Digit Recognition ANN Solution

December 31, 2019

[1]:
```python
import numpy as np

# keras import for the dataset
from keras.datasets import mnist
```

```
Using TensorFlow backend.
```

[2]:
```python
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

[3]:
```python
# each training and test element is a 28 x 28 pixel grayvalue image
print(X_train[0].shape)
print(X_train[0])
```

```
(28, 28)
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

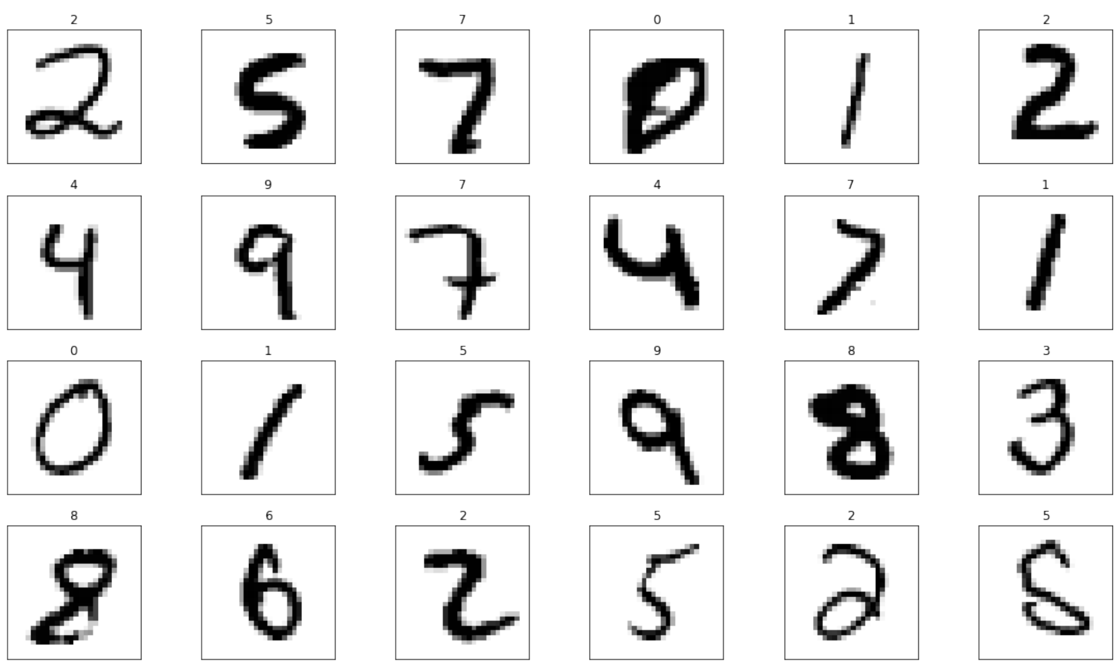
```
[  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

[4]:
```
# the corresponding label is the "real" digit
print(np.unique(y_train, return_counts=True))
print(y_train[0])
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([5923, 6742, 5958,
6131, 5842, 5421, 5918, 6265, 5851, 5949]))
5
```

[5]:
```
# imports for plotting
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt

%matplotlib inline
```

```python
import numpy as np
# choose 24 indices at random between 0 and len(X_train)-1
index = np.random.choice(np.arange(len(X_train)), 24, replace=False)
# subplots returns a tuple containing a Figure object and an array of the subplots' Axes objects
figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(16, 9))

# zip makes you can iterate over multiple iterables of data at the same time
for item in zip(axes.ravel(), X_train[index], y_train[index]):
    # ravel creates a onedimensional view of multidimensional array
    axes, image, target = item
    axes.imshow(image, cmap=plt.cm.gray_r)
    axes.set_xticks([])  # remove x-axis tick marks
    axes.set_yticks([])  # remove y-axis tick marks
    axes.set_title(target)

plt.tight_layout()
```



[6]:
```python
# let's print the shape before we reshape and normalize
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)
```

```
# building the input vector from the 28x28 pixels = linearize the image to get␣
↪a 784 (= 28x28) vector
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

# normalizing the data to help with the training
# normalized data leads to better models
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# print the final input shape ready for training
print("Train matrix shape", X_train.shape)
print("Test matrix shape", X_test.shape)
```

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
Train matrix shape (60000, 784)
Test matrix shape (10000, 784)
```

[7]:
```
# one-hot encoding using keras' numpy-related utilities
from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train)
print(y_train.shape)
print(y_train[0])  # one sample's categorical data
y_test = to_categorical(y_test)
print(y_test.shape)
```

```
(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
(10000, 10)
```

[8]:
```
# keras imports for building our neural network
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation

# building a linear stack of layers with the sequential model
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('sigmoid'))
model.add(Dropout(0.2))

model.add(Dense(512))
```

```
model.add(Activation('sigmoid'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))
```

[9]:
```
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],␣
↪optimizer='adam')
```

[10]:
```
# training the model and saving metrics in history
model.fit(X_train, y_train,epochs=20,verbose=2)
```

```
Epoch 1/20
 - 17s - loss: 0.3732 - accuracy: 0.8842
Epoch 2/20
 - 14s - loss: 0.1596 - accuracy: 0.9517
Epoch 3/20
 - 14s - loss: 0.1063 - accuracy: 0.9674
Epoch 4/20
 - 14s - loss: 0.0814 - accuracy: 0.9747
Epoch 5/20
 - 14s - loss: 0.0623 - accuracy: 0.9798
Epoch 6/20
 - 14s - loss: 0.0508 - accuracy: 0.9832
Epoch 7/20
 - 14s - loss: 0.0416 - accuracy: 0.9865
Epoch 8/20
 - 14s - loss: 0.0361 - accuracy: 0.9877
Epoch 9/20
 - 14s - loss: 0.0309 - accuracy: 0.9899
Epoch 10/20
 - 15s - loss: 0.0269 - accuracy: 0.9908
Epoch 11/20
 - 14s - loss: 0.0223 - accuracy: 0.9923
Epoch 12/20
 - 14s - loss: 0.0206 - accuracy: 0.9930
Epoch 13/20
 - 14s - loss: 0.0165 - accuracy: 0.9944
Epoch 14/20
 - 15s - loss: 0.0167 - accuracy: 0.9946
Epoch 15/20
 - 14s - loss: 0.0141 - accuracy: 0.9952
Epoch 16/20
 - 14s - loss: 0.0131 - accuracy: 0.9956
Epoch 17/20
 - 14s - loss: 0.0142 - accuracy: 0.9951
```

```
Epoch 18/20
 - 14s - loss: 0.0105 - accuracy: 0.9963
Epoch 19/20
 - 14s - loss: 0.0106 - accuracy: 0.9964
Epoch 20/20
 - 14s - loss: 0.0116 - accuracy: 0.9960
```

[10]: <keras.callbacks.callbacks.History at 0x7f4eb4ed9f10>

[11]:
```python
# saving the model
import os
save_dir = "./"
model_name = 'keras_mnist.h5'
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
```

[12]:
```python
mnist_model = load_model('keras_mnist.h5')
loss, accuracy = mnist_model.evaluate(X_test, y_test)

print("Test Loss", loss)
print("Test Accuracy", accuracy)
```

```
10000/10000 [==============================] - 1s 100us/step
Test Loss 0.07530979994986664
Test Accuracy 0.9828000068664551
```

[13]:
```python
# load the model and create predictions on the test set
model = load_model('keras_mnist.h5')

predictions = model.predict(X_test)
# The first digit should be a 7 (shown as 1. at index 7)
print(y_test[0])

# Check the probabilities returned by predict for first test sample
for index, probability in enumerate(predictions[0]):
    print(f'{index}: {probability:.10%}')

# Our model believes this digit is a 7 with nearly 100% certainty
# Not all predictions have this level of certainty
```

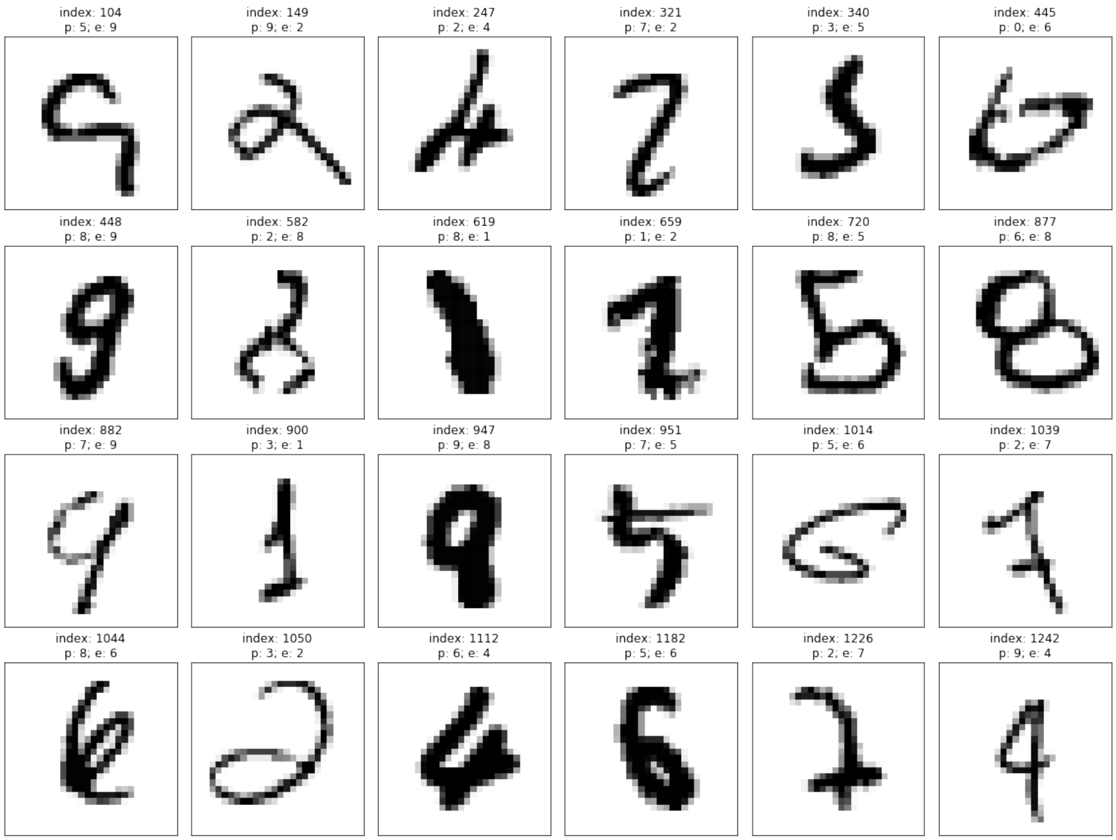
```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
0: 0.0000000000%
1: 0.0000001574%
2: 0.0000000001%
3: 0.0000000252%
4: 0.0000000000%
5: 0.0000000000%
6: 0.0000000000%
```

```
7: 100.0000000000%
8: 0.0000000001%
9: 0.0000002009%
```

- In the following snippet, p is the predicted value array, and e is the expected value array
- NumPy's argmax() function determines index of an array's highest valued element
- The function enumerate() receives and iterable and creates an iterator that, for each element, returns a tuple containing the element's index and value

[14]:
```
# Locating the Incorrect Predictions
images = X_test.reshape((10000, 28, 28))

incorrect_predictions = []

for i, (p, e) in enumerate(zip(predictions, y_test)):
    predicted, expected = np.argmax(p), np.argmax(e)

    if predicted != expected:  # prediction was incorrect
        incorrect_predictions.append((i, images[i], predicted, expected))
```

[15]:
```
%matplotlib inline

figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(16, 12))

for axes, item in zip(axes.ravel(), incorrect_predictions):
    index, image, predicted, expected = item
    axes.imshow(image, cmap=plt.cm.gray_r)
    axes.set_xticks([])  # remove x-axis tick marks
    axes.set_yticks([])  # remove y-axis tick marks
    axes.set_title(f'index: {index}\np: {predicted}; e: {expected}')
plt.tight_layout()
```



[16]:
```
predictions[1156]
```

[16]:
```
array([1.4936407e-07, 3.0611541e-06, 1.4423036e-07, 1.4282927e-03,
       4.6548826e-10, 1.3950827e-03, 2.1356957e-08, 9.5209759e-01,
       4.5051001e-02, 2.4725417e-05], dtype=float32)
```

[ ]:
```

```