

# Regresión Lineal Múltiple

## Objetivos

- Realizar predicciones para multiples entradas
- Usar la clase Linear para construir modelos complejos
- Construir un módulo personalizado

## Tabla de contenido

- Predicción
- Clase Linear
- Construir módulos personalizados

## Preparación

Importamos las librerías y establecemos la semilla aleatoria.

```
In [1]: # Import the libraries and set the random seed

from torch import nn
import torch
torch.manual_seed(1)
```

Out[1]: <torch.\_C.Generator at 0x2d3a8b1d250>

## Predicción

Establecemos los pesos y el sesgo

```
In [2]: # Set the weight and bias

w = torch.tensor([[2.0], [3.0]], requires_grad=True)
b = torch.tensor([1.0], requires_grad=True)
```

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

```
In [3]: # Define Prediction Function

def forward(x):
    yhat = torch.mm(x, w) + b
    return yhat
```

La función forward implementa la siguiente ecuación:

$$y = xw + b$$

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

```
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>)

$$b = \begin{bmatrix} -1 \end{bmatrix}, w = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$\hat{y} = xw + b$$

$$x = \begin{bmatrix} 3 & 2 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} + 1$$

$$\hat{y}: 9$$

## Cada fila del siguiente tensor representa una muestra:

```
In [5]: # Sample tensor X

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

```
In [6]: # Make the prediction of X

yhat = forward(X)
print("The result: ", yhat)
```

The result: tensor([ 6.,  
[ 9.],  
[12.]], grad\_fn=<AddBackward0>)

## Clase Linear

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

Creamos un modelo:

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

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

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>)

Predecimos para múltiples muestras `X`:

```
In [9]: # Make a prediction of X

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

The result: tensor([[ -0.0848],  
[-0.3969],  
[-0.7090]], grad\_fn=<AddmmBackward>)

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

$$X = \text{torch.tensor}(\begin{bmatrix} 1.0 & 1.0 \\ 1.0 & 2.0 \\ 1.0 & 3.0 \end{bmatrix})$$
$$\hat{y} = -1 + Xw$$

yhat:tensor([[-0.08],  
[-0.40],  
[-0.709]])

## Construir módulos personalizados

Construiremos un módulo personalizado.

```
In [10]: # Create linear_regression Class

class linear_regression(nn.Module):

    # Constructor
    def __init__(self, input_size, output_size):
        super(linear_regression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

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

Creamos un objeto regresión lineal. El tamaño de las características de entrada es 2.

```
In [11]: model = linear_regression(2, 1)
```

Esto ingresará la siguiente ecuación:

$$y = xw + b$$

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

```
In [12]: # Print model parameters

print("The parameters: ", list(model.parameters()))
```

The parameters: [Parameter containing:  
tensor([[ 0.3319, -0.6657]], requires\_grad=True), Parameter containing:  
tensor([0.4241], requires\_grad=True)]

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

```
In [13]: # Print model parameters

print("The parameters: ", model.state_dict())
```

The parameters: OrderedDict([('linear.weight', tensor([[ 0.3319, -0.6657]])), ('linear.bias', tensor([0.4241])])

Ahora ingresamos un tensor 1x2 y obtendremos un tensor 1x1.

```
In [14]: # Make a prediction of x

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

The result: tensor([-0.5754], grad\_fn=<AddmmBackward>)

La forma de la salida se muestra en la siguiente imagen:

$$y = xw + b$$

Realizamos una predicción para múltiples muestras:

```
In [15]: # Make a prediction of X

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

The result: tensor([[ 0.0903],  
[-0.5754],  
[-1.2411]], grad\_fn=<AddmmBackward>)

La forma se muestra en la imagen siguiente:

## Práctica

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

```
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>)