Visvesvaraya National Institute of Technology (VNIT), Nagpur

Machine Learning with Python (ECL443)

Lab Report

Submitted by :
Prajyot Jadhav (BT20ECE046)
Semester 7

Submitted to :
Dr. Saugata Sinha
(Course Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

Experiment-2

Aim: To implement a classifier model with ANN to distinguish between cancer and normal patients.

Abstract: In this assignment, using stochastic gradient descent (SGD), a neural network for binary classification is implemented for distinguishing between cancer and normal patients. The dataset is iterated over during the training process over a number of epochs. The SGD algorithm is used to update the model at each epoch. After each epoch, validation is performed using a separate validation dataset. This method provides information about the model's performance during training and aids in the prevention of overfitting. During training, backpropagation is used to compute gradients for weights and biases. The model's performance is evaluated after training. The model's true positive, true negative, false positive, and false negative predictions are all found by the confusion matrix. The models performance is analyzed after increasing the complexity of the neural network.

Introduction: In this experiment, the MATLAB dataset 'ovarian.mat' is used for a binary classification problem. The stochastic gradient descent algorithm is used for updating the model parameters. The neural network architecture incorporates the sigmoid activation functions. Sigmoid is used in the hidden layer to introduce non-linearity. The ROC curve and the Area Under the Curve (AUC) are calculated and visualized. These metrics provide a comprehensive view of the model's ability to discriminate between positive and negative classes across different threshold values. The complexity of the neural network architecture is increased and the change in the accuracy of the model is analyzed.

Method:

- The load_file function is defined to load and process the data from the .mat file. The data is loaded using loadmat function from scipy.io, input data (features) labeled as 'ovarianInputs' and the corresponding target data (labels) labeled as 'ovarianTargets' are extracted. The indices of the data are shuffled randomly to ensure randomness in subsequent operations. The data is split into training, testing, and validation sets based on specified proportions, and the feature data is normalized by subtracting the mean and dividing by the standard deviation. The splitting ratios are 70% for training, 15% for testing, and 15% for validation.
- Next, with the initialize function, the initial parameters of the neural network are set up. The number of input features (n_x), the number of neurons in the

first hidden layer (C1), and the number of neurons in the output layer (C2) are passed as arguments. The weights and biases (W1, b1, W2, b2) are initialized using random values.

- For the forward pass, for applying the sigmoid activation function, a function called sigmoid is defined which will return the sigmoid of the inputs. Also, for calculating the error between the predicted and ideal values, a function cost is defined which returns the error as: $error = \frac{1}{2}(y_{ideal} - y_{predicted})^2$. The function returns this mean squared error and the average cost.
- The forward function calculates the weighted sum (Z) of inputs (X) using the transpose of the weights matrix (W.T) and the biases (b). The resulting Z is then passed through an activation function. The backward function implements backpropagation. Depending on the activation function, it calculates the gradients of the loss with respect to the parameters. For sigmoid activation, the gradient is calculated as $\dot{\sigma}(x) = \sigma(x)(1 - \sigma(x))$. Similarly , the gradients of weights (dW) and biases (db) are computed using the chain rule. The update function is used to update the parameters using gradient descent. It updates the weights and biases (W and b) by subtracting the product of the learning rate and their respective gradients (dW and db).
- Stochastic gradient descent is implemented with the SGD function. It performs forward and backward propagation, calculates the loss, and updates the parameters for each sample in the training dataset using the previously defined functions.
- For training the neural network a learning rate of 0.001 is used. A loop iterates through a specified number of epochs. Within each epoch, the code iterates through the samples in the training dataset. For each sample, the neural network is updated using the SGD function.
- The confusion matrix is computed using the confusion_matrix function from sklearn.metrics. From the confusion matrix, true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are extracted. Specificity and sensitivity are calculated using the derived values from the confusion matrix. The ROC curve is calculated using the roc_curve function from sklearn.metrics. Similarly, the area under the curve (AUC) is calculated using the auc function from sklearn.metrics.

Results:

- The training, testing set and the validation set are obtained by splitting the original data in the ratio of 70 %, 15 % and 15 %. The training set of (40,4)

with the labels of (40,1), the testing set of (11,4) with the labels of (11,1) and the validation set of (,) with labels of (,) are obtained.

```
Training data size:  (100, 151)
Training label size:  (2, 151)
Testing data size:  (100, 32)
Testing label size:  (2, 32)
Validation data size:  (100, 33)
Validation label size:  (2, 33)
```

Figure 1: Training, testing and validation datasets

- The neural network was trained for 100 epochs with one hidden layer. The number of neurons in the hidden layer were 50 and there are 2 neurons in the output layer. An accuracy of 93.75 % was obtained. Also the specificity and sensitivity was calculated and was found out to be 0.66 and 0.517 respectively. The ROC curve is plotted and AUC of 0.907 is obtained.

```
Accuracy:  93.75 %
Specificity: 0.6666666666666666, Sensitivity: 0.5172413793103449
AUC =  0.907843137254902
```

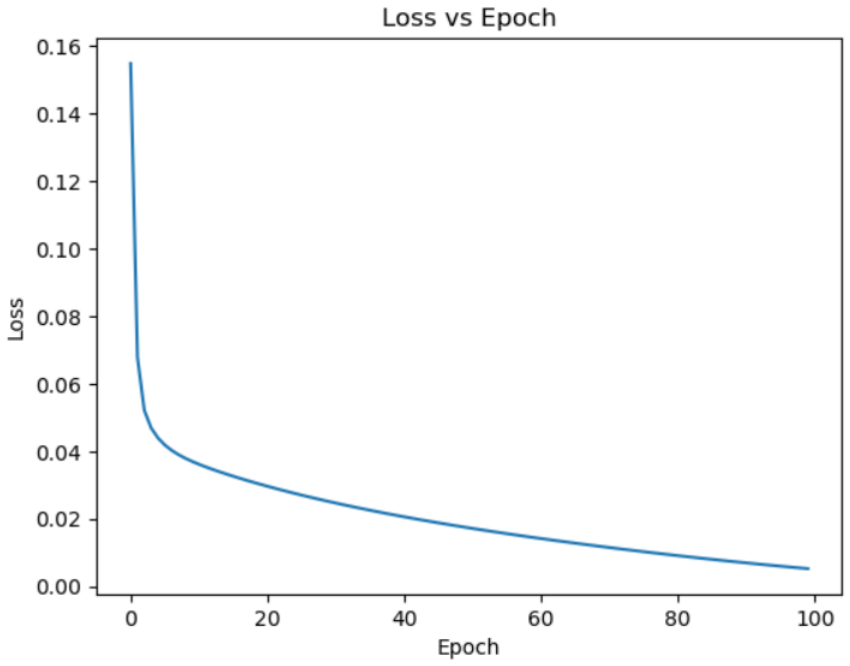

Figure 2: SGD Loss

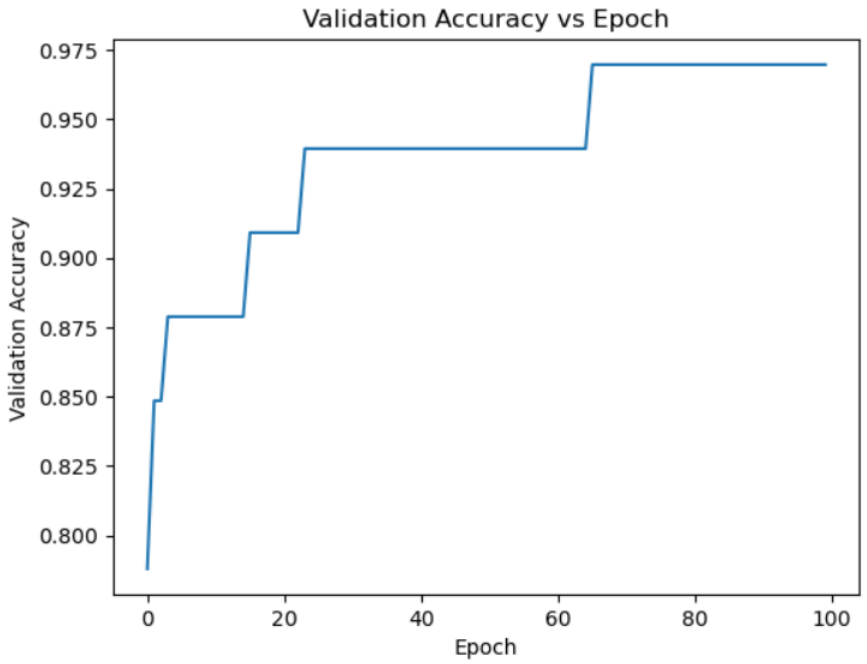

Figure 3: Validation set accuracy

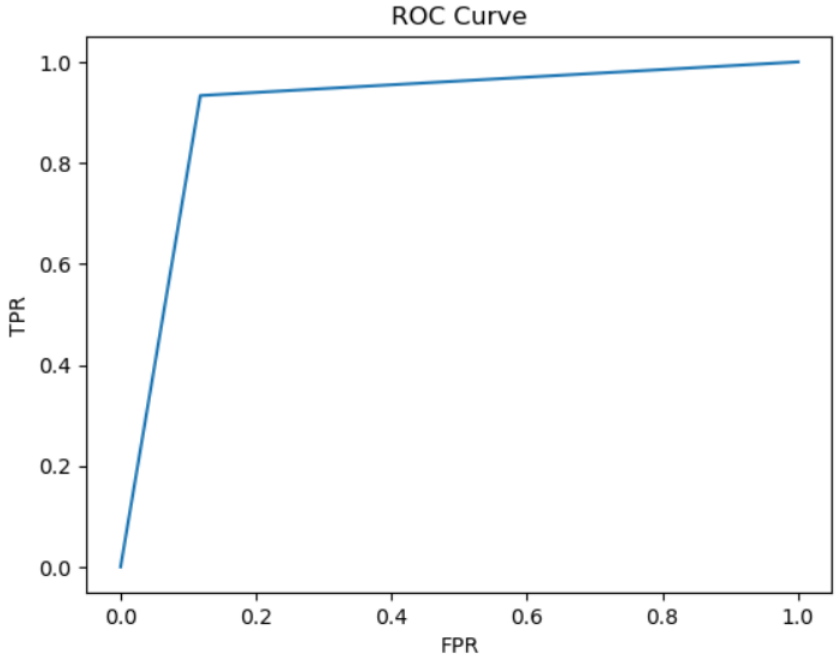

Figure 4: ROC Curve

- The above steps were repeated with 15 neurons in the first hidden layer. An accuracy of 92.15 % was obtained. Also the specificity and sensitivity was calculated and was found out to be 0.50 and 0.53 respectively. The ROC curve is plotted and AUC of 0.937 is obtained.

```
Accuracy:  92.1875 %
Specificity: 0.5, Sensitivity: 0.5333333333333333
AUC =  0.9372549019607843
```

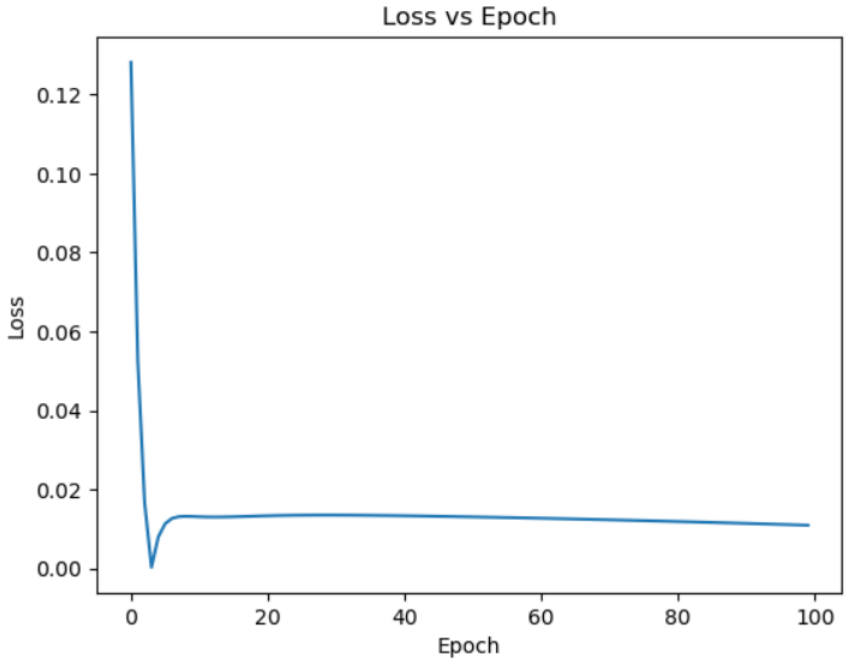

Figure 5: SGD Loss

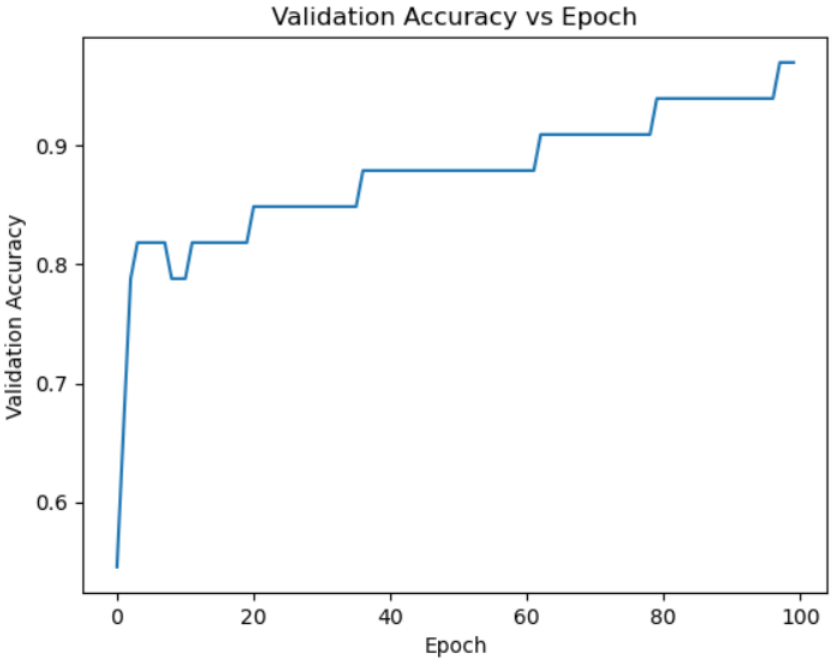

Figure 6: Validation set accuracy

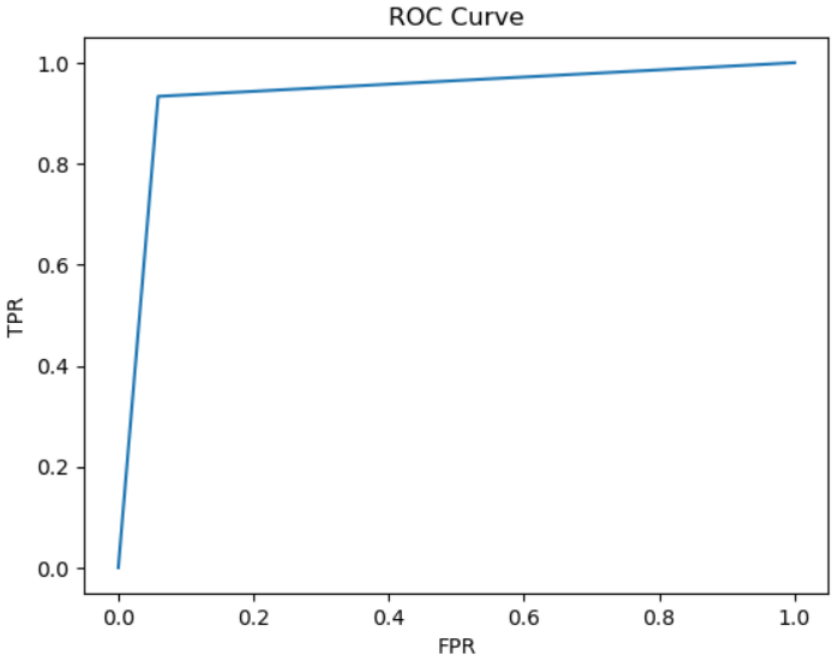

Figure 7: ROC Curve

Discussion:

- Binary classification is implemented with the stochastic gradient descent(SGD) algorithm. The training process involves iterating through the dataset over a number of epochs. For each epoch, the model is updated using the SGD algorithm. Additionally, validation is performed after each epoch using a separate validation dataset. This approach provides insights into the model's performance during training and helps prevent overfitting. Backpropagation is employed to compute gradients for weights and biases during training.
- After training, the model's performance is evaluated using various metrics. The confusion matrix provides insights into the model's true positive, true negative, false positive, and false negative predictions. Sensitivity (True Positive Rate) and Specificity (True Negative Rate) are computed to further analyze the model's accuracy.
- It was observed that on increasing the complexity of the network by increasing the number of neurons in the first hidden layer of the model, the accuracy obtained from the test set was slightly increased and overfitting was not observed. However, the specificity of the model reduced on increasing its complexity.
- It is possible that on further increasing the model complexity, by increasing the number of hidden layers, overfitting can occur. For such cases to prevent overfitting, we can observe the accuracy of the validation set after each epoch and decide when to stop the training process. .

Conclusion: The ANN classifier model is implemented to distinguish between cancer and normal patients. The stochastic gradient descent optimisation algorithm is used for updating the paramters. The model was trained for different architectures and the accuracy of the ANN architecture was obtained.

Appendix:

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

def load_file(path):
    np.random.seed(10)
    data = loadmat(path)
    X = data['ovarianInputs'].T
    Y = data['ovarianTargets'].T
    print(data.keys())
```

```
11      indices = np.random.permutation(len(Y))
12      X = X[indices]
13      Y = Y[indices]
14      X_train, X_test, X_val = np.split(X, [int(len(X)*0.7), ...
            int(len(X)*0.85)])
15      Y_train, Y_test, Y_val = np.split(Y, [int(len(Y)*0.7), ...
            int(len(Y)*0.85)])
16      #normalize data
17      X_train = (X_train - np.mean(X_train, ...
            axis=0))/np.std(X_train, axis=0)
18      X_test = (X_test - np.mean(X_test, axis=0))/np.std(X_test, ...
            axis=0)
19      X_val = (X_val - np.mean(X_val, axis=0))/np.std(X_val, axis=0)
20      print(X_train.T.shape, X_test.T.shape, X_val.T.shape, ...
            Y_train.T.shape, Y_test.T.shape, Y_val.T.shape)
21      return X_train.T, X_test.T,Y_train.T,Y_test.T,X_val.T,Y_val.T
22
23  dataset = ...
        load_file('C:/Users/Prajyot/Downloads/ovarian_dataset.mat')
24
25  m = 1
26  def initialize(n_x,C1,C2):
27  #     global W,b
28      np.random.seed(10)
29      W1 = np.random.randn(n_x,C1)*0.1
30      b1 = np.zeros((C1,1))
31      W2 = np.random.randn(C1,C2)*0.1
32      b2 = np.zeros((C2,1))
33
34      return W1, b1, W2, b2
35
36  def softmax(z):
37      t = np.exp(z)
38      a = t / np.sum(t, keepdims=True, axis=0)
39      return a
40
41  def sigmoid(z):
42      return 1/(1+np.exp(-z))
43
44  def forward(W, X, b,activation=None):
45  #     global Z,A
46      Z = np.dot(W.T, X) + b # Z.shape is (C,m)
47      if activation == 'sigmoid':
48          A = sigmoid(Z)
49      else:
50          A = Z
51      return Z, A
52  def cost(A, Y_hot):
53  #     global L,J
```

```
54 # Calculate Loss
55     L = 0.5*np.sum(A-Y_hot,keepdims=True, axis=0) # L.shape is (C,m)
56     J = np.mean(L)
57     return L,J
58 
59 # Genralized backprop function for multiple layers
60 def backward(X, Y_hot, A, Z, W, b, activation=None,cache=None):
61 #     global dW,db
62     if activation == 'softmax':
63         dZ = A - Y_hot
64     elif activation == 'sigmoid':
65         dZ = np.dot(cache[1],cache[0])*A*(1-A)
66     else:
67         dZ = A - Y_hot
68 
69     dW = np.dot(X, dZ.T)/m
70     db = np.mean(dZ, keepdims=True, axis=1)
71     return dW, db,dZ
72 
73 def update(W, b, dW, db, learning_rate):
74     W = W - learning_rate*dW
75     b = b - learning_rate*db
76     return W,b
77 
78 def SGD(X, Y_hot, W1, b1, W2, b2, learning_rate):
79     Z1, A1 = forward(W1, X, b1, 'sigmoid')
80     Z2, A2 = forward(W2, A1, b2, 'softmax')
81     L, J = cost(A2, Y_hot)
82     dW2, db2,dZ2 = backward(A1, Y_hot, A2, Z2, W2, b2)
83     dW1, db1,_ = backward(X, Y_hot, A1, Z1, W1, b1, ...
           'sigmoid',cache=(dZ2,W2))
84     W1,b1 = update(W1, b1, dW1, db1, learning_rate)
85     W2,b2 = update(W2, b2, dW2, db2, learning_rate)
86     return W1,b1,W2,b2,J
87 
88 
89 def predict(W1, b1, W2, b2, X):
90     _, A1 = forward(W1, X, b1, 'sigmoid')
91     _, A2 = forward(W2, A1, b2, 'softmax')
92     return A2
93 
94 def accuracy(Y_pred, Y):
95     return np.mean(Y_pred == Y)
96 
97 W1, b1, W2, b2 = initialize(100, 50, 2)
98 learning_rate = 0.001
99 costs = []
100 accs = []
101 #use SGD to train the model and validate at same time
```

```
for i in range(100):
    for j in range(dataset[0].shape[1]):
        X = dataset[0][:,j].reshape(-1,1)
        Y = dataset[2][:,j].reshape(-1,1)
        W1,b1,W2,b2,J = SGD(X, Y, W1, b1, W2, b2, learning_rate)
    costs.append(abs(J))
    #validate
    Y_pred = predict(W1, b1, W2, b2, dataset[4])
    acc = accuracy(np.argmax(Y_pred, axis=0), ...
        np.argmax(dataset[5], axis=0))
    print(f'Epoch {i+1}: Cost {J}, Val_accuracy {acc}')
    accs.append(acc)

plt.plot(costs)
plt.show()
plt.plot(accs)
plt.show()

Y_pred = predict(W1, b1, W2, b2, dataset[1])

# confusion matrix
from sklearn.metrics import confusion_matrix
Y_final = np.where(Y_pred ≥ 0.6, 1, 0)
accuracy(Y_final,dataset[3])
print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
cfm = confusion_matrix(np.argmax(dataset[3],axis=0), ...
    np.argmax(Y_pred, axis=0))
TP = cfm[0][0]
TN = cfm[0][1]
FP = cfm[1][0]
FN = cfm[1][1]
#Specificity
Specificity = TN/(TN+FP)
#Sensitivity
Sensitivity = TP/(TP+FN)
print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')

# ROC curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(np.argmax(dataset[3],axis=0), ...
    np.argmax(Y_pred, axis=0))
plt.plot(fpr, tpr)

#AUC
from sklearn.metrics import auc
print(auc(fpr, tpr))
```