hw7-8

November 10, 2023

1 Lab: Multilayer Perceptron (HW7-8) – Layth Aljorani

1.1 0. Set Up

[1]:
```
import tensorflow as tf
from tensorflow import keras
```

```
2023-11-09 23:52:09.652914: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-11-09 23:52:10.304472: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to
register cuDNN factory: Attempting to register factory for plugin cuDNN when one
has already been registered
2023-11-09 23:52:10.304500: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2023-11-09 23:52:10.307517: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2023-11-09 23:52:10.661429: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-11-09 23:52:10.665206: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2023-11-09 23:52:15.325565: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:
```
tf.__version__
```

[2]:
```
'2.14.0'
```

[3]:
```
mlp_model = keras.models.Sequential()
mlp_model.add(keras.layers.Dense(5, activation='relu', input_shape=(3,)))
```

```
mlp_model.add(keras.layers.Dense(10, activation='sigmoid'))
mlp_model.add(keras.layers.Dense(1, activation='sigmoid'))

mlp_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 20

 dense_1 (Dense)             (None, 10)                60

 dense_2 (Dense)             (None, 1)                 11

=================================================================
Total params: 91 (364.00 Byte)
Trainable params: 91 (364.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[4]:
```
# how many layer and neurons are hyper parameters

mlp_model = keras.models.Sequential(
    [
        # input shape is 3 features + bias; feeding into a 5 neuron Dense layer;
 ↪ thus 20 params
        keras.layers.Dense(5, activation='relu', input_shape=(3,)),
        # then 5 neurons + bias into 10 neurons Dense layer; thus 60 params
        keras.layers.Dense(10, activation='sigmoid'),
        # then 10 neurons + bias into one output layer; thus 11 params
        keras.layers.Dense(1, activation='sigmoid')
    ]
)

# 'mse' for regression
# 'binary_crossentropy' for classification
mlp_model.compile(loss='mse',
                  optimizer='adam')

mlp_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 5)                 20
```

```
 dense_4 (Dense)             (None, 10)                60

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 91 (364.00 Byte)
Trainable params: 91 (364.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[5]:
```
# mlp_model.fit(X, y, epochs=300) # epochs = iterations
```

1.2 1. Regression

1.2.1 SET UP

Consider the data "Advertising.csv".

[6]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

1. Scale the features using MinMaxScaler

[7]:
```
df = pd.read_csv("./Advertising.csv")
df = df.drop(df.columns[0], axis=1)
X = df[['TV', 'radio', 'newspaper']]
y = df[['sales']]

scaler = MinMaxScaler().fit(X)
X = scaler.transform(X)
```

2. Split the scaled data into 80% training data and 20% test data with train test split. Set random state = 42.

[8]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=True, random_state=42)
```

1.2.2 Exercise 1:

Perform the linear regression from scikit-learn to study the expense of TV, newspaper, and radio advertising on the sale

[9]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score, r2_score
```

1. Use the ordinary linear regression to train the model. What are the learned model parameters?

[10]:
```
model = LinearRegression()
model.fit(x_train, y_train)
a = model.coef_
b = model.intercept_

print("trained model parameters are:")
print(f"Model Coef: {a}")
print(f"Model intercept: {b}")
```

```
trained model parameters are:
Model Coef: [[13.22651832  9.38407469  0.3139387 ]]
Model intercept: [3.01120633]
```

2. Compare the R2 scores from training and test data, and discuss your observations?

[11]:
```
train_accuracy = model.score(x_train, y_train, sample_weight=None)
test_accuracy = model.score(x_test, y_test, sample_weight=None)

print(f'Accuracy for training data (R^2): {train_accuracy}')
print(f'Accuracy for testing data (R^2): {test_accuracy}')
```

```
Accuracy for training data (R^2): 0.8957008271017818
Accuracy for testing data (R^2): 0.899438024100912
```

1.2.3 Exercise 2:

Repeat the tasks in Exercise 1 with the KNN from scikit-learn

1. Choose the nearest neighbors K = 1. Use the KNeighborsRegressor to train the model.

[12]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[13]:
```
k = 1
neigh = KNeighborsRegressor(n_neighbors=k)
neigh.fit(x_train, y_train)
```

[13]:
```
KNeighborsRegressor(n_neighbors=1)
```

2. Compare the R2 scores from training and test data.

[14]:
```
ypred_train        = neigh.predict(x_train)
accuracy_train     = r2_score(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train)

ypred_test         = neigh.predict(x_test)
accuracy_test      = r2_score(y_test,ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
```

```
Accuracy for training data (R^2):  1.0
Accuracy for test data (R^2):  0.9025380308603167
```

3. Repeat your studies for K = 2, 3, . . . , 10. What are your observations?

[15]:
```
for k in range(2,10+1):
    neigh = KNeighborsRegressor(n_neighbors=k)
    neigh.fit(x_train, y_train)

    print(f"k: {k}")

    ypred_train        = neigh.predict(x_train)
    accuracy_train     = r2_score(y_train, ypred_train)
    print('Accuracy for training data (R^2): ', accuracy_train)

    ypred_test         = neigh.predict(x_test)
    accuracy_test      = r2_score(y_test,ypred_test)
    print('Accuracy for test data (R^2): ', accuracy_test)
    print()
```

```
k: 2
Accuracy for training data (R^2):  0.9812430719552749
Accuracy for test data (R^2):  0.9324894681867459

k: 3
Accuracy for training data (R^2):  0.9718620977717801
Accuracy for test data (R^2):  0.9364836092038614

k: 4
Accuracy for training data (R^2):  0.9671987204202612
Accuracy for test data (R^2):  0.9299073794472469

k: 5
Accuracy for training data (R^2):  0.9665527977251915
Accuracy for test data (R^2):  0.9337192077502264

k: 6
Accuracy for training data (R^2):  0.9632464215142899
Accuracy for test data (R^2):  0.9498780845328242

k: 7
Accuracy for training data (R^2):  0.9597421955914276
Accuracy for test data (R^2):  0.9503836147120786

k: 8
Accuracy for training data (R^2):  0.9570375379536363
Accuracy for test data (R^2):  0.9461417178612622

k: 9
```

```
Accuracy for training data (R^2):  0.9512721124742588
Accuracy for test data (R^2):  0.9420403901172719

k: 10
Accuracy for training data (R^2):  0.9477586389705528
Accuracy for test data (R^2):  0.9382764359650904
```

Best balance between training and testing accuracy seems to be K = 7

1.2.4 Exercise 3:

Repeat the tasks in Exercise 1 with neural networks.

1. Use Keras to build a neural network with input layer = output layer (i.e., only one layer), no activation function, choose mean square errors as the loss function.

[16]:
```
# how many layer and neurons are hyper parameters

p_model = keras.models.Sequential(
    [
        keras.layers.Dense(1, input_shape=(3,)),
    ]
)

# 'mse' for regression
# 'binary_crossentropy' for classification
p_model.compile(loss='mse',
                  optimizer='adam')

p_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 1)                 4

=================================================================
Total params: 4 (16.00 Byte)
Trainable params: 4 (16.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2. Use the training data to train the neural network with epochs = 5000. What are the weights and bias parameters?

[17]:
```
%%capture

EPOCHS=5_000
```

```
p_model.fit(x_train, y_train, epochs=EPOCHS) # epochs = iterations
```

[18]:
```
# Get the weights and biases of each layer
for layer in p_model.layers:
    weights, biases = layer.get_weights()
    print(f"Weights for {layer.name}: {weights}")
    print(f"Biases for {layer.name}: {biases}")
    print()
```

```
Weights for dense_6: [[13.143192  ]
 [ 9.259298  ]
 [ 0.44624153]]
Biases for dense_6: [3.0824132]
```

3. compare the R2 scores from training and test data.

[19]:
```
ypred_train        = p_model.predict(x_train)
accuracy_train     = r2_score(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')

ypred_test         = p_model.predict(x_test)
accuracy_test      = r2_score(y_test,ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
```

```
5/5 [==============================] - 0s 1ms/step
Accuracy for training data (R^2):  0.8956264807756238

2/2 [==============================] - 0s 3ms/step
Accuracy for test data (R^2):  0.8984392113011369
```

4. Compare the models from Exercise 1 and 3. What are your observations?

The model parameters are the pretty much the same. Given more epochs, they would probably be equivalent. They both have effectively the same accuracy as well.

1.2.5 Exercise 4:

Repeat the tasks in Exercise 1 with neural networks.

[20]:
```
import numpy as np
```

1. Use Keras to build a neural network with input layer (128 neurons), two hidden layers (64 and 32 neurons), and output layer (1 neuron). Set the activation function in input and hidden layers as 'ReLu'. Choose mean square errors as the loss function.

[21]:
```
# how many layer and neurons are hyper parameters

n_model = keras.models.Sequential(
```

```
    [
        keras.layers.Dense(128, activation='relu', input_shape=(3,)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(32, activation='relu'),
        keras.layers.Dense(1)

    ]
)

# 'mse' for regression
# 'binary_crossentropy' for classification
n_model.compile(loss='mse',
                  optimizer='adam')

n_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 128)               512

 dense_8 (Dense)             (None, 64)                8256

 dense_9 (Dense)             (None, 32)                2080

 dense_10 (Dense)            (None, 1)                 33

=================================================================
Total params: 10881 (42.50 KB)
Trainable params: 10881 (42.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2. Use the training data to train the neural network with epochs = 5000.

[22]:
```
%%capture

EPOCHS=5_000
n_model.fit(x_train, y_train, epochs=EPOCHS) # epochs = iterations
```

[23]:
```
# Get the weights and biases of each layer
print(f"There is a lot of weights and biases so I will only print their shape:␣
 ↪\n")
for layer in n_model.layers:
    weights, biases = layer.get_weights()
    print(f"Weights for {layer.name}: {np.shape(weights)}")
    print(f"Biases for {layer.name}: {np.shape(biases)}")
```

```
    print()
```

```
There is a lot of weights and biases so I will only print their shape:

Weights for dense_7: (3, 128)
Biases for dense_7: (128,)

Weights for dense_8: (128, 64)
Biases for dense_8: (64,)

Weights for dense_9: (64, 32)
Biases for dense_9: (32,)

Weights for dense_10: (32, 1)
Biases for dense_10: (1,)
```

3. Compare the R2 scores from training and test data

[24]:
```
ypred_train        = n_model.predict(x_train)
accuracy_train     = r2_score(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')

ypred_test         = n_model.predict(x_test)
accuracy_test      = r2_score(y_test,ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
```

```
5/5 [==============================] - 0s 2ms/step
Accuracy for training data (R^2):  0.9985499057259037

2/2 [==============================] - 0s 2ms/step
Accuracy for test data (R^2):  0.9912788515050607
```

4. Compare the models from Exercise 3 and 4. What are your observations?

The model in ex4 performs much better than the one from ex3 base on accuracy of training and testing models

1.3 2. Classification

1.3.1 Set up

Consider the data “diagnosis.csv”.

[25]:
```
from sklearn.preprocessing import LabelEncoder
```

1. Take the feature as “radius mean”, “texture mean”, “smoothness mean”.

[26]:
```
df = pd.read_csv("./diagnosis.csv")
X = df[["radius_mean", "texture_mean", "smoothness_mean"]]
y = df["diagnosis"]
labelEncoder = LabelEncoder()
y_encoded = labelEncoder.fit_transform(y)
```

2. Scale the features using MinMaxScaler

[27]:
```
scaler = MinMaxScaler().fit(X)
X = scaler.transform(X)
```

3. Split the scaled data into 80% training data and 20% test data with train test split. Set random state = 42

[28]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.
↪2, shuffle=True, random_state=42)
```

1.3.2 Exercise 5:

Perform the logistic regression from scikit-learn for the selected features.

[29]:
```
from sklearn.linear_model import LogisticRegression
```

1. Use the logistic regression to train the model. What are the learned model parameters?

[30]:
```
model = LogisticRegression()
model.fit(x_train, y_train)
a = model.coef_
b = model.intercept_

print("trained model parameters are:")
print(f"Model Coef: {a}")
print(f"Model intercept: {b}")
```

```
trained model parameters are:
Model Coef: [[7.34067597 3.30984355 3.77468867]]
Model intercept: [-5.65721378]
```

2. Compare the accuracy from the training and test data, and discuss your observations.

[31]:
```
ypred_train        = model.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = model.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
```

```
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.9120879120879121

Confusion matrix for training data:
 [[281   5]
 [ 35 134]]

Accuracy for test data (R^2):  0.9385964912280702

Confusion matrix for test data:
 [[71  0]
 [ 7 36]]
```

1.3.3 Exercise 6:

Repeat the tasks in Exercise 5 with the KNN from scikit-learn

[32]:
```
from sklearn.neighbors import KNeighborsClassifier
```

1. Choose the nearest neighbors K = 1. Use the KNeighborsClassifier to train the model
2. Compare the accuracy from training and test data

[33]:
```
k = 1
neigh = KNeighborsClassifier(n_neighbors=k)
neigh.fit(x_train, y_train)

ypred_train        = neigh.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train)
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = neigh.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  1.0
Confusion matrix for training data:
 [[286   0]
 [  0 169]]

Accuracy for test data (R^2):  0.9298245614035088
```

```
Confusion matrix for test data:
 [[65  6]
 [ 2 41]]
```

3. Repeat your studies for K = 3, 5, . . . , 9. What are your observations?

[34]:
```
for k in range(3,10+1,2):
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(x_train, y_train)

    print(f"k: {k}")

    ypred_train        = neigh.predict(x_train)
    accuracy_train     = accuracy_score(y_train, ypred_train)
    conf_matrix_train  = confusion_matrix(y_train, ypred_train)
    print('Accuracy for training data (R^2): ', accuracy_train)
    print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

    ypred_test         = neigh.predict(x_test)
    accuracy_test      = accuracy_score(y_test,ypred_test)
    conf_matrix_test   = confusion_matrix(y_test, ypred_test)
    print('Accuracy for test data (R^2): ', accuracy_test, '\n')
    print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
    print()
```

```
k: 3
Accuracy for training data (R^2):  0.9384615384615385
Confusion matrix for training data:
 [[276  10]
 [ 18 151]]

Accuracy for test data (R^2):  0.9035087719298246

Confusion matrix for test data:
 [[66  5]
 [ 6 37]]


k: 5
Accuracy for training data (R^2):  0.9274725274725275
Confusion matrix for training data:
 [[275  11]
 [ 22 147]]

Accuracy for test data (R^2):  0.9298245614035088

Confusion matrix for test data:
```

```
 [[68  3]
 [ 5 38]]


k: 7
Accuracy for training data (R^2):  0.9274725274725275
Confusion matrix for training data:
 [[275  11]
 [ 22 147]]

Accuracy for test data (R^2):  0.9473684210526315

Confusion matrix for test data:
 [[69  2]
 [ 4 39]]


k: 9
Accuracy for training data (R^2):  0.9296703296703297
Confusion matrix for training data:
 [[274  12]
 [ 20 149]]

Accuracy for test data (R^2):  0.956140350877193

Confusion matrix for test data:
 [[69  2]
 [ 3 40]]
```

k=9 seems to yield the best testing data accuracy. K is odd numbers because we need a tie breaker.

1.3.4 Exercise 7:

Repeat the tasks in Exercise 5 with neural networks

1. Use Keras to build a neural network with input layer = output layer (i.e., only one layer), sigmoid activation function, choose binary cross entropy as the loss function.

[35]:
```
# how many layer and neurons are hyper parameters

s_model = keras.models.Sequential(
    [
        keras.layers.Dense(1, activation='sigmoid', input_shape=(3,)),
    ]
)

# 'mse' for regression
```

```
# 'binary_crossentropy' for classification
s_model.compile(loss='binary_crossentropy',
                  optimizer='adam')

s_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 1)                 4

=================================================================
Total params: 4 (16.00 Byte)
Trainable params: 4 (16.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2. Use the training data to train the neural network with epochs = 5000. What are the weights and bias parameters

[36]:
```
%%capture

EPOCHS=5_000
s_model.fit(x_train, y_train, epochs=EPOCHS) # epochs = iterations
```

[37]:
```
# Get the weights and biases of each layer
for layer in s_model.layers:
    weights, biases = layer.get_weights()
    print(f"Weights for {layer.name}: {weights}")
    print(f"Biases for {layer.name}: {biases}")
    print()
```

```
Weights for dense_11: [[18.00087  ]
 [ 6.7832255]
 [ 9.31838  ]]
Biases for dense_11: [-12.649803]
```

3. Compare the accuracy from training and test data

[38]:
```
threshold = 0.5

ypred_train        = s_model.predict(x_train)
ypred_train         = (ypred_train >= threshold).astype(int)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print(f'Accuracy for training data (R^2);  {threshold} threshold:␣
 ↪{accuracy_train} \n')
```

```
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')


ypred_test         = s_model.predict(x_test)
ypred_test         = (ypred_test >= threshold).astype(int)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print(f'Accuracy for test data (R^2); {threshold} threshold: {accuracy_test}␣
 ↪\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
WARNING:tensorflow:5 out of the last 15 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f4384201620>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
15/15 [==============================] - 0s 1ms/step
Accuracy for training data (R^2);  0.5 threshold: 0.9230769230769231

Confusion matrix for training data:
 [[273  13]
 [ 22 147]]

4/4 [==============================] - 0s 2ms/step
Accuracy for test data (R^2); 0.5 threshold: 0.9385964912280702

Confusion matrix for test data:
 [[68  3]
 [ 4 39]]
```

4. Compare the models parameters from Exercise 7 and 5. Explain your observations.

The weights and bias look very different but the accuracy for both training and testing is effectively the same.

5. Repeat Tasks 2–4 by training the neural network with epochs = 12000. Compare your results with those observed with epochs = 5000. What are your observations?

[39]:
```
%%capture

EPOCHS=12_000
s_model.fit(x_train, y_train, epochs=EPOCHS) # epochs = iterations
```

[40]:
```
# Get the weights and biases of each layer
for layer in s_model.layers:
    weights, biases = layer.get_weights()
    print(f"Weights for {layer.name}: {weights}")
    print(f"Biases for {layer.name}: {biases}")
    print()
```

```
Weights for dense_11: [[24.523485]
 [ 9.577901]
 [13.666807]]
Biases for dense_11: [-17.632284]
```

[41]:
```
threshold = 0.5

ypred_train        = s_model.predict(x_train)
ypred_train         = (ypred_train >= threshold).astype(int)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print(f'Accuracy for training data (R^2);  {threshold} threshold:␣
 ↪{accuracy_train} \n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')


ypred_test         = s_model.predict(x_test)
ypred_test         = (ypred_test >= threshold).astype(int)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print(f'Accuracy for test data (R^2); {threshold} threshold: {accuracy_test}␣
 ↪\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
15/15 [==============================] - 0s 885us/step
Accuracy for training data (R^2);  0.5 threshold: 0.9274725274725275

Confusion matrix for training data:
 [[273  13]
 [ 20 149]]

4/4 [==============================] - 0s 1ms/step
Accuracy for test data (R^2); 0.5 threshold: 0.9473684210526315

Confusion matrix for test data:
 [[68  3]
 [ 3 40]]
```

Small addvantage in accuracy and very different weights and bias

1.3.5 Exercise 8:

Repeat the tasks in Exercise 5 with neural networks

1. Use Keras to build a neural network with input layer (128 neurons), two hidden layers (64 and 32 neurons), and output layer (1 neuron). Set the activation function in input and hidden layers as 'ReLu'. Choose binary cross entropy as the loss function

[42]:
```
# how many layer and neurons are hyper parameters

ms_model = keras.models.Sequential(
    [
        keras.layers.Dense(128, activation='relu', input_shape=(3,)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(32, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid'),
    ]
)

# 'mse' for regression
# 'binary_crossentropy' for classification
ms_model.compile(loss='binary_crossentropy',
                 optimizer='adam')

ms_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 128)               512

 dense_13 (Dense)            (None, 64)                8256

 dense_14 (Dense)            (None, 32)                2080

 dense_15 (Dense)            (None, 1)                 33

=================================================================
Total params: 10881 (42.50 KB)
Trainable params: 10881 (42.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2. Use the training data to train the neural network with epochs = 5000

[43]:
```
%%capture

EPOCHS=5_000
```

```
ms_model.fit(x_train, y_train, epochs=EPOCHS) # epochs = iterations
```

[44]:
```
# Get the weights and biases of each layer
for layer in ms_model.layers:
    weights, biases = layer.get_weights()
    print(f"Weights for {layer.name}: {np.shape(weights)}")
    print(f"Biases for {layer.name}: {np.shape(biases)}")
    print()
```

```
Weights for dense_12: (3, 128)
Biases for dense_12: (128,)

Weights for dense_13: (128, 64)
Biases for dense_13: (64,)

Weights for dense_14: (64, 32)
Biases for dense_14: (32,)

Weights for dense_15: (32, 1)
Biases for dense_15: (1,)
```

3. Compare the accuracy from training and test data.

[50]:
```
threshold = 0.5

ypred_train        = ms_model.predict(x_train)
ypred_train        = (ypred_train >= threshold).astype(int)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print(f'Accuracy for training data (R^2) @ {threshold} threshold:␣
 ↪{accuracy_train} \n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = ms_model.predict(x_test)
ypred_test     = (ypred_test >= threshold).astype(int)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print(f'Accuracy for test data (R^2) @ {threshold} threshold: {accuracy_test}␣
 ↪\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
15/15 [==============================] - 0s 1ms/step
Accuracy for training data (R^2) @ 0.5 threshold: 1.0

Confusion matrix for training data:
 [[286   0]
 [  0 169]]
```

```
4/4 [==============================] - 0s 1ms/step
Accuracy for test data (R^2) @ 0.5 threshold: 0.9385964912280702

Confusion matrix for test data:
 [[68  3]
 [ 4 39]]
```

[ ]:

4. Compare the models from Exercise 7 and 8. What are your observations?

This one seems to be overfitted as the training data achieved perfect accuracy

[ ]: