Pima Diabetes Prediction using ANN with PyTorch Library

Creating an Artificaial Neural Networks (ANN) using PyTorch

In [1]:

```
import pandas as pd
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [2]:

```
df.isnull().sum()
```

Out[2]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [3]:

```
import seaborn as sns
```

In [4]:

```
#import numpy as np
#df['Outcome'] = np.where(df['Outcome']==1, "Diabetic", "No Diabetic ")
```

In [5]:

```
df.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [6]:

In [6]:

```
sns.pairplot(df, hue = "Outcome")
```

Out[6]:

```
&lt;seaborn.axisgrid.PairGrid at 0x20aaece9b20&gt;
```

In [7]:

```
X = df.drop('Outcome', axis = 1).values # Independent features
y = df['Outcome'].values # Dependent features
```

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
= 0)
```

Create Tensors

In [9]:

```
# Libraries from PyTorch
import torch
import torch.nn as nn # Helps to create Models
```

```
import torch.nn.functional as F
```

In [10]:

```
# Creating Tensors
X_train = torch.FloatTensor(X_train) # Independent features has to be converted into Floa
ting point with Float Tensors
X_test = torch.FloatTensor(X_test)    # Independent features has to be converted into Floa
ting point with Float Tensors
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

Creating Model with PyTorch

In [11]:

```
df.shape
```

Out[11]:

```
(768, 9)
```

In [12]:

```
class ANN_Model(nn.Module):
    def __init__(self, input_features = 8, hidden1 = 20, hidden2 = 20, out_features = 2)
:
        super().__init__()
        self.f_connected1 = nn.Linear(input_features, hidden1)
        self.f_connected2 = nn.Linear(hidden1, hidden2)
        self.out = nn.Linear(hidden2, out_features)
    def forward(self, x):
        x = F.relu(self.f_connected1(x))
        x = F.relu(self.f_connected2(x))
        x = self.out(x)
        return x
```

In [13]:

```
##   Instantiate ANN_Model

torch.manual_seed(20)
model = ANN_Model()
```

In [14]:

```
model.parameters
```

Out[14]:

```
<bound method Module.parameters of ANN_Model(
  (f_connected1): Linear(in_features=8, out_features=20, bias=True)
  (f_connected2): Linear(in_features=20, out_features=20, bias=True)
  (out): Linear(in_features=20, out_features=2, bias=True)
)>
```

In [15]:

```
# Backward Propagation - Define the Loss Function & Optimizer
loss_function = nn.CrossEntropyLoss()  #for Multi Class Classification
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)  # lr - learning rate
```

In [16]:

```
epochs = 500
final_losses = []
for i in range(epochs):
    i = i + 1
    y_pred = model.forward(X_train)
    loss = loss_function(y_pred, y_train)
```

```
    final_losses.append(loss)
    if i%10 == 1:
        print("Epoch number : {} and the loss : {}".format(i, loss.item()))
    optimizer.zero_grad()  #Optimizer - To reduce the Loss & zero_grad - clears all grad
ients of all optimized class
    loss.backward()
    optimizer.step() #Performs Single optimization Step
```

```
Epoch number : 1 and the loss : 3.4572105407714844
Epoch number : 11 and the loss : 0.8019207119941711
Epoch number : 21 and the loss : 0.6090322136878967
Epoch number : 31 and the loss : 0.5917770862579346
Epoch number : 41 and the loss : 0.5679707527160645
Epoch number : 51 and the loss : 0.5529041886329651
Epoch number : 61 and the loss : 0.5410094857215881
Epoch number : 71 and the loss : 0.5310389995574951
Epoch number : 81 and the loss : 0.5220361351966858
Epoch number : 91 and the loss : 0.5135972499847412
Epoch number : 101 and the loss : 0.5061253905296326
Epoch number : 111 and the loss : 0.498340904712677
Epoch number : 121 and the loss : 0.4960551857948303
Epoch number : 131 and the loss : 0.48286372423171997
Epoch number : 141 and the loss : 0.4755900502204895
Epoch number : 151 and the loss : 0.48198607563972473
Epoch number : 161 and the loss : 0.48064836859703064
Epoch number : 171 and the loss : 0.4706920385360718
Epoch number : 181 and the loss : 0.45908692479133606
Epoch number : 191 and the loss : 0.4507930874824524
Epoch number : 201 and the loss : 0.444163978099823
Epoch number : 211 and the loss : 0.44218215346336365
Epoch number : 221 and the loss : 0.443209707736969
Epoch number : 231 and the loss : 0.43194958567619324
Epoch number : 241 and the loss : 0.43521177768707275
Epoch number : 251 and the loss : 0.41933000087738037
Epoch number : 261 and the loss : 0.4176027774810791
Epoch number : 271 and the loss : 0.43629634380340576
Epoch number : 281 and the loss : 0.4306843876838684
Epoch number : 291 and the loss : 0.41760239005088806
Epoch number : 301 and the loss : 0.4062289893627167
Epoch number : 311 and the loss : 0.40273022651672363
Epoch number : 321 and the loss : 0.3961154520511627
Epoch number : 331 and the loss : 0.4033721387386322
Epoch number : 341 and the loss : 0.40221765637397766
Epoch number : 351 and the loss : 0.3958454132080078
Epoch number : 361 and the loss : 0.38896337151527405
Epoch number : 371 and the loss : 0.3846299648284912
Epoch number : 381 and the loss : 0.3896012604236603
Epoch number : 391 and the loss : 0.3793337047100067
Epoch number : 401 and the loss : 0.3751954436302185
Epoch number : 411 and the loss : 0.3896487355232239
Epoch number : 421 and the loss : 0.39123857021331787
Epoch number : 431 and the loss : 0.37030020356178284
Epoch number : 441 and the loss : 0.36845123767852783
Epoch number : 451 and the loss : 0.36489173769950867
Epoch number : 461 and the loss : 0.36325761675834656
Epoch number : 471 and the loss : 0.443462997674942
Epoch number : 481 and the loss : 0.4384981393814087
Epoch number : 491 and the loss : 0.37451717257499695
```

In [17]:

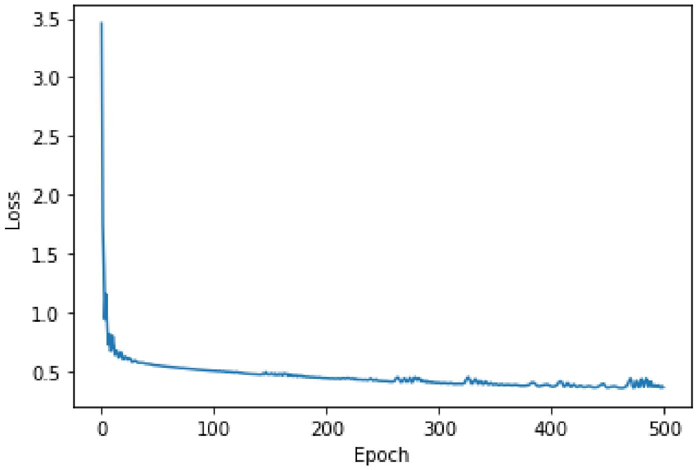
```
# Plot the Loss function
import matplotlib.pyplot as plt
%matplotlib inline
```

In [18]:

```
plt.plot(range(epochs), final_losses)
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

Out[18]:

```
Text(0.5, 0, 'Epoch')
```



In [19]:

```
# Prediction in X_test data
predictions = []
with torch.no_grad(): #To hold the Gradients in Output
    for i, data in enumerate(X_test):
        y_pred = model(data)
        predictions.append(y_pred.argmax().item())
        print(y_pred.argmax().item()) # argmax - to know which Index, item - Index '0' o
r '1'
```

```
1
0
0
1
0
0
1
1
0
0
1
1
0
1
0
0
1
0
0
0
1
0
0
0
0
1
0
1
0
0
0
0
0
1
0
1
1
0
0
1
0
0
```

0
1
1
0
0
0
1
0
0
0
1
0
0
1
0
1
0
0
1
0
0
1
1
1
1
0
0
0
0
0
0
1
1
0
0
1
0
0
0
1
0
0
0
0
0
0
1
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
1
1
0
0
0
1
0
1
1
0

```
0
1
0
1
0
1
0
1
0
0
0
0
0
0
0
0
1
0
0
0
0
1
0
0
1
0
0
1
0
0
1
0
0
0
0
1
0
0
0
0
```

In [20]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions)
cm
```

Out[20]:

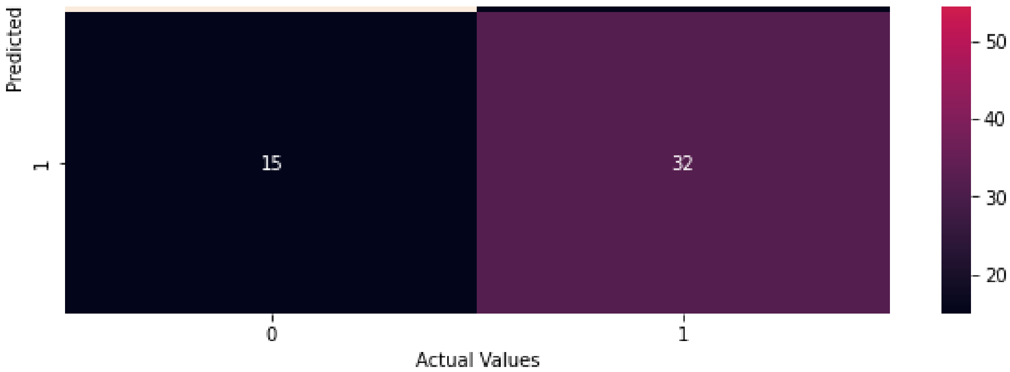
```
array([[92, 15],
       [15, 32]], dtype=int64)
```

In [21]:

```
plt.figure(figsize = (10, 6))
sns.heatmap(cm, annot = True)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

Out[21]:

```
Text(69.0, 0.5, 'Predicted Values')
```



In [22]:

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, predictions)
score
```

Out[22]:

```
0.8051948051948052
```

In [23]:

```
# Save the Models

torch.save(model, 'diabetes.pt') # PyTorch models need to be saved in '.pt'
```

In [24]:

```
# Save & Load the model
model = torch.load('diabetes.pt')
```

In [25]:

```
model.eval()
```

Out[25]:

```
ANN_Model(
  (f_connected1): Linear(in_features=8, out_features=20, bias=True)
  (f_connected2): Linear(in_features=20, out_features=20, bias=True)
  (out): Linear(in_features=20, out_features=2, bias=True)
)
```

In [26]:

```
# Prediction of new data point
list(df.iloc[0,:-1])
```

Out[26]:

```
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0]
```

In [27]:

```
# New Data
lst1 = [6.0, 138.0, 72.0, 40.0, 0.0, 25.6, 0.627, 45.0]
```

In [28]:

```
new_data = torch.tensor(lst1)
```

In [29]:

```
# Predict New Data using PyTorch

with torch.no_grad(): #To hold the Gradients in Output
    print(model(new_data))
    print(model(new_data).argmax().item())
```

```
tensor([1.6578, 1.4680])
0
```