PRÁCTICA DE LABORATORIO N°02

CLASIFICACIÓN DE DATOS PLANAR CON UNA RED NEURONAL CON UNA CAPA OCULTA

Es hora de construir tu primera red neuronal, que tendrá una capa oculta. Verá una gran diferencia entre este modelo y el que implementó utilizando la regresión logística.

NSTRUCCIONES:

No utilice bucles (for / while) en su código

OBJETIVO:

- Construye una red neuronal completa con una capa oculta
- Hacer un buen uso de una unidad no lineal.
- Implementar forward and back prop.
- Entrenar la red neuronal
- Analizar el impacto de variar el tamaño de la capa oculta, incluido el sobreajuste.

BIBLIOGRAFÍA

- http://scs.ryerson.ca/~aharley/neural-networks/
- http://cs231n.github.io/neural-networks-case-study/

1 - PACKAGES

Importar todos los paquetes que utilizaremos necesitará durante esta práctica.

- **Numpy:** es el paquete fundamental para la computación científica con Python.
- **Sklearn:** proporciona herramientas simples y eficientes para la minería de datos y el análisis de datos.
- **Matplotlib:** es una biblioteca para trazar gráficos en Python.
- **testCases:** proporciona algunos ejemplos de prueba para evaluar la corrección de sus funciones
- **planar_utils:** proporcionan varias funciones útiles usadas en esta asignación.

```
import numpy as np
import matplotlib.pyplot as plt
from testCases_v2 import *
import sklearn
import sklearn.datasets
import sklearn.linear_model
from planar_utils import plot_decision_boundary, sigmoid, load_planar_dataset, load_extra_datasets

%matplotlib inline
np.random.seed(1) # establece una semilla(seed) para que los resultados sean consistentes
```

2 – DATA SET

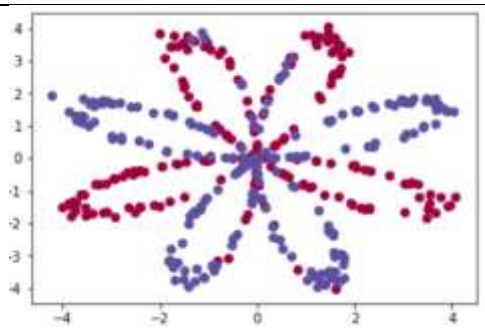
En primer lugar, vamos a obtener el conjunto de datos que va a trabajar. El siguiente código cargará un conjunto de datos de 2 clases de "flor" en las variables X e Y.

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

Visualice el conjunto de datos utilizando matplotlib. Los datos parecen una "flor" con algunos puntos rojos (etiqueta y = 0) y algunos azules (y = 1). Su objetivo es construir un modelo que se ajuste a estos datos.

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

Salida:



ACTIVIDAD 1: ¿Cuántos ejemplos de entrenamiento hay? y **¿Cuáles son las dimensiones (*shape*) de las variables X e Y? Sabiendo que se tiene:**

- a numpy-array (matriz) X que contiene las características (x1, x2)
- a numpy-array (vector) Y que contiene las etiquetas (red: 0, blue: 1).

```
shape_X =
shape_Y =
m =
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))
```

Salida Esperada:

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

3 – REGRESIÓN LOGÍSTICA SIMPLE

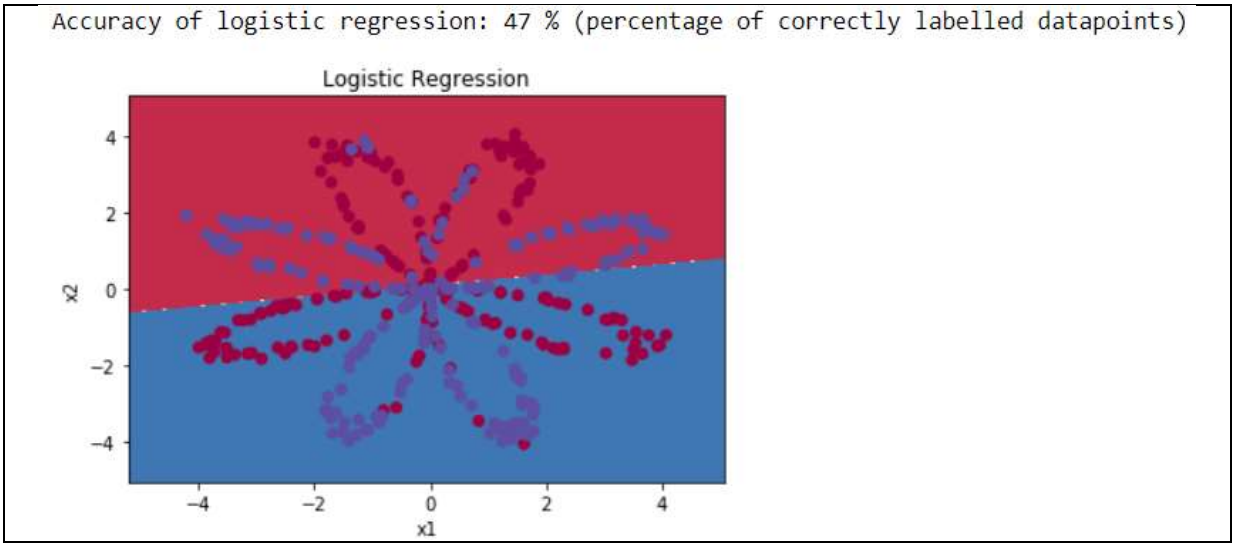
Antes de construir una red neuronal completa, veamos primero cómo funciona la regresión logística en este problema. Puedes usar las funciones incorporadas de sklearn para hacer eso. Ejecute el código siguiente para entrenar un clasificador de regresión logística en el conjunto de datos.

```
# Entrenando el clasificador de la regresión logística
clf = sklearn.linear_model.LogisticRegressionCV();
clf.fit(X.T, Y.T);
```

Ahora puede trazar el límite de decisión de estos modelos. Ejecuta el siguiente código.

```
# Plot el límite de decisión para la regresión logística
plot_decision_boundary(lambda x: clf.predict(x), X, Y)
plt.title("Logistic Regression")

# Imprimir la precesión ( accuracy)
LR_predictions = clf.predict(X.T)
print (‘Accuracy of logistic regression: %d ‘ % float((np.dot(Y,LR_predictions) + np.dot(1-Y,1-
LR_predictions))/float(Y.size)*100) +
      ‘% ‘ + “(percentage of correctly labelled datapoints)”)
```

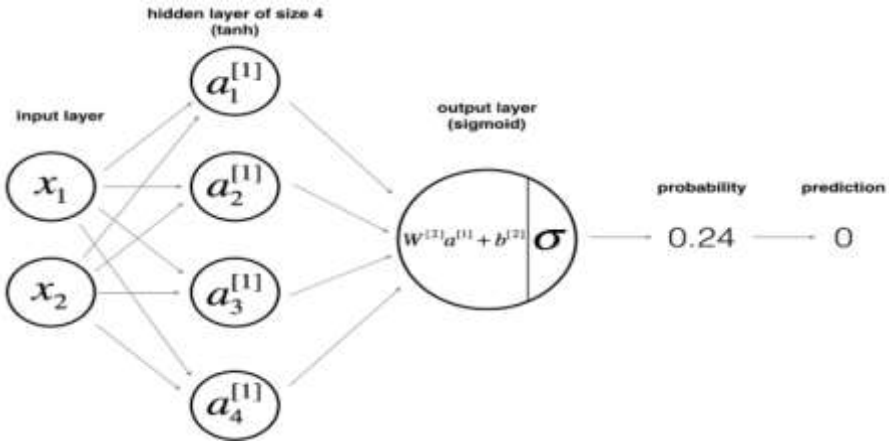


Interpretación: El conjunto de datos no se puede separar linealmente, por lo que la regresión logística no funciona bien. Esperemos que una red neuronal lo haga mejor. ¡Probemos esto ahora!

4 – MODELO DE RED NEURONAL

La regresión logística no funcionó bien en el "dataset de flores". Vas a entrenar una red neuronal con una sola capa oculta.

Aquí está el modelo:



Matemáticamente:

Para un ejemplo de entrenamiento $x^{(i)}$:

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]} \tag{1}$$

$$a^{[1](i)} = \tanh(z^{[1](i)}) \tag{2}$$

$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]} \tag{3}$$

$$\hat{y}^{(i)} = a^{[2](i)} = \sigma(z^{[2](i)}) \tag{4}$$

$$y^{(i)}_{prediction} = \begin{cases} 1 & \text{if } a^{[2](i)} > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Dadas las predicciones en todos los ejemplos, también puede calcular la función costo ***J*** de la siguiente manera:

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right) \quad (6)$$

Metodología general para construir una red neuronal

1. Defina la estructura de la red neuronal (número de unidades de entrada, número de unidades ocultas, etc.).
2. Inicializar los parámetros del modelo.
3. Loop:
    - Implementar la propagación hacia adelante.
    - Cálculo de la función de pérdida.
    - Implementar la propagación hacia atrás para obtener los gradientes.
    - Actualizar parámetros (gradiente descendiente).

A menudo se crean funciones de ayuda para calcular los pasos 1-3 y luego fusionarlos en una función que llamamos nn_model (). Una vez que haya creado nn_model () y haya aprendido los parámetros correctos, puede hacer predicciones sobre nuevos datos.

4.1 – ESTRUCTURA DE LA RED NEURONAL

ACTIVIDAD 2: Define 3 variables

- n_x: Tamaño de la capa de entrada
- n_h: Tamaño de la capa oculta (4)
- n_y: El tamaño de la capa de salida

Utilice las dimensiones de X e Y para encontrar n_x y n_y. Además, codifique el tamaño de la capa oculta para que sea 4.

```
def layer_sizes(X, Y):
    n_x =
    n_h =
    n_y =
    return (n_x, n_h, n_y)
```

```
X_assess, Y_assess = layer_sizes_test_case()
(n_x, n_h, n_y) = layer_sizes(X_assess, Y_assess)
print("Tamaño de la capa de entrada: n_x = " + str(n_x))
print("Tamaño de la capa oculta: n_h = " + str(n_h))
print("Tamaño de la capa de salida: n_y = " + str(n_y))
```

Salida esperada:

n_x	5
n_h	4
n_y	2

4.2 – INICIALIZACIÓN DE LOS PARÁMETROS DEL MODELO

ACTIVIDAD 3: Implemente la función `initialize_parameters()`

- Asegúrese de que los tamaños de sus parámetros son correctos. Consulte la figura de la red neuronal anterior si es necesario.
- Inicializará las matrices de pesos con valores aleatorios. Utilice: ***np.random.randn (a, b) * 0.01*** para inicializar aleatoriamente una matriz de forma (a, b).
- Inicializará los vectores de sesgo como ceros. Utilice: ***np.zeros ((a, b))*** para inicializar una matriz de forma (a, b) con ceros.

```
def initialize_parameters(n_x, n_h, n_y):
```

```
    np.random.seed(2) # configura una semilla para que su salida coincida con la esperada,
          aunque la inicialización es aleatoria.

    W1 =
    b1 =
    W2 =
    b2 =

    assert (W1.shape == (n_h, n_x))
    assert (b1.shape == ((n_h, 1)))
    assert (W2.shape == (n_y, n_h))
    assert (b2.shape == ((n_y, 1)))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

```
# Probando la Funcion
n_x, n_h, n_y = initialize_parameters_test_case()
parameters = initialize_parameters(n_x, n_h, n_y)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

Salida esperada:

W1	[[-0.00416758 -0.00056267] [-0.02136196 0.01640271] [-0.01793436 -0.00841747] [0.00502881 -0.01245288]]
b1	[[0.] [0.] [0.] [0.]]
W2	[[-0.01057952 -0.00909008 0.00551454 0.02292208]]
b2	[[0.]]

4.3 – FORWARD_PROPAGATION

ACTIVIDAD 4: Implemente la función `forward_propagation()`

- La función de activación tanh es parte de la biblioteca numpy: np.tanh ().
- Los pasos que debes implementar son:
  1. Recupere cada parámetro del diccionario "parameters" (que es la salida de initialize_parameters ()) mediante el uso de ***parameters ["..."].***
  2. Implementar la propagación hacia adelante. Calcule $Z^{[1]}, A^{[1]}, Z^{[2]}, A^{[2]}$
- Los valores necesarios en el back propagación se almacenan en "caché". El caché se dará como una entrada a la función de propagación hacia atrás.

```
def forward_propagation(X, parameters):
    # Recupera cada parámetro del diccionario "parameters"
    W1 =
    b1 =
    W2 =
    b2 =
```

```
    # Implementa Forward Propagation para calcular A2
    Z1 =
    A1 =
    Z2 =
    A2 =

    assert(A2.shape == (1, X.shape[1]))
    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2}

    return A2, cache
```

```
# Probando la función
X_assess, parameters = forward_propagation_test_case()
A2, cache = forward_propagation(X_assess, parameters)
# Note: Se calcula la media solo para asegurarnos de que su salida coincida
print(np.mean(cache['Z1']) ,np.mean(cache['A1']),np.mean(cache['Z2']),np.mean(cache['A2']))
```

Salida Esperada:

0.262818640198 0.091999045227 -1.30766601287 0.212877681719

ACTIVIDAD 5: Implemente la función costo J en compute_cost()

Ahora que ha calculado $A^{[2]}$(en la variable "A2" de Python), que contiene $A^{[2](i)}$ para cada ejemplo, puede calcular la función de costo como sigue:

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$

```
def compute_cost(A2, Y, parameters):

    m = Y.shape[1]

    cost =
    cost = np.squeeze(cost)

    assert(isinstance(cost, float))

    return cost

# Probando la función
A2, Y_assess, parameters = compute_cost_test_case()
print("cost = " + str(compute_cost(A2, Y_assess, parameters)))
```

Salida esperada:
cost = 0.692685886972..

4.4 – BACKWARD_PROPAGATION

ACTIVIDAD 6: Implemente la función backward_propagation().

La propagación hacia atrás suele ser la parte más difícil (más matemática) en el aprendizaje profundo, por lo que hará uso de las 6 ecuaciones siguientes:

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Para calcular dz1 necesitarás calcular $g^{[1]'}(Z^{[1]})$ dado que $g^{[1]'}(.)$ es la función de activación tanh, $Si\ a = g^{[1]'}(z)\ Entonces\ g^{[1]'}(z) = 1 - a^2$. Así que puedes calcular $g^{[1]'}(Z^{[1]})$ usando $(1 - np.power(A1,2))$

```
def backward_propagation(parameters, cache, X, Y):

    m = X.shape[1]

    W1 =
    W2 =
    # Recuperar los datos A1 and A2 del diccionario "cache".
    A1 =
    A2 =

    # Backward propagation: calcular dW1, db1, dW2, db2.
    dZ2 =
    dW2 =
    db2 =
    dZ1 =
    dW1 =
    db1 =

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads
```

```
# Probando la función
parameters, cache, X_assess, Y_assess = backward_propagation_test_case()

grads = backward_propagation(parameters, cache, X_assess, Y_assess)
print ("dW1 = "+ str(grads["dW1"]))
print ("db1 = "+ str(grads["db1"]))
print ("dW2 = "+ str(grads["dW2"]))
print ("db2 = "+ str(grads["db2"]))
```

Salida Esperada:

dW1 = [[0.00301023 -0.00747267][0.00257968 -0.00641288][-0.00156892 0.003893][-0.00652037 0.01618243]]
db1 = [[0.00176201][0.00150995][-0.00091736][-0.00381422]]
dW2 = [[0.00078841 0.01765429 -0.00084166 -0.01022527]]
db2 = [[-0.16655712]]

ACTIVIDAD 7: Actualización de los parámetros

Utilice la gradiente descendiente. Debe utilizar (dW1, db1, dW2, db2) para actualizar (W1, b1, W2, b2).

$\theta = \theta - \alpha \frac{\partial J}{\partial \theta}$, donde ***α*** es la tasa de aprendizaje y ***θ*** representa un parámetro.

```
def update_parameters(parameters, grads, learning_rate = 1.2):

    # Retrieve each parameter from the dictionary "parameters"
    W1 =
    b1 =
    W2 =
    b2 =

    # Retrieve each gradient from the dictionary "grads"
    dW1 =
    db1 =
    dW2 =
    db2 =

    # Actualizando los parámetros
    W1 =
    b1 =
    W2 =
    b2 =


    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters

#Probando la función
parameters, grads = update_parameters_test_case()
parameters = update_parameters(parameters, grads)

print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```