MNIST

April 18, 2024

[42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import keras
from sklearn.linear_model import LogisticRegression
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
```

[43]:
```
# MNIST Dataset parameters

num_clasess = 10 #total clases(0-9)
num_features = 784 # data features (img shape 28*28)

# Training parameters
learning_rate = 0.1
training_steps = 2000
batch_size = 256
display_step = 100
epochs = 20

# Network Parameters
n_hidden_1 = 128 # 1st layers number of neurons
n_hidden_2 = 256 # 2nd layer number of neurons
```

[44]:
```
# Prepare MNIST Data

(x_train, y_train),(x_test, y_test) = mnist.load_data()
print('Train - ',x_train.shape)
print('Test - ', x_test.shape)


# convert to float32
x_train, x_test = np.array(x_train,np.float32),np.array(x_test,np.float32)
```

```
# Flatten images in to 1D vector of 784 features (28 * 28)
x_train,x_test = x_train.reshape([-1,num_features]),x_test.
 ↪reshape([-1,num_features]) # -1 means we're reducing the 2D to 1D

# Normalize images values from [0-255] to [0,1]
x_train, x_test = x_train/255., x_test/255.

print('Flatten Train -', x_train.reshape)
print ('Flatten Test -', x_test.reshape)
```

```
Train -  (60000, 28, 28)
Test -  (10000, 28, 28)
Flatten Train - <built-in method reshape of numpy.ndarray object at
0x7f4f34500030>
Flatten Test - <built-in method reshape of numpy.ndarray object at
0x7f4f345004b0>
```

[45]:
```
plt.figure(figsize=(10,1))
for i in range(10):
    plt.subplot(1,10,i+1)
    plt.imshow(x_train[i].reshape(28,28), cmap="gray")
    plt.axis('off')
plt.show()
print('label for each of the above images %s' % (y_train[0:10]))
```

```
label for each of the above images [5 0 4 1 9 2 1 3 1 4]
```

[46]:
```
y_train[0]
```

[46]: 5

[47]:
```
y_test[1]
```

[47]: 2

[48]:
```
# Do one hot encoding
y_train_ohe = keras.utils.to_categorical(y_train,num_clasess)
y_test_ohe = keras.utils.to_categorical(y_test,num_clasess)
```

[49]:
```
y_train_ohe[0]
```

[49]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

0.1 Build ANN Model

[50]:
```
## Build NN Model

model = Sequential()
model.add(Dense(512,activation='relu',input_shape=(784,)))

# model.add droupout add 0.2
model.add(Dense(512,activation='relu'))

# model.add droupout add 0.2
model.add(Dense(num_clasess,activation='softmax'))

model.summary()

# Note:  Param # are total number of weights and biases
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 512)               401920

 dense_10 (Dense)            (None, 512)               262656

 dense_11 (Dense)            (None, 10)                5130

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[62]:
```
# Compile

model.compile(loss='categorical_crossentropy',
              optimizer = RMSprop(),
              metrics=['accuracy'])
```

[63]:
```
history = model.fit(x_train, y_train_ohe,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test,y_test_ohe))

# Model Evaluation
```

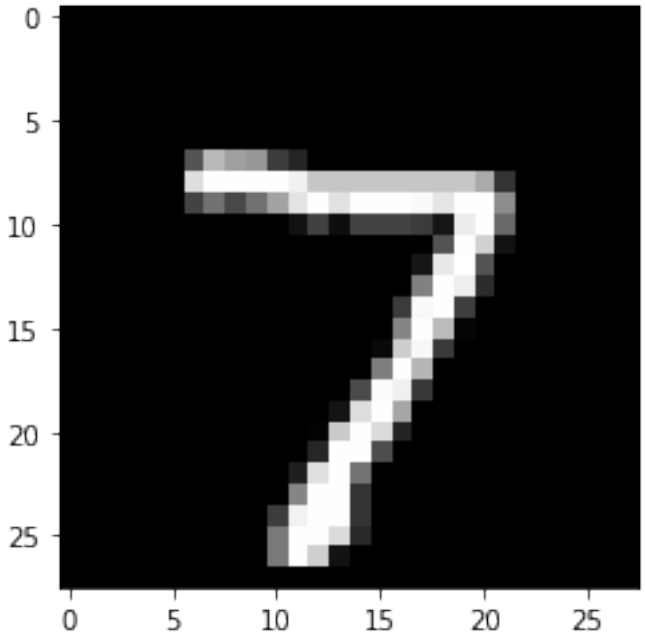
```
score = model.evaluate(x_test,y_test_ohe,verbose=0)
print('Test loss - ',score[0])
print('Test Accuracy - ', score[1])
```

```
Epoch 1/20
235/235 [==============================] - 3s 11ms/step - loss: 0.2900 -
accuracy: 0.9114 - val_loss: 0.1359 - val_accuracy: 0.9574
Epoch 2/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0993 -
accuracy: 0.9695 - val_loss: 0.1116 - val_accuracy: 0.9638
Epoch 3/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0629 -
accuracy: 0.9804 - val_loss: 0.1337 - val_accuracy: 0.9570
Epoch 4/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0435 -
accuracy: 0.9866 - val_loss: 0.0747 - val_accuracy: 0.9770
Epoch 5/20
235/235 [==============================] - 2s 10ms/step - loss: 0.0321 -
accuracy: 0.9898 - val_loss: 0.0729 - val_accuracy: 0.9770
Epoch 6/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0231 -
accuracy: 0.9930 - val_loss: 0.0748 - val_accuracy: 0.9768
Epoch 7/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0172 -
accuracy: 0.9947 - val_loss: 0.0605 - val_accuracy: 0.9823
Epoch 8/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0123 -
accuracy: 0.9965 - val_loss: 0.0995 - val_accuracy: 0.9714
Epoch 9/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0100 -
accuracy: 0.9969 - val_loss: 0.0702 - val_accuracy: 0.9806
Epoch 10/20
235/235 [==============================] - 2s 10ms/step - loss: 0.0073 -
accuracy: 0.9978 - val_loss: 0.0700 - val_accuracy: 0.9822
Epoch 11/20
235/235 [==============================] - 2s 10ms/step - loss: 0.0050 -
accuracy: 0.9988 - val_loss: 0.0706 - val_accuracy: 0.9832
Epoch 12/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0049 -
accuracy: 0.9986 - val_loss: 0.0689 - val_accuracy: 0.9842
Epoch 13/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0026 -
accuracy: 0.9993 - val_loss: 0.1092 - val_accuracy: 0.9776
Epoch 14/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0021 -
accuracy: 0.9994 - val_loss: 0.0708 - val_accuracy: 0.9844
Epoch 15/20
```

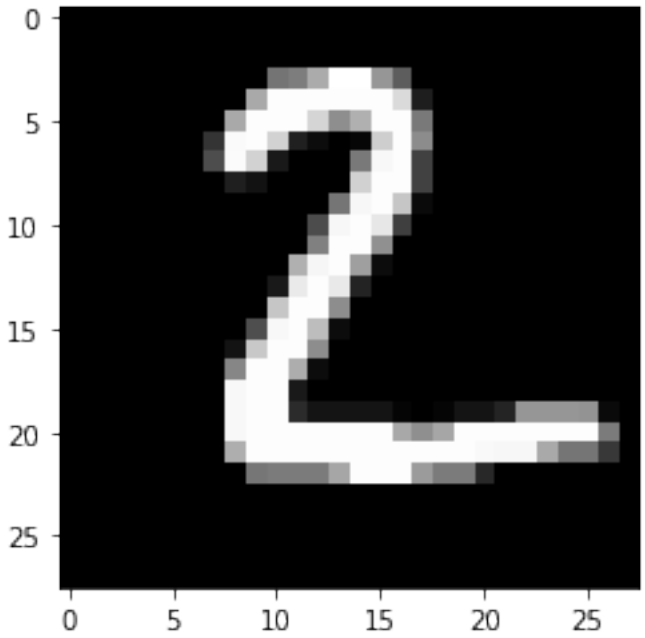
```
235/235 [==============================] - 2s 9ms/step - loss: 0.0013 -
accuracy: 0.9997 - val_loss: 0.0823 - val_accuracy: 0.9828
Epoch 16/20
235/235 [==============================] - 2s 9ms/step - loss: 0.0012 -
accuracy: 0.9995 - val_loss: 0.0716 - val_accuracy: 0.9844
Epoch 17/20
235/235 [==============================] - 2s 9ms/step - loss: 7.6316e-04 -
accuracy: 0.9998 - val_loss: 0.0728 - val_accuracy: 0.9855
Epoch 18/20
235/235 [==============================] - 2s 9ms/step - loss: 1.7632e-04 -
accuracy: 1.0000 - val_loss: 0.0728 - val_accuracy: 0.9857
Epoch 19/20
235/235 [==============================] - 2s 10ms/step - loss: 5.9863e-05 -
accuracy: 1.0000 - val_loss: 0.0726 - val_accuracy: 0.9858
Epoch 20/20
235/235 [==============================] - 2s 10ms/step - loss: 4.7001e-05 -
accuracy: 1.0000 - val_loss: 0.0733 - val_accuracy: 0.9860
Test loss -  0.07332673668861389
Test Accuracy -  0.9860000014305115
```

[67]:
```
# Predict 5 images from evaluation set
n_images = 5
test_images = x_test[:n_images]
predictions = model.predict(test_images)


# Display Image and Model prediction

for i in range(n_images):
    plt.imshow(np.reshape(test_images[i],[28,28]),cmap='gray')
    plt.show()
    print('model prediction: %i' % np.argmax(predictions[i]))
```

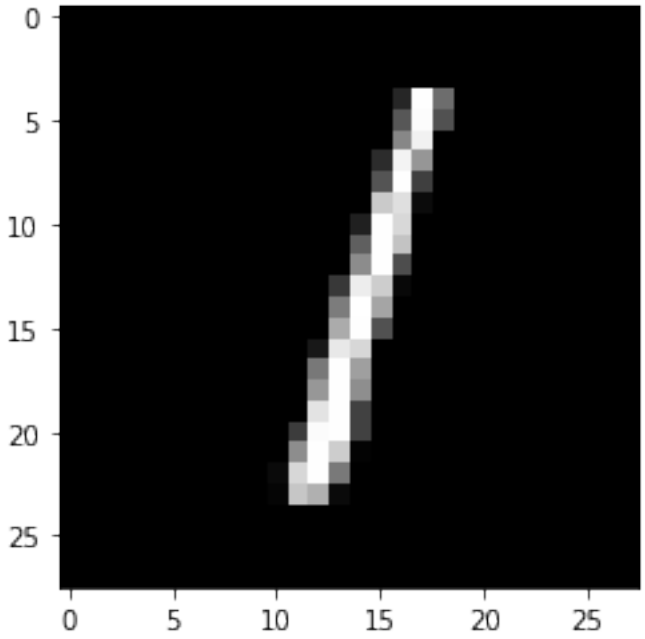
```
1/1 [==============================] - 0s 61ms/step
```



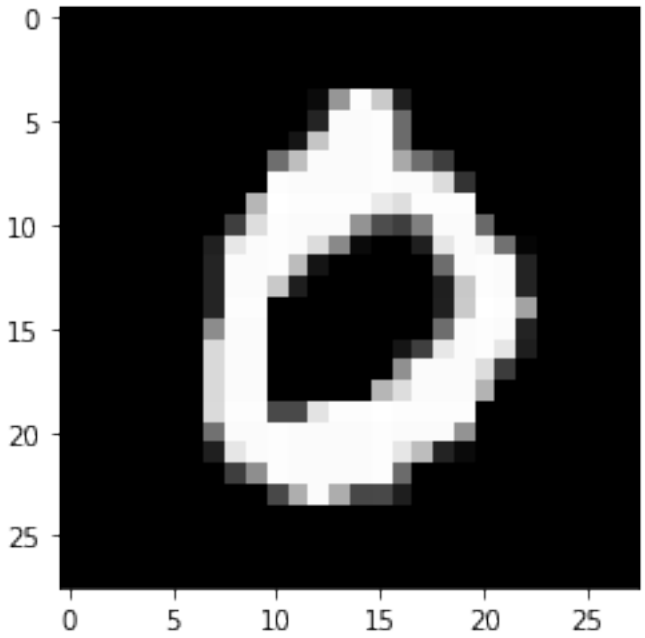

```
model prediction: 7
```



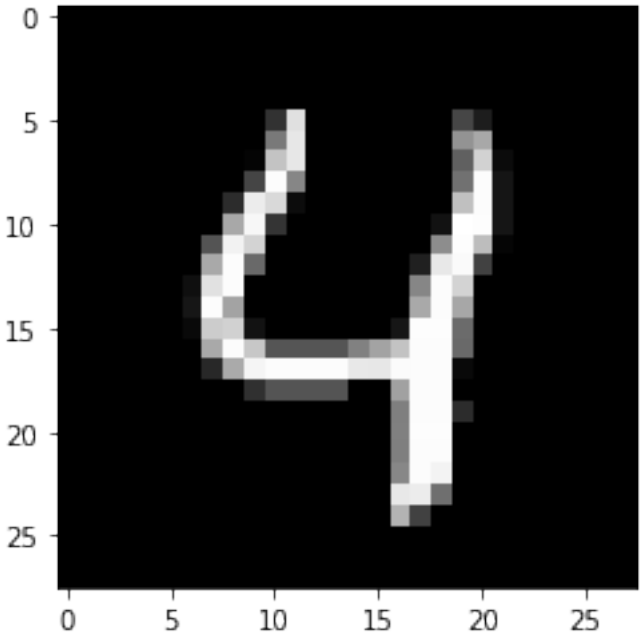

```
model prediction: 2
```




```
model prediction: 1
```

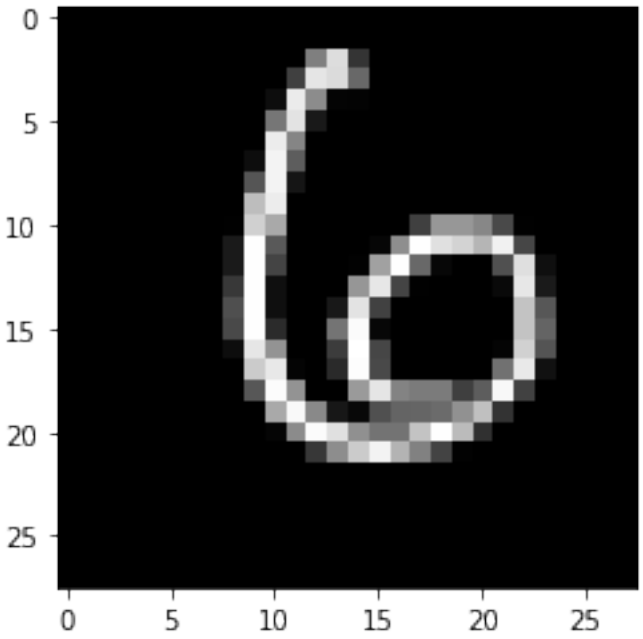



```
model prediction: 0
```




```
model prediction: 4
```

[103]:
```
plt.imshow(np.reshape(x_test[100],[28,28]),cmap='gray')
plt.show()
```



[ ]:

[ ]: