Linear Regression

STOCHASTIC

Gradient Descent

- picks random instances
- update the weights using only one training example at a time

model: $y = mx + b$

residual = y - y_pred

$$MSE = \frac{1}{n}(\text{residual} ** 2)$$

$$w = w - (lr \times dL/dw)$$

$$b = b - (lr \times dL/db)$$

where,

$$\frac{dL}{dw} = (y - y\_pred) \cdot x$$

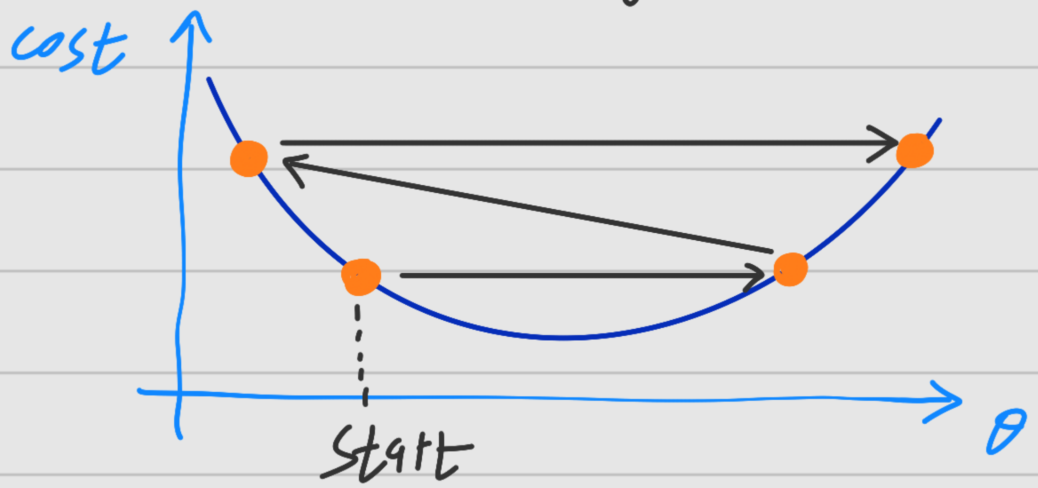
$$\frac{dL}{db} = (y - y\_pred) \cdot x$$

Learning Schedule Parameters:

↳ controls learning rate behaviour during training, directly affecting speed and accuracy.

→ Initial learning rate

Too high: Algorithm diverges, jumping all over the place & getting further from solution.



Too low: Algorithm will reach the solution but, will take too much time

$\eta(t) = \dfrac{t_0}{t + t_1}$ $\left\{ t_0, t_1 \text{ controls how fast learning rate decays} \right\}$

learning rate

· Slow down the updates to converge smoothly.

· helps escape local minima

$$\text{new } lr = \frac{\text{old } lr}{(1 + \text{decayrate} \cdot \text{step})}$$

$x_i \rightarrow x[i]$

$y_i \rightarrow y[i]$

$m, n \rightarrow X.\text{shape}$

y_pred = [Xi][weights]

weights = [zeros]

size(n)

Finding Gradient:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

$W \rightarrow [w_0, w_1, w_2, \ldots w_n]$ {weights}

$X \rightarrow [X_1, X_2, \ldots X_n]$ {input features with bias}

$\hat{y}_i \rightarrow$ predicted for ith sample

$y_i \rightarrow$ Actual value

Loss f^n: MSE

$$J(w) = \frac{1}{2}(\hat{y}_i - y_i)^2$$

$$= \frac{1}{2}(\underbrace{w^T x_i}_{\text{predicted}} - \overbrace{y_i}^{\text{Actual}})^2$$

To minimize loss, we take derivative of loss f^n wrt. weights

$$\frac{d}{dw_j}\left[\frac{1}{2}(w^T x_i - y_i)^2\right]$$

$$= \frac{1}{\cancel{2}} \times \cancel{2}(w^T x_i - y_i) \cdot \frac{d}{dw_j}(w^T x_i - \overbrace{y_i}^{\text{constant}})$$

$$\text{Gradient} = (w^T x_i - y_i) \cdot x_i$$

$$\text{Gradient} = (\hat{y}_i - y_i) \cdot x_i$$

$= \text{residual} \cdot \text{Input vector}$

Algorithm

↳ Parameters: X, y, steps, initial lr, decay

m, n → X.shape

weights → zeros array (n)

loss_list = []

for loop (steps):

for loops (m):

Xi → X[i]

yi → y[i]

y_pred → [Xi]·[weights]

loss = y - y_pred

gradient = loss · inputvector (Xi)

update learning rate

update weight

append the errors

return weights, loss History.