10.

Step 1 import the required libraries

```
import tensorflow as tf
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

Step 2 load the dataset

```
# load the data
df = pd.read_csv('diabetes.csv')
print(df.shape)
print(df.describe().transpose())
```

These lines of code simply load the dataset as a pandas dataframe and print a brief summary of the dataset

Output:

```
(768, 9)
                          count        mean  ...        75%     max
Pregnancies               768.0    3.845052  ...    6.00000   17.00
Glucose                   768.0  120.894531  ...  140.25000  199.00
BloodPressure             768.0   69.105469  ...   80.00000  122.00
SkinThickness             768.0   20.536458  ...   32.00000   99.00
Insulin                   768.0   79.799479  ...  127.25000  846.00
BMI                       768.0   31.992578  ...   36.60000   67.10
DiabetesPedigreeFunction  768.0    0.471876  ...    0.62625    2.42
Age                       768.0   33.240885  ...   41.00000   81.00
Outcome                   768.0    0.348958  ...    1.00000    1.00
```

Step 3 separate the predictor and the target variable then normalized the predictor variable and convert the target variable into one hot encoding

```
# separate predictor and target then normalize the data
target_column = ['Outcome']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
X = df[predictors].values
y = df[target_column].values
y_ohe = tf.keras.utils.to_categorical(y, num_classes = 2)
```

The first line of the code sets the target variable as 'Outcome' which refer to the Outcome column in the dataset. The 2nd line gives us the list of the features excluding the target variable as the predictor variable. The 3rd line normalizes the predictor variable. The 4th and 5th line stores the predictor variable and target variable into variable X and y respectively. The 6th converts the target variable into one hot encoding vector.

Step 4 split the data into train set and test set

```
# split data for train and test
X_train, X_test, y_train, y_test = train_test_split(X, y_ohe, test_size=0.3,
random_state=40)
print(X_train.shape); print(X_test.shape)
```

For the training set and test set I follow same split configuration which is 70% training set 30% test set

Output

```
(537, 8)
(231, 8)
```

Step 5 create a model and train it

```
# create a model
model = Sequential()
model.add(Dense(8, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(2, activation='softmax'))
callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
                                            min_delta=1e-4,
                                            patience=10)
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=500, batch_size=200,
          validation_data=(X_test, y_test), callbacks=[callback])
```

On the experiment https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn the author tried to create a classification model using MLP with this following configuration:

- Input layer with 8 neurons, 3 hidden layers of 8 neurons, output layer with 2 neurons.
- Activation function = Relu for the hidden layer, SoftMax for the output layer
- Loss function = log loss function / cross entropy
- Optimizer = adam
- Batch size = 200
- Early stop = when the loss score is not improving by at least 1e-4 for 10 iterations
- Max epoch = 500

Some of the information are retrieved from scikit-learn documentation https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

To create the model I utilized keras.Sequential() because it is suitable for a plain stack of layers where each layer has exactly one input tensor and one output tensor. The 2nd line until 4th line initiates the input layer and the 3 hidden layers which each has 8 neurons with ReLu activation function. The 5th line initiates the output layer which has 2 neurons with SoftMax activation function

The 6th line initiates early stopping. The 7th line configure the loss function and the optimizer. The 7th line trains the model with 500 epoch and batch size = 200 with early stop

Step 6 evaluate the model

```
# evaluate the model
predict_train = model.predict(X_train)
print('confusion matrix for train set')
print(confusion_matrix(y_train.argmax(axis=1),predict_train.argmax(axis=1)))
print(classification_report(y_train.argmax(axis=1),predict_train.argmax(axis=
1)))

predict_test = model.predict(X_test)
print('confusion matrix for test set')
print(confusion_matrix(y_test.argmax(axis=1),predict_test.argmax(axis=1)))
print(classification_report(y_test.argmax(axis=1),predict_test.argmax(axis=1)
))
```

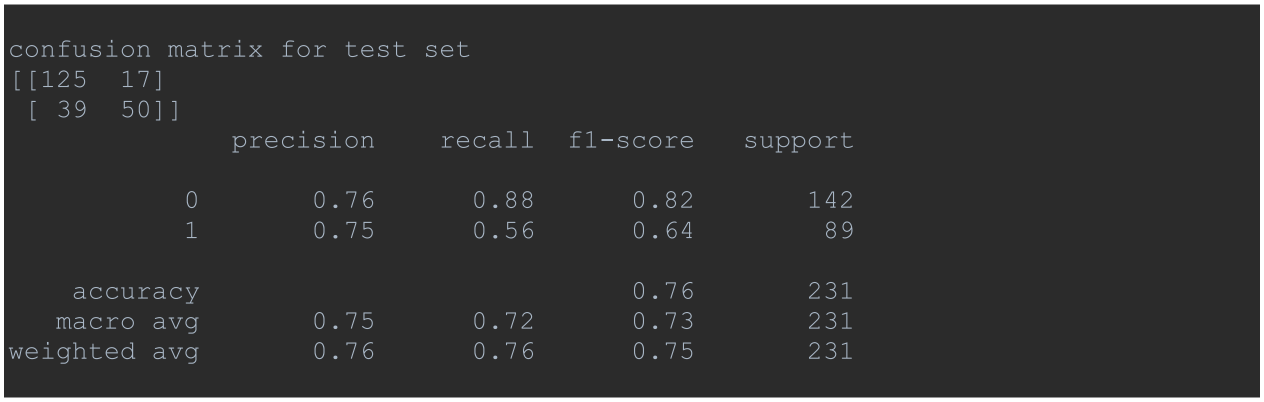
After the model trained, now we able to evaluate the model using confusion matrix. The 1st line of the code evaluate the model with the training set and the 2nd line evaluate the model with the test set

Output

```
confusion matrix for train set
[[317  41]
 [ 75 104]]
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       358
           1       0.72      0.58      0.64       179

    accuracy                           0.78       537
   macro avg       0.76      0.73      0.74       537
weighted avg       0.78      0.78      0.78       537
```



```
confusion matrix for test set
[[125  17]
 [ 39  50]]
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       142
           1       0.75      0.56      0.64        89

    accuracy                           0.76       231
   macro avg       0.75      0.72      0.73       231
weighted avg       0.76      0.76      0.75       231
```

11. I couldn't open this link https://keras.io/examples/mnist_mlp/ so I'm decided to make my own expriment

Step 1 import the required libraries

```
from sklearn.datasets import fetch_openml
from sklearn.neural_network import MLPClassifier
```

Step 2 load the dataset and normalize the predictor variable

```
#load the dataset
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
X = X / 255.
print(X.shape);print(y.shape)
```

To collect the data I used sklearn.datasets.fetch_openml. This library will download the MNIST dataset and separate it into X and y which is the predictor variable and target variable respectively. The 2nd line normalize the predictor variable. The 3rd line gives us the shape of the X and y. 70.000 means that the data contains 70.000 instance, and 784 is number of features on each instance

Output

```
(70000, 784)
(70000,)
```

Step 3 split the dataset into train set and test set

```
#split the data into train set and test set
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

MNIST contain 60.000 instance of training set and 10.000 test set. So we need to split the data accordingly. The 1st line of the code sets the X_train to contain the first 60.000 instance on the predictor variable and X_test to contain the rest 10.000 instance on the predictor variable. The 2nd line do the same thing with the 1st line but for the target variable. The 3rd and 4th line gives us the shape of each variable to check if we separate the data correctly

Output

```
(60000, 784) (10000, 784)
(60000,) (10000,)
```

Step 4 create a model and train it

```
#create a model and train it
mlp = MLPClassifier(hidden_layer_sizes=(50,),
                    activation='relu',
                    solver='adam',
                    verbose=True)
mlp.fit(X_train, y_train)
```

The 1st line until 4th line of the code creates a model with 1 hidden layer of 50 hidden neurons with ReLu activation function and adam optimizer. The 5th trains the model.

Step 5 evaluate the model

```
#print the prediction result
print('accuracy for train set:', mlp.score(X_train, y_train))
print('accuracy for test set:', mlp.score(X_test, y_test))
```

After the model trained, now we able to evaluate the model using accuracy score. The 1st line of the code calculate the accuracy of the model with the training set and the 2nd line calculate the accuracy of the module using the test set

Output

```
accuracy for train set: 0.9782
accuracy for test set: 0.9669
```

12.

a.i. The number of input layer units is equal to the number of input attributes, because we have 7 attributes and the last attribute is actually a class attribute so we will have 6 input attributes so 6 units in the input layer.

The number of hidden units in the hidden layer = (6+1)/2 = 3.5 = 3

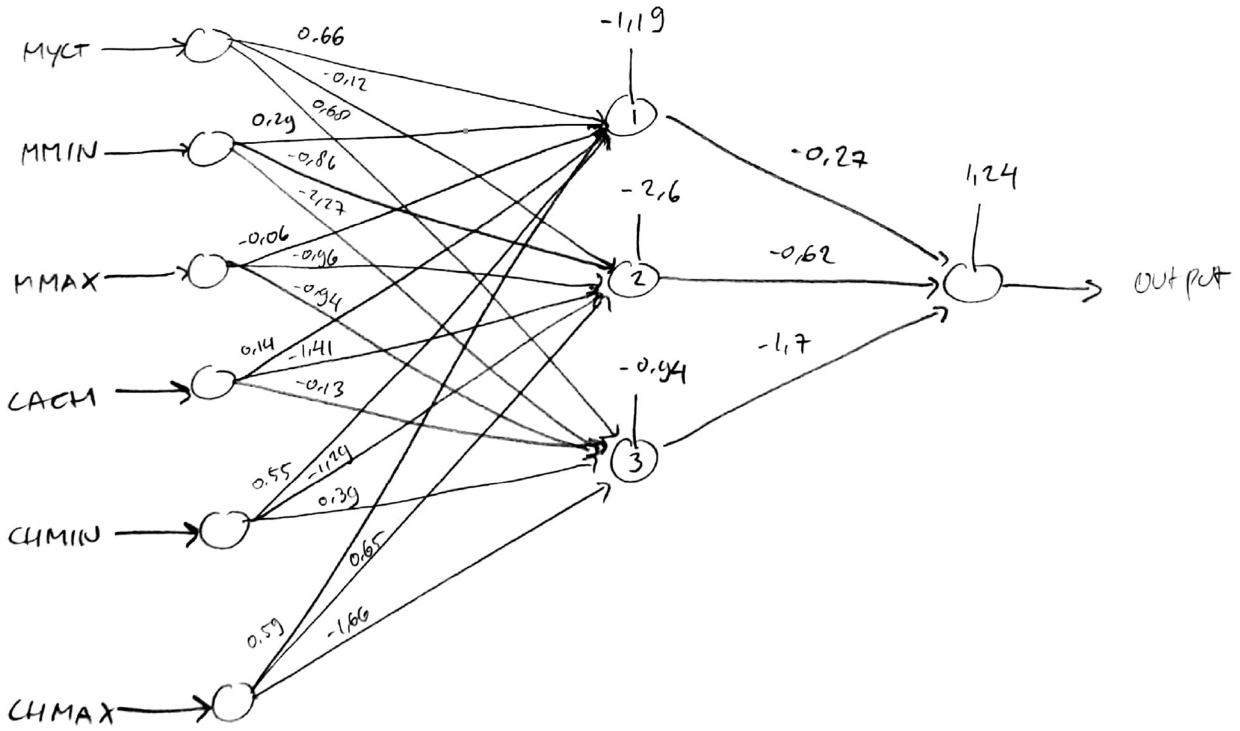
The number of units in the output layer is 1 because the class attributes has a continuous value

a.ii. Mean absolute error 116.7667

a.iii. Root mean squared error 120.516

a.iv.

a.v.



b.i Mean absolute error 150.5646

b.ii Root mean squared error 154.5293

b.iii.

c.i. It has 4 layers which is 1 input layer, 2 hidden layer, and 1 output layer

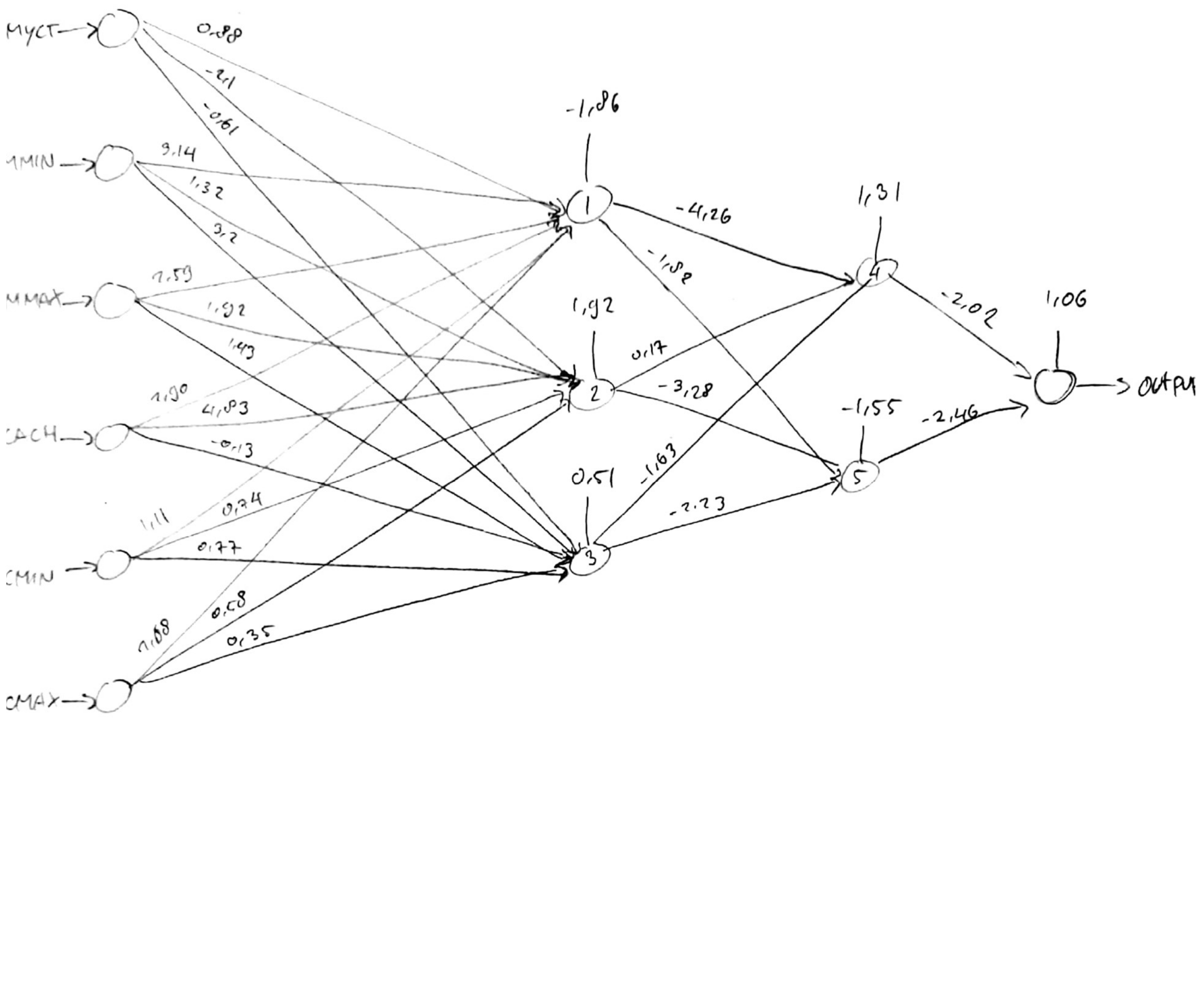
c.ii. input layer = 6, 1st hidden layer = 3, 2nd hidden layer = 2, output layer = 1

c.iii. Mean absolute error 182.867

c.iv. Root mean squared error 185.0802

c.v.

c.vi.


MYCT
MMIN
MMAX
CACH
CMIN
CMAX
0,88
-2,1
-0,61
3,14
1,32
3,2
1,59
1,92
1,43
1,90
4,83
-0,13
1,11
0,74
0,77
1,08
0,68
0,35
-1,86
1
1,92
2
0,51
3
-4,26
-1,08
0,17
-3,28
-1,63
-2,23
1,31
4
-1,55
5
-2,02
-2,46
1,06
output