Pratham Nagar - 64 Experiment 04 ~ Design a fully connected deep neural network with at least 2 hidden layers on Titanic survival classification by choosing appropriate Learning Algorithms. Pratham Nagar - 64

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score

# Load dataset from local file or alternative source
# df = pd.read_csv('your_dataset.csv')

# For demonstration, using a simple synthetic dataset approach
from sklearn.datasets import make_classification

# Create synthetic binary classification dataset
X, y = make_classification(n_samples=1000, n_features=10, n_redundant=2,
                           n_informative=8, random_state=42)

# Create DataFrame
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])])
df['target'] = y

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Convert to PyTorch tensors
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32).view(-1, 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

class BinaryClassifierDNN(nn.Module):
    def __init__(self, input_size=10):
        super(BinaryClassifierDNN, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)  # Added layer of 32 neurons
        self.fc3 = nn.Linear(32, 16)  # Added layer of 16 neurons
        self.fc4 = nn.Linear(16, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.dropout(self.relu(self.fc2(x)))
        x = self.relu(self.fc3(x))
        x = self.sigmoid(self.fc4(x))
        return x

model = BinaryClassifierDNN(input_size=X.shape[1])
criterion = nn.BCELoss()  # Binary Cross Entropy
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 100
for epoch in range(epochs):
    model.train()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

model.eval()
with torch.no_grad():
    predictions = model(X_test)
    predicted_classes = (predictions >= 0.5).float()
```

```
    acc = accuracy_score(y_test, predicted_classes)
    print(f"Test Accuracy: {acc * 100:.2f}%")
```

```
Epoch [10/100], Loss: 0.6850
Epoch [20/100], Loss: 0.6677
Epoch [30/100], Loss: 0.6377
Epoch [40/100], Loss: 0.5966
Epoch [50/100], Loss: 0.5491
Epoch [60/100], Loss: 0.4879
Epoch [70/100], Loss: 0.4438
Epoch [80/100], Loss: 0.3991
Epoch [90/100], Loss: 0.3563
Epoch [100/100], Loss: 0.3190
Test Accuracy: 84.50%
```