```
from google.colab import drive
drive.mount('/content/drive')

import os
# start your code here
os.chdir('/content/drive/MyDrive/DL/Homework2') #make the directory where this ipynb file is
# end your code here
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

1 - Packages

Let's first import all the packages that you will need during this assignment.

- numpy is the fundamental package for scientific computing with Python.
- sklearn provides simple and efficient tools for data mining and data analysis.
- matplotlib is a library for plotting graphs in Python.
- tensorflow is a library for deep learning.
- testCases provides some test examples to assess the correctness of your functions
- planar_utils provide various useful functions used in this assignment

```
# Package imports
import numpy as np
import matplotlib.pyplot as plt
from testCases_v2 import *
from planar_utils import load_planar_dataset
import tensorflow as tf

%matplotlib inline
```

2 - Dataset

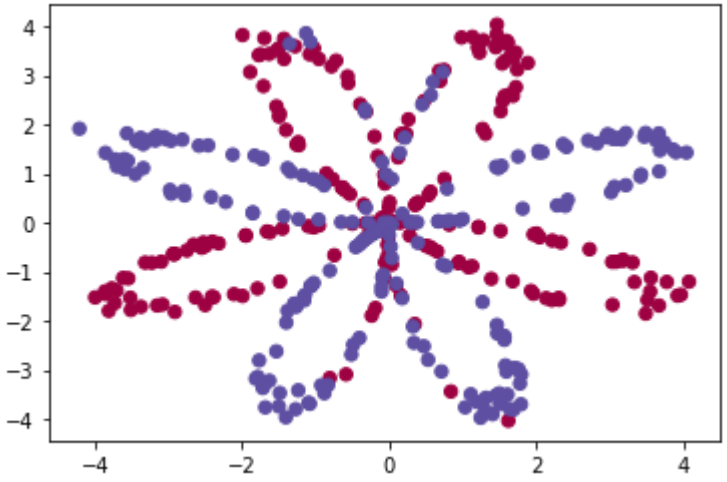
First, let's get the dataset you will work on. The following code will load a "flower" 2-class dataset into variables `X` and `Y`.

```
X, Y = load_planar_dataset()
X=X.T
Y=Y.T
```

Visualize the dataset using matplotlib. The data looks like a "flower" with some red (label y=0) and some blue (y=1) points. Your goal is to build a model to fit this data. In other words, we want the

classifier to define regions as either red or blue.

```
# Visualize the data:
plt.scatter(X[:,0], X[:,1], c=Y, s=40, cmap=plt.cm.Spectral);
```



You have: - a numpy-array (matrix) X that contains your features (x1, x2) - a numpy-array (vector) Y that contains your labels (red:0, blue:1).

Lets first get a better sense of what our data is like.

Exercise: How many training examples do you have? In addition, what is the `shape` of the variables `X` and `Y`?

Hint: How do you get the shape of a numpy array? (help)

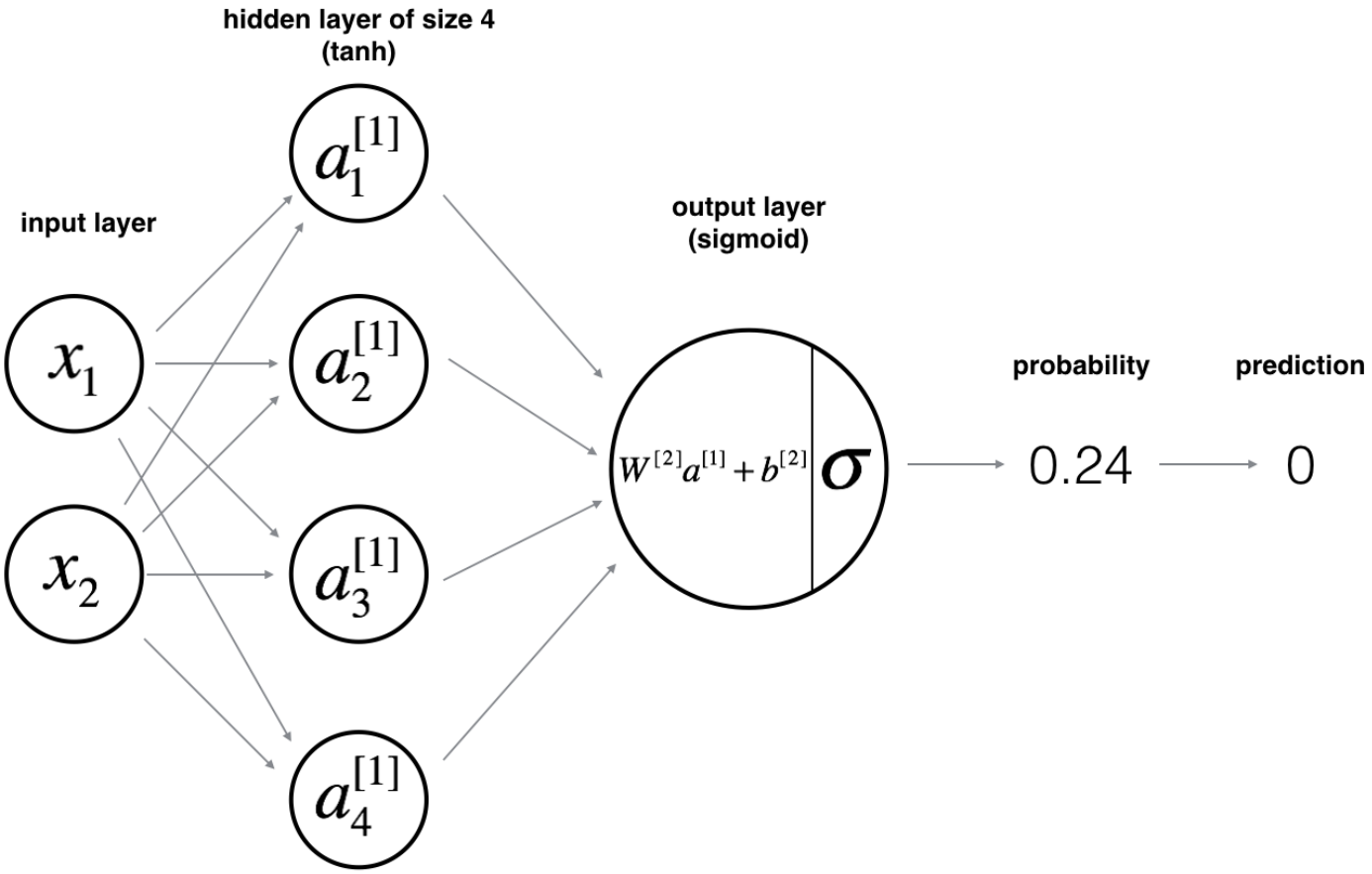
```
### START CODE HERE ### (≈ 3 lines of code)
shape_X = X.shape
shape_Y = Y.shape
m = X.shape[1]  # training set size
### END CODE HERE ###

print ('The shape of X is: ' + str(shape_X))
print ('The shape of Y is: ' + str(shape_Y))
print ('I have m = %d training examples!' % (m))
```

```
The shape of X is: (400, 2)
The shape of Y is: (400, 1)
I have m = 2 training examples!
```

3 - Create Your First Neural Network model with TensorFlow

Assume we are going to build a NN as follows.



We are going to introduce two popular ways to build it

3.1 - Sequential model

```
model_seq = tf.keras.Sequential([
    tf.keras.Input(shape=2), # Input layer: every sample (point) has two inputs
    tf.keras.layers.Dense(4, activation='tanh'), # Hidden layer 1: this layer has 4 neurons, and eve
    tf.keras.layers.Dense(1,activation='sigmoid'), # output layer: this layer has 1 neuron, and the
])

model_seq.summary()
```

Hint:

1. How do you build a sequential model? (help)
2. What are the layers in a NN model? (help)
3. What is the Input layer in a NN model? (help)
4. What is the Dense layer in a NN model? (help)
5. Your choices of activation functions. (help)

3.2 - API model

```
model_API_input = tf.keras.Input(shape=(2)) # Input_layer: every input has shape 2
x = tf.keras.layers.Dense(4, activation='tanh')(model_API_input) # Hidden layer 1: take layer 'model_
model_API_output = tf.keras.layers.Dense(1,activation='sigmoid')(x) # output layer: take layer 'x' a

model_API = tf.keras.Model(model_API_input, model_API_output, name="model_API")
model_API.summary()
```

Hint:

1. How do you build a API model? (help)

Exercise: Build your own NN model with either method.

You can build as many layers as you like.

For every layer, please feel free to choose the number of neurons.

Just to make sure that the shape of the input layer matches the size of each sample, and the number of neurons in the output layer matches the size of the output of each sample.

Please feel free to use 'your_model_name.summary()' to print out the model.

```
### START CODE HERE ###
model = tf.keras.Sequential([
    tf.keras.Input(shape=2),
    tf.keras.layers.Dense(4, activation='tanh'),
    tf.keras.layers.Dense(1,activation='sigmoid'),
])

### End CODE HERE ###
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 4)                 12

 dense_9 (Dense)             (None, 1)                 5

=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
```

4 - Train Your First Neural Network Model

4.1 - Compile your model

1. how we define our loss?
2. which method to use to find the minimum loss? Sometimes, you may need to specify the 'learning_rate', the step size for every jump.
3. Which metric to evaluate the model?

```
model_seq.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),  # Optimi
 # Loss function to minimize
 loss=tf.keras.losses.BinaryCrossentropy(), # usually used for binary classificatic
 # List of metrics to monitor
 metrics=['accuracy'])
```

Hint:

1. The available optimizers? (help)
2. The available loss functions? (help)
3. The available metric functions? (help)

4.2 - Train your model

We will use 'your_model_name.fit' to train the model. In this function, we have several parameters to tune, shown as below.

```
history = model_seq.fit(
    X, # input data
    Y, # real output data
    batch_size=64, # how many samples (points) do you want to use to update weights once? If you hav
    epochs=10, # how many iterations do you want to train your model? Your data will be reused to tr
)
```

Please feel free to change the 'batch_size' and the 'epochs' to see how the parameters influence the training process.

Hint:

1. How to train your model? (help)

Exercise: Compile and train your model.

Please feel free to adjust the parameters in both the 'compile' and the 'fit' function to improve the performance.

```
### START CODE HERE ###
model.compile(
    optimizer = tf.keras.optimizers.SGD(learning_rate=.009),
    loss = tf.keras.losses.BinaryCrossentropy(),
    metrics = ['accuracy']
)

history = model.fit(
    X,
    Y,
    batch_size=500,
    epochs=10,
)

### End CODE HERE ###
```

```
Epoch 1/10
1/1 [==============================] - 0s 402ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 2/10
1/1 [==============================] - 0s 6ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 3/10
1/1 [==============================] - 0s 7ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 4/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 5/10
1/1 [==============================] - 0s 9ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 6/10
1/1 [==============================] - 0s 9ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 7/10
1/1 [==============================] - 0s 6ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 8/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 9/10
1/1 [==============================] - 0s 5ms/step - loss: 0.2884 - accuracy: 0.8900
Epoch 10/10
1/1 [==============================] - 0s 12ms/step - loss: 0.2884 - accuracy: 0.8900
```

Expected output: You are free to design your own neural network, choose your own optimizer, adjust your 'epochs', 'learning rate', and 'batch_size'. At the end of the epochs, the 'accuracy' should be greater than 0.9

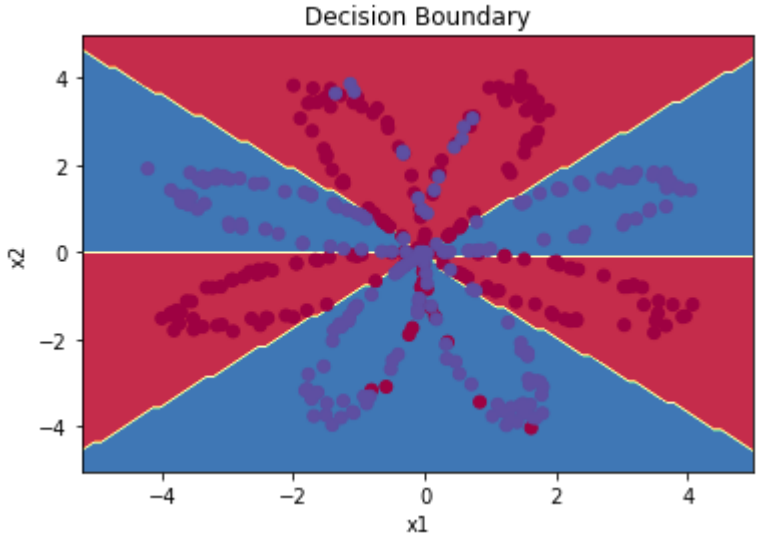
4 - Visualize your prediction

We define a function 'plot_boundary_tf(your_model_name, X, y)' to plot the boundary of the trained model. Please change the 'your_model_name' to the name of your model to visualize the result.

```
# Plot the decision boundary
def plot_boundary_tf(model, X, y):
    # Set min and max values and give it some padding
    x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
    y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
    h = 0.1
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole grid
    test_input=np.c_[xx.ravel(), yy.ravel()]
    Z = model.predict(test_input)
    Z = Z.reshape(xx.shape)
    Z01 = (Z>0.5)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z01, cmap=plt.cm.Spectral)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.scatter(X[0, :], X[1, :], c=y, cmap=plt.cm.Spectral)

### START CODE HERE ###
plot_boundary_tf(model, X.T, Y.T) # change the first parameter to be your model name to visua
### END CODE HERE ###
plt.title("Decision Boundary")
```

```
Text(0.5, 1.0, 'Decision Boundary')
```



✓ 0s completed at 10:21 PM