```
from torch import nn
import torch.optim as optim
```

In [25]:
```
# Converting  the   training and testing data into tensors
X = df.drop(['MDVP:Fo(Hz)','MDVP:Fhi(Hz)','MDVP:Flo(Hz)','HNR','status','name'],axi
y = df['status'].values.reshape(-1,1)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 1
```

In [26]:
```
X_train = torch.from_numpy(X_train).type(torch.Tensor)
X_test = torch.from_numpy(X_test).type(torch.Tensor)
y_train = torch.from_numpy(y_train).type(torch.Tensor)
y_test = torch.from_numpy(y_test).type(torch.Tensor)
```

In [27]:
```
X_train.shape
```

Out[27]:
```
torch.Size([156, 18])
```

In [28]:
```
y_train.shape
```

Out[28]:
```
torch.Size([156, 1])
```

In [29]:
```
class Parkinsons(nn.Module):
    def __init__(self,input_dim=18,hidden_dim=90,output_dim=1):
        super().__init__()
        self.input_dim=input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.linear1 = nn.Linear(input_dim,hidden_dim)
        self.linear2 = nn.Linear(hidden_dim,hidden_dim)
        self.fc = nn.Linear(hidden_dim,output_dim)
    def forward(self,x:torch.Tensor):
        x= f.relu(x)
        x = self.linear1(x)
        x = self.linear2(x)
        x = self.fc(x)
        x = torch.sigmoid(x)
        x = torch.round(x)
        return x
```

In [32]:
```
class Parkinsons_2(nn.Module):
    def __init__(self,input_dim=18,hidden_dim=90,output_dim=1):
        super().__init__()
        self.input_dim=input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.linear1 = nn.Linear(input_dim,hidden_dim)
        self.linear2 = nn.Linear(hidden_dim,hidden_dim)
        self.fc = nn.Linear(hidden_dim,output_dim)
    def forward(self,x:torch.Tensor):
        x= f.relu(x)
        x = self.linear1(x)
        x = self.linear2(x)
        x = self.fc(x)
        #x = torch.sigmoid(x)
        #x = torch.round(x)
        return x
```

In [35]:
```
model = Parkinsons()
model_2 = Parkinsons_2()
```

In [36]:
```
with torch.inference_mode():
    predictions = model_2.forward(X_test)
    print(predictions[:10])
```

```
tensor([[-0.0595],
        [-0.0452],
        [-0.0544],
        [-0.0290],
        [-0.0553],
        [-0.0507],
        [-0.0768],
        [-0.0578],
        [-0.0236],
        [-0.1295]])
```

In [33]:
```
with torch.inference_mode():
    predictions = model.forward(X_test)
    print(predictions[:10])
```

```
tensor([[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.]])
```

In [ ]: