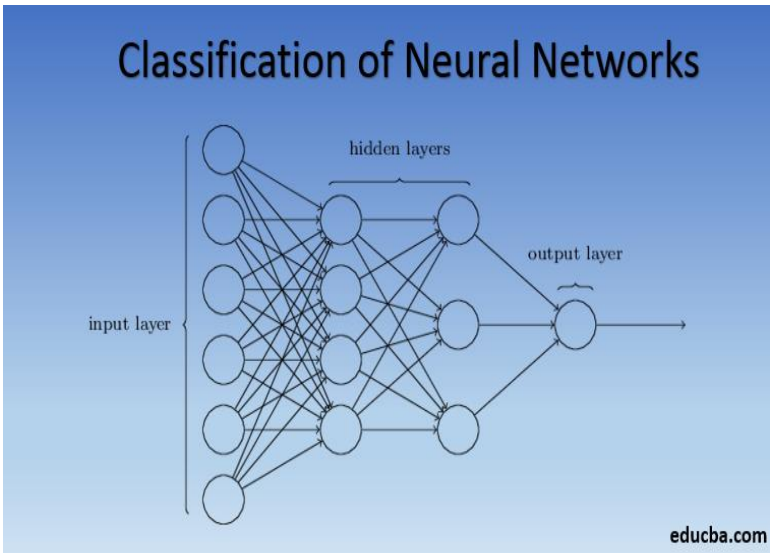
Problem definition & importance:

Artificial Neural Networks and Deep Neural Networks are effective for high dimensionality problems, but they are also theoretically complex. Fortunately, there are deep learning frameworks, like Tensor Flow, that can help you set deep neural networks faster, with only a few lines of code. Classification involves predicting which class an item belongs to. Some classifiers are binary, resulting in a yes/no decision. Others are multi-class, able to categorize an item into one of several categories. Classification is a very common use case of machine learning—classification algorithms are used to solve problems like email spam filtering, document categorization, speech recognition, image recognition, and handwriting recognition.

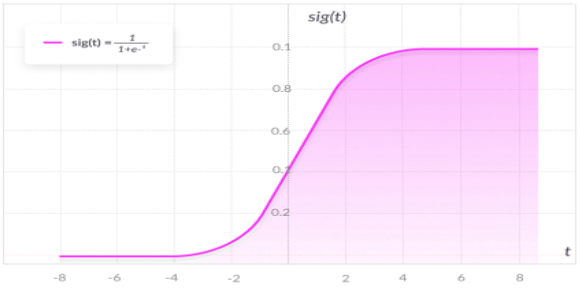


In this context, a neural network is one of several machine learning algorithms that can help solve classification problems. Its unique strength is its ability to dynamically create complex prediction functions, and emulate human thinking, in a way that no other algorithm can. There are many classification problems for which neural networks have yielded the best results. In real-world machine learning projects, you will find yourself iterating on the same classification problem, using different classifiers, and different parameters or structures of the same classifier. If you restrict yourself to "regular" classifiers besides neural networks, you can use great open source libraries like scikit-learn, which provide built-in implementations of all popular classifiers, and are relatively easy to get started with.

If you go down the neural network path, you will need to use the "heavier" deep learning frameworks such as Google's Tensor Flow, Keras and PyTorch. These frameworks support both ordinary classifiers like Naive Bayes or KNN, and are able to set up neural networks of amazing complexity with only a few lines of code. While these frameworks are very powerful, each of them has operating concepts you'll need to learn, and each has its learning curve.

Methods & Algorithms:

To understand classification with neural networks, it's essential to learn how other classification algorithms work, and their unique strengths. For many problems, a neural network may be unsuitable or "overkill". For others, it might be the only solution.



Classification with Neural Networks - Binary

Logistic Regression

Classifier type

Binary

How It Works

Analyzes a set of data points with one or more independent variables (input variables, which may affect the outcome) and finds the best fitting model to describe the data points, using the logistic regression

$$Y_i = \frac{1}{1-e^{-X_i'\beta}} + \varepsilon_i \text{ where } -\infty < x < \infty, y = 0,1$$

equation:

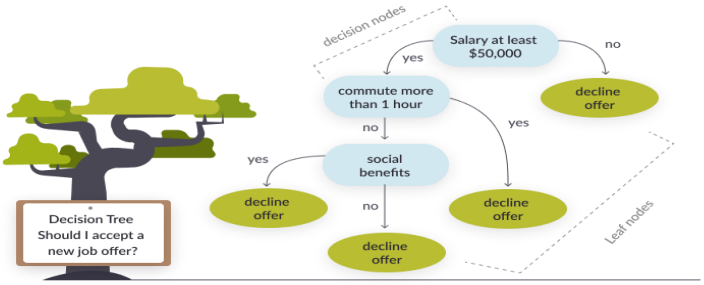
Strengths

Simple to implement and understand, very effective for problems in which the set of input variables is well known and closely correlated with the outcome.

Weaknesses

Less effective when some of the input variables are not known, or when there are complex relationships between the input variables.

Decision Tree Algorithm



Classifier type

Multiclass

How it works

Uses a tree structure with a set of "if-then" rules to classify data points. The rules are learned sequentially from the training data. The tree is constructed top-down; attributes at the top of the tree have a larger impact on the classification decision. The training process continues until it meets a termination condition.

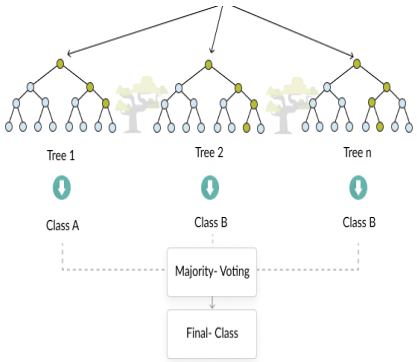
Strengths

Able to model complex decision processes, very intuitive interpretation of results.

Weaknesses

Can very easily overfit the data, by over-growing a tree with branches that reflect outliers in the data set. A way to deal with overfitting is pruning the model, either by preventing it from growing superfluous branches (pre-pruning), or removing them after the tree is grown (post-pruning).

Random Forest Algorithm



Classifier type

Multiclass

How it works

A more advanced version of the decision tree, which addresses overfitting by growing a large number of trees with random variations, then selecting and aggregating the best-performing decision trees. The "forest" is an ensemble of decision trees, typically done using a technique called "bagging".

Strengths

Provides the strengths of the decision tree algorithm, and is very effective at preventing overfitting and thus much more accurate, even compared to a decision tree with extensive manual pruning.

Weaknesses

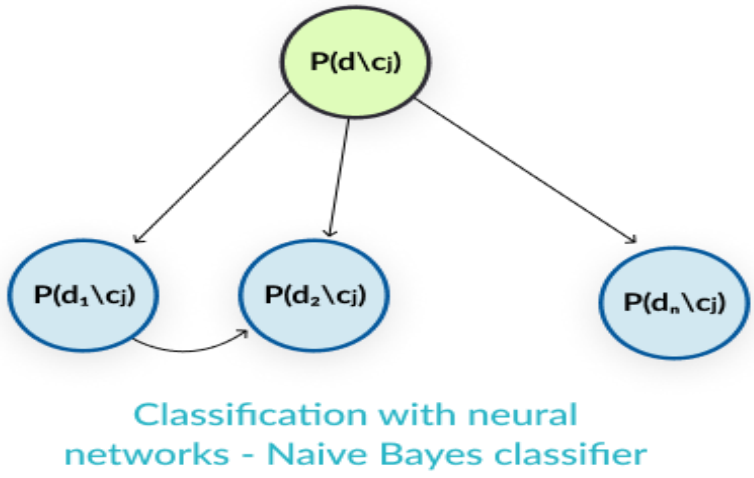
Not intuitive, difficult to understand why the model generates a specific outcome.

Naive Bayes Classifier

Classifier type

Multiclass

How it works



Classification with neural networks - Naive Bayes classifier

A probability-based classifier based on the Bayes algorithm. According to the concept of dependent probability, it calculates the probability that each of the features of a data point (the input variables) exists in each of the target classes. It then selects the category for which the probabilities are maximal. The model is based on an assumption (which is often not true) that the features are conditionally independent.

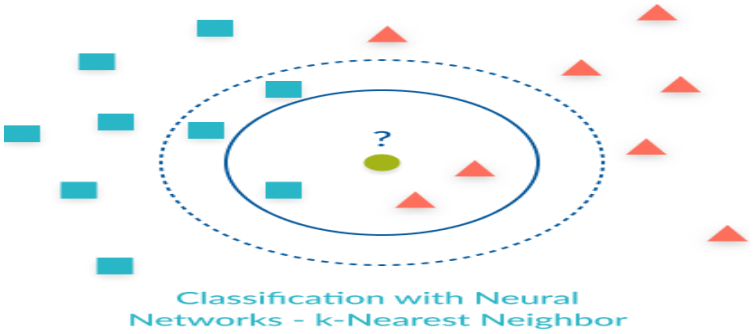
Strengths

Simple to implement and computationally light—the algorithm is linear and does not involve iterative calculations. Although its assumptions are not valid in most cases, Naive Bayes is surprisingly accurate for a large set of problems, scalable to very large data sets, and is used for many NLP models. Can also be used to construct multi-layer decision trees, with a Bayes classifier at every node.

Weaknesses

Very sensitive to the set of categories selected, which must be exhaustive. Problems where categories may be overlapping or there are unknown categories can dramatically reduce accuracy.

k-Nearest Neighbor (KNN)



Classifier type

Multiclass

How it works

Classifies each data point by analyzing its nearest neighbors from the training set. The current data point is assigned the class most commonly found among its neighbors. The algorithm is non-parametric (makes no assumptions on the underlying data) and uses lazy learning (does not pre-train, all training data is used during classification).

Strengths

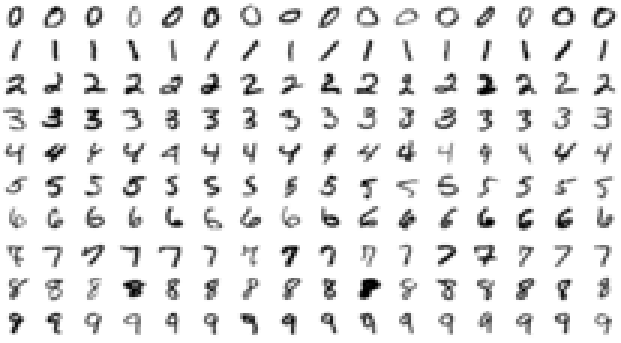
Very simple to implement and understand, and highly effective for many classification problems, especially with low dimensionality (small number of features or input variables).

Weaknesses

KNN's accuracy is not comparable to supervised learning methods. Not suitable for high dimensionality problems. Computationally intensive, especially with a large training set.

Data set description:

The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by



"re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.

The set of images in the MNIST database is a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively.

Some researchers have achieved "near-human performance" on the MNIST database, using a committee of neural networks; in the same paper, the authors achieve performance double that of humans on other recognition tasks. The highest error rate listed on the original website of the database is 12 percent, which is achieved using a simple linear classifier with no preprocessing.

In 2004, a best-case error rate of 0.42 percent was achieved on the database by researchers using a new classifier called the LIRA, which is a neural classifier with three neuron layers based on Rosenblatt's perceptron principles.

Some researchers have tested artificial intelligence systems using the database put under random distortions. The systems in these cases are usually neural networks and the distortions used tend to be either affine distortions or elastic distortions. Sometimes, these systems can be very successful; one such system achieved an error rate on the database of 0.39 percent.

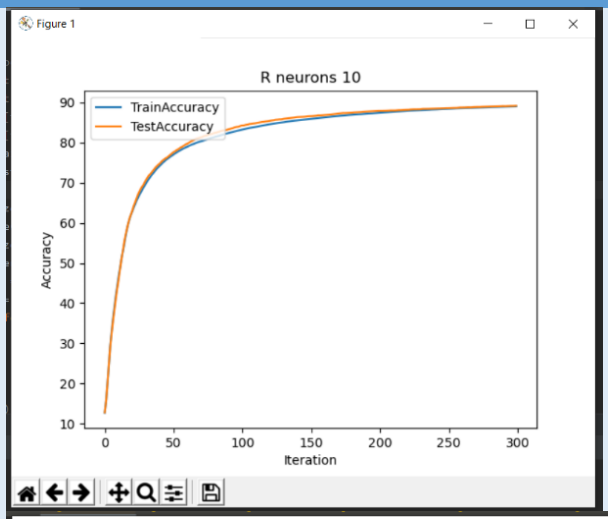
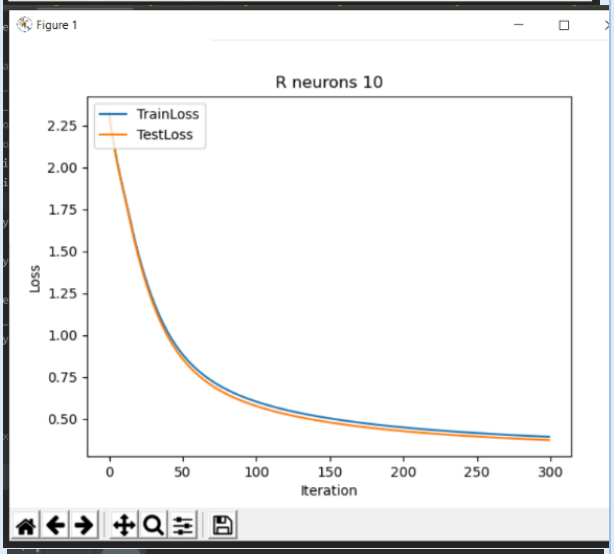
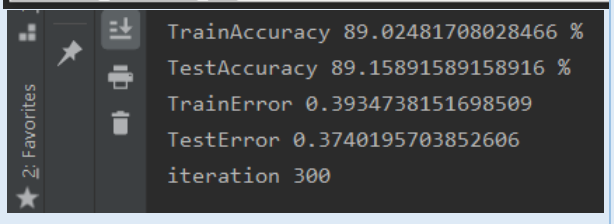
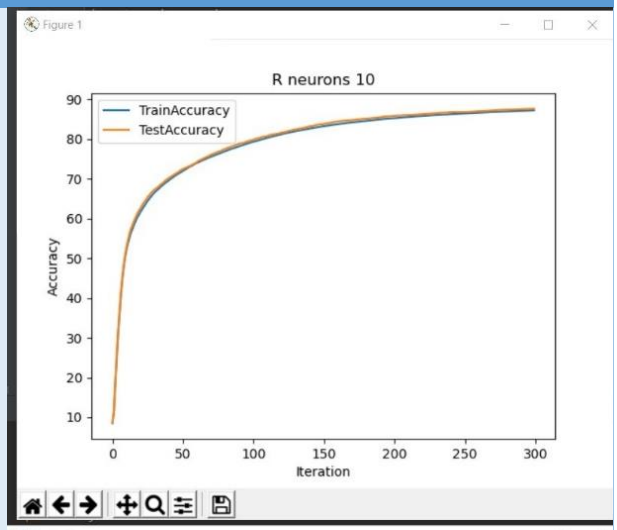
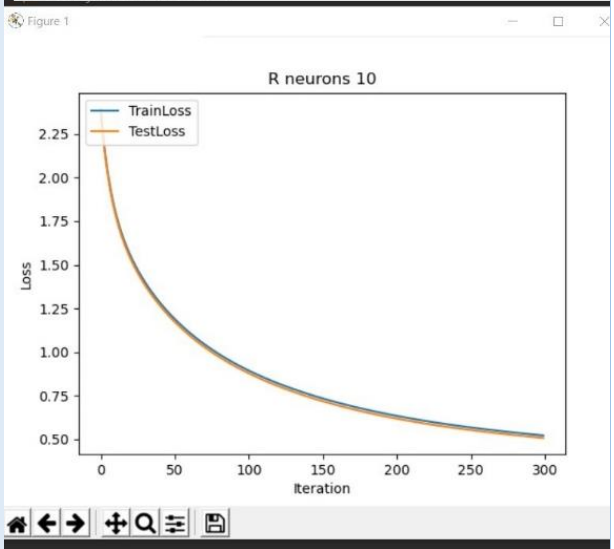
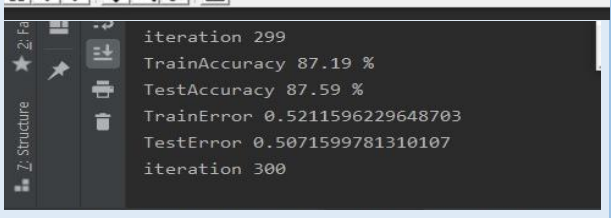
In 2011, an error rate of 0.27 percent, improving on the previous best result, was reported by researchers using a similar system of neural networks. In 2013, an approach based on regularization of neural networks using DropConnect has been

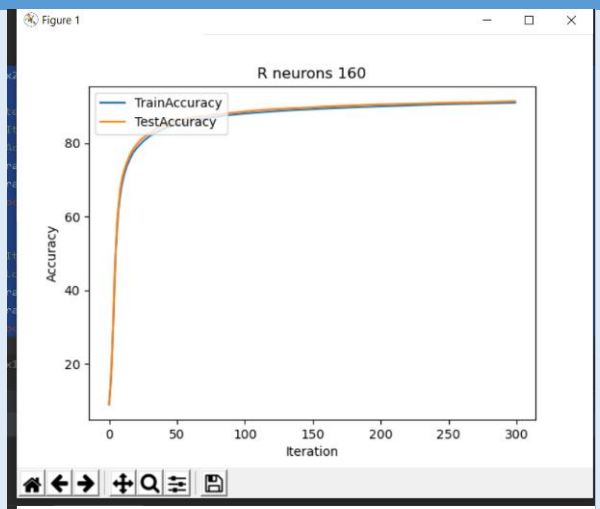
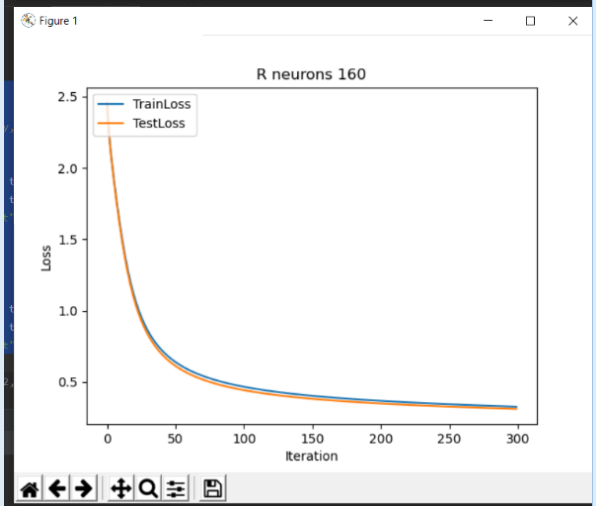
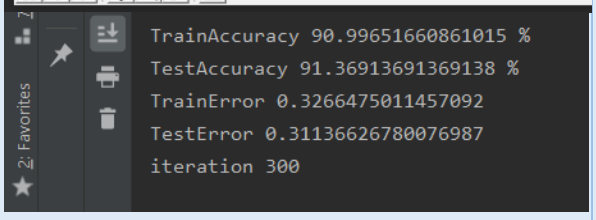
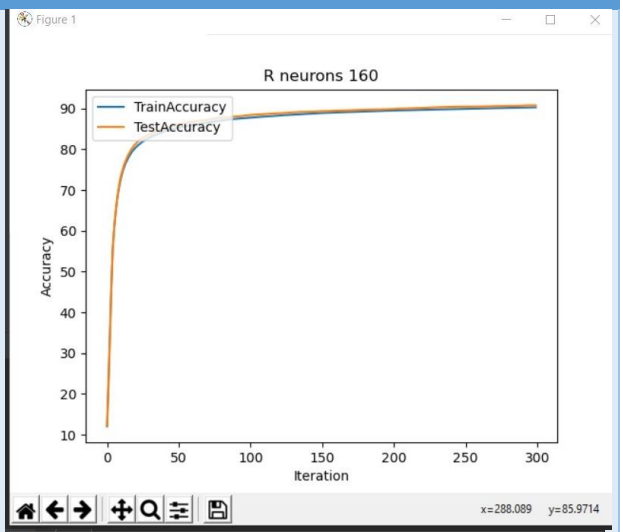
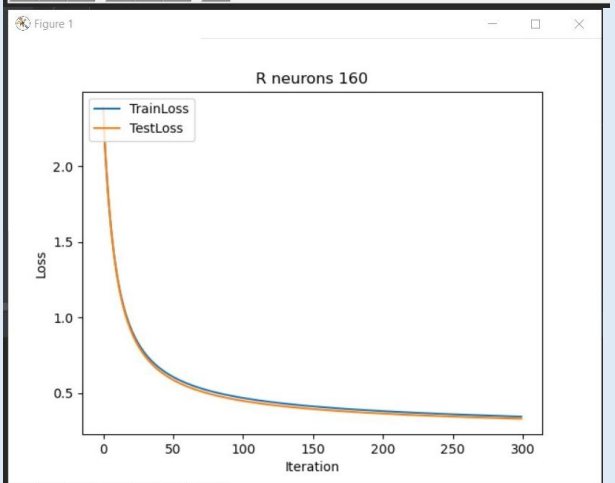
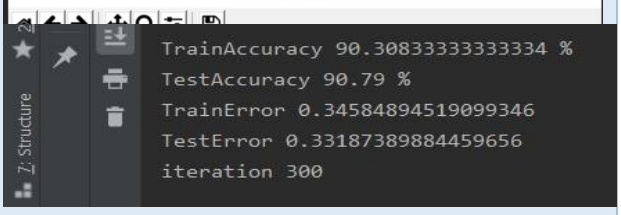
claimed to achieve a 0.21 percent error rate. In 2016 the single convolutional neural network best performance was 0.31 percent error rate. As of August 2018, the best performance of a single convolutional neural network trained on MNIST training data using realtime data augmentation is 0.26 percent error rate. Also, the Parallel Computing Center (Khmelnitskiy, Ukraine) obtained an ensemble of only 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate. Some images in the testing dataset are barely readable and may prevent reaching test error rates of 0%. In 2018 researchers from Department of System and Information Engineering, University of Virginia announced 0.18% error with simultaneous stacked three kind of neural networks.

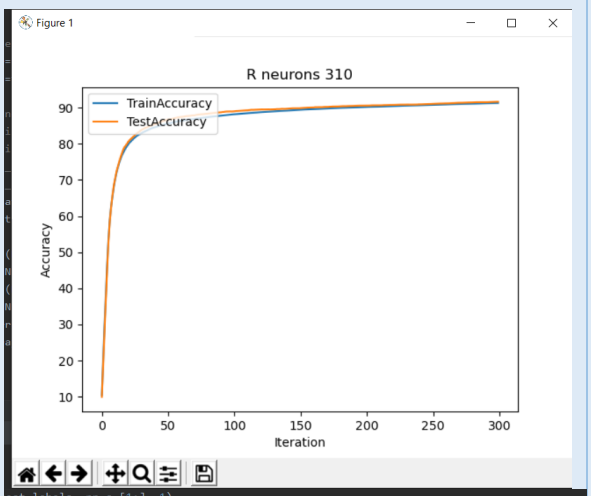
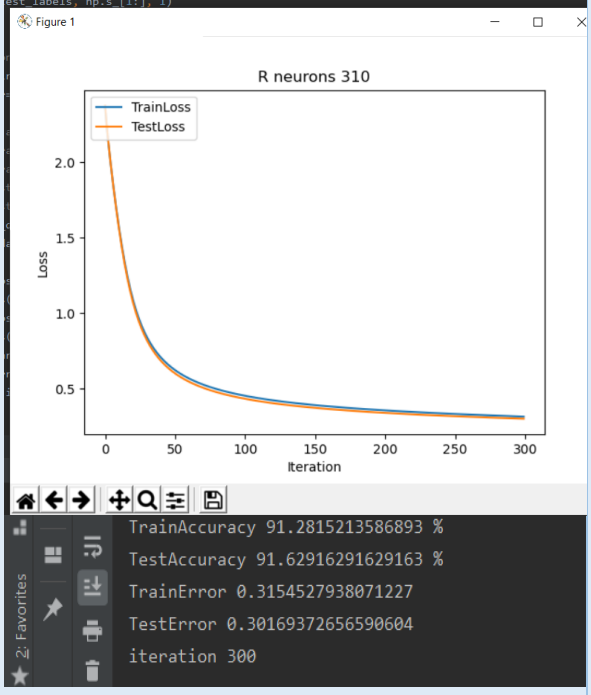
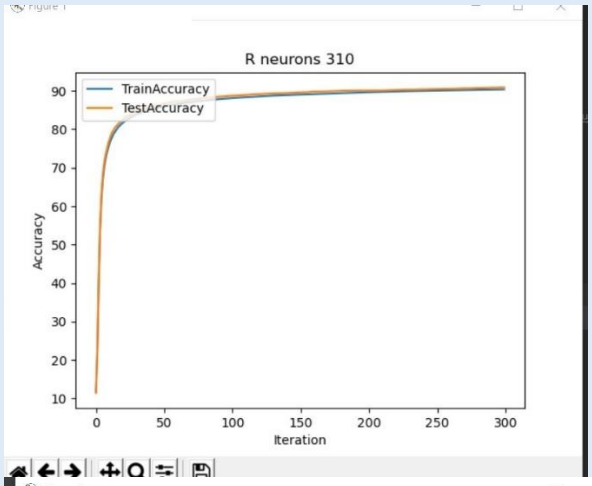
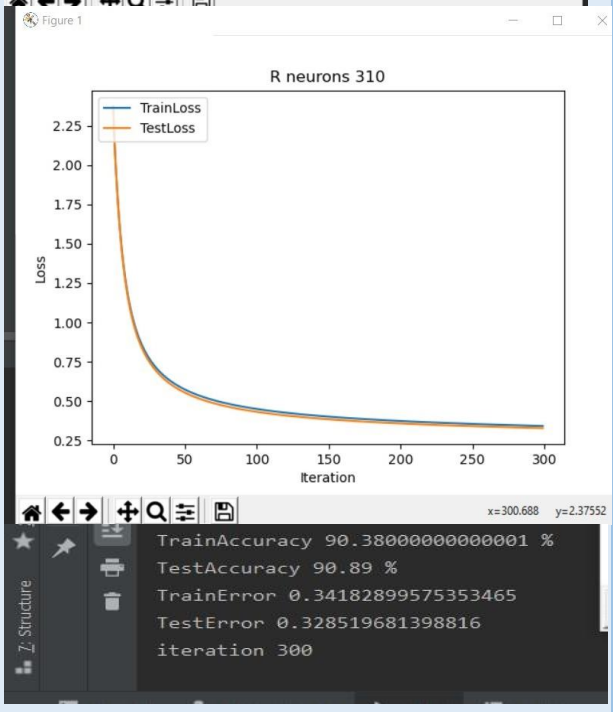
Experimental results & discussions:

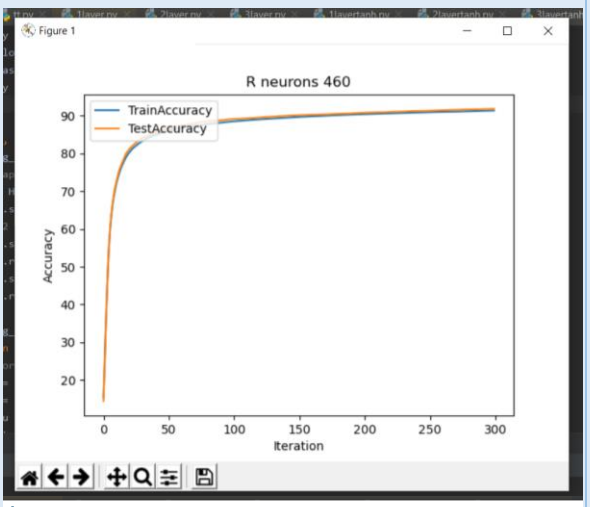
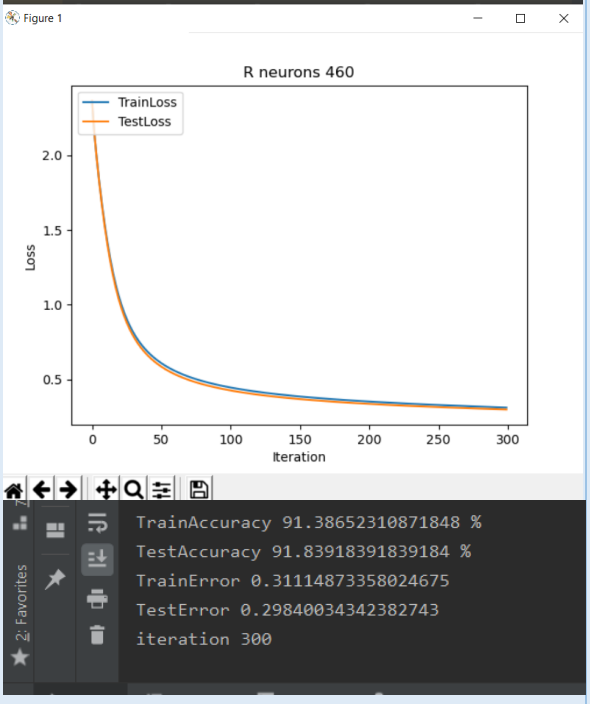
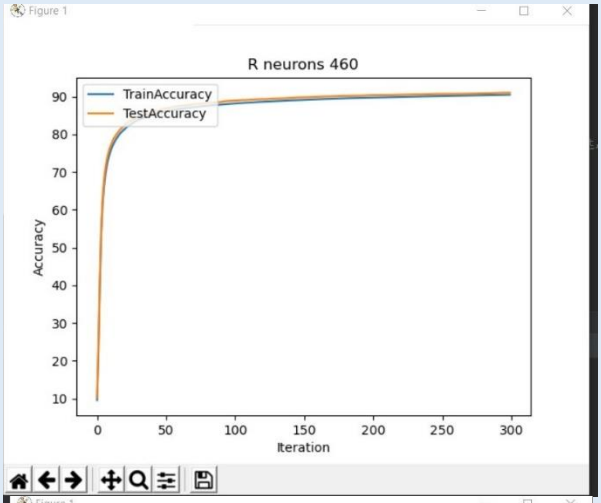
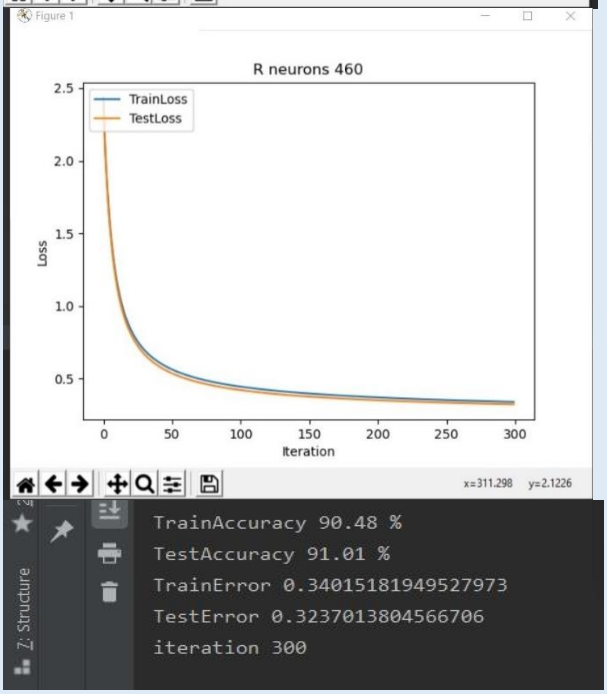
One Layer

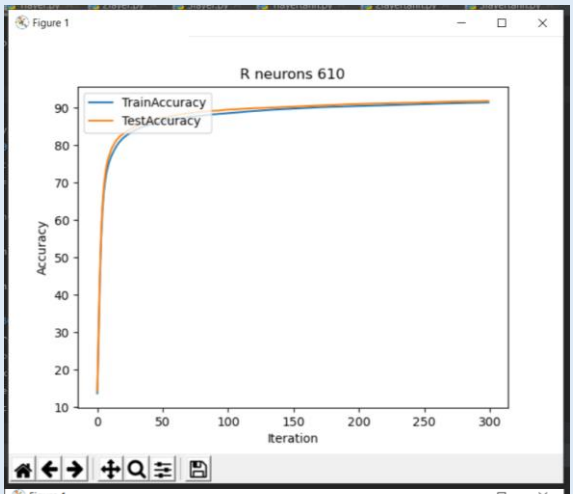
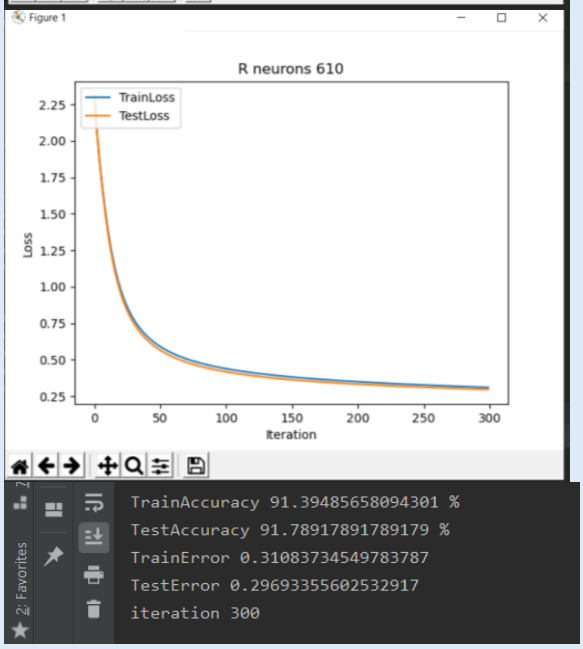
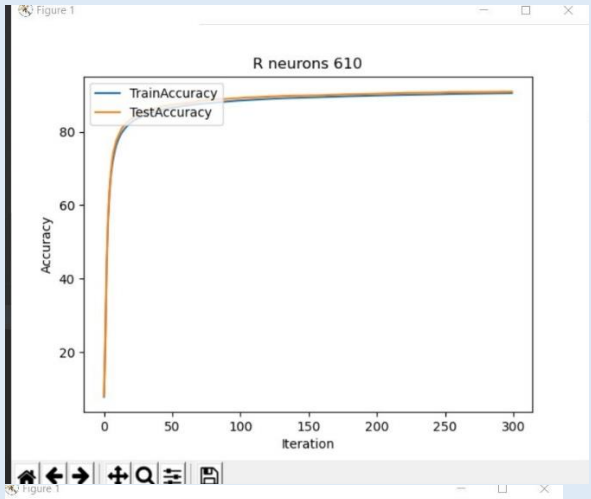
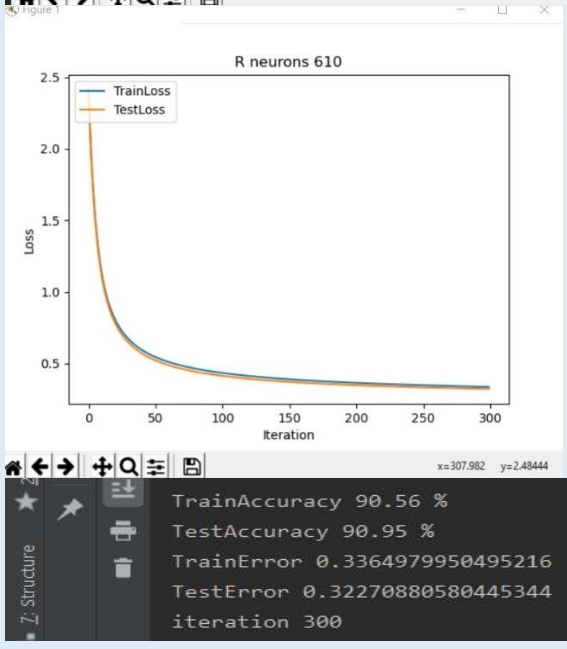
As we can observe that in the 1-layers the Relu exceeded the Tanh in the accuracy.

R neurons	Relu	Tanh
10	   Time Taken : 1.01 minute	   Time Taken : 1.53 minute

R neurons	Relu	Tanh
160	   Time Taken : 7.3 minute	   Time Taken : 7.85 minute

R neurons	Relu	Tanh
310	  Time Taken : 14.27 minute	  Time Taken : 14.7 minute

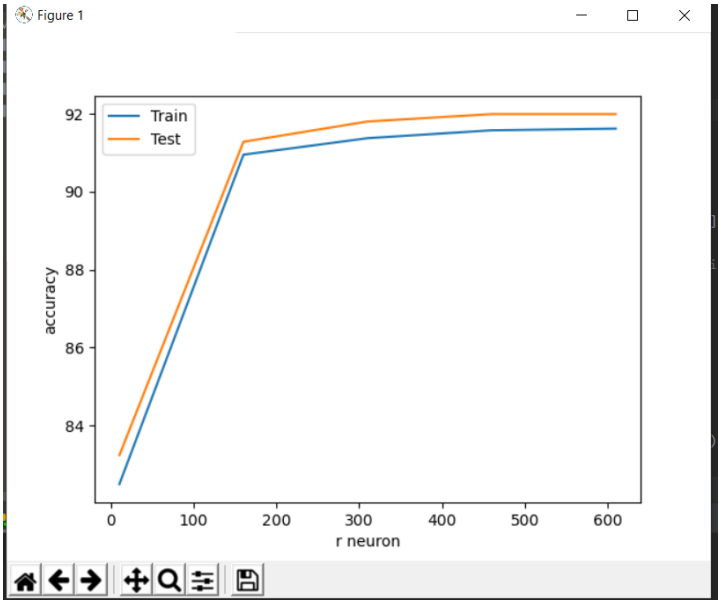
R neurons	Relu	Tanh
460	  Time Taken : 21.18 minute	  Time Taken : 21.5 minute

R neurons	Relu	Tanh
610	  Time Taken : 28.09 minute	  Time Taken : 29.01 minute

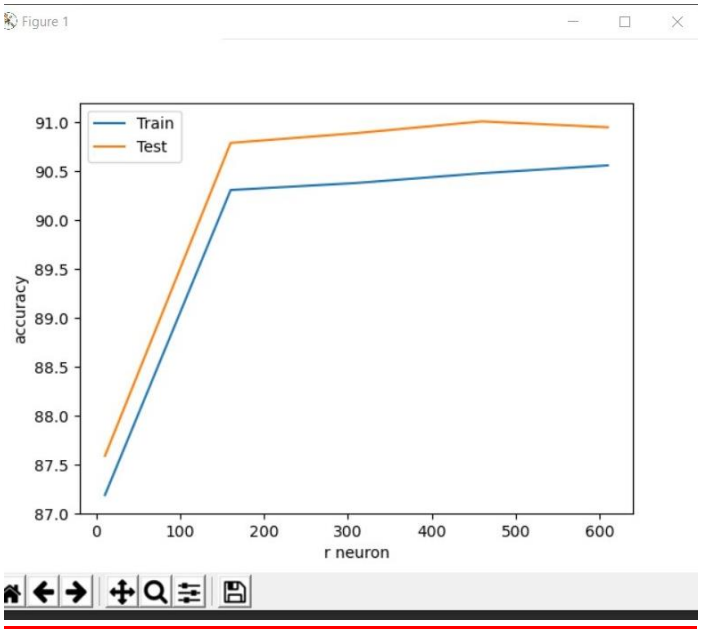
One Layer

Accuracy vs R neurons

Relu:

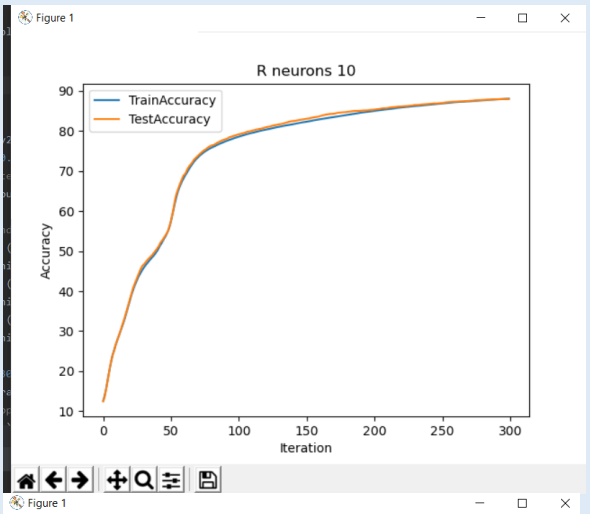
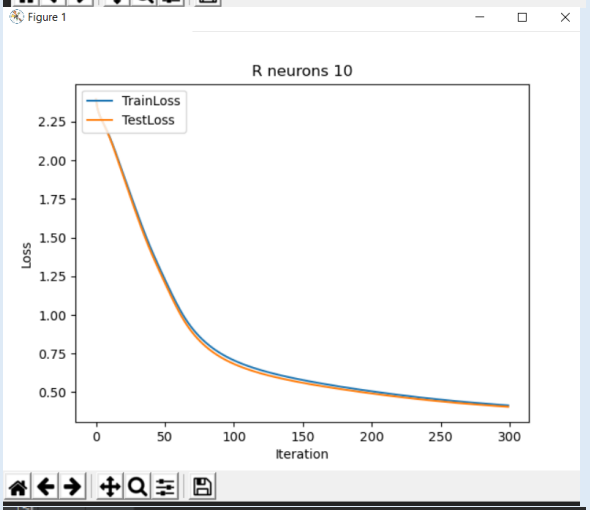
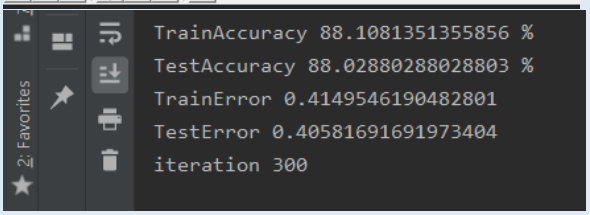
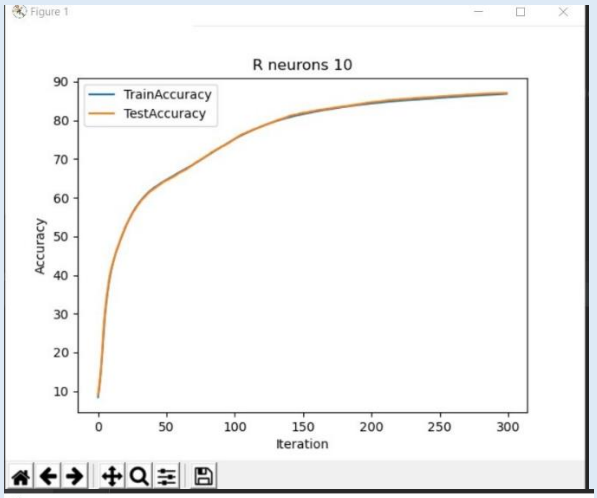
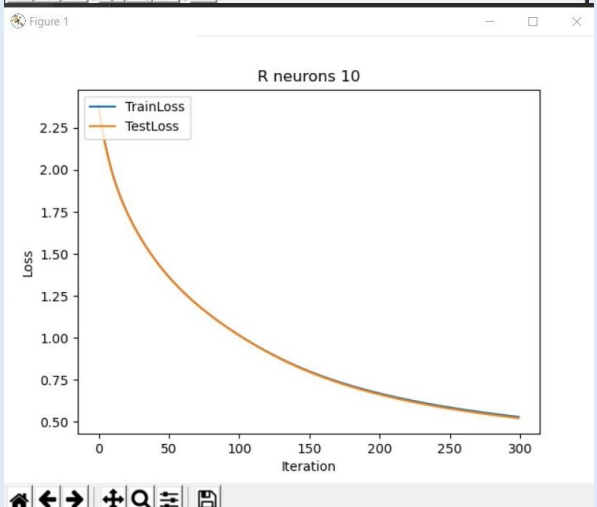
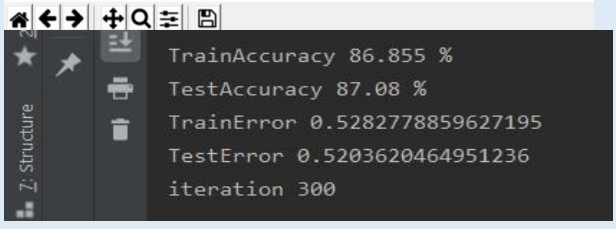


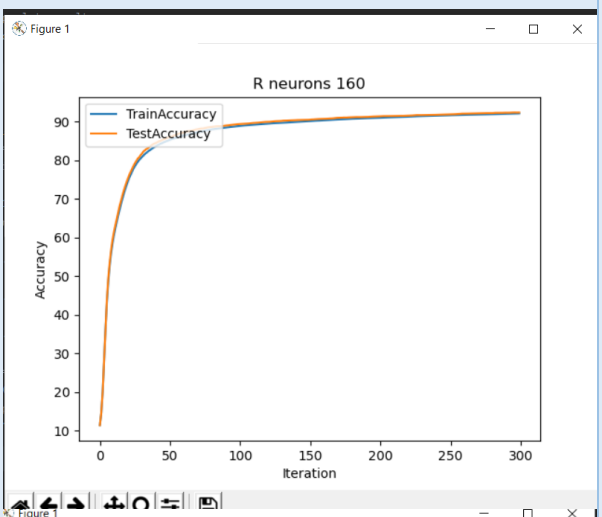
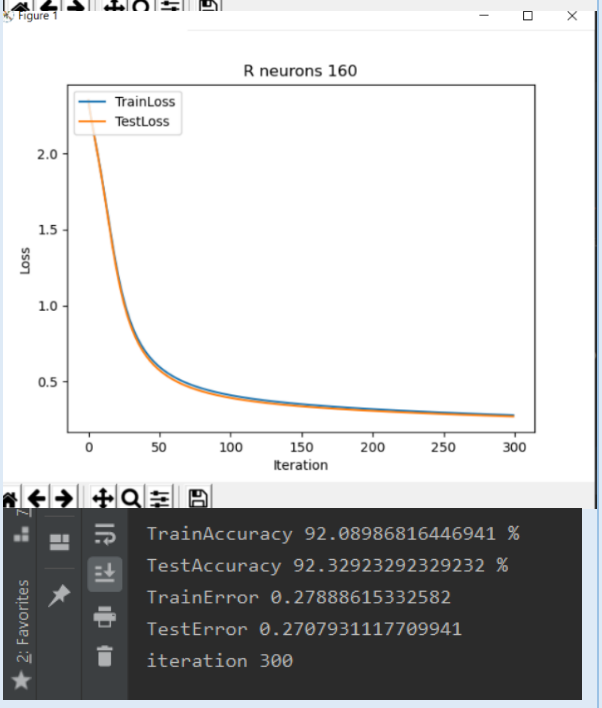
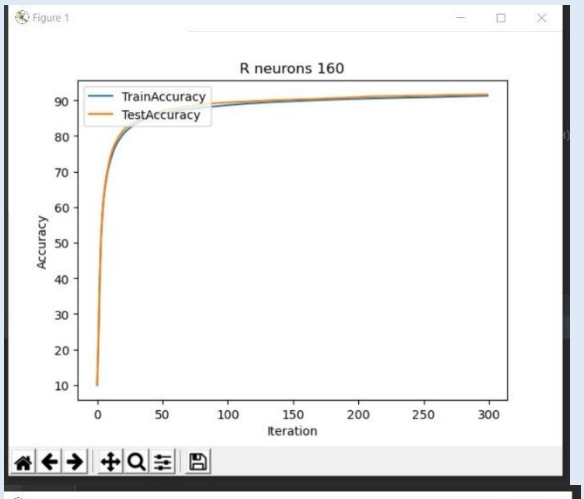
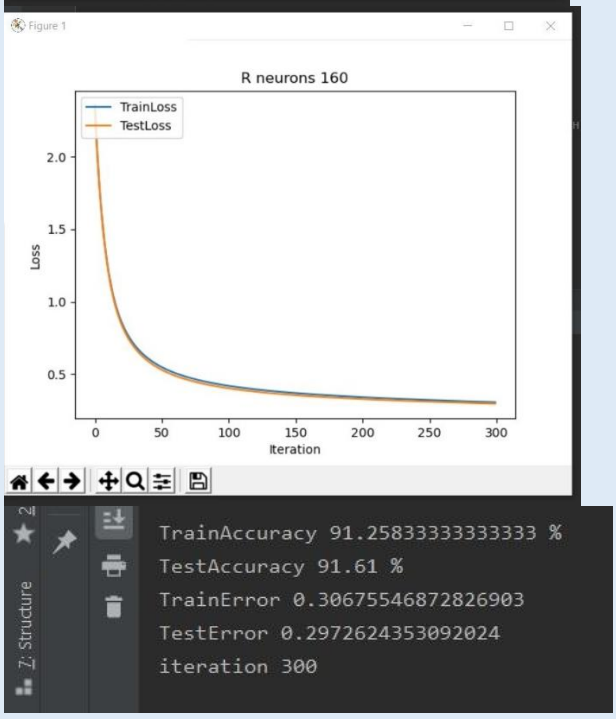
Tanh:

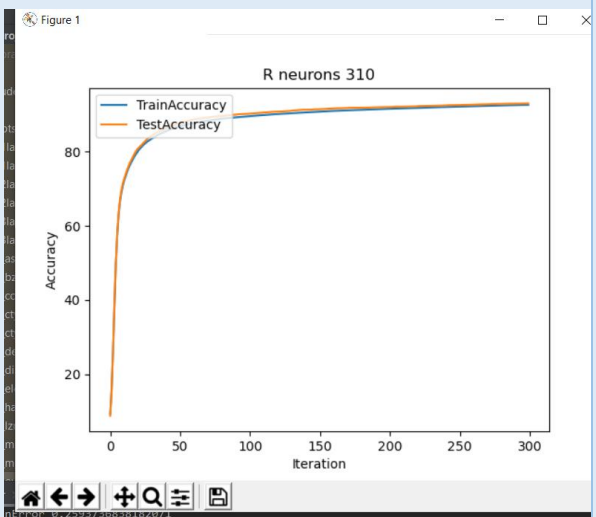
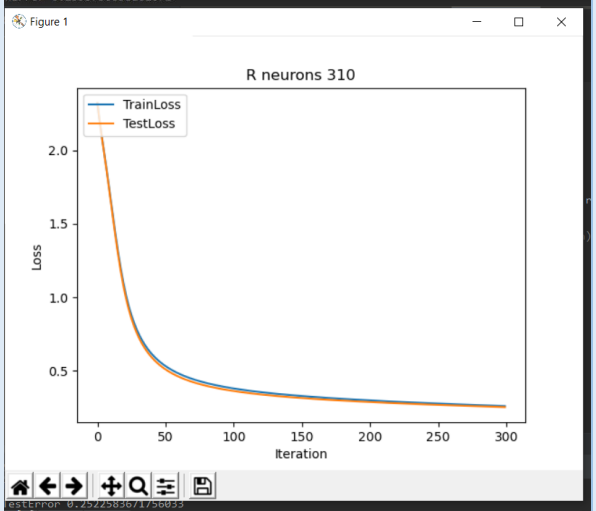
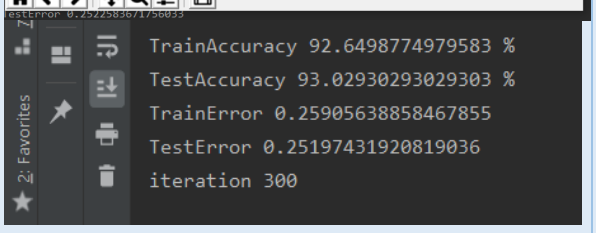
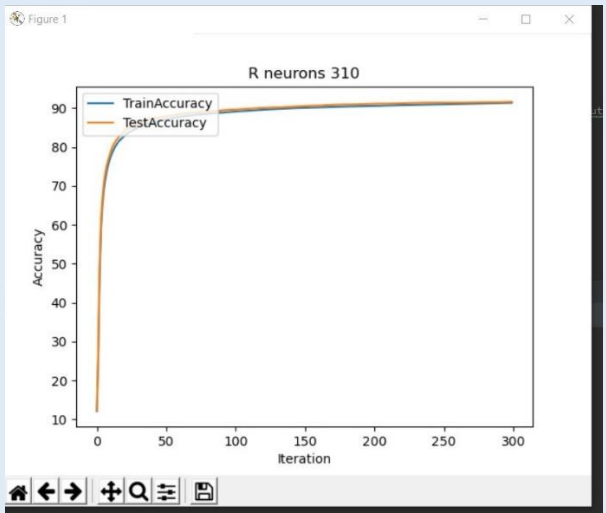
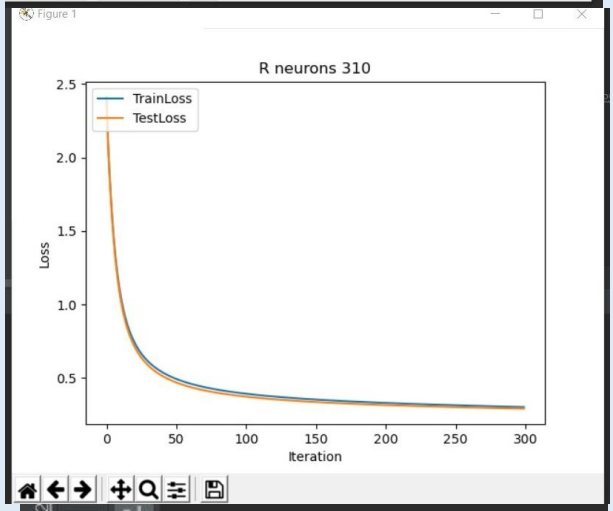
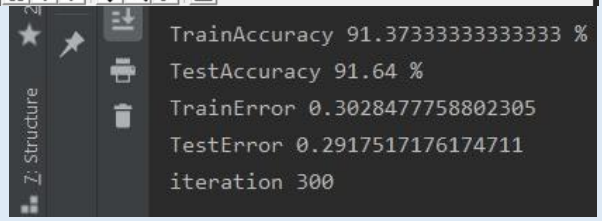


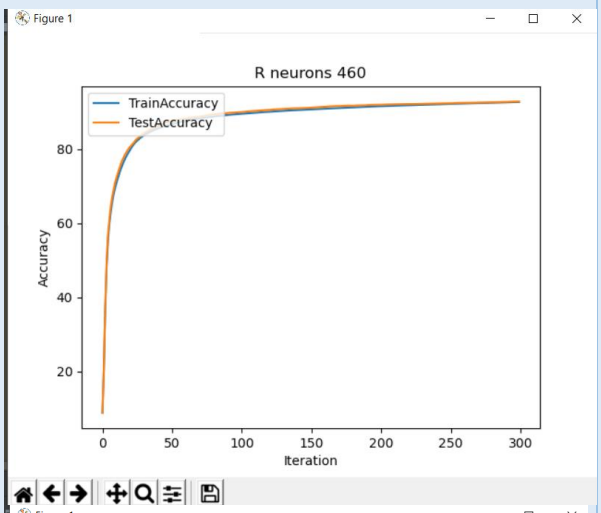
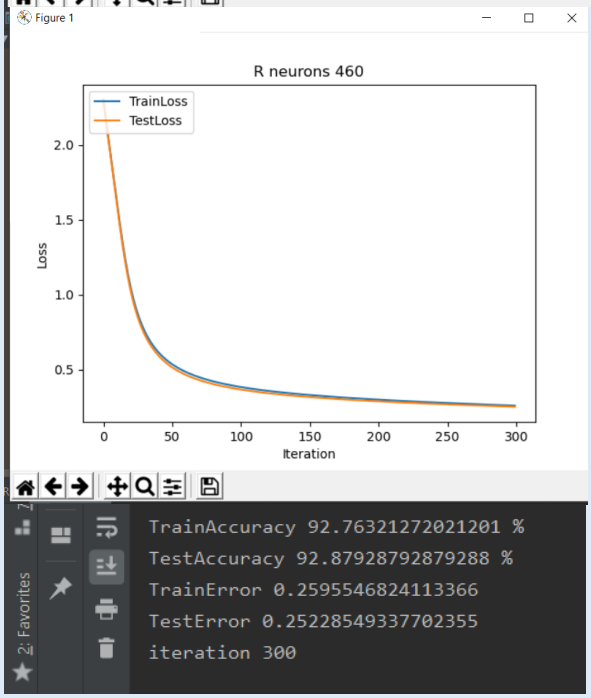
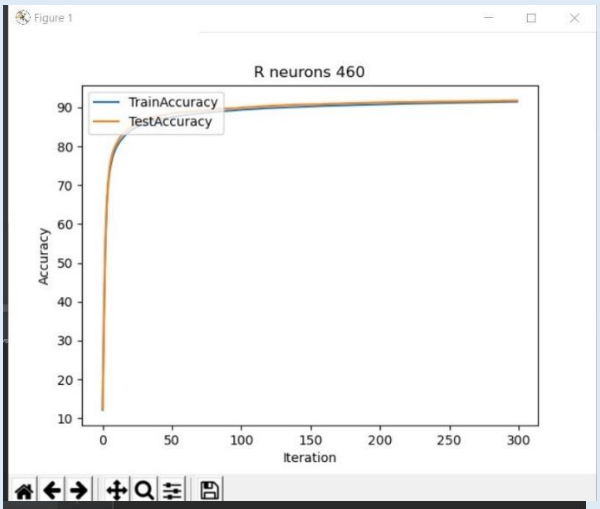
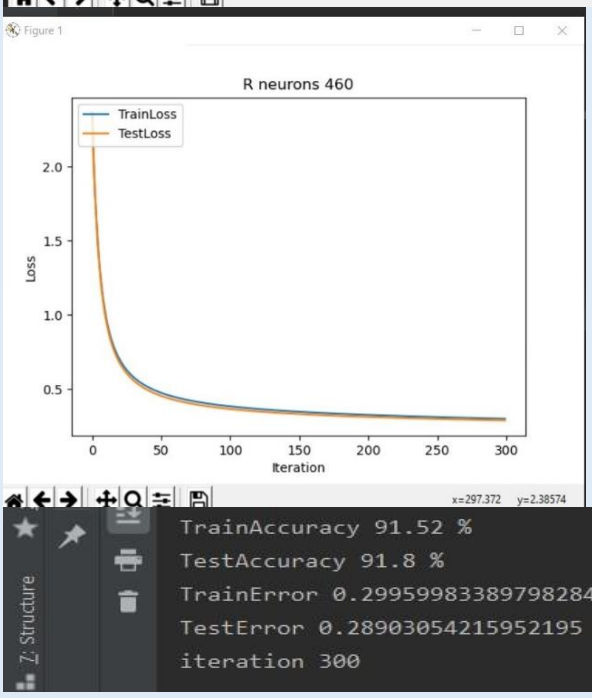
Two Layer

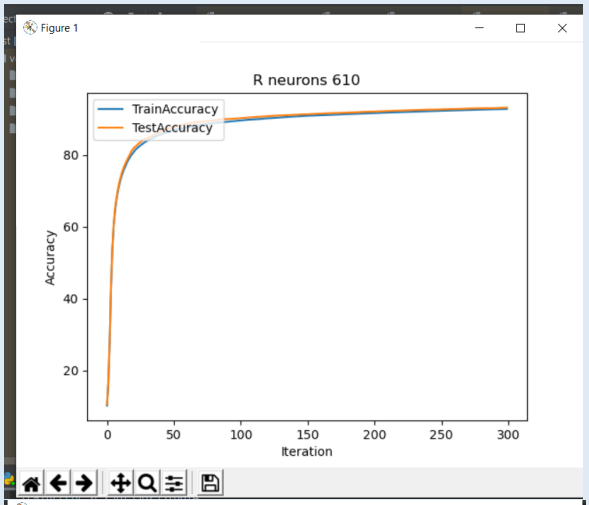
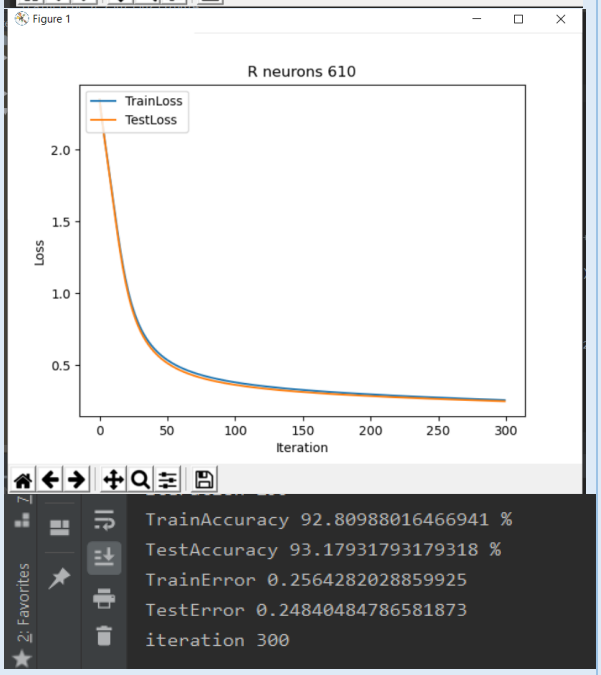
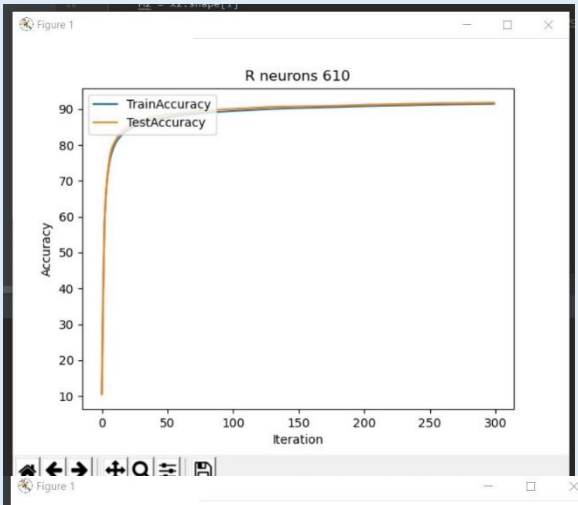
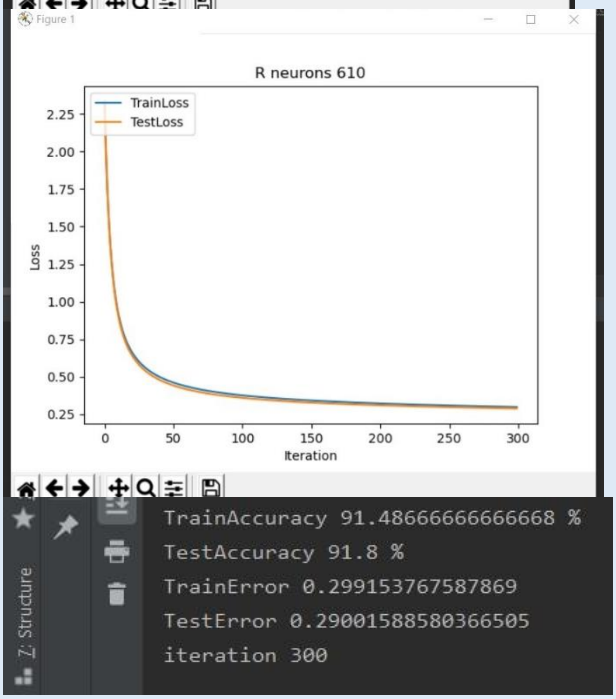
As we can observe that in the 2-layers the Relu exceeded the Tanh in the accuracy.

R neurons	Relu	Tanh
10	   Time Taken : 1.5 minute	   Time Taken : 1.8 minute

R neurons	Relu	Tanh
160	  Time Taken : 7.9 minute	  Time Taken : 8.01 minute

R neurons	Relu	Tanh
310	   Time Taken : 14.6 minute	   Time Taken : 14.8 minute

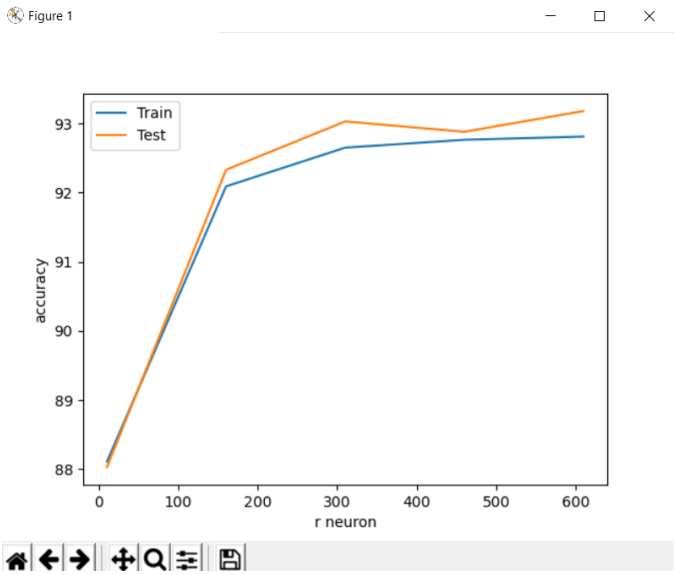
R neurons	Relu	Tanh
460	  Time Taken : 21.6 minute	  Time Taken : 21.8 minute

R neurons	Relu	Tanh
610	  Time Taken : 28.5 minute	  Time Taken : 28.8 minute

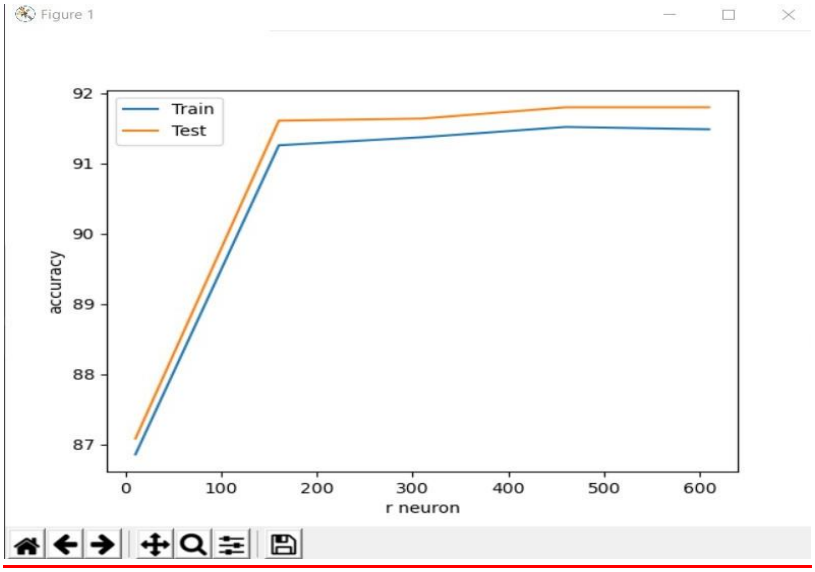
Two Layer

Accuracy vs R neurons

Relu:

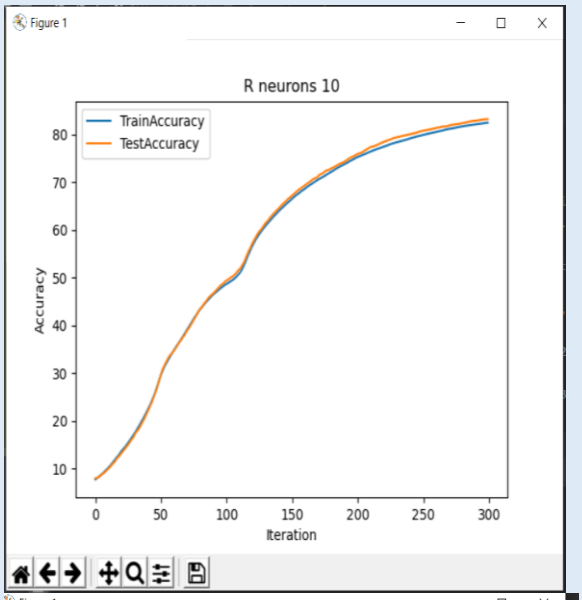
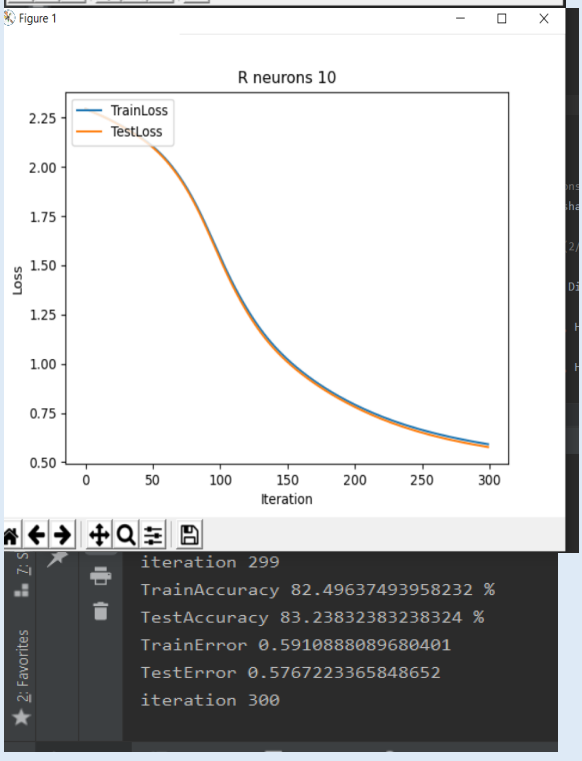
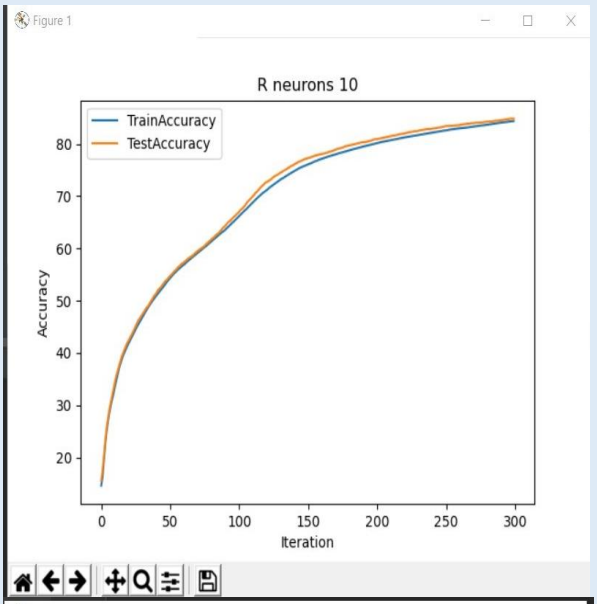
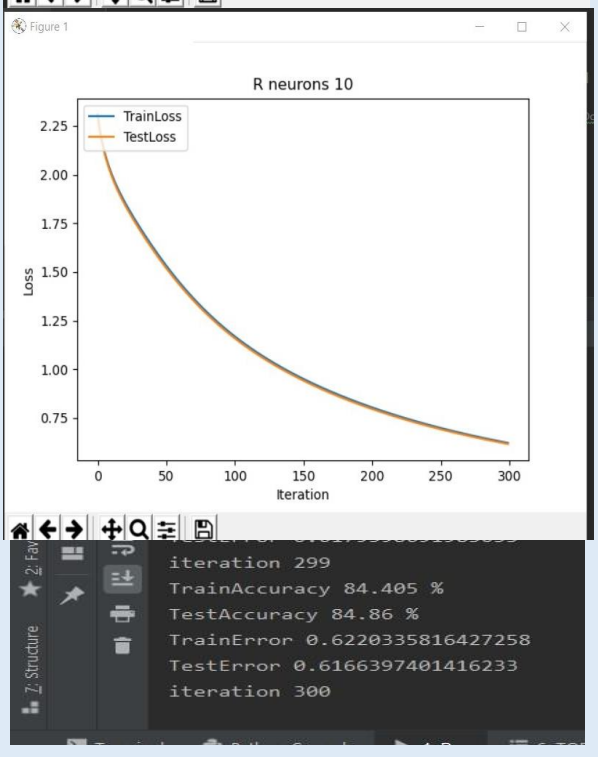


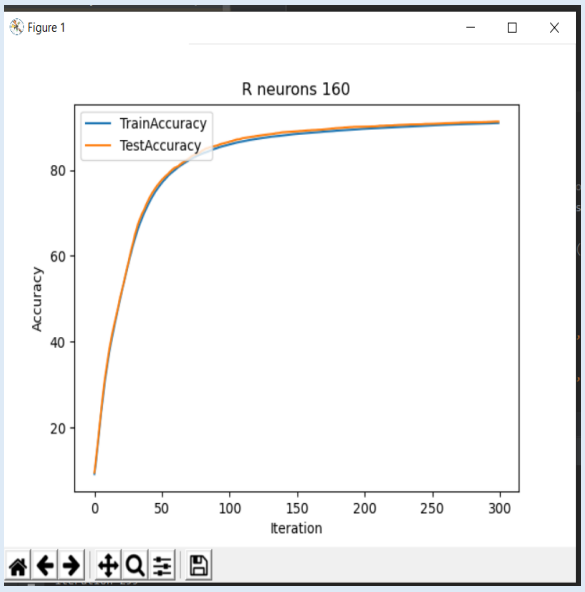
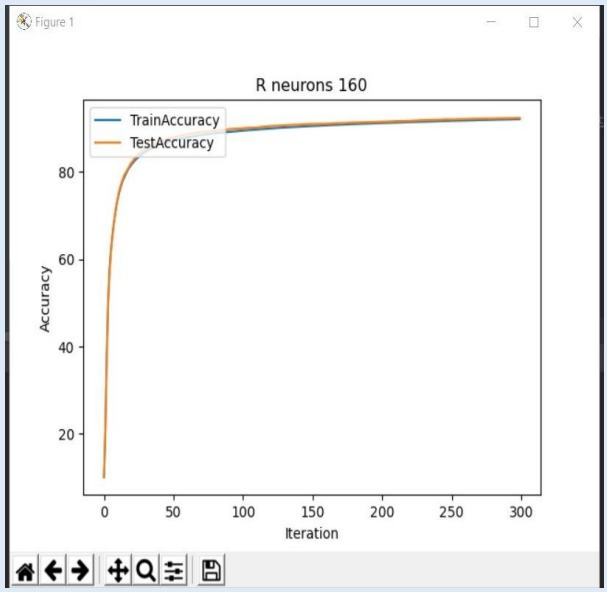
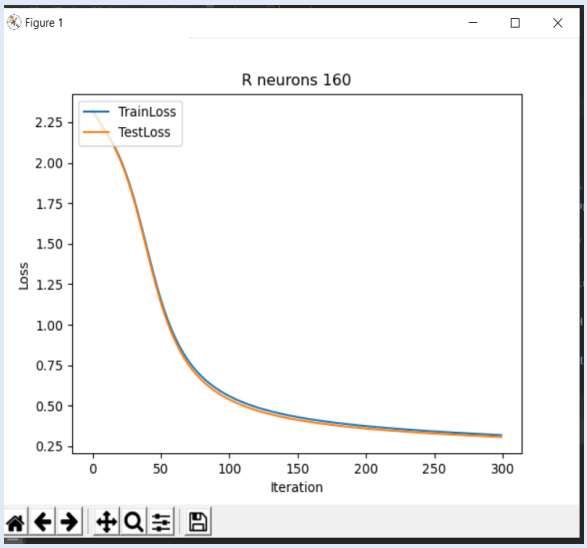
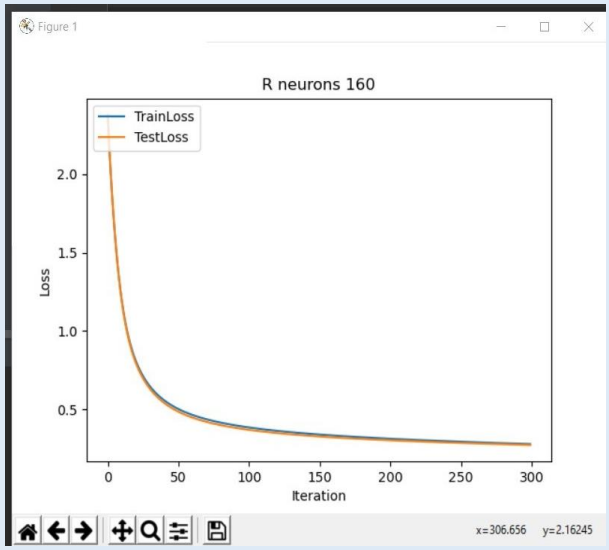
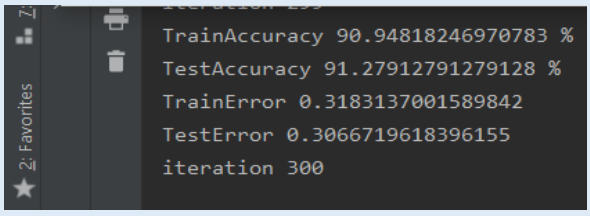
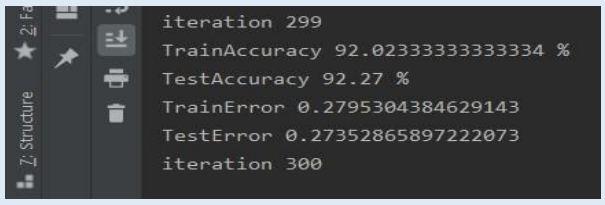
Tanh:

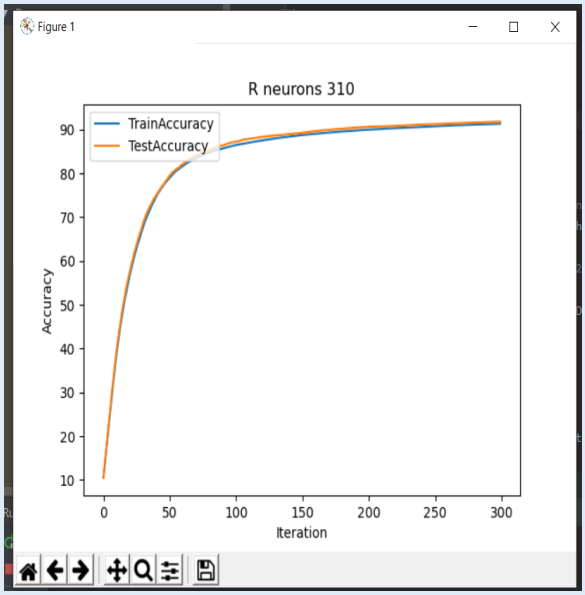
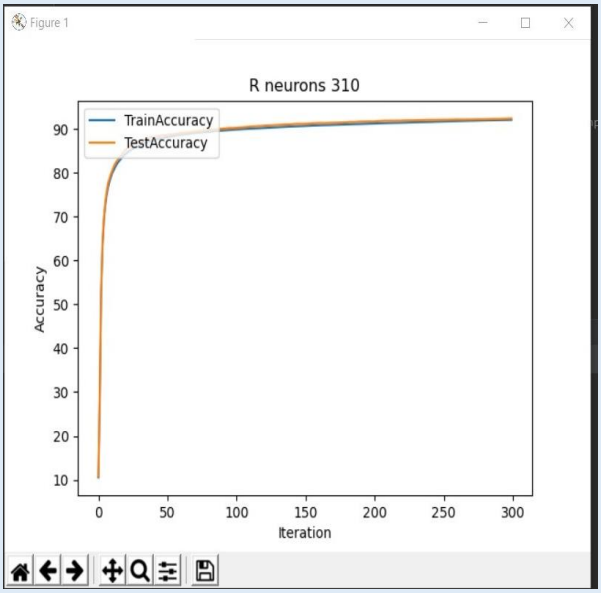
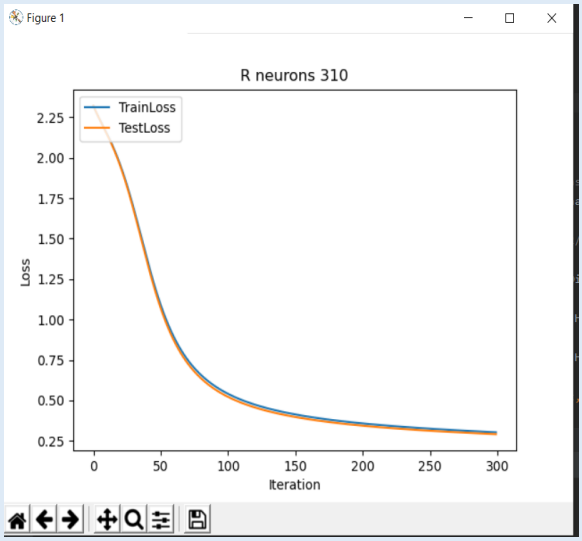
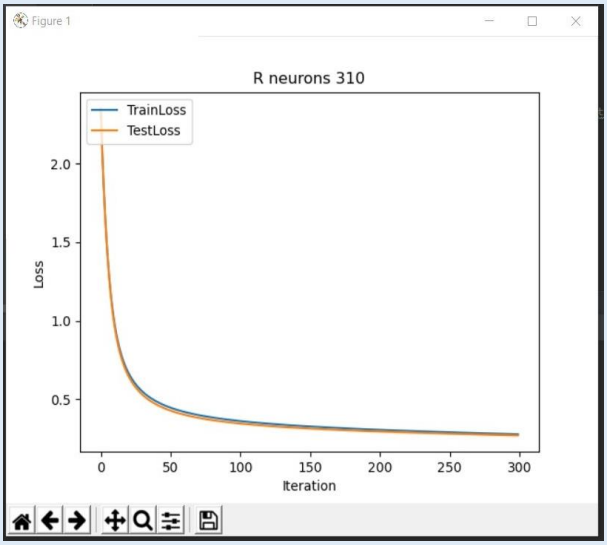
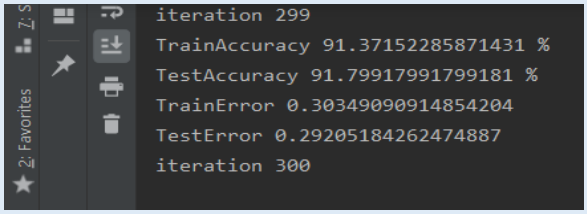
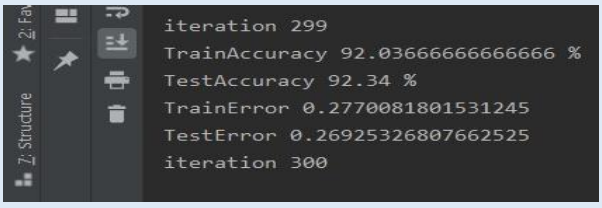


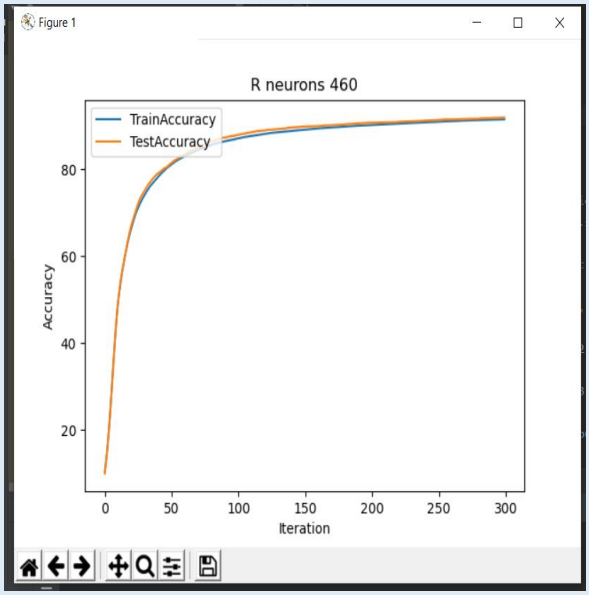
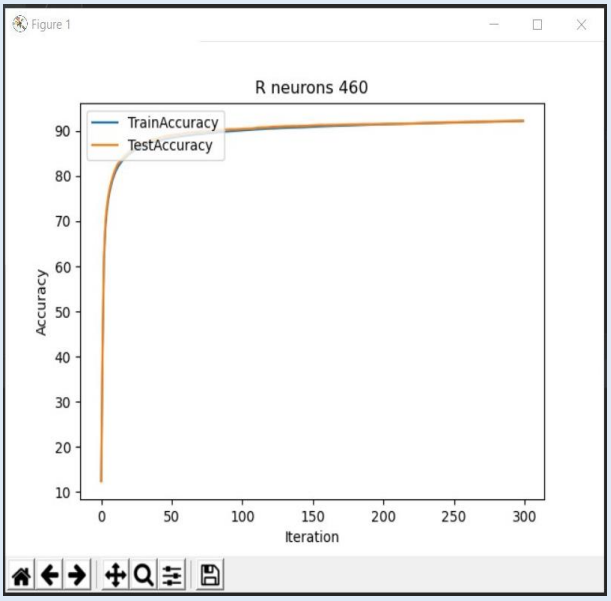
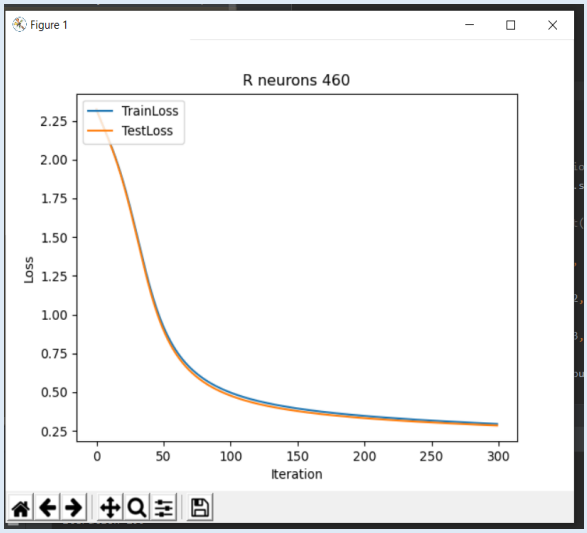
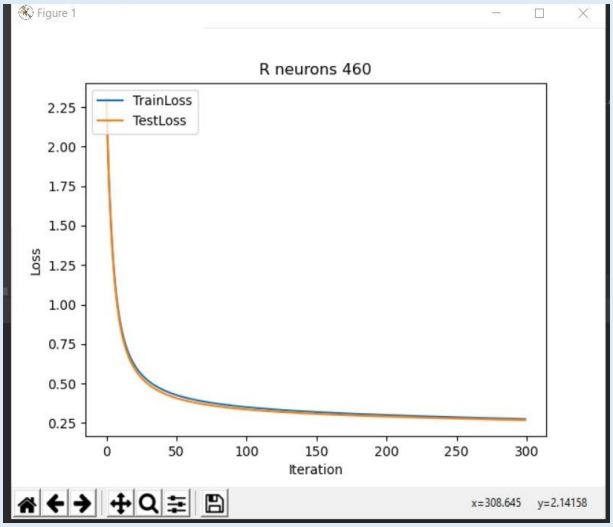
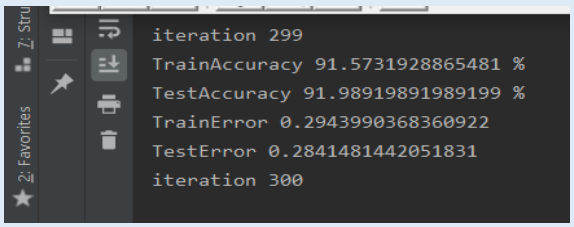
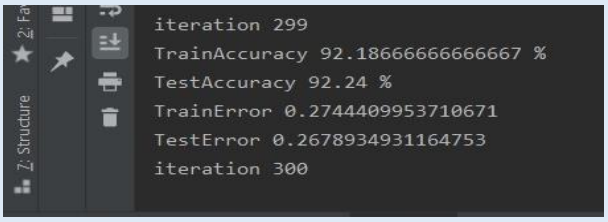
3-Layer

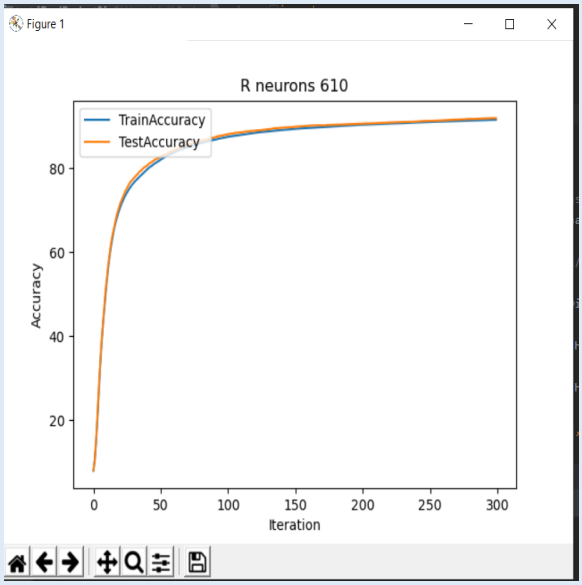
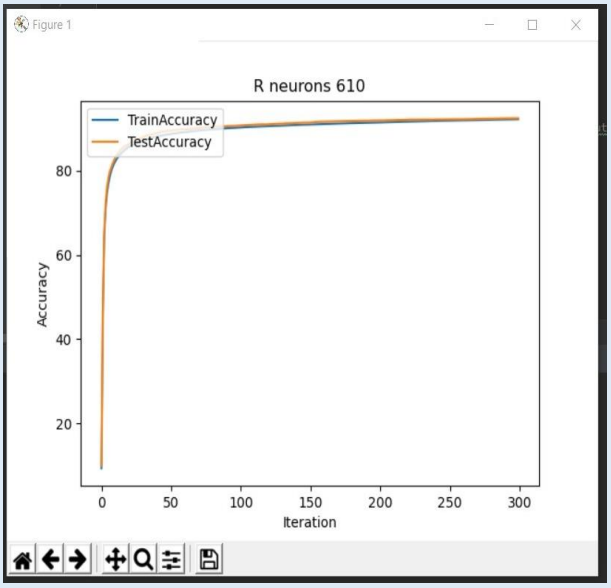
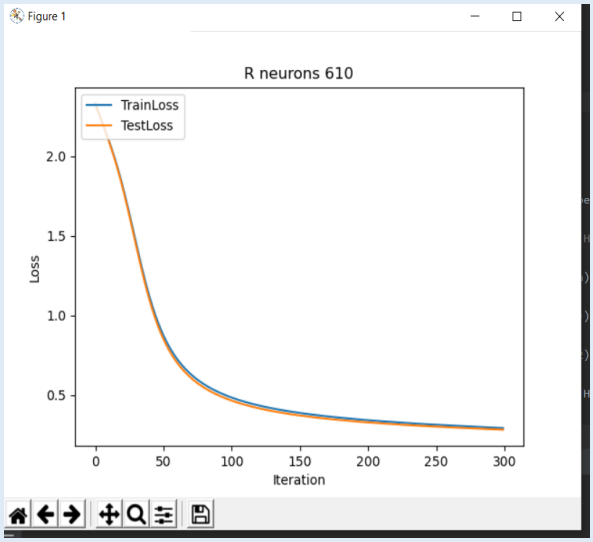
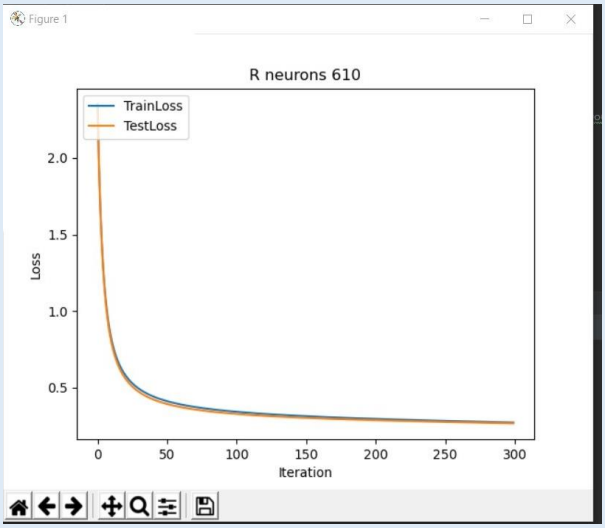
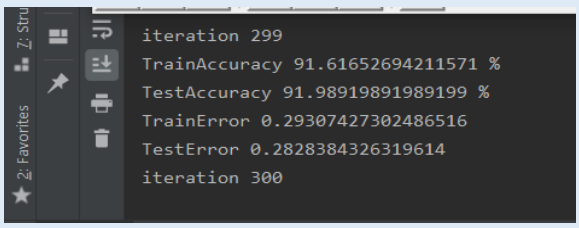
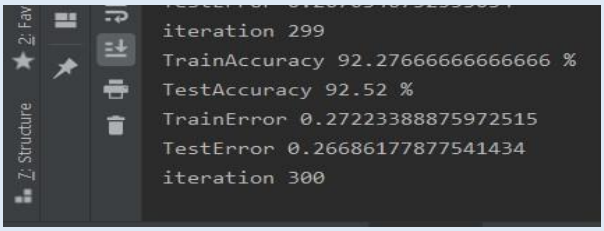
As we can observe that in the 3-layers the Tanh exceeded the Relu in the accuracy.

R neurons	Relu	Tanh
10	  Time Taken : 2.03 minute	  Time Taken : 2.3 minute

R neurons	Relu	Tanh
160		
		
		
	Time Taken : 8.1 minute	Time Taken : 8.45 minute

R neurons	Relu	Tanh
310		
		
		
	Time Taken : 14.9 minute	Time Taken : 15.22 minute

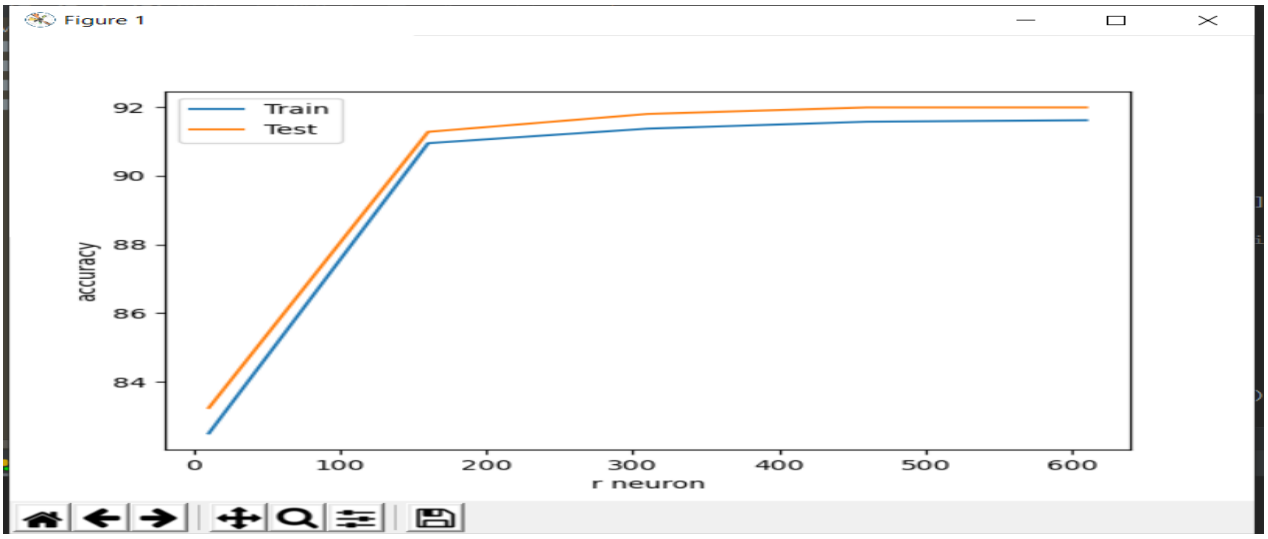
R neurons	Relu	Tanh
460		
		
		
	Time Taken : 21.95 minute	Time Taken : 22.15 minute

R neurons	Relu	Tanh
610		
		
		
	Time Taken : 30.1 minute	Time Taken : 31.43 minute

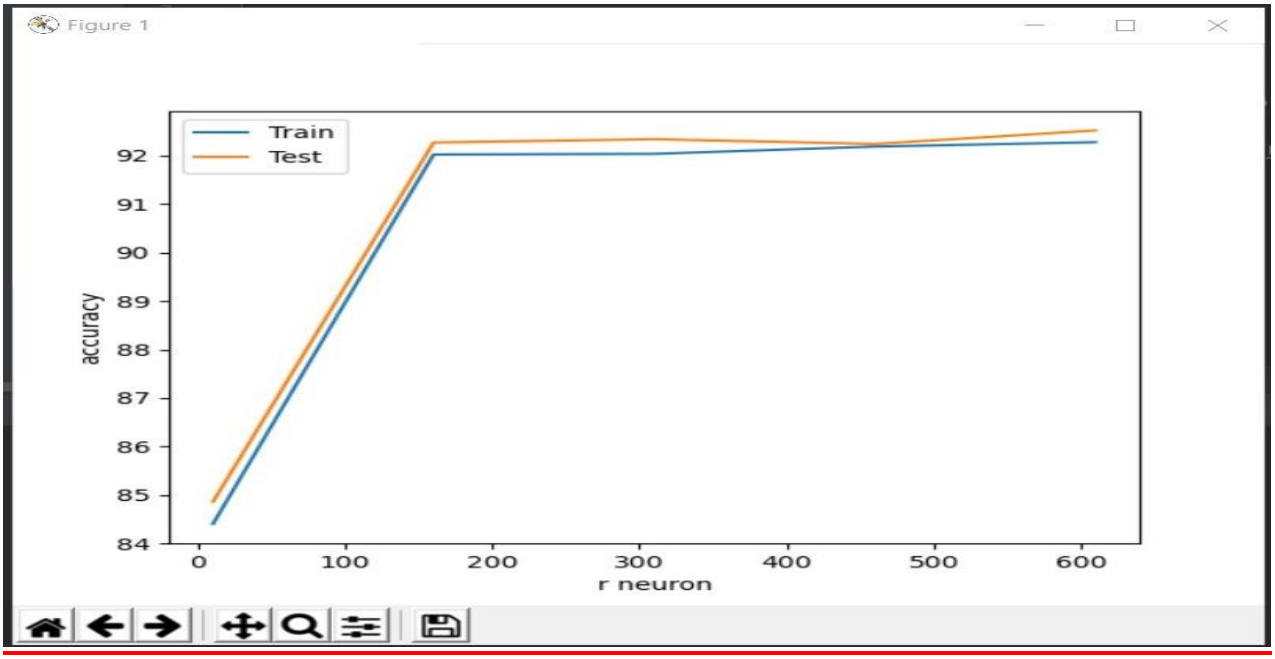
3-Layer

Accuracy vs R-neurons

Relu:



Tanh:



Appendix with codes

Relu 1 layer

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620

def error(x, y, x2, y2, r):
    learning_rate = 0.1
    # give appropriate number to the dimensions
    M, Din, H, Dout = x.shape[1], x.shape[0], r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + Dout))
    w2 = np.random.uniform(-R2, R2, size=(Dout, H))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.where(Z1 <= 0, 0, Z1)
        relu = w2.dot(A1)
        softmaxAnswer = (softmax(relu))

        AccsoftmaxAnswer = (softmax(relu)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(relu, y)

        # forward prop.for test data
        Z2 = w1.dot(x2)
        A2 = np.where(Z2 <= 0, 0, Z2)
        relu2 = w2.dot(A2)

        softmaxAnswer2 = softmax(relu2)

        AccsoftmaxAnswer2 = (softmax(relu2)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
None]).astype(int)
```

```
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(relu2, y2)

        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw2 = dY_hat.dot(A1.T)
        dA1 = (w2.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ1 = dA1 * (Z1 > 0).astype(np.int)  # or (A1>0)
        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross

        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2

        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)




def softmax(x):

    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0, keepdims=True)

def rfunction(x, y, x2, y2, r):

    r_array = [i for i in range(10, r, 150)]

    for i in range(10,r,150):
        print("R neuron= ",i)
        error(x, y, x2, y2, i)
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
```

```
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()


    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()







def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data

train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
```

```
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
lr = np.arange(10)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)

# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```

Relu 2 layers

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620


def error(x, y, x2, y2,r):
    learning_rate = 0.1
    # give appropriate number to the dimensions
    M, Din, H, H2, Dout = x.shape[1], x.shape[0], r, r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + H2))
    w2 = np.random.uniform(-R2, R2, size=(H2, H))
    R3 = np.sqrt(6 / (Dout + H2))
    w3 = np.random.uniform(-R3, R3, size=(Dout, H2))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.where(Z1 <= 0, 0, Z1)
        relu1 = w2.dot(A1)
        A2 = np.where(relu1 <= 0, 0, relu1)
        relu2 = w3.dot(A2)

        softmaxAnswer = (softmax(relu2))

        AccsoftmaxAnswer = (softmax(relu2)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(relu2, y)
        # forward prop.for test data
        Z1T = w1.dot(x2)
        A1T = np.where(Z1T <= 0, 0, Z1T)
        relu1T = w2.dot(A1T)
        A2T = np.where(relu1T <= 0, 0, relu1T)
        relu2T = w3.dot(A2T)

        softmaxAnswer2 = softmax(relu2T)

        AccsoftmaxAnswer2 = (softmax(relu2T)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
```

```
None]).astype(int)
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(relu2T, y2)

        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw3 = dY_hat.dot(A2.T)
        dA2 = (w3.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ2 = dA2 * (relu1 > 0).astype(np.int)  # or (A1>0)
        dw2 = dZ2.dot(A1.T)
        dA1 = (w2.T).dot(dZ2)
        dZ1 = dA1 * (Z1 > 0).astype(np.int)  # or (A1>0)
        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        w3 -= (learning_rate * dw3)

        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross


        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2
        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)


def softmax(x):

    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0, keepdims=True)


def rfunction(x, y, x2, y2, r):
    r_array = [i for i in range(10, r, 150)]

    for i in range(10, r, 150):
        print("R neuron= ", i)
        error(x, y, x2, y2, i)
```

```
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()


def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data

train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
```

```
lr = np.arange(10)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)
#
# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```

Relu 3 layers

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620


def error(x, y, x2, y2,r):
    learning_rate = 0.05
    # give appropriate number to the dimensions
    M, Din, H, H2, H3, Dout = x.shape[1], x.shape[0], r, r, r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + H2))
    w2 = np.random.uniform(-R2, R2, size=(H2, H))
    R3 = np.sqrt(6 / (H3 + H2))
    w3 = np.random.uniform(-R3, R3, size=(H3, H2))
    R4 = np.sqrt(6 / (Dout + H3))
    w4 = np.random.uniform(-R4, R4, size=(Dout, H3))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.where(Z1 <= 0, 0, Z1)
        relu1 = w2.dot(A1)
        A2 = np.where(relu1 <= 0, 0, relu1)
        relu2 = w3.dot(A2)
        A3 = np.where(relu2 <= 0, 0, relu2)
        relu3 = w4.dot(A3)

        softmaxAnswer = (softmax(relu3))

        AccsoftmaxAnswer = (softmax(relu3)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(relu3, y)
        # forward prop.for test data
        Z1T = w1.dot(x2)
        A1T = np.where(Z1T <= 0, 0, Z1T)
        relu1T = w2.dot(A1T)
        A2T = np.where(relu1T <= 0, 0, relu1T)
        relu2T = w3.dot(A2T)
        A3T = np.where(relu2T <= 0, 0, relu2T)
```

```
        relu3T = w4.dot(A3T)

        softmaxAnswer2 = softmax(relu3T)

        AccsoftmaxAnswer2 = (softmax(relu3T)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
None]).astype(int)
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(relu3T, y2)

        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw4 = dY_hat.dot(A3.T)
        dA3 = (w4.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ3 = dA3 * (relu2 > 0).astype(np.int)  # or (A1>0)
        dw3 = dZ3.dot(A2.T)
        dA2 = (w3.T).dot(dZ3)
        dZ2 = dA2 * (relu1 > 0).astype(np.int)  # or (A1>0)
        dw2 = dZ2.dot(A1.T)
        dA1 = (w2.T).dot(dZ2)
        dZ1 = dA1 * (Z1 > 0).astype(np.int)  # or (A1>0)
        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        w3 -= (learning_rate * dw3)
        w4 -= (learning_rate * dw4)

        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross


        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2
        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)

def softmax(x):

    e_x = np.exp(x)
```

```
    return e_x / e_x.sum(axis=0, keepdims=True)


def rfunction(x, y, x2, y2, r):
    r_array = [i for i in range(10, r, 150)]

    for i in range(10, r, 150):
        print("R neuron= ", i)
        error(x, y, x2, y2, i)
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()


def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data
```

```
train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
lr = np.arange(10)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)
#
# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```

Tanh 1 layer

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620


def error(x, y, x2, y2,r):
    learning_rate = 0.1
    # give appropriate number to the dimensions
    M, Din, H, Dout = x.shape[1], x.shape[0], r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + Dout))
    w2 = np.random.uniform(-R2, R2, size=(Dout, H))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.tanh(Z1)
        tanh1 = w2.dot(A1)
        softmaxAnswer = (softmax(tanh1))

        AccsoftmaxAnswer = (softmax(tanh1)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(tanh1, y)

        # forward prop.for test data
        Z2 = w1.dot(x2)
        A2 = np.tanh(Z2)
        tanh2T = w2.dot(A2)

        softmaxAnswer2 = softmax(tanh2T)

        AccsoftmaxAnswer2 = (softmax(tanh2T)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
None]).astype(int)
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(tanh2T, y2)
```

```
        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw2 = dY_hat.dot(A1.T)
        dA1 = (w2.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ1 = dA1 * (1.0 - (A1) ** 2)

        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross


        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2
        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)




def softmax(x):

    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0, keepdims=True)


def rfunction(x, y, x2, y2, r):
    r_array = [i for i in range(10, r, 150)]

    for i in range(10, r, 150):
        print("R neuron= ", i)
        error(x, y, x2, y2, i)
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()
```

```
        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()


def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data

train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
lr = np.arange(10)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
```

```
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)
#
# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```

Tanh 2 layers

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620


def error(x, y, x2, y2,r):
    learning_rate = 0.1
    # give appropriate number to the dimensions
    M, Din, H, H2, Dout = x.shape[1], x.shape[0], r, r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + H2))
    w2 = np.random.uniform(-R2, R2, size=(H2, H))
    R3 = np.sqrt(6 / (Dout + H2))
    w3 = np.random.uniform(-R3, R3, size=(Dout, H2))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.tanh(Z1)
        relu1 = w2.dot(A1)
        A2 = np.tanh(relu1)
        relu2 = w3.dot(A2)

        softmaxAnswer = (softmax(relu2))

        AccsoftmaxAnswer = (softmax(relu2)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(relu2, y)
        # forward prop.for test data
        Z1T = w1.dot(x2)
        A1T = np.tanh(Z1T)
        relu1T = w2.dot(A1T)
        A2T = np.tanh(relu1T)
        relu2T = w3.dot(A2T)

        softmaxAnswer2 = softmax(relu2T)

        AccsoftmaxAnswer2 = (softmax(relu2T)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
```

```
None]).astype(int)
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(relu2T, y2)

        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw3 = dY_hat.dot(A2.T)
        dA2 = (w3.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ2 = dA2 * (1.0 - (A2) ** 2)
        dw2 = dZ2.dot(A1.T)
        dA1 = (w2.T).dot(dZ2)
        dZ1 = dA1 * (1.0 - (A1) ** 2)
        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        w3 -= (learning_rate * dw3)

        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross

        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2
        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)


def softmax(x):

    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0, keepdims=True)


def rfunction(x, y, x2, y2, r):
    r_array = [i for i in range(10, r, 150)]

    for i in range(10, r, 150):
        print("R neuron= ", i)
        error(x, y, x2, y2, i)
        plt.xlabel('Iteration')
```

```
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()


def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data

train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
lr = np.arange(10)
```

```
# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)
#
# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```

Tanh 3 layers

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import array
r=620


def error(x, y, x2, y2,r):
    learning_rate = 0.1
    # give appropriate number to the dimensions
    M, Din, H, H2, H3, Dout = x.shape[1], x.shape[0], r, r, r, y.shape[0]
    M2 = x2.shape[1]
    # w1, w2 = np.random.rand(H,Din)*np.sqrt(2/(H+Din)),
np.random.rand(Dout,H)*np.sqrt(2/(Dout+H))
    R1 = np.sqrt(6 / (H + Din))
    w1 = np.random.uniform(-R1, R1, size=(H, Din))
    R2 = np.sqrt(6 / (H + H2))
    w2 = np.random.uniform(-R2, R2, size=(H2, H))
    R3 = np.sqrt(6 / (H3 + H2))
    w3 = np.random.uniform(-R3, R3, size=(H3, H2))
    R4 = np.sqrt(6 / (Dout + H3))
    w4 = np.random.uniform(-R4, R4, size=(Dout, H3))
    p = 0
    training_iter = 300
    for i in range(training_iter):
        # forward prop.
        Z1 = w1.dot(x)
        A1 = np.tanh(Z1)
        relu1 = w2.dot(A1)
        A2 = np.tanh(relu1)
        relu2 = w3.dot(A2)
        A3 = np.tanh(relu2)
        relu3 = w4.dot(A3)

        softmaxAnswer = (softmax(relu3))

        AccsoftmaxAnswer = (softmax(relu3)).T
        y_hat = (AccsoftmaxAnswer == AccsoftmaxAnswer.max(axis=1)[:,
None]).astype(int)
        accuracy = (np.sum(np.multiply(y_hat.T, y)) / M) * 100
        # accuracy = (y_hat.T == y).mean()
        print('TrainAccuracy', accuracy, "%")

        Cross = cross_entropy(relu3, y)
        # forward prop.for test data
        Z1T = w1.dot(x2)
        A1T = np.tanh(Z1T)
        relu1T = w2.dot(A1T)
        A2T = np.tanh(relu1T)
        relu2T = w3.dot(A2T)
        A3T = np.tanh(relu2T)
```

```
        relu3T = w4.dot(A3T)

        softmaxAnswer2 = softmax(relu3T)

        AccsoftmaxAnswer2 = (softmax(relu3T)).T
        y_hat2 = (AccsoftmaxAnswer2 == AccsoftmaxAnswer2.max(axis=1)[:,
None]).astype(int)
        accuracy2 = (np.sum(np.multiply(y_hat2.T, y2)) / M2) * 100
        # accuracy2 = (y_hat2.T == y2).mean()
        print('TestAccuracy', accuracy2, "%")

        TestCross = cross_entropy(relu3T, y2)

        # back prop.
        dY_hat = (softmaxAnswer - y) / M
        dw4 = dY_hat.dot(A3.T)
        dA3 = (w4.T).dot(dY_hat)  # this is the backpropgation through the layer
        dZ3 = dA3 * (1.0 - (A3) ** 2)
        dw3 = dZ3.dot(A2.T)
        dA2 = (w3.T).dot(dZ3)
        dZ2 = dA2 * (1.0 - (A2) ** 2)
        dw2 = dZ2.dot(A1.T)
        dA1 = (w2.T).dot(dZ2)
        dZ1 = dA1 * (1.0 - (A1) ** 2)
        dw1 = dZ1.dot(x.T)
        dX = (w1.T).dot(dZ1)

        # gd
        w1 -= (learning_rate * dw1)
        w2 -= (learning_rate * dw2)
        w3 -= (learning_rate * dw3)
        w4 -= (learning_rate * dw4)

        # print("w1", w1)
        # print("w2", w2)
        print("TrainError", Cross)

        # print("Lastw1", w1)
        # print("Lastw2", w2)
        print("TestError", TestCross)

        trainloss_array[p] = Cross
        testloss_array[p] = TestCross

        traincost_array[p] = accuracy
        testcost_array[p] = accuracy2
        p += 1
        print("iteration", p)
    train_errorrate_array.append(accuracy)
    test_errorrate_array.append(accuracy2)


def softmax(x):

    e_x = np.exp(x)
```

```
    return e_x / e_x.sum(axis=0, keepdims=True)


def rfunction(x, y, x2, y2, r):
    r_array = [i for i in range(10, r, 150)]

    for i in range(10, r, 150):
        print("R neuron= ", i)
        error(x, y, x2, y2, i)
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
        plt.plot(iteration_array, testcost_array, label="TestAccuracy")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

        plt.xlabel('Iteration')
        plt.ylabel('Loss')
        plt.plot(iteration_array, trainloss_array, label="TrainLoss")
        plt.plot(iteration_array, testloss_array, label="TestLoss")
        plt.legend(loc="upper left")
        plt.title("R neurons " + str(i))
        plt.show()

    # plt.plot(r_array, train_errorrate_array, r_array,test_errorrate_array)
    plt.xlabel('r neuron')
    plt.ylabel('accuracy')
    plt.plot(r_array, train_errorrate_array, label="Train")
    plt.plot(r_array, test_errorrate_array, label="Test")
    plt.legend(loc="upper left")
    plt.show()


def cross_entropy(X,y):

    m = y.shape[1]
    p = softmax(X)

    # loss = -np.mean(y * np.log(p + 1e-8))
    product = -(y * np.log(p + 1e-8))
    loss = np.sum(product) / m

    return loss

train_data_list = pd.read_csv('mnist_train.csv')
test_data_list = pd.read_csv('mnist_test.csv')
train_data = np.array(train_data_list)
test_data = np.array(test_data_list)



# give appropriate number to the dimensions
C = train_data
B = test_data
```

```
train_labels = train_data
test_labels = test_data
C = np.delete(C, 0, 1)
B = np.delete(B, 0, 1)
optimized_train_data=C/255
optimized_test_data=B/255
train_labels = np.delete(train_labels, np.s_[1:], 1)
test_labels = np.delete(test_labels, np.s_[1:], 1)
lr = np.arange(10)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# # we don't want zeroes and ones in the labels neither:
# train_labels_one_hot[train_labels_one_hot==0] = 0.01
# train_labels_one_hot[train_labels_one_hot==1] = 0.99
# test_labels_one_hot[test_labels_one_hot==0] = 0.01
# test_labels_one_hot[test_labels_one_hot==1] = 0.99
x1, y1 = optimized_train_data.T , train_labels_one_hot.T
x2, y2 = optimized_test_data.T , test_labels_one_hot.T
N=300
traincost_array = np.zeros(N)
testcost_array = np.zeros(N)
trainloss_array = np.zeros(N)
testloss_array = np.zeros(N)
train_errorrate_array = array.array('d', [])
test_errorrate_array = array.array('d', [])
iteration_array = [i for i in range(N)]




# error(x1,y1,x2,y2,r)
#
# # plt.plot(iteration_array,traincost_array,iteration_array,testcost_array )
# plt.xlabel('Iteration')
# plt.ylabel('Accuracy')
# plt.plot(iteration_array, traincost_array, label="TrainAccuracy")
# plt.plot(iteration_array, testcost_array, label="TestAccuracy")
# plt.legend(loc="upper left")
# plt.show()
#
# plt.xlabel('Iteration')
# plt.ylabel('Loss')
# plt.plot(iteration_array, trainloss_array, label="TrainLoss")
# plt.plot(iteration_array, testloss_array, label="TestLoss")
# plt.legend(loc="upper left")
# plt.show()
rfunction(x1, y1, x2, y2, r)
```