Two Layer Perceptron (Feed - Forward Neural Net)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the dataset

In [2]:
```
file_path = "Cirrhosis.csv"
df = pd.read_csv(file_path)
```

Data Analysis

In [3]:
```
print("Initial Dataset Info:")
print(df.info())
```

```
Initial Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 312 entries, 0 to 311
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   index          312 non-null    int64
 1   duration       312 non-null    int64
 2   status         312 non-null    int64
 3   drug           312 non-null    int64
 4   age            312 non-null    int64
 5   sex            312 non-null    int64
 6   ascites        312 non-null    int64
 7   hepatomology   312 non-null    int64
 8   spiders        312 non-null    int64
 9   edema          312 non-null    float64
 10  bilirubin      312 non-null    float64
 11  cholesterol    284 non-null    float64
 12  albumin        312 non-null    float64
 13  copper         310 non-null    float64
 14  phosphatase    312 non-null    float64
 15  SGOT           312 non-null    float64
 16  triglicerides  282 non-null    float64
 17  platelets      308 non-null    float64
 18  prothrombin    312 non-null    float64
 19  stage          312 non-null    int64
dtypes: float64(10), int64(10)
memory usage: 48.9 KB
None
```

In [4]:
```
df.head()
```

Out[4]:

	index	duration	status	drug	age	sex	ascites	hepatomology	spiders	edema
0	1	400	2	1	21464	1	1	1	1	1.0
1	2	4500	0	1	20617	1	0	1	1	0.0
2	3	1012	2	1	25594	0	0	0	0	0.5
3	4	1925	2	1	19994	1	0	1	1	0.5
4	5	1504	1	2	13918	1	0	1	1	0.0

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

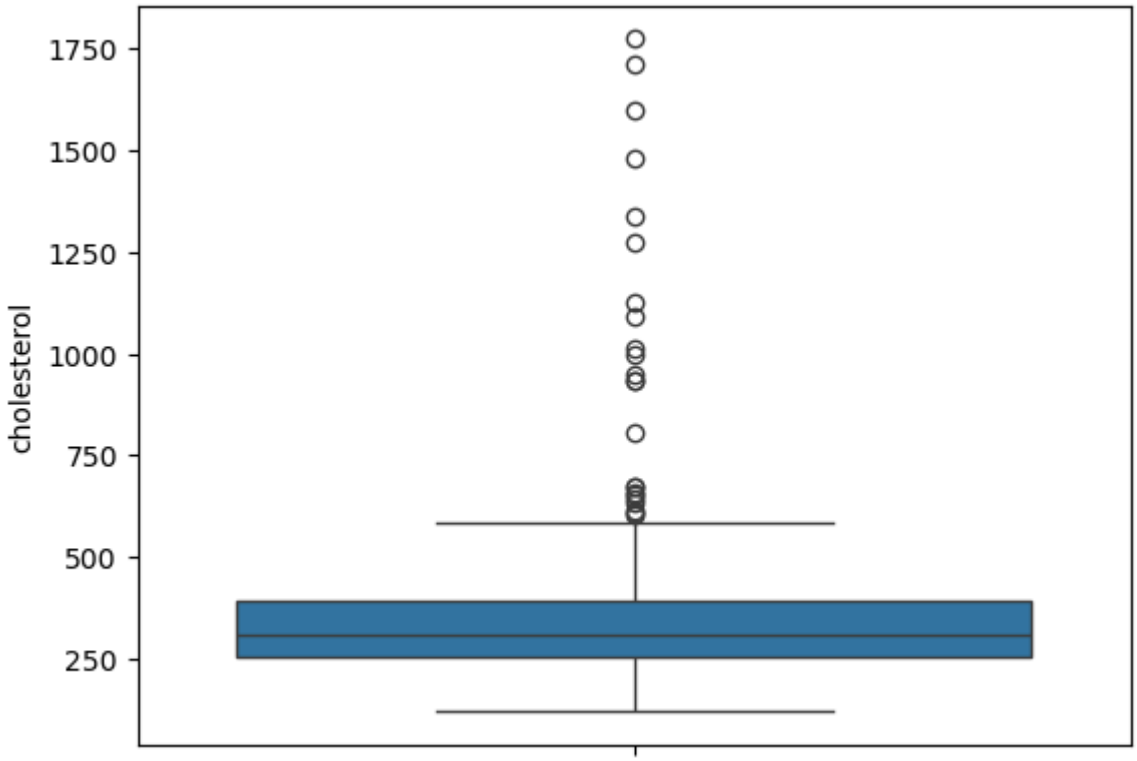
Out[5]:
```
index             0
duration          0
status            0
drug              0
age               0
sex               0
ascites           0
hepatomology      0
spiders           0
edema             0
bilirubin         0
cholesterol      28
albumin           0
copper            2
phosphatase       0
SGOT              0
triglicerides    30
platelets         4
prothrombin       0
stage             0
dtype: int64
```

In [6]:
```
df.fillna({"cholesterol":df["cholesterol"].median()}, inplace=True)
df.fillna({"triglicerides":df["triglicerides"].median()}, inplace=True)
df.fillna({"copper":df["copper"].median()}, inplace=True)
df.fillna({"platelets":df["platelets"].median()}, inplace=True)
```

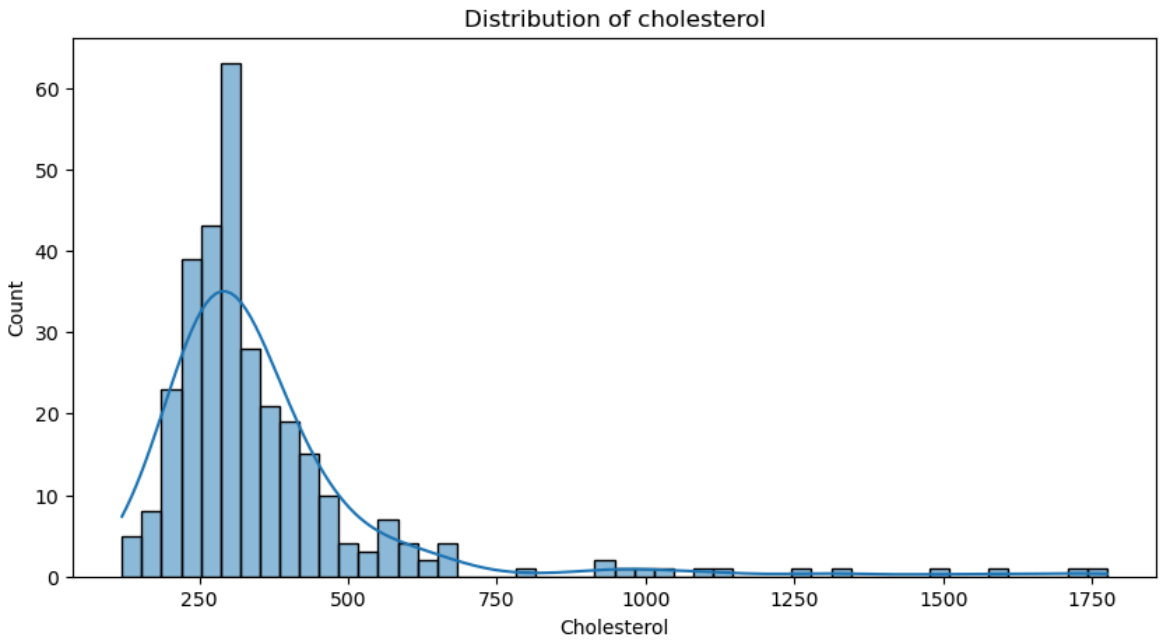
Data Visualization

In [7]:
```
sns.boxplot(y=df['cholesterol'])
```

Out[7]:
```
<Axes: ylabel='cholesterol'>
```



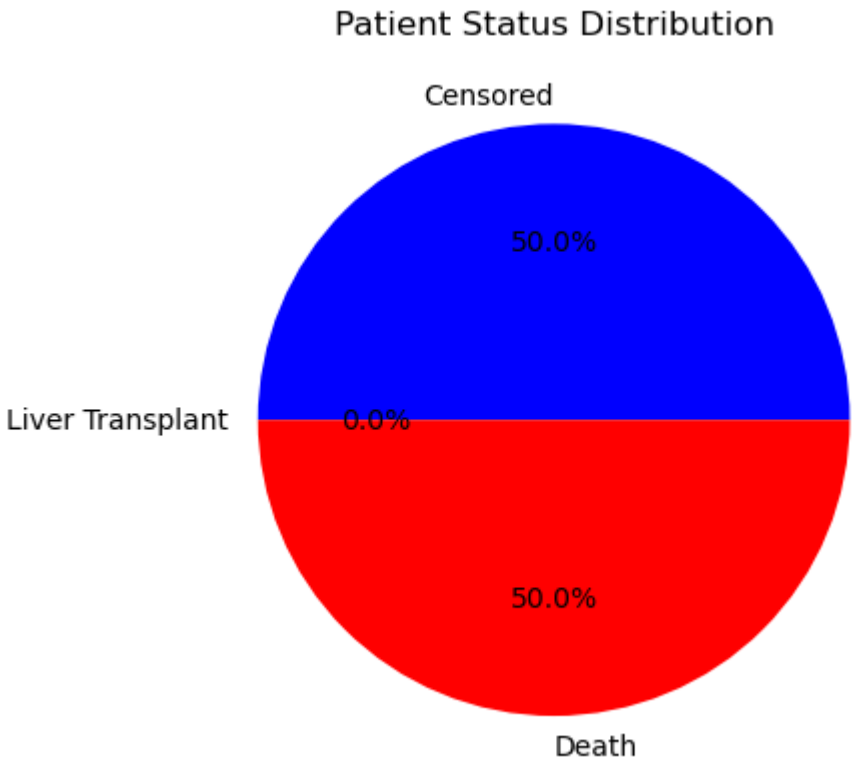
In [8]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df["cholesterol"], bins=50, kde=True) #Kernel Density Estimation
plt.title("Distribution of cholesterol")
plt.xlabel("Cholesterol")
plt.ylabel("Count")
plt.show()
```



In [9]:
```
# Features and target
X = df[["cholesterol", "bilirubin", "ascites"]].values # "sex",
y = df["status"].values
```

In [10]:
```
status_counts = df["status"]
labels = ['Censored', 'Liver Transplant', 'Death']
values = [status_counts[0], status_counts[1], status_counts[2]]
```

```
plt.pie(values, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange', 'red
plt.title('Patient Status Distribution')
plt.show()
```



Activation Functions

In [11]:
```
def linear(H):
  return H

def Sigmoid(H):
  return 1/(1+np.exp(-H))

def ReLU(H):
  return H*(H>0)  # condition if H >0 than H

def softmax(H):
  eH = np.exp(H)
  return eH/eH.sum(axis=1, keepdims=True)
```

Useful Functions

In [12]:
```
def one_hot_encode(y):
  N = len(y)
  K = len(set(y))
  Y = np.zeros((N,K))
  for i in range(N):
    Y[i,y[i]] = 1
  return Y

def accuracy(y, y_hat):
  return np.mean(y==y_hat)
```

```
def cross_entropy(Y, P_hat):
  return -np.sum(Y*np.log(P_hat))
```

Shallow ANN Class

In [13]:
```
class Shallow_ANN():
  def fit(self, X, y, neurons =6, eta = 1e3-3 ,epochs =1e3, show_curve = True):

    epochs = int(epochs)
    N,D = X.shape
    Y = one_hot_encode(y)
    K = Y.shape[1]

    # Initialize Weights and Biases
    self.W = {l: np.random.randn(M[0],M[1]) for l, M in enumerate(zip([D,neurons
    self.B = {l: np.random.randn(M) for l, M in enumerate([neurons, K],1)}

    # Define Activations
    self.a = {1:np.tanh, 2:softmax}

    J = np.zeros(epochs)

    #SGD Steps
    for epoch in range(epochs):
      self.__forward__(X)
      J[epoch] = cross_entropy(Y,self.Z[2]) # 2 second layer - output layer

      #Weight Update Rules for Layer 2
      self.W[2] -= eta*(1/N)*self.Z[1].T@(self.Z[2]-Y)
      self.B[2] -= eta*(1/N)*(self.Z[2]-Y).sum(axis=0)

      #Weight Update Rules for Layer 1
      self.W[1] -= eta*(1/N)*X.T@((self.Z[2]-Y)@self.W[2].T*(1-self.Z[1]**2))
      self.B[1] -= eta*(1/N)*((self.Z[2]-Y)@self.W[2].T*(1-self.Z[1]**2)).sum(ax

    if show_curve:
      plt.figure()
      plt.plot(J)
      plt.xlabel("epochs")
      plt.ylabel("J")
      plt.title("Training Curve")

  def __forward__(self,X):
    self.Z ={0:X}
    for l in sorted(self.W.keys()):
      self.Z[l] = self.a[l](self.Z[l-1]@self.W[l]+self.B[l])

  def predict(self, X):
    return self.Z[2].argmax(axis=1)
```

Implementation of ShallowANN Class

In [14]:
```
np.vstack(np.array(X))
X[0,-2]
```

Out[14]:
```
np.float64(14.5)
```

In [15]:
```
X0 = X[y == 0]  # Cluster for status 0
X1 = X[y == 1]  # Cluster for status 1
X2 = X[y == 2]  # Cluster for status 2
X = np.vstack((X0, X1, X2))
X
```

```
Out[15]: array([[3.020e+02, 1.100e+00, 0.000e+00],
                [3.220e+02, 1.000e+00, 0.000e+00],
                [2.810e+02, 7.000e-01, 0.000e+00],
                [2.040e+02, 7.000e-01, 0.000e+00],
                [2.350e+02, 7.000e-01, 0.000e+00],
                [2.520e+02, 6.000e-01, 0.000e+00],
                [2.980e+02, 7.000e-01, 0.000e+00],
                [3.700e+02, 7.000e-01, 0.000e+00],
                [2.620e+02, 1.800e+00, 0.000e+00],
                [3.640e+02, 8.000e-01, 0.000e+00],
                [1.720e+02, 3.000e-01, 0.000e+00],
                [3.095e+02, 1.300e+00, 0.000e+00],
                [3.095e+02, 2.100e+00, 0.000e+00],
                [3.610e+02, 1.100e+00, 0.000e+00],
                [3.095e+02, 6.000e-01, 0.000e+00],
                [3.160e+02, 5.000e-01, 0.000e+00],
                [2.590e+02, 1.900e+00, 0.000e+00],
                [2.420e+02, 7.000e-01, 0.000e+00],
                [6.040e+02, 9.000e-01, 0.000e+00],
                [2.160e+02, 6.000e-01, 0.000e+00],
                [2.560e+02, 1.200e+00, 0.000e+00],
                [1.740e+02, 7.000e-01, 0.000e+00],
                [3.095e+02, 6.000e-01, 0.000e+00],
                [2.580e+02, 1.200e+00, 0.000e+00],
                [3.200e+02, 5.000e-01, 0.000e+00],
                [1.320e+02, 7.000e-01, 0.000e+00],
                [3.150e+02, 8.000e-01, 0.000e+00],
                [2.500e+02, 1.300e+00, 0.000e+00],
                [2.630e+02, 4.000e-01, 0.000e+00],
                [2.960e+02, 6.000e-01, 0.000e+00],
                [3.530e+02, 1.300e+00, 0.000e+00],
                [3.095e+02, 1.000e+00, 0.000e+00],
                [2.390e+02, 1.000e+00, 0.000e+00],
                [4.600e+02, 1.800e+00, 0.000e+00],
                [4.000e+02, 9.000e-01, 0.000e+00],
                [2.480e+02, 9.000e-01, 0.000e+00],
                [2.120e+02, 6.000e-01, 0.000e+00],
                [1.200e+02, 5.000e-01, 0.000e+00],
                [3.030e+02, 7.000e-01, 0.000e+00],
                [4.580e+02, 3.000e+00, 0.000e+00],
                [2.980e+02, 6.000e-01, 0.000e+00],
                [2.510e+02, 6.000e-01, 0.000e+00],
                [2.680e+02, 5.000e-01, 0.000e+00],
                [4.200e+02, 9.000e-01, 0.000e+00],
                [4.480e+02, 1.900e+00, 0.000e+00],
                [5.780e+02, 7.000e-01, 0.000e+00],
                [2.630e+02, 4.000e-01, 0.000e+00],
                [2.630e+02, 8.000e-01, 0.000e+00],
                [3.990e+02, 1.100e+00, 0.000e+00],
                [3.280e+02, 1.100e+00, 0.000e+00],
                [2.900e+02, 1.100e+00, 0.000e+00],
                [3.460e+02, 9.000e-01, 0.000e+00],
                [3.090e+02, 7.000e-01, 0.000e+00],
                [3.095e+02, 1.200e+00, 0.000e+00],
                [2.880e+02, 1.200e+00, 0.000e+00],
                [3.095e+02, 1.000e+00, 0.000e+00],
                [4.600e+02, 9.000e-01, 0.000e+00],
                [2.170e+02, 5.000e-01, 0.000e+00],
                [2.200e+02, 6.000e-01, 0.000e+00],
                [2.860e+02, 6.000e-01, 0.000e+00],
```

```
[2.170e+02, 6.000e-01, 0.000e+00],
[5.020e+02, 2.300e+00, 0.000e+00],
[1.480e+03, 5.700e+00, 0.000e+00],
[2.570e+02, 4.000e-01, 0.000e+00],
[3.900e+02, 1.200e+00, 0.000e+00],
[3.095e+02, 5.000e-01, 0.000e+00],
[2.050e+02, 1.300e+00, 0.000e+00],
[2.360e+02, 3.000e+00, 0.000e+00],
[3.095e+02, 5.000e-01, 0.000e+00],
[2.830e+02, 8.000e-01, 0.000e+00],
[2.580e+02, 9.000e-01, 0.000e+00],
[3.095e+02, 6.000e-01, 0.000e+00],
[3.960e+02, 1.800e+00, 0.000e+00],
[4.780e+02, 4.700e+00, 0.000e+00],
[2.480e+02, 1.400e+00, 0.000e+00],
[3.095e+02, 6.000e-01, 0.000e+00],
[2.560e+02, 8.000e-01, 0.000e+00],
[1.870e+02, 7.000e-01, 0.000e+00],
[3.600e+02, 1.300e+00, 0.000e+00],
[3.095e+02, 2.300e+00, 0.000e+00],
[3.080e+02, 9.000e-01, 0.000e+00],
[2.930e+02, 1.500e+00, 0.000e+00],
[3.470e+02, 3.700e+00, 0.000e+00],
[2.260e+02, 1.400e+00, 0.000e+00],
[2.660e+02, 6.000e-01, 0.000e+00],
[2.860e+02, 7.000e-01, 0.000e+00],
[3.920e+02, 2.100e+00, 0.000e+00],
[2.360e+02, 4.700e+00, 0.000e+00],
[2.350e+02, 6.000e-01, 0.000e+00],
[2.230e+02, 5.000e-01, 0.000e+00],
[1.490e+02, 5.000e-01, 0.000e+00],
[2.130e+02, 6.000e-01, 0.000e+00],
[3.095e+02, 6.000e-01, 0.000e+00],
[2.520e+02, 7.000e-01, 0.000e+00],
[3.460e+02, 9.000e-01, 0.000e+00],
[3.095e+02, 1.300e+00, 0.000e+00],
[2.320e+02, 1.200e+00, 0.000e+00],
[4.000e+02, 5.000e-01, 0.000e+00],
[3.095e+02, 5.000e-01, 0.000e+00],
[3.095e+02, 5.000e-01, 0.000e+00],
[2.150e+02, 1.600e+00, 0.000e+00],
[3.600e+02, 9.000e-01, 0.000e+00],
[3.090e+02, 1.000e+00, 0.000e+00],
[2.740e+02, 7.000e-01, 0.000e+00],
[2.230e+02, 5.000e-01, 0.000e+00],
[2.150e+02, 7.000e-01, 0.000e+00],
[3.020e+02, 3.300e+00, 0.000e+00],
[2.670e+02, 4.000e-01, 0.000e+00],
[5.140e+02, 9.000e-01, 0.000e+00],
[5.780e+02, 9.000e-01, 0.000e+00],
[1.336e+03, 1.300e+01, 0.000e+00],
[2.530e+02, 1.500e+00, 0.000e+00],
[4.420e+02, 1.600e+00, 0.000e+00],
[2.800e+02, 6.000e-01, 0.000e+00],
[2.320e+02, 4.000e-01, 0.000e+00],
[3.540e+02, 1.900e+00, 0.000e+00],
[2.730e+02, 6.000e-01, 0.000e+00],
[3.240e+02, 8.000e-01, 0.000e+00],
[2.420e+02, 1.300e+00, 0.000e+00],
[2.990e+02, 6.000e-01, 0.000e+00],
```

```
[2.270e+02, 5.000e-01, 0.000e+00],
[2.460e+02, 1.100e+00, 0.000e+00],
[2.430e+02, 7.100e+00, 1.000e+00],
[1.930e+02, 7.000e-01, 0.000e+00],
[3.360e+02, 1.100e+00, 0.000e+00],
[2.800e+02, 5.000e-01, 0.000e+00],
[4.140e+02, 1.100e+00, 0.000e+00],
[2.770e+02, 3.100e+00, 0.000e+00],
[2.320e+02, 5.600e+00, 0.000e+00],
[3.750e+02, 3.200e+00, 0.000e+00],
[3.220e+02, 2.800e+00, 0.000e+00],
[3.180e+02, 5.000e-01, 0.000e+00],
[3.740e+02, 3.600e+00, 0.000e+00],
[4.480e+02, 1.000e+00, 0.000e+00],
[3.210e+02, 1.000e+00, 0.000e+00],
[2.260e+02, 5.000e-01, 0.000e+00],
[3.280e+02, 2.200e+00, 0.000e+00],
[5.720e+02, 2.200e+00, 0.000e+00],
[2.190e+02, 1.000e+00, 0.000e+00],
[3.170e+02, 1.000e+00, 0.000e+00],
[3.380e+02, 5.600e+00, 0.000e+00],
[1.980e+02, 5.000e-01, 0.000e+00],
[3.250e+02, 1.600e+00, 0.000e+00],
[3.040e+02, 1.300e+00, 0.000e+00],
[4.120e+02, 1.100e+00, 0.000e+00],
[2.910e+02, 1.300e+00, 0.000e+00],
[2.530e+02, 8.000e-01, 0.000e+00],
[3.100e+02, 2.000e+00, 0.000e+00],
[3.730e+02, 6.400e+00, 0.000e+00],
[2.940e+02, 1.400e+00, 0.000e+00],
[5.460e+02, 8.600e+00, 0.000e+00],
[1.940e+02, 8.500e+00, 0.000e+00],
[1.000e+03, 6.600e+00, 0.000e+00],
[3.280e+02, 8.000e-01, 0.000e+00],
[3.400e+02, 1.100e+00, 0.000e+00],
[3.420e+02, 2.400e+00, 0.000e+00],
[3.930e+02, 1.000e+00, 0.000e+00],
[3.350e+02, 7.000e-01, 0.000e+00],
[3.720e+02, 1.000e+00, 0.000e+00],
[2.190e+02, 5.000e-01, 0.000e+00],
[4.260e+02, 2.900e+00, 0.000e+00],
[2.390e+02, 6.000e-01, 0.000e+00],
[2.730e+02, 8.000e-01, 0.000e+00],
[2.460e+02, 4.000e-01, 0.000e+00],
[2.600e+02, 4.000e-01, 0.000e+00],
[4.340e+02, 1.700e+00, 0.000e+00],
[2.470e+02, 2.000e+00, 0.000e+00],
[5.760e+02, 6.400e+00, 0.000e+00],
[2.790e+02, 3.400e+00, 0.000e+00],
[4.640e+02, 1.100e+00, 0.000e+00],
[5.280e+02, 5.500e+00, 0.000e+00],
[3.250e+02, 3.500e+00, 0.000e+00],
[3.160e+02, 1.300e+00, 0.000e+00],
[4.500e+02, 3.400e+00, 0.000e+00],
[2.010e+02, 5.000e-01, 0.000e+00],
[3.160e+02, 4.400e+00, 0.000e+00],
[3.870e+02, 2.100e+00, 0.000e+00],
[1.712e+03, 6.100e+00, 0.000e+00],
[2.270e+02, 3.100e+00, 0.000e+00],
[4.320e+02, 1.100e+00, 0.000e+00],
```

```
[3.560e+02, 3.400e+00, 0.000e+00],
[3.480e+02, 3.500e+00, 0.000e+00],
[3.095e+02, 1.600e+00, 0.000e+00],
[3.100e+02, 8.700e+00, 0.000e+00],
[3.390e+02, 3.200e+00, 0.000e+00],
[6.460e+02, 2.400e+00, 0.000e+00],
[2.750e+02, 1.200e+00, 0.000e+00],
[2.610e+02, 1.450e+01, 1.000e+00],
[1.760e+02, 1.400e+00, 0.000e+00],
[2.440e+02, 1.800e+00, 0.000e+00],
[2.480e+02, 8.000e-01, 0.000e+00],
[2.800e+02, 3.000e-01, 0.000e+00],
[5.620e+02, 3.200e+00, 0.000e+00],
[2.000e+02, 1.260e+01, 1.000e+00],
[2.590e+02, 1.400e+00, 0.000e+00],
[2.360e+02, 3.600e+00, 0.000e+00],
[3.095e+02, 8.000e-01, 1.000e+00],
[2.310e+02, 8.000e-01, 0.000e+00],
[2.740e+02, 2.700e+00, 0.000e+00],
[1.780e+02, 1.140e+01, 0.000e+00],
[3.740e+02, 5.100e+00, 0.000e+00],
[2.710e+02, 3.400e+00, 0.000e+00],
[3.950e+02, 1.740e+01, 1.000e+00],
[4.560e+02, 2.100e+00, 0.000e+00],
[1.128e+03, 5.200e+00, 0.000e+00],
[1.750e+02, 2.160e+01, 1.000e+00],
[2.220e+02, 1.720e+01, 1.000e+00],
[2.600e+02, 3.600e+00, 0.000e+00],
[2.960e+02, 4.700e+00, 0.000e+00],
[2.100e+02, 8.000e-01, 0.000e+00],
[3.140e+02, 1.200e+00, 0.000e+00],
[3.340e+02, 7.100e+00, 1.000e+00],
[3.830e+02, 3.300e+00, 0.000e+00],
[2.820e+02, 7.000e-01, 0.000e+00],
[3.095e+02, 6.800e+00, 0.000e+00],
[2.990e+02, 3.300e+00, 0.000e+00],
[4.820e+02, 5.700e+00, 0.000e+00],
[3.095e+02, 8.000e-01, 0.000e+00],
[2.570e+02, 1.100e+00, 0.000e+00],
[2.760e+02, 8.000e-01, 0.000e+00],
[6.140e+02, 6.000e+00, 0.000e+00],
[3.095e+02, 2.600e+00, 0.000e+00],
[2.880e+02, 1.300e+00, 1.000e+00],
[4.160e+02, 1.800e+00, 0.000e+00],
[4.980e+02, 1.100e+00, 0.000e+00],
[2.600e+02, 2.300e+00, 0.000e+00],
[3.290e+02, 8.000e-01, 0.000e+00],
[3.020e+02, 1.300e+00, 1.000e+00],
[9.320e+02, 2.250e+01, 0.000e+00],
[3.730e+02, 2.100e+00, 0.000e+00],
[4.270e+02, 1.400e+00, 0.000e+00],
[4.660e+02, 1.100e+00, 0.000e+00],
[6.520e+02, 2.000e+01, 0.000e+00],
[5.580e+02, 8.400e+00, 0.000e+00],
[6.740e+02, 1.710e+01, 1.000e+00],
[3.940e+02, 1.220e+01, 0.000e+00],
[2.440e+02, 6.600e+00, 0.000e+00],
[4.360e+02, 6.300e+00, 0.000e+00],
[2.470e+02, 7.200e+00, 0.000e+00],
[4.480e+02, 1.440e+01, 0.000e+00],
```

```
[4.720e+02, 4.500e+00, 0.000e+00],
[2.620e+02, 2.100e+00, 0.000e+00],
[1.600e+03, 5.000e+00, 0.000e+00],
[3.450e+02, 1.100e+00, 0.000e+00],
[4.080e+02, 2.000e+00, 0.000e+00],
[6.600e+02, 1.600e+00, 0.000e+00],
[3.250e+02, 5.000e+00, 0.000e+00],
[2.060e+02, 1.400e+00, 1.000e+00],
[2.010e+02, 3.200e+00, 0.000e+00],
[3.095e+02, 1.740e+01, 1.000e+00],
[4.200e+02, 2.000e+00, 0.000e+00],
[1.780e+02, 2.300e+00, 0.000e+00],
[1.880e+02, 2.500e+00, 1.000e+00],
[3.030e+02, 1.100e+00, 0.000e+00],
[3.095e+02, 2.100e+00, 1.000e+00],
[1.270e+02, 4.000e-01, 0.000e+00],
[4.860e+02, 1.900e+00, 0.000e+00],
[2.670e+02, 2.000e+00, 0.000e+00],
[3.740e+02, 6.700e+00, 0.000e+00],
[2.590e+02, 3.200e+00, 0.000e+00],
[9.500e+02, 6.500e+00, 0.000e+00],
[3.900e+02, 3.500e+00, 0.000e+00],
[6.360e+02, 6.000e-01, 0.000e+00],
[1.510e+02, 1.300e+00, 1.000e+00],
[3.095e+02, 5.100e+00, 0.000e+00],
[2.690e+02, 1.200e+00, 1.000e+00],
[3.095e+02, 1.620e+01, 0.000e+00],
[1.775e+03, 1.740e+01, 0.000e+00],
[2.420e+02, 2.800e+00, 0.000e+00],
[3.310e+02, 1.500e+00, 0.000e+00],
[4.260e+02, 7.300e+00, 0.000e+00],
[3.640e+02, 1.000e+00, 0.000e+00],
[3.320e+02, 2.900e+00, 0.000e+00],
[5.560e+02, 2.800e+01, 0.000e+00],
[1.015e+03, 7.200e+00, 0.000e+00],
[2.570e+02, 3.000e+00, 0.000e+00],
[5.860e+02, 2.300e+00, 0.000e+00],
[1.680e+02, 2.400e+00, 0.000e+00],
[3.580e+02, 2.550e+01, 0.000e+00],
[3.170e+02, 2.500e+00, 0.000e+00],
[2.600e+02, 3.200e+00, 0.000e+00],
[2.330e+02, 3.000e-01, 0.000e+00],
[3.095e+02, 8.500e+00, 0.000e+00],
[1.960e+02, 4.000e+00, 0.000e+00],
[3.760e+02, 9.000e-01, 0.000e+00],
[4.080e+02, 1.300e+00, 0.000e+00],
[3.095e+02, 3.200e+00, 0.000e+00],
[6.740e+02, 1.100e+01, 0.000e+00],
[2.250e+02, 2.000e+00, 0.000e+00],
[8.080e+02, 1.400e+01, 0.000e+00],
[1.092e+03, 2.450e+01, 1.000e+00],
[9.320e+02, 1.080e+01, 0.000e+00],
[2.550e+02, 7.000e-01, 0.000e+00],
[3.820e+02, 2.500e+00, 1.000e+00],
[3.960e+02, 3.900e+00, 0.000e+00],
[4.040e+02, 9.000e-01, 0.000e+00],
[1.276e+03, 5.900e+00, 0.000e+00],
[6.080e+02, 1.140e+01, 0.000e+00],
[4.260e+02, 3.800e+00, 0.000e+00],
[3.720e+02, 4.500e+00, 0.000e+00],
```

```
       [4.480e+02, 1.410e+01, 1.000e+00],
       [3.160e+02, 2.300e+00, 0.000e+00],
       [1.910e+02, 4.500e+00, 1.000e+00],
       [5.180e+02, 3.400e+00, 0.000e+00],
       [3.000e+02, 8.000e-01, 0.000e+00],
       [4.680e+02, 8.000e+00, 0.000e+00],
       [3.500e+02, 3.900e+00, 0.000e+00],
       [2.220e+02, 6.600e+00, 1.000e+00],
       [3.440e+02, 6.400e+00, 1.000e+00],
       [1.750e+02, 1.790e+01, 1.000e+00],
       [4.160e+02, 4.000e+00, 0.000e+00],
       [3.095e+02, 5.200e+00, 0.000e+00]])
```

In [16]:
```
y =  np.array([0] * len(X0) + [1] * len(X1) + [2] * len(X2))
y
```

Out[16]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2])
```

In [17]:
```
X[y == 0][:,0]
```

Out[17]:
```
array([ 302. ,  322. ,  281. ,  204. ,  235. ,  252. ,  298. ,  370. ,
        262. ,  364. ,  172. ,  309.5,  309.5,  361. ,  309.5,  316. ,
        259. ,  242. ,  604. ,  216. ,  256. ,  174. ,  309.5,  258. ,
        320. ,  132. ,  315. ,  250. ,  263. ,  296. ,  353. ,  309.5,
        239. ,  460. ,  400. ,  248. ,  212. ,  120. ,  303. ,  458. ,
        298. ,  251. ,  268. ,  420. ,  448. ,  578. ,  263. ,  263. ,
        399. ,  328. ,  290. ,  346. ,  309. ,  309.5,  288. ,  309.5,
        460. ,  217. ,  220. ,  286. ,  217. ,  502. , 1480. ,  257. ,
        390. ,  309.5,  205. ,  236. ,  309.5,  283. ,  258. ,  309.5,
        396. ,  478. ,  248. ,  309.5,  256. ,  187. ,  360. ,  309.5,
        308. ,  293. ,  347. ,  226. ,  266. ,  286. ,  392. ,  236. ,
        235. ,  223. ,  149. ,  213. ,  309.5,  252. ,  346. ,  309.5,
        232. ,  400. ,  309.5,  309.5,  215. ,  360. ,  309. ,  274. ,
        223. ,  215. ,  302. ,  267. ,  514. ,  578. , 1336. ,  253. ,
        442. ,  280. ,  232. ,  354. ,  273. ,  324. ,  242. ,  299. ,
        227. ,  246. ,  243. ,  193. ,  336. ,  280. ,  414. ,  277. ,
        232. ,  375. ,  322. ,  318. ,  374. ,  448. ,  321. ,  226. ,
        328. ,  572. ,  219. ,  317. ,  338. ,  198. ,  325. ,  304. ,
        412. ,  291. ,  253. ,  310. ,  373. ,  294. ,  546. ,  194. ,
       1000. ,  328. ,  340. ,  342. ,  393. ,  335. ,  372. ,  219. ,
        426. ,  239. ,  273. ,  246. ,  260. ,  434. ,  247. ,  576. ])
```

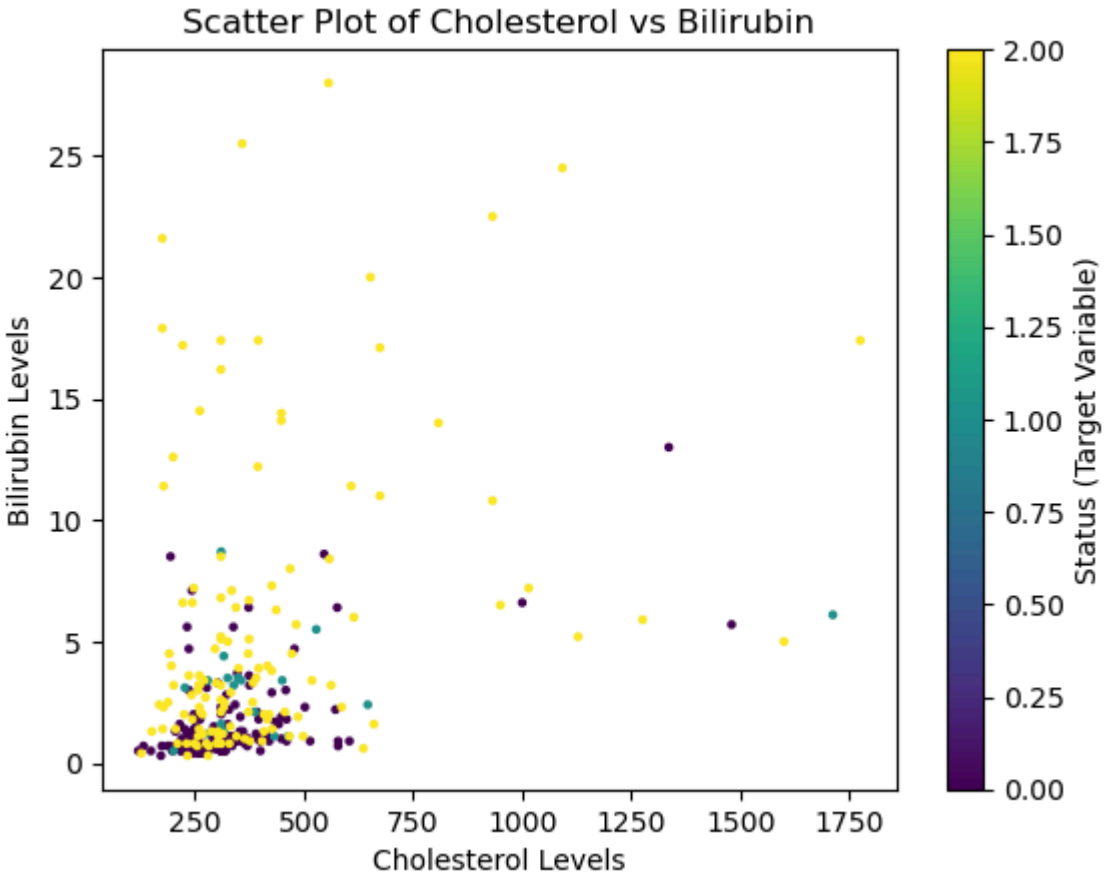
In [26]:
```
def main():
  D = 2
  K = 3
  N= int(K*1e3)
```

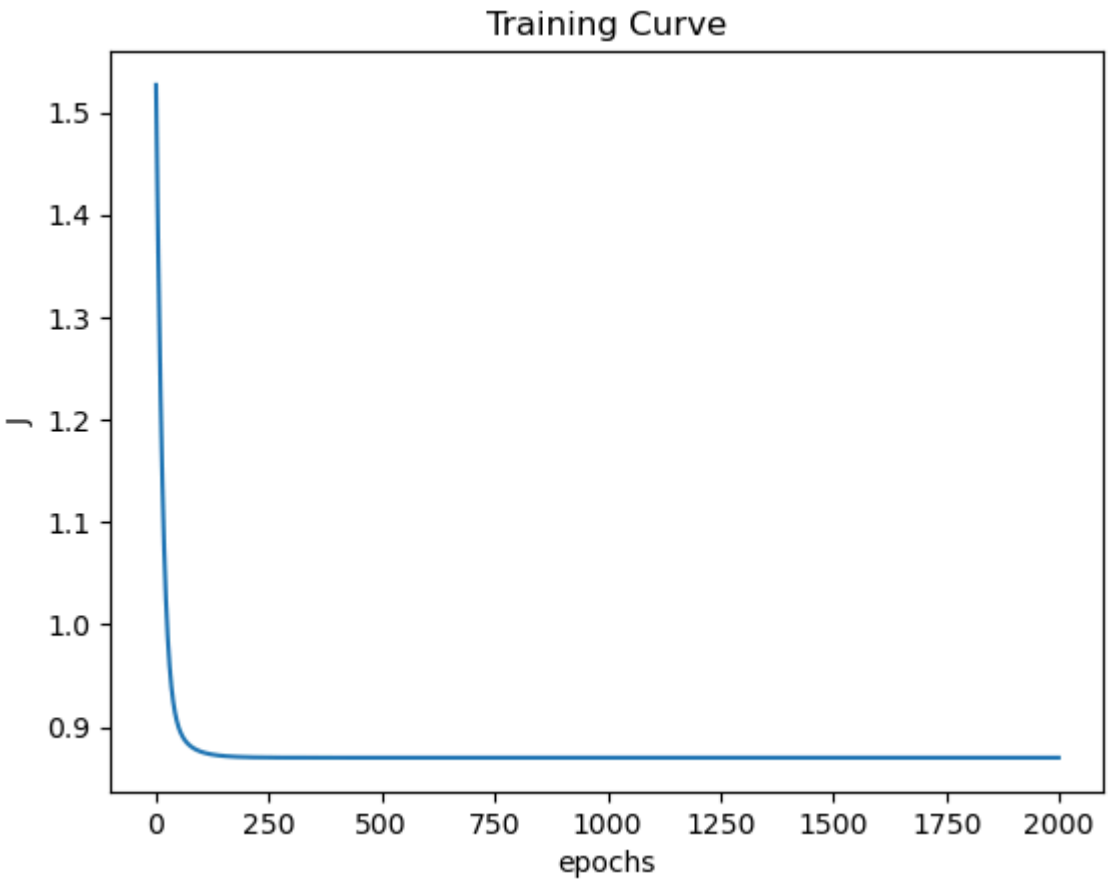
```
#X0= np.random.randn((N//K),D) + np.array([2] * D)
#X1= np.random.randn((N//K),D) + np.array([0] * D)
#X2= np.random.randn((N//K),D) + np.array([-2] * D)
#X=np.vstack((X0,X1,X2))

#y = np.array([0]*(N//K) + [1]*(N//K) + [2]*(N//K))#y=np.array(y)

plt.figure()
plt.scatter(X[:,0],X[:,1], c=y , s= 5)
plt.title("Scatter Plot of Cholesterol vs Bilirubin")
plt.xlabel("Cholesterol Levels")
plt.ylabel("Bilirubin Levels")
plt.colorbar(label="Status (Target Variable)")

my_ann = Shallow_ANN()
my_ann.fit(X, y, neurons= 3, eta=3e-2, epochs = 2e3, show_curve= True )
y_hat =my_ann.predict(X)


print("Accuracy: ", accuracy(y, y_hat))
print(my_ann.W)
print(my_ann.B)
```

In [27]:

```
if __name__=="__main__":
  main()
```

```
Accuracy:  0.5384615384615384
{1: array([[-0.10256169, -0.68193772, -0.64535878],
       [-1.21444228, -0.26039486,  0.972618  ],
       [-0.53777065,  1.28082855,  0.82354306]]), 2: array([[ 0.11578913,  0.3367
3323,  0.95085201],
       [-2.7582602 ,  0.98603883, -0.60703526],
       [ 2.31931383, -0.34591819, -0.12510496]])}
{1: array([-0.14521999, -0.09428958,  0.51327322]), 2: array([-0.85703214, -1.736
54602, -0.61081335])}
```





Artificial Neural Network Variable Architecture and Back Propogation

In [20]:
```
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(["#FF0000","#00FF00","#0000FF"])
cmap_light = ListedColormap(["#FFBBBB", "#BBFFBB","#BBBBFF"])
```

Useful Functions

In [21]:
```
# Activations

def linear(H):
  return H

def ReLU(H):
  return H*(H>0)

def sigmoid(H):
  return 1/(1+np.exp(-H))

def softmax(H):
  eH=np.exp(H)
  return eH/eH.sum(axis=1, keepdims=True)

#Loss Functions

def cross_entropy(Y, P_hat):
  return -(1/len(Y))*np.sum(Y*np.log(P_hat))

def OLS(Y, Y_hat):
  return (1/(2*len(Y)))*np.sum((Y-Y_hat)**2)

#Misc

def one_hot(y):
  N=len(y)
  K=len(set(y))
  Y = np.zeros((N,K))

  for i in range(N):
    Y[i,y[i]]=1

  return Y

def accuracy(y,y_hat):
  return np.mean(y==y_hat)

def R2(y,y_hat):
  return 1-np.sum((y-y_hat)**2)/np.sum((y - y.mean())**2)
```

Derivatives of Activation Functions

In [22]:
```
def derivative(Z, a):
  if a==linear:
    return 1
  elif a==sigmoid:
    return Z*(1-Z)
  elif a==np.tanh:
    return 1-Z*Z
```

```
  elif a==ReLU:
    return (Z>0).astype(int)
  else:
    ValueError("UnknownActivation")
```

ANN Class

In [28]:
```
class ANN():
  def __init__(self, architecture, activations=None, mode=0):
    self.mode=mode
    self.architecture=architecture
    self.activations = activations
    self.L = len(architecture)+1

  def fit (self, X, y, eta=1e-3, epochs=1e3, show_curve=True):
    epochs = int(epochs)

    if self.mode:
      Y=y
      K=1
    else:
      Y=one_hot(y)
      K=Y.shape[1]

    N, D = X.shape
    # Initialize Weights and Biases
    self.W = {l: np.random.randn(M[0],M[1]) for l, M in enumerate(zip(([D]+self.
    self.B = {l: np.random.randn(M) for l, M in enumerate(self.architecture+[K],


    #Activations
    if self.activations is None:
      self.a= {l: ReLU for l in range(1, self.L)}
    else:
      self.a={l: act for l,act in enumerate(self.activations, 1)}

    #Output Activation Functions
    if self.mode:
      self.a[self.L]=linear
    else:
      self.a[self.L]=softmax

    J = np.zeros(epochs)

    #SGD Progression
    for epoch in range(epochs):
      self.__forward__(X)
      if self.mode:
        J[epoch]=OLS(Y, self.Z[self.L])
      else:
        J[epoch]=cross_entropy(Y, self.Z[self.L])

      dH = (1/N)*(self.Z[self.L]-Y)

      for l in sorted(self.W.keys(), reverse=True):

        dW = self.Z[l-1].T@dH
        dB = dH.sum(axis=0)
```

```
        self.W[l] -= eta*dW
        self.B[l] -= eta*dB

        if l>1:
          dZ =dH@self.W[l].T
          dH = dZ*derivative(self.Z[l-1], self.a[l-1])

    if show_curve:
      plt.figure()
      plt.plot(J)
      plt.xlabel("epochs")
      plt.ylabel("J")
      plt.title("Training Curve")

  def __forward__(self,X):
    self.Z={0:X}
    for l in sorted(self.W.keys()):
      self.Z[l] = self.a[l](self.Z[l-1]@self.W[l]+self.B[l])


  def predict(self, X):
    self.__forward__(X)
    if self.mode:
      return self.Z[self.L]
    else:
      return self.Z[self.L].argmax(axis=1)
```

Implementation of ANN Classification

In [29]:

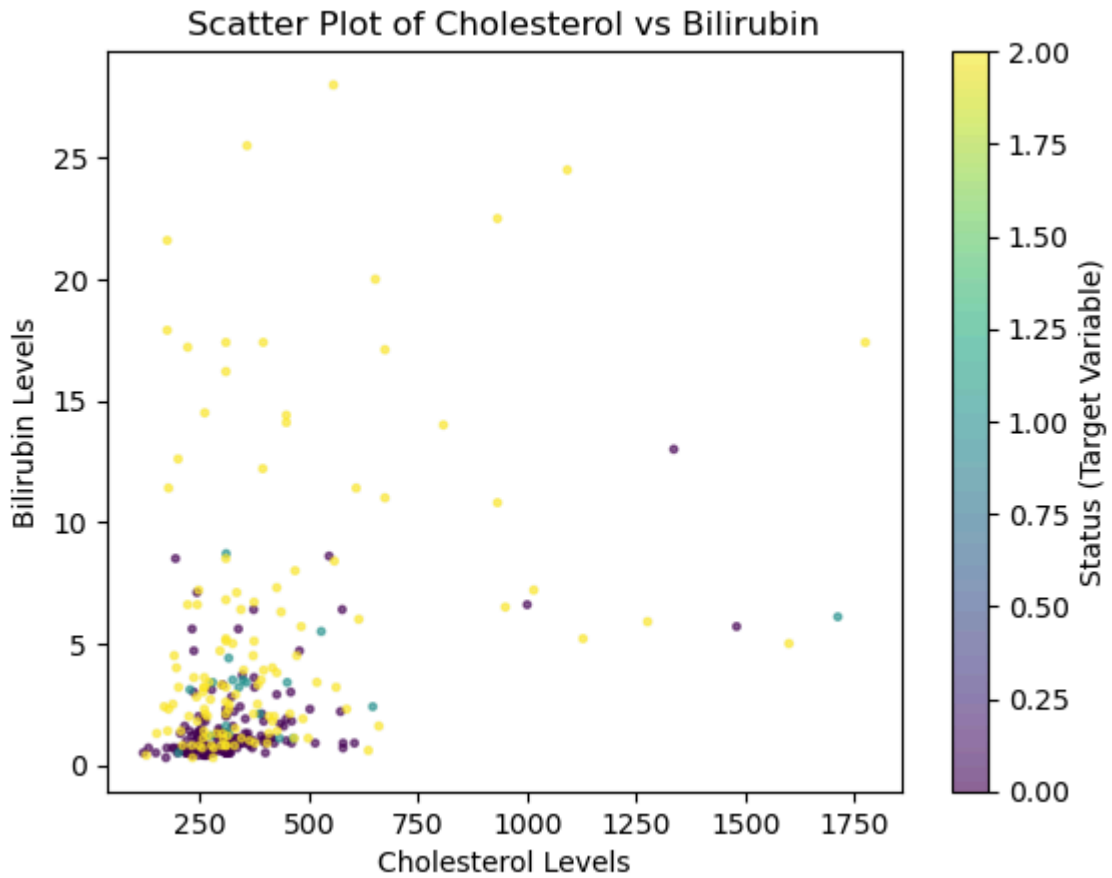
```
def main_class():
    D = 2
    K = 3
    N = int(K*1e3)


    plt.figure()
    plt.scatter(X[:,0],X[:,1], c=y, s=6, alpha=0.6)
    plt.title("Scatter Plot of Cholesterol vs Bilirubin")
    plt.xlabel("Cholesterol Levels")
    plt.ylabel("Bilirubin Levels")
    plt.colorbar(label="Status (Target Variable)")

    my_ann_classifier = ANN(architecture=[2,4],activations=[np.tanh,ReLU])#archi
    my_ann_classifier.fit(X,y,eta=2e-3,epochs=1e5)#eta=2e-3,epochs=1e4)
    y_hat = my_ann_classifier.predict(X)



    print(my_ann_classifier.W)
    print(my_ann_classifier.B)
    print(f"Training Accuracy: {accuracy(y,y_hat):0.4f}")

    x1 = np.linspace(X[:,0].min() - 1, X[:,0].max() + 1, 1000)
    x2 = np.linspace(X[:,1].min() - 1, X[:,1].max() + 1, 1000)

    xx1, xx2 = np.meshgrid(x1, x2)
    Z = my_ann_classifier.predict(np.c_[xx1.ravel(),xx2.ravel()]).reshape(*xx1.s
```

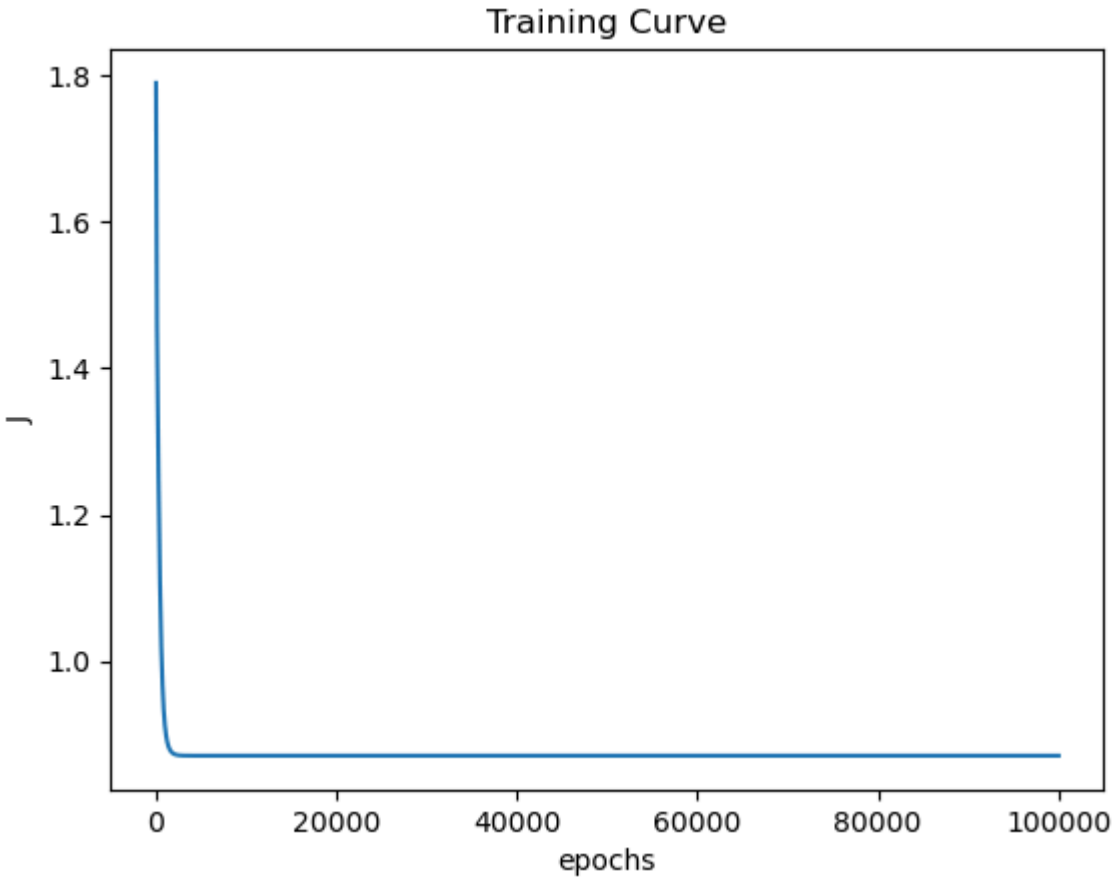
```
    plt.figure()
    plt.pcolormesh(xx1, xx2, Z, cmap = cmap_light)
    plt.scatter(X[:,0], X[:,1], c = y, cmap = cmap_bold,alpha=0.2)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    plt.title("Scatter Plot of Cholesterol vs Bilirubin")
    plt.xlabel("Cholesterol Levels")
    plt.ylabel("Bilirubin Levels")
    plt.colorbar(label="Status (Target Variable)")
    plt.show()

    plt.figure()
    plt.scatter(X[:,0],X[:,1],c=y_hat,s=6)
    plt.title("Scatter Plot of Cholesterol vs Bilirubin")
    plt.xlabel("Cholesterol Levels")
    plt.ylabel("Bilirubin Levels")
    plt.colorbar(label="Status (Target Variable)")
```

In [30]:
```
if __name__=="__main__":
  main_class()
```

```
{1: array([[-0.69988339, -0.75578334],
       [ 0.61182055, -0.95647639],
       [ 1.17776421,  0.75079025]]), 2: array([[ 0.09927232,  0.58902139, -0.2190
6135,  1.30795366],
       [-0.70116418,  0.82933996,  0.3337995 ,  0.20738769]]), 3: array([[-0.5932
0922, -1.58389074, -0.14144261],
       [ 0.57705063, -0.87550844, -0.38093903],
       [ 1.97387133,  2.02991465,  0.39705388],
       [-0.94748802,  0.64871452,  1.31302503]])}
{1: array([-2.17214746,  0.16242451]), 2: array([1.29459116, 1.27490898, 0.113319
39, 1.073858  ]), 3: array([1.58682808, 1.28611376, 0.43441013])}
Training Accuracy: 0.5385
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 2
      1 if __name__=="__main__":
----> 2   main_class()

Cell In[29], line 28, in main_class()
     25 x2 = np.linspace(X[:,1].min() - 1, X[:,1].max() + 1, 1000)
     27 xx1, xx2 = np.meshgrid(x1, x2)
---> 28 Z = my_ann_classifier.predict(np.c_[xx1.ravel(),xx2.ravel()]).reshape(*xx
1.shape)
     30 plt.figure()
     31 plt.pcolormesh(xx1, xx2, Z, cmap = cmap_light)

Cell In[28], line 74, in ANN.predict(self, X)
     73 def predict(self, X):
---> 74   self.__forward__(X)
     75   if self.mode:
     76     return self.Z[self.L]

Cell In[28], line 70, in ANN.__forward__(self, X)
     68 self.Z={0:X}
     69 for l in sorted(self.W.keys()):
---> 70   self.Z[l] = self.a[l](self.Z[l-1]@self.W[l]+self.B[l])

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with
gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 2)
```




Training Curve
1.8
1.6
1.4
1.2
1.0
J
0
20000
40000
60000
80000
100000
epochs

In [ ]: