Deep Learning Using TensorFlow

Dr. Ash Pahwa

Lesson 7:

Convolution Neural Network (CNN)

Lesson 7.2: MNIST Image Classification Application Using Linear Regression

Outline

- Step#1
  1. Load Libraries
  2. Download and Read MNIST data
  3. One-hot Encoding
  4. Function – Plotting images
  5. Plot a Few images

- Step#2
  1. Placeholder variables
  2. Variables
  3. Model
  4. Cost Function
  5. Optimization Function
  6. Performance Measures

- Step#3
  1. Create TensorFlow Session
  2. Initialize Variables
  3. Function: Optimization Iteration
  4. Function: Display Performance
  5. Function: Plot Model Weights

- Step#4
  1. Performance iteration count = 0
  2. Performance iteration count = 1
  3. Performance iteration count = 10
  4. Performance iteration count = 1000
  5. Print Confusion Matrix

Step#1

1. Load Libraries
2. Download and Read MNIST data
3. One-hot Encoding
4. Function – Plotting images
5. Plot a Few images

1.1 Load Libraries

```
###############################################
# MNIST Image Classification Using Linear Regression
#
################################################
# 1.1 Load the libraries
#

import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from sklearn.metrics import confusion_matrix

print(tf.__version__)
1.9.0
```

1.2 Download and Read MNIST data 12 MB of data

```
##################################################
# 1.2 Download and read MNIST data
#
from tensorflow.examples.tutorials.mnist import input_data
data = input_data.read_data_sets("MNIST_data/", one_hot = True)

print("The Training Set is:")
The Training Set is:
print(data.train.images.shape)
(55000, 784)
print(data.train.labels.shape)
(55000, 10)
print("The Test Set is:")
The Test Set is:
print(data.test.images.shape)
(10000, 784)
print(data.test.labels.shape)
(10000, 10)
print("The Validation Set is:")
The Validation Set is:
print(data.validation.images.shape)
(5000, 784)
print(data.validation.labels.shape)
(5000, 10)
```

		Count	Image Shape	Labels
1	Training	55,000	784 (28x28 pixels image)	Count = 10: 0-9
2	Test	10,000	784 (28x28 pixels image)	Count = 10: 0-9
3	Validation	5,000	784 (28x28 pixels image)	Count = 10: 0-9
	Total	70,000		

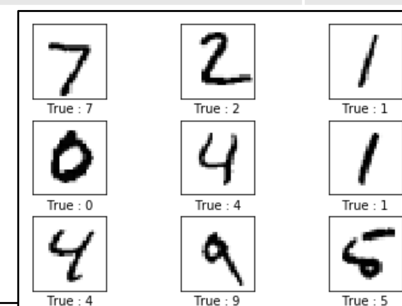
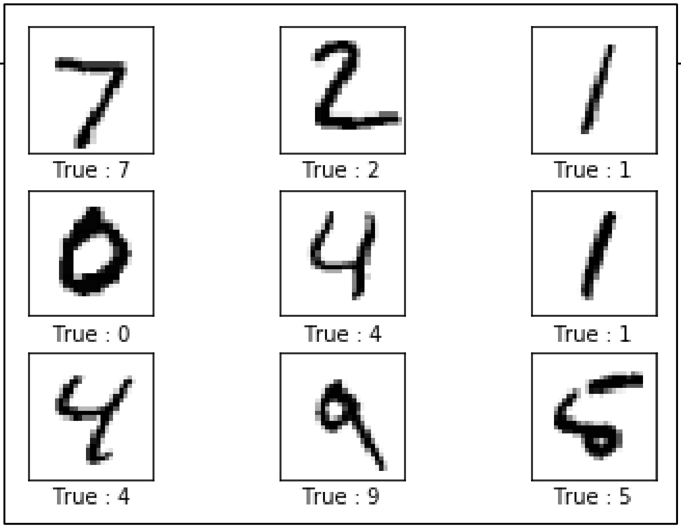


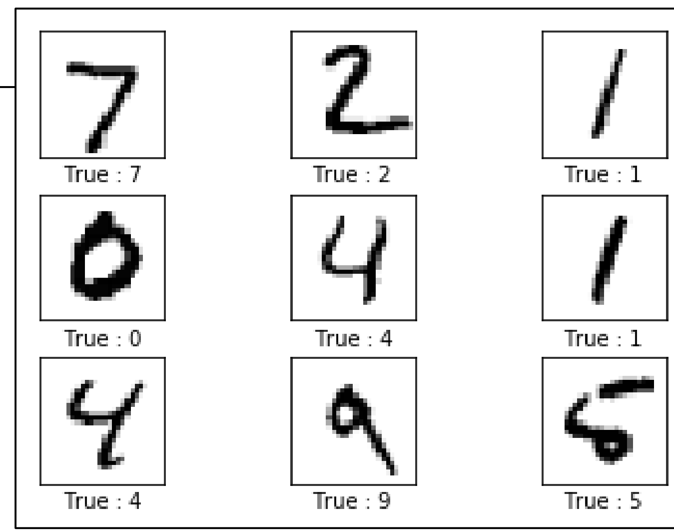
Image Dimensions

```
#########################################################
# the images are stored in one-dimensional arrays of this length.
#

img_size_flat = data.train.images[0].shape[0]
img_size_flat
Out[34]: 784

# Tuple with height and width of images used to reshape arrays.
img_shape = (28,28)

# Number of classes, one class for each of 10 digits.
num_classes = 10
```



1.3 One-hot Encoding

```
#####################################################
# 1.3 One hot encoding
#
data.test.labels[0:5,:]
Out[42]:
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]])

data.test.cls = np.array([label.argmax() for label in data.test.labels])

data.test.cls[0:5]
Out[44]: array([7, 2, 1, 0, 4], dtype=int64)
```

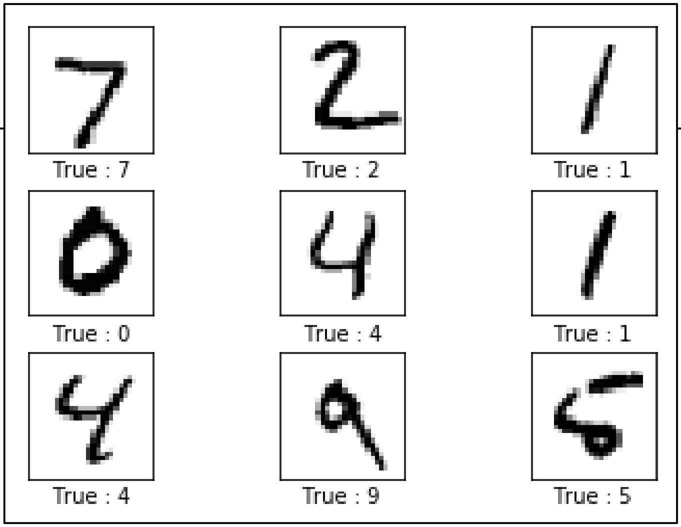


1.4 Function – Plotting Images

```
def plot_images(images, cls_true , cls_pred=None):
  assert len(images) == len(cls_true) == 9

  # Create figure with 3x3 subplots.
  fig, axes = plt.subplots(3,3)
  fig.subplots_adjust(hspace=0.3, wspace=0.3)

  for i, ax in enumerate(axes.flat):
    # Plot image.
    ax.imshow(images[i].reshape(img_shape), cmap='binary')

    # Show true and predicted classes
    if cls_pred is None:
      xlabel = 'True : {0}'.format(cls_true[i])
    else:
      xlabel = 'True : {0}, Pred : {1}'.format(cls_true[i], cls_pred[i])

    ax.set_xlabel(xlabel)

    # Remove ticks from the plot
    ax.set_xticks([])
    ax.set_yticks([])

  # Ensure the plot is shown correctly with multiple plots
  # in a single Notebook cell.
  plt.show()
```

1.5 Plot a Few Images

```
############################################
# 1.5 Plot a few images
# Get the first images from the Test-set.
#
images = data.test.images[0:9]

# Get the true classes for those images.
cls_true = [np.argmax(oh) for oh in data.test.labels[0:9] ]

# Plot the images and labels using our helper-function above.
plot_images(images, cls_true)
```



Step#2

1. Placeholder variables
2. Variables
3. Model
4. Cost Function
5. Optimization Function
6. Performance Measures

2.1 Placeholder Variables

Where the Input Data will be Assigned During Run Time

```
##############################################
# 2.1 Placeholder variables
#

x = tf.placeholder( tf.float32, [None, img_size_flat])

y_true = tf.placeholder( tf.float32, [None, num_classes])

y_true_cls = tf.placeholder( tf.int64, [None])
```

x = [?, 784]	Contains a batch of images
y_true = [?, 10]	Contains true one-hot encoded data
y_true_cls = [?]	Contains the number

2.2 Variables Weights and Bias

```
############################################
# 2.2 Variables
#

weights = tf.Variable(tf.zeros([img_size_flat, num_classes]))

bias = tf.Variable(tf.zeros([num_classes]))
```

weights = [784, 10]
bias = [10]

Neural Network:

- Input Layer has 784 neurons
- Output Layer has 10 neurons

Functions: 'softmax' and 'argmax'

$$Softmax = \frac{e^{y}}{\sum_1^n e^{y_i}}$$

C	D	E	F
		e^{y_i}	$\frac{e^{y_i}}{\sum_1^n e^{y_i}}$
	Logits Scores	EXP(n)	Softmax
	2	7.39	0.7
	1	2.72	0.2
	0.1	1.11	0.1
SUM	3.1	11.21	

```
import numpy as np
a = np.array([(8,2,7),(3,4,5)])

print(a)
[[8 2 7]
 [3 4 5]]

#########################################
print(a.sum(axis=0))
[11  6 12]

print(a.sum(axis=1))
[17 12]

#########################################

print(a.argmax(axis=0))
[0 1 0]

print(a.argmax(axis=1))
[0 2]
```

2.3 Model

```
################################################
# 2.3 Model
#

logits = tf.matmul(x, weights) + bias

y_pred = tf.nn.softmax(logits)

y_pred_cls = tf.argmax(y_pred, axis=1)
```

x = [?, 784]

weights = [784, 10]
bias = [10]

logits = [?, 784]*[784, 10] + [10]= [?, 10]

2.4 Cost Function

```
# 2.4 Cost Function
#

cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2( logits= logits, labels = y_true)

cost = tf.reduce_mean(cross_entropy)
```

2.5 Optimization Function

```
##############################################
# 2.5 Optimization Function
#

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5).minimize(cost)
```

2.6 Performance Measures

```
# 2.6 Performance measures
#

correct_prediction = tf.equal( y_pred_cls , y_true_cls)

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Step#3

1. Create TensorFlow Session
2. Initialize Variables
3. Function: Optimization Iteration
4. Function: Display Performance
5. Function: Plot Model Weights

3.1 Create TensorFlow Session

```
##############################################
# 3.1 Create TensorFlow Session
#

session = tf.Session()
```

3.2 Initialize Variables

```
#############################################
# 3.2 Initialize Variables
#

session.run(tf.global_variables_initializer())
```

3.3 Function: Optimization Iteration

```
###########################################
# 3.3 Function to perform optimization iteration
#

batch_size = 100

def optimize(num_iterations):
  for i in range(num_iterations):
    # Get a batch of training examples.
    # x_batch now holds a batch of images and
    # y_true_batch are the true labels for those images.
    x_batch, y_true_batch = data.train.next_batch(batch_size= batch_size)

    # Put the batch into a dict with the proper names
    # for placeholder variables in the TensorFlow graph.
    # Note that the placeholder for y_true_cls is not set
    # because it is not used during training.
    feed_dict_train = {x : x_batch,
                       y_true : y_true_batch}

    # Run the optimizer using this batch of training data.
    # TensorFlow assigns the variables in feed_dict train
    # to the placeholder variables and then runs the optimizer.
    session.run(optimizer, feed_dict = feed_dict_train)
```

3.4-1 Display Performance

```
###################################################
# 3.4 Optimization Iteration
#

feed_dict_test = {
                   x : data.test.images,
                   y_true : data.test.labels,
                   y_true_cls : [np.argmax(label) for label in data.test.labels]
                  }



def print_accuracy():
  # Use TensorFlow to compute the accuracy.
  acc = session.run(accuracy , feed_dict= feed_dict_test)

  # Print the accuracy.
  print('Accuracy on Test-st : {0:.1%}'.format(acc))
```

3.4-2 Display Performance

```
def print_confusion_matrix():
  # Get the true classifications for the Test-set.
  cls_true = [np.argmax(label) for label in data.test.labels]

  # Get the predicted classifications for the Test-set.
  cls_pred = session.run(y_pred_cls, feed_dict = feed_dict_test)

  # Get the confusion matrix using sklearn.
  cm = confusion_matrix(y_true = cls_true,
                        y_pred = cls_pred)

  # Print the confusion matrix as text.
  print(cm)

  # Plot the confusion matrix as an image.
  plt.imshow(cm, interpolation ='nearest', cmap = plt.cm.Blues)

  # Make various adjustments to the plot.
  plt.tight_layout()
  plt.colorbar()
  tick_marks = np.arange(num_classes)
  plt.xticks(tick_marks, range(num_classes))
  plt.yticks(tick_marks, range(num_classes))
  plt.xlabel('Predicted')
  plt.ylabel('True')

  # Ensure the plot is shown correctly with multiple plots
  # in a single Notebook cell.
  plt.show()
```

3.4-3 Display Performance

```
def plot_example_errors():
  # Use TensorFlow to get a list of boolean values
  # whether each test-image has been correctly classified,
  # and a list for the predicted class of each image.
  correct, cls_pred = session.run([correct_prediction,
                                  y_pred_cls],
                                  feed_dict = feed_dict_test)

  # Negate the boolean array.
  incorrect = (correct == False)

  # Get the images from the Test-set that have been
  # incorrectly classified
  images = data.test.images[incorrect]

  # Get the predicted classes for those images
  cls_pred = cls_pred[incorrect]

  # Get the true classes for those images.
  cls_true = [np.argmax(label) for label in data.test.labels[incorrect]]

  # Plot the first 9 images.
  plot_images(images = images[0:9],
              cls_true = cls_true[0:9],
              cls_pred = cls_pred[0:9])
```

3.5 Function: Plot Model Weights

```
def plot_weights():
  # Get the values for the weights from the TensorFlow variable
  w = session.run(weights)

  # Get the lowest and hightest values for the weights.
  # This is used to correct the colour intensity across
  # the images so they can be compared with each other.
  w_min = np.min(w)
  w_max = np.max(w)

  # Create figure with 3x4 sub-plots,
  # where the last 2 sub-plots are unused.
  fig, axes = plt.subplots(3,4)
  fig.subplots_adjust(hspace=0.3, wspace=0.3)

  for i, ax in enumerate(axes.flat):
    # Only use the weights for the first 10 sub-plots
    if i < 10:
      # Get the weights for the i'th digit and reshape it.
      # Note that w.shape == (img_size_flat, 10)
      image = w[:, i].reshape(img_shape)

      # Set the label for the sub-plot.
      ax.set_xlabel('Weights : {0}'.format(i))

      # Plot the image.
      ax.imshow(image, vmin=w_min, vmax= w_max,
                cmap = 'seismic')
    # Remove ticks from each sub-plot
    ax.set_xticks([])
    ax.set_yticks([])

  # Ensure the plot is shown correctly with multiple plots
  # in a single Notebook cell.
  plt.show()
```

Step#4

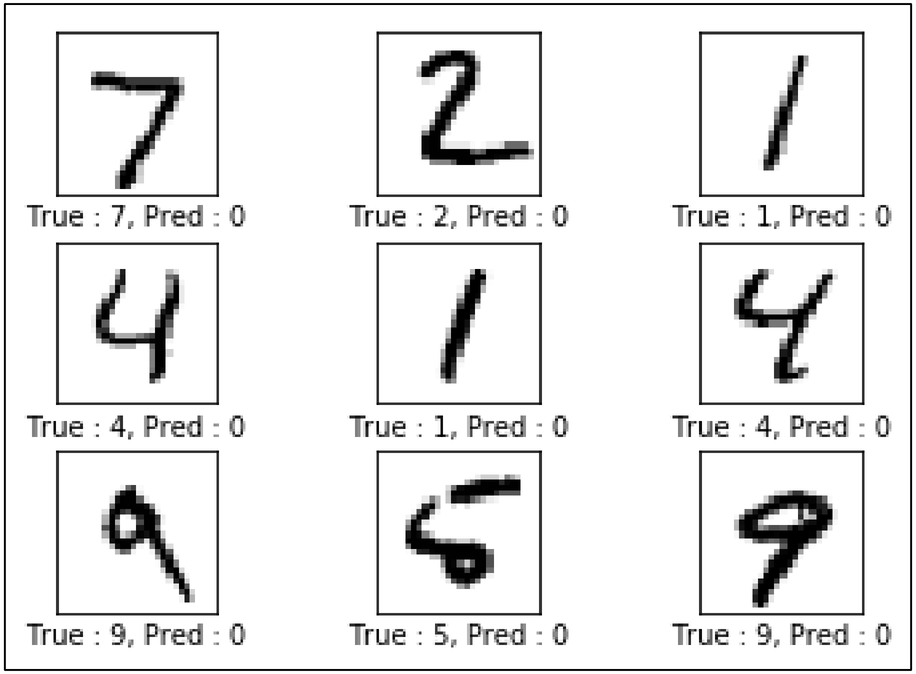
1. Performance iteration count = 0
2. Performance iteration count = 1
3. Performance iteration count = 10
4. Performance iteration count = 1000
5. Print Confusion Matrix

4.1 Performance: Iteration count = 0

```
# 4.1 Performance before any optimization
#

print_accuracy()
Accuracy on Test-st : 9.8%
```

4.1 Performance: Iteration count = 0

```
plot_example_errors()
```

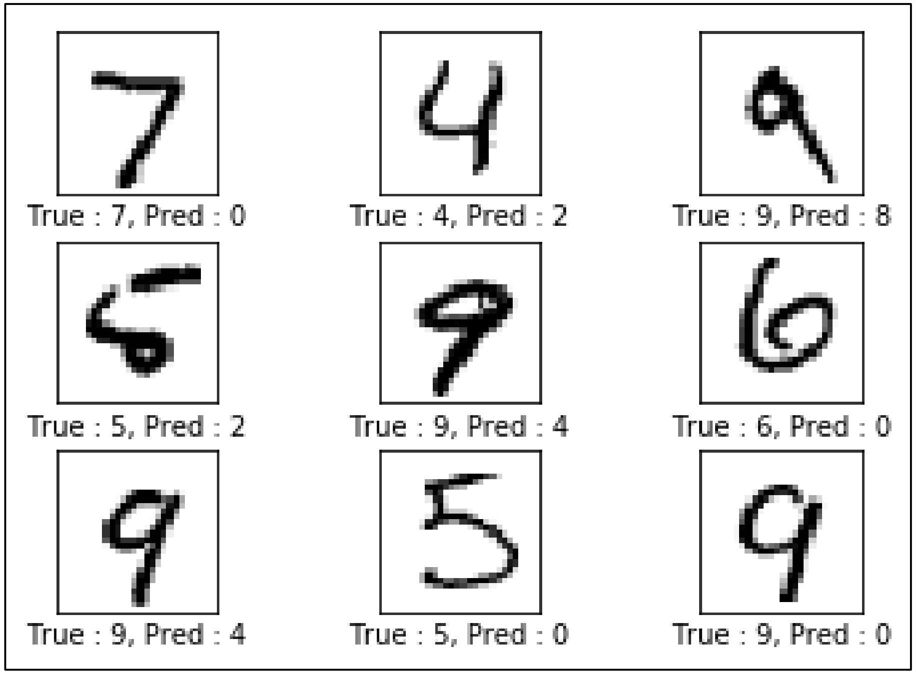


4.2 Performance: Iteration count = 1

```
#############################################
# 4.2 Performance Iteration#1
#
# Number of iteration means how many of batchs are iterated
#
optimize(num_iterations= 1)

print_accuracy()
Accuracy on Test-st : 50.9%
```

4.2 Performance: Iteration count = 1

```
plot_example_errors()
```

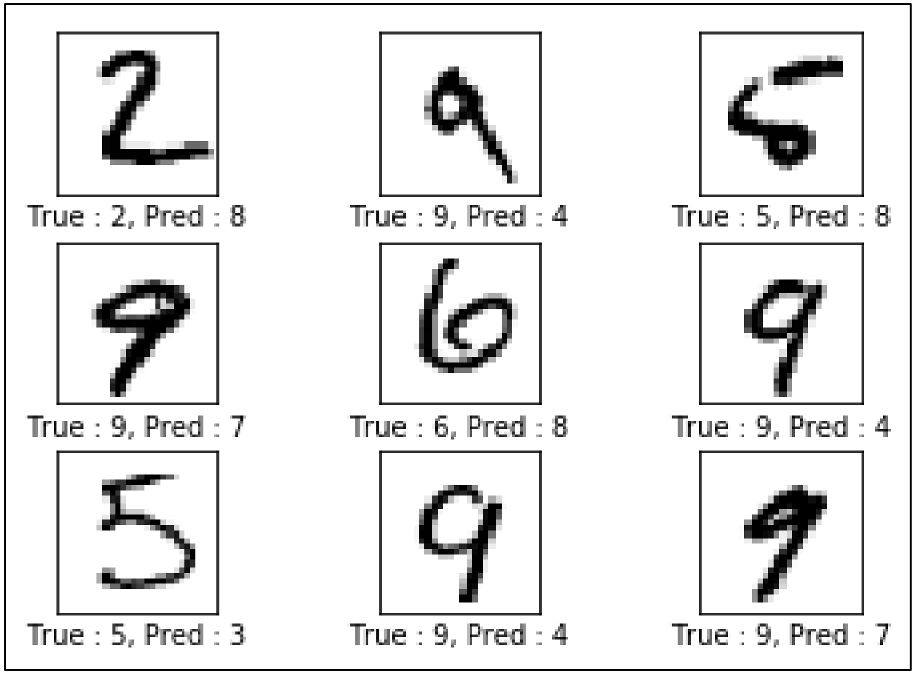


4.3 Performance: Iteration count = 10

```
##############################################
# 4.3 Performance Iteration #10
# We've already performed 1 iteration.
#
optimize( num_iterations=9 )

print_accuracy()
Accuracy on Test-st : 70.7%
```

4.3 Performance: Iteration count = 10

```
plot_example_errors()
```

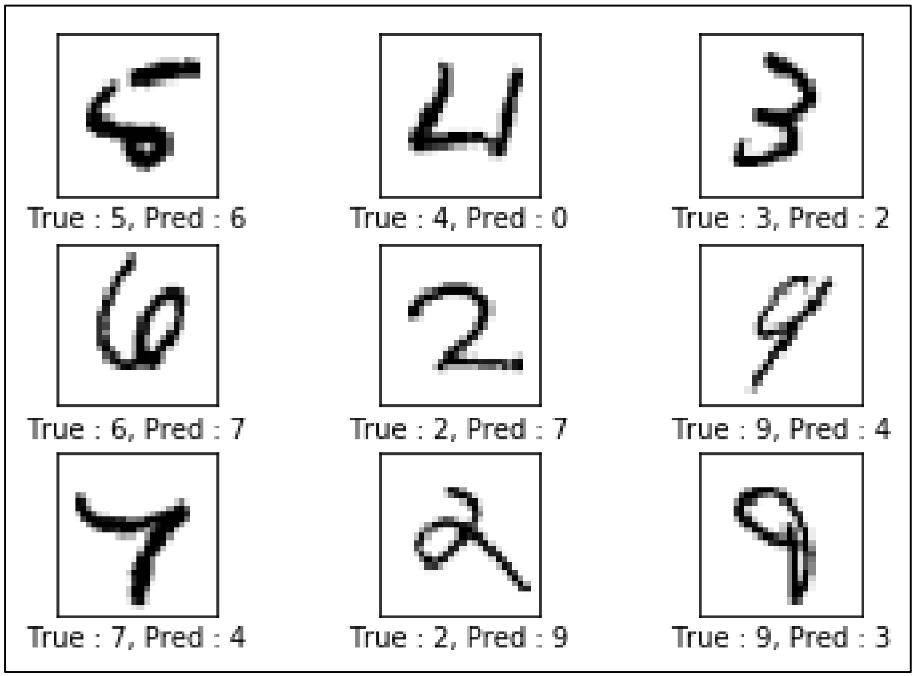


4.4 Performance: Iteration count = 1000

```
##################################################
# 4.4 Performance Iteration #1000
# We've already performed 10 iterations.
#
optimize(num_iterations=990)

print_accuracy()
Accuracy on Test-st : 91.8%
```

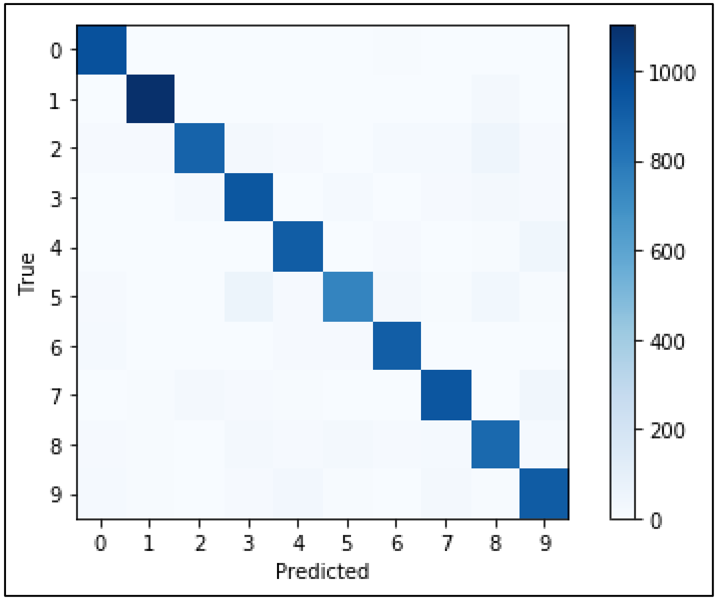
4.4 Performance: Iteration count = 1000

```
plot_example_errors()
```



4.5 Print Confusion Matrix

```
###################################################

# 4.5 Print confusion Matrix

#

print_confusion_matrix()
[[ 964    0    2    3    0    3    5    1    2    0]
 [    0 1104    2    2    1    2    4    2   18    0]
 [   11   10  887   24   12    1   13   14   50   10]
 [    3    0   13  936    1   17    1   10   20    9]
 [    4    1    2    2  910    0   12    2    8   41]
 [   10    3    3   57    9  746   19    7   31    7]
 [   14    3    3    2   10   12  909    3    2    0]
 [    3    8   18    9    5    1    0  944    3   37]
 [    9    5    3   25    9   25   10   14  859   15]
 [   13    6    1   11   27    6    0   22    7  916]]
```



Summary

- Step#1
1. Load Libraries
2. Download and Read MNIST data
3. One-hot Encoding
4. Function – Plotting images
5. Plot a Few images

- Step#2
1. Placeholder variables
2. Variables
3. Model
4. Cost Function
5. Optimization Function
6. Performance Measures

- Step#3
1. Create TensorFlow Session
2. Initialize Variables
3. Function: Optimization Iteration
4. Function: Display Performance
5. Function: Plot Model Weights

- Step#4
1. Performance iteration count = 0
2. Performance iteration count = 1
3. Performance iteration count = 10
4. Performance iteration count = 1000
5. Print Confusion Matrix