Name: Deep Dodhiwala
UID: 2018140016
Batch A
Class: BE IT

Aim:
To build a classifier model using MultiLayer Perceptron Neural Network for a given scenario

<u>Task 1:</u> Prediction of heart disease using multilayer perceptron neural network

Problem solved by paper: The authors present a prediction system for heart disease using multilayer perceptron neural network

Description of MLP used:
a. Input Representation: There are 13 features used as input. All of them are continuous values except gender which is denoted by 1 or 0.
b. No. of hidden layers: 1
c. Activation Function(s) used: Not specified. Have assumed they used ReLu as it is default for all libraries.

Dataset Description: Cleveland heart disease database is used to feed the input to neural network. It consists of 14 features, with one of them being the target feature.

Results observed by the authors: The authors claim the following results with a 70-30 split

For Table 1, epochs were kept as 1000 and number of neurons in hidden layer was varied

TABLE I. PERFORMANCE OF THE SYSTEM WITH DIFFERENT NUMBER OF NEURONS

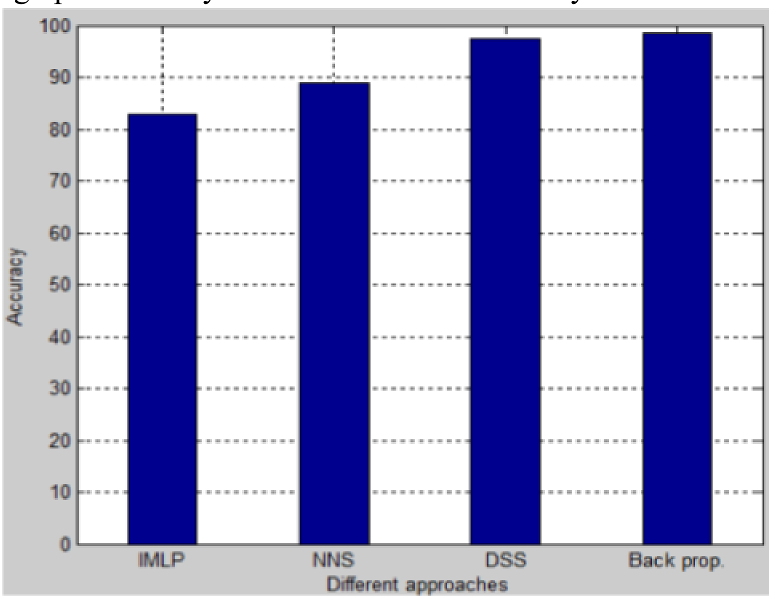
No. of Neurons	Acc	Sens.	Spec.	Error
5	92.92	92	93.75	7.07
10	95.75	95	96.42	4.24
15	96.69	96	98.21	3.30
20	98.58	98	98.21	1.41

For Table 3, number of neurons in hidden layer was set to 5 and epochs were varied

TABLE III. PERFORMANCE OF THE SYSTEM WITH DIFFERENT NUMBER OF EPOCHS

No. of Epochs	Acc	Sens.	Spec.	Error
1000	93.39	92	94.64	6.60
2000	94.33	93	95.53	5.66
3000	95.75	94	97.32	4.24
4000	97.64	98	97.32	2.35

A graph shown by them which shows accuracy of different methods



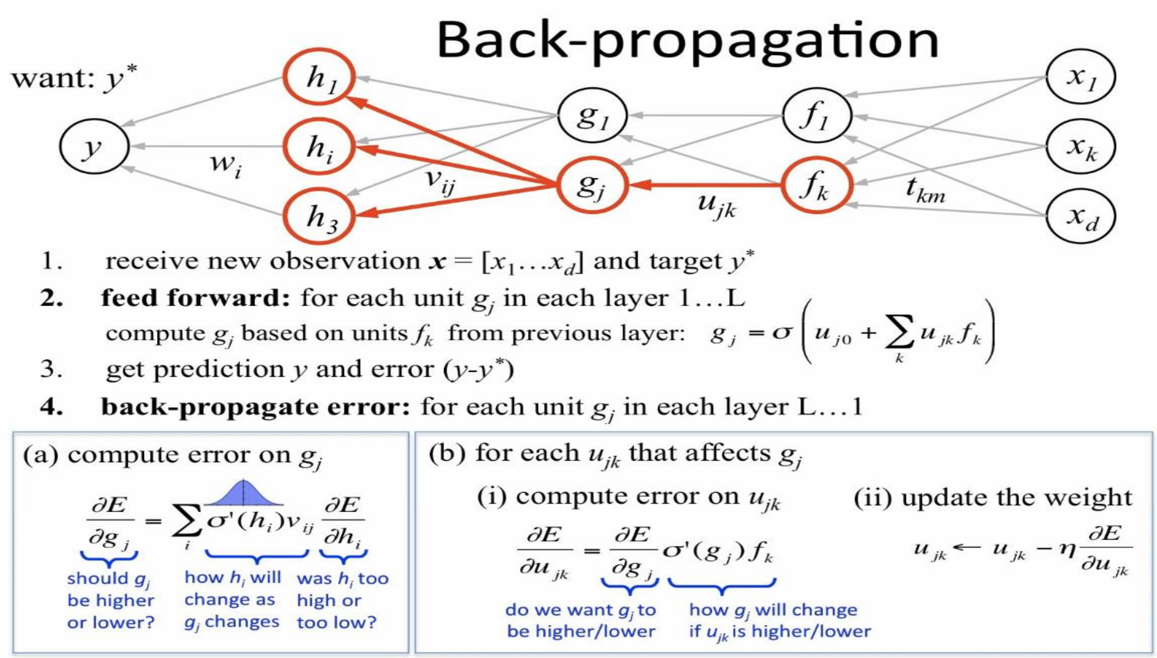
Observations and Conclusion: The authors claim that their usage of MLP network of 1 hidden layer provides the best results when the number of neurons in hidden layer are set to 20 and the epochs are fixed to 1000. They also claim their results are better than alternative methods as shown in the graph.

Task 2: Implement the chosen research paper by using the same components mentioned in the paper. Also vary the hyperparameters (hidden layer neurons, learning rate, activation function, optimizer) of the model built and obtain a comparative analysis.

Tool/Language:
Programming language: Python
Libraries: numpy/pandas/matplotlib/sklearn/tensorflow/keras/pytorch

Report Summary: The best results are obtained with the original approach with 5 neurons and 4000 training epochs, with the optimizer and activation function being Adam and ReLu respectively.

Algorithm: The algorithm here used is backpropagation



Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('heart.csv')
df = df.sample(frac=1).reset_index(drop=True)
df.head()

X = df[df.columns[:-1]]
y = df[df.columns[-1]]

X

# They haven't mentioned normalization, but their claimed high accuracies wouldn't be possible without it.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.metrics import confusion_matrix
def calc_metrics(cm):
    total = cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1]
    print("Accuracy = ",(cm[0,0]+cm[1,1]) * 100/total)
```

```
    print("Sensitivity = ",(cm[0,0]) * 100/(cm[0,0]+cm[0,1]))
    print("Specificity = ",(cm[1,1]) * 100/(cm[1,0]+cm[1,1]))

from sklearn.neural_network import MLPClassifier
def solve(hn, iters, solver, activation, X_train, X_test, y_train, y_test):
    clf = MLPClassifier(random_state=1,
                        hidden_layer_sizes=(hn,),
                        learning_rate='adaptive',
                        max_iter=iters,
                        activation=activation,
                        solver=solver).fit(X_train, y_train)

    cm = confusion_matrix(y_test, clf.predict(X_test))
    print(cm)
    calc_metrics(cm)

"""# Performance with Different Number of Neurons"""

solve(5, 1000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(10, 1000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(15, 1000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(20, 1000, 'adam', 'relu', X_train, X_test, y_train, y_test)

plt.plot([89.010989, 84.615384, 79.120879, 84.615384], color='red', alpha=0.8,
label='Accuracy')
plt.plot([86.11111, 72.2222, 66.6666, 72.2222], color='green', alpha=0.8, label='Sensitivity')
plt.plot([90.9090, 92.7272, 87.2727, 92.727], color='magenta', alpha=0.8, label='Specificity')
x = [5, 10, 15, 20]
# create an index for each tick position
xi = list(range(len(x)))
plt.xticks(xi,x)
plt.legend()
plt.show()

"""# Performance with Different Number of Training Epochs"""

solve(5, 1000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(5, 2000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(5, 3000, 'adam', 'relu', X_train, X_test, y_train, y_test)

solve(5, 4000, 'adam', 'relu', X_train, X_test, y_train, y_test)
```

```
plt.plot([89.01098, 89.01098, 89.01098, 89.01098], color='red', alpha=0.8, label='Accuracy')
plt.plot([86.111, 86.111, 86.111, 86.111], color='green', alpha=0.8, label='Sensitivity')
plt.plot([90.9090, 90.90, 90.90, 90.90], color='magenta', alpha=0.8, label='Specificity')
x = [1000, 2000, 3000, 4000]
# create an index for each tick position
xi = list(range(len(x)))
plt.xticks(xi,x)
plt.legend()
plt.show()

"""# Changing the Solver from ADAM to SGD"""

# Higher accuracy and sensitivity as compared to original 20 neuron and 1000 epoch setup
solve(20, 1000, 'sgd', 'relu', X_train, X_test, y_train, y_test)

x = ['Accuracy', 'Sensitivity', 'Specificity']
X_axis = np.arange(len(x))

original = [84.6153846153, 72.222222, 92.7272]
sgd = [85.714, 75, 92.72]

plt.bar(X_axis - 0.2, original, 0.4, label = 'Adam')
plt.bar(X_axis + 0.2, sgd, 0.4, label = 'SGD')

plt.xticks(X_axis, x)
plt.legend()
plt.show()

# Lower accuracy, sensitivity and specificity as compared to original 5 neuron and 4000 epoch
setup
solve(5, 4000, 'sgd', 'relu', X_train, X_test, y_train, y_test)

x = ['Accuracy', 'Sensitivity', 'Specificity']
X_axis = np.arange(len(x))

original = [89.010, 86.11, 90.90]
sgd = [84.615, 80.55, 80.55]

plt.bar(X_axis - 0.2, original, 0.4, label = 'Adam')
plt.bar(X_axis + 0.2, sgd, 0.4, label = 'SGD')

plt.xticks(X_axis, x)
plt.legend()
plt.show()
```

```
"""# Changing the Activation function from ReLu to TanH"""

# Lower accuracy, specificity and sensitivity as compared to original 20 neuron and 1000 epoch setup.
solve(20, 1000, 'adam', 'tanh', X_train, X_test, y_train, y_test)

x = ['Accuracy', 'Sensitivity', 'Specificity']
X_axis = np.arange(len(x))

original = [84.6153846153, 72.222222, 92.7272]
tanh = [82.4175, 69.44, 90.90]

plt.bar(X_axis - 0.2, original, 0.4, label = 'ReLu')
plt.bar(X_axis + 0.2, tanh, 0.4, label = 'TanH')

plt.xticks(X_axis, x)
plt.legend()
plt.show()

# Significantly lower accuracy and sensitivity but higher specificity as compared to original 20 neuron and 1000 epoch setup.
solve(5, 4000, 'adam', 'tanh', X_train, X_test, y_train, y_test)

x = ['Accuracy', 'Sensitivity', 'Specificity']
X_axis = np.arange(len(x))

original = [89.010, 86.11, 90.90]
tanh = [83.51, 69.44, 92.72]

plt.bar(X_axis - 0.2, original, 0.4, label = 'ReLu')
plt.bar(X_axis + 0.2, tanh, 0.4, label = 'TanH')

plt.xticks(X_axis, x)
plt.legend()
plt.show()
```

Results:

Chart of Accuracy, Sensitivity and Specificity when neurons are varied

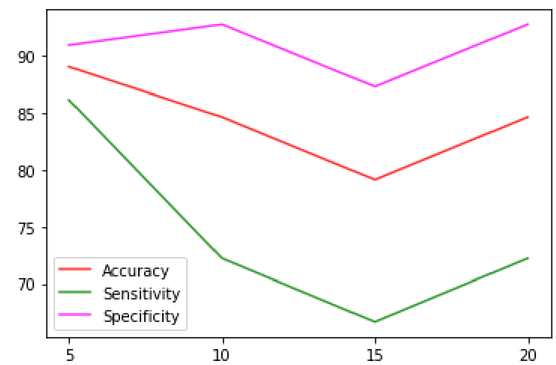
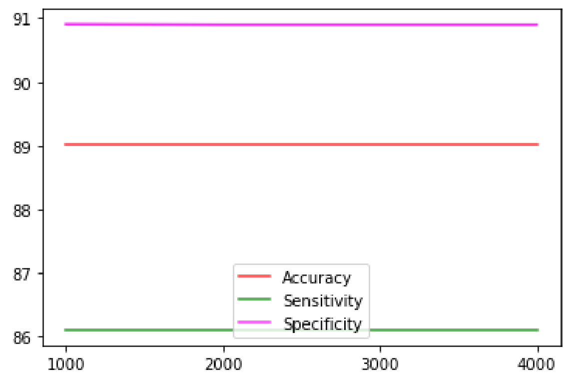
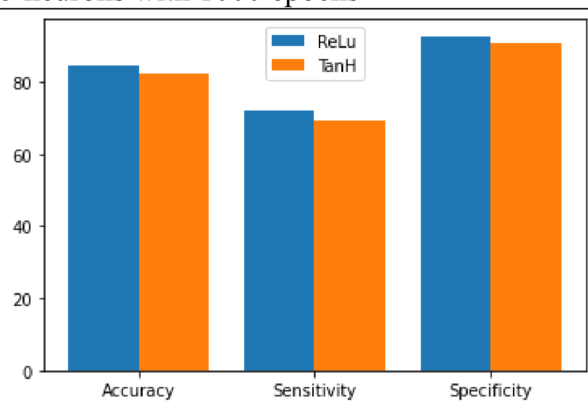
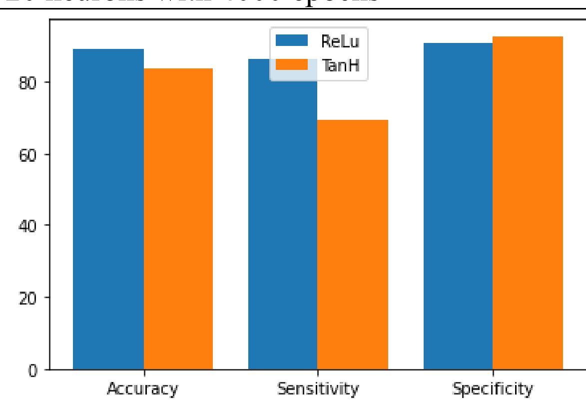


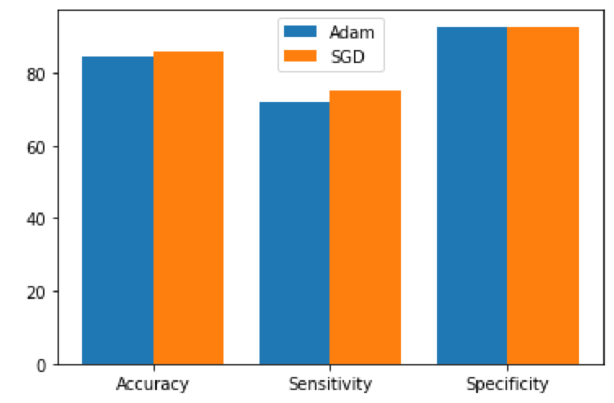
Chart of Accuracy, Sensitivity and Specificity when epochs are varied but hidden layer neurons are 5. Since there are so few weights, they get optimized quickly and using more epochs is redundant as done in the paper.

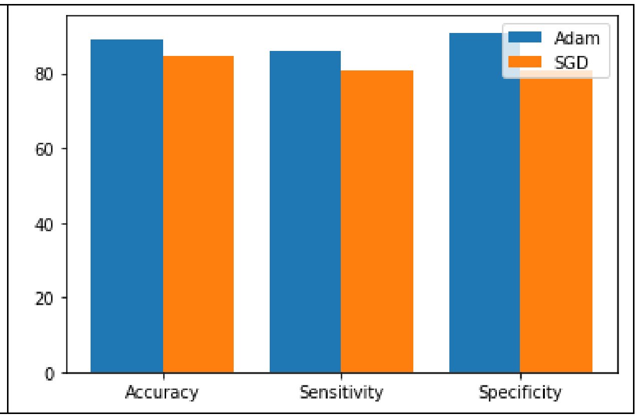


Varying the program and comparing it with two approaches of the authors

ReLu (Author) vs TanH (variation)	
5 neurons with 1000 epochs	20 neurons with 4000 epochs
	

Adam (Author) vs SGD (variation)	
5 neurons with 1000 epochs	20 neurons with 4000 epochs





Conclusion: Thus, the best results are obtained with the original approach with 20 neurons and 4000 training epochs, with the optimizer and activation function being Adam and ReLu respectively. Changing the optimizer to SGD or changing the activation function to TanH results in a decrease in the metrics used by the authors.