Tutorial 6 - Neural Network

Part B: classification on real world dataset

Oct, 2023

Yuan Zhong

Official website: https://pytorch.org/get-started/locally/

Importing The Required Libraries

In [5]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

import torch
import torch.nn as nn
import matplotlib.pyplot as plt
%matplotlib inline
```

In [6]:
```
df1 = pd.read_csv('./data/Iris.csv')
```

In [7]:
```
df1.head()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Creating A Function To Get The Details of The Dataset

In [8]:
```
def get_info_dataframe(dataframe):
    print(f"DATAFRAME GENERAL INFO - \n")
    print(dataframe.info(),"\n")
    print(f"DATAFRAME MISSING INFO - \n")
    print(dataframe.isnull().sum(),"\n")
    print(f"DATAFRAME SHAPE INFO - \n")
    print(dataframe.shape)

get_info_dataframe(df1)
```

```
DATAFRAME GENERAL INFO -

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None

DATAFRAME MISSING INFO -

Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64

DATAFRAME SHAPE INFO -

(150, 6)
```

In [9]:
```
df1['Species'].unique()
```

Out[9]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

LabelEncoding The Attributes of The Target Column

In [10]:
```
df1['Species'] = df1['Species'].map({'Iris-setosa':0,'Iris-versicolor':1,'Ir
```

In [11]:
```
df1.head()
```

Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

In [12]:
```
df1.drop(['Id'],axis=1,inplace=True)
```

In [13]:
```
df1.head()
```

Out[13]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [14]:
```
X = df1.drop(["Species"],axis=1).values
y = df1["Species"].values
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [16]:
```
scaler = StandardScaler()
```

Doing The Train Test Split And Scaling The Data

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, ra
```

In [33]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train.shape)
```

```
(105, 4)
```

Converting From Numpy Array To Torch Tensor

In [19]:
```
X_train = torch.FloatTensor(X_train)
```

```
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

Creating Our Neural Network Model For Classification

In [20]:
```
class NeuralNetworkClassificationModel(nn.Module):
    def __init__(self,input_dim,output_dim):
        super(NeuralNetworkClassificationModel,self).__init__()
        self.input_layer    = nn.Linear(input_dim,128)
        self.hidden_layer1  = nn.Linear(128,64)
        self.output_layer   = nn.Linear(64,output_dim)
        self.relu = nn.ReLU()


    def forward(self,x):
        out =  self.relu(self.input_layer(x))
        out =  self.relu(self.hidden_layer1(out))
        out =  self.output_layer(out)
        return out
```

In [21]:
```
# input_dim = 4 because we have 4 inputs namely sepal_length,sepal_width,pet
# output_dim = 3 because we have namely 3 categories setosa,versicolor and v
input_dim  = 4
output_dim = 3
model = NeuralNetworkClassificationModel(input_dim,output_dim)
```

In [22]:
```
# creating our optimizer and loss function object
learning_rate = 0.01
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
```

In [24]:
```
def train_network(model,optimizer,criterion,X_train,y_train,X_test,y_test,nu
    for epoch in range(num_epochs):
        #clear out the gradients from the last step loss.backward()
        optimizer.zero_grad()

        #forward feed
        output_train = model(X_train)

        #calculate the loss
        loss_train = criterion(output_train, y_train)

        #backward propagation: calculate gradients
        loss_train.backward()

        #update the weights
        optimizer.step()


        output_test = model(X_test)
```

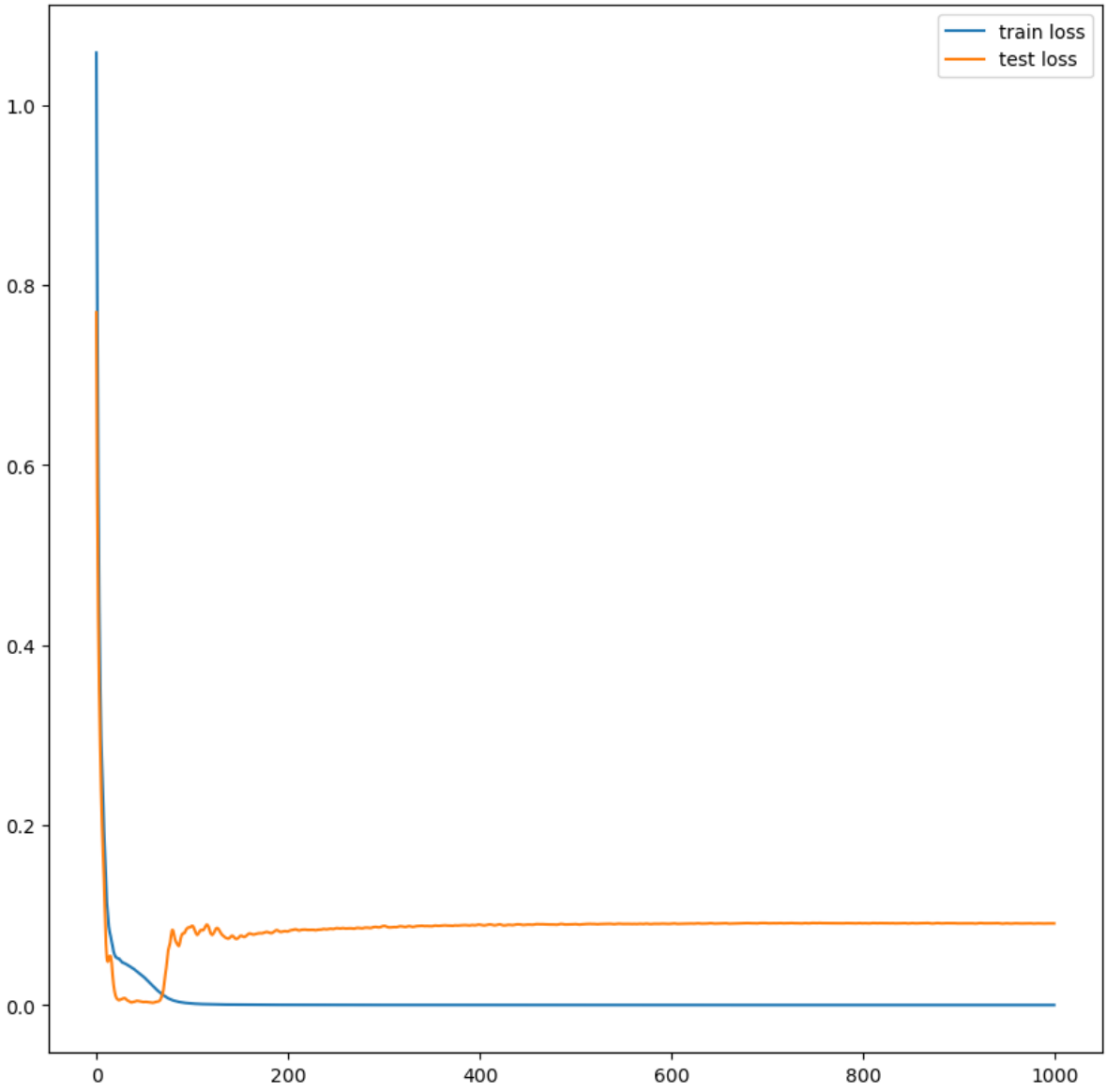
```
        loss_test = criterion(output_test,y_test)

        train_losses[epoch] = loss_train.item()
        test_losses[epoch] = loss_test.item()

        if (epoch + 1) % 50 == 0:
            print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {loss_train.it
```

In [25]:
```
num_epochs = 1000
train_losses = np.zeros(num_epochs)
test_losses  = np.zeros(num_epochs)
```

In [26]:
```
train_network(model,optimizer,criterion,X_train,y_train,X_test,y_test,num_ep
```

```
Epoch 50/1000, Train Loss: 0.0318, Test Loss: 0.0034
Epoch 100/1000, Train Loss: 0.0018, Test Loss: 0.0875
Epoch 150/1000, Train Loss: 0.0004, Test Loss: 0.0757
Epoch 200/1000, Train Loss: 0.0002, Test Loss: 0.0818
Epoch 250/1000, Train Loss: 0.0001, Test Loss: 0.0851
Epoch 300/1000, Train Loss: 0.0001, Test Loss: 0.0878
Epoch 350/1000, Train Loss: 0.0001, Test Loss: 0.0877
Epoch 400/1000, Train Loss: 0.0001, Test Loss: 0.0894
Epoch 450/1000, Train Loss: 0.0000, Test Loss: 0.0894
Epoch 500/1000, Train Loss: 0.0000, Test Loss: 0.0899
Epoch 550/1000, Train Loss: 0.0000, Test Loss: 0.0902
Epoch 600/1000, Train Loss: 0.0000, Test Loss: 0.0905
Epoch 650/1000, Train Loss: 0.0000, Test Loss: 0.0908
Epoch 700/1000, Train Loss: 0.0000, Test Loss: 0.0911
Epoch 750/1000, Train Loss: 0.0000, Test Loss: 0.0911
Epoch 800/1000, Train Loss: 0.0000, Test Loss: 0.0909
Epoch 850/1000, Train Loss: 0.0000, Test Loss: 0.0910
Epoch 900/1000, Train Loss: 0.0000, Test Loss: 0.0909
Epoch 950/1000, Train Loss: 0.0000, Test Loss: 0.0909
Epoch 1000/1000, Train Loss: 0.0000, Test Loss: 0.0908
```

In [27]:
```
plt.figure(figsize=(10,10))
plt.plot(train_losses, label='train loss')
plt.plot(test_losses, label='test loss')
plt.legend()
plt.show()
```



In [28]:
```
predictions_train = []
predictions_test =  []
with torch.no_grad():
    predictions_train = model(X_train)
    predictions_test = model(X_test)
```

In [29]:
```
# Check how the predicted outputs look like and after taking argmax compare
#predictions_train
#y_train,y_test
```

In [30]:
```
def get_accuracy_multiclass(pred_arr,original_arr):
    if len(pred_arr)!=len(original_arr):
        return False
    pred_arr = pred_arr.numpy()
    original_arr = original_arr.numpy()
    final_pred= []
    # we will get something like this in the pred_arr [32.1680,12.9350,-58.4
    # so will be taking the index of that argument which has the highest val
    for i in range(len(pred_arr)):
        final_pred.append(np.argmax(pred_arr[i]))
```

```
    final_pred = np.array(final_pred)
    count = 0
    #here we are doing a simple comparison between the predicted_arr and the
    for i in range(len(original_arr)):
        if final_pred[i] == original_arr[i]:
            count+=1
    return count/len(final_pred)
```

In [31]:
```
train_acc = get_accuracy_multiclass(predictions_train,y_train)
test_acc  = get_accuracy_multiclass(predictions_test,y_test)
```

In [32]:
```
print(f"Training Accuracy: {round(train_acc*100,3)}")
print(f"Test Accuracy: {round(test_acc*100,3)}")
```

```
Training Accuracy: 100.0
Test Accuracy: 97.778
```