Name: Paarth Kapur
Roll No: 16010420038
Div: A2
Deep Learning Exp 2

Results:

```
import math
import pandas as pd
from keras import models, layers, optimizers, regularizers
import numpy as np
import random
from sklearn import model_selection, preprocessing
import tensorflow as tf
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```
[9] file_name = 'SAheart.data'
    data = pd.read_csv(file_name, sep=',', index_col=0)
```

```
[10] data['famhist'] = data['famhist'] == 'Present'
     data.head()
```

row.names	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
1	160	12.00	5.73	23.11	True	49	25.30	97.20	52	1
2	144	0.01	4.41	28.61	False	55	28.87	2.06	63	1
3	118	0.08	3.48	32.28	True	52	29.14	3.81	46	0
4	170	7.50	6.41	38.03	True	51	31.99	24.26	58	1

```
[11] n_test = int(math.ceil(len(data) * 0.3))
     random.seed(42)
     test_ixs = random.sample(list(range(len(data))), n_test)
     train_ixs = [ix for ix in range(len(data)) if ix not in test_ixs]
     train = data.iloc[train_ixs, :]
     test = data.iloc[test_ixs, :]
     print(len(train))
     print(len(test))
```

```
323
139
```

```
features = ['adiposity', 'age']
response = 'chd'
x_train = train[features]
y_train = train[response]
x_test = test[features]
y_test = test[response]
```

```
[13] x_train = preprocessing.normalize(x_train)
     x_test = preprocessing.normalize(x_test)
```

```
[14] hidden_units = 10
     activation = 'relu'
     l2 = 0.01
     learning_rate = 0.01
     epochs = 5
     batch_size = 16
```

```
model = models.Sequential()


model.add(layers.Dense(input_dim=len(features),
                       units=hidden_units,
                       activation=activation))


model.add(layers.Dense(input_dim=hidden_units,
                       units=1,
                       activation='sigmoid'))


model.compile(loss='binary_crossentropy',

              optimizer=optimizers.Adam(lr=learning_rate),
              metrics=['accuracy'])
```

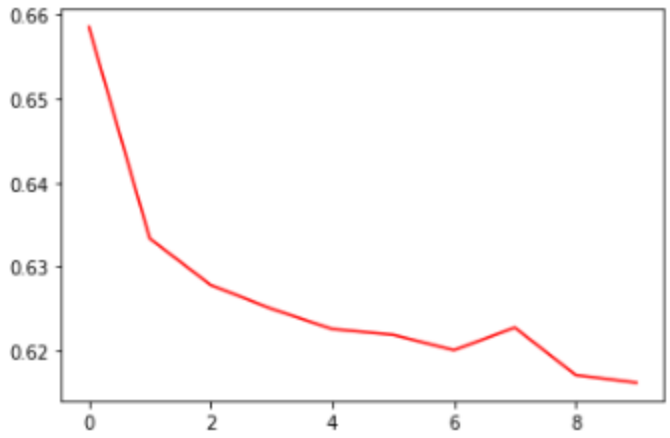
```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

```
history = model.fit(x_train, y_train, epochs=10, batch_size=batch_size)


train_acc = model.evaluate(x_train, y_train, batch_size=32)[1]
test_acc = model.evaluate(x_test, y_test, batch_size=32)[1]
print('Training accuracy: %s' % train_acc)
print('Testing accuracy: %s' % test_acc)

losses = history.history['loss']
plt.plot(range(len(losses)), losses, 'r')
plt.show()
```

```
Epoch 1/10
21/21 [==============================] - 1s 4ms/step - loss: 0.6585 - accuracy: 0.6192
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6333 - accuracy: 0.6718
Epoch 3/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6278 - accuracy: 0.6718
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6250 - accuracy: 0.6718
Epoch 5/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6225 - accuracy: 0.6718
Epoch 6/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6219 - accuracy: 0.6718
Epoch 7/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6200 - accuracy: 0.6718
Epoch 8/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6227 - accuracy: 0.6718
Epoch 9/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6171 - accuracy: 0.6718
Epoch 10/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6162 - accuracy: 0.6718
11/11 [==============================] - 0s 2ms/step - loss: 0.6142 - accuracy: 0.6718
5/5 [==============================] - 0s 4ms/step - loss: 0.6604 - accuracy: 0.6115
Training accuracy: 0.6718266010284424
Testing accuracy: 0.6115108132362366
```



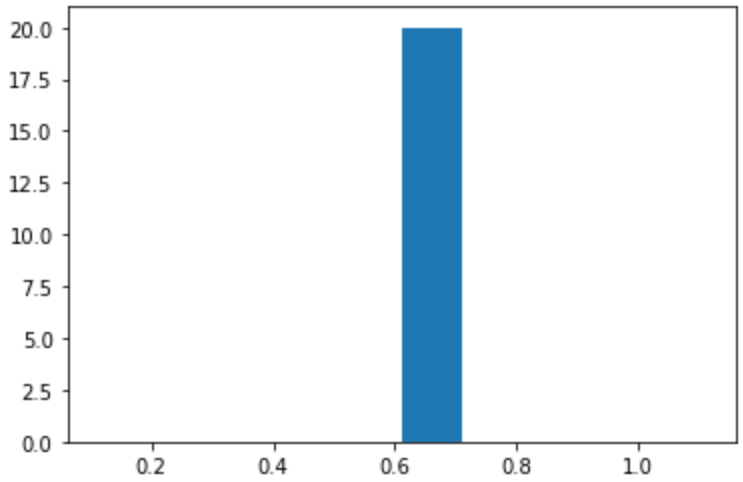

```python
def train_and_evaluate(model, x_train, y_train, x_test, y_test, n=20):
    train_accs = []
    test_accs = []
    with tqdm(total=n) as progress_bar:
        for _ in range(n):
            model.fit(
                x_train,
                y_train,
                epochs=epoc
                batch_size=batch_size,
                verbose=False)
            train_accs.append(model.evaluate(x_train, y_train, batch_size=32, verbose=False)[1])
            test_accs.append(model.evaluate(x_test, y_test, batch_size=32, verbose=False)[1])
            progress_bar.update()
    print('Avgerage Training Accuracy: %s' % np.average(train_accs))
    print('Avgerage Testing Accuracy: %s' % np.average(test_accs))
    return train_accs, test_accs
_, test_accs = train_and_evaluate(model, x_train, y_train, x_test, y_test)
```

int: batch_size

16

```
100%|██████████| 20/20 [00:05<00:00,  3.39it/s]Avgerage Training Accuracy: 0.6713622093200684
Avgerage Testing Accuracy: 0.6115108132362366
```

```
plt.hist(test_accs)
plt.show()
```




```
[19] print('Min: %s' % np.min(test_accs))
     print('Max: %s' % np.max(test_accs))
```

```
Min: 0.6115108132362366
Max: 0.6115108132362366
```

```
hidden_units = 10
activation = 'relu'
l2 = 0.01
learning_rate = 0.01
epochs = 5
batch_size = 16
```

```
[21] # create a sequential model
     model = models.Sequential()

     # add the hidden layer
     model.add(layers.Dense(input_dim=len(features),
                            units=hidden_units,
                            activation=activation))

     # add the output layer
     model.add(layers.Dense(input_dim=hidden_units,
                            units=1,
                            activation='sigmoid'))

     # define our loss function and optimizer
     model.compile(loss='binary_crossentropy',
                   # Adam is a kind of gradient descent
                   optimizer=optimizers.Adam(lr=learning_rate),
                   metrics=['accuracy'])
```

```
_, __ = train_and_evaluate(model, x_train, y_train, x_test, y_test)
```

```
100%|██████████| 20/20 [00:07<00:00,  2.69it/s]Avgerage Training Accuracy: 0.6718266010284424
Avgerage Testing Accuracy: 0.6115108132362366
```

Post-lab questions:

1. What is perceptron?
   a. a single layer feed-forward neural network with pre-processing
   b. an auto-associative neural network
   c. a double layer auto-associative neural network
   d. a neural network that contains feedback

Ans. a

2. A 4-input neuron has weights 1, 2, 3 and 4. The transfer function is linear with

the constant of proportionality being equal to 2. The inputs are 4, 10, 5 and 20 respectively. What will be the output?

a. 76
b. 119
c. 123
d. 238

Ans. d

3. A perceptron adds up all the weighted inputs it receives, and if it exceeds a certain value, it outputs a 1, otherwise it just outputs a 0.

a. True
b. False
c. Sometimes – it can also output intermediate values as well
d. Can't say

Ans. a

Outcome:

CO2: Comprehend the Deep Network concepts.

Conclusion:

Successfully Implemeneted feed forward neural network and soft max function in python..