Deep Learning Lab Session

First Lab Session - 3 Hours

Artificial Neural Networks for Handwritten Digits Recognition

Student 1: # Zakaria EL KHAMAR **Student 2:** # Oussama EL MAATAOUI

The aim of this session is to practice with Artificial Neural Networks. Answers and experiments should be made by groups of one or two students. Each group should fill and run appropriate notebook cells.

To generate your final report, use print as PDF (Ctrl+P). Do not forget to run all your cells before generating your final report and do not forget to include the names of all participants in the group. The lab session should be completed by April 7th 2017.

Introduction

In this session, your will implement, train and test a Neural Network for the Handwritten Digits Recognition problem [1] (http://yann.lecun.com/exdb/mnist/) 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.

<img src="Nimages/mnist.png",width="350" height="500" align="center">

Figure 1: MNIST digits examples

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 (NeuralNetwork.py)
2. transfer_functions.py (transfer_functions.py)
3. utils.py (utils.py)

Functions defined inside the python files mentionned above can be imported using the python command : from filename import *

You will use the following libraries:

1. numpy (http://cs231n.github.io/python-numpy-tutorial/): for creating arrays and using methods to manipulate arrays.
2. matplotlib (http://matplotlib.org/): for making plots

Section 1 : My First Neural Network

Part 1: 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

<img src="Nimages/NN.png", width="700" height="900">

Figure 2: Neural network

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.

Your answer goes here :

$w_{1,3} = 0.3043$

$w_{1,4} = -0.5027$

$w_{2,3} = 0.801$

$w_{2,4} = 0.1993$

$w_{6,3} = 0.2054$

$w_{6,4} = -0.4034$

$w_{3,5} = -0.6254$

$w_{4,5} = 0.3874$

$w_{7,5} = 0.4606$

Part 2: Neural Network Implementation

Please read all source files carefully and understand the data structures and all functions. You are to complete the missing code. First you should define the neural network (using the NeuralNetwork class, see in the NeuralNetwork.py (NeuralNetwork.py) file) and reinitialise weights. Then you will to complete the Feed Forward and the Back-propagation functions.

Question 1.2.1: Define the neural network corresponding to the one in part 1

In [1]:
```
from NeuralNetwork import *
#create the network
my_first_net = NeuralNetwork(2,2,1,iterations = 50, learning_rate = 1.0)
```

In [2]:
```
#Data preparation
X=[0.8,0.2]
Y=[0.4]
data=[]
data.append(X)
data.append(Y)

#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_first_net.weights_initialisation(wi,wo)
print(my_first_net.W_input_to_hidden)
print(my_first_net.W_hidden_to_output)
```

```
[[ 0.3 -0.5]
 [ 0.8  0.2]
 [ 0.2 -0.4]]
[[-0.6]
 [ 0.4]
 [ 0.5]]
```

Question 1.2.2: Implement the Feed Forward function (feedForward(X) in the NeuralNetwork.py file)

In [3]:
```
# Implement it in the NeuralNetwork.py file and when finalised copy and paste your FeedForward function her
e
def feedForward(self, inputs):
        # Compute input activations
        self.a_input= np.append(np.array(inputs),1)


        #Compute  hidden activations
        for i in range (self.hidden-1) :
            transpose_weights = np.transpose(self.W_input_to_hidden)
            self.a_hidden[i] = sigmoid(np.dot(transpose_weights[i],self.a_input))
        self.a_hidden[self.hidden-1] = 1.0

        # Compute output activations
        for j in range (self.output) :
            transpose_weights_O = np.transpose(self.W_hidden_to_output)
            self.a_out[j] = sigmoid(np.dot(transpose_weights_O[j],self.a_hidden))

        return self.a_out
```

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

In [4]:
```
#test my  Feed Forward function
Output_activation=my_first_net.feedForward(X)
print("output activation =%.3f" %(Output_activation))
```

```
output activation =0.560
```

Question 1.2.3: Implement the Back-propagation Algorithm (backPropagate(Y) in the NeuralNetwork.py file)

In [ ]:
```
# Implement it in the NeuralNetwork.py file and when finalised copy and paste your FeedForward function her
e

def backPropagate(self, targets):
        self.targets = targets
        # calculate error terms for output
        self.error_o = np.ones(self.output)
        for i in range(self.output) :
            self.error_o[i] = (self.a_out[i] - self.targets[i]) * self.a_out[i] * (1-self.a_out[i])

        # calculate error terms for hidden
        self.error_h = np.ones(self.hidden)
        for j in range(self.hidden) :
            for i in range(self.output) :
                self.error_h[j] += self.error_o[i] * self.W_hidden_to_output[j][i]
            self.error_h[j] = self.error_h[j] * self.a_hidden[j] * (1- self.a_hidden[j])
        # update output weights
        for i in range(self.output) :
            for j in range(self.hidden) :
                self.W_hidden_to_output[j][i] -= self.error_o[i] * self.a_hidden[j]* self.learning_rate
        # update input weights
        for i in range(self.hidden-1) :
            for j in range(self.input) :
                self.W_input_to_hidden[j][i] -= self.error_h[i] * self.a_input[j] * self.learning_rate
        # calculate error
        self.error = 0
        self.a = self.a_input[0:2]
        for i in range(self.output) :

            self.error += (self.feedForward(self.a)[i] -self.targets[i] ) ** 2
        return self.error
```

Comment

We first programmed it this way, then we discovered in the section 2 that it is too computationnaly demanding. So we wrote it using vectors and matrices :

In [11]:
```
def feedForward(self, inputs):

        # Compute input activations
        self.a_input= np.append(np.array(inputs),1)

        #Compute  hidden activations
        self.a_hidden = sigmoid(np.dot(self.a_input,self.W_input_to_hidden))
        self.a_hidden = np.append(self.a_hidden,1)

        # Compute output activations
        self.a_out = sigmoid(np.dot(self.a_hidden,self.W_hidden_to_output))
        return(self.a_out)
```

In [6]:
```
# Implement it in the NeuralNetwork.py file and when finalised copy and paste your backpropagation function
here

def backPropagate(self, targets):

    vector_dsigmoid = np.vectorize(dsigmoid)

        # calculate error terms for output
    error_out = (self.a_out - targets) * vector_dsigmoid(self.a_out)

        # calculate error terms for hidden
    error_hidden = error_out.dot(self.W_hidden_to_output.transpose()) * vector_dsigmoid(self.a_hidden)

        # update output weights
    self.W_hidden_to_output -= self.learning_rate * np.outer(self.a_hidden, error_out)

        # update input weights
    self.W_input_to_hidden -= self.learning_rate * np.outer(self.a_input, error_hidden[0:-1])


        # calculate error
    error = np.sum((self.a_out - targets)**2 )


    return error
```

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

In [7]:
```
#test my  Back-propagation function
my_first_net.backPropagate(Y)
#Print weights after backpropagation
print('wi_new=', my_first_net.W_input_to_hidden)
print('wo_new=', my_first_net.W_hidden_to_output)
```

```
('wi_new=', array([[ 0.30432265, -0.50273473],
       [ 0.80108066,  0.19931632],
       [ 0.20540332, -0.40341841]]))
('wo_new=', array([[-0.62541468],
       [ 0.38745727],
       [ 0.46063746]]))
```

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

Section 2 : The MNIST Challenge!

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.

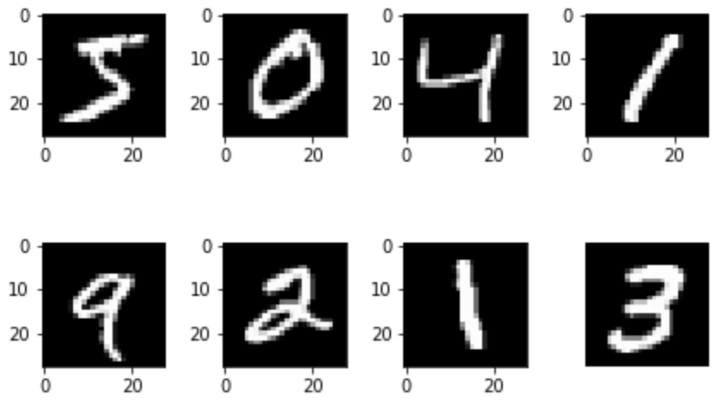
The MNIST dataset is available in the Data folder. To get the training, testing and validation data, run the the load_data() function.

In [1]:
```
from utils import *
training_data, validation_data, test_data=load_data()
```

```
Loading MNIST data .....
Done.
```

MNIST Dataset Digits Visualisation

In [2]:
```
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')  # cmap='gray' is for black and white picture.
plt.axis('off')  # do not show axis value
plt.tight_layout()   # automatic padding between subplots
plt.show()
```



Part 1: Creating the Neural Networks

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.

Question 2.1.1: Create the network described above using the NeuralNetwork class

In [3]:
```
#create the network
from NeuralNetwork import *
my_mnist_net = NeuralNetwork(784,30,10,iterations = 50, learning_rate = 1.0)
```

Question 2.1.2: Add the information about the performance of the neural network on the test set at each epoch

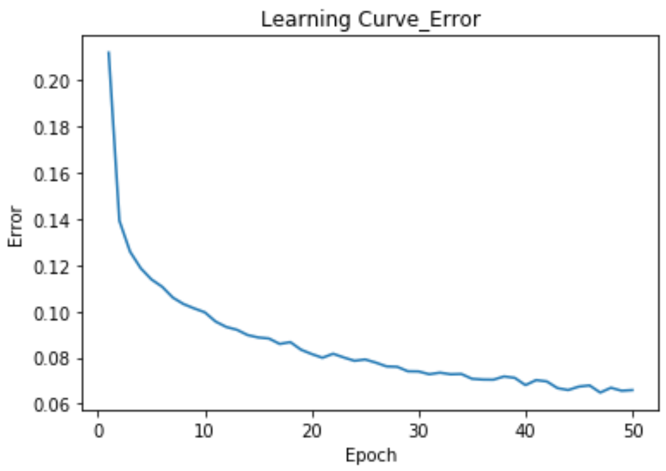
In [4]:
```
test_accuracy=my_mnist_net.predict(test_data)/100
print('Test_Accuracy  %-2.2f' % test_accuracy)
```

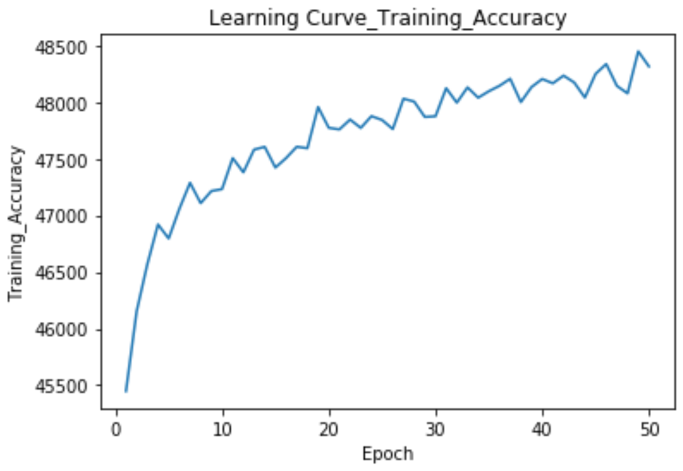
```
Test_Accuracy  8.79
```

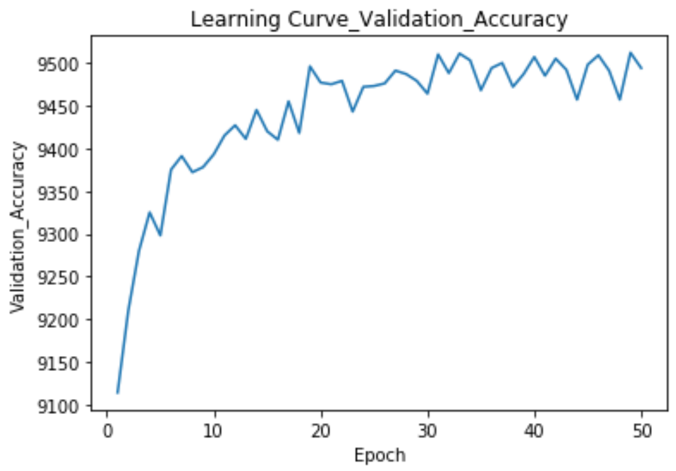
Question 2.1.3: Train the Neural Network and comment your findings

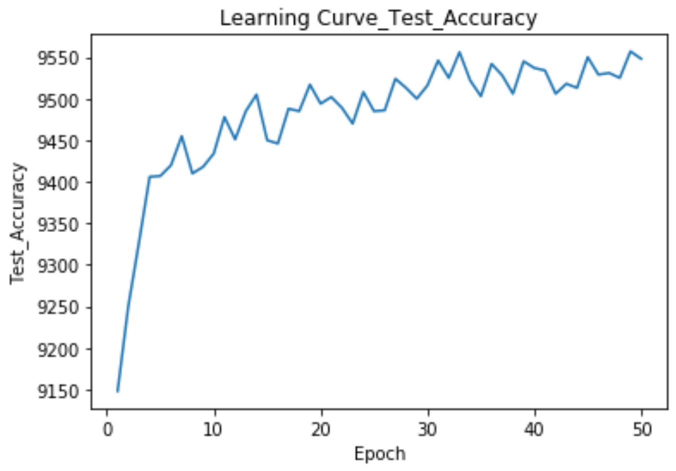
In [6]:
```
#train your network
my_mnist_net.train(training_data,validation_data,test_data)

#save your model in Models/ using a distinguishing name for your model (architecture, learning rate, etc...
)
my_mnist_net.save('Models/NET_784_30_10_1.0')
```

```
Iteration:  1/50[==============] -Error: 0.2119198242  -Training_Accuracy:  90.90  -time: 17.78
Iteration:  2/50[==============] -Error: 0.1392392300  -Training_Accuracy:  92.31  -time: 35.97
Iteration:  3/50[==============] -Error: 0.1258422241  -Training_Accuracy:  93.14  -time: 53.47
Iteration:  4/50[==============] -Error: 0.1186574515  -Training_Accuracy:  93.84  -time: 71.65
Iteration:  5/50[==============] -Error: 0.1138305294  -Training_Accuracy:  93.59  -time: 89.50
Iteration:  6/50[==============] -Error: 0.1106696862  -Training_Accuracy:  94.11  -time: 107.28
Iteration:  7/50[==============] -Error: 0.1059771247  -Training_Accuracy:  94.57  -time: 125.16
Iteration:  8/50[==============] -Error: 0.1031771729  -Training_Accuracy:  94.21  -time: 146.09
Iteration:  9/50[==============] -Error: 0.1012919590  -Training_Accuracy:  94.43  -time: 166.49
Iteration: 10/50[==============] -Error: 0.0995852512  -Training_Accuracy:  94.46  -time: 186.63
Iteration: 11/50[==============] -Error: 0.0956151303  -Training_Accuracy:  95.01  -time: 205.44
Iteration: 12/50[==============] -Error: 0.0932624453  -Training_Accuracy:  94.76  -time: 224.79
Iteration: 13/50[==============] -Error: 0.0920419379  -Training_Accuracy:  95.16  -time: 247.09
Iteration: 14/50[==============] -Error: 0.0897589301  -Training_Accuracy:  95.21  -time: 267.41
Iteration: 15/50[==============] -Error: 0.0887148493  -Training_Accuracy:  94.84  -time: 285.36
Iteration: 16/50[==============] -Error: 0.0882594743  -Training_Accuracy:  95.01  -time: 304.78
Iteration: 17/50[==============] -Error: 0.0859189984  -Training_Accuracy:  95.21  -time: 329.73
Iteration: 18/50[==============] -Error: 0.0867103951  -Training_Accuracy:  95.18  -time: 350.57
Iteration: 19/50[==============] -Error: 0.0834142806  -Training_Accuracy:  95.91  -time: 370.54
Iteration: 20/50[==============] -Error: 0.0815434346  -Training_Accuracy:  95.54  -time: 392.20
Iteration: 21/50[==============] -Error: 0.0799248659  -Training_Accuracy:  95.52  -time: 413.28
Iteration: 22/50[==============] -Error: 0.0816697861  -Training_Accuracy:  95.69  -time: 434.46
Iteration: 23/50[==============] -Error: 0.0800643981  -Training_Accuracy:  95.54  -time: 453.45
Iteration: 24/50[==============] -Error: 0.0786231609  -Training_Accuracy:  95.75  -time: 470.96
Iteration: 25/50[==============] -Error: 0.0791899732  -Training_Accuracy:  95.68  -time: 489.44
Iteration: 26/50[==============] -Error: 0.0778288648  -Training_Accuracy:  95.52  -time: 509.08
Iteration: 27/50[==============] -Error: 0.0762033255  -Training_Accuracy:  96.06  -time: 527.17
Iteration: 28/50[==============] -Error: 0.0760206298  -Training_Accuracy:  96.01  -time: 544.70
Iteration: 29/50[==============] -Error: 0.0740905061  -Training_Accuracy:  95.74  -time: 562.49
Iteration: 30/50[==============] -Error: 0.0740106910  -Training_Accuracy:  95.75  -time: 584.48
Iteration: 31/50[==============] -Error: 0.0728092107  -Training_Accuracy:  96.24  -time: 605.67
Iteration: 32/50[==============] -Error: 0.0734913606  -Training_Accuracy:  95.99  -time: 626.83
Iteration: 33/50[==============] -Error: 0.0727898826  -Training_Accuracy:  96.26  -time: 644.69
Iteration: 34/50[==============] -Error: 0.0729246583  -Training_Accuracy:  96.08  -time: 662.22
Iteration: 35/50[==============] -Error: 0.0708363154  -Training_Accuracy:  96.19  -time: 679.85
Iteration: 36/50[==============] -Error: 0.0705093099  -Training_Accuracy:  96.29  -time: 697.67
Iteration: 37/50[==============] -Error: 0.0704495636  -Training_Accuracy:  96.41  -time: 717.82
Iteration: 38/50[==============] -Error: 0.0718494799  -Training_Accuracy:  96.00  -time: 736.65
Iteration: 39/50[==============] -Error: 0.0712456469  -Training_Accuracy:  96.26  -time: 754.07
Iteration: 40/50[==============] -Error: 0.0680997413  -Training_Accuracy:  96.41  -time: 771.77
Iteration: 41/50[==============] -Error: 0.0702789943  -Training_Accuracy:  96.33  -time: 789.40
Iteration: 42/50[==============] -Error: 0.0696876296  -Training_Accuracy:  96.47  -time: 807.04
Iteration: 43/50[==============] -Error: 0.0667998391  -Training_Accuracy:  96.34  -time: 824.41
Iteration: 44/50[==============] -Error: 0.0659739168  -Training_Accuracy:  96.08  -time: 841.98
```


Learning Curve_Error
Error
Epoch
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0
10
20
30
40
50


Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
48500
48000
47500
47000
46500
46000
45500
0
10
20
30
40
50


Learning Curve_Validation_Accuracy
Validation_Accuracy
9500
9450
9400
9350
9300
9250
9200
9150
9100
0
10
20
30
40
50
Epoch


Learning Curve_Test_Accuracy
Test_Accuracy
9550
9500
9450
9400
9350
9300
9250
9200
9150
0
10
20
30
40
50
Epoch

Comment

The first thing to notice is that the error is decreasing throughout iterations, which is normal while training a network. We can notice also that the Training accuracy has an increasing allure but has small ups and downs throughout iterations.

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 [29]:
```
#Your implementation goes here
import os
import numpy as np
def guess(im) :
    path = '/notebooks/elkhamar/TP1/Images_test/'
    pic= misc.imread(os.path.join(path, im ), flatten= True)
    image = misc.imresize(pic,(28,28))
    image = image.reshape(1,784)
    prediction = np.argmax( my_mnist_net.feedForward(image) )

    return prediction
```

In [30]:
```
print ('the network guess the digit 4 as', guess('4.bmp'))
print ('the network guess the digit 5 as', guess('5.bmp'))
print ('the network guess the digit 9 as', guess('9.bmp'))
```

```
('the network guess the digit 4 as', 5)
('the network guess the digit 5 as', 5)
('the network guess the digit 9 as', 0)
```

Comment

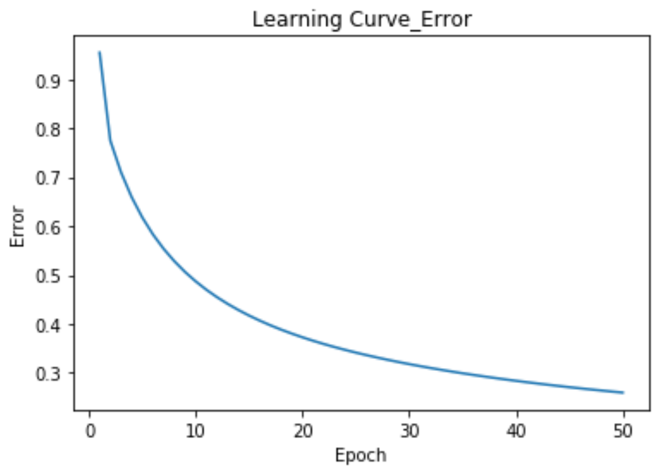
At first, the fact that the Neural Network recognises correctly the number 5 but not the number 4 and 9 seems weird. But when we've taken a look at the '4.bmp' , '5.bmp' , and '9.bmp' pictures, we understood why. Actually, '5.bmp'is the only picture among the three that is written in white on a black background, and this is actually the type of pictures the network has been trained on.

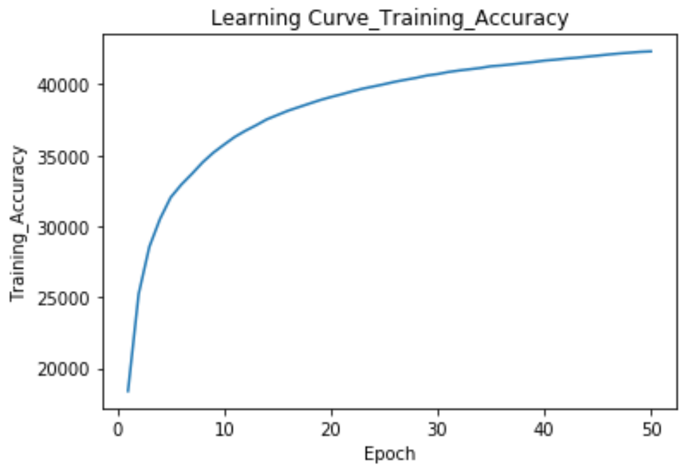
Part 2: Change the neural network structure and parameters to optimize performance

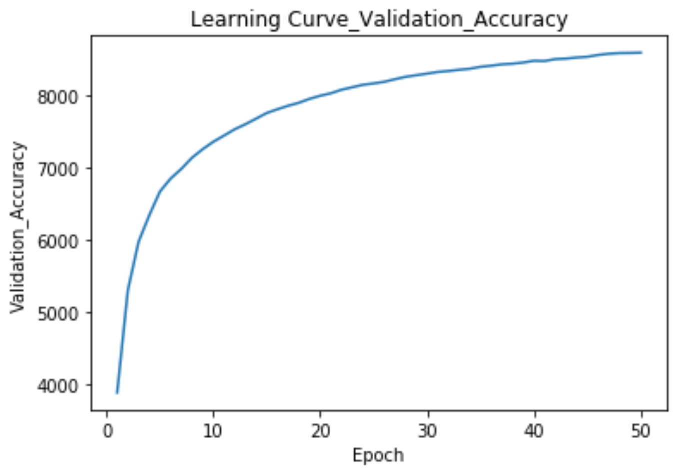
Question 2.2.1: Change the learning rate (0.001, 0.1, 1.0 , 10). Train the new neural nets with the original specifications (Part 2.1), for 50 iterations. Plot test accuracy vs iteration for each learning rate on the same graph. Report the maximum test accuracy achieved for each learning rate. Which one achieves the maximum test accuracy?

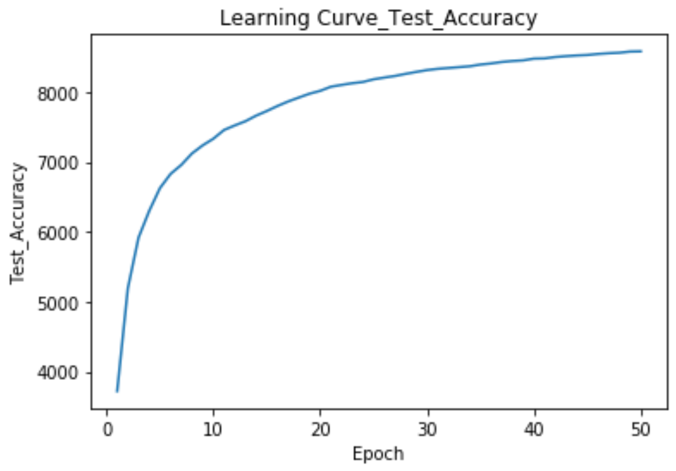
In [7]:
```
#Your implementation with a learning rate of 0.001 goes here

my_net_2 = NeuralNetwork(784,30,10,iterations = 50, learning_rate = 0.001)

my_net_2.train(training_data,validation_data,test_data)

my_net_2.save('Models/NET_784_30_10_0.001')
```

```
Iteration:  1/50[==============] -Error: 0.9546645863  -Training_Accuracy:  36.79  -time: 18.63
Iteration:  2/50[==============] -Error: 0.7748265747  -Training_Accuracy:  50.48  -time: 38.57
Iteration:  3/50[==============] -Error: 0.7102855346  -Training_Accuracy:  57.11  -time: 56.10
Iteration:  4/50[==============] -Error: 0.6592362701  -Training_Accuracy:  61.04  -time: 73.69
Iteration:  5/50[==============] -Error: 0.6175817082  -Training_Accuracy:  64.08  -time: 90.92
Iteration:  6/50[==============] -Error: 0.5828158703  -Training_Accuracy:  65.88  -time: 108.43
Iteration:  7/50[==============] -Error: 0.5533952856  -Training_Accuracy:  67.41  -time: 126.10
Iteration:  8/50[==============] -Error: 0.5281723099  -Training_Accuracy:  69.03  -time: 143.39
Iteration:  9/50[==============] -Error: 0.5061876548  -Training_Accuracy:  70.38  -time: 162.91
Iteration: 10/50[==============] -Error: 0.4869345047  -Training_Accuracy:  71.51  -time: 182.71
Iteration: 11/50[==============] -Error: 0.4698101216  -Training_Accuracy:  72.59  -time: 200.97
Iteration: 12/50[==============] -Error: 0.4545650828  -Training_Accuracy:  73.46  -time: 220.30
Iteration: 13/50[==============] -Error: 0.4408081722  -Training_Accuracy:  74.23  -time: 239.40
Iteration: 14/50[==============] -Error: 0.4283749955  -Training_Accuracy:  75.07  -time: 257.10
Iteration: 15/50[==============] -Error: 0.4170072945  -Training_Accuracy:  75.68  -time: 275.53
Iteration: 16/50[==============] -Error: 0.4066334223  -Training_Accuracy:  76.29  -time: 297.28
Iteration: 17/50[==============] -Error: 0.3970526304  -Training_Accuracy:  76.79  -time: 318.72
Iteration: 18/50[==============] -Error: 0.3882549757  -Training_Accuracy:  77.29  -time: 338.86
Iteration: 19/50[==============] -Error: 0.3801205342  -Training_Accuracy:  77.79  -time: 356.32
Iteration: 20/50[==============] -Error: 0.3725304899  -Training_Accuracy:  78.21  -time: 373.79
Iteration: 21/50[==============] -Error: 0.3654345101  -Training_Accuracy:  78.60  -time: 391.09
Iteration: 22/50[==============] -Error: 0.3588278265  -Training_Accuracy:  79.01  -time: 408.88
Iteration: 23/50[==============] -Error: 0.3526108318  -Training_Accuracy:  79.40  -time: 426.32
Iteration: 24/50[==============] -Error: 0.3467981932  -Training_Accuracy:  79.70  -time: 443.64
Iteration: 25/50[==============] -Error: 0.3412582668  -Training_Accuracy:  80.02  -time: 464.10
Iteration: 26/50[==============] -Error: 0.3360827713  -Training_Accuracy:  80.36  -time: 481.84
Iteration: 27/50[==============] -Error: 0.3311566388  -Training_Accuracy:  80.65  -time: 499.17
Iteration: 28/50[==============] -Error: 0.3264472564  -Training_Accuracy:  80.92  -time: 516.34
Iteration: 29/50[==============] -Error: 0.3219722208  -Training_Accuracy:  81.25  -time: 533.39
Iteration: 30/50[==============] -Error: 0.3177161316  -Training_Accuracy:  81.45  -time: 550.69
Iteration: 31/50[==============] -Error: 0.3136410000  -Training_Accuracy:  81.74  -time: 568.04
Iteration: 32/50[==============] -Error: 0.3097493840  -Training_Accuracy:  81.95  -time: 585.48
Iteration: 33/50[==============] -Error: 0.3060153719  -Training_Accuracy:  82.13  -time: 602.91
Iteration: 34/50[==============] -Error: 0.3024156646  -Training_Accuracy:  82.30  -time: 620.22
Iteration: 35/50[==============] -Error: 0.2989666655  -Training_Accuracy:  82.55  -time: 637.95
Iteration: 36/50[==============] -Error: 0.2956721777  -Training_Accuracy:  82.67  -time: 656.83
Iteration: 37/50[==============] -Error: 0.2924710503  -Training_Accuracy:  82.83  -time: 674.30
Iteration: 38/50[==============] -Error: 0.2893722868  -Training_Accuracy:  82.99  -time: 691.73
Iteration: 39/50[==============] -Error: 0.2863910421  -Training_Accuracy:  83.15  -time: 710.83
Iteration: 40/50[==============] -Error: 0.2835380953  -Training_Accuracy:  83.35  -time: 728.43
Iteration: 41/50[==============] -Error: 0.2807647471  -Training_Accuracy:  83.47  -time: 745.79
Iteration: 42/50[==============] -Error: 0.2780733669  -Training_Accuracy:  83.63  -time: 763.00
Iteration: 43/50[==============] -Error: 0.2754824314  -Training_Accuracy:  83.75  -time: 780.76
Iteration: 44/50[==============] -Error: 0.2729840529  -Training_Accuracy:  83.90  -time: 798.26
```


Learning Curve_Error
Error
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50


Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
40000
35000
30000
25000
20000
0
10
20
30
40
50

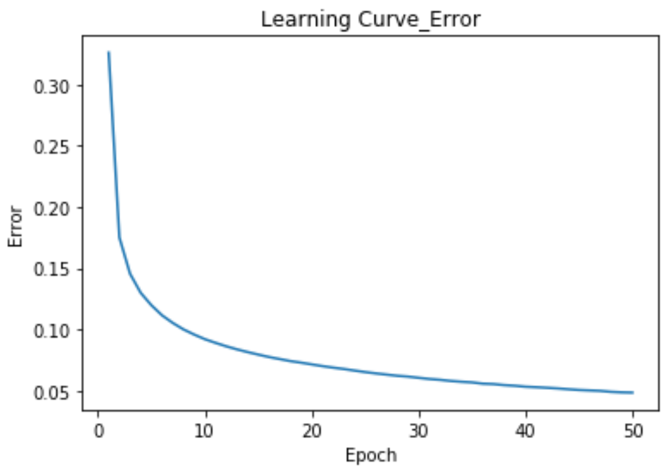

Learning Curve_Validation_Accuracy
Validation_Accuracy
4000
5000
6000
7000
8000
0
10
20
30
40
50
Epoch

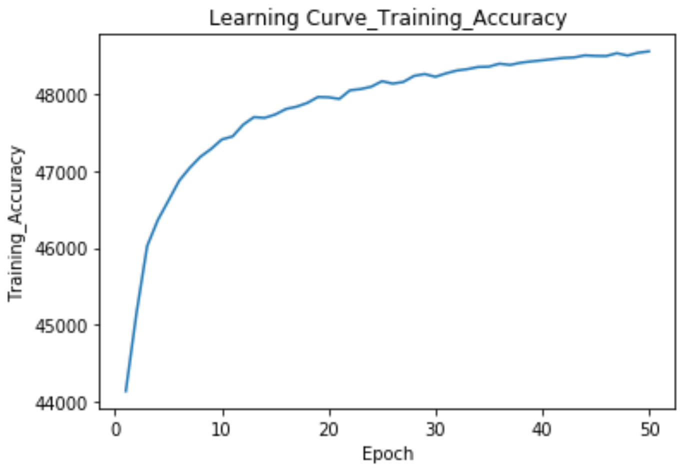

Learning Curve_Test_Accuracy
Test_Accuracy
4000
5000
6000
7000
8000
0
10
20
30
40
50
Epoch

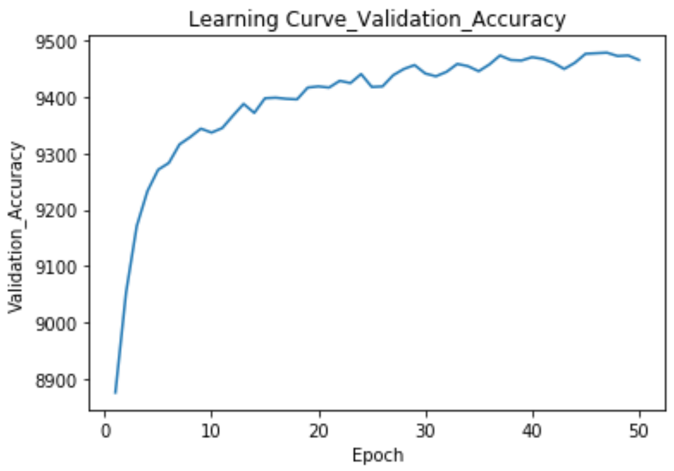
In [8]:

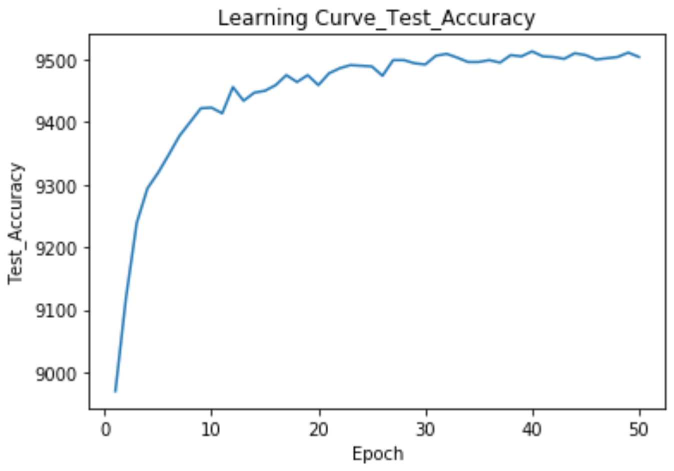
```
#Your implementation with a learning rate of 0.1 goes here

my_net_3 = NeuralNetwork(784,30,10,iterations = 50, learning_rate = 0.1)

my_net_3.train(training_data,validation_data,test_data)

my_net_3.save('Models/NET_784_30_10_0.1')
```

```
Iteration:  1/50[==============] -Error: 0.3260575001  -Training_Accuracy:  88.26  -time: 18.72
Iteration:  2/50[==============] -Error: 0.1751931103  -Training_Accuracy:  90.30  -time: 37.03
Iteration:  3/50[==============] -Error: 0.1455235146  -Training_Accuracy:  92.05  -time: 55.09
Iteration:  4/50[==============] -Error: 0.1299195928  -Training_Accuracy:  92.73  -time: 72.75
Iteration:  5/50[==============] -Error: 0.1198021188  -Training_Accuracy:  93.24  -time: 90.40
Iteration:  6/50[==============] -Error: 0.1115586053  -Training_Accuracy:  93.75  -time: 110.36
Iteration:  7/50[==============] -Error: 0.1053728066  -Training_Accuracy:  94.09  -time: 130.91
Iteration:  8/50[==============] -Error: 0.1000958948  -Training_Accuracy:  94.38  -time: 150.11
Iteration:  9/50[==============] -Error: 0.0958923625  -Training_Accuracy:  94.59  -time: 169.89
Iteration: 10/50[==============] -Error: 0.0921147170  -Training_Accuracy:  94.83  -time: 189.00
Iteration: 11/50[==============] -Error: 0.0891213559  -Training_Accuracy:  94.91  -time: 207.03
Iteration: 12/50[==============] -Error: 0.0863151549  -Training_Accuracy:  95.22  -time: 224.52
Iteration: 13/50[==============] -Error: 0.0838245506  -Training_Accuracy:  95.41  -time: 243.01
Iteration: 14/50[==============] -Error: 0.0815883357  -Training_Accuracy:  95.39  -time: 264.74
Iteration: 15/50[==============] -Error: 0.0795380595  -Training_Accuracy:  95.48  -time: 282.54
Iteration: 16/50[==============] -Error: 0.0775718387  -Training_Accuracy:  95.63  -time: 302.39
Iteration: 17/50[==============] -Error: 0.0759047584  -Training_Accuracy:  95.69  -time: 321.85
Iteration: 18/50[==============] -Error: 0.0742277186  -Training_Accuracy:  95.78  -time: 339.89
Iteration: 19/50[==============] -Error: 0.0729699371  -Training_Accuracy:  95.94  -time: 357.49
Iteration: 20/50[==============] -Error: 0.0715413057  -Training_Accuracy:  95.93  -time: 374.95
Iteration: 21/50[==============] -Error: 0.0701288010  -Training_Accuracy:  95.89  -time: 393.03
Iteration: 22/50[==============] -Error: 0.0689024612  -Training_Accuracy:  96.11  -time: 410.78
Iteration: 23/50[==============] -Error: 0.0677880580  -Training_Accuracy:  96.15  -time: 428.87
Iteration: 24/50[==============] -Error: 0.0665125595  -Training_Accuracy:  96.21  -time: 446.57
Iteration: 25/50[==============] -Error: 0.0652694186  -Training_Accuracy:  96.35  -time: 464.45
Iteration: 26/50[==============] -Error: 0.0641885006  -Training_Accuracy:  96.29  -time: 482.31
Iteration: 27/50[==============] -Error: 0.0632141256  -Training_Accuracy:  96.33  -time: 500.05
Iteration: 28/50[==============] -Error: 0.0622394028  -Training_Accuracy:  96.49  -time: 517.62
Iteration: 29/50[==============] -Error: 0.0615784710  -Training_Accuracy:  96.54  -time: 535.47
Iteration: 30/50[==============] -Error: 0.0605817431  -Training_Accuracy:  96.47  -time: 553.23
Iteration: 31/50[==============] -Error: 0.0596394973  -Training_Accuracy:  96.56  -time: 570.99
Iteration: 32/50[==============] -Error: 0.0589528076  -Training_Accuracy:  96.63  -time: 590.48
Iteration: 33/50[==============] -Error: 0.0580259015  -Training_Accuracy:  96.67  -time: 607.87
Iteration: 34/50[==============] -Error: 0.0573743247  -Training_Accuracy:  96.72  -time: 625.70
Iteration: 35/50[==============] -Error: 0.0568552563  -Training_Accuracy:  96.73  -time: 643.52
Iteration: 36/50[==============] -Error: 0.0558059584  -Training_Accuracy:  96.81  -time: 661.35
Iteration: 37/50[==============] -Error: 0.0554658156  -Training_Accuracy:  96.78  -time: 678.80
Iteration: 38/50[==============] -Error: 0.0546317990  -Training_Accuracy:  96.83  -time: 697.96
Iteration: 39/50[==============] -Error: 0.0540996728  -Training_Accuracy:  96.87  -time: 717.53
Iteration: 40/50[==============] -Error: 0.0533241529  -Training_Accuracy:  96.90  -time: 736.57
Iteration: 41/50[==============] -Error: 0.0529205874  -Training_Accuracy:  96.93  -time: 755.36
Iteration: 42/50[==============] -Error: 0.0524353538  -Training_Accuracy:  96.96  -time: 774.54
Iteration: 43/50[==============] -Error: 0.0519219265  -Training_Accuracy:  96.97  -time: 792.26
Iteration: 44/50[==============] -Error: 0.0512635016  -Training_Accuracy:  97.03  -time: 810.22
```


Learning Curve_Error
Error
0.30
0.25
0.20
0.15
0.10
0.05
0
10
20
30
40
50
Epoch


Learning Curve_Training_Accuracy
Training_Accuracy
48000
47000
46000
45000
44000
0
10
20
30
40
50
Epoch

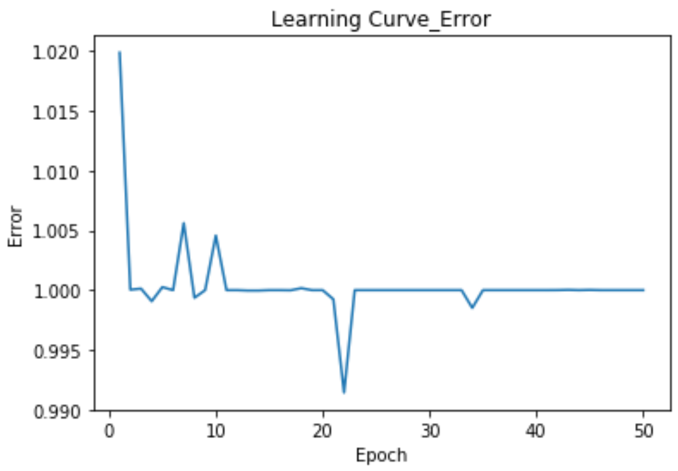

Learning Curve_Validation_Accuracy
Validation_Accuracy
Epoch
9500
9400
9300
9200
9100
9000
8900
0
10
20
30
40
50

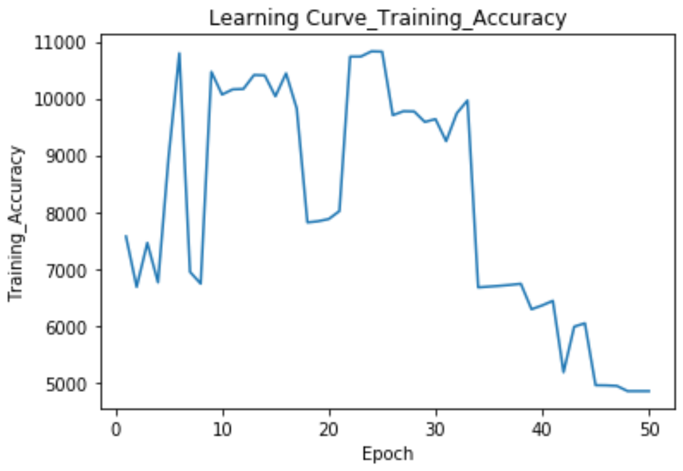

Learning Curve_Test_Accuracy
Test_Accuracy
Epoch
9500
9400
9300
9200
9100
9000
0
10
20
30
40
50

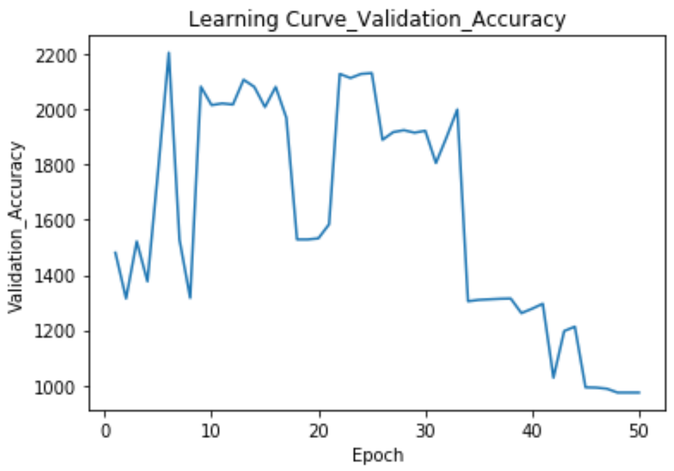
In [9]:
```
#Your implementation with a learning rate of 10 goes here


my_net_5 = NeuralNetwork(784,30,10,iterations = 50, learning_rate = 10)

my_net_5.train(training_data,validation_data,test_data)


my_net_5.save('Models/NET_784_30_10_10')
```

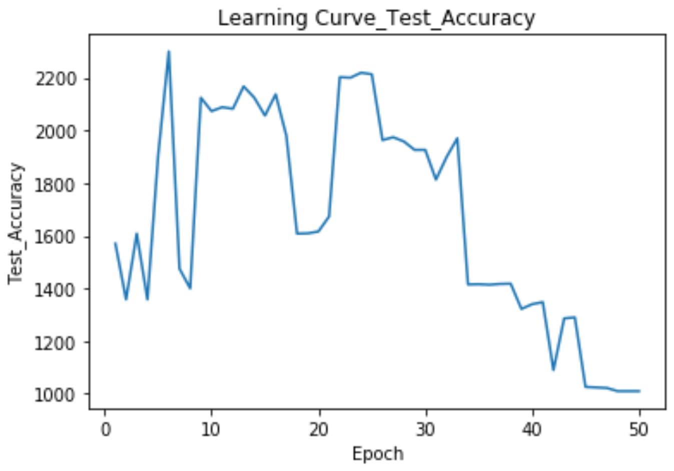
```
Iteration:  1/50[==============] -Error: 1.0198895170  -Training_Accuracy:  15.14  -time: 18.54
Iteration:  2/50[==============] -Error: 1.0000367030  -Training_Accuracy:  13.36  -time: 36.16
Iteration:  3/50[==============] -Error: 1.0001319551  -Training_Accuracy:  14.91  -time: 54.87

transfer_functions.py:7: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))
```

```
Iteration:  4/50[==============] -Error: 0.9990715188  -Training_Accuracy:  13.52  -time: 75.34
Iteration:  5/50[==============] -Error: 1.0002570902  -Training_Accuracy:  17.96  -time: 95.15
Iteration:  6/50[==============] -Error: 0.9999974584  -Training_Accuracy:  21.60  -time: 113.64
Iteration:  7/50[==============] -Error: 1.0056173602  -Training_Accuracy:  13.89  -time: 132.02
Iteration:  8/50[==============] -Error: 0.9993578818  -Training_Accuracy:  13.47  -time: 150.22
Iteration:  9/50[==============] -Error: 1.0000005797  -Training_Accuracy:  20.94  -time: 168.05
Iteration: 10/50[==============] -Error: 1.0045760118  -Training_Accuracy:  20.14  -time: 185.61
Iteration: 11/50[==============] -Error: 0.9999991897  -Training_Accuracy:  20.32  -time: 203.34
Iteration: 12/50[==============] -Error: 0.9999995777  -Training_Accuracy:  20.33  -time: 221.23
Iteration: 13/50[==============] -Error: 0.9999657106  -Training_Accuracy:  20.83  -time: 238.86
Iteration: 14/50[==============] -Error: 0.9999663323  -Training_Accuracy:  20.81  -time: 256.33
Iteration: 15/50[==============] -Error: 1.0000001113  -Training_Accuracy:  20.08  -time: 273.71
Iteration: 16/50[==============] -Error: 0.9999993411  -Training_Accuracy:  20.88  -time: 291.33
Iteration: 17/50[==============] -Error: 0.9999879867  -Training_Accuracy:  19.65  -time: 308.92
Iteration: 18/50[==============] -Error: 1.0001667123  -Training_Accuracy:  15.63  -time: 326.70
Iteration: 19/50[==============] -Error: 0.9999998342  -Training_Accuracy:  15.67  -time: 344.77
Iteration: 20/50[==============] -Error: 0.9999997566  -Training_Accuracy:  15.75  -time: 362.43
Iteration: 21/50[==============] -Error: 0.9992352895  -Training_Accuracy:  16.03  -time: 380.68
Iteration: 22/50[==============] -Error: 0.9914216681  -Training_Accuracy:  21.47  -time: 400.71
Iteration: 23/50[==============] -Error: 0.9999999139  -Training_Accuracy:  21.48  -time: 419.45
Iteration: 24/50[==============] -Error: 0.9999999073  -Training_Accuracy:  21.66  -time: 438.86
Iteration: 25/50[==============] -Error: 0.9999998993  -Training_Accuracy:  21.65  -time: 458.40
Iteration: 26/50[==============] -Error: 0.9999998895  -Training_Accuracy:  19.41  -time: 477.95
Iteration: 27/50[==============] -Error: 0.9999998771  -Training_Accuracy:  19.55  -time: 499.09
Iteration: 28/50[==============] -Error: 0.9999998608  -Training_Accuracy:  19.54  -time: 519.93
Iteration: 29/50[==============] -Error: 0.9999998384  -Training_Accuracy:  19.17  -time: 538.32
Iteration: 30/50[==============] -Error: 0.9999998053  -Training_Accuracy:  19.27  -time: 557.81
Iteration: 31/50[==============] -Error: 0.9999997503  -Training_Accuracy:  18.49  -time: 577.30
Iteration: 32/50[==============] -Error: 0.9999996376  -Training_Accuracy:  19.48  -time: 598.00
Iteration: 33/50[==============] -Error: 0.9999992268  -Training_Accuracy:  19.93  -time: 617.92
Iteration: 34/50[==============] -Error: 0.9985116854  -Training_Accuracy:  13.34  -time: 636.78
Iteration: 35/50[==============] -Error: 0.9999998915  -Training_Accuracy:  13.37  -time: 656.35
Iteration: 36/50[==============] -Error: 0.9999998771  -Training_Accuracy:  13.40  -time: 674.67
Iteration: 37/50[==============] -Error: 0.9999998574  -Training_Accuracy:  13.43  -time: 694.29
Iteration: 38/50[==============] -Error: 0.9999998287  -Training_Accuracy:  13.47  -time: 713.31
Iteration: 39/50[==============] -Error: 0.9999997823  -Training_Accuracy:  12.57  -time: 731.15
Iteration: 40/50[==============] -Error: 0.9999996912  -Training_Accuracy:  12.71  -time: 749.08
Iteration: 41/50[==============] -Error: 0.9999993699  -Training_Accuracy:  12.87  -time: 766.86
Iteration: 42/50[==============] -Error: 1.0000040265  -Training_Accuracy:  10.35  -time: 784.49
Iteration: 43/50[==============] -Error: 1.0000235025  -Training_Accuracy:  11.96  -time: 802.08
Iteration: 44/50[==============] -Error: 0.9999996768  -Training_Accuracy:  12.08  -time: 820.21
Iteration: 45/50[==============] -Error: 1.0000185250  -Training_Accuracy:  9.89  -time: 838.04
Iteration: 46/50[==============] -Error: 0.9999999307  -Training_Accuracy:  9.89  -time: 855.70
Iteration: 47/50[==============] -Error: 0.9999999208  -Training_Accuracy:  9.87  -time: 873.29
```


Learning Curve_Error
Error
Epoch
1.020
1.015
1.010
1.005
1.000
0.995
0.990
0
10
20
30
40
50

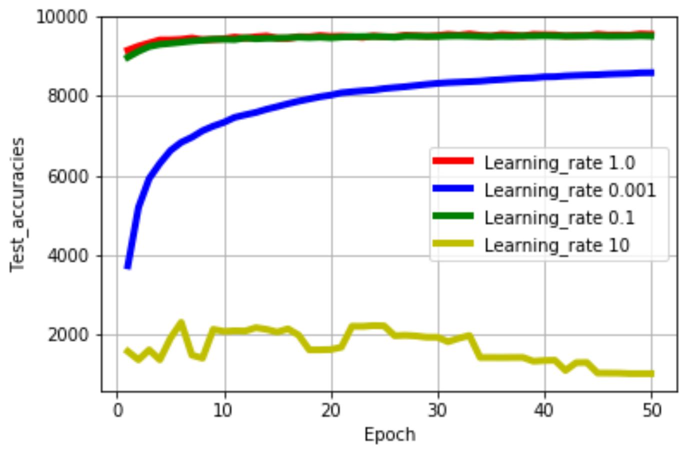

Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
11000
10000
9000
8000
7000
6000
5000
0
10
20
30
40
50


Learning Curve_Validation_Accuracy
Validation_Accuracy
Epoch
2200
2000
1800
1600
1400
1200
1000
0
10
20
30
40
50


Learning Curve_Test_Accuracy
Test_Accuracy
Epoch
2200
2000
1800
1600
1400
1200
1000
0
10
20
30
40
50

In [27]:
```
import matplotlib.pyplot as plt
plt.grid(True)
plt.plot(range(1,my_mnist_net.iterations+1), my_mnist_net.Test_accuracies,color ='r',linewidth = '4', Label
= 'Learning_rate 1.0 ')
plt.hold(True)
plt.plot(range(1,my_net_2.iterations+1), my_net_2.Test_accuracies,color ='b',linewidth = '4',Label = 'Learn
ing_rate 0.001 ')
plt.plot(range(1,my_net_3.iterations+1), my_net_3.Test_accuracies,color ='g',linewidth = '4',Label = 'Learn
ing_rate 0.1 ')
plt.plot(range(1,my_net_5.iterations+1), my_net_5.Test_accuracies,color ='y',linewidth = '4',Label = 'Learn
ing_rate 10 ')
plt.xlabel('Epoch')
plt.ylabel('Test_accuracies')
plt.grid(True)
plt.legend()
plt.show()
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel/__main__.py:4: MatplotlibDeprecationWarning: pyplot.hold i
s deprecated.
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
```



In [21]:
```
print('The maximum test accuracy value for the learning rate 0.001 is',max(my_net_2.Test_accuracies))
print('The maximum test accuracy value for the learning rate 0.1 is',max(my_net_3.Test_accuracies))
print('The maximum test accuracy value for the learning rate 1.0 is',max(my_mnist_net.Test_accuracies))
print('The maximum test accuracy value for the learning rate 10 is',max(my_net_5.Test_accuracies))
```

```
('The maximum test accuracy value for the learning rate 0.001 is', 8582.0)
('The maximum test accuracy value for the learning rate 0.1 is', 9513.0)
('The maximum test accuracy value for the learning rate 1.0 is', 9557.0)
('The maximum test accuracy value for the learning rate 10 is', 2300.0)
```

Maximum Test_accuracy

The maximum Test_accuracy is 9557.0 and it is achieved with the learning rate of 1.0

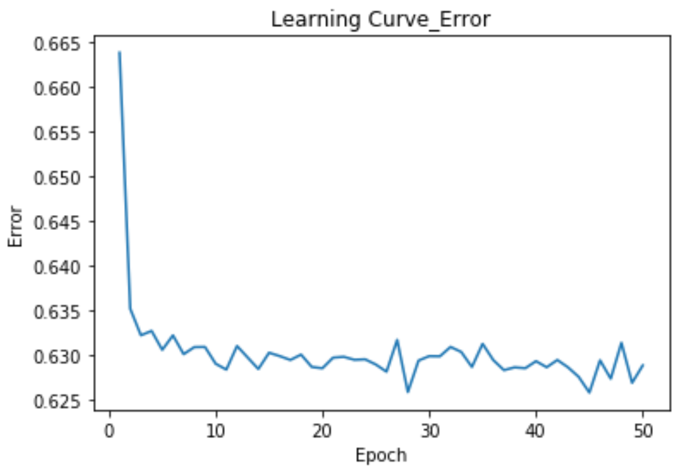
Comment

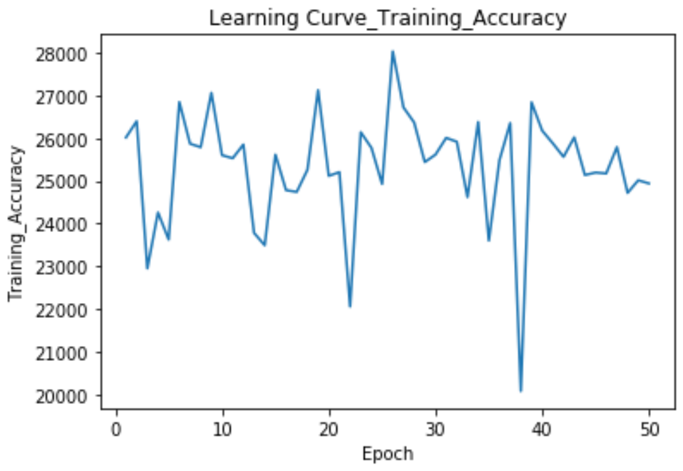
We can notice that the best learning rates are 1.0 or 0.1 . When the learning rate is too low (0.001) , the maximum test accuracy is lower which may be due to the overfitting while training. When the learning rate is too high (10), the update of the weights in every step is too important and the algorithm diverge. Thus, the maximum test accuracy in this case is too low.

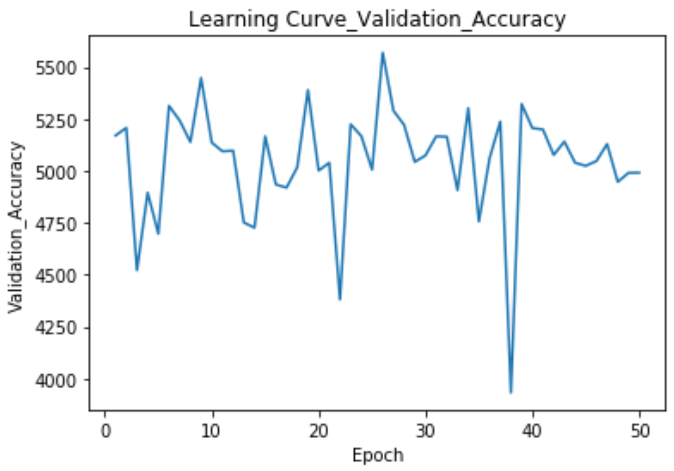
Question 2.2.2 : initialize all weights to 0. Plot the training accuracy curve. Comment your results

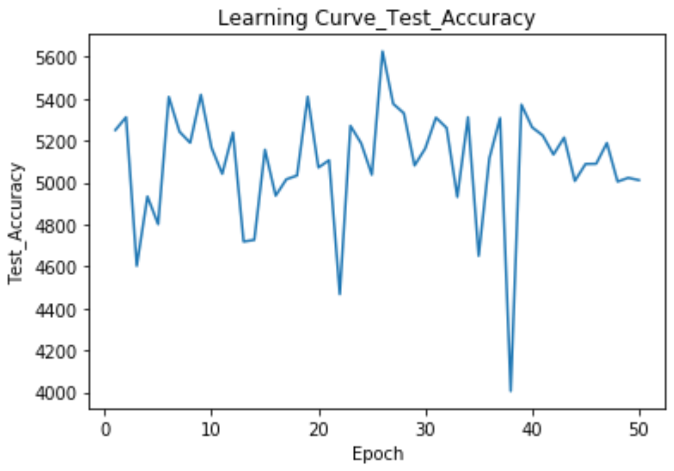
In [4]:
```
#Your implementation goes here
from NeuralNetwork import *
my_net= NeuralNetwork(784,30,10,iterations = 50, learning_rate = 1.0)

wi=np.zeros((785,30))
wo=np.zeros((31,10))
my_net.weights_initialisation(wi,wo)

my_net.train(training_data,validation_data,test_data)

my_net.save('Models/NET_init0_784_30_10_1.0')
```

```
Iteration:  1/50[==============] -Error: 0.6638017006  -Training_Accuracy:  52.03  -time: 17.25
Iteration:  2/50[==============] -Error: 0.6351524598  -Training_Accuracy:  52.79  -time: 34.49
Iteration:  3/50[==============] -Error: 0.6321809492  -Training_Accuracy:  45.89  -time: 52.94
Iteration:  4/50[==============] -Error: 0.6326718735  -Training_Accuracy:  48.51  -time: 71.59
Iteration:  5/50[==============] -Error: 0.6305511868  -Training_Accuracy:  47.25  -time: 88.92
Iteration:  6/50[==============] -Error: 0.6321737852  -Training_Accuracy:  53.69  -time: 106.18
Iteration:  7/50[==============] -Error: 0.6300718538  -Training_Accuracy:  51.74  -time: 123.38
Iteration:  8/50[==============] -Error: 0.6308581845  -Training_Accuracy:  51.57  -time: 140.55
Iteration:  9/50[==============] -Error: 0.6308791352  -Training_Accuracy:  54.12  -time: 157.50
Iteration: 10/50[==============] -Error: 0.6290007982  -Training_Accuracy:  51.20  -time: 174.85
Iteration: 11/50[==============] -Error: 0.6283421605  -Training_Accuracy:  51.05  -time: 192.12
Iteration: 12/50[==============] -Error: 0.6309857468  -Training_Accuracy:  51.70  -time: 209.25
Iteration: 13/50[==============] -Error: 0.6296946258  -Training_Accuracy:  47.56  -time: 226.27
Iteration: 14/50[==============] -Error: 0.6284092869  -Training_Accuracy:  46.98  -time: 243.53
Iteration: 15/50[==============] -Error: 0.6302313435  -Training_Accuracy:  51.22  -time: 260.76
Iteration: 16/50[==============] -Error: 0.6298556144  -Training_Accuracy:  49.56  -time: 277.87
Iteration: 17/50[==============] -Error: 0.6294222675  -Training_Accuracy:  49.48  -time: 294.97
Iteration: 18/50[==============] -Error: 0.6300220772  -Training_Accuracy:  50.51  -time: 311.89
Iteration: 19/50[==============] -Error: 0.6286411988  -Training_Accuracy:  54.24  -time: 329.05
Iteration: 20/50[==============] -Error: 0.6284932664  -Training_Accuracy:  50.23  -time: 346.41
Iteration: 21/50[==============] -Error: 0.6296826037  -Training_Accuracy:  50.40  -time: 363.52
Iteration: 22/50[==============] -Error: 0.6297732154  -Training_Accuracy:  44.11  -time: 380.92
Iteration: 23/50[==============] -Error: 0.6294388749  -Training_Accuracy:  52.27  -time: 398.46
Iteration: 24/50[==============] -Error: 0.6294932910  -Training_Accuracy:  51.54  -time: 415.64
Iteration: 25/50[==============] -Error: 0.6289130874  -Training_Accuracy:  49.85  -time: 432.86
Iteration: 26/50[==============] -Error: 0.6281087446  -Training_Accuracy:  56.06  -time: 449.74
Iteration: 27/50[==============] -Error: 0.6316674852  -Training_Accuracy:  53.44  -time: 466.82
Iteration: 28/50[==============] -Error: 0.6258288129  -Training_Accuracy:  52.73  -time: 484.16
Iteration: 29/50[==============] -Error: 0.6293570602  -Training_Accuracy:  50.87  -time: 501.44
Iteration: 30/50[==============] -Error: 0.6298508803  -Training_Accuracy:  51.22  -time: 518.72
Iteration: 31/50[==============] -Error: 0.6298337061  -Training_Accuracy:  52.01  -time: 535.84
Iteration: 32/50[==============] -Error: 0.6308802720  -Training_Accuracy:  51.83  -time: 553.17
Iteration: 33/50[==============] -Error: 0.6303194316  -Training_Accuracy:  49.23  -time: 570.47
Iteration: 34/50[==============] -Error: 0.6286329462  -Training_Accuracy:  52.75  -time: 587.67
Iteration: 35/50[==============] -Error: 0.6312316071  -Training_Accuracy:  47.20  -time: 604.76
Iteration: 36/50[==============] -Error: 0.6294093993  -Training_Accuracy:  50.98  -time: 622.03
Iteration: 37/50[==============] -Error: 0.6282827410  -Training_Accuracy:  52.70  -time: 640.09
Iteration: 38/50[==============] -Error: 0.6285975517  -Training_Accuracy:  40.15  -time: 657.33
Iteration: 39/50[==============] -Error: 0.6285002691  -Training_Accuracy:  53.68  -time: 674.44
Iteration: 40/50[==============] -Error: 0.6292874268  -Training_Accuracy:  52.35  -time: 691.65
Iteration: 41/50[==============] -Error: 0.6286066060  -Training_Accuracy:  51.75  -time: 709.02
Iteration: 42/50[==============] -Error: 0.6294212408  -Training_Accuracy:  51.12  -time: 726.15
Iteration: 43/50[==============] -Error: 0.6285968477  -Training_Accuracy:  52.03  -time: 743.09
Iteration: 44/50[==============] -Error: 0.6275293042  -Training_Accuracy:  50.27  -time: 760.73
```


Learning Curve_Error
Error
Epoch
0.665
0.660
0.655
0.650
0.645
0.640
0.635
0.630
0.625
0
10
20
30
40
50


Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
28000
27000
26000
25000
24000
23000
22000
21000
20000
0
10
20
30
40
50





```
In [5]: max(my_net.Test_accuracies)

Out[5]: 5625.0
```

Comment

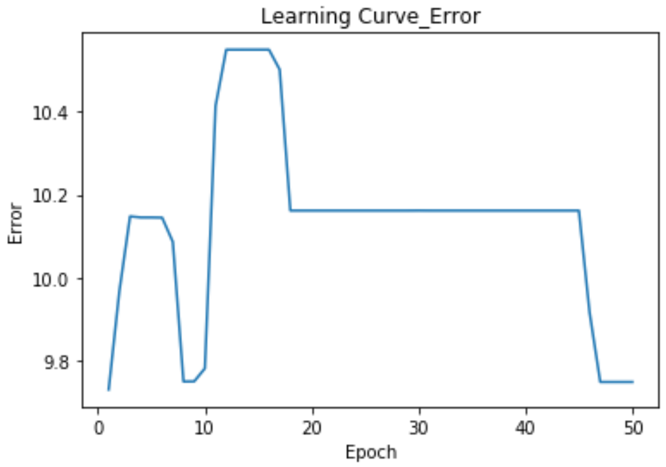
We can notice that the ending error around the last iteration is pretty high compared to the one of the learning of the same network with the same learning rate (1.0) but using a different initialization of the weights (erro of 0.64 vs 0.06). Also, the maximum test_accuracy has decreased (9557.0 vs 5625.0). We can conclude that the initialization of all the weights to zero impacts negativly the learning because in the one hand the network get stuck in local minimas and on the other hand when all the weights are initialized to the same value all the neurons follow the same gradient and end up doing exactly the same things one as other. Thus, The learning doesn't allow to train the network in a relevant way.

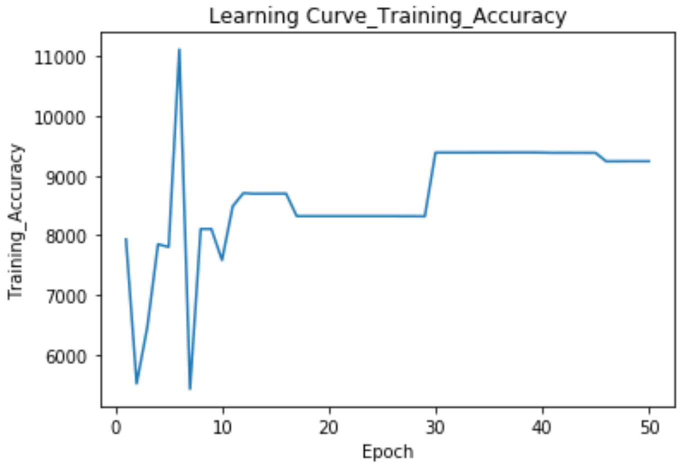
Question 2.2.3 : Try with a different transfer function (such as tanh). File transfer_functions.py provides you the python implementation of the tanh function and its derivative

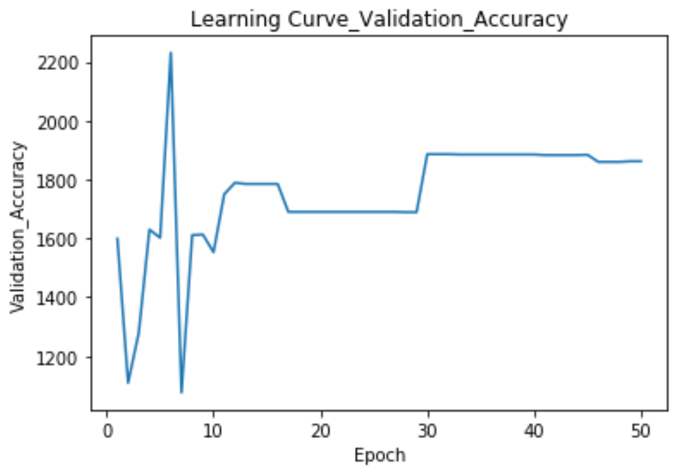
In [4]:

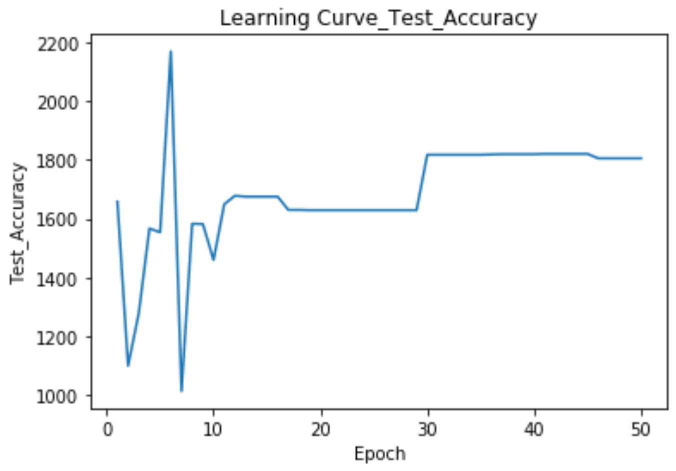
```
#Your implementation goes here : tanh for 1.0
from NeuralNetwork import *

my_net_tanh= NeuralNetwork(784,30,10,iterations = 50, learning_rate = 1.0)

my_net_tanh.train(training_data,validation_data,test_data)

my_net_tanh.save('Models/NET_tanh_784_30_10_1.0')
```

```
Iteration:  1/50[==============] -Error: 9.7294453636  -Training_Accuracy:  15.87  -time: 17.24
Iteration:  2/50[==============] -Error: 9.9673493507  -Training_Accuracy:  11.03  -time: 32.68
Iteration:  3/50[==============] -Error: 10.1478540862  -Training_Accuracy:  12.93  -time: 52.13
Iteration:  4/50[==============] -Error: 10.1452598371  -Training_Accuracy:  15.70  -time: 69.50
Iteration:  5/50[==============] -Error: 10.1452302374  -Training_Accuracy:  15.61  -time: 87.14
Iteration:  6/50[==============] -Error: 10.1448225426  -Training_Accuracy:  22.24  -time: 105.43
Iteration:  7/50[==============] -Error: 10.0863611487  -Training_Accuracy:  10.84  -time: 124.08
Iteration:  8/50[==============] -Error: 9.7489498051  -Training_Accuracy:  16.21  -time: 140.60
Iteration:  9/50[==============] -Error: 9.7491966719  -Training_Accuracy:  16.21  -time: 156.82
Iteration: 10/50[==============] -Error: 9.7805739199  -Training_Accuracy:  15.17  -time: 172.57
Iteration: 11/50[==============] -Error: 10.4148184271  -Training_Accuracy:  16.97  -time: 188.56
Iteration: 12/50[==============] -Error: 10.5505267630  -Training_Accuracy:  17.42  -time: 204.71
Iteration: 13/50[==============] -Error: 10.5505402775  -Training_Accuracy:  17.40  -time: 220.65
Iteration: 14/50[==============] -Error: 10.5505598017  -Training_Accuracy:  17.40  -time: 236.34
Iteration: 15/50[==============] -Error: 10.5505596986  -Training_Accuracy:  17.40  -time: 252.01
Iteration: 16/50[==============] -Error: 10.5505593227  -Training_Accuracy:  17.40  -time: 267.86
Iteration: 17/50[==============] -Error: 10.5024425033  -Training_Accuracy:  16.64  -time: 284.50
Iteration: 18/50[==============] -Error: 10.1618399796  -Training_Accuracy:  16.64  -time: 304.76
Iteration: 19/50[==============] -Error: 10.1618399771  -Training_Accuracy:  16.64  -time: 323.62
Iteration: 20/50[==============] -Error: 10.1618399741  -Training_Accuracy:  16.64  -time: 340.75
Iteration: 21/50[==============] -Error: 10.1618399702  -Training_Accuracy:  16.64  -time: 359.30
Iteration: 22/50[==============] -Error: 10.1618399653  -Training_Accuracy:  16.64  -time: 376.37
Iteration: 23/50[==============] -Error: 10.1618399587  -Training_Accuracy:  16.64  -time: 393.27
Iteration: 24/50[==============] -Error: 10.1618399497  -Training_Accuracy:  16.64  -time: 409.36
Iteration: 25/50[==============] -Error: 10.1618399366  -Training_Accuracy:  16.64  -time: 425.77
Iteration: 26/50[==============] -Error: 10.1618399164  -Training_Accuracy:  16.64  -time: 442.22
Iteration: 27/50[==============] -Error: 10.1618398815  -Training_Accuracy:  16.64  -time: 458.94
Iteration: 28/50[==============] -Error: 10.1618398089  -Training_Accuracy:  16.64  -time: 475.00
Iteration: 29/50[==============] -Error: 10.1618395969  -Training_Accuracy:  16.64  -time: 490.53
Iteration: 30/50[==============] -Error: 10.1619759806  -Training_Accuracy:  18.78  -time: 506.18
Iteration: 31/50[==============] -Error: 10.1618398792  -Training_Accuracy:  18.78  -time: 522.24
Iteration: 32/50[==============] -Error: 10.1618398549  -Training_Accuracy:  18.78  -time: 542.09
Iteration: 33/50[==============] -Error: 10.1618398147  -Training_Accuracy:  18.78  -time: 563.00
Iteration: 34/50[==============] -Error: 10.1618397340  -Training_Accuracy:  18.78  -time: 579.84
Iteration: 35/50[==============] -Error: 10.1618394813  -Training_Accuracy:  18.78  -time: 596.06
Iteration: 36/50[==============] -Error: 10.1618268525  -Training_Accuracy:  18.78  -time: 611.72
Iteration: 37/50[==============] -Error: 10.1618399414  -Training_Accuracy:  18.78  -time: 627.89
Iteration: 38/50[==============] -Error: 10.1618399359  -Training_Accuracy:  18.78  -time: 646.18
Iteration: 39/50[==============] -Error: 10.1618399290  -Training_Accuracy:  18.78  -time: 663.01
Iteration: 40/50[==============] -Error: 10.1618399198  -Training_Accuracy:  18.78  -time: 679.72
Iteration: 41/50[==============] -Error: 10.1618399070  -Training_Accuracy:  18.77  -time: 696.25
Iteration: 42/50[==============] -Error: 10.1618398876  -Training_Accuracy:  18.77  -time: 713.12
Iteration: 43/50[==============] -Error: 10.1618398550  -Training_Accuracy:  18.77  -time: 729.73
Iteration: 44/50[==============] -Error: 10.1618397864  -Training_Accuracy:  18.77  -time: 746.85
```


Learning Curve_Error
Error
Epoch
10.4
10.2
10.0
9.8
0
10
20
30
40
50

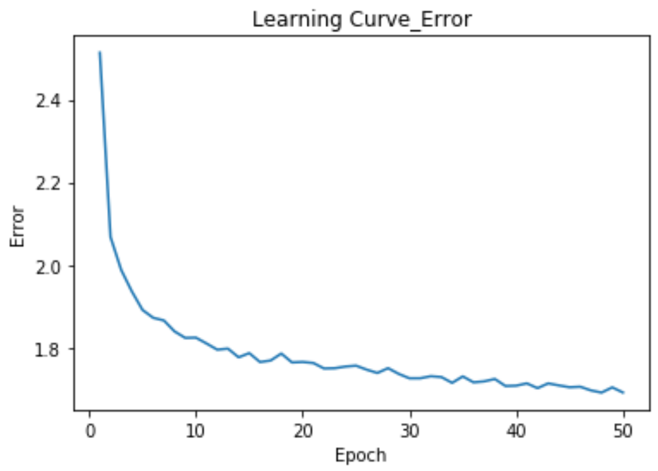

Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
11000
10000
9000
8000
7000
6000
0
10
20
30
40
50

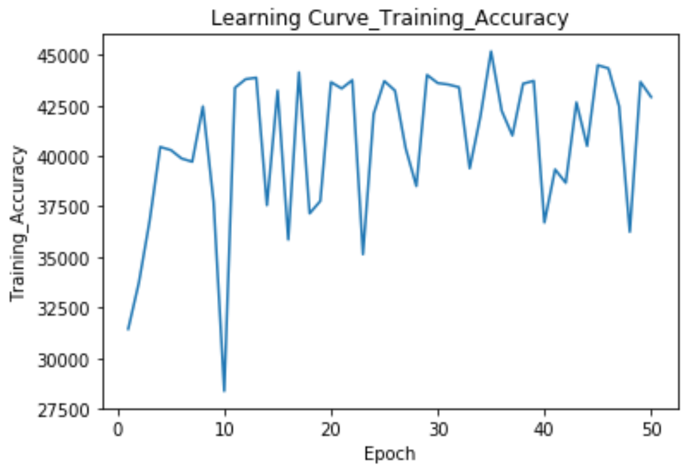

Learning Curve_Validation_Accuracy
Validation_Accuracy
2200
2000
1800
1600
1400
1200
0
10
20
30
40
50
Epoch

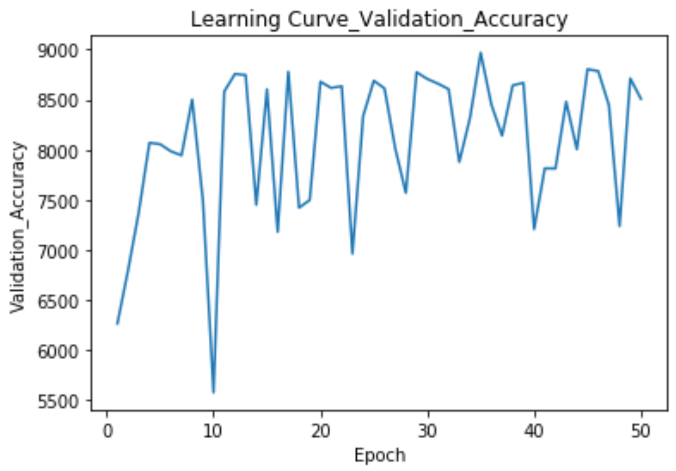

Learning Curve_Test_Accuracy
Test_Accuracy
2200
2000
1800
1600
1400
1200
1000
0
10
20
30
40
50
Epoch

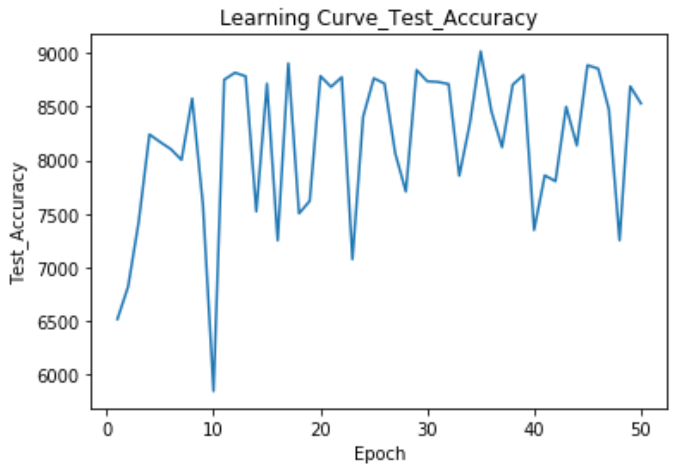
In [3]:
```
#Your implementation goes here : tanh for 0.1

from NeuralNetwork import *

my_net_tanh= NeuralNetwork(784,30,10,iterations = 50, learning_rate = 0.1)

my_net_tanh.train(training_data,validation_data,test_data)

my_net_tanh.save('Models/NET_tanh_784_30_10_1.0')
```

```
Iteration:  1/50[==============] -Error: 2.5134770938  -Training_Accuracy:  62.88  -time: 15.84
Iteration:  2/50[==============] -Error: 2.0678680916  -Training_Accuracy:  67.46  -time: 34.22
Iteration:  3/50[==============] -Error: 1.9881971922  -Training_Accuracy:  73.53  -time: 56.29
Iteration:  4/50[==============] -Error: 1.9361057086  -Training_Accuracy:  80.89  -time: 75.54
Iteration:  5/50[==============] -Error: 1.8913399654  -Training_Accuracy:  80.58  -time: 92.16
Iteration:  6/50[==============] -Error: 1.8723466872  -Training_Accuracy:  79.75  -time: 109.55
Iteration:  7/50[==============] -Error: 1.8659739510  -Training_Accuracy:  79.42  -time: 127.27
Iteration:  8/50[==============] -Error: 1.8394611234  -Training_Accuracy:  84.90  -time: 143.75
Iteration:  9/50[==============] -Error: 1.8237523666  -Training_Accuracy:  75.47  -time: 160.96
Iteration: 10/50[==============] -Error: 1.8243832063  -Training_Accuracy:  56.74  -time: 180.56
Iteration: 11/50[==============] -Error: 1.8101506340  -Training_Accuracy:  86.74  -time: 200.27
Iteration: 12/50[==============] -Error: 1.7952795950  -Training_Accuracy:  87.59  -time: 220.79
Iteration: 13/50[==============] -Error: 1.7978720397  -Training_Accuracy:  87.74  -time: 237.11
Iteration: 14/50[==============] -Error: 1.7768007743  -Training_Accuracy:  75.12  -time: 253.32
Iteration: 15/50[==============] -Error: 1.7869924359  -Training_Accuracy:  86.48  -time: 269.69
Iteration: 16/50[==============] -Error: 1.7653942322  -Training_Accuracy:  71.72  -time: 286.91
Iteration: 17/50[==============] -Error: 1.7690566761  -Training_Accuracy:  88.27  -time: 303.22
Iteration: 18/50[==============] -Error: 1.7859406167  -Training_Accuracy:  74.31  -time: 319.04
Iteration: 19/50[==============] -Error: 1.7644384416  -Training_Accuracy:  75.54  -time: 336.27
Iteration: 20/50[==============] -Error: 1.7655822678  -Training_Accuracy:  87.30  -time: 354.38
Iteration: 21/50[==============] -Error: 1.7631096192  -Training_Accuracy:  86.67  -time: 375.87
Iteration: 22/50[==============] -Error: 1.7498058543  -Training_Accuracy:  87.50  -time: 394.68
Iteration: 23/50[==============] -Error: 1.7504591321  -Training_Accuracy:  70.27  -time: 410.63
Iteration: 24/50[==============] -Error: 1.7544234450  -Training_Accuracy:  84.16  -time: 429.77
Iteration: 25/50[==============] -Error: 1.7566152288  -Training_Accuracy:  87.41  -time: 446.66
Iteration: 26/50[==============] -Error: 1.7469226102  -Training_Accuracy:  86.46  -time: 464.14
Iteration: 27/50[==============] -Error: 1.7389827446  -Training_Accuracy:  80.75  -time: 481.10
Iteration: 28/50[==============] -Error: 1.7505433104  -Training_Accuracy:  77.01  -time: 498.07
Iteration: 29/50[==============] -Error: 1.7367157506  -Training_Accuracy:  88.02  -time: 514.65
Iteration: 30/50[==============] -Error: 1.7260911137  -Training_Accuracy:  87.21  -time: 530.96
Iteration: 31/50[==============] -Error: 1.7261177183  -Training_Accuracy:  87.06  -time: 546.66
Iteration: 32/50[==============] -Error: 1.7310801143  -Training_Accuracy:  86.79  -time: 562.56
Iteration: 33/50[==============] -Error: 1.7290395616  -Training_Accuracy:  78.77  -time: 578.48
Iteration: 34/50[==============] -Error: 1.7150313239  -Training_Accuracy:  83.90  -time: 594.28
Iteration: 35/50[==============] -Error: 1.7307180726  -Training_Accuracy:  90.34  -time: 610.01
Iteration: 36/50[==============] -Error: 1.7163434654  -Training_Accuracy:  84.46  -time: 627.89
Iteration: 37/50[==============] -Error: 1.7186695518  -Training_Accuracy:  82.02  -time: 644.69
Iteration: 38/50[==============] -Error: 1.7244069849  -Training_Accuracy:  87.14  -time: 661.80
Iteration: 39/50[==============] -Error: 1.7073978638  -Training_Accuracy:  87.43  -time: 680.47
Iteration: 40/50[==============] -Error: 1.7082302960  -Training_Accuracy:  73.41  -time: 697.77
Iteration: 41/50[==============] -Error: 1.7137960861  -Training_Accuracy:  78.66  -time: 715.15
Iteration: 42/50[==============] -Error: 1.7022534809  -Training_Accuracy:  77.34  -time: 732.04
Iteration: 43/50[==============] -Error: 1.7137387966  -Training_Accuracy:  85.30  -time: 749.23
Iteration: 44/50[==============] -Error: 1.7088558585  -Training_Accuracy:  80.99  -time: 766.19
```


Learning Curve_Error
Error
Epoch
2.4
2.2
2.0
1.8
0
10
20
30
40
50


Learning Curve_Training_Accuracy
Training_Accuracy
Epoch
45000
42500
40000
37500
35000
32500
30000
27500
0
10
20
30
40
50





In [7]:
```
print(max(my_net_tanh.Test_accuracies))
my_net_tanh.save('Models/NET_tanh_784_30_10_0.1')
```

```
9015.0
```

Comment 1

One important thing that we noticed is that a good learning rate for the sigmoid is not necessarly a good one also for the tanh. For instance, the learning rate 1.0 has given a good performance in the sigmoid case, but poor one using the tanh(2200 maximum test_accuracy for tanh vs 9557.0 maximum test_accuracy for sigmoid).

Comment 2

We tried the tanh with another learning rate and notice that the performance is better : 9015.0 for maximum test accuracy with 0.1 as a learning rate vs 2200 for maximum test accuracy with 1.0 as a learning rate. The ending error also has been decreased From 9.7478399753 with a learning rate of 1.0 to 1.6917229289 with a learning rate of 0.1 .

Comment 3

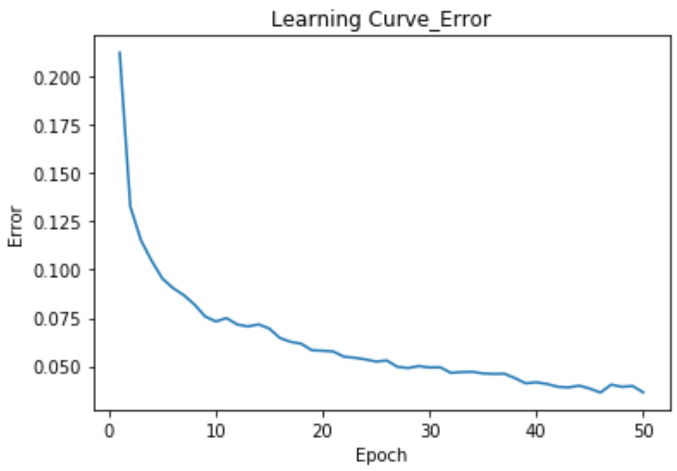
We can add that the choise of the transfer function must be based on the type of data. In fact, the tanh is centred around 0 (in [-1,1]) which is not the case for sigmoid (in [0,1]). Thus, the use of sigmoid is more likely to cause saturation in the later layers. In addition, the gradients of the tanh are stronger than the sigmoid ones (because of their respective derivatives supports).

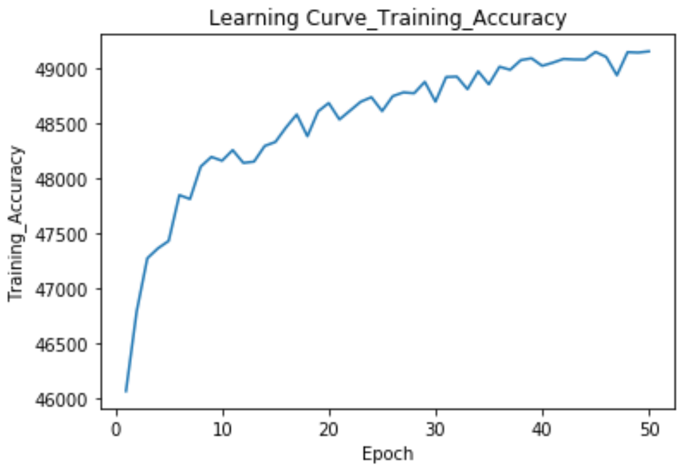
Question 2.2.4 : Add more neurons in the hidden layer (try with 100, 200, 300). Plot the curve representing the validation accuracy versus the number of neurons in the hidden layer. (Choose and justify other hyper-parameters)

In [20]:

```
#Your implementation goes here

my_net_more1= NeuralNetwork(784,100,10,iterations = 50, learning_rate = 1.0)

my_net_more1.train(training_data,validation_data)

my_net_more1.save('Models/NET_784_100_10_1.0')
```

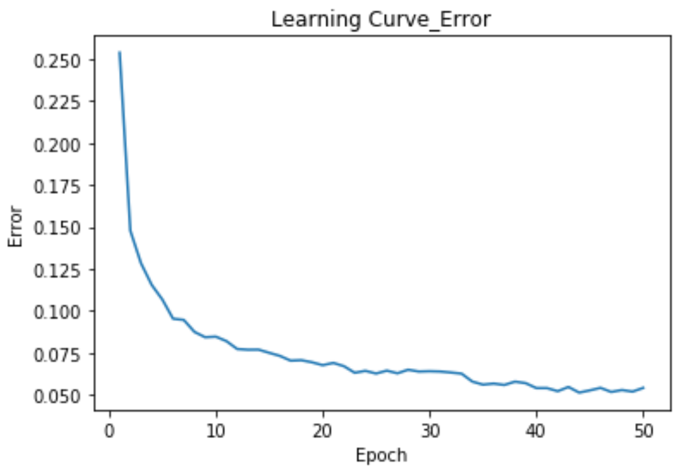
```
Iteration:  1/50[==============] -Error: 0.2122374744  -Training_Accuracy:  92.12  -time: 100.93
Iteration:  2/50[==============] -Error: 0.1325932746  -Training_Accuracy:  93.58  -time: 201.06
Iteration:  3/50[==============] -Error: 0.1150762410  -Training_Accuracy:  94.55  -time: 301.27
Iteration:  4/50[==============] -Error: 0.1045063353  -Training_Accuracy:  94.73  -time: 402.33
Iteration:  5/50[==============] -Error: 0.0953571069  -Training_Accuracy:  94.86  -time: 502.64
Iteration:  6/50[==============] -Error: 0.0904396897  -Training_Accuracy:  95.70  -time: 603.53
Iteration:  7/50[==============] -Error: 0.0867947471  -Training_Accuracy:  95.63  -time: 703.60
Iteration:  8/50[==============] -Error: 0.0819722212  -Training_Accuracy:  96.22  -time: 804.31
Iteration:  9/50[==============] -Error: 0.0757577762  -Training_Accuracy:  96.39  -time: 904.36
Iteration: 10/50[==============] -Error: 0.0731367633  -Training_Accuracy:  96.32  -time: 1004.39
Iteration: 11/50[==============] -Error: 0.0749242314  -Training_Accuracy:  96.52  -time: 1104.32
Iteration: 12/50[==============] -Error: 0.0716476032  -Training_Accuracy:  96.28  -time: 1204.33
Iteration: 13/50[==============] -Error: 0.0705831452  -Training_Accuracy:  96.31  -time: 1304.27
Iteration: 14/50[==============] -Error: 0.0716616442  -Training_Accuracy:  96.59  -time: 1404.38
Iteration: 15/50[==============] -Error: 0.0695417183  -Training_Accuracy:  96.67  -time: 1504.35
Iteration: 16/50[==============] -Error: 0.0645976694  -Training_Accuracy:  96.93  -time: 1604.34
Iteration: 17/50[==============] -Error: 0.0626272060  -Training_Accuracy:  97.17  -time: 1704.31
Iteration: 18/50[==============] -Error: 0.0616239015  -Training_Accuracy:  96.77  -time: 1804.30
Iteration: 19/50[==============] -Error: 0.0582663845  -Training_Accuracy:  97.22  -time: 1904.38
Iteration: 20/50[==============] -Error: 0.0580446493  -Training_Accuracy:  97.38  -time: 2004.53
Iteration: 21/50[==============] -Error: 0.0576426638  -Training_Accuracy:  97.08  -time: 2104.39
Iteration: 22/50[==============] -Error: 0.0549102306  -Training_Accuracy:  97.24  -time: 2204.22
Iteration: 23/50[==============] -Error: 0.0544124963  -Training_Accuracy:  97.40  -time: 2304.07
Iteration: 24/50[==============] -Error: 0.0535316423  -Training_Accuracy:  97.48  -time: 2404.00
Iteration: 25/50[==============] -Error: 0.0524266760  -Training_Accuracy:  97.23  -time: 2504.85
Iteration: 26/50[==============] -Error: 0.0529879391  -Training_Accuracy:  97.50  -time: 2604.87
Iteration: 27/50[==============] -Error: 0.0496728654  -Training_Accuracy:  97.57  -time: 2705.50
Iteration: 28/50[==============] -Error: 0.0490588575  -Training_Accuracy:  97.55  -time: 2805.60
Iteration: 29/50[==============] -Error: 0.0501225993  -Training_Accuracy:  97.76  -time: 2906.23
Iteration: 30/50[==============] -Error: 0.0493416326  -Training_Accuracy:  97.40  -time: 3006.23
Iteration: 31/50[==============] -Error: 0.0494553250  -Training_Accuracy:  97.85  -time: 3106.27
Iteration: 32/50[==============] -Error: 0.0465718428  -Training_Accuracy:  97.86  -time: 3206.09
Iteration: 33/50[==============] -Error: 0.0469994772  -Training_Accuracy:  97.63  -time: 3306.90
Iteration: 34/50[==============] -Error: 0.0471296886  -Training_Accuracy:  97.95  -time: 3406.74
Iteration: 35/50[==============] -Error: 0.0462957637  -Training_Accuracy:  97.72  -time: 3507.41
Iteration: 36/50[==============] -Error: 0.0460515962  -Training_Accuracy:  98.04  -time: 3607.20
Iteration: 37/50[==============] -Error: 0.0461639610  -Training_Accuracy:  97.98  -time: 3707.94
Iteration: 38/50[==============] -Error: 0.0438584742  -Training_Accuracy:  98.16  -time: 3807.98
Iteration: 39/50[==============] -Error: 0.0411670783  -Training_Accuracy:  98.19  -time: 3908.57
Iteration: 40/50[==============] -Error: 0.0416952047  -Training_Accuracy:  98.05  -time: 4008.55
Iteration: 41/50[==============] -Error: 0.0408245871  -Training_Accuracy:  98.11  -time: 4109.26
Iteration: 42/50[==============] -Error: 0.0393796728  -Training_Accuracy:  98.18  -time: 4209.28
Iteration: 43/50[==============] -Error: 0.0390601539  -Training_Accuracy:  98.17  -time: 4309.23
Iteration: 44/50[==============] -Error: 0.0399562669  -Training_Accuracy:  98.17  -time: 4410.04
```

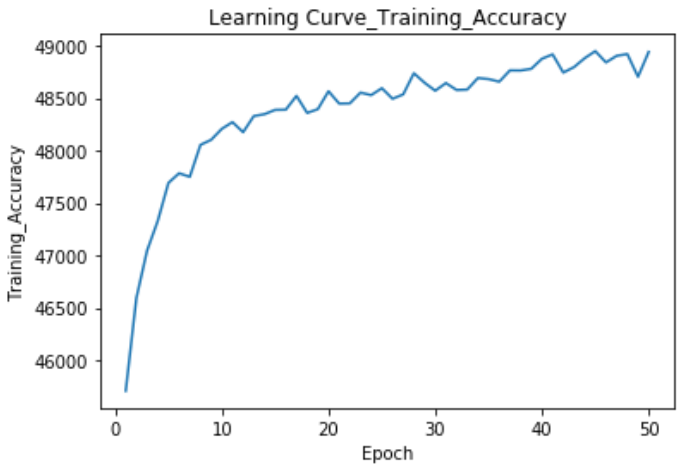

Learning Curve_Error
Error
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
10
20
30
40
50
Epoch


Learning Curve_Training_Accuracy
Training_Accuracy
49000
48500
48000
47500
47000
46500
46000
0
10
20
30
40
50
Epoch

In [21]:
```
my_net_more2= NeuralNetwork(784,200,10,iterations = 50, learning_rate = 1.0)

my_net_more2.train(training_data,validation_data)

my_net_more2.save('Models/NET_784_200_10_1.0')
```

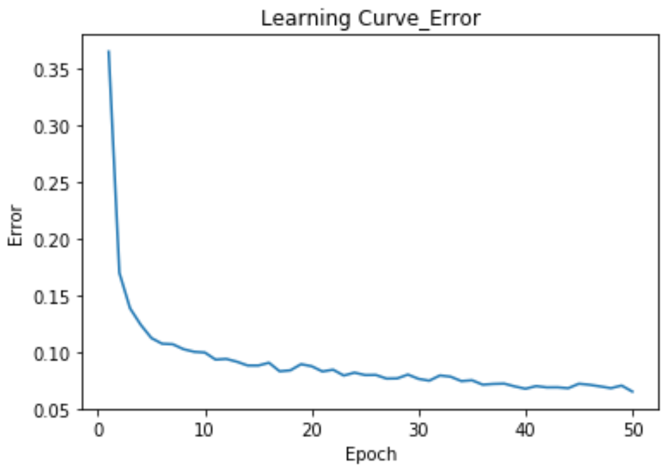
```
Iteration:  1/50[==============] -Error: 0.2537643622  -Training_Accuracy:  91.41  -time: 206.53
Iteration:  2/50[==============] -Error: 0.1477816113  -Training_Accuracy:  93.19  -time: 412.59
Iteration:  3/50[==============] -Error: 0.1283718321  -Training_Accuracy:  94.09  -time: 619.11
Iteration:  4/50[==============] -Error: 0.1155108596  -Training_Accuracy:  94.66  -time: 809.92
Iteration:  5/50[==============] -Error: 0.1067554807  -Training_Accuracy:  95.38  -time: 996.58
Iteration:  6/50[==============] -Error: 0.0953698929  -Training_Accuracy:  95.57  -time: 1183.15
Iteration:  7/50[==============] -Error: 0.0946699454  -Training_Accuracy:  95.50  -time: 1369.86
Iteration:  8/50[==============] -Error: 0.0875342900  -Training_Accuracy:  96.11  -time: 1556.25
Iteration:  9/50[==============] -Error: 0.0843540199  -Training_Accuracy:  96.20  -time: 1742.93
Iteration: 10/50[==============] -Error: 0.0847082260  -Training_Accuracy:  96.41  -time: 1929.56
Iteration: 11/50[==============] -Error: 0.0820048744  -Training_Accuracy:  96.54  -time: 2116.49
Iteration: 12/50[==============] -Error: 0.0773419947  -Training_Accuracy:  96.35  -time: 2303.00
Iteration: 13/50[==============] -Error: 0.0769179808  -Training_Accuracy:  96.66  -time: 2489.59
Iteration: 14/50[==============] -Error: 0.0769683287  -Training_Accuracy:  96.69  -time: 2676.03
Iteration: 15/50[==============] -Error: 0.0750669318  -Training_Accuracy:  96.78  -time: 2862.52
Iteration: 16/50[==============] -Error: 0.0732408211  -Training_Accuracy:  96.78  -time: 3048.71
Iteration: 17/50[==============] -Error: 0.0704082579  -Training_Accuracy:  97.04  -time: 3235.07
Iteration: 18/50[==============] -Error: 0.0706746096  -Training_Accuracy:  96.72  -time: 3421.35
Iteration: 19/50[==============] -Error: 0.0694506078  -Training_Accuracy:  96.79  -time: 3608.05
Iteration: 20/50[==============] -Error: 0.0677150289  -Training_Accuracy:  97.13  -time: 3794.67
Iteration: 21/50[==============] -Error: 0.0690623023  -Training_Accuracy:  96.90  -time: 3981.58
Iteration: 22/50[==============] -Error: 0.0670862652  -Training_Accuracy:  96.90  -time: 4168.53
Iteration: 23/50[==============] -Error: 0.0631871205  -Training_Accuracy:  97.11  -time: 4355.08
Iteration: 24/50[==============] -Error: 0.0643395685  -Training_Accuracy:  97.06  -time: 4542.03
Iteration: 25/50[==============] -Error: 0.0627461493  -Training_Accuracy:  97.19  -time: 4728.51
Iteration: 26/50[==============] -Error: 0.0644572795  -Training_Accuracy:  96.99  -time: 4915.01
Iteration: 27/50[==============] -Error: 0.0628797246  -Training_Accuracy:  97.07  -time: 5101.54
Iteration: 28/50[==============] -Error: 0.0650165320  -Training_Accuracy:  97.48  -time: 5288.22
Iteration: 29/50[==============] -Error: 0.0639210935  -Training_Accuracy:  97.29  -time: 5474.66
Iteration: 30/50[==============] -Error: 0.0641075970  -Training_Accuracy:  97.14  -time: 5661.29
Iteration: 31/50[==============] -Error: 0.0639311854  -Training_Accuracy:  97.29  -time: 5848.03
Iteration: 32/50[==============] -Error: 0.0633571495  -Training_Accuracy:  97.16  -time: 6034.23
Iteration: 33/50[==============] -Error: 0.0626791583  -Training_Accuracy:  97.16  -time: 6220.59
Iteration: 34/50[==============] -Error: 0.0579665272  -Training_Accuracy:  97.38  -time: 6407.33
Iteration: 35/50[==============] -Error: 0.0561278034  -Training_Accuracy:  97.36  -time: 6593.98
Iteration: 36/50[==============] -Error: 0.0567615913  -Training_Accuracy:  97.31  -time: 6780.43
Iteration: 37/50[==============] -Error: 0.0559402909  -Training_Accuracy:  97.53  -time: 6967.00
Iteration: 38/50[==============] -Error: 0.0579243741  -Training_Accuracy:  97.53  -time: 7153.26
Iteration: 39/50[==============] -Error: 0.0570096747  -Training_Accuracy:  97.56  -time: 7339.43
Iteration: 40/50[==============] -Error: 0.0540615408  -Training_Accuracy:  97.75  -time: 7525.80
Iteration: 41/50[==============] -Error: 0.0540525496  -Training_Accuracy:  97.84  -time: 7712.24
Iteration: 42/50[==============] -Error: 0.0522232909  -Training_Accuracy:  97.49  -time: 7898.57
Iteration: 43/50[==============] -Error: 0.0547411162  -Training_Accuracy:  97.59  -time: 8084.96
Iteration: 44/50[==============] -Error: 0.0513964566  -Training_Accuracy:  97.76  -time: 8271.12
```

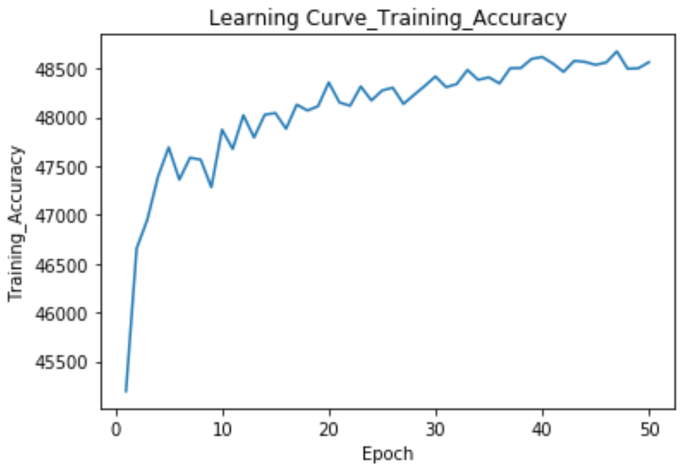

Learning Curve_Error
Error
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
10
20
30
40
50
Epoch


Learning Curve_Training_Accuracy
Training_Accuracy
49000
48500
48000
47500
47000
46500
46000
0
10
20
30
40
50
Epoch

In [22]:
```
my_net_more3= NeuralNetwork(784,300,10,iterations = 50, learning_rate = 1.0)

my_net_more3.train(training_data,validation_data)

my_net_more3.save('Models/NET_784_300_10_1.0')
```

```
Iteration:  1/50[==============] -Error: 0.3646148428  -Training_Accuracy:  90.39  -time: 282.64
Iteration:  2/50[==============] -Error: 0.1698727775  -Training_Accuracy:  93.31  -time: 565.29
Iteration:  3/50[==============] -Error: 0.1386434756  -Training_Accuracy:  93.90  -time: 848.00
Iteration:  4/50[==============] -Error: 0.1241428607  -Training_Accuracy:  94.78  -time: 1131.29
Iteration:  5/50[==============] -Error: 0.1124727677  -Training_Accuracy:  95.38  -time: 1414.43
Iteration:  6/50[==============] -Error: 0.1074828613  -Training_Accuracy:  94.72  -time: 1697.16
Iteration:  7/50[==============] -Error: 0.1070230362  -Training_Accuracy:  95.17  -time: 1976.39
Iteration:  8/50[==============] -Error: 0.1026681851  -Training_Accuracy:  95.13  -time: 2255.97
Iteration:  9/50[==============] -Error: 0.1003808506  -Training_Accuracy:  94.57  -time: 2535.64
Iteration: 10/50[==============] -Error: 0.0997363414  -Training_Accuracy:  95.75  -time: 2815.21
Iteration: 11/50[==============] -Error: 0.0935083462  -Training_Accuracy:  95.35  -time: 3094.56
Iteration: 12/50[==============] -Error: 0.0941077864  -Training_Accuracy:  96.04  -time: 3373.71
Iteration: 13/50[==============] -Error: 0.0916342401  -Training_Accuracy:  95.59  -time: 3653.61
Iteration: 14/50[==============] -Error: 0.0883226886  -Training_Accuracy:  96.05  -time: 3932.94
Iteration: 15/50[==============] -Error: 0.0882704394  -Training_Accuracy:  96.09  -time: 4212.69
Iteration: 16/50[==============] -Error: 0.0907428227  -Training_Accuracy:  95.76  -time: 4492.40
Iteration: 17/50[==============] -Error: 0.0832009311  -Training_Accuracy:  96.26  -time: 4771.68
Iteration: 18/50[==============] -Error: 0.0840268520  -Training_Accuracy:  96.14  -time: 5051.16
Iteration: 19/50[==============] -Error: 0.0894501281  -Training_Accuracy:  96.23  -time: 5330.68
Iteration: 20/50[==============] -Error: 0.0876538926  -Training_Accuracy:  96.71  -time: 5610.73
Iteration: 21/50[==============] -Error: 0.0830618219  -Training_Accuracy:  96.30  -time: 5889.98
Iteration: 22/50[==============] -Error: 0.0845969260  -Training_Accuracy:  96.24  -time: 6169.39
Iteration: 23/50[==============] -Error: 0.0794684497  -Training_Accuracy:  96.63  -time: 6448.88
Iteration: 24/50[==============] -Error: 0.0820281815  -Training_Accuracy:  96.34  -time: 6727.82
Iteration: 25/50[==============] -Error: 0.0798766652  -Training_Accuracy:  96.55  -time: 7007.01
Iteration: 26/50[==============] -Error: 0.0799794329  -Training_Accuracy:  96.61  -time: 7286.03
Iteration: 27/50[==============] -Error: 0.0767926858  -Training_Accuracy:  96.27  -time: 7565.54
Iteration: 28/50[==============] -Error: 0.0769461995  -Training_Accuracy:  96.46  -time: 7844.35
Iteration: 29/50[==============] -Error: 0.0803268844  -Training_Accuracy:  96.64  -time: 8123.73
Iteration: 30/50[==============] -Error: 0.0765385694  -Training_Accuracy:  96.84  -time: 8402.81
Iteration: 31/50[==============] -Error: 0.0749326351  -Training_Accuracy:  96.62  -time: 8681.89
Iteration: 32/50[==============] -Error: 0.0795653241  -Training_Accuracy:  96.68  -time: 8960.99
Iteration: 33/50[==============] -Error: 0.0784619722  -Training_Accuracy:  96.97  -time: 9240.48
Iteration: 34/50[==============] -Error: 0.0745858375  -Training_Accuracy:  96.77  -time: 9519.46
Iteration: 35/50[==============] -Error: 0.0753060968  -Training_Accuracy:  96.82  -time: 9798.93
Iteration: 36/50[==============] -Error: 0.0714276487  -Training_Accuracy:  96.69  -time: 10078.25
Iteration: 37/50[==============] -Error: 0.0720923616  -Training_Accuracy:  97.00  -time: 10357.61
Iteration: 38/50[==============] -Error: 0.0723648921  -Training_Accuracy:  97.01  -time: 10636.40
Iteration: 39/50[==============] -Error: 0.0698950830  -Training_Accuracy:  97.19  -time: 10915.37
Iteration: 40/50[==============] -Error: 0.0678577288  -Training_Accuracy:  97.24  -time: 11194.02
Iteration: 41/50[==============] -Error: 0.0701682648  -Training_Accuracy:  97.10  -time: 11472.65
Iteration: 42/50[==============] -Error: 0.0690317915  -Training_Accuracy:  96.93  -time: 11751.27
Iteration: 43/50[==============] -Error: 0.0690926277  -Training_Accuracy:  97.16  -time: 12030.83
Iteration: 44/50[==============] -Error: 0.0683089771  -Training_Accuracy:  97.13  -time: 12310.09
```

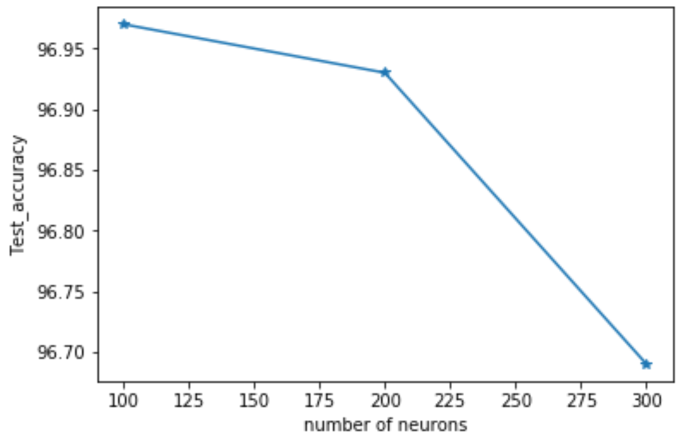

Learning Curve_Error
Error
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
10
20
30
40
50
Epoch


Learning Curve_Training_Accuracy
Training_Accuracy
48500
48000
47500
47000
46500
46000
45500
0
10
20
30
40
50
Epoch

In [5]:
```
from NeuralNetwork import *
my_net_more1= NeuralNetwork(784,100,10,iterations = 50, learning_rate = 1.0)
my_net_more2= NeuralNetwork(784,200,10,iterations = 50, learning_rate = 1.0)
my_net_more3= NeuralNetwork(784,300,10,iterations = 50, learning_rate = 1.0)
my_net_more1.load('Models/NET_784_100_10_1.0')
my_net_more2.load('Models/NET_784_200_10_1.0')
my_net_more3.load('Models/NET_784_300_10_1.0')
test_accuracy_100=my_net_more1.predict(test_data)/100
print('Test_Accuracy 100 neurons %-2.2f' % test_accuracy)
test_accuracy_200=my_net_more2.predict(test_data)/100
print('Test_Accuracy 200 neurons %-2.2f' % test_accuracy)
test_accuracy_200=my_net_more3.predict(test_data)/100
print('Test_Accuracy 300 neurons %-2.2f' % test_accuracy)
```

```
Test_Accuracy 100 neurons 96.69
Test_Accuracy 200 neurons 96.69
Test_Accuracy 300 neurons 96.69
```

In [7]:
```
import matplotlib.pyplot as plt
plt.plot([100,200,300],[test_accuracy_100,test_accuracy_200,test_accuracy_300],marker="*")
plt.xlabel('number of neurons')
plt.ylabel('Test_accuracy')
plt.show()
```



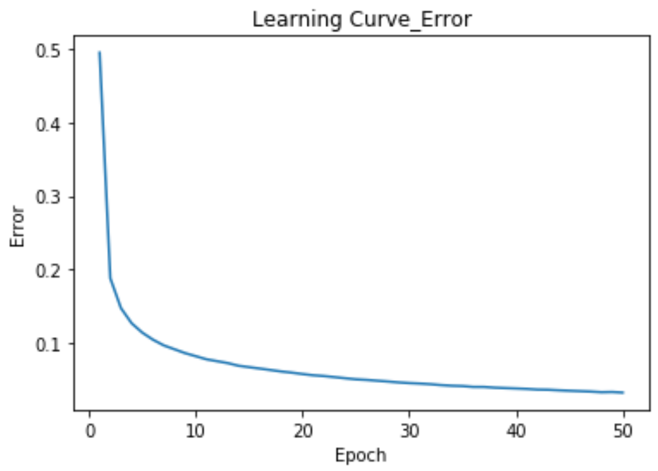
Comment

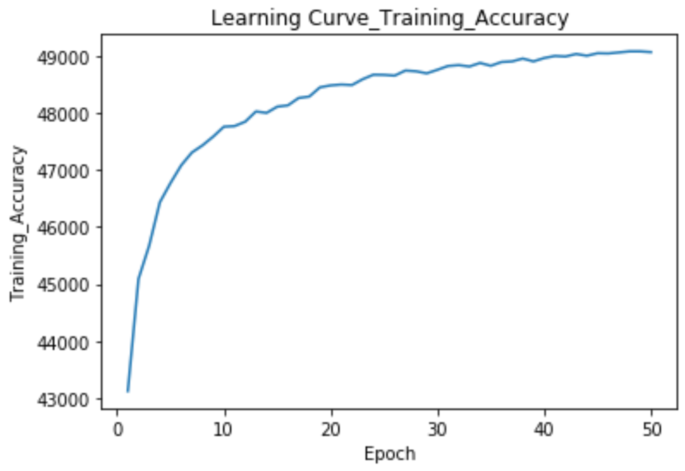
We can notice that the more we put neurons in the hidden layer, the more the training takes time, which is logical because more neurons implies more computations. When setting the number of neurons to 100, we noticed an improvment in the training efficiency. In fact the maximum Training accuracy has been increased to 98.32% (instead of 96.63 % for 30 neurons for the same learning rate 1.0) and the ending error has been decreasd to 0.0364878094(instead of 0.0659517623 for 30 neurons for the same learning rate 1.0). In the other hand, We couldn't notice any real improvment when adding more neurons (figure above). We think that an optimal configuration of the neural network is not necessarly the one with as many neurons as possible in the hidden layer, but it's a configuration where the number of neurons is wisely choosen based on the task the network will be in charge of, the complexity and the number of computations, and so on. There are many rules to choose the number of neurons in the hidden layer : For instance, the optimal size of a hidden layer is usually between the size of the input and the size of the output layers.

Question 2.2.5 : Add one additionnal hidden layers and train your network, discuss your results with different setting.

In [ ]:
```
#Your implementation goes here : 2 hidden layers of 30 neurons

from NeuralNetwork import *

my_net_2_hidden= NeuralNetwork(784,30,30,10,iterations = 50, learning_rate = 0.1)

my_net_2_hidden.train(training_data,validation_data)

my_net_2_hidden.save('Models/NET_2_hidden_784_300_10_1.0')
```

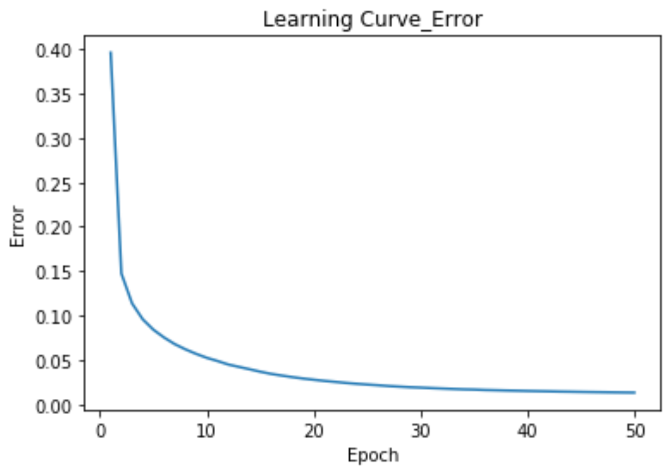
```
Iteration:  1/50[==============] -Error: 0.4958451581  -Training_Accuracy:  86.25  -time: 22.51
Iteration:  2/50[==============] -Error: 0.1876477042  -Training_Accuracy:  90.18  -time: 46.63
Iteration:  3/50[==============] -Error: 0.1468198012  -Training_Accuracy:  91.34  -time: 68.42
Iteration:  4/50[==============] -Error: 0.1261324293  -Training_Accuracy:  92.86  -time: 89.74
Iteration:  5/50[==============] -Error: 0.1133154399  -Training_Accuracy:  93.54  -time: 111.85
Iteration:  6/50[==============] -Error: 0.1036588296  -Training_Accuracy:  94.16  -time: 133.89
Iteration:  7/50[==============] -Error: 0.0959699322  -Training_Accuracy:  94.61  -time: 155.63
Iteration:  8/50[==============] -Error: 0.0906537206  -Training_Accuracy:  94.86  -time: 177.28
Iteration:  9/50[==============] -Error: 0.0853656335  -Training_Accuracy:  95.17  -time: 198.85
Iteration: 10/50[==============] -Error: 0.0811849937  -Training_Accuracy:  95.51  -time: 220.08
Iteration: 11/50[==============] -Error: 0.0769290217  -Training_Accuracy:  95.53  -time: 242.32
Iteration: 12/50[==============] -Error: 0.0743252287  -Training_Accuracy:  95.69  -time: 263.59
Iteration: 13/50[==============] -Error: 0.0717307519  -Training_Accuracy:  96.05  -time: 284.63
Iteration: 14/50[==============] -Error: 0.0681158787  -Training_Accuracy:  96.00  -time: 305.81
Iteration: 15/50[==============] -Error: 0.0661567471  -Training_Accuracy:  96.21  -time: 327.09
Iteration: 16/50[==============] -Error: 0.0642372681  -Training_Accuracy:  96.26  -time: 348.46
Iteration: 17/50[==============] -Error: 0.0622921211  -Training_Accuracy:  96.52  -time: 370.45
Iteration: 18/50[==============] -Error: 0.0602674688  -Training_Accuracy:  96.57  -time: 391.64
Iteration: 19/50[==============] -Error: 0.0586723121  -Training_Accuracy:  96.89  -time: 413.65
Iteration: 20/50[==============] -Error: 0.0567714946  -Training_Accuracy:  96.96  -time: 435.28
Iteration: 21/50[==============] -Error: 0.0550991748  -Training_Accuracy:  96.99  -time: 456.42
Iteration: 22/50[==============] -Error: 0.0540188908  -Training_Accuracy:  96.97  -time: 477.85
Iteration: 23/50[==============] -Error: 0.0525590792  -Training_Accuracy:  97.18  -time: 498.83
Iteration: 24/50[==============] -Error: 0.0510027916  -Training_Accuracy:  97.33  -time: 519.54
Iteration: 25/50[==============] -Error: 0.0495596629  -Training_Accuracy:  97.33  -time: 540.70
Iteration: 26/50[==============] -Error: 0.0487631406  -Training_Accuracy:  97.30  -time: 561.87
Iteration: 27/50[==============] -Error: 0.0476292072  -Training_Accuracy:  97.48  -time: 583.03
Iteration: 28/50[==============] -Error: 0.0465481396  -Training_Accuracy:  97.45  -time: 604.18
Iteration: 29/50[==============] -Error: 0.0453375842  -Training_Accuracy:  97.38  -time: 625.36
Iteration: 30/50[==============] -Error: 0.0443958361  -Training_Accuracy:  97.51  -time: 646.26
Iteration: 31/50[==============] -Error: 0.0437461256  -Training_Accuracy:  97.64  -time: 667.03
Iteration: 32/50[==============] -Error: 0.0428567081  -Training_Accuracy:  97.67  -time: 688.21
Iteration: 33/50[==============] -Error: 0.0416263729  -Training_Accuracy:  97.62  -time: 709.46
Iteration: 34/50[==============] -Error: 0.0407699249  -Training_Accuracy:  97.75  -time: 730.80
Iteration: 35/50[==============] -Error: 0.0403217049  -Training_Accuracy:  97.65  -time: 752.60
Iteration: 36/50[==============] -Error: 0.0391938265  -Training_Accuracy:  97.78  -time: 774.66
Iteration: 37/50[==============] -Error: 0.0391326436  -Training_Accuracy:  97.80  -time: 802.99
Iteration: 38/50[==============] -Error: 0.0381294739  -Training_Accuracy:  97.90  -time: 833.26
Iteration: 39/50[==============] -Error: 0.0375513219  -Training_Accuracy:  97.80  -time: 858.04
Iteration: 40/50[==============] -Error: 0.0370367825  -Training_Accuracy:  97.92  -time: 886.57
Iteration: 41/50[==============] -Error: 0.0364702262  -Training_Accuracy:  97.99  -time: 910.92
Iteration: 42/50[==============] -Error: 0.0355536194  -Training_Accuracy:  97.97  -time: 932.34
Iteration: 43/50[==============] -Error: 0.0353046257  -Training_Accuracy:  98.06  -time: 954.73
Iteration: 44/50[==============] -Error: 0.0345286320  -Training_Accuracy:  98.00  -time: 975.82
```

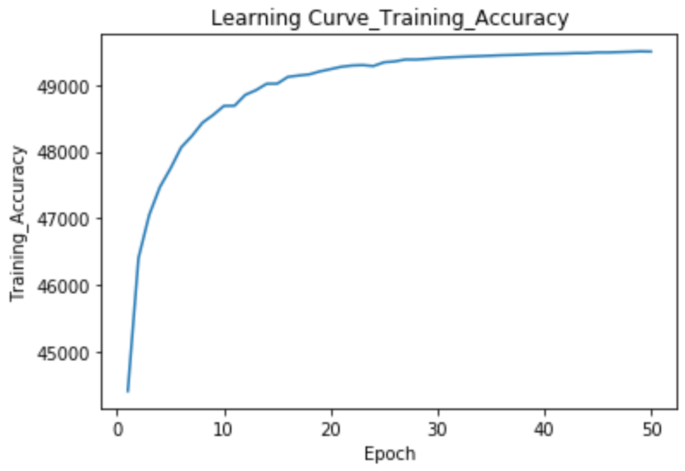

Learning Curve_Error
Error
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epoch


Learning Curve_Training_Accuracy
Training_Accuracy
49000
48000
47000
46000
45000
44000
43000
0
10
20
30
40
50
Epoch

In [4]:
```
#Your implementation goes here : 2 hidden layers : the first of 100 neurons and the second of 30 neurons
from NeuralNetwork import *

my_net_2_hidden= NeuralNetwork(784,100,30,10,iterations = 50, learning_rate = 0.1)

my_net_2_hidden.train(training_data,validation_data)

my_net_2_hidden.save('Models/NET_2_hidden_784_100_30_10_0.1')
```

```
Iteration:  1/50[==============] -Error: 0.3962768588  -Training_Accuracy:  88.83  -time: 30.53
Iteration:  2/50[==============] -Error: 0.1473634930  -Training_Accuracy:  92.82  -time: 61.12
Iteration:  3/50[==============] -Error: 0.1136800584  -Training_Accuracy:  94.10  -time: 92.97
Iteration:  4/50[==============] -Error: 0.0956984715  -Training_Accuracy:  94.94  -time: 127.64
Iteration:  5/50[==============] -Error: 0.0839190422  -Training_Accuracy:  95.50  -time: 159.48
Iteration:  6/50[==============] -Error: 0.0751379274  -Training_Accuracy:  96.13  -time: 190.33
Iteration:  7/50[==============] -Error: 0.0676879157  -Training_Accuracy:  96.47  -time: 221.44
Iteration:  8/50[==============] -Error: 0.0618081878  -Training_Accuracy:  96.87  -time: 252.92
Iteration:  9/50[==============] -Error: 0.0567371408  -Training_Accuracy:  97.10  -time: 283.18
Iteration: 10/50[==============] -Error: 0.0522521614  -Training_Accuracy:  97.37  -time: 314.97
Iteration: 11/50[==============] -Error: 0.0486940843  -Training_Accuracy:  97.38  -time: 345.84
Iteration: 12/50[==============] -Error: 0.0447642829  -Training_Accuracy:  97.70  -time: 377.25
Iteration: 13/50[==============] -Error: 0.0420598255  -Training_Accuracy:  97.84  -time: 407.67
Iteration: 14/50[==============] -Error: 0.0394052024  -Training_Accuracy:  98.03  -time: 438.94
Iteration: 15/50[==============] -Error: 0.0366391802  -Training_Accuracy:  98.04  -time: 470.40
Iteration: 16/50[==============] -Error: 0.0342059658  -Training_Accuracy:  98.24  -time: 502.38
Iteration: 17/50[==============] -Error: 0.0322998992  -Training_Accuracy:  98.28  -time: 533.50
Iteration: 18/50[==============] -Error: 0.0305246730  -Training_Accuracy:  98.32  -time: 564.24
Iteration: 19/50[==============] -Error: 0.0289376377  -Training_Accuracy:  98.41  -time: 595.45
Iteration: 20/50[==============] -Error: 0.0275818405  -Training_Accuracy:  98.47  -time: 625.89
Iteration: 21/50[==============] -Error: 0.0262660282  -Training_Accuracy:  98.54  -time: 656.80
Iteration: 22/50[==============] -Error: 0.0251876709  -Training_Accuracy:  98.58  -time: 688.00
Iteration: 23/50[==============] -Error: 0.0240065971  -Training_Accuracy:  98.59  -time: 719.17
Iteration: 24/50[==============] -Error: 0.0230345546  -Training_Accuracy:  98.57  -time: 750.12
Iteration: 25/50[==============] -Error: 0.0221862777  -Training_Accuracy:  98.68  -time: 781.12
Iteration: 26/50[==============] -Error: 0.0213894606  -Training_Accuracy:  98.70  -time: 812.41
Iteration: 27/50[==============] -Error: 0.0204894308  -Training_Accuracy:  98.76  -time: 843.35
Iteration: 28/50[==============] -Error: 0.0198202191  -Training_Accuracy:  98.76  -time: 874.27
Iteration: 29/50[==============] -Error: 0.0191665531  -Training_Accuracy:  98.78  -time: 905.24
Iteration: 30/50[==============] -Error: 0.0187064500  -Training_Accuracy:  98.80  -time: 936.30
Iteration: 31/50[==============] -Error: 0.0181595164  -Training_Accuracy:  98.82  -time: 968.03
Iteration: 32/50[==============] -Error: 0.0176839402  -Training_Accuracy:  98.84  -time: 1001.77
Iteration: 33/50[==============] -Error: 0.0172036061  -Training_Accuracy:  98.85  -time: 1032.16
Iteration: 34/50[==============] -Error: 0.0167615208  -Training_Accuracy:  98.86  -time: 1063.32
Iteration: 35/50[==============] -Error: 0.0165048475  -Training_Accuracy:  98.88  -time: 1095.91
Iteration: 36/50[==============] -Error: 0.0160333220  -Training_Accuracy:  98.89  -time: 1127.16
Iteration: 37/50[==============] -Error: 0.0157605115  -Training_Accuracy:  98.90  -time: 1160.47
Iteration: 38/50[==============] -Error: 0.0154445640  -Training_Accuracy:  98.91  -time: 1190.88
Iteration: 39/50[==============] -Error: 0.0151908401  -Training_Accuracy:  98.92  -time: 1224.48
Iteration: 40/50[==============] -Error: 0.0149261921  -Training_Accuracy:  98.93  -time: 1254.28
Iteration: 41/50[==============] -Error: 0.0147040342  -Training_Accuracy:  98.94  -time: 1285.25
Iteration: 42/50[==============] -Error: 0.0144950738  -Training_Accuracy:  98.94  -time: 1315.06
Iteration: 43/50[==============] -Error: 0.0142474143  -Training_Accuracy:  98.95  -time: 1345.51
Iteration: 44/50[==============] -Error: 0.0140799990  -Training_Accuracy:  98.95  -time: 1375.36
```


Learning Curve_Error
Error
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Epoch
0
10
20
30
40
50


Learning Curve_Training_Accuracy
Training_Accuracy
49000
48000
47000
46000
45000
Epoch
0
10
20
30
40
50

Comment

We've runned two experiments : the first one with 2 hidden layers 30 neurons each, and the second with 2 hidden layers, 100 neurons for the first and 30 neurons for the second. Obviously, the second one spend a longer time in the training because more neurons implies more computations. On the other hand, the second experiment has shown that increasing the number of neurons of the first hidden layer gives a better perfomance on the training set (99% maximum training accuracy)