Neural Network(NN)

May 7, 2020

[13]:
```
#Amul Neupane
#Machine Learning Fundamentals


%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

[14]:
```
image_size = 28 # width and length
no_of_different_labels = 10 # i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
train_data = np.loadtxt("mnist_train.csv", delimiter=",")
test_data = np.loadtxt("mnist_test.csv", delimiter=",")
test_data[:10]
```

[14]:
```
array([[7., 0., 0., …, 0., 0., 0.],
       [2., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       …,
       [9., 0., 0., …, 0., 0., 0.],
       [5., 0., 0., …, 0., 0., 0.],
       [9., 0., 0., …, 0., 0., 0.]])
```

[15]:
```
test_data[test_data==255]
test_data.shape
```

[15]:
```
(10000, 785)
```

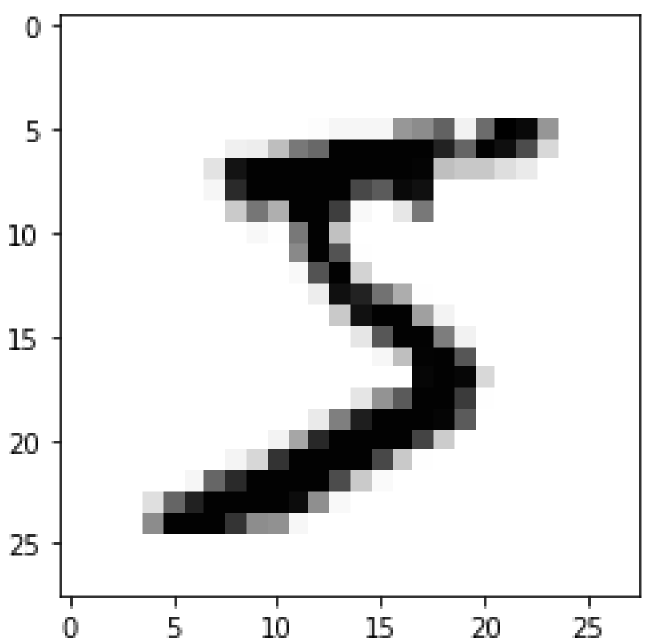
[16]:
```
fac = 0.99 / 255
train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01
train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])
```

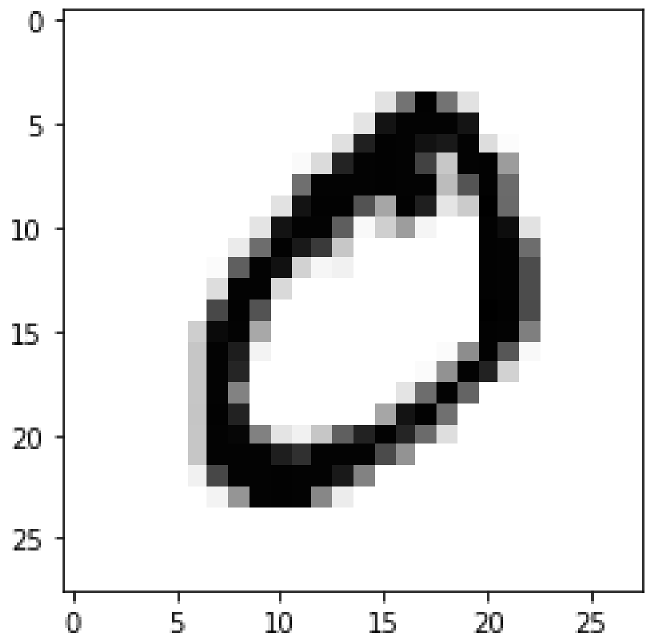
[17]:
```
import numpy as np

lr = np.arange(10)


for label in range(10):
    one_hot = (lr==label).astype(np.int)
    print("label: ", label, " in one-hot representation: ", one_hot)
```

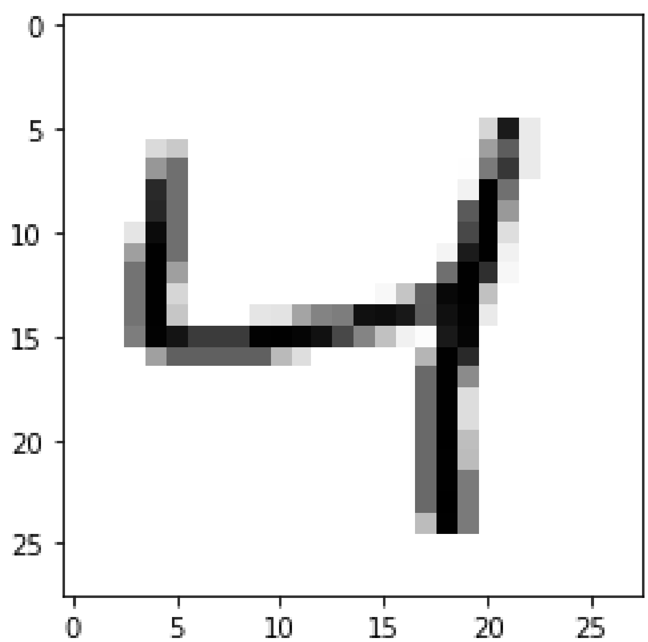
```
label: 0 in one-hot representation: [1 0 0 0 0 0 0 0 0 0]
label: 1 in one-hot representation: [0 1 0 0 0 0 0 0 0 0]
label: 2 in one-hot representation: [0 0 1 0 0 0 0 0 0 0]
label: 3 in one-hot representation: [0 0 0 1 0 0 0 0 0 0]
label: 4 in one-hot representation: [0 0 0 0 1 0 0 0 0 0]
label: 5 in one-hot representation: [0 0 0 0 0 1 0 0 0 0]
label: 6 in one-hot representation: [0 0 0 0 0 0 1 0 0 0]
label: 7 in one-hot representation: [0 0 0 0 0 0 0 1 0 0]
label: 8 in one-hot representation: [0 0 0 0 0 0 0 0 1 0]
label: 9 in one-hot representation: [0 0 0 0 0 0 0 0 0 1]
```

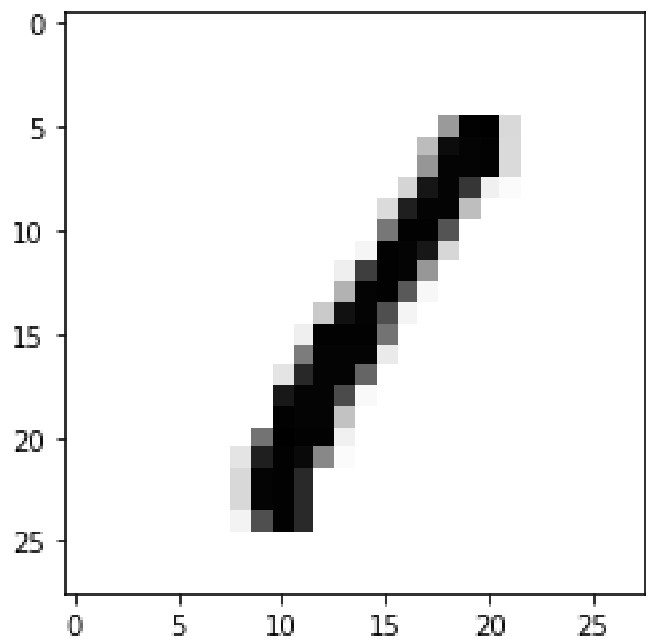
[18]:
```
lr = np.arange(no_of_different_labels) #
transform labels into one hot representation
train_labels_one_hot =
(lr==train_labels).astype(np.float)
test_labels_one_hot =
(lr==test_labels).astype(np.float) # we don't want
zeroes and ones in the labels neither:
train_labels_one_hot[train_labels_one_hot==0] =
0.01 train_labels_one_hot[train_labels_one_hot==1]
= 0.99 test_labels_one_hot[test_labels_one_hot==0]
= 0.01 test_labels_one_hot[test_labels_one_hot==1]
= 0.99
```

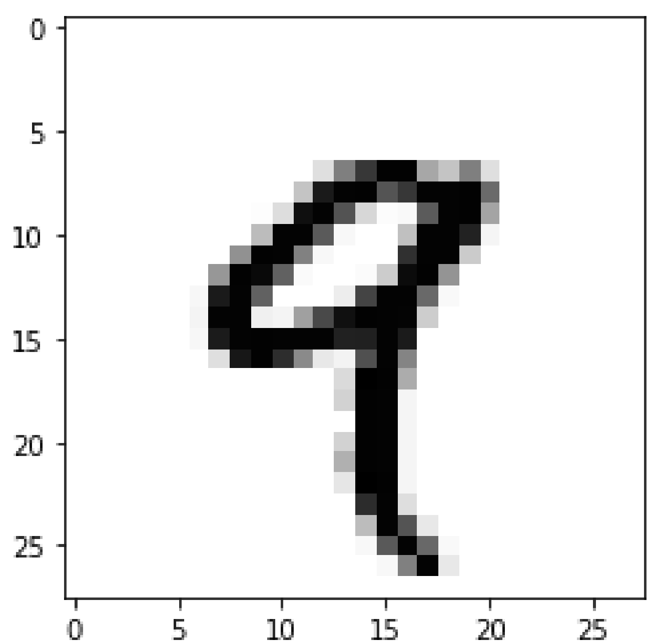
[19]:
```
for i in range(10):
    img =
    train_imgs[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.show()
```

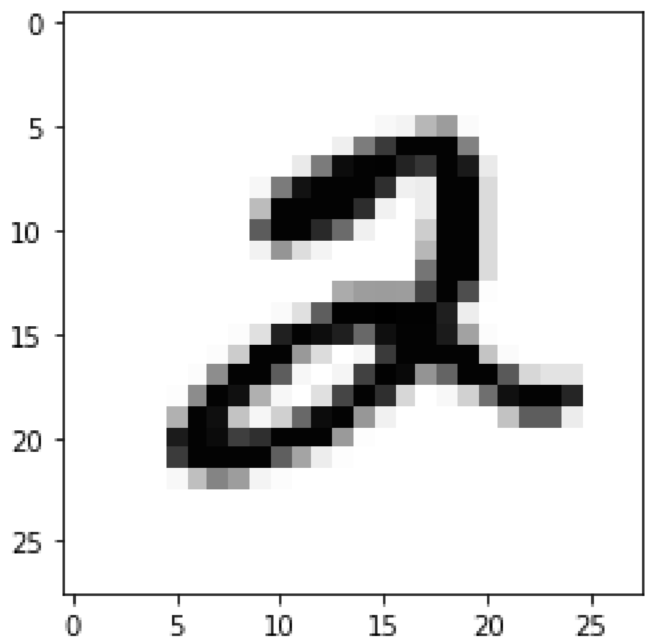


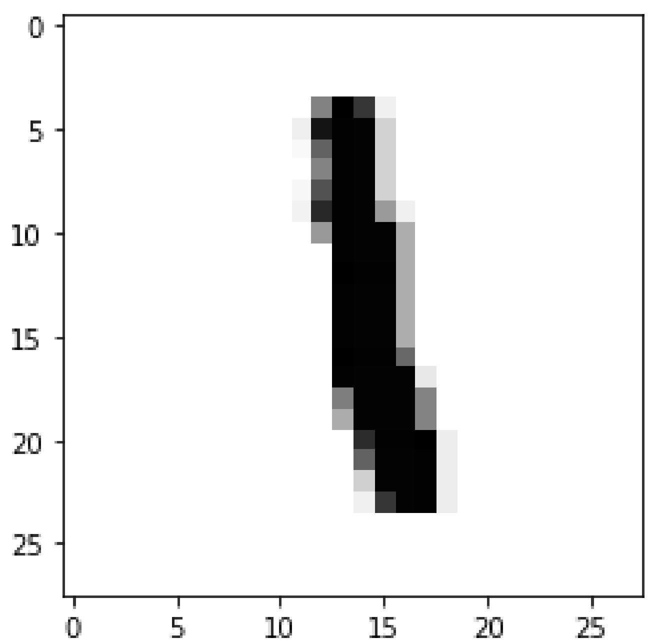

0
5
10
15
20
25
0
5
10
15
20
25

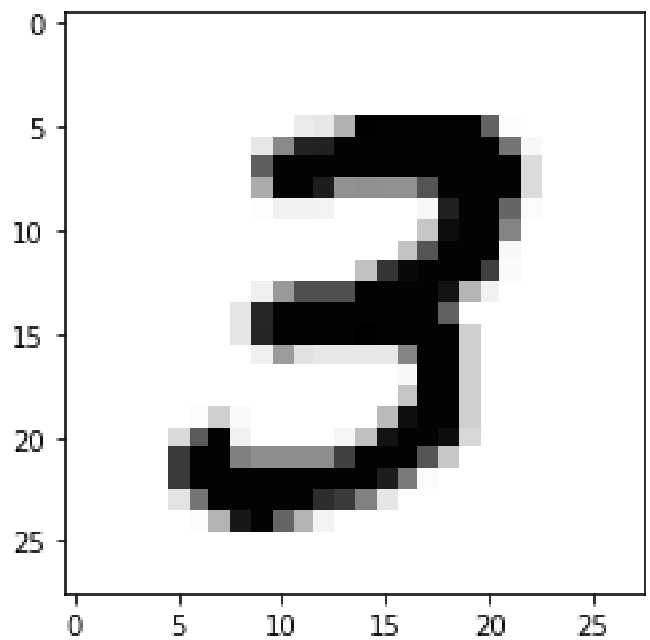

0
5
10
15
20
25
0
5
10
15
20
25

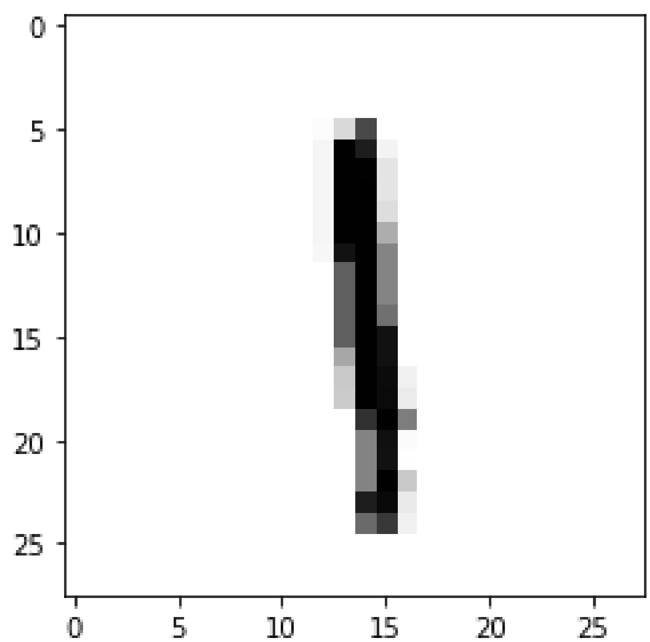

0
5
10
15
20
25
0
5
10
15
20
25

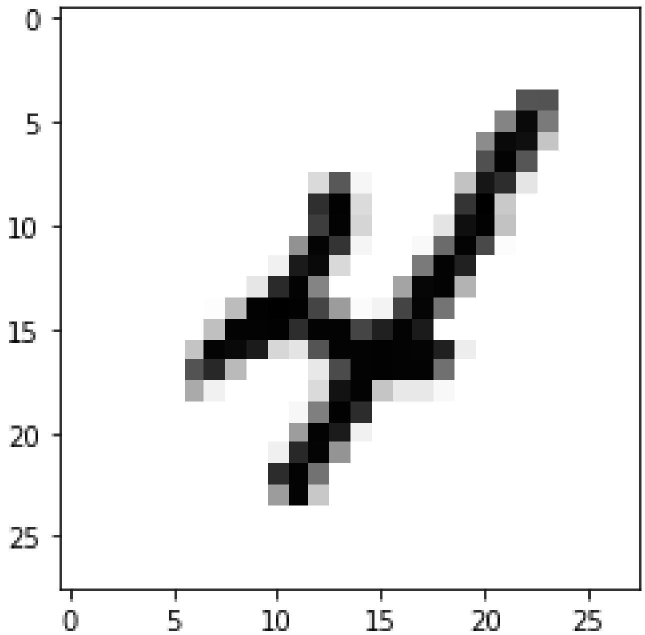

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25


0
5
10
15
20
25



[20]:
```
import numpy as np

@np.vectorize
def sigmoid(x):
```

```
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.create_weight_matrices()

    def create_weight_matrices(self):
        """
        A method to initialize the weight
        matrices of the neural network
        """
        rad = 1 / np.sqrt(self.no_of_in_nodes)
        X = truncated_normal(mean=0,
                             sd=1,
                             low=-rad,
                             upp=rad)
        self.wih = X.rvs((self.no_of_hidden_nodes,
                               self.no_of_in_nodes))
        rad = 1 / np.sqrt(self.no_of_hidden_nodes)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.who = X.rvs((self.no_of_out_nodes,
                                self.no_of_hidden_nodes))


    def train(self, input_vector, target_vector):
        """
        input_vector and target_vector can
        be tuple, list or ndarray
```

```
        """

        input_vector = np.array(input_vector, ndmin=2).T
        target_vector = np.array(target_vector, ndmin=2).T

        output_vector1 = np.dot(self.wih,
                                input_vector)
        output_hidden = activation_function(output_vector1)

        output_vector2 = np.dot(self.who,
                                output_hidden)
        output_network = activation_function(output_vector2)

        output_errors = target_vector - output_network
        # update the weights:
        tmp = output_errors * output_network \
              * (1.0 - output_network)
        tmp = self.learning_rate  * np.dot(tmp,
                                           output_hidden.T)
        self.who += tmp


        # calculate hidden errors:
        hidden_errors = np.dot(self.who.T,
                               output_errors)
        # update the weights:
        tmp = hidden_errors * output_hidden * \
              (1.0 - output_hidden)
        self.wih += self.learning_rate \
                          * np.dot(tmp, input_vector.T)




    def run(self, input_vector):
        # input_vector can be tuple, list or ndarray
        input_vector = np.array(input_vector, ndmin=2).T

        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector
```

```
    def confusion_matrix(self, data_array, labels):
        cm = np.zeros((10, 10), int)
        for i in range(len(data_array)):
            res = self.run(data_array[i])
            res_max = res.argmax()
            target = labels[i][0]
            cm[res_max, int(target)] += 1
        return cm

    def precision(self, label, confusion_matrix):
        col = confusion_matrix[:, label]
        return confusion_matrix[label, label] / col.sum()

    def recall(self, label, confusion_matrix):
        row = confusion_matrix[label, :]
        return confusion_matrix[label, label] / row.sum()


    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

[21]:
```
ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                    no_of_out_nodes = 10,
                    no_of_hidden_nodes = 100,
                    learning_rate = 0.1)


for i in range(len(train_imgs)):
    ANN.train(train_imgs[i], train_labels_one_hot[i])
```

[22]:
```
for i in range(20):
    res = ANN.run(test_imgs[i])
    print(test_labels[i], np.argmax(res), np.max(res))
```

```
[7.] 7 0.9842548971366076
[2.] 2 0.9653023396196856
[1.] 1 0.9892975291775858 [0.] 0 0.9762807408727853
```

```
[4.] 4 0.966594439838561
[1.] 1 0.9885194311291068
[4.] 4 0.9796051458149712
[9.] 9 0.9873741454725488
[5.] 5 0.42485686807350503
[9.] 9 0.9341463979200473
[0.] 0 0.97665545422355
[6.] 6 0.747960016606698
[9.] 9 0.9918615724951698
[0.] 0 0.9773171789413514
[1.] 1 0.9925353898764244
[5.] 5 0.9192422056600144
[9.] 9 0.9936136684680693
[7.] 7 0.9754448384597486
[3.] 3 0.7916037440633885
[4.] 4 0.9896618681664278
```

[23]:
```
corrects, wrongs = ANN.evaluate(train_imgs,
train_labels) print("accuracy train: ", corrects /
( corrects + wrongs)) corrects, wrongs =
ANN.evaluate(test_imgs, test_labels)
print("accuracy: test", corrects / ( corrects +
wrongs))

cm = ANN.confusion_matrix(train_imgs,
train_labels) print(cm)

for i in range(10):
    print("digit: ", i, "precision: ", ANN.precision(i, cm),
  "recall: ", ANN. ↪recall(i, cm))
```

```
accuracy train: 0.94725
accuracy: test 0.945
[[5808    0   51   17   10   30   35   11   19   30]
 [   0 6627   73   28   15   28   22   63   96   10]
 [   2   19 5422   42   21   10    8   33    6    2]
 [   3   38  124 5819    0  135    5   33  140   75]
 [  14   11   57   11 5450   28    8   40   27   62]
 [   7    3    6   54    0 4995   37    2   15    7]
 [  31    3   57   18   45   58 5763    4   26    4]
 [   0   10   48   42    4    5    1 5837    0   34]
 [  47   15  101   43    6   59   38   18 5419   30]
 [  11   16   19   57  291   73    1  224  103 5695]]
digit: 0 precision: 0.9805841634306939 recall:
0.9662285809349526 digit: 1 precision: 0.9829427469593592
recall: 0.9518816432059753 digit: 2 precision:
```

```
0.9100369251426653 recall: 0.974303683737646 digit: 3
precision: 0.9491110748654379 recall: 0.9132140615191463 digit:
4 precision: 0.9328996918863403 recall: 0.9548002803083392
digit: 5 precision: 0.9214167127836193 recall:
0.9744440109246976 digit: 6 precision: 0.973808719161879
recall: 0.9590614078881677 digit: 7 precision:
0.931683958499601 recall: 0.9759237585688012 digit: 8
precision: 0.9261664672705521 recall: 0.9381925207756233 digit:
9 precision: 0.9573037485291646 recall: 0.8775038520801233
```

[ ]:

[ ]: