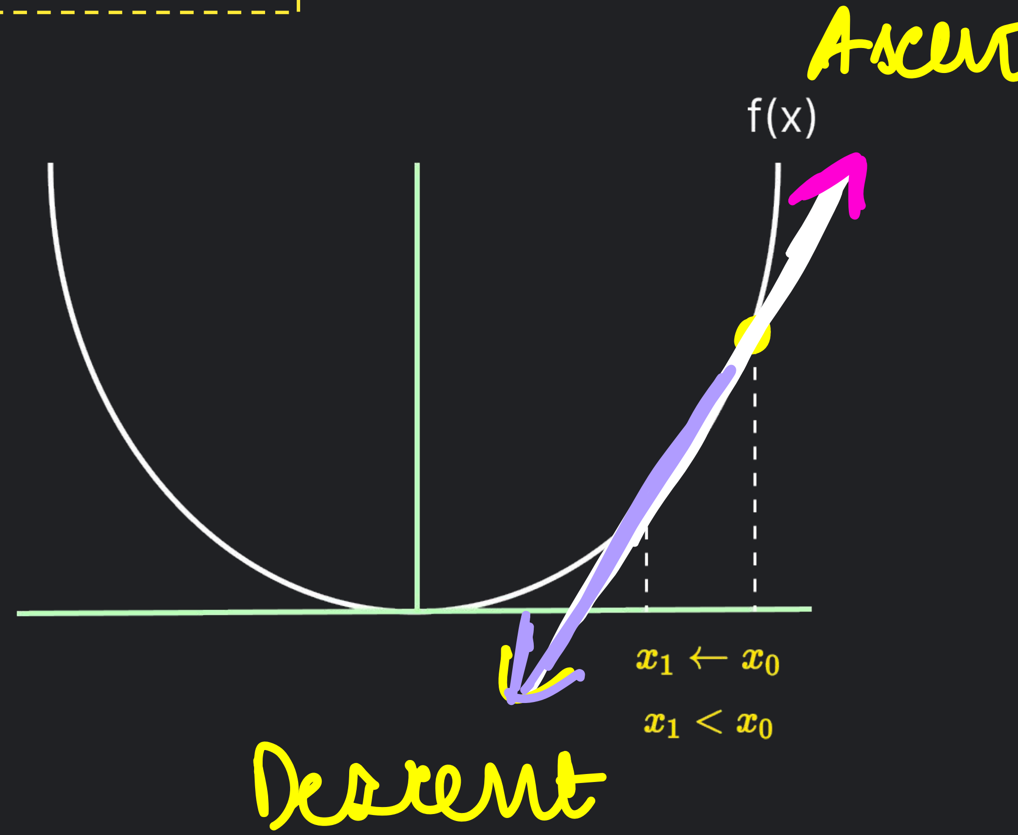
Gradient Descent Revision

https://colab.research.google.com/drive/1KxIeE7hhOlx-Zp-hdAdhN_6uCJ3UKwdc?usp=sharing

$y = x^2$



① Standing at a random point

② $\left.\dfrac{dy}{dx}\right|_{x=x_0} = +ve$

③ Go to towards minima, direction of steepesest Descent

$$x_1 \longrightarrow x_0 - \eta \left.\frac{dy}{dx}\right|_{x_0}$$

learning rate

gradient

Why GD?

To optimize the loss fn and find its minima

To minimize MSE

$$\sum_{i=0}^{m} (y^i - \hat{y}^i)^2$$

Quadratic function

ith sample

$$\sum_{i=0}^{m} \left(\underbrace{y^i}_{\text{constant}} - \left(\underbrace{w_0}_{\text{variable}} + \underbrace{w_1}_{\text{variable}} \underbrace{x_1^i}_{\text{constant}} \right) \right)^2$$

Using GD find w_0, w_1 s.t that MSE is minimized

Linear Regression Helper Functions

https://colab.research.google.com/drive/1KxIeE7hhOlx-Zp-hdAdhN_6uCJ3UKwdc?usp=sharing

—>>> We define our linear regression class and set LR & no. of iterations

```
import numpy as np
class LinearRegression() :
  def __init__(self, learning_rate=0.01, iterations=5):
    self.learning_rate = learning_rate
    self.iterations = iterations
```

eta

5 steps

—>>> Next we define our predicted Fn

$w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 - - - w_dx_d$

Remember : $\hat{y} = w_t X + w_0$ ——> We will call this as bias

```
def predict(self, X):
    return np.dot(X, self.W)+self.b

LinearRegression.predict=predict
```

(m,d) (d,1) (1,1)

(m,1) + (1,1)

$\langle w_1\ w_2\ .\ .\ w_d \rangle$

$\langle x_1\ .\ .\ .\ x_d \rangle$

Broadcasted m times

→ (m,1)

https://colab.research.google.com/drive/1KxIeE7hhOlx-Zp-hdAdhN_6uCJ3UKwdc?usp=sharing

—->>> Finally we evaluate our evaluation metric $R^2 Score$

```python
def r2_score(self, X, y):
  y_ = predict(self,X)
  ss_res = np.sum((y-y_)**2)
  ss_tot = np.sum((y- y.mean())**2)
  score = (1- ss_res/ss_tot)
  return score

LinearRegression.score=r2_score
```

EXAMPLE of a single predictor

$$L = \min_{w_0, w_1} \frac{1}{m} \sum_{i=1}^{m} \left[y^i - (w_0 + w_1 x_i^i) \right]^2$$

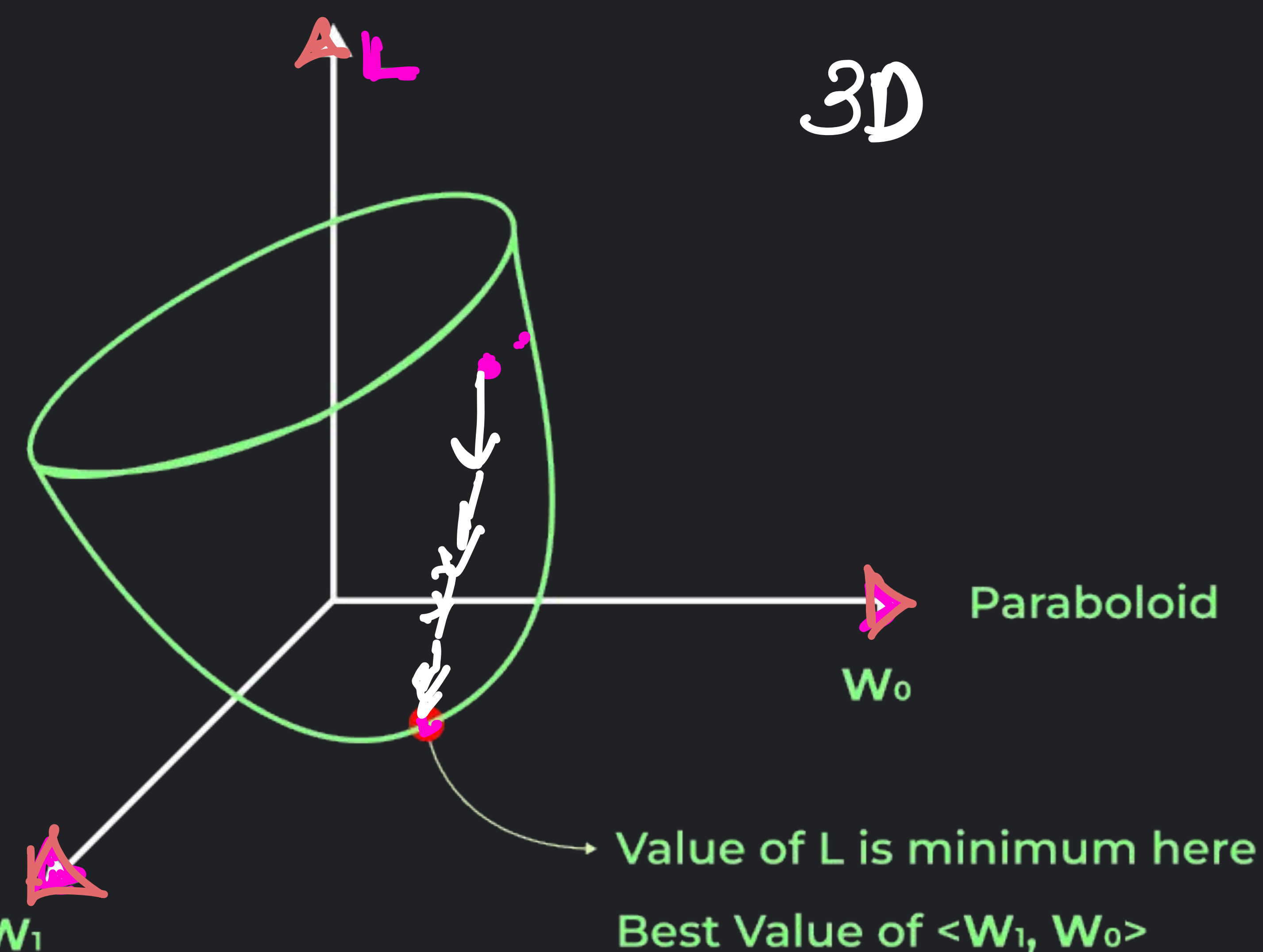
Quadratic

Paraboloid

2 parameter w_0, w_1

Error

3D



How will L change for multiple predictors ??

$$L = \min_{w_0, w_1, \ldots, w_d} \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} - (w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_d x_d^{(i)}]^2$$

d+1 parameters - < w_0, w_1, w_2 - - . w_d >

loss function - d+2 dimensional paraboloid

$\frac{\partial L}{\partial w_0}$. - - . $\frac{\partial L}{\partial w_d}$

$\frac{\partial L}{\partial w_1}$

$\frac{\partial L}{\partial w_2}$

How to find global minima?

$$\frac{\partial L}{\partial w_0} = 0$$

$$\frac{\partial L}{\partial w_1} = 0$$

$$\frac{\partial L}{\partial w_2} = 0$$

$$\vdots$$

$$\frac{\partial L}{\partial w_d} = 0$$

System of linear eq.

$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w_0 \rightarrow w_0 - \alpha \frac{\partial L}{\partial w_0}$$

$$w_d \longrightarrow w_d - \frac{\partial L}{\partial w_d} \alpha$$

Optimization

Take a partial derivative of L w.r.t each variable

$$\frac{\partial L}{\partial w_0} = \frac{\partial \sum (y^i - (\hat{y}^i))^2}{\partial w_0}$$

$$\frac{\partial x^2}{\partial x} = 2x$$

$$= \sum \frac{\partial (y^i - \hat{y}^i)^2}{\partial w_0}$$

constant

$$= \sum 2(y^i - \hat{y}) \frac{\partial (y^i - \hat{y}^i)}{\partial w_0}$$

$$w_2x_2 + w_1x_1 + w_0$$

$$= \sum -2(y^i - \hat{y}) \frac{\partial \hat{y}^i}{\partial w_0}$$

constant

$$= \sum -2(y^i - \hat{y}) \frac{\partial (w_2x_2 + w_1x_1 + w_0)}{\partial w_0}$$

$$= -\sum 2(y^i - \hat{y}^i) = -2.\sum \text{error}$$

$$L(w_2, w_1, w_0) = (y - (w_d x_2 + w_1 x_1 + w_0))^2$$

$$\frac{\partial L}{\partial w_1} = \sum -2(y^i - \hat{y}^i)\,\frac{\partial (w_2 x_2 + w_1 x_1 + w_0)}{\partial w_1}$$

$$= \sum -2(y^i - \hat{y}^i)\, x_1$$

$$= -2 \sum \text{error} \cdot x_1$$

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \cdots + w_d x_d$$

$w_0 \downarrow$ w_1: $-2.e$

$w_1 x_1 \downarrow$ w_2: $-2ex_1$

$w_2 x_2 \downarrow$ w_3: $-2ex_2$

$w_d x_d \downarrow$ $-2.e.x_d$

Notice a pattern?

The partial derivative

-2 . Error . Input value of wt.

For m points

$$\frac{\partial L}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m} -2(y-\hat{y})$$

$$\frac{\partial L}{\partial w_d} = \frac{1}{m}\sum_{i=1}^{m} -2(y-\hat{y})x_d$$

How to update the weights ??

$$w_0 \longrightarrow w_0 - \alpha \frac{\partial L}{\partial w_0}$$

Learning Rate

$$w_1 \longrightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

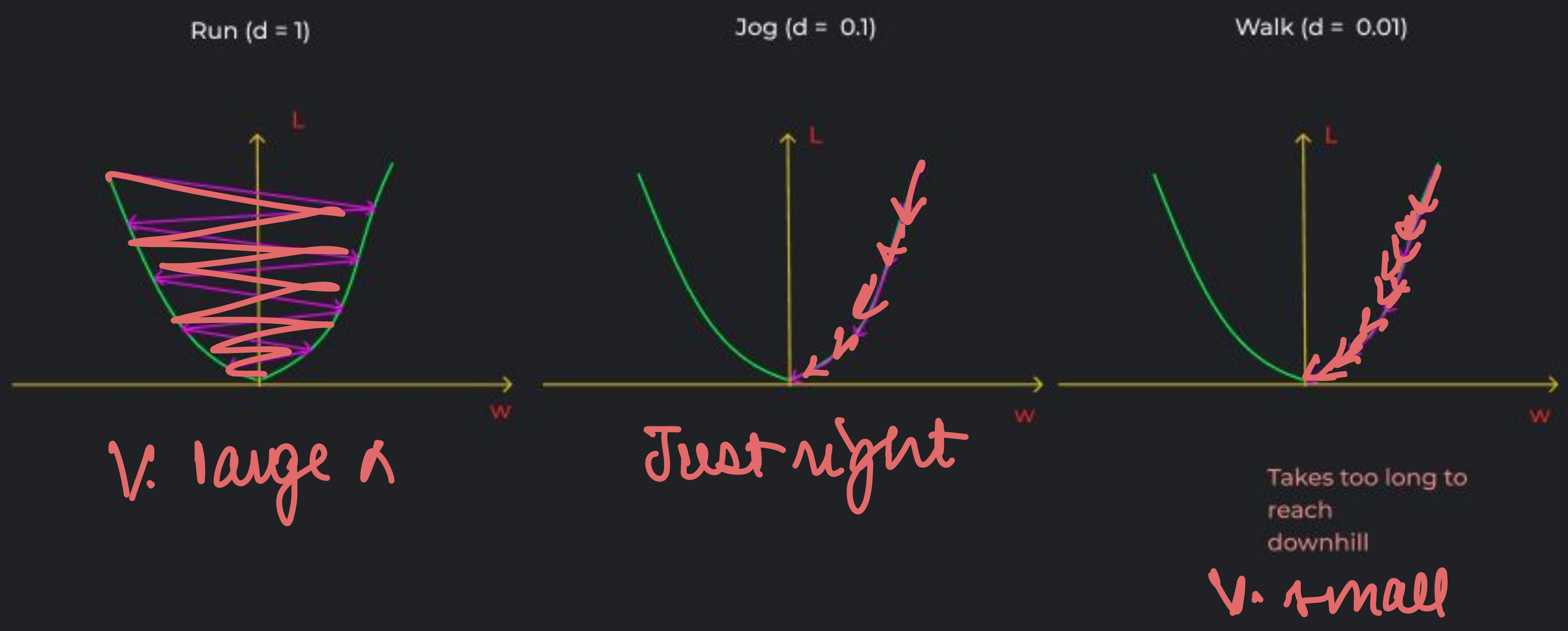
$$w_d \longrightarrow w_d - \alpha \frac{\partial L}{\partial w_d}$$

$\alpha \rightarrow$ v. large $\rightarrow$ Big steps $\rightarrow$ sprint (running)

$\alpha \rightarrow$ v. small $\rightarrow$ Baby steps $\rightarrow$ walk slowly

$\alpha \rightarrow$ midway $\rightarrow$ Right speed $\rightarrow$ Jogging

How to decide the right Linear Regression ??



Implementation of Wt.s update and GD

Weights update

Figure out dimension matching

```
[ ] def update_weights(self):
      Y_pred = self.predict( self.X )
      # calculate gradients
      dW = - (2*(self.X.T ).dot(self.Y - Y_pred))/self.m
      db = - 2*np.sum(self.Y - Y_pred)/self.m
      # update weights
      self.W = self.W - self.learning_rate * dW
      self.b = self.b - self.learning_rate * db
      return self

    LinearRegression.update_weights=update_weights
```

Note:

Lecture: Hyperparameter Tuning

~~Using 1 point of updation is called stochastic gradient descent.~~

~~Whereas if we update the weights after m points, it's called batch gradient descent.~~