Deep Learning - Handwritten Digits Recognition

April 7, 2017

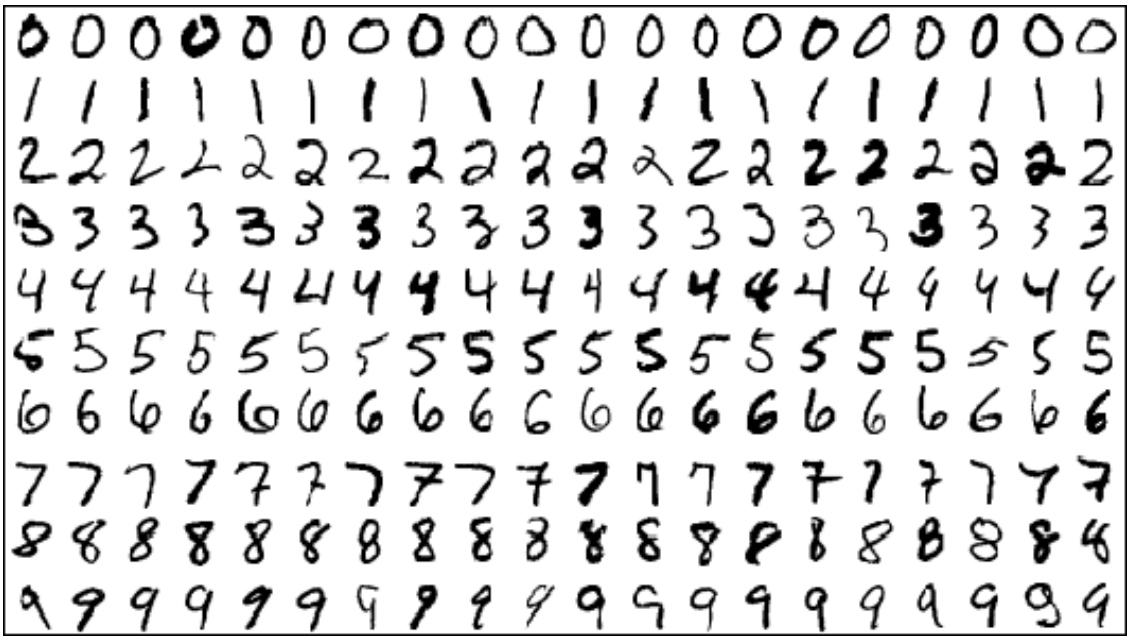
BENEDETTO, Luca | IBARRONDO, Alberto

1 INTRODUCTION

In this session, your will implement, train and test a Neural Network for the Handwritten Digits Recognition problem [1] with different settings of hyper parameters. You will use the MNIST dataset which was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). Images of digits were taken from a variety of scanned documents, normalized in size and centered.

```
In [5]: # FIGURE 1: MNIST digits examples
```

```
Out[5]:
```



This assignment includes a written part of programms to help you understand how to build and train your neural net and then to test your code and get restults.

1. NeuralNetwork.py
2. transfer_functions.py
3. utils.py

We use the following libraries:

1. numpy : for creating arrays and using methods to manipulate arrays.
2. matplotlib : for making plots

2 A PRIMITIVE NEURAL NETWORK

2.1 Manual design of a Neural Network

Before designing and writing your code, you will first work on a neural network by hand. Consider the above Neural network with two inputs $X = (x1, x2)$, one hidden layers and a single output unit (y). The initial weights are set to random values. Neurons 6 and 7 represent the bias. Bias values are equal to 1.
Training sample, X = (0.8, 0.2), whose class label is Y=0.4.

Assume that the neurons have a Sigmoid activation function $f(x) = \frac{1}{(1+e^{-x})}$ and the learning rate μ=1

2.1.1 Question 1.1.1

Compute the new values of weights $w_{i,j}$ after a forward pass and a backward pass. $w_{i,j}$ is the weight of the connexion between neuron i and neuron j.

```
In [6]: import numpy as np
        import matplotlib as plt
        import NNet as nn
        import transfer_functions as tf
```

```
In [7]: # Calculating manually all the values
        # Initial values
        x = [0.8, 0.2]
        w1 = [[0.3, -0.5], [0.8, 0.2], [0.2, -0.4]]
        w2 = [-0.6, 0.4, 0.5]
        y = 0.4
        u = 1

        #Feedforward computation
        inpt = np.append(x, 1.0)

        u1 = np.dot(inpt, w1)
        o1 = np.append(tf.sigmoid(u1), 1.0)

        u2 = np.dot(o1, w2)
        o2 = tf.sigmoid(u2)
```

```
        #Backward computation
        E = (1.0/2.0)*((y-o2)**2.0)

        dEdu2 = (y-o2)*tf.dsigmoid(o2)
        dEdu1 = np.dot(w2, dEdu2)*tf.dsigmoid(o1)

        w2 += u*o1.T.dot(dEdu2)
        w1 += u*np.outer(inpt, dEdu1[:-1])

        print('u1 : ', u1)
        print('o1: ', o1)
        print('u2 : ', u2)
        print('o2: ', o2)
        print('w2 : ', w2)
        print('w1: ', w1)
```

```
u1 :  [ 0.6  -0.76]
o1:  [ 0.64565631  0.31864627  1.        ]
u2 :  0.240064722749
o2:  0.55972959911
w2 :  [-0.62541468  0.38745727  0.46063746]
w1:  [[ 0.30432265 -0.50273473]
 [ 0.80108066  0.19931632]
 [ 0.20540332 -0.40341841]]
```

VALUES COMPUTED:

$w_{1,3} = 0.30345983$
$w_{1,4} = -0.50218887$
$w_{2,3} = 0.80108066$
$w_{2,4} = 0.19931632$
$w_{6,3} = 0.20540332$
$w_{6,4} = -0.40341841$
$w_{3,5} = -0.62541468$
$w_{4,5} = 0.38745727$
$w_{7,5} = 0.46063746$

2.2 Basic Neural Network Implementation on Python

2.2.1 Question 1.2.1

Define the neural network corresponding to the one in part 2.1

```
In [8]: #create the network
        my_nnet = nn.NNet(n_input=2,netDims=[2,1],learn=1)
```

```
In [9]: #Data preparation
        X=[0.8,0.2]
        Y=[0.4]

        #initialize weights
        wi=np.array([[0.3,-0.5],[0.8,0.2],[0.2,-0.4]])
        wo=np.array([[-0.6],[0.4],[0.5]])

        my_nnet.init_w([wi,wo])
        print(my_nnet.W)

[array([[ 0.3, -0.5],
       [ 0.8,  0.2],
       [ 0.2, -0.4]]), array([[-0.6],
       [ 0.4],
       [ 0.5]])]
```

2.2.2 Question 1.2.2

Implement the Feed Forward function (feedForward(X) in the NeuralNetwork.py file)

```
In [10]: def feedForward(selfa, inputs):
             # Set input with untouched bias. 1st activation
             selfa.input[0:selfa.n_input-1] = inputs
             selfa.values[0] = selfa.tf(np.append(
                 np.dot(selfa.input,
                        selfa.W[0]),
                 1.0))

             #Hidden activations
             for layer in range(1, selfa.n_layers-1):
                 selfa.values[layer] = selfa.tf(np.append
                     (np.dot(selfa.values[layer-1],
                             selfa.W[layer]),
                      1.0))

             #Output activation (no bias)
             selfa.values[-1] = selfa.tf(
                 np.dot(selfa.values[-2], selfa.W[-1]))
             selfa.output = selfa.values[-1]

             return selfa.values[-1]
```

Check that our network outputs the expected value (the one you computed in question 1.1)

```
In [11]: # test my  Feed Forward function
         Output_activation=my_nnet.feedForward(X)
         print("output activation =%.3f" %(Output_activation))

output activation =0.560
```

2.2.3 Question 1.2.3

Implement the Back-propagation Algorithm (backPropagate(Y) in the NeuralNetwork.py file)

```
In [12]: def backPropagate(selfa, targets):
             selfa.dEdU[-1] =   (selfa.output-targets) * \
                                 selfa.dtf(selfa.output)
             selfa.dEdU[-2] = np.multiply(
                         np.dot(selfa.W[-1], selfa.dEdU[-1]),
                         selfa.dtf(selfa.values[-2]))
             # calculate error terms for hidden layers
             for layer in range(selfa.n_layers-2, 0, -1):
                 selfa.dEdU[layer-1] = np.multiply(
                     np.dot(selfa.W[layer],
                            selfa.dEdU[layer][:-1]),
                     selfa.dtf(selfa.values[layer-1]))

             # update network weights
             selfa.W[-1] -= selfa.learn * \
                           np.outer(selfa.values[-2],
                                    selfa.dEdU[-1])

             for layer in range(1, selfa.n_layers-1):
                 selfa.W[layer] -= selfa.learn * \
                                   np.outer(selfa.values[layer-1],
                                            selfa.dEdU[layer][:-1])
             selfa.W[0] -= selfa.learn * \
                           np.outer(selfa.input,
                                    selfa.dEdU[0][:-1])

             # calculate error
             E = (1.0/2.0)*((targets-selfa.output)**2.0)
```

Checking that the gradient values and weight updates are correct (similar to the ones you computed in question 1.1)

```
In [13]: #test Back-propagation function
         wi=np.array([[0.3,-0.5],[0.8,0.2],[0.2,-0.4]])
         wo=np.array([[-0.6],[0.4],[0.5]])
         my_nnet.init_w([wi,wo])
```

```
        Output_activation=my_nnet.feedForward(X)
        my_nnet.backPropagate(Y)

        #Print weights after backpropagation and comparing
        print(w2)
        print(my_nnet.W[-1].T)
        print(my_nnet.W[-1].T==w2)
        print(w1)
        print(my_nnet.W[0])
        print(my_nnet.W[0]==w1)
```

```
[-0.62541468  0.38745727  0.46063746]
[[-0.62541468  0.38745727  0.46063746]]
[[ True  True  True]]
[[ 0.30432265 -0.50273473]
 [ 0.80108066  0.19931632]
 [ 0.20540332 -0.40341841]]
[[ 0.30432265 -0.50273473]
 [ 0.80108066  0.19931632]
 [ 0.20540332 -0.40341841]]
[[ True  True]
 [ True  True]
 [ True  True]]
```

Our Feed Forward and Back-Propagation implementations are working, Great!! Let's tackle a real world problem.

3 THE MNIST CHALLENGE

3.1 Data Preparation

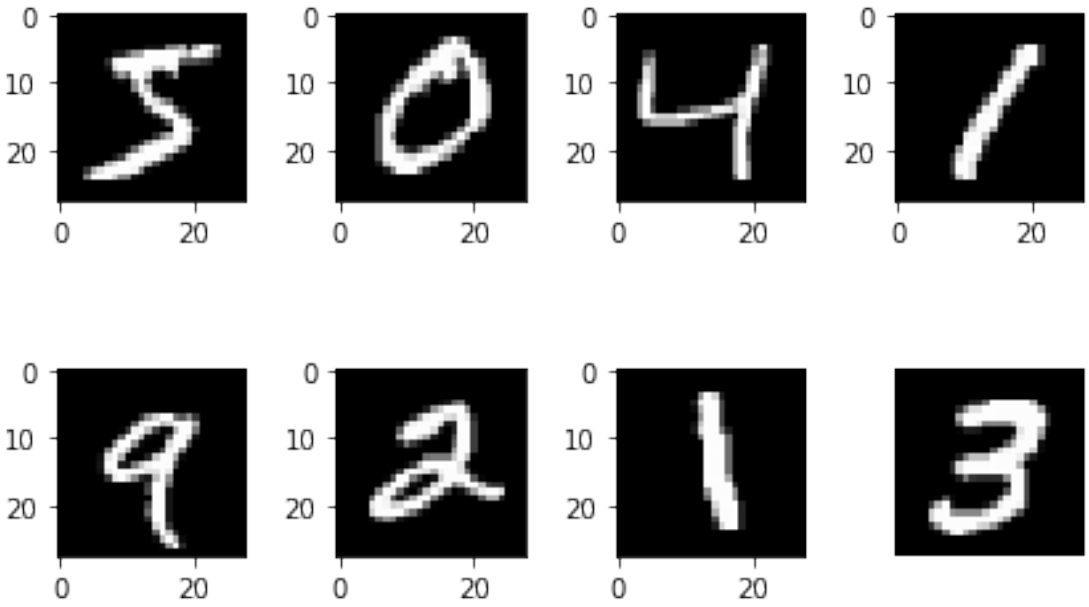
The MNIST dataset consists of handwritten digit images it contains 60,000 examples for the training set and 10,000 examples for testing. In this Lab Session, the official training set of 60,000 is divided into an actual training set of 50,000 examples, 10,000 validation examples and 10,000 examples for test. All digit images have been size-normalized and centered in a fixed size image of 28 x 28 pixels. The images are stored in byte form you will use the NumPy python library to read the data files into NumPy arrays that we will use to train the ANN.

The MNIST dataset is available in the Data folder. To get the training, testing and validation data, run the the load_data() function.

```
In [19]: from utils import *
         import gzip

         np.random.seed(1990)

         with gzip.open('mnist.pkl.gz', 'r') as f:
                 train_set, valid_set, test_set = \
```

```
                    pickle.load(f, encoding='latin1')
        f.close()

        training_data = [(train_set[0][i],
                          [1 if j == train_set[1][i]\
                           else 0 for j in range(10)]) \
                        for i in np.arange(len(train_set[0]))]

        test_data =     [(test_set[0][i],
                          [1 if j == test_set[1][i]\
                           else 0 for j in range(10)]) \
                        for i in np.arange(len(test_set[0]))]
        validation_data = [(valid_set[0][i],
                          [1 if j == valid_set[1][i] \
                           else 0 for j in range(10)]) \
                        for i in np.arange(len(valid_set[0]))]
```

In [20]:
```
        # MNIST Dataset Digits Visualisation
        ROW = 2
        COLUMN = 4
        for i in range(ROW * COLUMN):
            # train[i][0] is i-th image data with size 28x28
            image = training_data[i][0].reshape(28, 28)
            plt.subplot(ROW, COLUMN, i+1)
            plt.imshow(image, cmap='gray')
        plt.axis('off')
        plt.tight_layout()    # padding between subplots
        plt.show()
```



3.2 NNet Implementation

The input layer of the neural network contains neurons encoding the values of the input pixels. The training data for the network will consist of many 28 by 28 pixel images of scanned handwritten digits, and so the input layer contains 784=28×28 neurons. The second layer of the network is a hidden layer, we set the neuron number in the hidden layer to 30. The output layer contains 10 neurons.

3.2.1 Question 2.1.1

Create the network described above using the NeuralNetwork class

```
In [21]: #create the network
         my_mnist_net = nn.NNet(n_input=784,
                                netDims=[30,10],
                                n_iter=50,
                                learn=0.1)
```

3.2.2 Question 2.1.2

Add the information about the performance of the neural network on the test set at each epoch

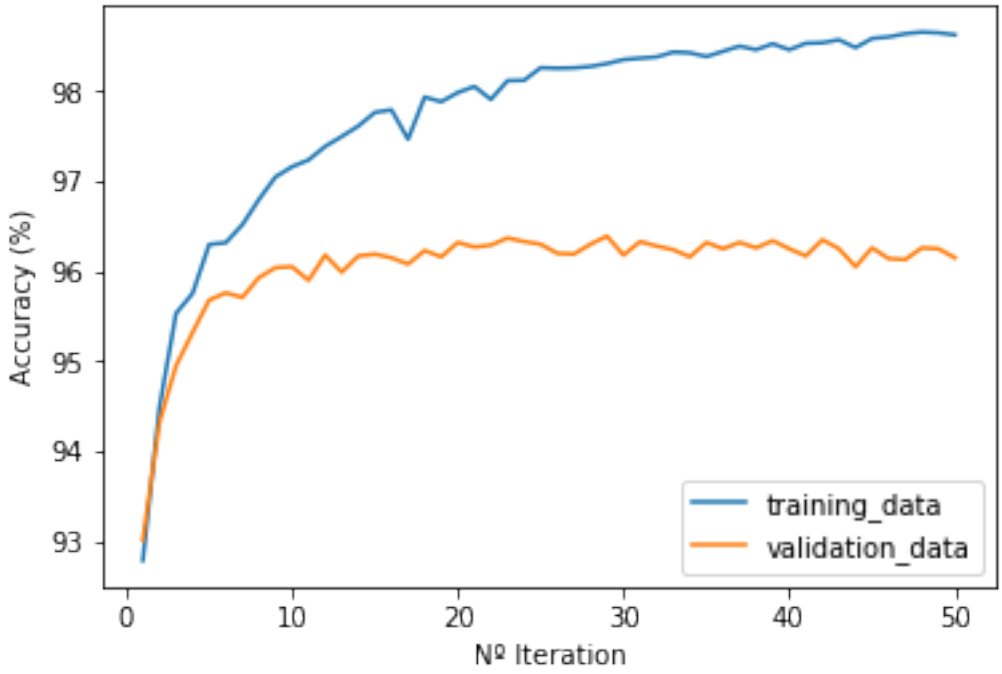
```
In [22]: test_accuracy=my_mnist_net.predict(test_data)/100
         print('Test_Accuracy  %-2.2f' % test_accuracy)

Test_Accuracy  9.28
```

3.2.3 Question 2.1.3

Train the Neural Network and comment your findings

```
In [23]: #train your network
         Tr_acc1, Val_acc1 = my_mnist_net.train(training_data,
                                                validation_data,
                                                True, True, True)
         my_mnist_net.save('mnist_nD30-10_it20_l01.model')

iter:  1/50 --> E: 0.1147973388  -Training_Accuracy:  92.79  -t: 8.11
iter:  2/50 --> E: 0.0553362555  -Training_Accuracy:  94.49  -t: 16.36
iter:  3/50 --> E: 0.0456861361  -Training_Accuracy:  95.53  -t: 24.68
iter:  4/50 --> E: 0.0408844618  -Training_Accuracy:  95.75  -t: 32.74
```

```
iter:  5/50 --> E: 0.0374169214  -Training_Accuracy:  96.30  -t: 40.90
iter:  6/50 --> E: 0.0347601436  -Training_Accuracy:  96.32  -t: 49.35
iter:  7/50 --> E: 0.0329985071  -Training_Accuracy:  96.52  -t: 57.44
iter:  8/50 --> E: 0.0312234614  -Training_Accuracy:  96.80  -t: 65.46
iter:  9/50 --> E: 0.0298837979  -Training_Accuracy:  97.05  -t: 74.14
iter: 10/50 --> E: 0.0288048911  -Training_Accuracy:  97.16  -t: 82.32
iter: 11/50 --> E: 0.0275502516  -Training_Accuracy:  97.24  -t: 90.53
iter: 12/50 --> E: 0.0266483035  -Training_Accuracy:  97.39  -t: 98.70
iter: 13/50 --> E: 0.0258145082  -Training_Accuracy:  97.50  -t: 107.06
iter: 14/50 --> E: 0.0248970720  -Training_Accuracy:  97.61  -t: 115.30
iter: 15/50 --> E: 0.0241583811  -Training_Accuracy:  97.77  -t: 124.03
iter: 16/50 --> E: 0.0234566254  -Training_Accuracy:  97.79  -t: 132.51
iter: 17/50 --> E: 0.0228321190  -Training_Accuracy:  97.47  -t: 140.63
iter: 18/50 --> E: 0.0222547701  -Training_Accuracy:  97.94  -t: 148.82
iter: 19/50 --> E: 0.0217058345  -Training_Accuracy:  97.88  -t: 156.91
iter: 20/50 --> E: 0.0210515393  -Training_Accuracy:  97.99  -t: 165.28
iter: 21/50 --> E: 0.0206983081  -Training_Accuracy:  98.05  -t: 173.84
iter: 22/50 --> E: 0.0201379276  -Training_Accuracy:  97.91  -t: 182.05
iter: 23/50 --> E: 0.0199634118  -Training_Accuracy:  98.12  -t: 190.49
iter: 24/50 --> E: 0.0193877310  -Training_Accuracy:  98.12  -t: 198.81
iter: 25/50 --> E: 0.0189849534  -Training_Accuracy:  98.26  -t: 206.93
iter: 26/50 --> E: 0.0186349595  -Training_Accuracy:  98.26  -t: 215.01
iter: 27/50 --> E: 0.0182543092  -Training_Accuracy:  98.26  -t: 223.15
iter: 28/50 --> E: 0.0177926842  -Training_Accuracy:  98.28  -t: 231.19
iter: 29/50 --> E: 0.0176327682  -Training_Accuracy:  98.31  -t: 239.17
iter: 30/50 --> E: 0.0170640397  -Training_Accuracy:  98.35  -t: 247.24
iter: 31/50 --> E: 0.0168797115  -Training_Accuracy:  98.37  -t: 255.21
iter: 32/50 --> E: 0.0165992933  -Training_Accuracy:  98.38  -t: 263.37
iter: 33/50 --> E: 0.0164396123  -Training_Accuracy:  98.44  -t: 271.69
iter: 34/50 --> E: 0.0160046244  -Training_Accuracy:  98.43  -t: 280.50
iter: 35/50 --> E: 0.0158575558  -Training_Accuracy:  98.39  -t: 288.54
iter: 36/50 --> E: 0.0155338574  -Training_Accuracy:  98.44  -t: 297.13
iter: 37/50 --> E: 0.0153680970  -Training_Accuracy:  98.50  -t: 305.36
iter: 38/50 --> E: 0.0151950065  -Training_Accuracy:  98.46  -t: 313.48
iter: 39/50 --> E: 0.0149670859  -Training_Accuracy:  98.53  -t: 321.77
iter: 40/50 --> E: 0.0146800978  -Training_Accuracy:  98.46  -t: 330.01
iter: 41/50 --> E: 0.0143037657  -Training_Accuracy:  98.53  -t: 338.13
iter: 42/50 --> E: 0.0142719489  -Training_Accuracy:  98.54  -t: 346.23
iter: 43/50 --> E: 0.0140652390  -Training_Accuracy:  98.57  -t: 354.31
iter: 44/50 --> E: 0.0139481231  -Training_Accuracy:  98.49  -t: 362.29
iter: 45/50 --> E: 0.0136008247  -Training_Accuracy:  98.59  -t: 370.35
iter: 46/50 --> E: 0.0135190362  -Training_Accuracy:  98.60  -t: 378.54
iter: 47/50 --> E: 0.0133454012  -Training_Accuracy:  98.64  -t: 386.84
iter: 48/50 --> E: 0.0130880541  -Training_Accuracy:  98.66  -t: 394.90
iter: 49/50 --> E: 0.0130971693  -Training_Accuracy:  98.65  -t: 402.96
iter: 50/50 --> E: 0.0127649936  -Training_Accuracy:  98.63  -t: 411.01
```



COMMENT:
After iteration nº 10 there is almost no improvement in the accuracy on the validation test.

3.2.4 Question 2.1.4

Guess digit, Implement and test a python function that predict the class of a digit (the folder images_test contains some examples of images of digits)

```
In [13]: #Your implementation goes here
         def guess(nnet_obj, digit_img):
             digit_class = my_mnist_net.feedForward(digit_img)
             return np.argmax(digit_class)


         digit_image = test_data[0][0]
         digit_val = guess(my_mnist_net, digit_img=digit_image)
         print(digit_val)
```

```
7
```

3.3 NNet Optimization

Change the neural network structure and parameters to optimize performance

3.3.1 Question 2.2.1

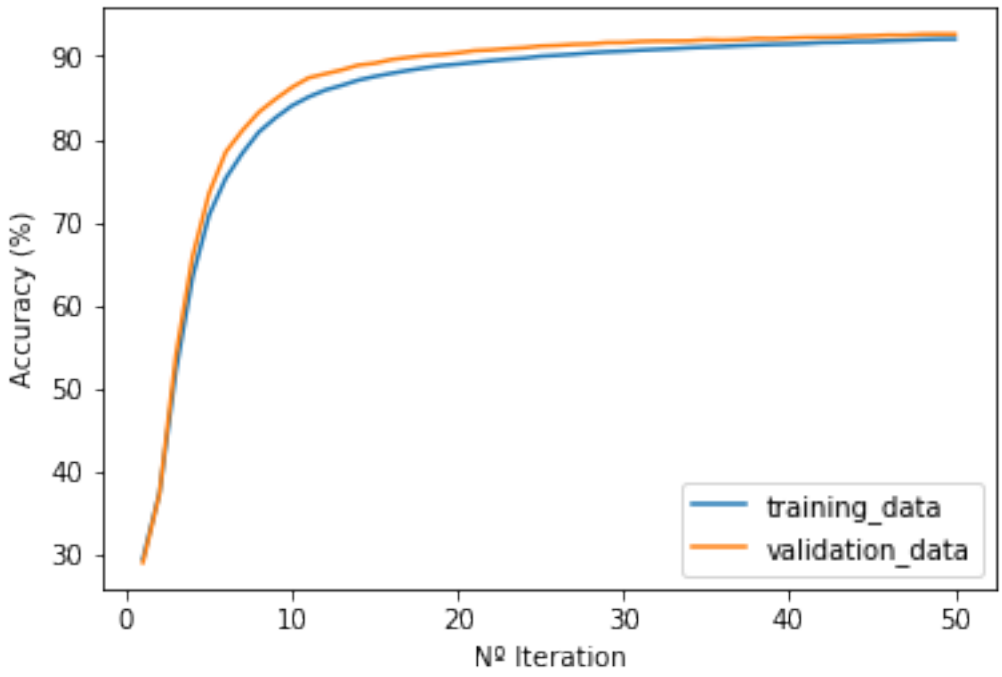
Change the learning rate (0.001, 0.1, 1.0 , 10). Train the new neural nets with the original specifications (Part 2.1), for 50 iterations. Plot test accuracy vs iteration for each learning rate on the same graph. Report the maximum test accuracy achieved for each learning rate. Which one achieves the maximum test accuracy?

```
In [25]: # Implementation with a learning rate of 0.001
         my_mnist_net2 = nn.NNet(n_input=784,
                                 netDims=[30,10],
                                 n_iter=50,
                                 learn=0.001)
```

```
In [14]: Tr_acc2, Val_acc2 = my_mnist_net2.train(training_data,
                                                 validation_data,
                                                 True, True, True)
         my_mnist_net2.save("mnist_nD30-10_it50_l0001.model")
```

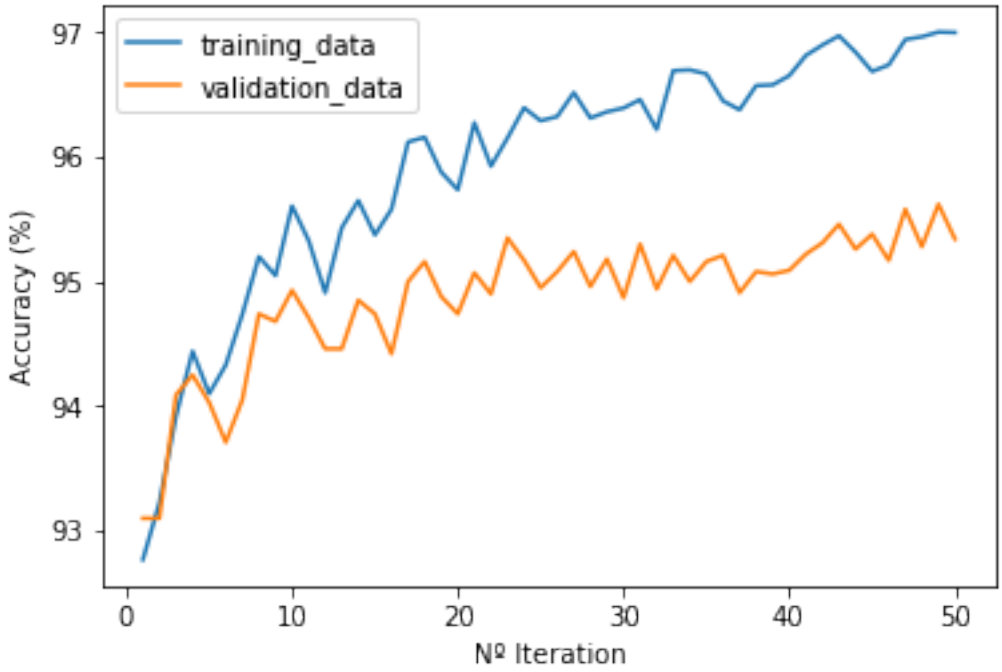
```
iter:  1/50 --> E: 0.5750299447  -Training_Accuracy:  29.38  -t: 8.69
iter:  2/50 --> E: 0.4163030463  -Training_Accuracy:  37.31  -t: 17.25
iter:  3/50 --> E: 0.3739675123  -Training_Accuracy:  52.12  -t: 25.61
iter:  4/50 --> E: 0.3310268428  -Training_Accuracy:  63.39  -t: 34.01
iter:  5/50 --> E: 0.2919505869  -Training_Accuracy:  70.87  -t: 42.37
iter:  6/50 --> E: 0.2574201758  -Training_Accuracy:  75.28  -t: 50.59
iter:  7/50 --> E: 0.2282355101  -Training_Accuracy:  78.30  -t: 59.02
iter:  8/50 --> E: 0.2046967249  -Training_Accuracy:  80.86  -t: 67.44
iter:  9/50 --> E: 0.1862060003  -Training_Accuracy:  82.57  -t: 75.84
iter: 10/50 --> E: 0.1714471452  -Training_Accuracy:  84.03  -t: 84.30
iter: 11/50 --> E: 0.1593253771  -Training_Accuracy:  85.09  -t: 92.69
iter: 12/50 --> E: 0.1492450400  -Training_Accuracy:  85.91  -t: 101.25
iter: 13/50 --> E: 0.1407860433  -Training_Accuracy:  86.48  -t: 109.67
iter: 14/50 --> E: 0.1336292585  -Training_Accuracy:  87.11  -t: 118.23
iter: 15/50 --> E: 0.1274952849  -Training_Accuracy:  87.55  -t: 127.03
iter: 16/50 --> E: 0.1222216991  -Training_Accuracy:  87.93  -t: 135.47
iter: 17/50 --> E: 0.1176152156  -Training_Accuracy:  88.28  -t: 143.83
iter: 18/50 --> E: 0.1135964364  -Training_Accuracy:  88.59  -t: 152.48
iter: 19/50 --> E: 0.1100456767  -Training_Accuracy:  88.86  -t: 161.06
iter: 20/50 --> E: 0.1068850359  -Training_Accuracy:  89.04  -t: 169.49
iter: 21/50 --> E: 0.1040450365  -Training_Accuracy:  89.25  -t: 177.97
iter: 22/50 --> E: 0.1014824544  -Training_Accuracy:  89.45  -t: 186.63
iter: 23/50 --> E: 0.0991614842  -Training_Accuracy:  89.63  -t: 195.38
iter: 24/50 --> E: 0.0970478546  -Training_Accuracy:  89.76  -t: 203.88
iter: 25/50 --> E: 0.0950956476  -Training_Accuracy:  89.99  -t: 212.48
iter: 26/50 --> E: 0.0933126571  -Training_Accuracy:  90.10  -t: 221.04
iter: 27/50 --> E: 0.0916589667  -Training_Accuracy:  90.23  -t: 229.64
```

```
iter: 28/50 --> E: 0.0901116597  -Training_Accuracy:  90.41  -t: 238.36
iter: 29/50 --> E: 0.0886814974  -Training_Accuracy:  90.52  -t: 247.00
iter: 30/50 --> E: 0.0873384869  -Training_Accuracy:  90.61  -t: 255.31
iter: 31/50 --> E: 0.0860814690  -Training_Accuracy:  90.72  -t: 263.81
iter: 32/50 --> E: 0.0849021891  -Training_Accuracy:  90.81  -t: 272.53
iter: 33/50 --> E: 0.0837814620  -Training_Accuracy:  90.90  -t: 281.24
iter: 34/50 --> E: 0.0827232309  -Training_Accuracy:  91.01  -t: 289.58
iter: 35/50 --> E: 0.0817270600  -Training_Accuracy:  91.09  -t: 297.90
iter: 36/50 --> E: 0.0807839389  -Training_Accuracy:  91.16  -t: 306.28
iter: 37/50 --> E: 0.0798780551  -Training_Accuracy:  91.27  -t: 314.75
iter: 38/50 --> E: 0.0790253457  -Training_Accuracy:  91.34  -t: 323.02
iter: 39/50 --> E: 0.0782028859  -Training_Accuracy:  91.41  -t: 331.25
iter: 40/50 --> E: 0.0774324525  -Training_Accuracy:  91.45  -t: 339.73
iter: 41/50 --> E: 0.0766811706  -Training_Accuracy:  91.53  -t: 348.24
iter: 42/50 --> E: 0.0759576694  -Training_Accuracy:  91.64  -t: 356.55
iter: 43/50 --> E: 0.0752914150  -Training_Accuracy:  91.67  -t: 364.94
iter: 44/50 --> E: 0.0746153629  -Training_Accuracy:  91.75  -t: 373.11
iter: 45/50 --> E: 0.0739705916  -Training_Accuracy:  91.76  -t: 381.50
iter: 46/50 --> E: 0.0733857698  -Training_Accuracy:  91.87  -t: 390.12
iter: 47/50 --> E: 0.0728001400  -Training_Accuracy:  91.92  -t: 398.40
iter: 48/50 --> E: 0.0722338628  -Training_Accuracy:  92.00  -t: 406.81
iter: 49/50 --> E: 0.0716804774  -Training_Accuracy:  92.05  -t: 415.15
iter: 50/50 --> E: 0.0711421346  -Training_Accuracy:  92.07  -t: 423.80
```



```
In [26]: # Implementation with a learning rate of 1.0
         my_mnist_net4 = nn.NNet(n_input=784,
                                 netDims=[30,10],
                                 n_iter=50,
                                 learn=1)

In [15]: Tr_acc4, Val_acc4 = my_mnist_net4.train(training_data,
                                                 validation_data,
                                                 True, True, True)
         my_mnist_net4.save("mnist_nD30-10_it50_l1.model")

iter:  1/50 --> E: 0.0869943052  -Training_Accuracy:  92.77  -t: 8.52
iter:  2/50 --> E: 0.0616437472  -Training_Accuracy:  93.23  -t: 16.69
iter:  3/50 --> E: 0.0566070897  -Training_Accuracy:  93.92  -t: 25.10
iter:  4/50 --> E: 0.0529684859  -Training_Accuracy:  94.44  -t: 33.57
iter:  5/50 --> E: 0.0495543964  -Training_Accuracy:  94.10  -t: 42.33
iter:  6/50 --> E: 0.0486023943  -Training_Accuracy:  94.33  -t: 50.75
iter:  7/50 --> E: 0.0476317300  -Training_Accuracy:  94.74  -t: 59.28
iter:  8/50 --> E: 0.0464348046  -Training_Accuracy:  95.20  -t: 68.18
iter:  9/50 --> E: 0.0448266871  -Training_Accuracy:  95.05  -t: 76.87
iter: 10/50 --> E: 0.0444483969  -Training_Accuracy:  95.61  -t: 85.31
iter: 11/50 --> E: 0.0423387844  -Training_Accuracy:  95.33  -t: 93.63
iter: 12/50 --> E: 0.0414947883  -Training_Accuracy:  94.91  -t: 102.18
iter: 13/50 --> E: 0.0409368524  -Training_Accuracy:  95.43  -t: 110.68
iter: 14/50 --> E: 0.0406828149  -Training_Accuracy:  95.65  -t: 119.19
iter: 15/50 --> E: 0.0396263533  -Training_Accuracy:  95.37  -t: 128.09
iter: 16/50 --> E: 0.0390285316  -Training_Accuracy:  95.58  -t: 136.64
iter: 17/50 --> E: 0.0391683163  -Training_Accuracy:  96.12  -t: 144.82
iter: 18/50 --> E: 0.0373204663  -Training_Accuracy:  96.16  -t: 153.24
iter: 19/50 --> E: 0.0385252333  -Training_Accuracy:  95.88  -t: 162.24
iter: 20/50 --> E: 0.0370650365  -Training_Accuracy:  95.74  -t: 172.37
iter: 21/50 --> E: 0.0365606420  -Training_Accuracy:  96.27  -t: 181.73
iter: 22/50 --> E: 0.0352071166  -Training_Accuracy:  95.93  -t: 190.12
iter: 23/50 --> E: 0.0349661933  -Training_Accuracy:  96.15  -t: 198.52
iter: 24/50 --> E: 0.0351279882  -Training_Accuracy:  96.39  -t: 207.07
iter: 25/50 --> E: 0.0346201246  -Training_Accuracy:  96.29  -t: 215.42
iter: 26/50 --> E: 0.0335173137  -Training_Accuracy:  96.32  -t: 223.72
iter: 27/50 --> E: 0.0332820651  -Training_Accuracy:  96.52  -t: 232.08
iter: 28/50 --> E: 0.0328952733  -Training_Accuracy:  96.31  -t: 240.45
iter: 29/50 --> E: 0.0332991684  -Training_Accuracy:  96.36  -t: 249.05
iter: 30/50 --> E: 0.0328406520  -Training_Accuracy:  96.39  -t: 257.87
iter: 31/50 --> E: 0.0320786491  -Training_Accuracy:  96.46  -t: 266.34
iter: 32/50 --> E: 0.0323606912  -Training_Accuracy:  96.22  -t: 274.86
iter: 33/50 --> E: 0.0329196417  -Training_Accuracy:  96.69  -t: 283.51
iter: 34/50 --> E: 0.0321592196  -Training_Accuracy:  96.70  -t: 292.08
iter: 35/50 --> E: 0.0312963616  -Training_Accuracy:  96.67  -t: 300.99
iter: 36/50 --> E: 0.0328466620  -Training_Accuracy:  96.45  -t: 309.83
iter: 37/50 --> E: 0.0320993295  -Training_Accuracy:  96.38  -t: 318.73
```

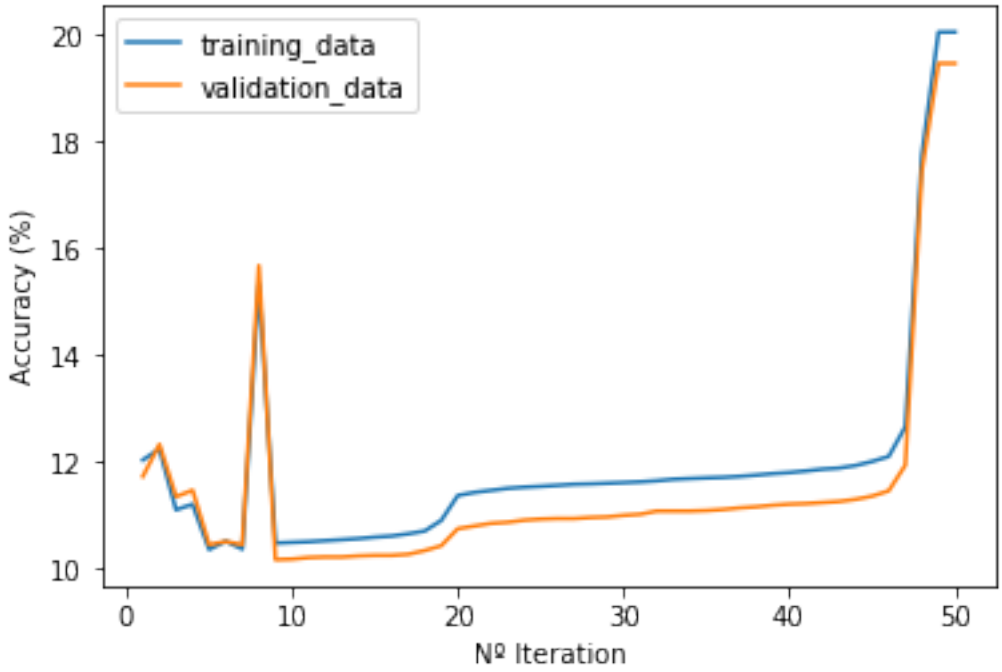
```
iter: 38/50 --> E: 0.0312184469  -Training_Accuracy:  96.57  -t: 327.19
iter: 39/50 --> E: 0.0307414249  -Training_Accuracy:  96.58  -t: 335.81
iter: 40/50 --> E: 0.0310736935  -Training_Accuracy:  96.65  -t: 344.34
iter: 41/50 --> E: 0.0297107040  -Training_Accuracy:  96.81  -t: 352.65
iter: 42/50 --> E: 0.0309107600  -Training_Accuracy:  96.90  -t: 361.20
iter: 43/50 --> E: 0.0295105135  -Training_Accuracy:  96.97  -t: 369.80
iter: 44/50 --> E: 0.0288079727  -Training_Accuracy:  96.84  -t: 378.37
iter: 45/50 --> E: 0.0293670171  -Training_Accuracy:  96.69  -t: 387.00
iter: 46/50 --> E: 0.0300072902  -Training_Accuracy:  96.74  -t: 395.51
iter: 47/50 --> E: 0.0297783560  -Training_Accuracy:  96.94  -t: 404.11
iter: 48/50 --> E: 0.0287263193  -Training_Accuracy:  96.96  -t: 412.76
iter: 49/50 --> E: 0.0281946280  -Training_Accuracy:  97.00  -t: 421.34
iter: 50/50 --> E: 0.0283231795  -Training_Accuracy:  97.00  -t: 429.77
```




```
In [27]: # Implementation with a learning rate of 10
         my_mnist_net5 = nn.NNet(n_input=784,
                                 netDims=[30,10],
                                 n_iter=50,
                                 learn=10)

In [16]: Tr_acc5, Val_acc5 = my_mnist_net5.train(training_data,
                                                 validation_data,
                                                 True, True, True)
         my_mnist_net5.save("mnist_nD30-10_it50_l10.model")
```

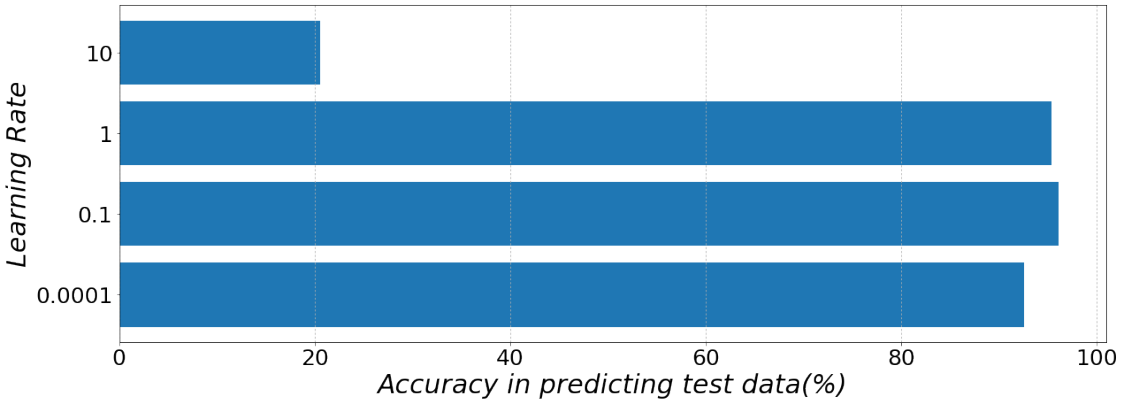
```
C:\Users\AlbertoIbarrondo\Documents\DeepLearning\AlbertoTopVersion\transfer_functio
  def sigmoid(x):  return 1 / (1 + np.exp(-x))


iter:  1/50 --> E: 0.5277766907  -Training_Accuracy:  12.04  -t: 8.45
iter:  2/50 --> E: 0.5001586211  -Training_Accuracy:  12.24  -t: 16.75
iter:  3/50 --> E: 0.4999279853  -Training_Accuracy:  11.10  -t: 25.16
iter:  4/50 --> E: 0.4999997894  -Training_Accuracy:  11.21  -t: 33.47
iter:  5/50 --> E: 0.4999999096  -Training_Accuracy:  10.36  -t: 41.98
iter:  6/50 --> E: 0.5000322736  -Training_Accuracy:  10.52  -t: 50.74
iter:  7/50 --> E: 0.4999996628  -Training_Accuracy:  10.37  -t: 60.13
iter:  8/50 --> E: 0.5000021289  -Training_Accuracy:  15.33  -t: 68.76
iter:  9/50 --> E: 0.5000208514  -Training_Accuracy:  10.48  -t: 77.24
iter: 10/50 --> E: 0.4999999882  -Training_Accuracy:  10.49  -t: 86.00
iter: 11/50 --> E: 0.4999999880  -Training_Accuracy:  10.51  -t: 94.36
iter: 12/50 --> E: 0.4999999878  -Training_Accuracy:  10.53  -t: 102.79
iter: 13/50 --> E: 0.4999999876  -Training_Accuracy:  10.54  -t: 111.06
iter: 14/50 --> E: 0.4999999874  -Training_Accuracy:  10.56  -t: 119.43
iter: 15/50 --> E: 0.4999999872  -Training_Accuracy:  10.59  -t: 127.65
iter: 16/50 --> E: 0.4999999869  -Training_Accuracy:  10.61  -t: 135.86
iter: 17/50 --> E: 0.4999999867  -Training_Accuracy:  10.65  -t: 144.23
iter: 18/50 --> E: 0.4999999864  -Training_Accuracy:  10.71  -t: 152.64
iter: 19/50 --> E: 0.4999999861  -Training_Accuracy:  10.90  -t: 161.12
iter: 20/50 --> E: 0.4999999858  -Training_Accuracy:  11.37  -t: 169.77
iter: 21/50 --> E: 0.4999999855  -Training_Accuracy:  11.43  -t: 178.44
iter: 22/50 --> E: 0.4999999851  -Training_Accuracy:  11.47  -t: 187.01
iter: 23/50 --> E: 0.4999999848  -Training_Accuracy:  11.51  -t: 195.57
iter: 24/50 --> E: 0.4999999844  -Training_Accuracy:  11.53  -t: 203.95
iter: 25/50 --> E: 0.4999999840  -Training_Accuracy:  11.54  -t: 212.29
iter: 26/50 --> E: 0.4999999835  -Training_Accuracy:  11.56  -t: 220.70
iter: 27/50 --> E: 0.4999999831  -Training_Accuracy:  11.58  -t: 229.10
iter: 28/50 --> E: 0.4999999825  -Training_Accuracy:  11.59  -t: 237.50
iter: 29/50 --> E: 0.4999999820  -Training_Accuracy:  11.60  -t: 245.85
iter: 30/50 --> E: 0.4999999813  -Training_Accuracy:  11.61  -t: 254.25
iter: 31/50 --> E: 0.4999999807  -Training_Accuracy:  11.63  -t: 262.64
iter: 32/50 --> E: 0.4999999799  -Training_Accuracy:  11.65  -t: 271.13
iter: 33/50 --> E: 0.4999999791  -Training_Accuracy:  11.68  -t: 279.50
iter: 34/50 --> E: 0.4999999781  -Training_Accuracy:  11.69  -t: 288.52
iter: 35/50 --> E: 0.4999999770  -Training_Accuracy:  11.70  -t: 298.45
iter: 36/50 --> E: 0.4999999758  -Training_Accuracy:  11.71  -t: 307.85
iter: 37/50 --> E: 0.4999999744  -Training_Accuracy:  11.73  -t: 316.51
iter: 38/50 --> E: 0.4999999727  -Training_Accuracy:  11.75  -t: 325.02
iter: 39/50 --> E: 0.4999999708  -Training_Accuracy:  11.78  -t: 333.68
iter: 40/50 --> E: 0.4999999683  -Training_Accuracy:  11.80  -t: 342.27
iter: 41/50 --> E: 0.4999999653  -Training_Accuracy:  11.83  -t: 350.86
iter: 42/50 --> E: 0.4999999613  -Training_Accuracy:  11.87  -t: 359.48
iter: 43/50 --> E: 0.4999999560  -Training_Accuracy:  11.89  -t: 368.05
iter: 44/50 --> E: 0.4999999484  -Training_Accuracy:  11.93  -t: 376.67
```

```
iter: 45/50 --> E: 0.4999999363  -Training_Accuracy:  12.01  -t: 385.31
iter: 46/50 --> E: 0.4999999144  -Training_Accuracy:  12.10  -t: 393.76
iter: 47/50 --> E: 0.4999998601  -Training_Accuracy:  12.66  -t: 402.41
iter: 48/50 --> E: 0.5000640842  -Training_Accuracy:  17.76  -t: 410.95
iter: 49/50 --> E: 0.4988409765  -Training_Accuracy:  20.06  -t: 419.55
iter: 50/50 --> E: 0.4999999801  -Training_Accuracy:  20.06  -t: 428.16
```




```
In [36]: my_mnist_net.load('mnist_nD30-10_it20_l01.model')
         my_mnist_net2.load("mnist_nD30-10_it50_l0001.model")
         my_mnist_net4.load("mnist_nD30-10_it50_l1.model")
         my_mnist_net5.load("mnist_nD30-10_it50_l10.model")

In [92]: learn_rates = [0.0001, 0.1, 1, 10]
         x_range = [1,2,3,4]
         accuracies = [0,0,0,0]
         accuracies[0] = my_mnist_net2.predict(test_data)/len(test_data)*100
         accuracies[1] = my_mnist_net.predict(test_data)/len(test_data)*100
         accuracies[2] = my_mnist_net4.predict(test_data)/len(test_data)*100
         accuracies[3] = my_mnist_net5.predict(test_data)/len(test_data)*100

         plt.figure(figsize=[20,7])
         plt.barh(x_range, accuracies)
         plt.yticks(x_range, learn_rates, fontsize=25)
         plt.xticks(fontsize=25)
```

```
plt.ylabel('Learning Rate', fontsize=30, fontstyle='oblique')
plt.xlabel('Accuracy in predicting test data(%)',
           fontsize=30,
           fontstyle='oblique')
plt.grid(True, axis='x', ls=':')
plt.show()
print('Accuracy of lr=0.0001 on test data: ', accuracies[0], '%')
print('Accuracy of lr=0.1 on test data: ', accuracies[1], '%')
print('Accuracy of lr=1 on test data: ', accuracies[2], '%')
print('Accuracy of lr=10 on test data: ', accuracies[3], '%')
```




```
Accuracy of lr=0.0001 on test data:  92.56 %
Accuracy of lr=0.1 on test data:  96.12 %
Accuracy of lr=1 on test data:  95.38 %
Accuracy of lr=10 on test data:  20.59 %
```

COMMENT: Overall, the best learning rate is 0.1. With 0.001 the learning curve steadily increases, but it's significantly when compared to 0.1. Also, clearly the learning rate of 10 is completely out of boundaries, not achieving almost any improvement at all. Comparing the accuracies in the prediction of test data, the best one is LEARNING RATE = 0.1

3.3.2 Question 2.2.2

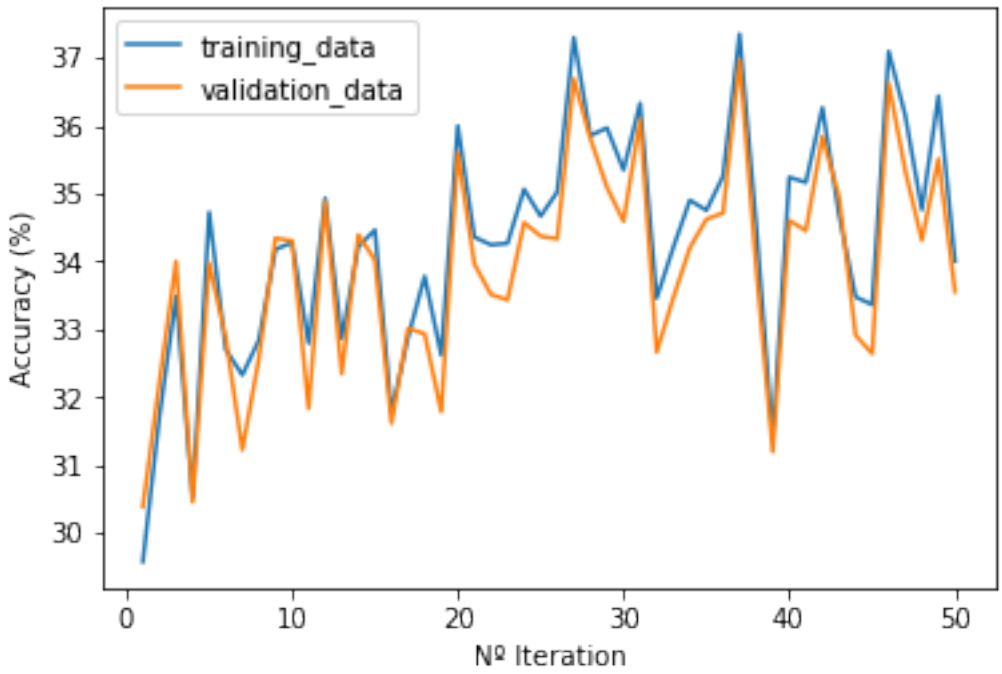
initialize all weights to 0. Plot the training accuracy curve. Comment your results

```
In [ ]: # NNet with weights to 0
        my_mnist_net_all0 = nn.NNet(n_input=784,
                                    netDims=[30,10],
                                    n_iter=50,
                                    learn=0.1)
```

```
In [63]: W0_zeros = np.zeros([785,30])
         W1_zeros = np.zeros([31,10])
         my_mnist_net_all0.init_w([W0_zeros, W1_zeros])
         Tr_acc0, Val_acc0 = my_mnist_net_all0.train(training_data,
                                                     validation_data,
                                                     True, True, True)
         my_mnist_net_all0.save("mnistZeros_nD30-10_it50_l01.model")
```

```
iter:  1/50 --> E: 0.4111486307  -Training_Accuracy:  29.57  -t: 8.42
iter:  2/50 --> E: 0.3716331103  -Training_Accuracy:  31.71  -t: 16.61
iter:  3/50 --> E: 0.3700833178  -Training_Accuracy:  33.49  -t: 24.76
iter:  4/50 --> E: 0.3693742264  -Training_Accuracy:  30.52  -t: 33.18
iter:  5/50 --> E: 0.3688787600  -Training_Accuracy:  34.72  -t: 41.34
iter:  6/50 --> E: 0.3687365868  -Training_Accuracy:  32.70  -t: 49.52
iter:  7/50 --> E: 0.3685479100  -Training_Accuracy:  32.32  -t: 57.89
iter:  8/50 --> E: 0.3683705919  -Training_Accuracy:  32.84  -t: 66.24
iter:  9/50 --> E: 0.3682131519  -Training_Accuracy:  34.17  -t: 74.39
iter: 10/50 --> E: 0.3681069767  -Training_Accuracy:  34.28  -t: 82.91
iter: 11/50 --> E: 0.3679885920  -Training_Accuracy:  32.79  -t: 91.09
iter: 12/50 --> E: 0.3678892802  -Training_Accuracy:  34.93  -t: 99.91
iter: 13/50 --> E: 0.3678665387  -Training_Accuracy:  32.86  -t: 108.22
iter: 14/50 --> E: 0.3677965832  -Training_Accuracy:  34.22  -t: 116.52
iter: 15/50 --> E: 0.3677304508  -Training_Accuracy:  34.46  -t: 124.73
iter: 16/50 --> E: 0.3675962711  -Training_Accuracy:  31.77  -t: 133.23
iter: 17/50 --> E: 0.3675672035  -Training_Accuracy:  32.88  -t: 141.45
iter: 18/50 --> E: 0.3675086535  -Training_Accuracy:  33.78  -t: 149.74
iter: 19/50 --> E: 0.3675421576  -Training_Accuracy:  32.62  -t: 157.97
iter: 20/50 --> E: 0.3674933173  -Training_Accuracy:  36.00  -t: 166.20
iter: 21/50 --> E: 0.3672942227  -Training_Accuracy:  34.36  -t: 174.34
iter: 22/50 --> E: 0.3672899530  -Training_Accuracy:  34.24  -t: 182.62
iter: 23/50 --> E: 0.3672985773  -Training_Accuracy:  34.26  -t: 190.86
iter: 24/50 --> E: 0.3672241324  -Training_Accuracy:  35.06  -t: 199.09
iter: 25/50 --> E: 0.3671269861  -Training_Accuracy:  34.66  -t: 207.55
iter: 26/50 --> E: 0.3670427599  -Training_Accuracy:  35.02  -t: 215.83
iter: 27/50 --> E: 0.3670719115  -Training_Accuracy:  37.30  -t: 224.14
iter: 28/50 --> E: 0.3670168598  -Training_Accuracy:  35.85  -t: 232.28
iter: 29/50 --> E: 0.3670403598  -Training_Accuracy:  35.97  -t: 240.32
iter: 30/50 --> E: 0.3669704041  -Training_Accuracy:  35.35  -t: 248.38
iter: 31/50 --> E: 0.3668792304  -Training_Accuracy:  36.33  -t: 256.39
iter: 32/50 --> E: 0.3669023938  -Training_Accuracy:  33.46  -t: 264.45
iter: 33/50 --> E: 0.3669138532  -Training_Accuracy:  34.20  -t: 272.46
iter: 34/50 --> E: 0.3667756691  -Training_Accuracy:  34.90  -t: 280.66
iter: 35/50 --> E: 0.3667633007  -Training_Accuracy:  34.75  -t: 288.92
iter: 36/50 --> E: 0.3667044275  -Training_Accuracy:  35.25  -t: 297.11
iter: 37/50 --> E: 0.3667793977  -Training_Accuracy:  37.35  -t: 305.26
iter: 38/50 --> E: 0.3667288536  -Training_Accuracy:  34.55  -t: 313.39
iter: 39/50 --> E: 0.3666745158  -Training_Accuracy:  31.40  -t: 321.61
iter: 40/50 --> E: 0.3666421364  -Training_Accuracy:  35.24  -t: 329.87
```

```
iter: 41/50 --> E: 0.3665789366  -Training_Accuracy:  35.16  -t: 338.03
iter: 42/50 --> E: 0.3665869277  -Training_Accuracy:  36.26  -t: 346.01
iter: 43/50 --> E: 0.3665309900  -Training_Accuracy:  34.69  -t: 354.05
iter: 44/50 --> E: 0.3665450500  -Training_Accuracy:  33.47  -t: 362.33
iter: 45/50 --> E: 0.3664494992  -Training_Accuracy:  33.36  -t: 370.62
iter: 46/50 --> E: 0.3664788861  -Training_Accuracy:  37.09  -t: 379.21
iter: 47/50 --> E: 0.3664096826  -Training_Accuracy:  36.15  -t: 387.54
iter: 48/50 --> E: 0.3664528778  -Training_Accuracy:  34.76  -t: 395.80
iter: 49/50 --> E: 0.3664050886  -Training_Accuracy:  36.44  -t: 404.38
iter: 50/50 --> E: 0.3663732688  -Training_Accuracy:  34.00  -t: 412.63
```



Comment Since al weights are symmetric, even when using the best learning rate (0.1), there is almost no improvement since there is no learning.

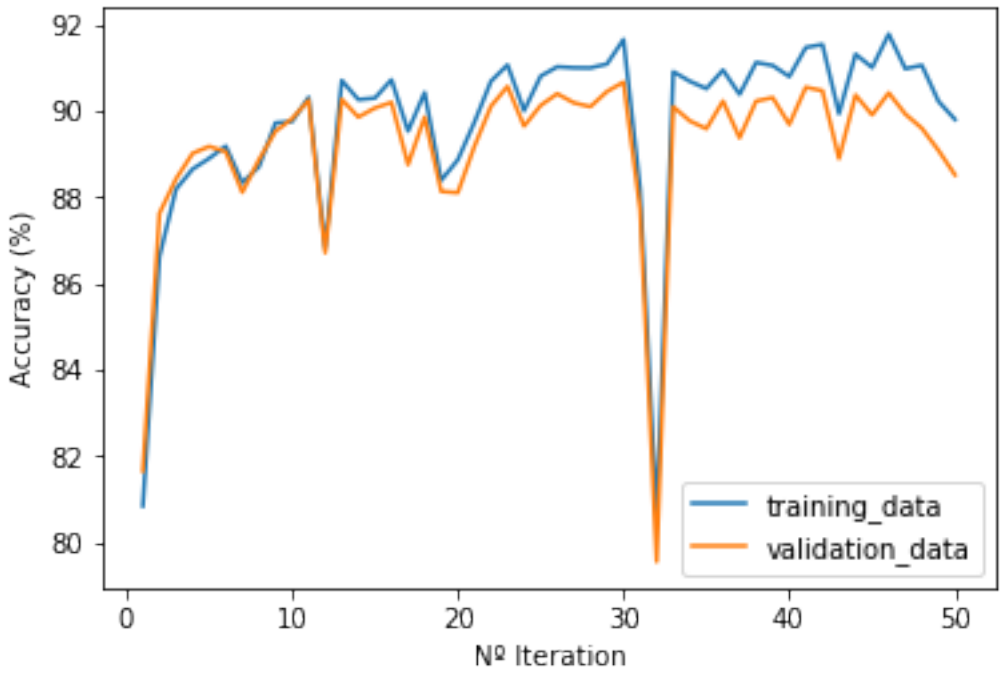
3.3.3 Question 2.2.3

Try with a different transfer function (such as tanh).

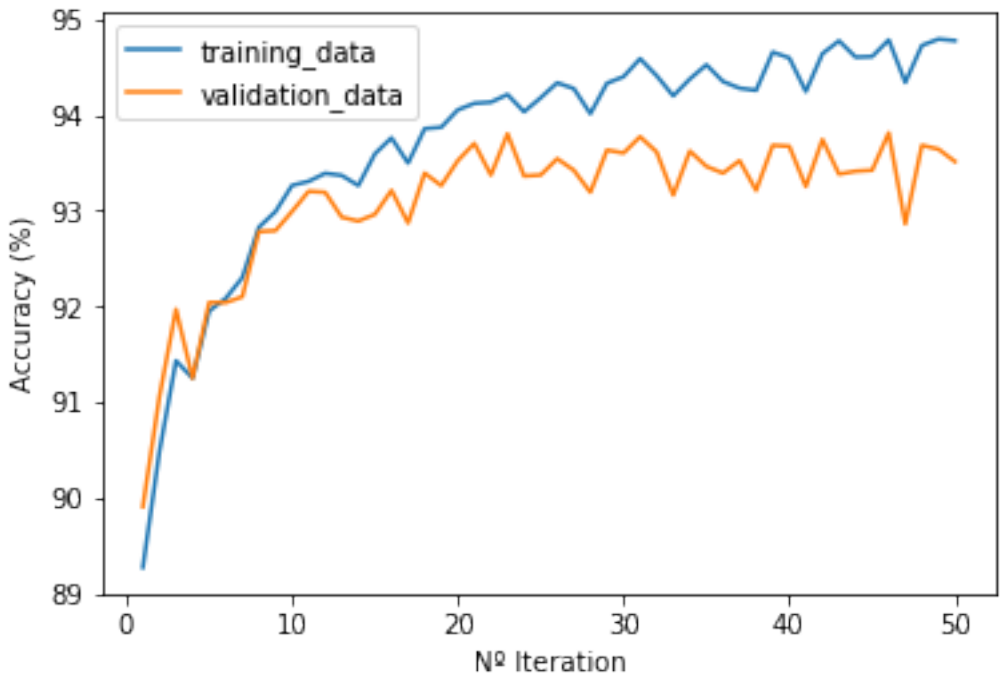
```
In [ ]: #NNet with tanh as transfer function
        my_mnist_net_tanh = nn.NNet(n_input=784,
                                    netDims=[30,10],
                                    n_iter=50,
```

```
                                        learn=0.1,
                                        tf=tf.tanh,
                                        dtf=tf.dtanh)

In [17]: Tr_accTanh, Val_accTanh = my_mnist_net_tanh.train(training_data,
                                                           validation_data,
                                                           True, True, True)
         my_mnist_net_tanh.save("mnistTanh_nD30-10_it50_l01.model")

iter:  1/50 --> E: 0.6551802134  -Training_Accuracy:  80.85  -t: 7.55
iter:  2/50 --> E: 0.3187645623  -Training_Accuracy:  86.62  -t: 15.06
iter:  3/50 --> E: 0.2254234698  -Training_Accuracy:  88.18  -t: 22.47
iter:  4/50 --> E: 0.2008916854  -Training_Accuracy:  88.65  -t: 29.82
iter:  5/50 --> E: 0.1886025766  -Training_Accuracy:  88.89  -t: 37.19
iter:  6/50 --> E: 0.1798545705  -Training_Accuracy:  89.18  -t: 44.50
iter:  7/50 --> E: 0.1751538077  -Training_Accuracy:  88.33  -t: 51.87
iter:  8/50 --> E: 0.1717451991  -Training_Accuracy:  88.70  -t: 59.26
iter:  9/50 --> E: 0.1651267158  -Training_Accuracy:  89.71  -t: 66.59
iter: 10/50 --> E: 0.1631927344  -Training_Accuracy:  89.75  -t: 73.98
iter: 11/50 --> E: 0.1603812964  -Training_Accuracy:  90.31  -t: 81.89
iter: 12/50 --> E: 0.1578629626  -Training_Accuracy:  86.83  -t: 89.59
iter: 13/50 --> E: 0.1561781881  -Training_Accuracy:  90.70  -t: 97.16
iter: 14/50 --> E: 0.1585662188  -Training_Accuracy:  90.25  -t: 104.66
iter: 15/50 --> E: 0.1550276907  -Training_Accuracy:  90.29  -t: 112.17
iter: 16/50 --> E: 0.1540838458  -Training_Accuracy:  90.71  -t: 119.59
iter: 17/50 --> E: 0.1526670973  -Training_Accuracy:  89.53  -t: 127.09
iter: 18/50 --> E: 0.1536943702  -Training_Accuracy:  90.42  -t: 135.03
iter: 19/50 --> E: 0.1506909296  -Training_Accuracy:  88.39  -t: 142.91
iter: 20/50 --> E: 0.1511284857  -Training_Accuracy:  88.86  -t: 150.49
iter: 21/50 --> E: 0.1487472069  -Training_Accuracy:  89.73  -t: 157.99
iter: 22/50 --> E: 0.1490044880  -Training_Accuracy:  90.68  -t: 165.52
iter: 23/50 --> E: 0.1471987331  -Training_Accuracy:  91.06  -t: 172.97
iter: 24/50 --> E: 0.1465759223  -Training_Accuracy:  90.00  -t: 180.31
iter: 25/50 --> E: 0.1469047242  -Training_Accuracy:  90.80  -t: 187.63
iter: 26/50 --> E: 0.1475381949  -Training_Accuracy:  91.01  -t: 195.00
iter: 27/50 --> E: 0.1461873132  -Training_Accuracy:  90.99  -t: 202.59
iter: 28/50 --> E: 0.1462962581  -Training_Accuracy:  90.99  -t: 210.40
iter: 29/50 --> E: 0.1455122107  -Training_Accuracy:  91.08  -t: 218.58
iter: 30/50 --> E: 0.1442139806  -Training_Accuracy:  91.65  -t: 226.79
iter: 31/50 --> E: 0.1430524037  -Training_Accuracy:  88.19  -t: 234.56
iter: 32/50 --> E: 0.1419424535  -Training_Accuracy:  80.62  -t: 242.05
iter: 33/50 --> E: 0.1422258316  -Training_Accuracy:  90.90  -t: 249.44
iter: 34/50 --> E: 0.1423085740  -Training_Accuracy:  90.68  -t: 257.09
iter: 35/50 --> E: 0.1423223483  -Training_Accuracy:  90.51  -t: 264.90
iter: 36/50 --> E: 0.1400285642  -Training_Accuracy:  90.94  -t: 272.38
iter: 37/50 --> E: 0.1396966826  -Training_Accuracy:  90.38  -t: 279.94
iter: 38/50 --> E: 0.1403713662  -Training_Accuracy:  91.11  -t: 288.00
iter: 39/50 --> E: 0.1409697874  -Training_Accuracy:  91.04  -t: 295.67
```

```
iter: 40/50 --> E: 0.1403633989  -Training_Accuracy:  90.79  -t: 303.35
iter: 41/50 --> E: 0.1393338393  -Training_Accuracy:  91.46  -t: 311.28
iter: 42/50 --> E: 0.1382701573  -Training_Accuracy:  91.53  -t: 318.89
iter: 43/50 --> E: 0.1381466756  -Training_Accuracy:  89.93  -t: 326.98
iter: 44/50 --> E: 0.1383648107  -Training_Accuracy:  91.31  -t: 334.59
iter: 45/50 --> E: 0.1377941233  -Training_Accuracy:  91.00  -t: 342.10
iter: 46/50 --> E: 0.1382886715  -Training_Accuracy:  91.77  -t: 349.60
iter: 47/50 --> E: 0.1375977170  -Training_Accuracy:  90.97  -t: 357.17
iter: 48/50 --> E: 0.1379894322  -Training_Accuracy:  91.05  -t: 364.59
iter: 49/50 --> E: 0.1369042682  -Training_Accuracy:  90.22  -t: 372.14
iter: 50/50 --> E: 0.1368395578  -Training_Accuracy:  89.79  -t: 379.72
```




```
In [78]: #NNet with tanh as transfer function
         my_mnist_net_tanh2 = nn.NNet(n_input=784,
                                      netDims=[30,10],
                                      n_iter=50,
                                      learn=0.01,     # CHANGED LEARNING RATE
                                      tf=tf.tanh,
                                      dtf=tf.dtanh)

In [79]: Tr_accTanh2, Val_accTanh2 = my_mnist_net_tanh2.train( training_data,
                                                               validation_data,
                                                               False, True, True)
         my_mnist_net_tanh2.save("mnistTanh2_nD30-10_it50_l01.model")
```




```
In [84]: print('Accuracy on test data for tanh, lr=0.01:  ' \
               , my_mnist_net_tanh2.predict(test_data)/len(test_data)*100, '%')

Accuracy on test data for tanh, lr=0.01:   93.15 %
```

Comments: It seems that not only does it achieve a lower final accuracy (~90%), but it oscillates periodically. After analyzing the result, we concluded that a different learning rate is needed in order to properly train the NNet with tanh. We rerun it with 0.01, obtaining better results (limiting oscillations).

despite our efforts, we can state that sigmoid function is better on this case (97% accuracy compared to the tanh's 93%)

3.3.4 Question 2.2.4

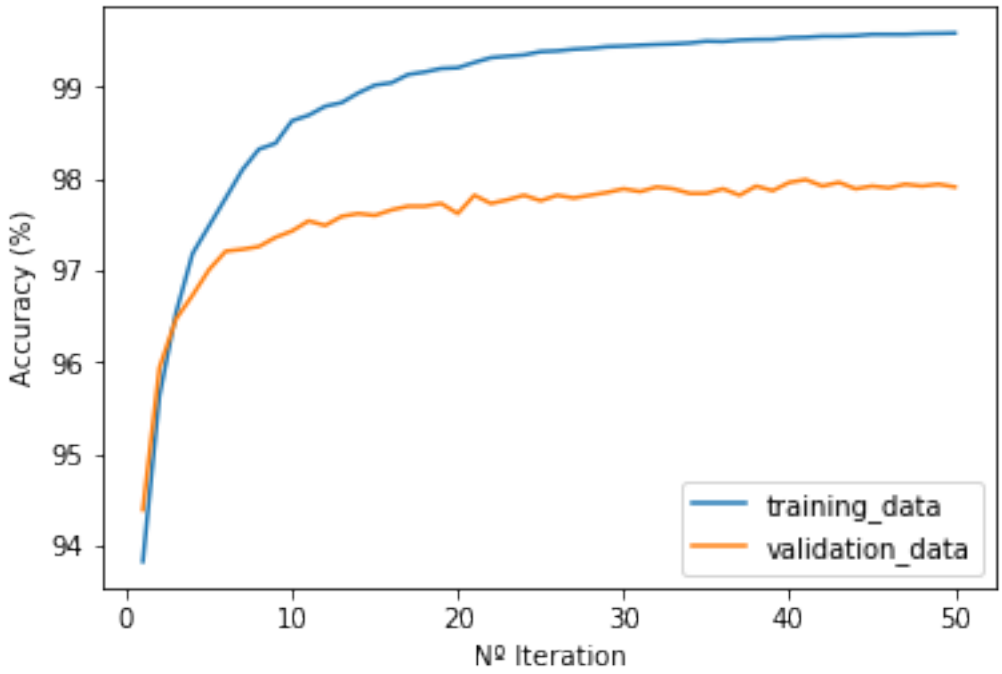
Add more neurons in the hidden layer (try with 100, 200, 300). Plot the curve representing the validation accuracy versus the number of neurons in the hidden layer.

```
In [64]: # 100 hidden neurons
         my_mnist_net_100 = nn.NNet(n_input=784,
                                    netDims=[100,10],
                                    n_iter=50,
                                    learn=0.1)
```

```
In [19]: Tr_acc100, Val_acc100 = my_mnist_net_100.train(training_data,
                                                        validation_data,
                                                        True, True, True)
         my_mnist_net_100.save("mnist_nD100-10_it50_l01.model")
```

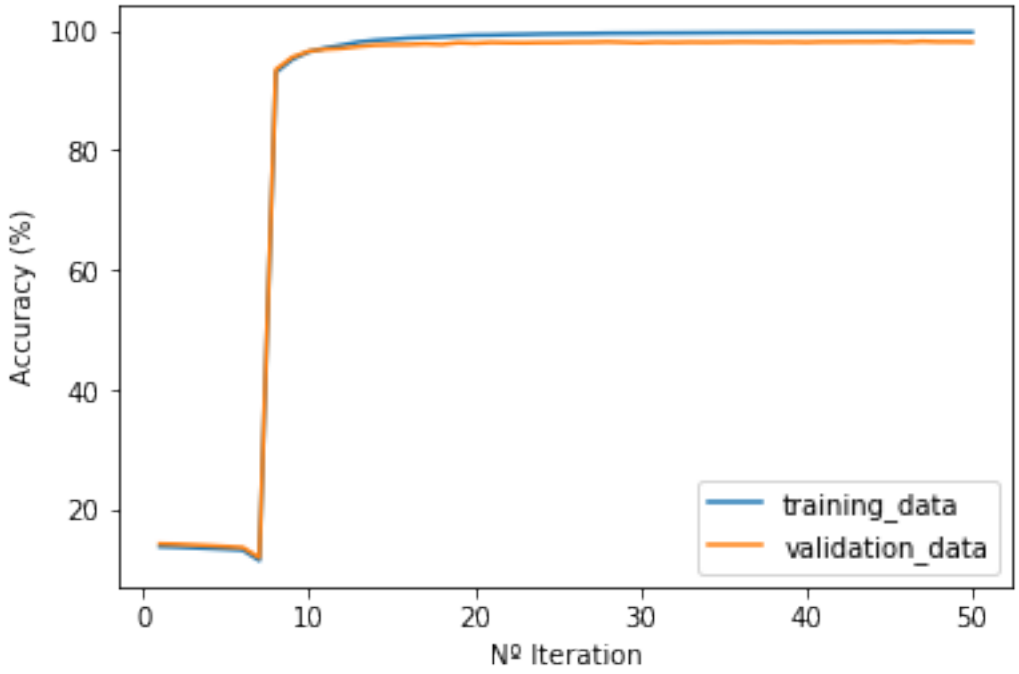
```
iter:  1/50 --> E: 0.1486648873  -Training_Accuracy:  93.83  -t: 46.37
iter:  2/50 --> E: 0.0481223547  -Training_Accuracy:  95.63  -t: 94.01
iter:  3/50 --> E: 0.0373463533  -Training_Accuracy:  96.54  -t: 150.01
iter:  4/50 --> E: 0.0311599441  -Training_Accuracy:  97.18  -t: 202.76
iter:  5/50 --> E: 0.0270190395  -Training_Accuracy:  97.49  -t: 256.61
iter:  6/50 --> E: 0.0238932706  -Training_Accuracy:  97.79  -t: 310.90
iter:  7/50 --> E: 0.0214591803  -Training_Accuracy:  98.10  -t: 365.59
iter:  8/50 --> E: 0.0193909499  -Training_Accuracy:  98.32  -t: 416.36
iter:  9/50 --> E: 0.0177294226  -Training_Accuracy:  98.39  -t: 458.78
iter: 10/50 --> E: 0.0164367169  -Training_Accuracy:  98.63  -t: 498.48
iter: 11/50 --> E: 0.0150771843  -Training_Accuracy:  98.69  -t: 536.39
iter: 12/50 --> E: 0.0139723445  -Training_Accuracy:  98.79  -t: 573.80
iter: 13/50 --> E: 0.0129469392  -Training_Accuracy:  98.83  -t: 611.77
iter: 14/50 --> E: 0.0120800595  -Training_Accuracy:  98.94  -t: 649.14
iter: 15/50 --> E: 0.0112620433  -Training_Accuracy:  99.02  -t: 686.08
iter: 16/50 --> E: 0.0106686530  -Training_Accuracy:  99.05  -t: 723.14
iter: 17/50 --> E: 0.0099812965  -Training_Accuracy:  99.14  -t: 760.11
iter: 18/50 --> E: 0.0094097889  -Training_Accuracy:  99.16  -t: 797.20
iter: 19/50 --> E: 0.0088456838  -Training_Accuracy:  99.20  -t: 834.20
iter: 20/50 --> E: 0.0083832652  -Training_Accuracy:  99.21  -t: 871.07
iter: 21/50 --> E: 0.0079313001  -Training_Accuracy:  99.27  -t: 908.05
iter: 22/50 --> E: 0.0074355455  -Training_Accuracy:  99.32  -t: 945.05
iter: 23/50 --> E: 0.0070681225  -Training_Accuracy:  99.34  -t: 982.04
iter: 24/50 --> E: 0.0068185048  -Training_Accuracy:  99.35  -t: 1018.88
iter: 25/50 --> E: 0.0064542181  -Training_Accuracy:  99.39  -t: 1055.76
iter: 26/50 --> E: 0.0061486633  -Training_Accuracy:  99.39  -t: 1092.60
iter: 27/50 --> E: 0.0058865883  -Training_Accuracy:  99.41  -t: 1129.92
iter: 28/50 --> E: 0.0056448655  -Training_Accuracy:  99.42  -t: 1166.74
iter: 29/50 --> E: 0.0053946926  -Training_Accuracy:  99.44  -t: 1203.66
iter: 30/50 --> E: 0.0051916567  -Training_Accuracy:  99.45  -t: 1240.55
iter: 31/50 --> E: 0.0050034353  -Training_Accuracy:  99.46  -t: 1280.83
iter: 32/50 --> E: 0.0048122344  -Training_Accuracy:  99.46  -t: 1319.52
iter: 33/50 --> E: 0.0046555936  -Training_Accuracy:  99.47  -t: 1357.36
iter: 34/50 --> E: 0.0044748580  -Training_Accuracy:  99.48  -t: 1394.38
iter: 35/50 --> E: 0.0043066606  -Training_Accuracy:  99.50  -t: 1431.67
iter: 36/50 --> E: 0.0042218592  -Training_Accuracy:  99.50  -t: 1470.14
iter: 37/50 --> E: 0.0040896211  -Training_Accuracy:  99.51  -t: 1508.69
iter: 38/50 --> E: 0.0040077006  -Training_Accuracy:  99.52  -t: 1546.27
iter: 39/50 --> E: 0.0038888060  -Training_Accuracy:  99.52  -t: 1583.44
iter: 40/50 --> E: 0.0037954427  -Training_Accuracy:  99.54  -t: 1625.37
iter: 41/50 --> E: 0.0037097560  -Training_Accuracy:  99.54  -t: 1668.39
iter: 42/50 --> E: 0.0036302499  -Training_Accuracy:  99.55  -t: 1706.24
iter: 43/50 --> E: 0.0035158455  -Training_Accuracy:  99.55  -t: 1752.10
```

```
iter: 44/50 --> E: 0.0034520999  -Training_Accuracy:  99.56  -t: 1793.07
iter: 45/50 --> E: 0.0033721018  -Training_Accuracy:  99.57  -t: 1837.70
iter: 46/50 --> E: 0.0033273868  -Training_Accuracy:  99.57  -t: 1878.48
iter: 47/50 --> E: 0.0032597245  -Training_Accuracy:  99.57  -t: 1916.24
iter: 48/50 --> E: 0.0032032391  -Training_Accuracy:  99.58  -t: 1955.88
iter: 49/50 --> E: 0.0031536687  -Training_Accuracy:  99.58  -t: 1993.84
iter: 50/50 --> E: 0.0030854841  -Training_Accuracy:  99.59  -t: 2032.86
```




```
In [65]: # 200 hidden neurons
         my_mnist_net_200 = nn.NNet(n_input=784,
                                    netDims=[200,10],
                                    n_iter=50,
                                    learn=0.1)

In [21]: Tr_acc200, Val_acc200 = my_mnist_net_200.train(training_data,
                                                        validation_data,
                                                        True, True, True)
         my_mnist_net_200.save("mnist_nD200-10_it50_l01.model")

iter:  1/50 --> E: 4.4999983929  -Training_Accuracy:  13.93  -t: 66.67
iter:  2/50 --> E: 4.4999981394  -Training_Accuracy:  13.85  -t: 135.88
iter:  3/50 --> E: 4.4999977801  -Training_Accuracy:  13.74  -t: 202.61
iter:  4/50 --> E: 4.4999972208  -Training_Accuracy:  13.62  -t: 271.98
iter:  5/50 --> E: 4.4999961881  -Training_Accuracy:  13.50  -t: 340.52
```

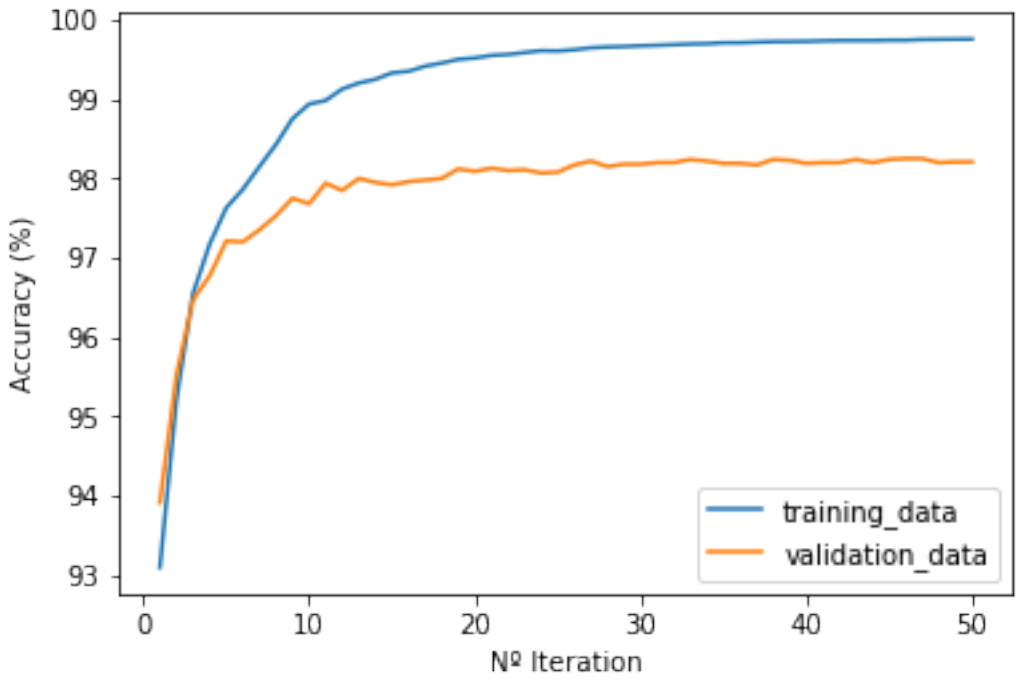
```
iter:  6/50 --> E: 4.4999931642  -Training_Accuracy:  13.36  -t: 409.19
iter:  7/50 --> E: 4.2696162814  -Training_Accuracy:  11.62  -t: 477.48
iter:  8/50 --> E: 0.8136454887  -Training_Accuracy:  92.97  -t: 544.22
iter:  9/50 --> E: 0.0509215721  -Training_Accuracy:  95.14  -t: 618.90
iter: 10/50 --> E: 0.0380593811  -Training_Accuracy:  96.40  -t: 687.48
iter: 11/50 --> E: 0.0311215563  -Training_Accuracy:  97.03  -t: 759.22
iter: 12/50 --> E: 0.0262554340  -Training_Accuracy:  97.50  -t: 824.82
iter: 13/50 --> E: 0.0227428447  -Training_Accuracy:  98.02  -t: 890.90
iter: 14/50 --> E: 0.0200161531  -Training_Accuracy:  98.30  -t: 957.99
iter: 15/50 --> E: 0.0178362423  -Training_Accuracy:  98.48  -t: 1026.82
iter: 16/50 --> E: 0.0159113496  -Training_Accuracy:  98.69  -t: 1097.33
iter: 17/50 --> E: 0.0145313265  -Training_Accuracy:  98.80  -t: 1164.69
iter: 18/50 --> E: 0.0130241812  -Training_Accuracy:  98.91  -t: 1232.44
iter: 19/50 --> E: 0.0118651899  -Training_Accuracy:  99.04  -t: 1304.70
iter: 20/50 --> E: 0.0108024943  -Training_Accuracy:  99.16  -t: 1373.97
iter: 21/50 --> E: 0.0099066161  -Training_Accuracy:  99.18  -t: 1440.93
iter: 22/50 --> E: 0.0090870950  -Training_Accuracy:  99.25  -t: 1509.60
iter: 23/50 --> E: 0.0083637677  -Training_Accuracy:  99.29  -t: 1575.12
iter: 24/50 --> E: 0.0076999964  -Training_Accuracy:  99.35  -t: 1643.00
iter: 25/50 --> E: 0.0071878357  -Training_Accuracy:  99.39  -t: 1711.24
iter: 26/50 --> E: 0.0066516290  -Training_Accuracy:  99.41  -t: 1776.33
iter: 27/50 --> E: 0.0062573276  -Training_Accuracy:  99.43  -t: 1841.51
iter: 28/50 --> E: 0.0058616831  -Training_Accuracy:  99.47  -t: 1908.28
iter: 29/50 --> E: 0.0054684152  -Training_Accuracy:  99.49  -t: 1975.82
iter: 30/50 --> E: 0.0051409496  -Training_Accuracy:  99.50  -t: 2047.22
iter: 31/50 --> E: 0.0048156459  -Training_Accuracy:  99.52  -t: 2115.99
iter: 32/50 --> E: 0.0045425009  -Training_Accuracy:  99.53  -t: 2184.97
iter: 33/50 --> E: 0.0043482776  -Training_Accuracy:  99.54  -t: 2254.23
iter: 34/50 --> E: 0.0041039209  -Training_Accuracy:  99.56  -t: 2326.64
iter: 35/50 --> E: 0.0039312526  -Training_Accuracy:  99.57  -t: 2400.09
iter: 36/50 --> E: 0.0037732251  -Training_Accuracy:  99.58  -t: 2470.21
iter: 37/50 --> E: 0.0036208538  -Training_Accuracy:  99.59  -t: 2539.89
iter: 38/50 --> E: 0.0034969571  -Training_Accuracy:  99.60  -t: 2607.26
iter: 39/50 --> E: 0.0033416947  -Training_Accuracy:  99.60  -t: 2676.31
iter: 40/50 --> E: 0.0032299487  -Training_Accuracy:  99.61  -t: 2746.20
iter: 41/50 --> E: 0.0031325509  -Training_Accuracy:  99.62  -t: 2816.91
iter: 42/50 --> E: 0.0030437148  -Training_Accuracy:  99.62  -t: 2886.77
iter: 43/50 --> E: 0.0029576864  -Training_Accuracy:  99.63  -t: 2952.29
iter: 44/50 --> E: 0.0028784952  -Training_Accuracy:  99.64  -t: 3017.97
iter: 45/50 --> E: 0.0028120449  -Training_Accuracy:  99.64  -t: 3085.21
iter: 46/50 --> E: 0.0027396760  -Training_Accuracy:  99.65  -t: 3152.80
iter: 47/50 --> E: 0.0026819551  -Training_Accuracy:  99.66  -t: 3219.55
iter: 48/50 --> E: 0.0026173603  -Training_Accuracy:  99.66  -t: 3287.27
iter: 49/50 --> E: 0.0025639354  -Training_Accuracy:  99.67  -t: 3356.84
iter: 50/50 --> E: 0.0025067429  -Training_Accuracy:  99.67  -t: 3426.28
```




```
In [85]: # 300 hidden neurons
         my_mnist_net_300 = nn.NNet(n_input=784,
                                    netDims=[300,10],
                                    n_iter=50,
                                    learn=0.1)

In [91]: Tr_acc300, Val_acc300 = my_mnist_net_300.train(training_data,
                                                        validation_data,
                                                        True, True, True)
         my_mnist_net_300.save('mnist_nD300-10_it50_l01.model')
```

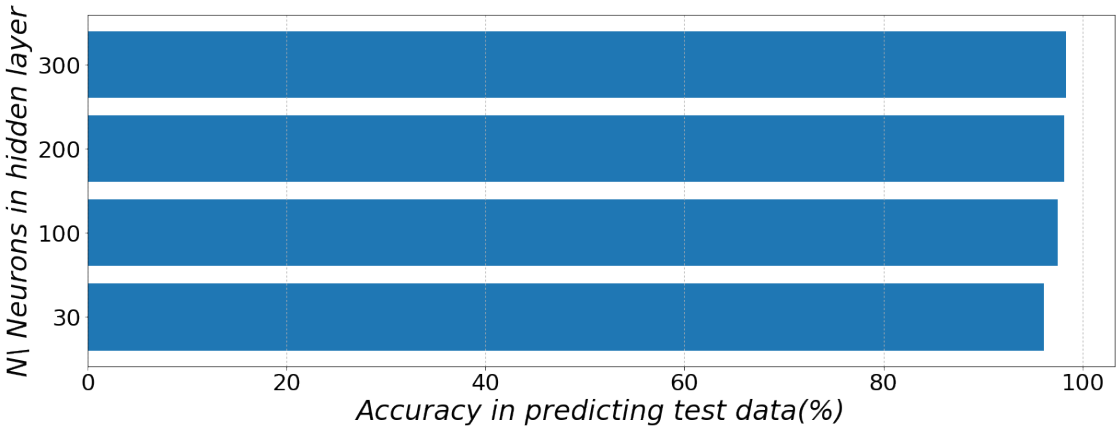
```
iter:  1/50 --> E: 0.1049612095  -Training_Accuracy:  93.09  -t: 85.05
iter:  2/50 --> E: 0.0529917190  -Training_Accuracy:  95.21  -t: 170.47
iter:  3/50 --> E: 0.0388029736  -Training_Accuracy:  96.55  -t: 255.59
iter:  4/50 --> E: 0.0309945032  -Training_Accuracy:  97.17  -t: 340.66
iter:  5/50 --> E: 0.0257559440  -Training_Accuracy:  97.63  -t: 425.90
iter:  6/50 --> E: 0.0221621472  -Training_Accuracy:  97.86  -t: 511.30
iter:  7/50 --> E: 0.0191225638  -Training_Accuracy:  98.15  -t: 596.11
iter:  8/50 --> E: 0.0167346066  -Training_Accuracy:  98.43  -t: 681.25
iter:  9/50 --> E: 0.0148907601  -Training_Accuracy:  98.75  -t: 766.32
iter: 10/50 --> E: 0.0132401383  -Training_Accuracy:  98.94  -t: 851.62
iter: 11/50 --> E: 0.0117740454  -Training_Accuracy:  98.98  -t: 936.82
iter: 12/50 --> E: 0.0105600532  -Training_Accuracy:  99.13  -t: 1022.27
iter: 13/50 --> E: 0.0095468072  -Training_Accuracy:  99.20  -t: 1107.59
iter: 14/50 --> E: 0.0087030590  -Training_Accuracy:  99.25  -t: 1192.82
```

```
iter: 15/50 --> E: 0.0079681491  -Training_Accuracy:  99.33  -t: 1278.83
iter: 16/50 --> E: 0.0071797080  -Training_Accuracy:  99.35  -t: 1364.00
iter: 17/50 --> E: 0.0066252627  -Training_Accuracy:  99.42  -t: 1449.46
iter: 18/50 --> E: 0.0061078043  -Training_Accuracy:  99.46  -t: 1534.62
iter: 19/50 --> E: 0.0056602655  -Training_Accuracy:  99.50  -t: 1621.25
iter: 20/50 --> E: 0.0051701583  -Training_Accuracy:  99.52  -t: 1709.55
iter: 21/50 --> E: 0.0048079952  -Training_Accuracy:  99.55  -t: 1799.96
iter: 22/50 --> E: 0.0044432394  -Training_Accuracy:  99.56  -t: 1891.54
iter: 23/50 --> E: 0.0041562491  -Training_Accuracy:  99.59  -t: 1982.21
iter: 24/50 --> E: 0.0039258599  -Training_Accuracy:  99.61  -t: 2067.89
iter: 25/50 --> E: 0.0037224550  -Training_Accuracy:  99.61  -t: 2154.89
iter: 26/50 --> E: 0.0034914878  -Training_Accuracy:  99.62  -t: 2242.03
iter: 27/50 --> E: 0.0033066309  -Training_Accuracy:  99.65  -t: 2329.15
iter: 28/50 --> E: 0.0031831145  -Training_Accuracy:  99.66  -t: 2414.39
iter: 29/50 --> E: 0.0030349770  -Training_Accuracy:  99.66  -t: 2500.69
iter: 30/50 --> E: 0.0029268490  -Training_Accuracy:  99.67  -t: 2588.06
iter: 31/50 --> E: 0.0028105304  -Training_Accuracy:  99.68  -t: 2674.42
iter: 32/50 --> E: 0.0026908205  -Training_Accuracy:  99.69  -t: 2758.94
iter: 33/50 --> E: 0.0025880148  -Training_Accuracy:  99.70  -t: 2843.80
iter: 34/50 --> E: 0.0024913068  -Training_Accuracy:  99.70  -t: 2929.48
iter: 35/50 --> E: 0.0024174231  -Training_Accuracy:  99.71  -t: 3014.37
iter: 36/50 --> E: 0.0023275222  -Training_Accuracy:  99.71  -t: 3099.24
iter: 37/50 --> E: 0.0022579543  -Training_Accuracy:  99.72  -t: 3188.75
iter: 38/50 --> E: 0.0021906128  -Training_Accuracy:  99.73  -t: 3276.32
iter: 39/50 --> E: 0.0021393906  -Training_Accuracy:  99.73  -t: 3363.12
iter: 40/50 --> E: 0.0021073285  -Training_Accuracy:  99.73  -t: 3449.52
iter: 41/50 --> E: 0.0020517296  -Training_Accuracy:  99.74  -t: 3537.69
iter: 42/50 --> E: 0.0020137671  -Training_Accuracy:  99.74  -t: 3624.67
iter: 43/50 --> E: 0.0019749591  -Training_Accuracy:  99.74  -t: 3713.14
iter: 44/50 --> E: 0.0019401873  -Training_Accuracy:  99.74  -t: 3801.16
iter: 45/50 --> E: 0.0019044590  -Training_Accuracy:  99.74  -t: 3890.82
iter: 46/50 --> E: 0.0018652426  -Training_Accuracy:  99.74  -t: 3978.33
iter: 47/50 --> E: 0.0018318800  -Training_Accuracy:  99.75  -t: 4065.86
iter: 48/50 --> E: 0.0017979260  -Training_Accuracy:  99.75  -t: 4154.21
iter: 49/50 --> E: 0.0017740611  -Training_Accuracy:  99.76  -t: 4239.89
iter: 50/50 --> E: 0.0017396917  -Training_Accuracy:  99.76  -t: 4325.26
```




```
In [68]: my_mnist_net_100.load("mnist_nD100-10_it50_l01.model")
         my_mnist_net_200.load("mnist_nD200-10_it50_l01.model")
         my_mnist_net_300.load('mnist_nD300-10_it50_l01.model')

In [93]: learn_rates = [30, 100, 200, 300]
         x_range = [1,2,3,4]
         accuracies = [0,0,0,0]
         accuracies[0] = my_mnist_net.predict(test_data)/len(test_data)*100
         accuracies[1] = my_mnist_net_100.predict(test_data)/len(test_data)*100
         accuracies[2] = my_mnist_net_200.predict(test_data)/len(test_data)*100
         accuracies[3] = my_mnist_net_300.predict(test_data)/len(test_data)*100

         plt.figure(figsize=[20,7])
         plt.barh(x_range, accuracies)
         plt.yticks(x_range, learn_rates, fontsize=25)
         plt.xticks(fontsize=25)
         plt.ylabel('N\ Neurons in hidden layer',
                    fontsize=30, fontstyle='oblique')
         plt.xlabel('Accuracy in predicting test data(%)',
                    fontsize=30,
                    fontstyle='oblique')
         plt.grid(True, axis='x', ls=':')
         plt.show()
         print('Accuracy of my_mnist_net on test data: ', accuracies[0], '%')
         print('Accuracy of my_mnist_net_100 on test data: ', accuracies[1], '%')
```

```
print('Accuracy of my_mnist_net_200 on test data: ', accuracies[2], '%')
print('Accuracy of my_mnist_net_300 on test data: ', accuracies[3], '%')
```




```
Accuracy of my_mnist_net on test data:  96.12 %
Accuracy of my_mnist_net_100 on test data:  97.57000000000001 %
Accuracy of my_mnist_net_200 on test data:  98.17 %
Accuracy of my_mnist_net_300 on test data:  98.33 %
```

COMMENT As we can clearly see, adding more neurons in the hidden layer does increase the the accuracy of the NNet. Thus, using 300 neurons would be the best choice if highest accuracy is the objective. Nevertheless, the time that it takes to train also increases substantially.

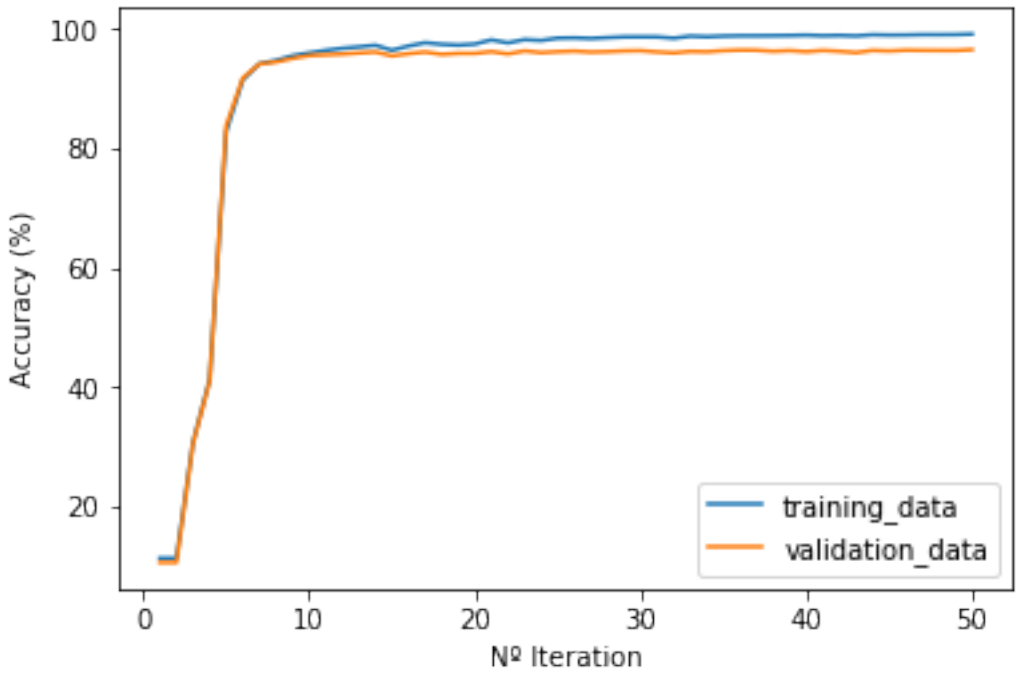
3.3.5 Question 2.2.5

Add one additionnal hidden layers and train your network, discuss your results with different setting.

```
In [ ]: my_final_net = nn.NNet(n_input=784,
                               netDims=[40, 30, 20, 10],
                               n_iter=50,
                               learn=0.1)

In [3]: Tr_accFin, Val_accFin = my_final_net.train(training_data,
                                                   validation_data,
                                                   True, True, True)
        my_final_net.save("mnist_nD40-30-20-10_it50_l005.model")
```

```
iter:  1/50 --> E: 0.4542410481  -Training_Accuracy:  11.36  -t: 13.97
iter:  2/50 --> E: 0.4505962036  -Training_Accuracy:  11.36  -t: 28.07
iter:  3/50 --> E: 0.4269295590  -Training_Accuracy:  30.80  -t: 42.06
```

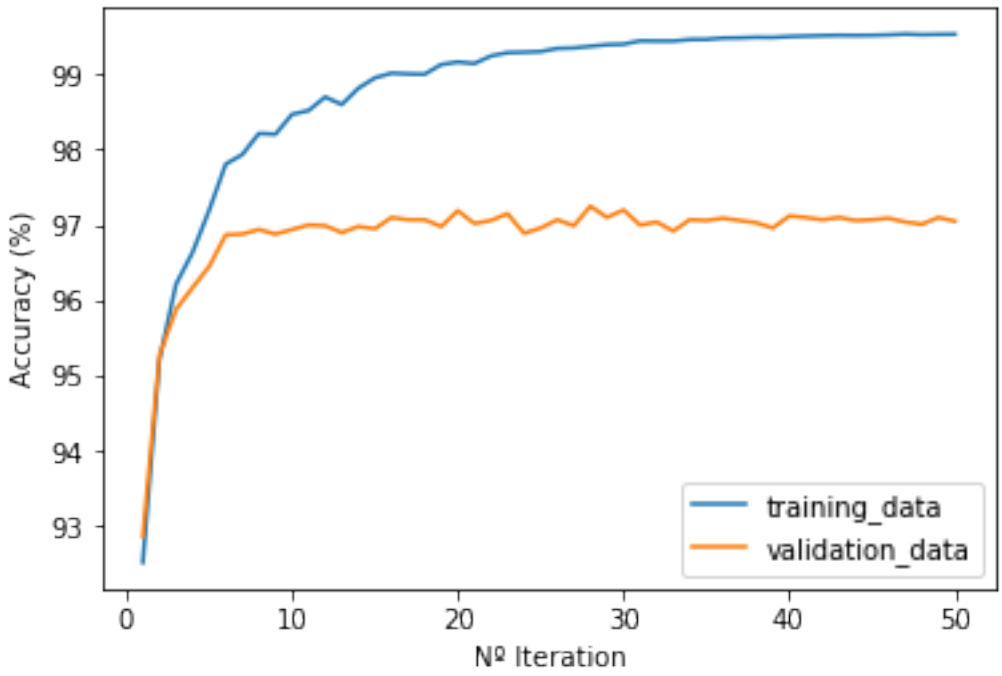
```
iter:  4/50 --> E: 0.3508702696  -Training_Accuracy:  41.26  -t: 56.02
iter:  5/50 --> E: 0.2507930599  -Training_Accuracy:  82.82  -t: 70.01
iter:  6/50 --> E: 0.0938990633  -Training_Accuracy:  91.50  -t: 83.85
iter:  7/50 --> E: 0.0589105144  -Training_Accuracy:  94.20  -t: 97.76
iter:  8/50 --> E: 0.0470973265  -Training_Accuracy:  94.75  -t: 111.74
iter:  9/50 --> E: 0.0404974185  -Training_Accuracy:  95.54  -t: 126.12
iter: 10/50 --> E: 0.0359570604  -Training_Accuracy:  96.01  -t: 141.37
iter: 11/50 --> E: 0.0324975302  -Training_Accuracy:  96.43  -t: 156.33
iter: 12/50 --> E: 0.0303520843  -Training_Accuracy:  96.79  -t: 170.84
iter: 13/50 --> E: 0.0282469211  -Training_Accuracy:  97.02  -t: 185.27
iter: 14/50 --> E: 0.0262667683  -Training_Accuracy:  97.30  -t: 199.22
iter: 15/50 --> E: 0.0249745936  -Training_Accuracy:  96.46  -t: 213.19
iter: 16/50 --> E: 0.0230694222  -Training_Accuracy:  97.17  -t: 227.29
iter: 17/50 --> E: 0.0221622333  -Training_Accuracy:  97.73  -t: 241.32
iter: 18/50 --> E: 0.0205153140  -Training_Accuracy:  97.46  -t: 255.40
iter: 19/50 --> E: 0.0202580803  -Training_Accuracy:  97.33  -t: 269.35
iter: 20/50 --> E: 0.0190820080  -Training_Accuracy:  97.50  -t: 283.28
iter: 21/50 --> E: 0.0181468483  -Training_Accuracy:  98.20  -t: 298.00
iter: 22/50 --> E: 0.0173802682  -Training_Accuracy:  97.70  -t: 312.06
iter: 23/50 --> E: 0.0165595328  -Training_Accuracy:  98.25  -t: 326.05
iter: 24/50 --> E: 0.0156784605  -Training_Accuracy:  98.10  -t: 340.19
iter: 25/50 --> E: 0.0151297120  -Training_Accuracy:  98.49  -t: 354.24
iter: 26/50 --> E: 0.0143756456  -Training_Accuracy:  98.53  -t: 368.33
iter: 27/50 --> E: 0.0141032247  -Training_Accuracy:  98.44  -t: 382.35
iter: 28/50 --> E: 0.0139319507  -Training_Accuracy:  98.58  -t: 396.55
iter: 29/50 --> E: 0.0130328014  -Training_Accuracy:  98.71  -t: 410.53
iter: 30/50 --> E: 0.0125493962  -Training_Accuracy:  98.71  -t: 424.53
iter: 31/50 --> E: 0.0120523204  -Training_Accuracy:  98.70  -t: 438.68
iter: 32/50 --> E: 0.0117045723  -Training_Accuracy:  98.48  -t: 452.72
iter: 33/50 --> E: 0.0117924803  -Training_Accuracy:  98.83  -t: 466.81
iter: 34/50 --> E: 0.0107845727  -Training_Accuracy:  98.73  -t: 480.86
iter: 35/50 --> E: 0.0111075992  -Training_Accuracy:  98.84  -t: 494.88
iter: 36/50 --> E: 0.0103706261  -Training_Accuracy:  98.89  -t: 508.79
iter: 37/50 --> E: 0.0101111351  -Training_Accuracy:  98.89  -t: 522.71
iter: 38/50 --> E: 0.0100552931  -Training_Accuracy:  98.91  -t: 536.65
iter: 39/50 --> E: 0.0097755509  -Training_Accuracy:  98.94  -t: 550.64
iter: 40/50 --> E: 0.0097477602  -Training_Accuracy:  98.99  -t: 564.52
iter: 41/50 --> E: 0.0092065547  -Training_Accuracy:  98.90  -t: 578.50
iter: 42/50 --> E: 0.0092692358  -Training_Accuracy:  98.95  -t: 592.33
iter: 43/50 --> E: 0.0088590607  -Training_Accuracy:  98.87  -t: 606.30
iter: 44/50 --> E: 0.0090793420  -Training_Accuracy:  99.05  -t: 620.28
iter: 45/50 --> E: 0.0088262950  -Training_Accuracy:  99.00  -t: 634.18
iter: 46/50 --> E: 0.0083577899  -Training_Accuracy:  99.02  -t: 648.08
iter: 47/50 --> E: 0.0079184430  -Training_Accuracy:  99.07  -t: 661.95
iter: 48/50 --> E: 0.0081114227  -Training_Accuracy:  99.07  -t: 675.91
iter: 49/50 --> E: 0.0077722262  -Training_Accuracy:  99.09  -t: 689.82
iter: 50/50 --> E: 0.0077525163  -Training_Accuracy:  99.17  -t: 703.72
```




```
In [70]: my_final_net = nn.NNet(n_input=784,
                                netDims=[50, 50, 10],
                                n_iter=50,
                                learn=0.1)

In [4]: Tr_accFin, Val_accFin = my_final_net.train(training_data,
                                                   validation_data,
                                                   True,True, True)
        my_final_net.save("mnist_nD50-50-10_it50_l005.model")
```

```
iter:  1/50 --> E: 0.1931058091  -Training_Accuracy:  92.52  -t: 13.41
iter:  2/50 --> E: 0.0510291682  -Training_Accuracy:  95.21  -t: 27.03
iter:  3/50 --> E: 0.0382512938  -Training_Accuracy:  96.21  -t: 40.53
iter:  4/50 --> E: 0.0314694751  -Training_Accuracy:  96.64  -t: 54.08
iter:  5/50 --> E: 0.0269607170  -Training_Accuracy:  97.20  -t: 67.76
iter:  6/50 --> E: 0.0237940038  -Training_Accuracy:  97.81  -t: 81.11
iter:  7/50 --> E: 0.0211354928  -Training_Accuracy:  97.93  -t: 94.50
iter:  8/50 --> E: 0.0188876801  -Training_Accuracy:  98.21  -t: 108.14
iter:  9/50 --> E: 0.0173905502  -Training_Accuracy:  98.20  -t: 121.57
iter: 10/50 --> E: 0.0163844649  -Training_Accuracy:  98.47  -t: 134.97
iter: 11/50 --> E: 0.0148305431  -Training_Accuracy:  98.52  -t: 148.54
iter: 12/50 --> E: 0.0138089674  -Training_Accuracy:  98.70  -t: 162.00
iter: 13/50 --> E: 0.0126515454  -Training_Accuracy:  98.60  -t: 175.45
iter: 14/50 --> E: 0.0119518641  -Training_Accuracy:  98.81  -t: 189.11
iter: 15/50 --> E: 0.0112232718  -Training_Accuracy:  98.95  -t: 202.53
```

```
iter: 16/50 --> E: 0.0105047386  -Training_Accuracy:  99.01  -t: 215.90
iter: 17/50 --> E: 0.0099270803  -Training_Accuracy:  99.00  -t: 229.47
iter: 18/50 --> E: 0.0094859649  -Training_Accuracy:  99.00  -t: 242.92
iter: 19/50 --> E: 0.0087353985  -Training_Accuracy:  99.13  -t: 256.30
iter: 20/50 --> E: 0.0083747351  -Training_Accuracy:  99.16  -t: 269.94
iter: 21/50 --> E: 0.0077831442  -Training_Accuracy:  99.14  -t: 283.29
iter: 22/50 --> E: 0.0074634499  -Training_Accuracy:  99.24  -t: 296.67
iter: 23/50 --> E: 0.0070260952  -Training_Accuracy:  99.29  -t: 310.36
iter: 24/50 --> E: 0.0066626493  -Training_Accuracy:  99.29  -t: 323.79
iter: 25/50 --> E: 0.0061958670  -Training_Accuracy:  99.30  -t: 337.17
iter: 26/50 --> E: 0.0059491881  -Training_Accuracy:  99.34  -t: 350.84
iter: 27/50 --> E: 0.0058142449  -Training_Accuracy:  99.35  -t: 364.31
iter: 28/50 --> E: 0.0056916360  -Training_Accuracy:  99.37  -t: 377.77
iter: 29/50 --> E: 0.0053383234  -Training_Accuracy:  99.39  -t: 391.33
iter: 30/50 --> E: 0.0051009069  -Training_Accuracy:  99.40  -t: 404.73
iter: 31/50 --> E: 0.0050562872  -Training_Accuracy:  99.44  -t: 418.25
iter: 32/50 --> E: 0.0047960074  -Training_Accuracy:  99.44  -t: 431.92
iter: 33/50 --> E: 0.0044830402  -Training_Accuracy:  99.44  -t: 445.32
iter: 34/50 --> E: 0.0042889840  -Training_Accuracy:  99.46  -t: 458.70
iter: 35/50 --> E: 0.0042212734  -Training_Accuracy:  99.46  -t: 472.26
iter: 36/50 --> E: 0.0040044918  -Training_Accuracy:  99.48  -t: 485.76
iter: 37/50 --> E: 0.0040144894  -Training_Accuracy:  99.48  -t: 499.28
iter: 38/50 --> E: 0.0038647428  -Training_Accuracy:  99.49  -t: 512.86
iter: 39/50 --> E: 0.0037044652  -Training_Accuracy:  99.49  -t: 526.35
iter: 40/50 --> E: 0.0036852508  -Training_Accuracy:  99.50  -t: 539.86
iter: 41/50 --> E: 0.0035512697  -Training_Accuracy:  99.51  -t: 553.41
iter: 42/50 --> E: 0.0034855690  -Training_Accuracy:  99.51  -t: 566.92
iter: 43/50 --> E: 0.0034291928  -Training_Accuracy:  99.52  -t: 580.36
iter: 44/50 --> E: 0.0033657491  -Training_Accuracy:  99.51  -t: 594.01
iter: 45/50 --> E: 0.0033137351  -Training_Accuracy:  99.52  -t: 607.44
iter: 46/50 --> E: 0.0032712876  -Training_Accuracy:  99.52  -t: 620.84
iter: 47/50 --> E: 0.0032577264  -Training_Accuracy:  99.53  -t: 634.43
iter: 48/50 --> E: 0.0032355677  -Training_Accuracy:  99.52  -t: 647.90
iter: 49/50 --> E: 0.0032092787  -Training_Accuracy:  99.53  -t: 661.45
iter: 50/50 --> E: 0.0033581043  -Training_Accuracy:  99.53  -t: 675.04
```




```
In [94]: print('Accuracy of my_final_net on test data: ', my_final_net.predict(test
```

```
Accuracy of my_final_net on test data:  97.09 %
```

COMMENT Although there is an increase in accuracy when compared to the shallow architectures, the final accuracy with an extra layer (97%) is still a step below using 300 neurons in the hidden layer (98.33%). Nevertheless, the decrease in training time is a parameter to be considered.

3.4 Storing and Shipping Results

```
In [55]: # Save all the accuracies
         Tr_accuracy ={'learn0.1':Tr_acc1,
                       'learn0.001':Tr_acc2,
                       'learn1':Tr_acc4,
                       'learn10':Tr_acc5,
                       'Weq0':Tr_acc0,
                       'tfTanh':Tr_accTanh,
                       'nHid100':Tr_acc100,
                       'nHid200':Tr_acc200,
                       'nHid300':Tr_acc300}
         Val_accuracy ={'learn0.1':Val_acc1,
                       'learn0.001':Val_acc2,
                       'learn1':Val_acc4,
```

```
                        'learn10':Val_acc5,
                        'Weq0':Val_acc0,
                        'tfTanh':Val_accTanh,
                        'nHid100':Val_acc100,
                        'nHid200':Val_acc200,
                        'nHid300':Val_acc300}
            with open("Accuracies.vectors", 'wb') as f:
                pickle.dump({'Tr_accuracy':Tr_accuracy,
                             'Val_accuracy':Val_accuracy}, f )
```

In [ ]:
```
# Reload accuracies
with open("Accuracies.vectors", 'rb') as f:
    data = pickle.load(f)
Tr_accuracy = data['Tr_accuracy']
Val_accuracy = data['Val_accuracy']
```