UNIVERSITY COLLEGE LONDON
DEPARTMENT OF SPACE AND CLIMATE PHYSICS

Candidate Code: HYXC3

Programme Title: MSc Scientific Computing

Module Code: SPCE0038

Module Title: Machine Learning with Big Data

End Assessment

In submitting this coursework, I assert that the work presented is entirely my own except where properly marked and cited.

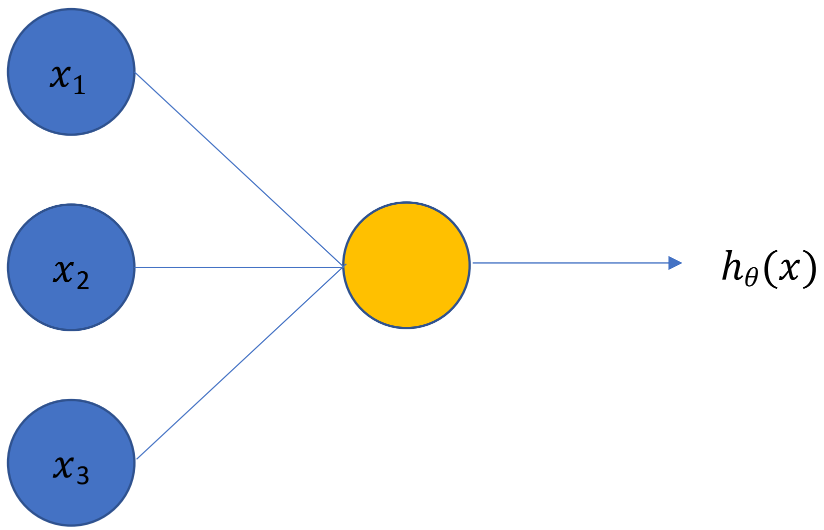
Date of Submission:	dd/mm/yy

Question 1

1(a)

Referring to the diagram of the Basic Logistic Unit on the following page:

Question 1(a) – Basic Logistic Unit



Weighted Sum: $z = \sum_{j=1}^{n} \theta_j x_j = \theta^T x$

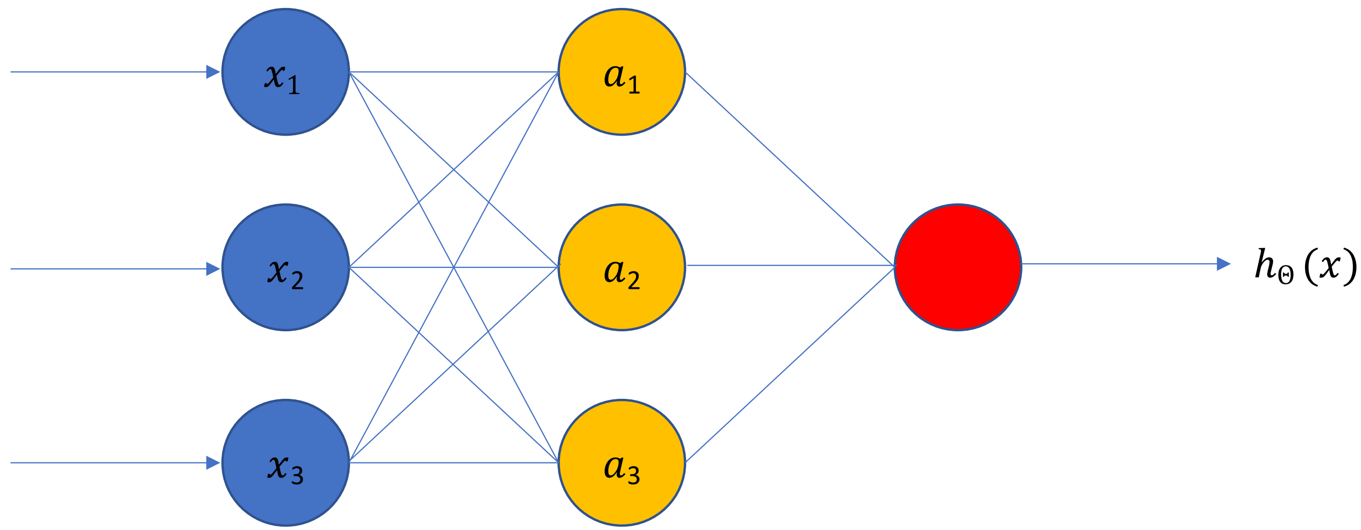
Activations: $a = h(z)$ non-linear activation function h

1(b)

TODO

1(c)

Question 1(c) – Fully Connected, Feed Forward, Artificial Neural Network



Input Layer Logistic Units

Hidden Layer Logistic Units

Output Node

Weighted Sums: $z_j = \sum_{i=1}^{n} \theta_{ij} x_i$

Activations: $a_j = h(z_j)$

1(d)

TODO

1(e)

TODO

1(f)

Artificial Neural Networks (ANNs) are described as *shallow* or *deep*, and *wide* or *narrow*. *shallow* or *deep* refers to the number of layers in the network, and *wide* or *narrow* refers to the number of nodes in each layer.

The *credit assignment path*, or CAP, of a neural network is a measure of the number of data transformations that occur as data passes through the network. For *feed-forward* networks the CAP is the number of *hidden layers* plus one.

A *deep* neural network is generally considered to be a network with multiple layers and a CAP > 2.

1(g)

TODO

Question 2

2(a)

TODO

2(b)

TODO

2(c)

TODO

2(d)

TODO

2(e)

TODO

2(f)

TODO

2(g)

TODO

2(h)

TODO

Question 3

3(a)

TODO

3(b)

TODO

3(c)

TODO

3(d)

TODO

3(e)

TODO

3(f)

TODO

Question 4

4(a)

TODO

4(b)

TODO

4(c)

TODO

4(d)

TODO

4(e)

TODO

4(f)

TODO

question_4f

May 7, 2020

[ ]:
```python
# Fetch batch function:

def fetch_batch(epoch, batch_index, batch_size):

    return X_batch, y_batch


# Set up computational graph:

import tensorflow as tf
reset_graph ()

n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing_data_target, dtype=tf.float32, name="y")

theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0), name="theta")
y_pred = tf .matmul(X, theta , name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
training_op = optimizer.minimize(mse)


# Execute:

init = tf.global_variables_initializer()

with
tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE=", mse.eval()) sess.run(training_op)
    best_theta = theta.eval()
```

```python
# Fetch batch function:

def fetch_batch(epoch, batch_index, batch_size):
    return X_batch, y_batch


# Set up computational graph:

import tensorflow as tf
reset_graph ()

n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing_data_target, dtype=tf.float32, name="y")

theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0), name="theta")
y_pred = tf .matmul(X, theta , name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
training_op = optimizer.minimize(mse)


# Execute:

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE=", mse.eval())
            sess.run(training_op)
    best_theta = theta.eval()
```

Listing 1: Question 4f

Question 5

5(a)

TODO

5(b)

TODO

5(c)

TODO

5(d)

TODO

5(e)

TODO

5(f)

TODO