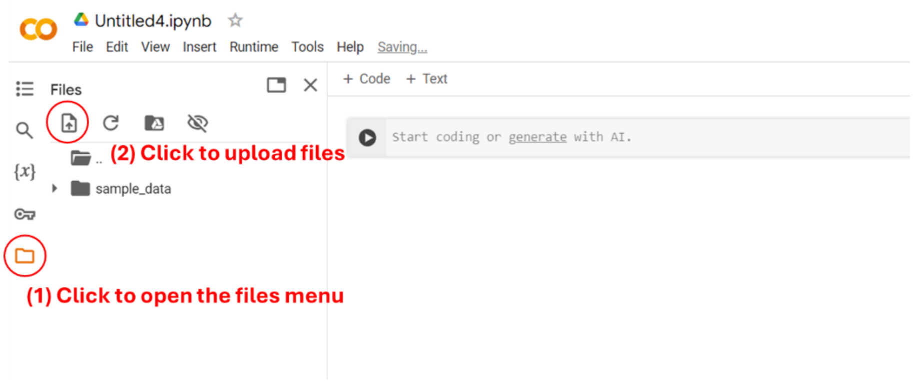
EE6310

Problem Set 12

In this problem set we will use the pytorch to implement the neural network and train it. We will work with the logistic regression data set from the homework 8:

Metallic y	**H%** x_1	**Li%** x_2	**Be%** x_3	**…**	**U%** x_{79}
1	0.41	0	0	…	0
0	0	0	0	…	0
0	0	0.04	0	…	0
…	…	…	…	…	

The full data can be downloaded from the weekly learning menu named "HW12.csv." To upload the csv file to Google Colab, Follow the following steps:


Untitled4.ipynb
File Edit View Insert Runtime Tools Help Saving...
Files
+ Code + Text
Start coding or generate with AI.
(2) Click to upload files
sample_data
(1) Click to open the files menu

The model we will implement in this work is:

$$g^{[1]}(Z^{[1]}) = \text{ReLU}(Z^{[1]})$$
$$g^{[2]}(Z^{[2]}) = \text{ReLU}(Z^{[2]})$$
$$g^{[3]}(Z^{[3]}) = \sigma(Z^{[3]})$$

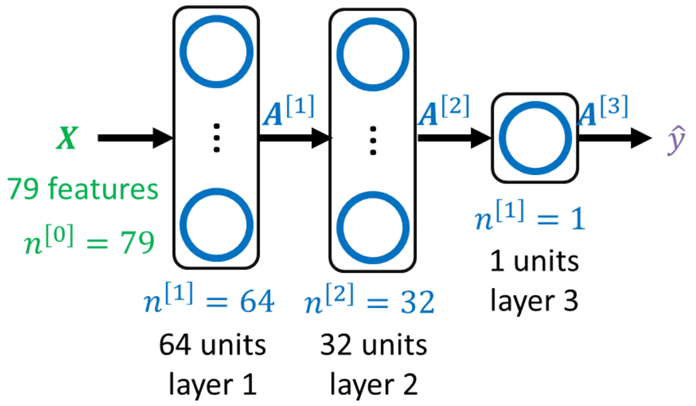


Figure 1. The neural network model to implement in this practice.

(a) Let's load the data set and convert it to torch tensor by filling the code between "### START CODE HERE ###" and "### END CODE HERE ###" Converting numpy array to torch tensor is not discussed in the lecture. Utilize Google to do it!

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
df = pd.read_csv('HW12.csv')
data = df.to_numpy()
# Convert the numpy array to torch tensor
### START CODE HERE ### (≈ 1 line of code)

### END CODE HERE ###
data = data.type(torch.float32)
Y = data[:,0]
X = data[:,1:]
```

(b) We will make a three-layer neural network as shown below. The first hidden layer will have 64 units, and the second hidden layer will have 32 hidden units. Calculate the number of units for each layer by filling the code between "### START CODE HERE ###" and "### END CODE HERE ###"

```
def layer_sizes(X, Y):
    """
    Arguments:
    X -- input dataset of shape (number of examples, input size)
    Y -- labels of shape (number of examples, output size)

    Returns:
    n_0 -- the size of the input layer
    n_1 -- the size of the first hidden layer
    n_2 -- the size of the second hidden layer
    n_3 -- the size of the output layer
    """
    # define the number of units for each layer
    ### START CODE HERE ### (≈ 3 lines of code)




    ### END CODE HERE ###
    return n_0, n_1, n_2, n_3

n_0,n_1,n_2,n_3 = layer_sizes(X, Y)
```

(c) Now we will make the pytorch module to make the neural network. Have a look at the model in the Figure 1. The activation function for the first two layers is ReLU, and the final layer is sigmoid. Google how to implement the ReLU activation function with pytorch. We will use the logit cross entropy loss. Fill the blank appropriately between "### START CODE HERE ###" and "### END CODE HERE ###"

```
class Model(nn.Module):
    def __init__(self,n_0,n_1,n_2,n_3):
        super().__init__()
        # Make 3 linear layers.
        # Also save the activation function as the class attribute
        ### START CODE HERE ### (≈ 4 line of code)






        ### END CODE HERE ###

    def forward(self, X):
        # perform forward. With two tanh and sigmoid
        ### START CODE HERE ### (≈ 5 line of code)






        ### END CODE HERE ###
        return Z3
```

If you have implemented appropriately, the model will be trained with the following code.

```
NN = Model(n_0,n_1,n_2,n_3)
import torch.optim as optim
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.SGD(NN.parameters(), lr=0.1)
for i in range(50000):
    Z = NN(X)
    optimizer.zero_grad()
    loss = criterion(Z,Y)
    loss.backward()
    optimizer.step()
    if i % 2000 == 0:
        print(f'{i} loss: {loss:.3f}')
```

```
Z = NN(X)
A = torch.sigmoid(Z)
y_hat = 1*(A>0.5)
accuracy = torch.mean((Y==y_hat).type(torch.float))
print(f'The model accuracy is {accuracy:.3f} %')
```