In [ ]:
```
import numpy as np
```

Explanation

If I have this equation:

$$y = (x-1)^2$$

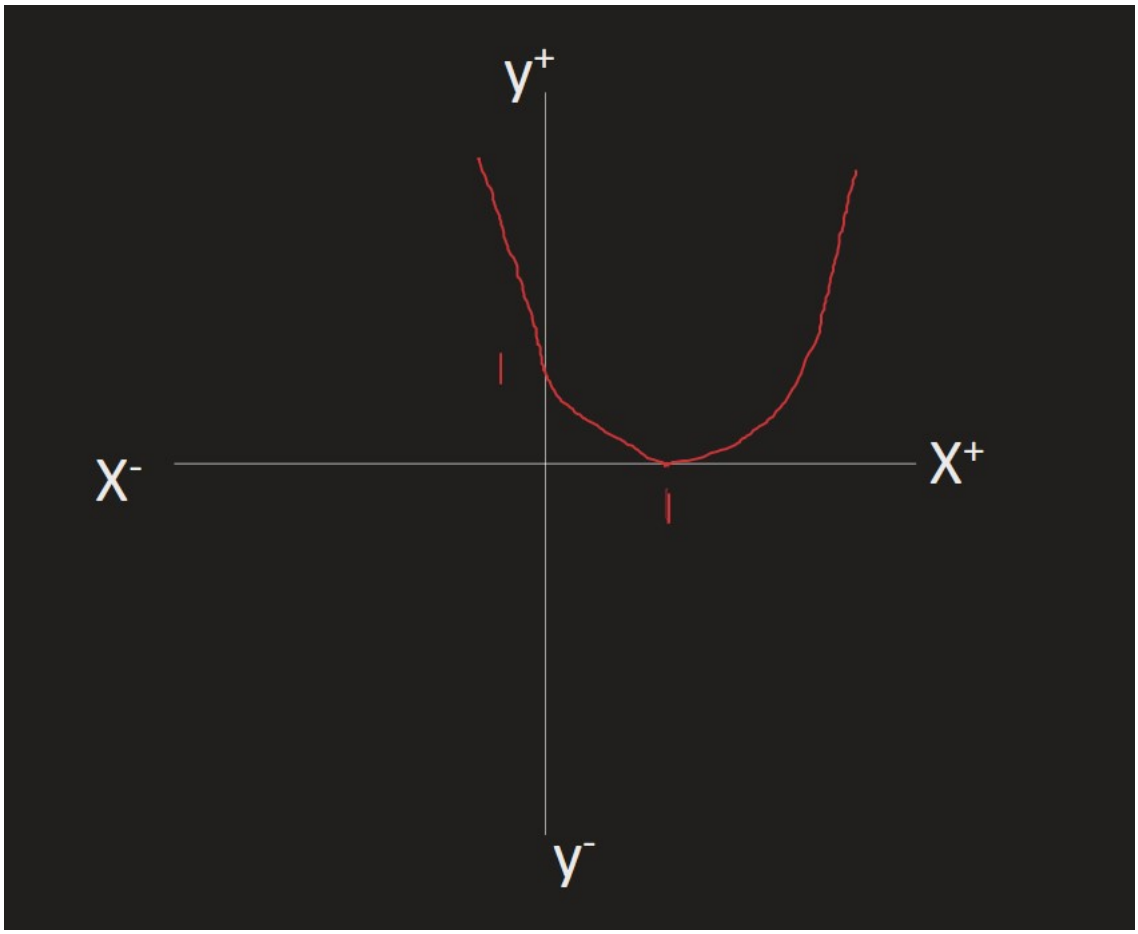
The minimum value of x that makes y get the smallest value is 1

We have three way to get x:

1. Drivative and make it equal 0 :

$$2(x-1) = 0, 2x = 2, x = 1$$

2. Draw the equation and from it will know the value of x :



3. Gradient Descent :

$$x_{new} := x_{old} - \alpha \frac{dy}{dx}$$

Solve this equation

In [ ]:
```python
def gradient_descent(X_old, alpha, iterates):
    # number of iteration
    for i in range(iterates):
        # the equation
        X_new = X_old - alpha * (2*X_old - 2)
        # put the result in x old to repeat till finish iteration
        X_old = X_new
    # return minimum x
    return X_new
```

In [ ]:
```python
gradient_descent(10, 0.001, 100000)
```

Out[ ]:
```
1.0000000000000555
```

one_variable

In preivous example we use dy/dx in gradient.

But in data what's the dy/dx ?

linear equation:

$$y = \sum_{i=1}^{m} w_i x_i + b$$

Cost function:

$$C = \frac{1}{2m} \sum_{i=1}^{m} (target - y)^2$$

From cost function I need to change values of weights and bais

$$\frac{\partial C}{\partial w_i}, \frac{\partial C}{\partial b}$$

we need to use chain rules :

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial y} * \frac{\partial y}{\partial w_i}$$

$$\frac{\partial C}{\partial y} = \frac{1}{m}\sum_{i=1}^{m}(target - y)$$

$$\frac{\partial y}{\partial w_i} = x$$

then it will be:

$$\frac{\partial C}{\partial w_i} = \frac{1}{m} * \sum(target - y) * x$$

for bais too:

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial y} * \frac{\partial y}{\partial b}$$

$$\frac{\partial y}{\partial b} = 1$$

then it will be:

$$\frac{\partial C}{\partial b} = \frac{1}{m} * \sum(target - y)$$

In [ ]:
```
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
```

In [ ]:
```
def gradient_descent(W_old, b_old, alpha, iterates):
    # number of iteration
    for i in range(iterates):
        # pred
        predict = x * W_old + b_old
        # the equation
        W_new = W_old - alpha * ((np.sum(predict - y))/x.shape[0])
```

```
        # put the result in x old to repeat till finish iteration
        W_old = W_new

        b_new = b_old - alpha * ((np.sum(predict - y))/x.shape[0])
        # put the result in b old to repeat till finish iteration
        b_old=b_new
    # return minimum W and b
    return W_new, b_new
```

In [ ]:
```
w, b = gradient_descent(1, 1, 0.001, 10000000)
```

In [ ]:
```
w, b
```

Out[ ]:
```
(1.4999999999999725, 1.4999999999999725)
```

In [ ]:
```
prediction = x * w + b
prediction
```

Out[ ]:
```
array([3. , 4.5, 6. , 7.5, 9. ])
```

In [ ]:
```
y
```

Out[ ]:
```
array([ 2,  4,  6,  8, 10])
```

Mean Absolute Error

In [ ]:
```
def mae_metric(actual, predicted):
        sum_error = 0.0
        for i in range(len(actual)):
                sum_error += abs(predicted[i] - actual[i])
        return sum_error / float(len(actual))
```

In [ ]:
```
mae_metric(y,prediction)
```

Out[ ]:
```
0.600000000000055
```

In [ ]:
```
# with one line same as function above
np.sum(abs(y- prediction)) / len(y)
```

Out[ ]:
```
0.600000000000055
```

Multivariable

we will add one's in features x for bais

In [ ]:
```
x = np.matrix([[1, 1, 2],
               [1, 2, 4],
               [1, 3, 6],
               [1, 4, 8],
               [1, 5,  10]])

y = np.matrix([[2],
               [3],
               [4],
               [5],
               [6]])
```

In [ ]:
```
x.shape
```

Out[ ]:
```
(5, 3)
```

In [ ]:
```
def gradient_descent(Ws_old, alpha, iterates):
    # assign weights new to contain zero's
    Ws_new = np.matrix(np.zeros(Ws_old.shape))
    # number of thetas
    parameter = int(Ws_old.ravel().shape[1])

    # loop in range number of iterates
    for i in range(iterates):
        # prediction
        predict = x * Ws_old.reshape(-1, 1)

        # loop in range number of thetas
        for j in range(parameter):

            # new theta = old theta - alph * (sum of predict - real
values)/number of rows
            Ws_new[0, j] = Ws_old[0, j] - alpha * ((np.sum(predict -
y))/x.shape[0])
```

```
        # put old thetas = new thetas
        Ws_old = Ws_new
    # return last values of weights
    return Ws_old
```

In [ ]:
```
# first theta is bais
thetas = np.matrix([0.5, 0.5, 0.5])
# call function
weights = gradient_descent(thetas, 0.001, 1000000)
```

In [ ]:
```
# see the weights
weights
```

Out[ ]:
```
matrix([[0.4, 0.4, 0.4]])
```

In [ ]:
```
# predict values
prediction = x * weights.reshape(-1, 1)
prediction
```

Out[ ]:
```
matrix([[1.6],
        [2.8],
        [4. ],
        [5.2],
        [6.4]])
```

In [ ]:
```
y
```

Out[ ]:
```
matrix([[2],
        [3],
        [4],
        [5],
        [6]])
```

In [ ]:
```

```