# Regresión Lineal Múltiple

### **Objetivos** Realizar predicciones para multiples entradas

- Usar la clase Linear para construir modelos complejos
- Construir un módulo personalizado

#### Predicción Clase Linear

Tabla de contenido

- Construir módulos personalizados
- Preparación

#### # Import the libraries and set the random seed In [1]: from torch import nn

import torch

w = torch.tensor([[2.0], [3.0]], requires\_grad=True)

b = torch.tensor([[1.0]], requires\_grad=True)

Importamos las librerías y establecemos la semilla aleatoria.

```
torch.manual seed(1)
Out[1]: <torch._C.Generator at 0x2d3a8b1d250>
       Predicción
```

In [2]: # Set the weight and bias

## Definimos los parámetros torch.mm usa multiplicación de matrices.

In [3]: # Define Prediction Function

Establecemos los pesos y el sesgo

```
def forward(x):
     yhat = torch.mm(x, w) + b
      return yhat
La función forward implementa la siguiente ecuación:
```

y = xw + b

 $\widehat{y} = xw + b$ 

muestra:

In [6]: # Make the prediction of X

Clase Linear

Creamos un modelo:

model = nn.Linear(2, 1)

Predecimos para múltiples muestras X:

# Make a prediction of X

print("The result: ", yhat)

The result: tensor([[-0.0848],[-0.3969],

yhat:tensor([[-0.08],

Construiremos un módulo personalizado.

Esto ingresará la siguiente ecuación:

[-0.40],

[-0.709]]

Construir módulos personalizados

yhat = model(X)

complejos.

In [9]:

In [5]: # Sample tensor X

```
In [4]:
         # Calculate yhat
         x = torch.tensor([[1.0, 2.0]])
         yhat = forward(x)
         print("The result: ", yhat)
         The result: tensor([[9.]], grad fn=<AddBackward0>)
                b = \boxed{-1} w = \boxed{2 \choose 3}
```

Siendo la entrada es un tensor 1x2 (ya que tenemos un w 2x1), la salida será un tensor 1x1:

yhat = forward(X) print("The result: ", yhat) The result: tensor([[ 6.], [12.]], grad\_fn=<AddBackward0>)

Podemos usar la clase linear para realizar una predicción. También la usará para construir modelos más

Cada fila del siguiente tensor representa una

X = torch.tensor([[1.0, 1.0], [1.0, 2.0], [1.0, 3.0]])

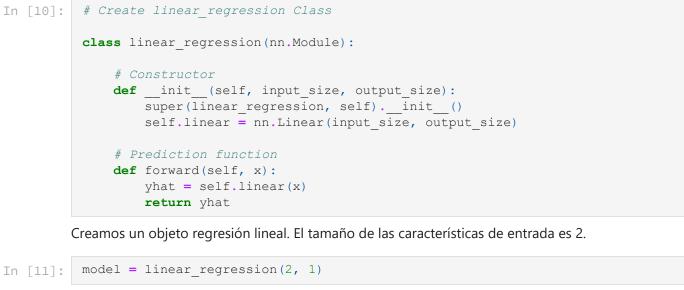
In [7]: # Make a linear regression model using build-in function

```
Realizamos una predicción para 1 muestra:
In [8]: # Make a prediction of x
         yhat = model(x)
         print("The result: ", yhat)
         The result: tensor([[-0.3969]], grad_fn=<AddmmBackward>)
```

X=torch.tensor([[1.0,1.0],[ $\hat{y} = -1 + Xw$ yhat=model (X)

La función realiza la multiplicación matricial como se muestra en la imagen de abajo:

[-0.7090]], grad fn=<AddmmBackward>)



y = xw + b

X

# Print model parameters In [12]: print("The parameters: ", list(model.parameters())) The parameters: [Parameter containing:

También puede ver los parámetros usando el método state\_dict():

tensor([0.4241], requires grad=True)]

In [15]:

**Práctica** 

Puede ver los parámetros inizialidos aleatoriamente usando el método parameters():

tensor([[ 0.3319, -0.6657]], requires grad=True), Parameter containing:

```
In [13]:
          # Print model parameters
           print("The parameters: ", model.state_dict())
          The parameters: OrderedDict([('linear.weight', tensor([[ 0.3319, -0.6657]])), ('linea
          r.bias', tensor([0.4241]))])
         Ahora ingresamos un tensor 1x2 y obtendremos un tensor 1x1.
          # Make a prediction of x
In [14]:
           yhat = model(x)
           print("The result: ", yhat)
          The result: tensor([[-0.5754]], grad fn=\langle AddmmBackward \rangle)
         La forma de la salida se muestra en la siguiente imagen:
```

y = xw + b



```
In [16]: # Practice: Build a model to predict the follow tensor.
          X = torch.tensor([[11.0, 12.0, 13, 14], [11, 12, 13, 14]])
          # solución
          model = linear_regression(4, 1)
          yhat = model(X)
          print("The result: ", yhat)
         The result: tensor([[2.1062],
                 [2.1062]], grad fn=<AddmmBackward>)
```

Construya un objeto del tipo linear\_regression. Prediga la salida del siguiente tensor: