

```
In [ ]: import torch
        torch.__version__
```

```
Out[ ]: '2.0.0+cu118'
```

Createing An ANN using Potorch

dataset link- <https://www.kaggle.com/datasets/saurabh00007/diabetescsv>
(<https://www.kaggle.com/datasets/saurabh00007/diabetescsv>)

```
In [1]: from google.colab import drive
        drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [4]: import pandas as pd
        df=pd.read_csv('/content/drive/MyDrive/cnn batch size 32/diabetes.csv')
        df.head()
```

```
Out[4]:
```

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

```
In [5]: df.isnull().sum()
```


```
Out[5]: Pregnancies      0
        Glucose          0
        BloodPressure    0
        SkinThickness    0
        Insulin          0
        BMI              0
        DiabetesPedigreeFunction  0
        Age              0
        Outcome          0
        dtype: int64
```

```
In [10]: import seaborn as sns
```

```
In [11]: import numpy as np
df['Outcome']=np.where(df['Outcome']==1,"Diabetic","No Diabetic")
df.head()
```

Out[11]:

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



```
In [12]: sns.pairplot(df,hue="Outcome")
```

```
Out[12]: <seaborn.axisgrid.PairGrid at 0x7fe3bd56ce50>
```



```
In [14]: df=pd.read_csv('/content/drive/MyDrive/cnn batch size 32/diabetes.csv')
df.head()
```

```
Out[14]:
```

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

```
In [15]: X=df.drop('Outcome',axis=1).values### independent features
y=df['Outcome'].values###dependent features
```

```
In [16]: 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)
```

```
In [17]: ##### Libraries From Pytorch
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
In [18]: df.shape
```

```
Out[18]: (768, 9)
```

```
In [20]: ##### Creating Tensors
X_train=torch.FloatTensor(X_train)
X_test=torch.FloatTensor(X_test)
y_train=torch.LongTensor(y_train)
y_test=torch.LongTensor(y_test)
```

```
In [21]: ##### Creating Modelwith Pytorch

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 [23]: #####instantiate my ANN_model
torch.manual_seed(20)
model=ANN_Model()
```

```
In [24]: model.parameters
```

```
Out[24]: <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 [25]: ###Backward Propogation-- Define the loss_function,define the optimizer
loss_function=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.01)
```

```
In [26]: 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()
             loss.backward()
             optimizer.step()
```

Epoch number: 1 and the loss : 3.4572105407714844
Epoch number: 11 and the loss : 0.8019208312034607
Epoch number: 21 and the loss : 0.6090322136878967
Epoch number: 31 and the loss : 0.5917770266532898
Epoch number: 41 and the loss : 0.5679708123207092
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 : 0.5135971903800964
Epoch number: 101 and the loss : 0.5061254501342773
Epoch number: 111 and the loss : 0.49834102392196655
Epoch number: 121 and the loss : 0.4960551857948303
Epoch number: 131 and the loss : 0.48286372423171997
Epoch number: 141 and the loss : 0.4756035804748535
Epoch number: 151 and the loss : 0.48334649205207825
Epoch number: 161 and the loss : 0.4882740080356598
Epoch number: 171 and the loss : 0.4693370461463928
Epoch number: 181 and the loss : 0.46626102924346924
Epoch number: 191 and the loss : 0.45597586035728455
Epoch number: 201 and the loss : 0.44663771986961365
Epoch number: 211 and the loss : 0.43963825702667236
Epoch number: 221 and the loss : 0.4364195168018341
Epoch number: 231 and the loss : 0.4408119022846222
Epoch number: 241 and the loss : 0.4275389313697815
Epoch number: 251 and the loss : 0.42173999547958374
Epoch number: 261 and the loss : 0.4312404692173004
Epoch number: 271 and the loss : 0.4249121844768524
Epoch number: 281 and the loss : 0.41044458746910095
Epoch number: 291 and the loss : 0.43008580803871155
Epoch number: 301 and the loss : 0.40997204184532166
Epoch number: 311 and the loss : 0.39859500527381897
Epoch number: 321 and the loss : 0.4065549969673157
Epoch number: 331 and the loss : 0.399066299200058
Epoch number: 341 and the loss : 0.40190550684928894
Epoch number: 351 and the loss : 0.39513570070266724
Epoch number: 361 and the loss : 0.38684454560279846
Epoch number: 371 and the loss : 0.3830900490283966
Epoch number: 381 and the loss : 0.3788066506385803
Epoch number: 391 and the loss : 0.3929040729999542
Epoch number: 401 and the loss : 0.3939976692199707
Epoch number: 411 and the loss : 0.389658123254776
Epoch number: 421 and the loss : 0.3854386806488037
Epoch number: 431 and the loss : 0.3908206522464752
Epoch number: 441 and the loss : 0.37238621711730957
Epoch number: 451 and the loss : 0.3696228563785553
Epoch number: 461 and the loss : 0.3620612919330597
Epoch number: 471 and the loss : 0.36269620060920715
Epoch number: 481 and the loss : 0.4455106258392334
Epoch number: 491 and the loss : 0.37344980239868164

```
In [27]: ### plot the loss function  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
In [29]: ##### Prediction In X_test data
         predictions=[]
         with torch.no_grad():
             for i,data in enumerate(X_test):
                 y_pred=model(data)
                 predictions.append(y_pred.argmax().item())
             print(y_pred.argmax().item())
```

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

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

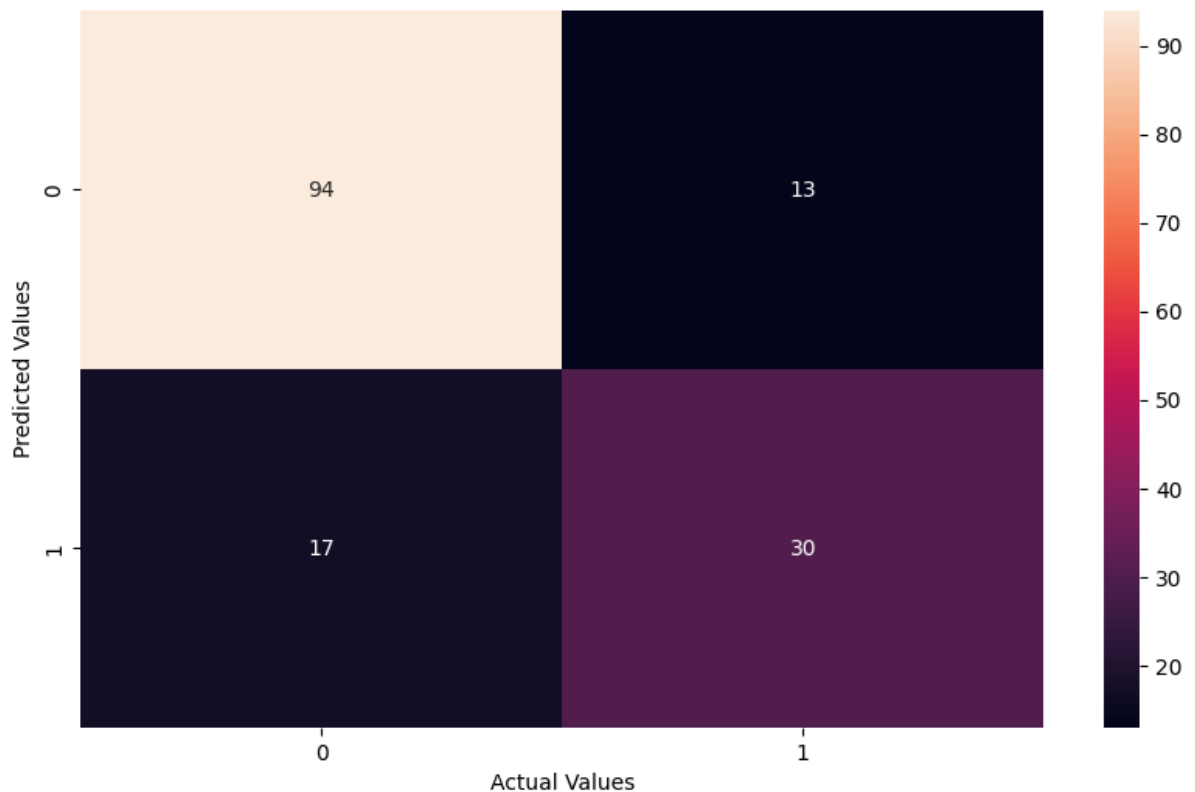
0
1
0
1
0
1
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
1
1
0
1
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0

```
In [30]: from sklearn.metrics import confusion_matrix  
cm=confusion_matrix(y_test,predictions)  
cm
```

```
Out[30]: array([[94, 13],  
               [17, 30]])
```

```
In [31]: plt.figure(figsize=(10,6))
sns.heatmap(cm,annot=True)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

```
Out[31]: Text(95.7222222222221, 0.5, 'Predicted Values')
```



```
In [32]: from sklearn.metrics import accuracy_score
score=accuracy_score(y_test,predictions)
score
```

```
Out[32]: 0.8051948051948052
```

```
In [33]: ##### Save the model
torch.save(model,'diabetes.pt')
```

```
In [34]: ##### Save And Load the model
model=torch.load('diabetes.pt')
```

```
In [35]: model.eval()
```

```
Out[35]: 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 [36]: ### Predcition of new data point  
list(df.iloc[0,:-1])
```

```
Out[36]: [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0]
```

```
In [37]: #### New Data  
lst1=[6.0, 130.0, 72.0, 40.0, 0.0, 25.6, 0.627, 45.0]
```

```
In [38]: new_data=torch.tensor(lst1)
```

```
In [39]: #### Predict new data using Pytorch  
with torch.no_grad():  
    print(model(new_data))  
    print(model(new_data).argmax().item())
```

```
tensor([1.5195, 0.9168])  
0
```

```
In [ ]:
```