Homework 5: MNIST Handwriting Recognition

Eunji Lindsey Lee

(*e-mail:* elee0025@uw.edu)

Abstract

This paper explores the concept of neural network and its application in recognizing handwritten digits from MNIST dataset. It builds a neural network classification model, optimized to make an accurate detector of digits.

Contents

1 Introduction and Overview

Imagine you are looking at the handwritten digits. How long will you take to actually recognize them? For most of us, recognizing them will only take about a second or two. Such a recognition process feels like effortless for most of us. However, image recognition is not as easy as it sounds when it comes to computer programming. It is actually extremely difficult to program to mimic what our brains seem to be doing so well unconsciously.

Our brains are biologically designed with billions of neurons, and they are vastly connected to one another through dendrites, which receive inputs and produce an appropriate outputs to another neuron [**2**]. Neurons in isolation are not very smart but these massive connections are the ones that make our brains intelligent. A concept of ”neural network” has been developed, inspired by our biological neural network systems. The neural network

was designed to learn from a large training data and through each layers, it learns more and more complex and abstract structures of the images or any other types of dataset.

Further development of neural network has led to the development of deep learning algorithm, which has gained popularity over a last few years, and artificial intelligence (AI) techniques (which comes directly from deep learning) has contributed significantly on the development of technology today, from image recognition to speech interfaces in Amazon's Alexa, Google's voice search, etc., and to language translation [**1**]. The concept of neural network, as well as deep learning already took lots of parts our day-to-day lives, perhaps without us even realizing it. We believe it is important to understand the concept.

This paper explores basic theoretical concepts behind neural network, especially the concept of perceptron neuron, and simple implementations of neural network using python together with python's well-developed machine learning library *scikit-learn*, which made the computing simple. We will not get into details about the concept of classification and supervised learning, as we had already covered the details about machine learning theoretical backgrounds in homework 3 paper.

2 Theoretical Background

2.1 Perceptron Neuron

A single perceptron neuron is designed to take several weighted inputs and produces a binary output depending on its activation function (transfer function). There are various standard activation functions, which maps inputs to the output, and different types of neurons are built depending on the activation function. Neurons with an activation function of logistic (soft step)[**5**], or often referred as a sigmoid function,

$$f(x) = \frac{1}{1+e^{-x}}$$

is called the sigmoid neuron, and is more common in building neuron models than a perceptron neuron [**4**], but we will not cover the concept of sigmoid neurons in this paper. However, to understand the concept of sigmoid neuron, understanding the basic concept of perceptron neuron is critical.

Recall that each neurons take multiple inputs of different weights. From each neuron, it calculates the weighted sum of its inputs, and projects the output signal as a binary value of 0 or 1 depending on the weighted sum and the parameter, threshold value. A weighted sum is the sum of each weights ω_i times inputs x_i, which is denoted as $\sum_j \omega_i * x_i$. If the weighted sum is less than or equals to the threshold value θ, then the output of 0 is produced and output of 1 is produced, otherwise. Therefore, we get different neural network model by varying each weights and the threshold value.

2.2 Neural Network

Artificial neural network architecture looks like figure 1 [**5**], where inputs from the input layer are processed through the artificial mesh to several hidden layers (in this case 2 hidden layers), then outputs from the output layer are produced.

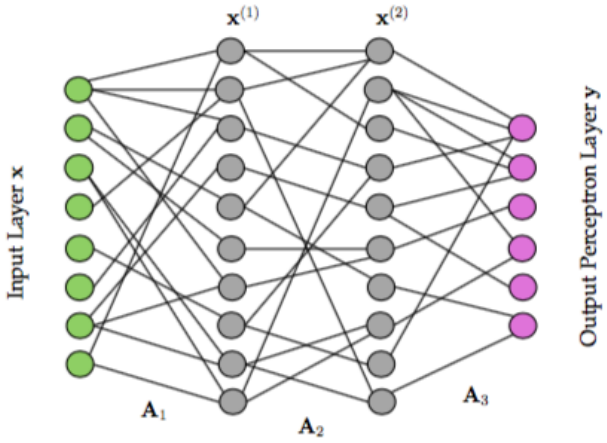


Figure 1. Illustration of neural network architecture

Hidden layers are simply the layers that are between input and output layers and it's where transformation of inputs occur. As more layers are added, the more abstract and more complex level of decisions occurs. Deep learning consist of multiple layers of multiple neurons.

The number of layers and the number of neurons to be used for building a model can be chosen with great flexibility, but it should be chosen carefully to build a good classification model.

3 Algorithm Implementation and Development

3.1 Processing Digits Image Data

For simplicity of data, we used the csv file provided by Kaggle. They cleaned and arranged the data in a way that's convenient for an analysis. It has a total of 42000 data points, which were put as rows in the csv file, and each row represents the label of digit and the image of digit (images are represented as pixels).

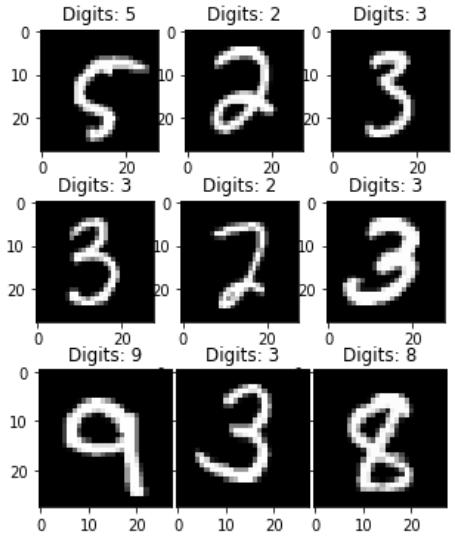


Figure 2. 9 Example images of digits with labels

3.2 Building Neural Network Model

In order to do the classification, we separated the data into data set(digit images) and label set. Then we cross-validated training data set and label set by a ratio of 8 to , so that now there are total of 33600 training sets and 8400 testing sets.

We used MLPClassifier from python library to create 1-layer and 2-layer perceptron based multilayer neural network models. As the following algorithm uses a stochastic process (i.e. random process), we set the random number seed to an arbitrary integer value. We played around with parameters and tuned to build a model with relatively high accuracy and f1 score.

4 Computational Results

4.1 1-Layer Model

We built a 1-layer neural network model, and it has a pretty high average f1 score (it measures the average of precision and recall) of 0.83 for all digits even though it was just one layered model. We only created 10 neurons for the layer just enough for each digits to be evaluated. Figure 3 shows the confusion matrix and it shows how well it predicted the each of the 10 digits (digits 0-9). Figure 4 shows the result of predictions for each 9 randomly chosen digits from the testing set.

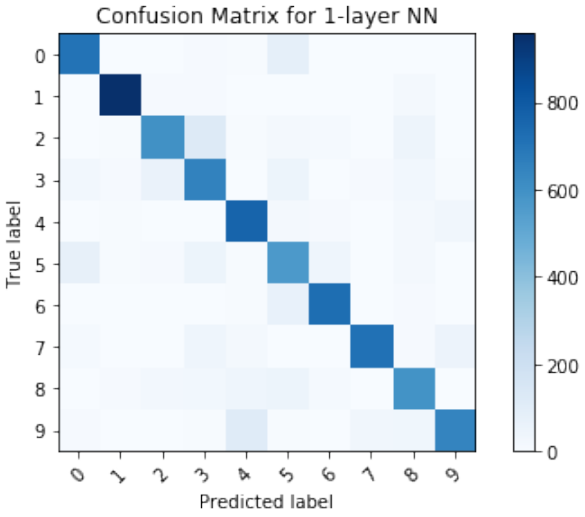


Figure 3. Confusion matrix for 1-layered model

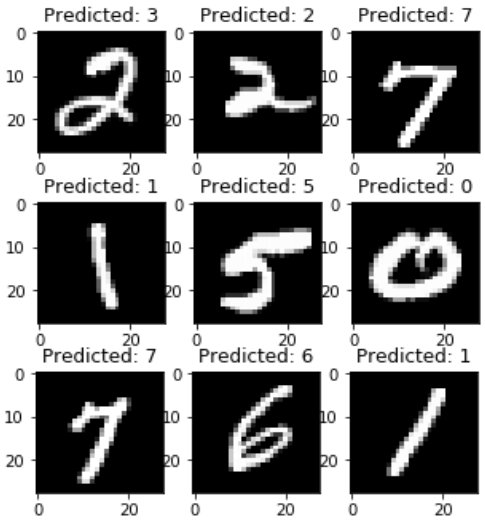


Figure 4. 9 Example images of predicted digits with labels

4.2 2-Layer Model

We built a 2-layer neural network model. It has an overall better performance compared to 1-layer model. We obtained an average of 0.92 f1-score for all digits. The figure 5 shows the performance of the classifier as a confusion matrix, and the figure 6 shows the images of predicted labels for a chosen digits from the testing set.

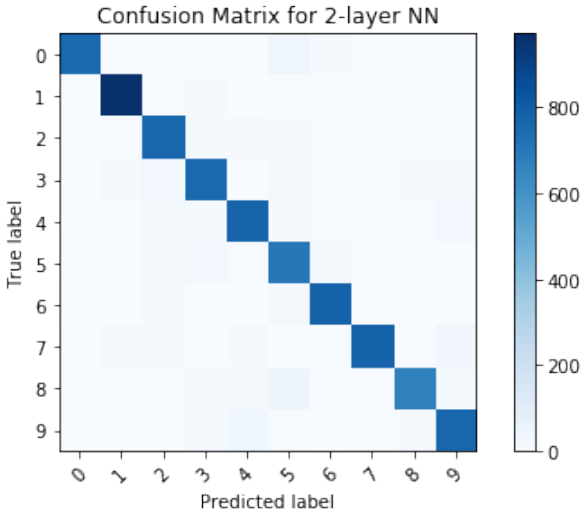


Figure 5. Confusion matrix for 2-layered model

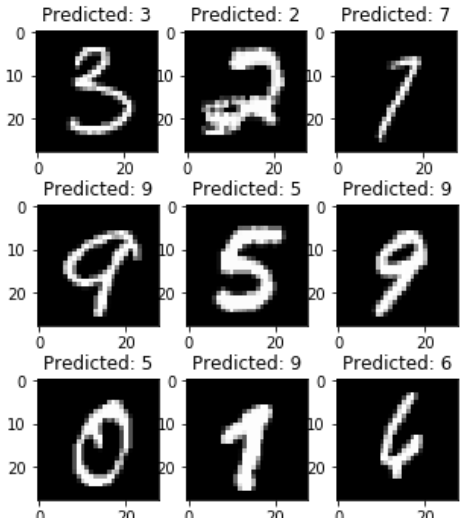


Figure 6. 9 Example images of predicted digits with labels

5 Summary and Conclusions

This paper has explored the basics of neural network, especially multilayer neural network using perceptrons. Although we had built a very basic 1 and 2 layered neural network models, the results we had obtained have a fairly high accuracy and f1 scores surprisingly. When building a classification model, we manually adjusted each parameters of the "Multilayer Perceptron Classifier" (from python library *scikit-learn*). Since we did not have enough knowledge of the effects of each parameters, it took some time to find a satisfactory classifier. The above computational result is the outcome of this adjustment. To build an effective model, we need some better algorithm to carefully parameters. Also, choosing the right number of neurons and the number of layers will be crucial in building a better classifier. Since this paper only covers the very basics of neural network, there are lots of room to further explore the applications and algorithms of neural network. For example, we had only talked about just one type of neural network in this paper, which was a multilayer neural network. There are many other types such as convolutional neural network and modular neural network. Comparisons between these different types of neural networks can be made to understand them better.

6 Appendix

6.1 Jupyter Notebook Codes

[1]:
```
%matplotlib inline

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_____csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_____selection import train_____test_____split
import random

import os
print(os.listdir("../input"))
```

```
['train.csv', 'test.csv']
```

Load Train and Test data

[2]:
```
# Read csv file as DataFrame
train = pd.read_____csv('../input/train.csv')
test = pd.read_____csv('../input/test.csv')
```

[3]:
```
print("The size of train dataset:", train.shape)
train.head()
```

```
The size of train dataset: (42000, 785)
```

[3]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pi
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

5 rows × 785 columns

[4]:
```
# test dataset doesn't include label
print("The size of test dataset:", test.shape)
test.head()
```

```
The size of test dataset: (28000, 784)
```

[4]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	p
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

5 rows × 784 columns

[6]:
```
# Separate into dataset and feature (target variable)
X_____train = train.iloc[:, 1:].as_____matrix()
y_____train = train.iloc[:, 0].as_____matrix()
X_____test = test
```

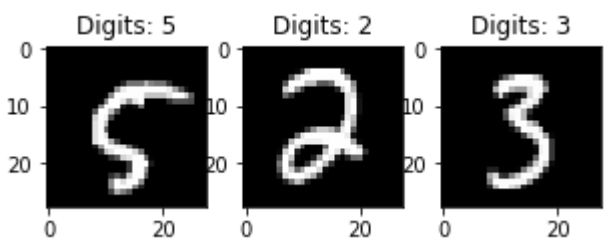
Preprocessing images

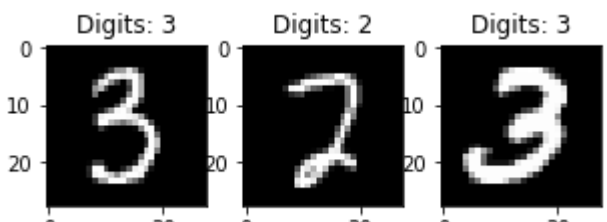
9 example images of digits

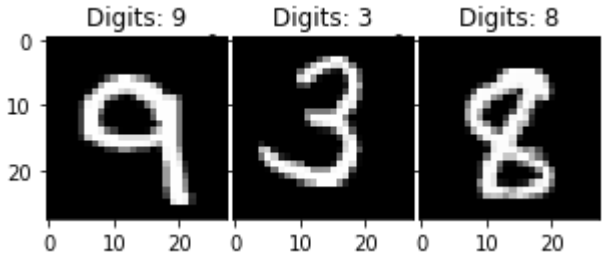
[166]:
```
# create 9 random lists of indices
randomdigit = []
for j in range(9):
  randomdigit.append(random.randint(1, len(X_____train)))
randomdigit
```

[166]:
```
[5923, 38055, 32671, 6816, 3479, 17767, 15640, 4253, 23181]
```

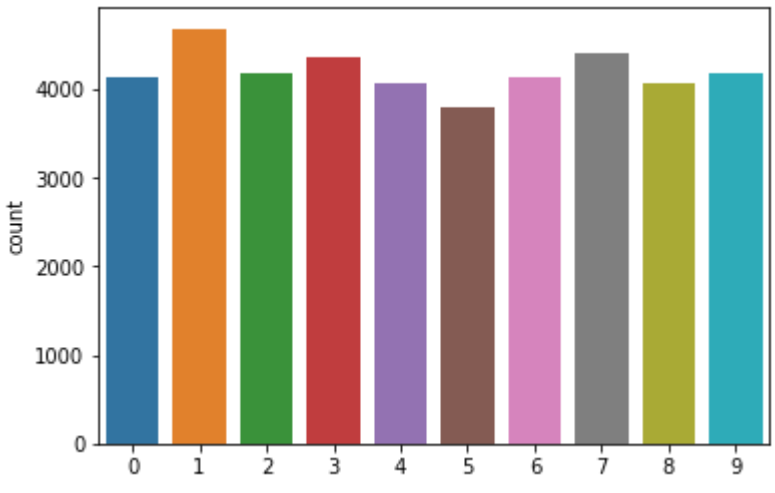
[167]:
```
for i, j in zip(range(3), range(len(randomdigit))):
  plt.figure(figsize=(5,5))
  plt.subplot(3, 3, i*3+1), plt.imshow(X_____train[randomdigit[j*3]].reshape(28,28), cmap='gray')
  plt.title('Digits: {}'.format(y_____train[randomdigit[j*3]]))
  plt.subplot(3, 3, i*3+2), plt.imshow(X_____train[randomdigit[j*3+1]].reshape(28,28), cmap='gray')
  plt.title('Digits: {}'.format(y_____train[randomdigit[j*3+1]]))
  plt.subplot(3, 3, i*3+3), plt.imshow(X_____train[randomdigit[j*3+2]].reshape(28,28), cmap='gray')
  plt.title('Digits: {}'.format(y_____train[randomdigit[j*3+2]]))
plt.subplots_____adjust(wspace=0, hspace=0)
plt.show()
```







[9]:
```
# Show the distribution of digits
sns.countplot(y_____train)
plt.show()
```



It shows that dataset of digits are **uniformly** distributed, which tells us that it's a good data to work with! We don't have to normalize the data.

Training model with 1-layer of Neural Network

Cross-Validation

[10]:
```
# Split training datasets into random train and test subsets
img_____train, img_____test, label_____train, label_____test = train_____test_____split(X_____train, y_____train, te
print("The dataset was splitted into", len(img_____train), "training sets")
print("The dataset dataset was splitted into", len(img_____test), "testing sets")
```

```
The dataset was splitted into 33600 training sets
The dataset dataset was splitted into 8400 testing sets
```

Build a 1-layer training perceptron neural network model

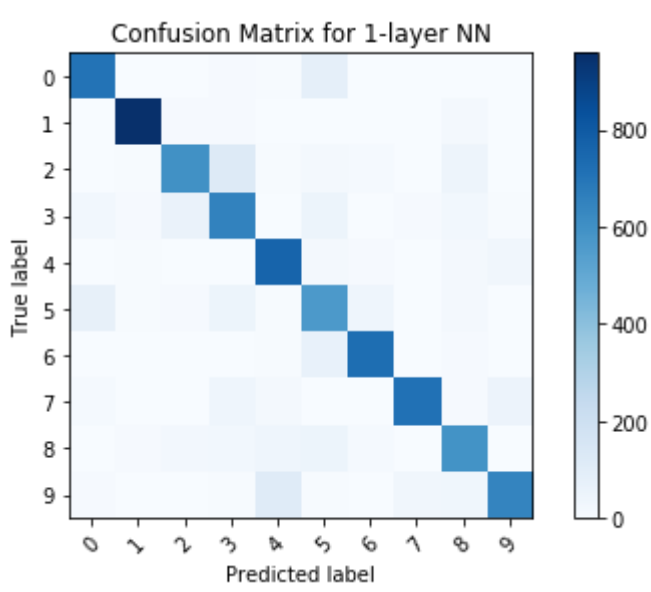
[14]:
```
from sklearn.neural_____network import MLPClassifier

mlp = MLPClassifier(hidden_____layer_____sizes=(10,), max_____iter=100, learning_____rate='adaptive', random___
mlp.fit(img_____train, label_____train)
```

[14]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_____size='auto', beta_____1=0.9,
    beta_____2=0.999, early_____stopping=False, epsilon=1e-08,
    hidden_____layer_____sizes=(10,), learning_____rate='adaptive',
    learning_____rate_____init=0.001, max_____iter=100, momentum=0.9,
    nesterovs_____momentum=True, power_____t=0.5, random_____state=10, shuffle=True,
    solver='adam', tol=0.0001, validation_____fraction=0.1, verbose=False,
    warm_____start=False)
```

[12]:
```
## The confusion matrix provides more detailed information about Classifier performance.
def plot_confusion_matrix(cm, title, lists, cmap=plt.cm.Blues):
  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(lists))
  plt.xticks(tick_marks, lists, rotation=45)
  plt.yticks(tick_marks, lists)
  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
```

[19]:
```
predictions = mlp.predict(img_test)
from sklearn.metrics import classification_report,confusion_matrix
cm = confusion_matrix(label_test, predictions)
print(classification_report(label_test, predictions))
plot_confusion_matrix(cm, 'Confusion Matrix for 1-layer NN',  list(range(0,10)))
```

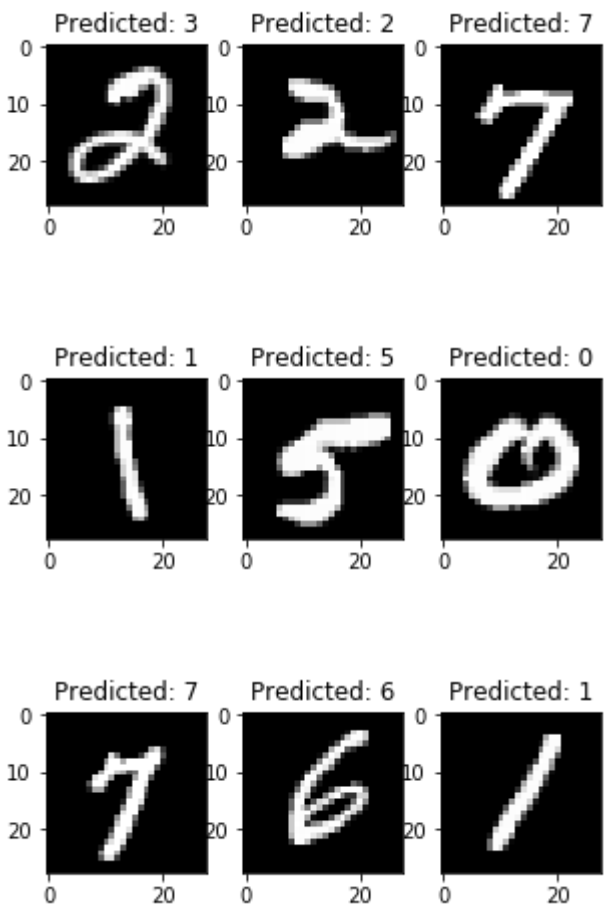


Plots of Images

[163]:
```
# create 9 random lists of indices
randomdigit = []
for j in range(9):
  randomdigit.append(random.randint(1,8400))
randomdigit
```

[163]:
```
[2160, 5893, 6499, 5391, 3863, 3151, 4200, 7571, 7593]
```

[164]:
```
# plot 9 predicted digits with associated images
for i, j in zip(range(3), range(len(randomdigit))):
    plt.figure(figsize=(5,5))
    plt.subplot(3, 3, i*3+1), plt.imshow(img_____test[randomdigit[j*3]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions[randomdigit[j*3]]))
    plt.subplot(3, 3, i*3+2), plt.imshow(img_____test[randomdigit[j*3+1]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions[randomdigit[j*3+1]]))
    plt.subplot(3, 3, i*3+3), plt.imshow(img_____test[randomdigit[j*3+2]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions[randomdigit[j*3+2]]))
plt.show()
```



2-layer network

[126]:
```
mlp2 = MLPClassifier(hidden_____layer_____sizes=(20, 10), max_____iter=300, learning_____rate='adaptive', randon
mlp2.fit(img_____train, label_____train)
```

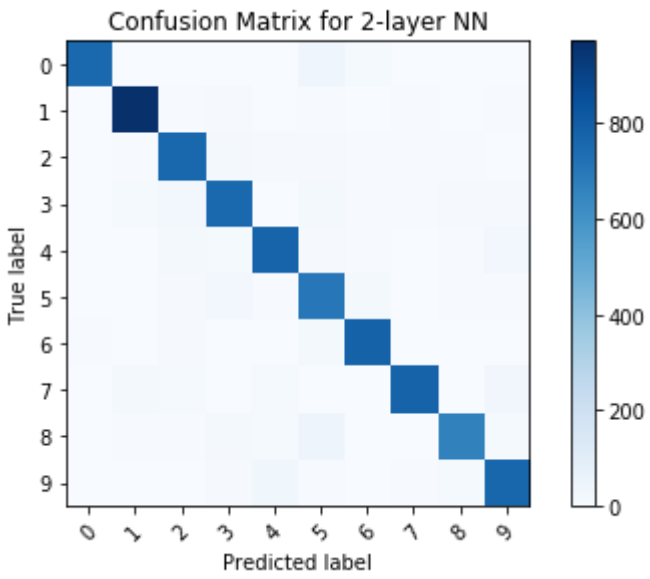
[126]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_____size='auto', beta_____1=0.9,
    beta_____2=0.999, early_____stopping=False, epsilon=1e-08,
    hidden_____layer_____sizes=(20, 10), learning_____rate='adaptive',
    learning_____rate_____init=0.001, max_____iter=300, momentum=0.9,
    nesterovs_____momentum=True, power_____t=0.5, random_____state=10, shuffle=True,
    solver='adam', tol=0.0001, validation_____fraction=0.1, verbose=False,
    warm_____start=False)
```

[127]:
```
predictions2 = mlp2.predict(img_____test)
cm2 = confusion_____matrix(label_____test, predictions2)
print(classification_____report(label_____test, predictions2))
plot_____confusion_____matrix(cm2, 'Confusion Matrix for 2-layer NN',  list(range(0,10)))
```

```
             precision    recall  f1-score   support

          0       0.98      0.93      0.96       810
          1       0.96      0.97      0.96      1004
          2       0.90      0.93      0.91       821
          3       0.89      0.90      0.89       841
          4       0.91      0.91      0.91       854
          5       0.82      0.90      0.86       781
          6       0.94      0.96      0.95       816
          7       0.97      0.91      0.94       858
          8       0.94      0.86      0.90       771
          9       0.89      0.91      0.90       844

avg / total       0.92      0.92      0.92      8400
```

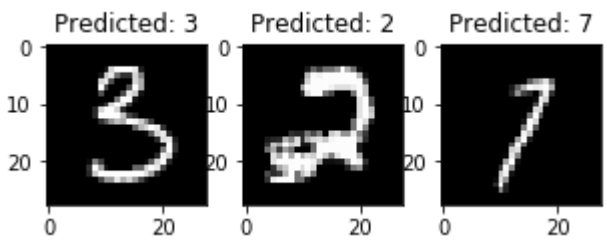


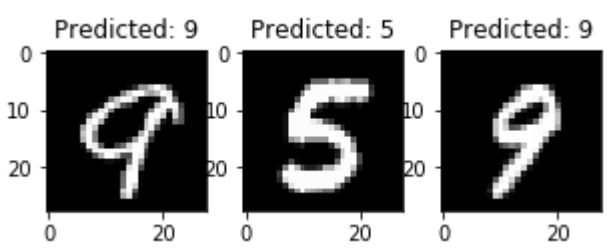
Plots of Images

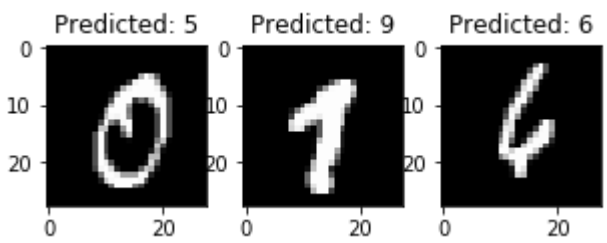
[159]:
```
# create 9 random lists of indices
randomdigit = []
for j in range(9):
  randomdigit.append(random.randint(1,8400))
randomdigit
```

[159]:
```
[7147, 8383, 1644, 7668, 3698, 453, 456, 391, 5950]
```

[160]:
```
# plot 9 predicted digits with associated images
for i, j in zip(range(3), range(len(randomdigit))):
    plt.figure(figsize=(5,5))
    plt.subplot(3, 3, i*3+1), plt.imshow(img_____test[randomdigit[j*3]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions2[randomdigit[j*3]]))
    plt.subplot(3, 3, i*3+2), plt.imshow(img_____test[randomdigit[j*3+1]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions2[randomdigit[j*3+1]]))
    plt.subplot(3, 3, i*3+3), plt.imshow(img_____test[randomdigit[j*3+2]].reshape(28,28), cmap='gray')
    plt.title('Predicted: {}'.format(predictions2[randomdigit[j*3+2]]))
plt.show()
```







References

[1] Why Deep Learning Is Suddenly Changing Your Life. Fortune, fortune.com/ai-artificial-intelligence-deep-machine-learning/.

[2] Portilla, Jose. A Beginner's Guide to Neural Networks in Python and SciKit Learn 0.18. Springboard Blog, 7 May 2017, www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/.

[3] Sklearn.neural-network.MLPClassifier. Sklearn.neural-network.MLPClassifier - scikit-Learn 0.19.1 documentation, scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[4] Nielsen, Michael A. Neural Networks and Deep Learning. Neural networks and deep learning, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap1.html.

[5] text provided by a Professor. "Chapter 4 Neural Networks and Deep Learning".