output layer. The associated code for this image can be reviewed in Appendix B below.

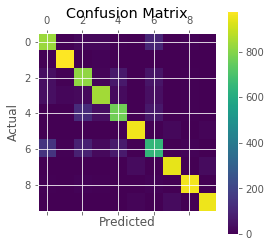
In addition to the forward propagation, the AdamOptimizer[[7]](#footnote-7) was used for each epoch iteration to specify the type of backward propagation. Default values, with a learning rate of 0.0005 portrayed that the number of iterations stabilize at about 14 iterations.



**Figure 3**. Neural network cost indicates stabilization at 14 iterations. The associated code for this image can be reviewed in Appendix C below.

Finally, the overall train accuracy for the generated neural network was 0.935, while the test accuracy was 0.889 (see Appendix D).

**Note:** the implemented neural network was an adaptation from Vivian Rajkuma[[8]](#footnote-8).



SVM Trained Classifier Accuracy: 0.8723

Predicted Values: [9 2 1 ..., 8 1 5]

Classifier on Validation Images: 0.8723

Confusion Matrix:

[[834 5 21 20 4 2 104 0 9 1]

[ 4 981 1 8 3 0 3 0 0 0]

[ 39 6 819 12 66 0 54 0 4 0]

[ 41 16 25 846 29 0 39 0 4 0]

[ 2 1 121 39 769 0 63 0 5 0]

[ 0 0 0 1 0 960 0 22 3 14]

[147 4 91 26 72 0 650 0 10 0]

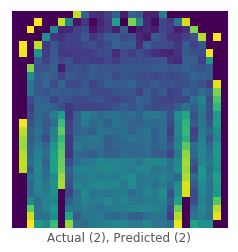
[ 0 0 0 0 0 27 0 944 0 29]

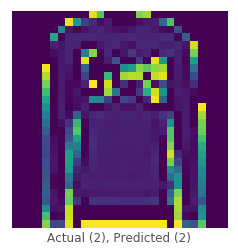
[ 6 0 10 4 2 3 11 2 962 0]

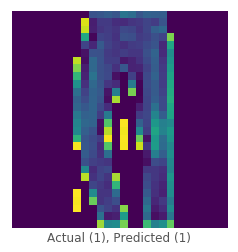
[ 0 1 0 0 0 12 1 28 0 958]]

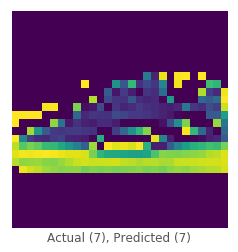
**Figure 4:** An svm model was created with 87.2% accuracy for both the train and test dataset. The associated code can be reviewed in Appendix E below.

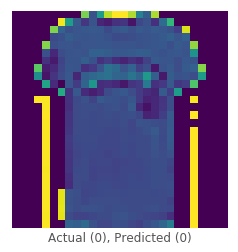
The results of the svm prediction matches earlier test accuracy:

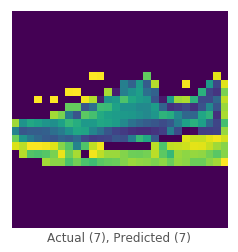


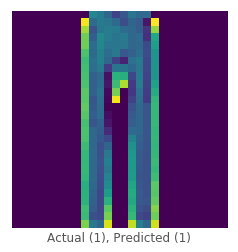


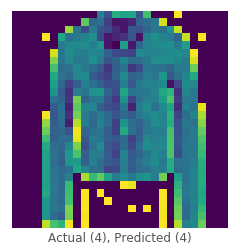


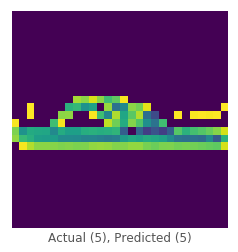


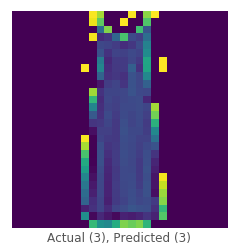


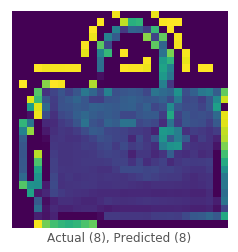


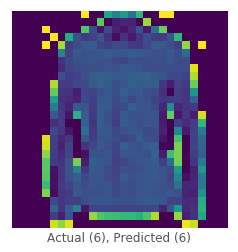


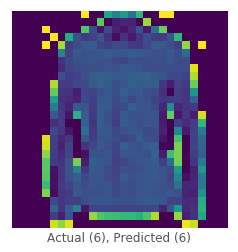


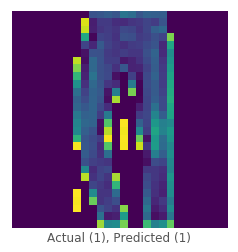


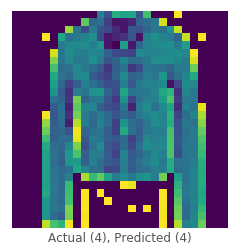












**Figure 5:** An svm fashion prediction. The associated code can be reviewed in Appendix F below.

It is generally known that svm is much less performant than the neural network. For this reason, the corresponding benchmark has been omitted as an exercise. However, this study clearly illustrates that the neural network implementation is better performing. However, the degree of performance is not significantly large. Therefore, it would be interesting to see if adjustments in the neural network weights, number of layers and different backward propagation techniques could increase the difference between the two implementations. Similarly, changing the parameters for the svm, including the kernel and margin of the separating plane would be an interesting future study.

Compared to the traditional MNIST number classification, the fashion variant is not as accurate. Specifically, the MNIST techniques have been found in the upper 98-99%. However, as stated earlier, more clever techniques could be imposed to draw higher results. This could involve a series of ensemble learners. Though a simple additional approach could involve appending to the provided dataset. The additional data could be as simple as a set of images not belonging to any of the target classes. For example, additional images of non-fashion images could be used during the train. This would be analogous to extending the one versus rest, taken advantage of within the sklearn framework.

**Conclusions**

Overall this study has succeeded in demonstrating how a multilayer neural network can outperform the traditional support vector machine (svm). However, as stated in the results, it would be interesting to explore the adjustment of parameters for both techniques. Specifically, adjusting the penalty and changing the kernel to radial in the svm. However, this may not improve the performance gap between the two approaches, since the neural network implements linear regression.

At a higher level, the fashion MNIST dataset is more interesting than its predecessor. More improvements, and additional techniques will be needed to achieve the same or close level of accuracy as the numbers example. Instead of implementing the ensemble support vector machine, ensembling with other modeling techniques could smooth the prediction results with competitive accuracy.

**Appendix A**

First 25 fashion MNIST images:

viz\_tensor(range(25), fashion\_mnist, labels)

Implements the respective function:

def viz\_tensor(instances, fashion\_mnist, labels):

plt.figure(figsize=(10,10))

for i, instance in enumerate(instances):

sample = fashion\_mnist.train.images[instance].reshape(28,28)

sample\_label = np.where(fashion\_mnist.train.labels[instance] == 1)[0][0]

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(sample, cmap='Greys')

plt.xlabel('index = {index} ({label})'.format(

index=sample\_label,

label=labels[sample\_label]

))

**Appendix B**

Network layer attributes including weight:

def initialize\_parameters(hidden\_1, hidden\_2, n\_input, n\_classes):

tf.set\_random\_seed(11)

# first hidden layer

W1 = tf.get\_variable(

'W1',

[hidden\_1, n\_input],

initializer=tf.contrib.layers.xavier\_initializer(seed=11)

)

b1 = tf.get\_variable('b1',

[hidden\_1, 1],

initializer=tf.zeros\_initializer()

)

# second hidden layer

W2 = tf.get\_variable(

'W2',

[hidden\_2, hidden\_1],

initializer=tf.contrib.layers.xavier\_initializer(seed=11)

)

b2 = tf.get\_variable(

'b2',

[hidden\_2, 1],

initializer=tf.zeros\_initializer()

)

# output layer

W3 = tf.get\_variable(

'W3',

[n\_classes, hidden\_2],

initializer=tf.contrib.layers.xavier\_initializer(seed=42)

)

b3 = tf.get\_variable(

'b3',

[n\_classes, 1],

initializer=tf.zeros\_initializer()

)

return { 'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2, 'W3': W3, 'b3': b3 }

Forward propagation definition:

def forward\_propagation(X, parameters):

# Retrieve parameters from dictionary

W1 = parameters['W1']

b1 = parameters['b1']

W2 = parameters['W2']

b2 = parameters['b2']

W3 = parameters['W3']

b3 = parameters['b3']

# Carry out forward propagation

Z1 = tf.add(tf.matmul(W1,X), b1)

A1 = tf.nn.relu(Z1)

Z2 = tf.add(tf.matmul(W2,A1), b2)

A2 = tf.nn.relu(Z2)

Z3 = tf.add(tf.matmul(W3,A2), b3)

return Z3

**Appendix C**

Calculate cost of neural network per iteration:

def compute\_cost(Z3, Y, labels):

# get logits (predictions) and labels

logits = tf.transpose(Z3)

labels = tf.transpose(Y)

# compute cost

cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(

logits=logits,

labels=labels

))

return cost

**Appendix D**

Create tensorflow neural network:

def model(train, test, labels, learning\_rate=0.0001, num\_epochs=16, minibatch\_size=32, print\_cost=True, hidden\_1=128, hidden\_2=128, n\_input=784, n\_classes=10):

ops.reset\_default\_graph()

tf.set\_random\_seed(42)

seed = 42

(n\_x, m) = train.images.T.shape

n\_y = train.labels.T.shape[0]

costs = []

X, Y = create\_placeholders(n\_x, n\_y)

parameters = initialize\_parameters(hidden\_1, hidden\_2, n\_input, n\_classes)

Z3 = forward\_propagation(X, parameters)

cost = compute\_cost(Z3, Y, labels)

optimizer = tf.train.AdamOptimizer(learning\_rate).minimize(cost)

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

# initialization

sess.run(init)

for epoch in range(num\_epochs):

epoch\_cost = 0.

num\_minibatches = int(m / minibatch\_size)

seed = seed + 1

for i in range(num\_minibatches):

minibatch\_X, minibatch\_Y = train.next\_batch(minibatch\_size)

# optimizer and cost function

\_, minibatch\_cost = sess.run(

[optimizer, cost],

feed\_dict={X: minibatch\_X.T, Y: minibatch\_Y.T}

)

epoch\_cost += minibatch\_cost / num\_minibatches

if print\_cost == True:

print('Cost after epoch {epoch\_num}: {cost}'.format(

epoch\_num=epoch,

cost=epoch\_cost

))

costs.append(epoch\_cost)

# plot costs

plt.figure(figsize=(16,5))

plt.plot(np.squeeze(costs), color='#2A688B')

plt.xlim(0, num\_epochs-1)

plt.ylabel('cost')

plt.xlabel('iterations')

plt.title('learning rate = {rate}'.format(rate=learning\_rate))

plt.show()

parameters = sess.run(parameters)

print('Parameters have been trained!')

correct\_prediction = tf.equal(tf.argmax(Z3), tf.argmax(Y))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, 'float'))

print ('Train Accuracy:', accuracy.eval({

X: train.images.T,

Y: train.labels.T

}))

print ('Test Accuracy:', accuracy.eval({

X: test.images.T,

Y: test.labels.T

}))

**Appendix E**

Generate svm model:

def fit(train):

X = train[0]

y = train[1]

clf = svm.SVC(gamma=0.1, kernel='poly')

clf.fit(X, y)

return clf

Predict using svm model

def predict(clf, test):

return clf.predict(test[0])

Determine svm accuracy

def accuracy(clf, test, y\_pred):

accuracy\_classifier = clf.score(test[0], test[1])

accuracy\_prediction = accuracy\_score(test[1], y\_pred)

confusion = confusion\_matrix(test[1], y\_pred)

print('\nSVM Trained Classifier Accuracy: {}'.format(accuracy\_classifier))

print('Predicted Values: {}'.format(y\_pred))

print('Acc Classifier on Validation Images: {}'.format(accuracy\_prediction))

print('Confusion Matrix: {}\n'.format(confusion))

plt.matshow(confusion)

plt.title('Confusion Matrix for Test Data')

plt.colorbar()

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

**Appendix F**

Visualize svm results

def viz\_svm(test, labels):

arr = labels.values()

a = np.random.randint(1, 40, 15)

for i in a:

two\_d = (np.reshape(test[0][i], (28, 28)) \* 255).astype(np.uint8)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(two\_d, interpolation='nearest')

plt.xlabel('Actual ({actual}), Predicted ({predicted})'.format(

actual=test[1][i],

predicted=predicted\_labels[i]

))

plt.show()

1. https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion [↑](#footnote-ref-1)
2. https://github.com/jeff1evesque/ist-718-lab/tree/master/data [↑](#footnote-ref-2)
3. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/learn/python/learn/datasets/mnist.py#L232-L294 [↑](#footnote-ref-3)
4. https://github.com/zalandoresearch/fashion-mnist/blob/985a10ca70f411cd0caf41613718b812ad064736/utils/mnist\_reader.py [↑](#footnote-ref-4)
5. https://github.com/zalandoresearch/fashion-mnist#get-the-data [↑](#footnote-ref-5)
6. https://github.com/jeff1evesque/ist-718-lab/blob/master/lab3/svm.py [↑](#footnote-ref-6)
7. https://www.tensorflow.org/api\_docs/python/tf/train/AdamOptimizer [↑](#footnote-ref-7)
8. https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a [↑](#footnote-ref-8)