Gradient Descent Algorithm (GDA)

21 September 2025 11:24

* Pro-tip Loss Function vs Cost Function

Loss Function: It is the function to capture the difference between actual and predicted value.

for a single row or a single record or a single training example.

$$\text{error}) = \left[y_i - \hat{y}_i \right] \text{ for some } i^{th} \text{ row}$$

y_i: actual value. $\hat{y}_i$: predicted value

Cost Function

X1	X2	X3	X4	X5	X6	X7	Yi

actual value: $Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$

predicted value: $\hat{Y}_1, \hat{Y}_2, \hat{Y}_3, \hat{Y}_4, \hat{Y}_5, \hat{Y}_6$

$= e_1$, $= e_2$, $= e_3$, ..., $= e_6$

Regression Model

[average of errors]

in some way → Cost Function

for regression : MSE → Mean Squared Error

Cost Function:

It is used to refer an average of the loss functions in some way over the entire training dataset

Some famous cost Functions

MSE : Mean Squared Error

MSE : Mean squared Error

① error = $(y_i - \hat{y}_i)$

② squaring the error = $(y_i - \hat{y}_i)^2$

③ Mean of the squared error = $\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$

avg.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

$$\sqrt{MSE}$$

Task : Read about different cost functions for classification problems. → What is a logit ???

- Logistic Regression
- Decision Trees
- Support vector Machines
  - ↳ SVM...

Binary as well as Multi-class.

Gradient Descent Algorithm (GDA)

Gradient : is the rate of change → differentiation (partial derivatives)
or
derivative

+ → (slope / rate of change)

Descent coming down (direction is downwards)

→ Gradient descent tells us how steep the slope is and which direction decreases the error.

[∇J : gradient of J → it points in the direction of maximum increase of the cost/error function
↓
cost function]

GD: Gradient Descent moves in the opposite direction of the gradient because we want to minimize the error.

Gradient descent algorithm

repeat until convergence {

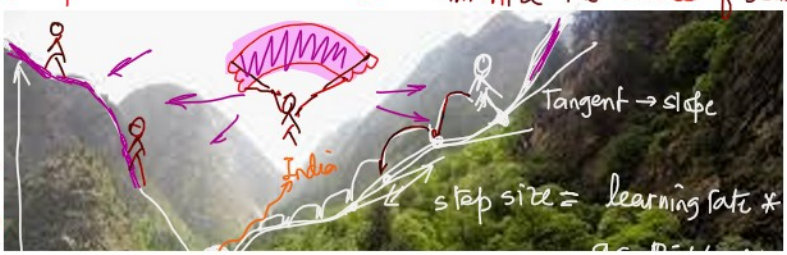
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Intuition : Abhinandan – Mig-21 Pilot got captured on the enemy's land.

© apc: Abhinandan needs to minimize the chance of getting caught.



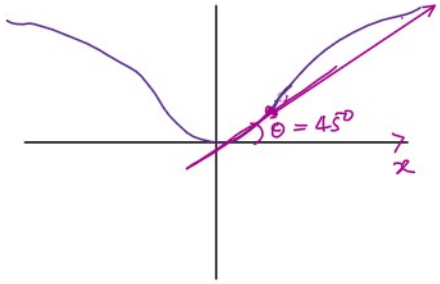
cost / loss Function

Gradient → slope → $\frac{dy}{dx}$ → $\tan\theta$

Descent → coming down.

Gradient Descent → follows the path of the steepest descent taking steps in the direction that decreases the slope and brings Abhinandan closer to the local minimum (coming to plains)

$$\tan\theta = \text{slope} = m = \frac{dy}{dx}$$



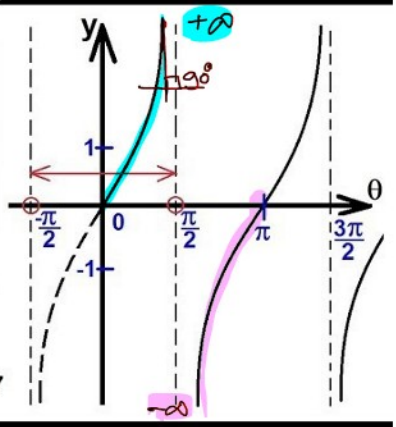
$\tan(\theta) = \tan 45^\circ = 1$

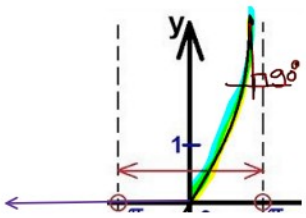
$\downarrow$

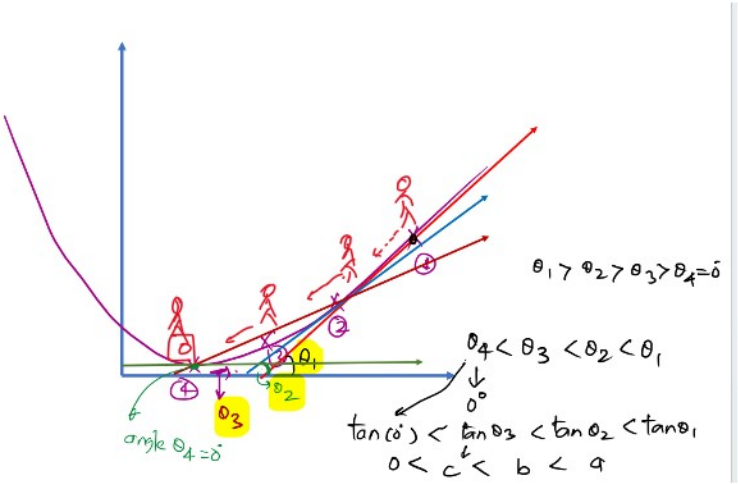
$\text{slope} = \frac{dy}{dx} = m = 1$

Graph for tangent function

θ (degree)	θ (radian)	tanθ
0	0	0
30	$\pi/6$	$1/\sqrt{3}$ 0.577
45	$\pi/4$	1
60	$\pi/3$	$\sqrt{3}$ 1.732
90	$\pi/2$	∞
120	$2\pi/3$	$-\sqrt{3}$ -1.732
135	$3\pi/4$	-1
150	$5\pi/6$	$-1/\sqrt{3}$ -0.577



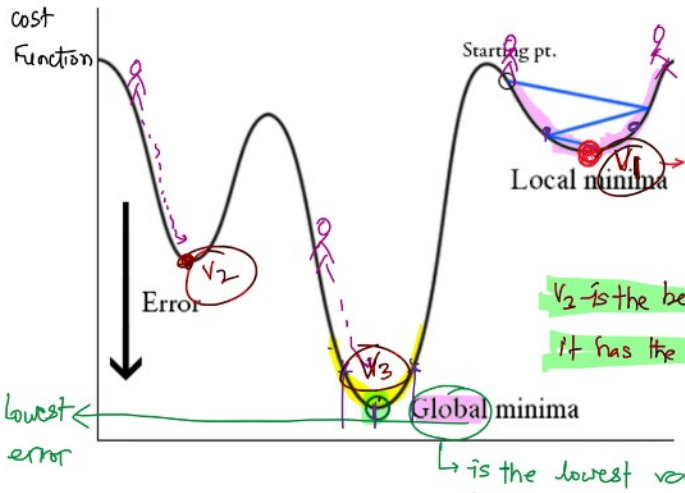




① As 'θ' increases from 0° till 90° → tanθ → slope also increases

② and as θ increases from 90° till 180° → tanθ → slope also decreases

Local Minimum Vs Global Minimum

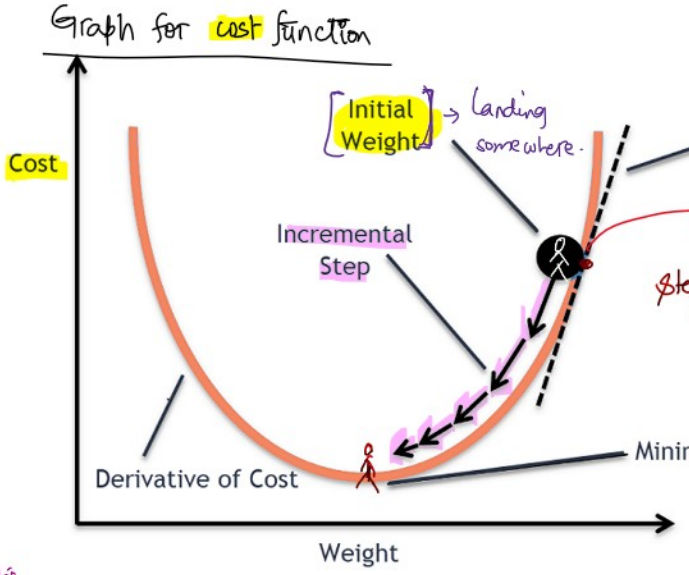


[local minimum]

V_2 is the best for the model as it has the lowest error.

is the lowest value of the cost function in the entire graph space.

* [Global minimum is the lowest of all local minima]



Initialization of the model → set random values for → weights, biases.

step size = [learning rate * Gradient of cost]

set by a modeller.

Pro-tip

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

cost Function : $J(\theta_0, \theta_1)$

partial derivative of $J(\theta_0, \theta_1)$ w.r.t. θ_0, θ_1 partially.

gradient of cost function

(for $j = 1$ and $j = 0$)

θ_0, θ_1 simultaneously.

}

↳ gradient of cost function

α / η : learning rate.

- learning rate

direction is opposite to the gradient hence it's called gradient descent.

$$\theta_1)_{new} = \theta_1)_{old} - (\text{learning rate} * \text{gradient of cost})$$

$\theta_1)_{new} \triangleq \theta_1)_{old}$.

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$ ← updating the θ_1 value.

$$\theta_1)_{new} = \theta_1)_{old} - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

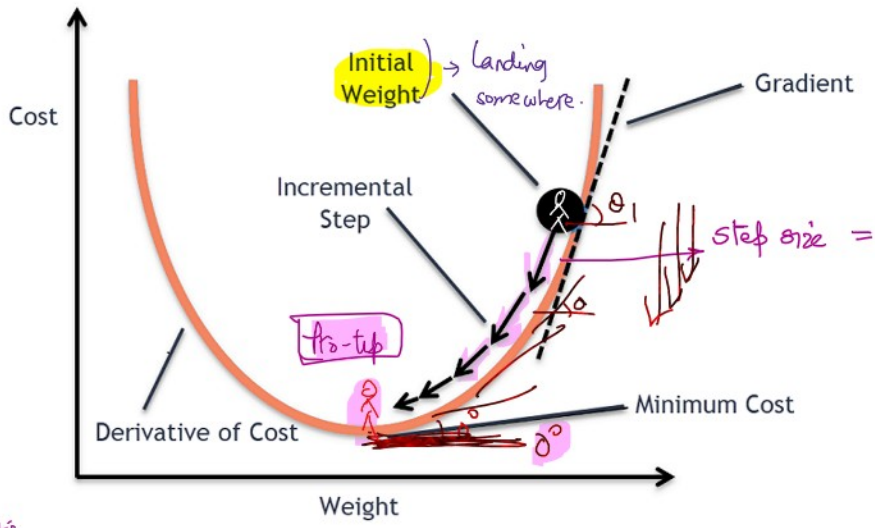
bias, weight

cost Function

gradient of cost function

← learning Rate * Gradient of Cost Function →

← step size →



step size = learning rate * gradient(cost)

constant * gradient [cost]

($\alpha = 0.01$)

Pro-tip

gradient becomes zero when $\theta = 0$

$$\theta_1)_{new} = \theta_1)_{old} - \alpha \cdot \frac{\partial}{\partial \theta_1}\left[J(\theta_0, \theta_1)\right]$$

$$\left[\theta_1)_{new} \triangleq \theta_1)_{old}\right] \rightarrow \text{GDA converges.}$$

Example:

θ_0 : bias

θ_1 : weight

$\alpha = 0.1$

$\theta_1)_{initial} = 2.1$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = 1.5$ slope | gradient

$\theta_1)_{new} = \theta_1)_{initial} - 0.1 * 1.5$

$\theta_1)_{new} = 2.1 - 0.1 * 1.5 =$ 2.1 - (0.1 * 1.5) = 1.95

$\theta_1)_{new} = 1.95$

$\theta_1)_{new_2} = 1.95 - 0.1 * [\ ]$

Learning Rate (α or η)

What is a hyper parameter ?

→ A hyper parameter is a setting chosen before training a machine learning model.

→ it controls how the learning process happens but it is not learned from the data itself → which is not a hyper parameter.

alpha (α)	weights (W)	biases (b)
↓	✓	✓
is a hyperparameter		

Examples of hyper parameters

① Linear / Logistic Regression

→ learning rate (gradient descent)

→ regularization [L1 → Lasso, L2 → Ridge]

② Decision Trees

→ minimum samples / leaf

→ max_depth of the tree

③ Neural Networks

- learning rate
- number of hidden layers
- number of neurons per layer
- batch size
- dropout rate
- optimizer choice (SGD, MBGD, Adam etc.)

Ingredients: Base, chopped olives, Jalepinos, chillies etc. ← Training data

Recipe steps: mixing // etc. ← ML algorithm.

Oven temperature, baking time: ← Hyperparameters

↓

Final Pizza taste ← Model Accuracy

Learning Rate (α or η)

Learning rate is a hyperparameter in a gradient based optimization technique that determines the step size at each iteration while moving towards the minimum of the cost function.

$$\text{step size} = \alpha * (\text{gradient of the cost function})$$

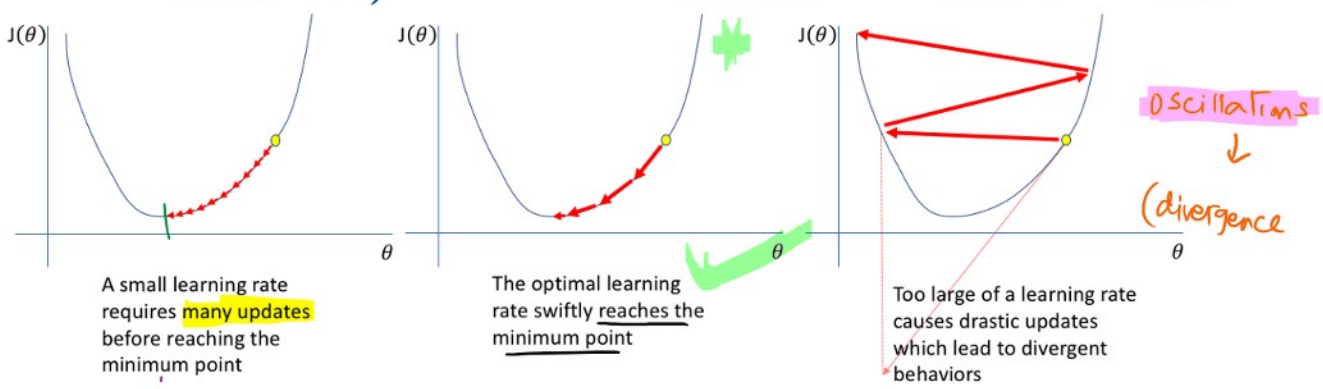
Right choice for learning rate

$$\alpha \in [0, 1]$$

balanced learning rate α: [0.01 – 0.4]

($\alpha = 0.01$)

Too low ($\alpha = 0.0001$) | Just right $\alpha = 0.1$ or 0.2 | Too high ($\alpha = 0.875$)



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

Too large of a learning rate causes drastic updates which lead to divergent behaviors

A small learning rate requires many updates before reaching the minimum point

↓

slow training

↓

more no. of epochs

↓

Project budget ↑↑↑

Training runtime ↑↑↑

The optimal learning rate swiftly reaches the minimum point

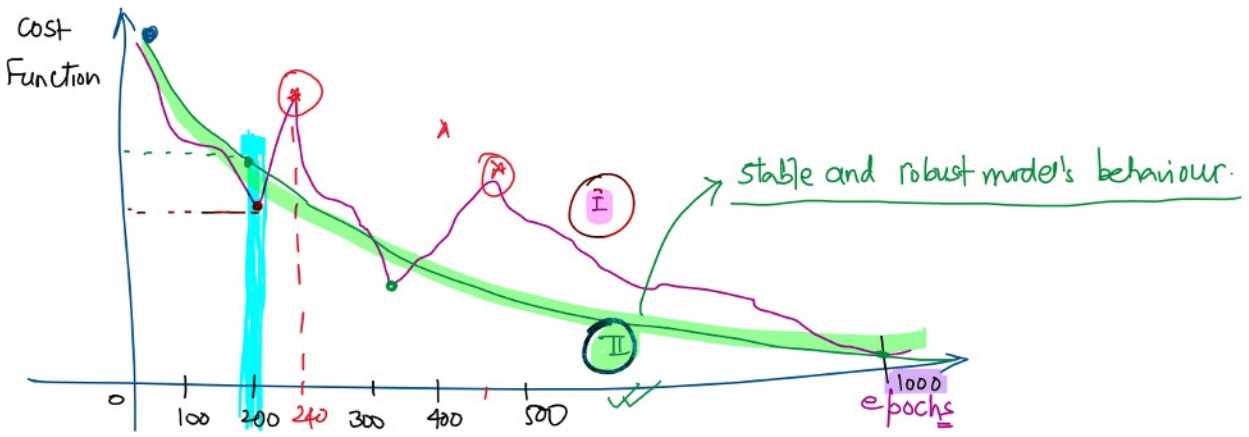
Too large of a learning rate causes drastic updates which lead to divergent behaviors

$\alpha = [0.01, 0.1, 0.2, 0.4]$

Pro-tip Why is setting up the learning rate so critical ??.

Learning rate affects the speed of convergence

- **Accuracy:** Whether the GDA can find the optimal solution or overshoot it, it depends on α.
- **stability:** presenting divergence or oscillations during training is a critical step to ensure robustness of the model.



Note: As per the best practices, start with a default

learning rate 'α' or 'η' → [0.01 – 0.2]

α = 0.01 *

α = [0.01 — 0.4]

(0.01, 0.03, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4]

10 ✓