* Linear Regression prediction (in 1 dimension)
It is a method to help us understand the relationship between 2 variables:- The prediction (independent) variable x sometimes called a feature.
The target dependent variable Y
The relationship between the variables.

$$[Y = b + wx]$$

This is sometimes called a linear Mapping but in the one dimensional case, it just the equation of a line.
The Parameters:-

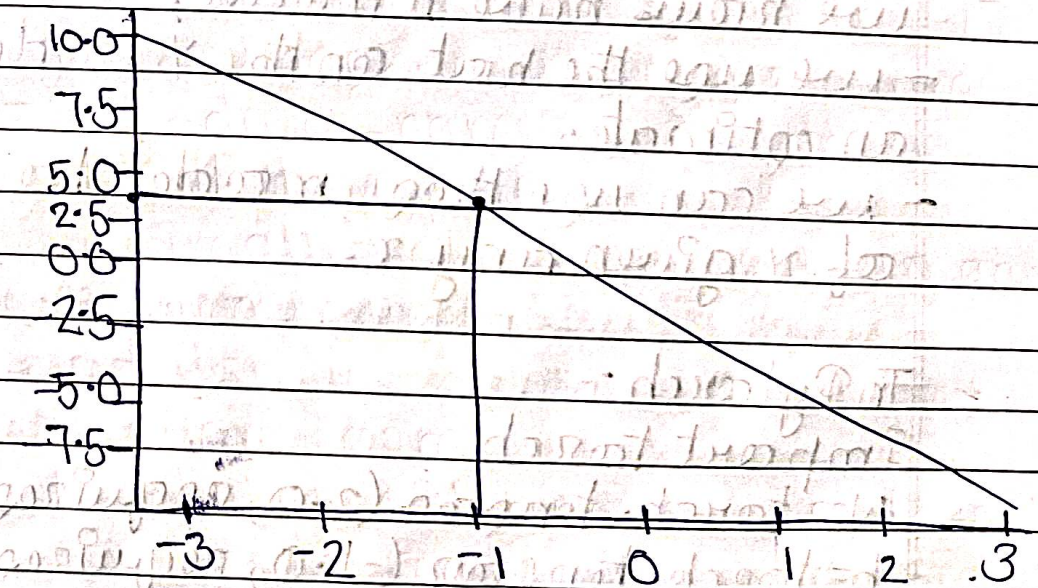
- b :- bias
- W :- The slope or weight

- Simple linear Regression :- Forward

$\hat{Y} = 1 - 3x$

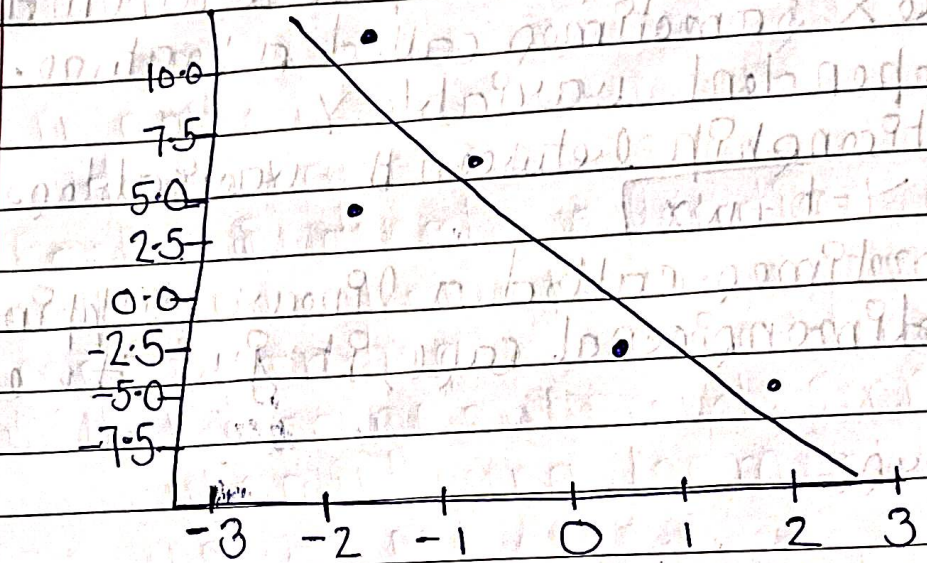
$= 1 - 3(-1)$

$= 4$



The line will map the dependent variable x, to the estimated value of Y, the hat on the Y indicates it's an estimate.
In Pytorch, we call the prediction step the forward step

In linear regression we have 2 steps:- we have a set of training Points



train (b, w)

Forward ($\hat{Y} = b + wx$)

- we use these training points to fit on train the model & get parameters.
- we then use these parameters in the model & then we now have a model;
- we use the hat on the Y to denote these models an estimate.
- we can use these model to predict any value of Y given any x.

In Pytorch:-

```
Import torch
w = torch.tensor(2.0, requires_grad = True)
b = torch.tensor(-1.0, requires_grad = True)
def forward(x):
    Y = w * x + b
    return Y
```

classmate
Date
Page

- The Parameter "in_features" is the size of each input sample or the no of columns.
- The Parameter "out_features" is the size of each output sample.

- we will us ... an to produce an object in the ...
- we will use ... make a prediction.

```
From t...
model= ... 1, out_features=1)
Y= model(x)
```

- To get the same results everytime we run the code we use the random seed.

```
From torch.nn Import Linear
torch.manual_seed(1)
model= linear(in_features=1, out_features=1)
Print(list(model.Parameters()))
```

• custom modules:-

- In Pytorch it is customary to make a custom module to create a model, using the Package nn.module.
- It's just python object.
- A custom module allows us to wrap multiple objects to make more complex modules.

custom modules for linear regression:-

```
Import torch.nn as nn
class LR(nn.module):
    def __init__(self, in_size, output_size):
        super(LR, self).__init__()
        self.linear=nn.linear(in_size, output_size)

    def forward(self,x):
        out=self.linear(x)
        return out
```

- we will use the constructor linear to produce an object in these case called model.
we will use the object model to make a prediction.

```
From torch.nn import linear
model = Linear(in_features=1, out_features=1)
Y= model(x)
```

To get the same results everytime we run the code we use the random seed.

```
From torch.nn import linear
torch.manual_seed(1)
model = linear(in_features=1, out_features=1)
Print(list(model.Parameters()))
```

ustom modules:-
n Pytorch it is customary to make a custom module to create a model, using the Package n.module.
ta just python object.
custom module allows us to wrap multiple ject to make more complex modules.
custom modules for linear regression:-

```
mport torch.nn as nn
ass LR(nn.module):
    def __init__(self, in_size, output_size):
        super(LR, self).__init__()
        self.linear = nn.linear(in_size, output_size)
    def forward(self, x):
```

* Linear Regression training :-
Illustration of a dataset using mathematical notation :-

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

- Imagine this set with many thousands of ordered pairs.
- Each pair denotes a location, or data point, within a cartesian plain.
- corresponding x, y co ordinates are marked with the same subscript, linking them together. Subscript range from n=1 to N with N defining the set size.
- commonly we will encounter datasets organized as tensors.

• (Simple) Linear Regression
- Regression is a method to predict a continuous value.
- Predicting housing price (y) giving the size of the house (x)
- Predicting stalk (y) prices using interest rates (x)
- Fuel economy of car (y) give horsepower (x)

* Loss :-
- Loss is a precursor to cost.
- The loss tells you how bad your model's prediction is compared to the actual value.
- In simple :-
It's just the average of the squared differences between what you predicted and what it really is

MSE :- Mean Squared Error (MSE)

- If predictions are close → loss is small → good!
- If predictions are far → loss is big → bad!

In Pytorch to calculate this automatically for you

```
import torch
import torch.nn as nn
loss_fn = nn.MSELoss
loss = loss_fn(y_pred, y_true)
```

* Gradient Descent :-

It is an optimization algorithm used to update model parameters (like weights and biases) to minimize the loss (error)

In Pytorch GD works like this :-

- Define a model (some weights you want to learn).
- compute the loss
- Compute the gradients (how to change weights to reduce loss).
- update the weights using the gradients.

Eg :-

```
import torch
x = torch.tensor([1.0, 2.0, 3.0])
y_true = torch.tensor([2.0, 4.0, 6.0])
w = torch.tensor([0.0], requires_grad = True)
lr = 0.01
for epoch in range(100):
    y_pred = w * x
    loss = ((y_true - y_pred) ** 2).mean()
    loss.backward()
```

The difference between cost & loss:-

cost:- The average loss over the entire training dataset.

loss:- The error for one data point (or one batch).

with torch
W -= l
W.grad.z
Print(f'Epoch
{l

* cost function:-
It is just a measure of how wrong our model
- It compares:-
The model's prediction
The actual true value (what it should be)
And gives you one number → The error
goal is to:- make this number as small as possible

• Mean Squared Error (MSE)
It is one of the most common cost Fn, espically for regression (Predicting numbers).

How it works (MSE):-
- For each data point:-
Find the error → (True value - Predicted value)
- Square that error:-
(Squaring makes it positive and penalizes bigger mistakes more)
- Add up all the squared errors.
- Divide by the number of data points → take the mean

MSE Formula:-

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_{true} - y_{pred})^2$$

```
with torch.no_grad():
    w -= lr * w.grad
    w.grad.zero_()
print(f'Epoch {epoch+1}: w={w.item():.4f},
      {loss.item():.4f}')
```

Cost function:-
It is just a measure of how wrong our model
It compares:-
The model's prediction
The actual true value (what it should be)
And gives you one number → The error
Goal is to:- make this number as small as poss

Mean Squared Error (MSE)
It is one of the most common cost fn, espical
for regression (Predicting numbers).

How it works (MSE):-
- For each data point:-
Find the error → (True value - Predicted value)
- Square that error:-
(Squaring makes it positive and penalizes big
mistakes more)
- Add up all the squared errors.
- Divide by the number of data points → take the
MSE Formula:-