LAB 7

Name: Yeo Zheng Xu Isaac

In [55]:
```
import tensorflow.compat.v1 as tf
import pandas as pd
import numpy as np
from IPython.display import display
```

Load Dataset from local file

In [56]:
```
dataset = pd.read_csv('C:/Users/Isaac Yeo/Downloads/abalone.data',sep=',',head
er= None)
print("shape of dataset",dataset.shape)
```

```
shape of dataset (4177, 9)
```

Prepare data by adding Column names and encoded

In [57]:
```
column_names=['Sex','Length','Dismeer','Height','Whole weight','Shucked weigh
t',
              'Viscera weight','Shell weight','Rings']
dataset.columns=column_names
dataset.head()
```

Out[57]:

	Sex	Length	Dismeer	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

In [58]:
```
encoded_dataset = pd.get_dummies(dataset,columns=['Sex'])
encoded_dataset.head()
```

Out[58]:

	Length	Dismeer	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	0	0	1
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0	0	1
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1	0	0
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	0	0	1
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	0	1	0

Split train and test set

7 train set and 3 test set data

In [59]:
```
#Split data into 7/10 train set and  3/10 test set
train_set=encoded_dataset.sample(frac=0.7,random_state=49)
test_set=encoded_dataset.drop(train_set.index)
```

In [60]:
```
#Target variable in Ring column
#Generate train data and train labels
x_train = train_set.drop('Rings',axis=1)
x_train.reset_index(inplace=True,drop=True)
y_train = train_set['Rings']

#Generate test data and test labels
x_test = test_set.drop('Rings',axis=1)
x_train.reset_index(inplace=True,drop=True)
y_test = test_set['Rings']

#Set correct dimension tensorflow
y_train = np.expand_dims(y_train,axis=1)
y_test = np.expand_dims(y_test,axis=1)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(2924, 10)
(2924, 1)
(1253, 10)
(1253, 1)
```

Create function for batches

In [85]:
```
def generate_batches(x,y,batch_size=64):
    #shuffle first
    x =x.sample(frac=1)
    y =y[x.index]
    num_batches = len(x) // batch_size
    batches=[]
    for i in range(num_batches):
        start_index = batch_size * i
        end_index = start_index + batch_size
        batches.append((
            x.iloc[start_index: end_index, :],
            y[start_index:end_index]
        ))

    #leftover data
    if(num_batches * batch_size <len(x)):
        start_index = num_batches * batch_size
        batches.append((
            x.iloc[start_index:,:],
            y[start_index:]
        ))
    return batches
```

Tensorflow implementation

In [86]:
```
n_features = encoded_dataset.shape[1] - 1
n_neurons_1st = 64 # number of neurons in frist hidden layer
n_neurons_2nd = 64 # number of neurons in second hideen layer
n_neurons_out = 1  # output layer has 1 neuron (regression problem)
tf.disable_eager_execution()

tf.reset_default_graph()
#placeholders,x=data y_true = target
x = tf.placeholder(tf.float32,shape=[None,n_features],name="x")
y_true =tf.placeholder(tf.float32,shape=[None,n_neurons_out],name="y")

#variables
#first hideen layer weights and bias
w_1st = tf.get_variable("weights_1st",
                        dtype=tf.float32,
                        shape=[n_features,n_neurons_1st],
                        initializer=tf.glorot_uniform_initializer())

b_1st = tf.get_variable("bias_1st",
                        dtype=tf.float32,
                        shape=[1,n_neurons_1st],
                        initializer= tf.zeros_initializer())

w_2nd = tf.get_variable("weights_2nd",
                        dtype=tf.float32,
                        shape=[n_neurons_1st,n_neurons_2nd],
                        initializer=tf.glorot_uniform_initializer())

b_2nd = tf.get_variable("bias_2nd",
                        dtype=tf.float32,
                        shape=[1,n_neurons_2nd],
                        initializer= tf.zeros_initializer())

w_out = tf.get_variable("weights_out",
                        dtype=tf.float32,
                        shape=[n_neurons_2nd,n_neurons_out],
                        initializer=tf.glorot_uniform_initializer())

b_out = tf.get_variable("bias_out",
                        dtype=tf.float32,
                        shape=[1,n_neurons_out],
                        initializer= tf.zeros_initializer())

#compute output of each layer
x_1st = tf.matmul(x,w_1st)+ b_1st
x_1st = tf.nn.relu(x_1st)#output 1st layer

x_2nd = tf.matmul(x_1st,w_2nd)+ b_2nd
x_2nd = tf.nn.relu(x_2nd)#output 2md layer

x_out = tf.matmul(x_2nd,w_out)+ b_out #final layer output
y_hat = x_out

#L2 regularization term
l2_lambda = 0.001 #lambda coefficient
l2_reg = (l2_lambda * tf.nn.l2_loss(w_1st)+l2_lambda * tf.nn.l2_loss(w_2nd)
```

```
          + l2_lambda * tf.nn.l2_loss(w_out))

#optimization(MSE) with l2 regualarization
cost = tf.losses.mean_squared_error(y_true,y_hat)+ l2_reg
learning_rate =0.01 #learning rate
optimizer = tf.train.GradientDescentOptimizer(learning_rate)#gradient descent
 optimizer
training_op = optimizer.minimize(cost)
```

In [87]:
```
# execution the model
init =tf.global_variables_initializer()
n_epochs =500
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        batches = generate_batches(x_train, y_train, batch_size = 64)
        for batch in batches:
            x_train_batch = batch[0]
            y_train_batch = batch[1]
            sess.run(training_op,feed_dict={x: x_train_batch, y_true:y_train_b
atch})
        #print training cost every 10 epochs
        if epoch %10 ==0:
            training_cost = sess.run(cost,feed_dict={x:x_train,y_true:y_train
})
            print("Epoch",epoch,"training cost:",training_cost)
    print("training MSE:",sess.run(cost,feed_dict={x:x_train,y_true:y_train}))
    test_cost = sess.run(cost,feed_dict={x:x_test,y_true:y_test})
    print("Test MSE", test_cost)
```

```
Epoch 0 training cost: 7.0647607
Epoch 10 training cost: 7.6836667
Epoch 20 training cost: 5.336216
Epoch 30 training cost: 5.0658083
Epoch 40 training cost: 5.074678
Epoch 50 training cost: 5.1944776
Epoch 60 training cost: 7.1789274
Epoch 70 training cost: 4.640029
Epoch 80 training cost: 4.7236104
Epoch 90 training cost: 5.561826
Epoch 100 training cost: 4.494275
Epoch 110 training cost: 4.6226497
Epoch 120 training cost: 4.4736934
Epoch 130 training cost: 4.4664564
Epoch 140 training cost: 4.47509
Epoch 150 training cost: 4.4797983
Epoch 160 training cost: 4.6541133
Epoch 170 training cost: 4.5808387
Epoch 180 training cost: 4.6826215
Epoch 190 training cost: 4.434239
Epoch 200 training cost: 5.861637
Epoch 210 training cost: 4.4010854
Epoch 220 training cost: 4.3763075
Epoch 230 training cost: 4.4984837
Epoch 240 training cost: 4.4400992
Epoch 250 training cost: 4.5635295
Epoch 260 training cost: 4.6314626
Epoch 270 training cost: 4.7157216
Epoch 280 training cost: 4.392329
Epoch 290 training cost: 4.365957
Epoch 300 training cost: 4.3747897
Epoch 310 training cost: 4.354889
Epoch 320 training cost: 4.747701
Epoch 330 training cost: 4.427187
Epoch 340 training cost: 4.355788
Epoch 350 training cost: 4.9809375
Epoch 360 training cost: 4.4220366
Epoch 370 training cost: 4.3739285
Epoch 380 training cost: 4.3673058
Epoch 390 training cost: 4.946102
Epoch 400 training cost: 4.4780426
Epoch 410 training cost: 4.373232
Epoch 420 training cost: 4.4390693
Epoch 430 training cost: 4.303337
Epoch 440 training cost: 4.3100567
Epoch 450 training cost: 4.397085
Epoch 460 training cost: 4.361586
Epoch 470 training cost: 4.688975
Epoch 480 training cost: 5.390245
Epoch 490 training cost: 4.667224
training MSE: 4.476651
Test MSE 4.301572
```

Architecture

- First Layer (input layer) have 10 neurons which is equal to the number of features. Train set+ Test set = 10.

- Hidden Layer 1 have 64 neurons. Relu(Rectified Linear Unit) activation function is use on its output.

- Hidden Layer 2 have 64 neurons. Relu(Rectified Linear Unit) activation function is use on its output.

- Output Layer have 1 neuron as it is regression problem. No activation function use

- Regularization we apply L2 regularization for both hidden layer and output layer. Coefficient is 0.001. (Task Requirement 8)

- Learning rate = 0.01

- Optimizer =GradientDescentOptimizer (Task Requirement 2)

- Cost Function is mean square error. (Task requirement 7)

- Epoch = 500

*** The best MSE on test set achived is around 4.++. There is no overfitting too because the train MSE and test MSE is around the same value**

Keras Implementation

- We will use Keras as an altemative implementation here. The same architecture and parameters used in Tensorflow implementation will be used here

- We use the same parameters so that we can compare with TensorFlow implementation.

In [92]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers,regularizers
```

In [93]:
```
model = Sequential()
n_nodes =64 #number of nodes in each hidden layer
l2_lambda = 0.001 #l2 regularization coefficient

model.add(Dense(n_nodes,activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda),input_dim = n_features))
model.add(Dense(n_nodes,activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda)))
model.add(Dense(1,kernel_regularizer=regularizers.l2(l2_lambda)))

n_epochs = 500 #number  of epochs
learning_rate = 0.01 # learning rate
batch_size =64 #batch size

model.compile(optimizer=tf.train.GradientDescentOptimizer(learning_rate),loss=
'mse')
history = model.fit(x_train,y_train,
                   batch_size=batch_size,
                   epochs=n_epochs,
                   validation_split = 0.2,
                   verbose=False)
score = model.evaluate(x_test,y_test,batch_size=batch_size)
```

```
1253/1253 [==============================] - 0s 68us/sample - loss: 4.2422
```

In [94]:
```
print("keras train MSE:",model.evaluate(x_train,y_train,batch_size=128))
print("keras test MSE:",score)
```

```
2924/2924 [==============================] - 0s 52us/sample - loss: 4.4189
keras train MSE: 4.4189409890024836
keras test MSE: 4.242174203930525
```

The result with Keras implementation is quite similar with Tensorflow implementation result MSE is around 4.++

In [ ]: