Regresion Logística

Objetivo

- Crear un objeto de regresión logística usando nn.Sequential model.

Tabla de contenido

Preparación

In [29]:

```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

In [30]:

```
# Import the libraries we need for this lab

import torch.nn as nn
import torch
import matplotlib.pyplot as plt
```

Establecemos la semilla aleatoria:

In [31]:

```
# Set the random seed

torch.manual_seed(2)
```

Out[31]:

```
<torch._C.Generator at 0x1ad7da062d0>
```

Función logística

Creamos un tensor:

In [32]:

```
z = torch.arange(-100, 100, 0.1).view(-1, 1)
print("The tensor: ", z)
```

```
The tensor:  tensor([[-100.0000],
        [ -99.9000],
        [ -99.8000],
        ...,
        [  99.7000],
        [  99.8000],
        [  99.9000]])
```

Creamos un objeto sigmoide:

In [33]:

```
# Create sigmoid object

sig = nn.Sigmoid()
```

Aplicamos:

In [34]:

```
# Use sigmoid object to calculate the

yhat = sig(z)
```

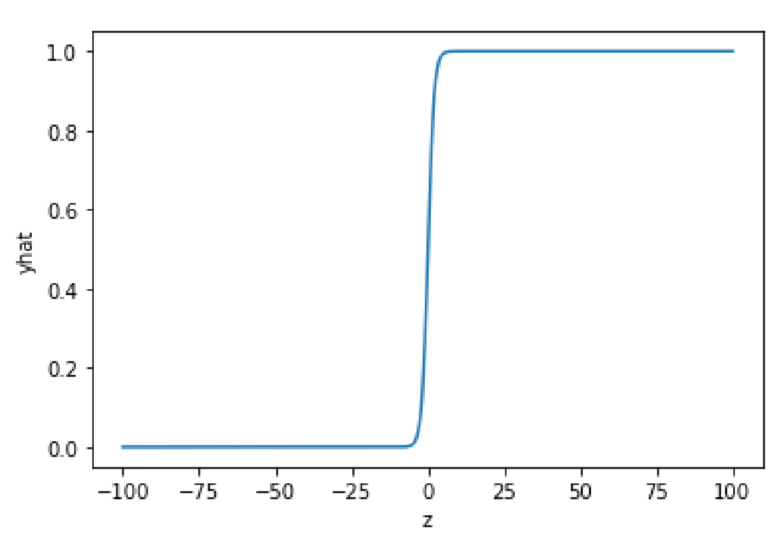
Graficamos:

In [35]:

```
plt.plot(z.numpy(), yhat.numpy())
plt.xlabel('z')
plt.ylabel('yhat')
```

Out[35]:

```
Text(0, 0.5, 'yhat')
```

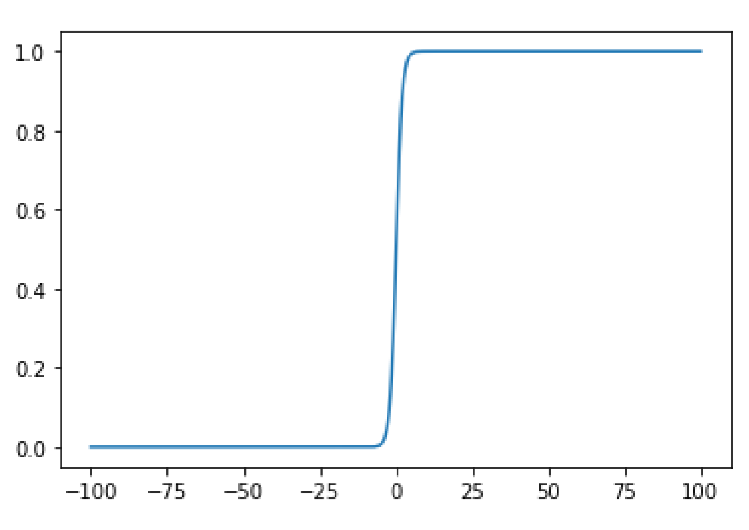


In [36]:

```
yhat = torch.sigmoid(z)
plt.plot(z.numpy(), yhat.numpy())
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x1ad0089c7c0>]
```



Regresión Logística con `nn.Sequential`

Creamos un tensor 1x1 donde x representa una muestra de datos con 1 dimensión, y un tensor 2x1 X que representa 2 muestras de datos en 1 dimensión:

In [37]:

```
# Create x and X tensor

x = torch.tensor([[1.0]])
X = torch.tensor([[1.0], [100]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1.]])
X =  tensor([[  1.],
        [100.]])
```

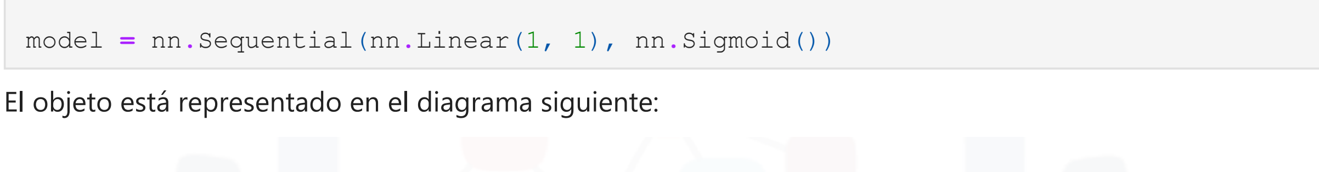
Creamos un objeto regresión logística con nn.Sequential con una entrada unidimensional:

In [38]:

```
# Use sequential function to create model

model = nn.Sequential(nn.Linear(1, 1), nn.Sigmoid())
```

El objeto está representado en el diagrama siguiente:



En este caso, los parámetros fueron inicializados aleatoriamente. Puede verlos de la siguiente forma:

In [39]:

```
# Print the parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[0.2294]], requires_grad=True), Parameter containing:
tensor([-0.2380], requires_grad=True)]

model.state_dict():
  OrderedDict([('0.weight', tensor([[0.2294]])), ('0.bias', tensor([-0.2380]))])
```

Realizamos una predicción con una muestra:

In [40]:

```
# The prediction for x

yhat = model(x)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.4979]], grad_fn=<SigmoidBackward>)
```

Llamar al objeto con el tensor `X` realiza la siguiente operación **los valores pueden no ser los mismos que los del diagrama dependiendo de la versión de PyTorch** :

$$\mathbf{x} = \begin{bmatrix} 1 \\ 100 \end{bmatrix}$$

$$\hat{y} = \sigma(-\mathbf{0.23} + 0.23 \begin{bmatrix} 1 \\ 100 \end{bmatrix})$$

$$\hat{y} = \sigma(\begin{bmatrix} 0 \\ 22 \end{bmatrix})$$

$$\hat{y} = \begin{bmatrix} \sigma(0) \\ \sigma(22) \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

Realizamos una predicción con múltiples muestras:

In [41]:

```
# The prediction for X

yhat = model(X)
yhat
```

Out[41]:

```
tensor([[0.4979],
        [1.0000]], grad_fn=<SigmoidBackward>)
```

Creamos un tensor 1x2 donde x representa una muestra una muestra de datos unidimensional, y uno 2x3 X que representa una muestra de datos en 2 dimensiones:

In [42]:

```
# Create and print samples

x = torch.tensor([[1.0, 1.0]])
X = torch.tensor([[1.0, 1.0], [1.0, 2.0], [1.0, 3.0]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1., 1.]])
X =  tensor([[1., 1.],
        [1., 2.],
        [1., 3.]])
```

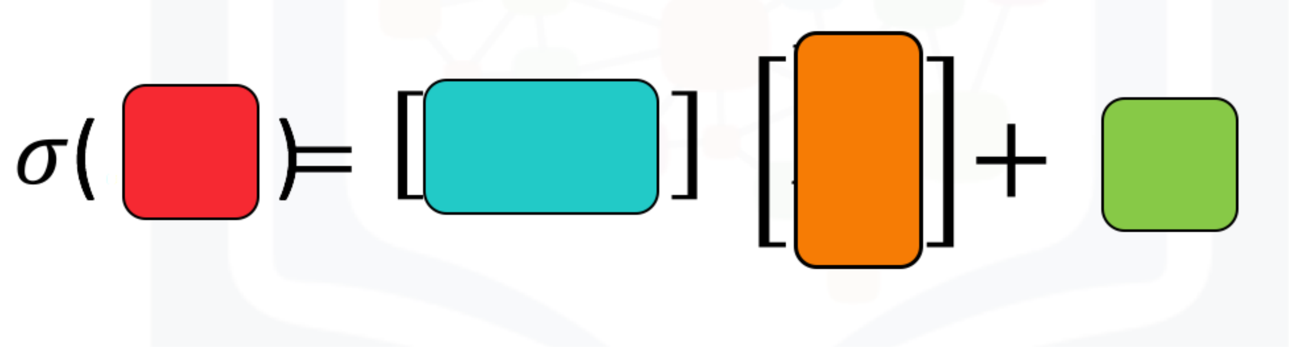
Creamos un objeto regresión logística con nn.Sequential con una entrada en 2 dimensiones:

In [43]:

```
# Create new model using nn.sequential()

model = nn.Sequential(nn.Linear(2, 1), nn.Sigmoid())
```

El objeto le aplicará la sigmoide a la salida de la función lineal como se muestra en el siguiente diagrama:



Los parámetros fueron inicializados aleatoriamente. Puede ver los mismos de la siguiente forma:

In [44]:

```
# Print the parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[ 0.1939, -0.0361]], requires_grad=True), Parameter containing:
tensor([0.3021], requires_grad=True)]

model.state_dict():
  OrderedDict([('0.weight', tensor([[ 0.1939, -0.0361]])), ('0.bias', tensor([0.302
1]))])
```

Realizamos una predicción para 1 muestra:

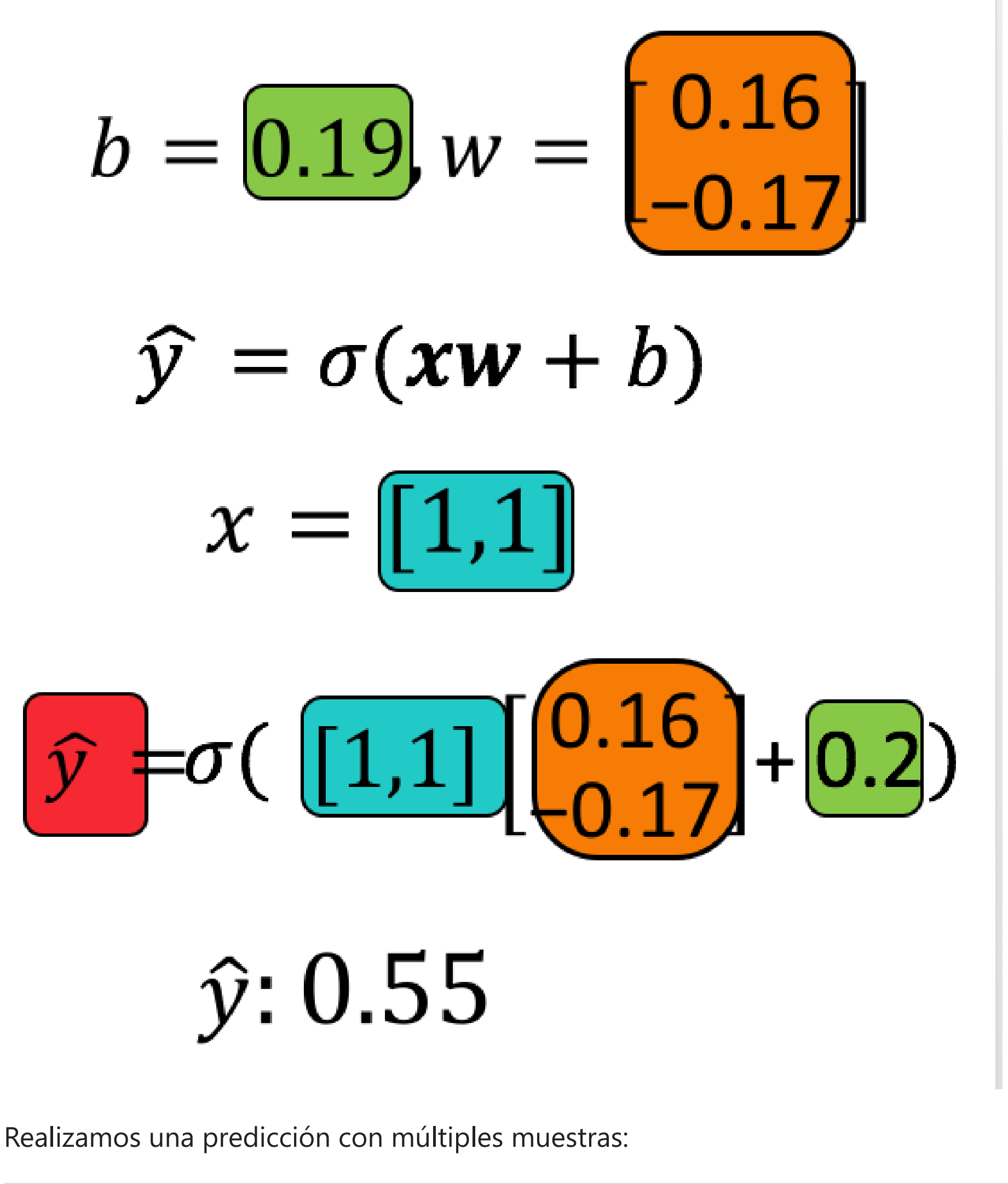
In [45]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.6130]], grad_fn=<SigmoidBackward>)
```

La operación se representa en el siguiente diagrama:



Realizamos una predicción con múltiples muestras:

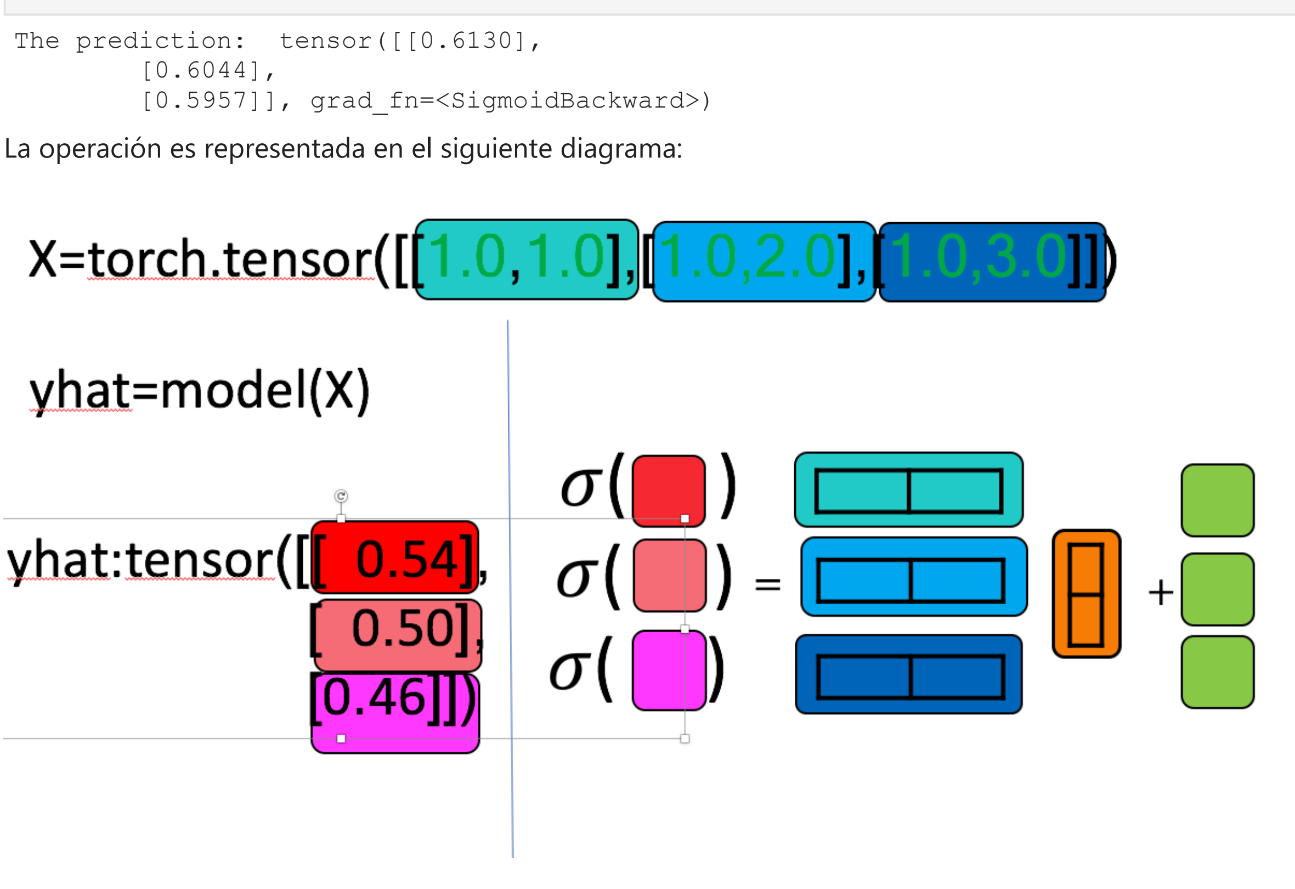
In [46]:

```
# The prediction of X

yhat = model(X)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.6130],
        [0.6044],
        [0.5957]], grad_fn=<SigmoidBackward>)
```

La operación es representada en el siguiente diagrama:



Construir módulos personalizados

Creamos un módulo personalizado logistic_regression:

In [47]:

```
# Create logistic_regression custom class

class logistic_regression(nn.Module):

    # Constructor
    def __init__(self, n_inputs):
        super(logistic_regression, self).__init__()
        self.linear = nn.Linear(n_inputs, 1)

    # Prediction
    def forward(self, x):
        yhat = torch.sigmoid(self.linear(x))
        return yhat
```

Creamos un tensor 1x1 x y un tensor 3x1 X:

In [48]:

```
# Create x and X tensor

x = torch.tensor([[1.0]])
X = torch.tensor([[-100], [0], [100.0]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1.]])
X =  tensor([[-100.],
        [   0.],
        [ 100.]])
```

Crear un modelo para predecir en 1 dimensión:

In [49]:

```
# Create logistic regression model

model = logistic_regression(1)
```

En este caso los parámetros fueron inicializados aleatoriamente. Puede verlos de la siguiente forma:

In [50]:

```
# Print parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[0.2381]], requires_grad=True), Parameter containing:
tensor([-0.1149], requires_grad=True)]

model.state_dict():
  OrderedDict([('linear.weight', tensor([[0.2381]])), ('linear.bias', tensor([-0.114
9]))])
```

Realizamos una predicción con una muestra:

In [23]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[0.5307]], grad_fn=<SigmoidBackward>)
```

Realizamos una predicción con múltiples muestras:

In [51]:

```
# Make the prediction of X

yhat = model(X)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[4.0805e-11],
        [4.7130e-01],
        [1.0000e+00]], grad_fn=<SigmoidBackward>)
```

Creamos un objeto regresión logística con una función con 2 entradas:

In [52]:

```
# Create logistic regression model

model = logistic_regression(2)
```

Creamos un tensor 1x2 x y uno 3x2 X:

In [53]:

```
# Create x and X tensor

x = torch.tensor([[1.0, 2.0]])
X = torch.tensor([[100, -100], [0.0, 0.0], [-100, 100]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1., 2.]])
X =  tensor([[ 100., -100.],
        [   0.,    0.],
        [-100.,  100.]])
```

Realizamos una predicción:

In [54]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[0.2943]], grad_fn=<SigmoidBackward>)
```

Realizamos una predicción con múltiples muestras:

In [55]:

```
# Make the prediction of X

yhat = model(X)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[7.7529e-33],
        [3.4841e-01],
        [1.0000e+00]], grad_fn=<SigmoidBackward>)
```

Práctica

Realice su propio `my_model` e imprima la predicción:

In [56]:

```
# Practice: Make your model and make the prediction

X = torch.tensor([-10.0])
#
my_model = nn.Sequential(nn.Linear(1, 1),nn.Sigmoid())
yhat = my_model(X)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([0.2231], grad_fn=<SigmoidBackward>)
```