1. **Description of the selected forecasting problem**

Iris flower can be classified into three related species - Setosa, Versicolor and Virginica. The specific type of the species can be judged according to the width and length of its Petal and Sepal. Hence, the goal of this assignment is to judge the type of the iris flower based on the input width and length values.

1. **Overview of the network architecture(s) you have used including important parameters that affect the network operation. Make sure you include parameters you will tune for your problem.**

For this assignment, I chose Feed-Forward Neural Network(FFNN) with at most 10 layers as my deep learning models. The reason for doing so is because my set is some data with 4 features being classified into 3 categories. For the features, they are mutually independent. The values of the features and the result categories determined by the features have a non-linear relationship. Hence, a fully connected feed-forward neural network with multiple layers can be an appropriate model to describe their relationships.

The Convolutional Neural Network(CNN), on the contrary, is not suitable for my data set. Convolutional Neural Network is designed based on the assumptions that the data is locality(spacial relevant), hence, for data like image pixels which can fit this strong assumption well, it can get a stronger results. Otherwise, using such network doesn’t make sense and even leads to a worse results.

The Recurrent/Recursive Neural Network(RNN) is based on an assumption that the data is sequential relevant. Therefore, RNN is suitable for scenario like time series or sentimental prediction. Again, the data that this assignment uses does not cater such assumption.

1. **Description of the process you have used including data pre-processing, feature generation, model training, and evaluation.**

There are 6 columns in the raw data - id, SepalLength, SepalWidth, PetalLength, PetalWidth and categories. The sepal and petal size are float numbers ranged from 1 to 10. Because the values of these four features are closed, they don’t need to be scaled before learning process.

Samples in the same category are put together in the raw data. Hence, in order to separate the data, I shuffled the data by using shuffle() function after I loaded the data in Python in order to make sure that they can be separate equally without bias later.

After shuffling the data, The values of SepalLength, SepalWidth, PetalLength, PetalWidth is extracted as input vectors and the categories as output vectors for each rows. In order to use cross validation as the evaluation method, all the samples are separated into 5 parts. Each part will be used as test set once while the rest of four will be used as train set so there will be 5 iterations for one training process. The final evaluation result will be the average of the five iterations.

The parameters to be tuned for the neural network are numbers of layers, the neurons for each layers and the activation model. There are four activations:

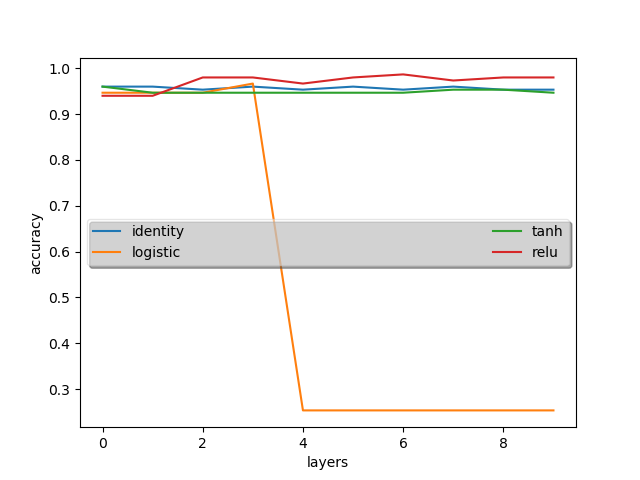
identity( f(x)=x), logistic(f(x) = 1 / (1 + exp(-x))), tanh(f(x) =tanh(x))

and relu(f(x) = max(0,x)). The numbers of layers will be set from 1 to 10 and the numbers of neurons will be set to be 100 for the first layer and minus 10 for the next layer. For example, the neurons of a network with 5 layers will be (100,90,80,70,60).

There are four activation models, for each models there will be layers from 1 to 10 with 5 iterations for each. Therefore, the total iterations will be 4 \* 10 \* 5 = 200.

1. **Results. This should include accuracy measures (more than one) achieved through the training process, graphs demonstrating final accuracy as well as the accuracy through the tuning process.**

**The result of the network is as follow:**



**the activation is identity**

the layers is (100,), the average accuracy is 0.9600000000000002

the layers is (100, 90), the average accuracy is 0.9600000000000002

the layers is (100, 90, 80), the average accuracy is 0.9533333333333334

the layers is (100, 90, 80, 70), the average accuracy is 0.9600000000000002

the layers is (100, 90, 80, 70, 60), the average accuracy is 0.9533333333333334

the layers is (100, 90, 80, 70, 60, 50), the average accuracy is 0.9600000000000002

the layers is (100, 90, 80, 70, 60, 50, 40), the average accuracy is 0.9533333333333334

the layers is (100, 90, 80, 70, 60, 50, 40, 30), the average accuracy is 0.9600000000000002

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20), the average accuracy is 0.9533333333333334

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20, 10), the average accuracy is 0.9533333333333334

**the activation is logistic**

the layers is (100,), the average accuracy is 0.9466666666666667

the layers is (100, 90), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70), the average accuracy is 0.9666666666666666

the layers is (100, 90, 80, 70, 60), the average accuracy is 0.2533333333333333

the layers is (100, 90, 80, 70, 60, 50), the average accuracy is 0.2533333333333333

the layers is (100, 90, 80, 70, 60, 50, 40), the average accuracy is 0.2533333333333333

the layers is (100, 90, 80, 70, 60, 50, 40, 30), the average accuracy is 0.2533333333333333

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20), the average accuracy is 0.2533333333333333

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20, 10), the average accuracy is 0.2533333333333333

**the activation is tanh**

the layers is (100,), the average accuracy is 0.96

the layers is (100, 90), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70, 60), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70, 60, 50), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70, 60, 50, 40), the average accuracy is 0.9466666666666667

the layers is (100, 90, 80, 70, 60, 50, 40, 30), the average accuracy is 0.9533333333333335

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20), the average accuracy is 0.9533333333333334

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20, 10), the average accuracy is 0.9466666666666667

**the activation is relu**

the layers is (100,), the average accuracy is 0.9400000000000001

the layers is (100, 90), the average accuracy is 0.9400000000000001

the layers is (100, 90, 80), the average accuracy is 0.9800000000000001

the layers is (100, 90, 80, 70), the average accuracy is 0.9800000000000001

the layers is (100, 90, 80, 70, 60), the average accuracy is 0.9666666666666666

the layers is (100, 90, 80, 70, 60, 50), the average accuracy is 0.9800000000000001

the layers is (100, 90, 80, 70, 60, 50, 40), the average accuracy is 0.9866666666666667

the layers is (100, 90, 80, 70, 60, 50, 40, 30), the average accuracy is 0.9733333333333334

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20), the average accuracy is 0.9800000000000001

the layers is (100, 90, 80, 70, 60, 50, 40, 30, 20, 10), the average accuracy is 0.9800000000000001

As we can see, the neural network with relu activation has a higher performance in average. So we conclude that relu is best activation for this dataset. Within the relu activation, the network with 7 layers(100,90,80,70,60,50,40) can get the highest accuracy. In conclusion, the final parameters will be 7 layers(100,90,80,70,60,50,40) with relu activation. The final accuracy is 0.9866666666666667.

Comparing to other algorithms：

Accuracy for KNearest is 0.9615384615384616

Accuracy for Support Vector Machine is: 0.9615384615384616

Accuracy for Gaussian Naive Bayes is 0.9230769230769231

Neural Network performs slightly better than other algorithms for this dataset.

1. **Code**

**import csv**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import random**

**import matplotlib**

**from sklearn.neural\_network import MLPClassifier**

**class NNClassification(object):**

**"""docstring for NNClassification"""**

**def \_\_init\_\_(self):**

**self.setX = []**

**self.setY = []**

**def loadData(self, filename):**

**"""**

**load and shuffle data from dataset**

**"""**

**with open(filename) as f:**

**lines = csv.reader(f)**

**X = [i for i in lines]**

**random.shuffle(X) # shuffle the dataset**

**for i in X:**

**l = []**

**for j in range(1, 5):**

**l.append(float(i[j]))**

**self.setX.append(l)**

**self.setY.append(i[-1])**

**# if random.random() < 0.7:**

**# TrainingX.append(l)**

**# TrainingY.append(i[-1])**

**# else:**

**# TestX.append(l)**

**# TestY.append(i[-1])**

**def dataTrain(self, i, layers, activation):**

**"""**

**train data with cross validation**

**"""**

**TrainingX = []**

**TrainingY = []**

**TestX = []**

**TestY = []**

**size = int(len(self.setX) / 5)**

**TestX = self.setX[i \* size: i \* size + size]**

**TestY = self.setY[i \* size: i \* size + size]**

**TrainingX = self.setX[:]**

**del TrainingX[i \* size:**

**i \* size + size]**

**TrainingY = self.setY[:]**

**del TrainingY[i \* size:**

**i \* size + size]**

**clf = MLPClassifier(solver='lbfgs', alpha=1e-5,**

**hidden\_layer\_sizes=layers, random\_state=1,**

**activation=activation)**

**# initial classification model**

**clf.fit(TrainingX, TrainingY)**

**y\_ = clf.predict(TestX) # predict the result of the test set**

**return self.accuracy(TestY, y\_)**

**def accuracy(self, TestY, realY):**

**"""**

**calculate the accuracy of test set**

**"""**

**count = 0**

**for i in range(len(TestY)):**

**if TestY[i] == realY[i]:**

**count += 1**

**return count / len(TestY)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**obj = NNClassification()**

**obj.loadData('iris.csv')**

**activation = ('identity', 'logistic', 'tanh',**

**'relu') # for kinds of activations**

**for act in activation:**

**layers = ()**

**neutrons = 100**

**avgAccuracy = []**

**print("the activation is {0}".format(act))**

**for i in range(10):**

**accuracy = []**

**layers = layers + (neutrons,)**

**neutrons -= 10 # neutrons minus 10 for each layers**

**for i in range(5):**

**accuracy.append(obj.dataTrain(i, layers, act))**

**print("the layers is {0}, the average accuracy is {1}".format(**

**layers, sum(accuracy) / 5))**

**avgAccuracy.append(sum(accuracy)/5)**

**plt.subplot()**

**plt.plot([i for i in range(10)], avgAccuracy, label = act)**

**plt.xlabel('layers')**

**plt.ylabel('accuracy')**

**leg = plt.legend(loc='best', ncol=2, mode="expand", shadow=True, fancybox=True)**

**leg.get\_frame().set\_alpha(0.5)**

**plt.show()**