# Deep Learning

# Naval Mine Detector Application

In this use case, they have been provided with a SONAR data set which contains the data about 208 patterns obtained by bouncing sonar signals off a metal cylinder (naval mine) and a rock at various angles and under various conditions.
Now, as we know, a naval mine is a self-contained explosive device placed in water to damage or destroy surface ships or submarines.
So, our goal is to build a model that can predict whether the object is a naval mine or rock based on our data set.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

#########################################

# Reading the dataset
def read_dataset():
    df = pd.read_csv("sonar.csv")
    print("Marvellous Infosystems : Dataset loaded succesfully")
    print("Marvellous Infosystems : Number of columns: ",len(df.columns))

    #Fetures of dataset
    X = df[df.columns[0:60]].values

    #Label of dataset
    y = df[df.columns[60]]

    # Encode the dependant variable
    encoder = LabelEncoder()

    # Encode character labeles into integer ie 1 or 0 (One hot encode)
    encoder.fit(y)
    y = encoder.transform(y)
    Y = one_hot_encode(y)

    print("Marvellous Infosystems : X.shape",X.shape)

    return (X,Y)

#########################################

# Define the encoder function to set M => 1, R => 0
def one_hot_encode(labels):
    n_labels = len(labels)
    n_unique_labels = len(np.unique(labels))
    one_hot_encode = np.zeros((n_labels,n_unique_labels))
    one_hot_encode[np.arange(n_labels),labels] = 1
    return one_hot_encode

#########################################

# Model for training
def multilayer_perceptron(x, weights, biases):

    # Hidden layer with RELU activations
```

```
    # First layer performs matrix multiplication of input with weights
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)

    # Hidden layer with sigmoid activations
    # Second layer performs matrix multiplication of layer1 with weights
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.sigmoid(layer_2)

    # Hidden layer with sigmoid activations
    # Third layer performs matrix multiplication of layer2 with weights
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    layer_3 = tf.nn.sigmoid(layer_3)

    # Hidden layer with RELU activations
    # Forth layer performs matrix multiplication of layer3 with weights
    layer_4 = tf.add(tf.matmul(layer_3, weights['h4']), biases['b4'])
    layer_4 = tf.nn.relu(layer_4)

    # Output layer with linear activations
    out_layer = tf.matmul(layer_4, weights['out']) + biases['out']
    return out_layer

##########################################

def main():
    # Read the dataset
    X, Y = read_dataset()

    # Shuffle the dataset to mix up the rows
    X, Y = shuffle(X, Y, random_state=1)

    # Convert the dataset into train and test datasets
    # For  testing we use 10% and for training we use 90%
    train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=0.30, random_state=415)

    # Inspect the shape of the train and test datasets
    print("Marvellous Infosystems : train_x.shape",train_x.shape)
    print("Marvellous Infosystems : train_y.shape",train_y.shape)
    print("Marvellous Infosystems : test_x.shape",test_x.shape)
    print("Marvellous Infosystems : test_y.shape",test_y.shape)

    # Define the parameters which are required for rensors ie hyperparameters

    # Change in a variable in each iteration
    learning_rate = 0.3

    # Total number of iterations to minimize the error
    training_epochs = 1000

```

```
cost_history = np.empty(shape=[1], dtype=float)

# Number of features <=> number of columns
n_dim = X.shape[1]
print("Number of columens are n_dim",n_dim)

# As we have tow classess as R and M
n_class = 2

# Path which contains model files
model_path = "Marvellous"

# Define the number of hidden layers an the
# number of neurons for each layer

# Number of hidden layers are 4
# NNeurons in each layer are 60
n_hidden_1 = 60
n_hidden_2 = 60
n_hidden_3 = 60
n_hidden_4 = 60

# Placeholder to store Inputs
x = tf.compat.v1.placeholder(tf.float32,[None, n_dim])

# Placeholder to store Outputs
y_ = tf.compat.v1.placeholder(tf.float32,[None, n_class])

# Model parameters
W = tf.Variable(tf.zeros([n_dim,n_class]))
b = tf.Variable(tf.zeros([n_class]))

# define the weights and the biases for each layer
# Create variable which contens random values
weights = {
    'h1': tf.Variable(tf.random.truncated_normal([n_dim, n_hidden_1])),
    'h2': tf.Variable(tf.random.truncated_normal([n_hidden_1, n_hidden_2])),
    'h3': tf.Variable(tf.random.truncated_normal([n_hidden_2, n_hidden_3])),
    'h4': tf.Variable(tf.random.truncated_normal([n_hidden_3, n_hidden_4])),
    'out': tf.Variable(tf.random.truncated_normal([n_hidden_4, n_class])    ),
    }

# Create variable which contens random values
biases = {
    'b1': tf.Variable(tf.random.truncated_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random.truncated_normal([n_hidden_2])),
    'b3': tf.Variable(tf.random.truncated_normal([n_hidden_3])),
    'b4': tf.Variable(tf.random.truncated_normal([n_hidden_4])),
    'out': tf.Variable(tf.random.truncated_normal([n_class])),
    }
```

```
        cost = sess.run(cost_function,feed_dict={x:train_x, y_:train_y})
        cost_history = np.append(cost_history, cost)
        correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        pred_y = sess.run(y,feed_dict={x:test_x} )
        mse = tf.reduce_mean(tf.square(pred_y - test_y))
        mse_ = sess.run(mse)
        accuracy = (sess.run(accuracy,feed_dict={x:train_x, y_:train_y}))
        accuracy_history.append(accuracy)
        print('epoch: ', epoch,' - ', 'cost: ', cost, " - MSE: ", mse_, "- Train Accuracy: ", accuracy)

    # Model gets saved in the file
    save_path = saver.save(sess, model_path)
    print("Model saved in file: %s", save_path)

    # Display graph for accuracy history
    plt.plot(accuracy_history)
    plt.title("Marvellous Infosystems : Accuracy History")
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.show()

    # Display graph for Loss calculation
    plt.plot(range(len(cost_history)),cost_history)
    plt.title("Marvellous Infosystems : Loss calculation")
    plt.axis([0,training_epochs,0,np.max(cost_history)/100])
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

    # Print the final mean square error
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.square(pred_y - test_y))
    print("Test Accuracy: ", (sess.run(y, feed_dict={x:test_x, y_:test_y} )))

    # Print the final mean square error
    pred_y = sess.run(y, feed_dict={x:test_x})
    mse = tf.reduce_mean(tf.square(pred_y- test_y))
    print("Mean Square Error : %.4f" % sess.run(mse))

if __name__ == "__main__":
    main()
```