Gradient Descent (Algorithom)

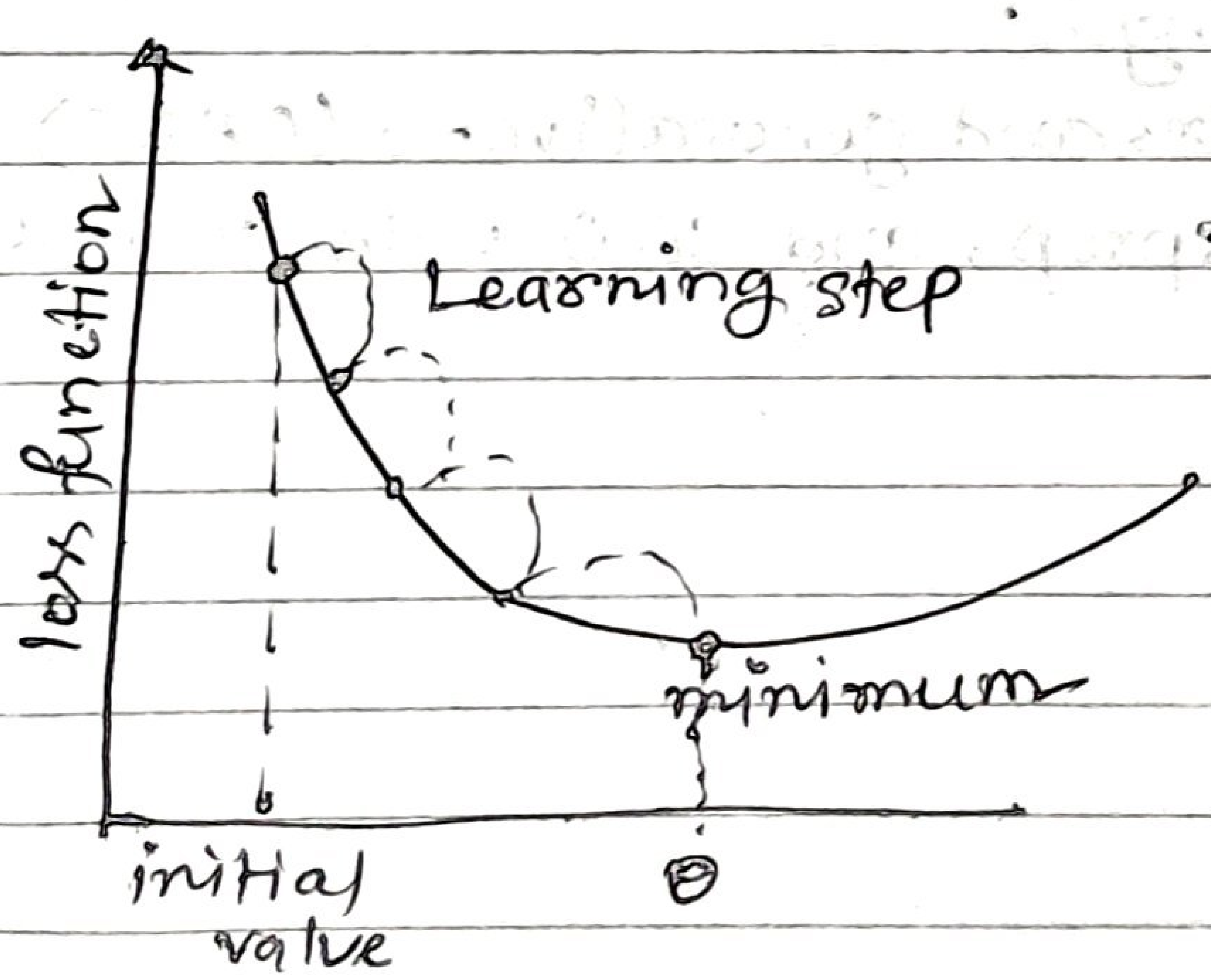
is a first order iterative optimization algorithm for finding the minimum of a function. Gradient descent can be performed on any defferentiable loss function. The main goal of gradient descent is to minimize a cost or loss function, optimizing model parameters for better performance.

1. Objective Function: The function you want to minimize (cost or loss function).

2. Gradient function calculation: Compute the gradient of the function, indicating the direction of the steepest ascent.

3. Learning rate: A hyperparameter that determines the size of the steps taken towards the minimum.

4. Update Rule: A formula to iteratively adjust the parameters in the opposite direction of the gradient to minimize the function



The main equation for updating parameters in GD.

$$x_{next} = x - \alpha . \nabla f(x) \qquad = w - \eta \frac{\partial L}{\partial w}$$

where, x_{next} = update parameter value.

x = current parameter value.

α = the Learning rate

$\Delta f(x)$ = is the gradient of the function f at x.

Some loss functions are widely used in ML and DL models

1. (MSE/MAE:) used for regression task.
2. (Cross - Entropy loss (or log Loss):) used for binary and multi-class classification task, output is probability value between 0 and 1.

3. (Hinge Loss:) Binary classification task, especially with support vector machines (SVMs).

4. Huber Loss: Regression task, MSE, MAE
5. Softmax-cross Entropy loss: multiclass- classification task.

Importance of Learning rate:

a. too big (x)
b. too small (x)

The typical range for the learning rate (α or η) is from 0.00001 to 0.1

Gradient descent : Batch, mini-batch & Stochastic

* Batch gradient descent: uses the 'Entire dataset' more stable but can be slow for large datasets

* Mini-Batch Gradient descent: uses small ~~data~~ batches balancing the speed, stability of the updates.

* Stochastic Gradient Descent: Updates parameters for each data point, can be faster but noisier.

BGD: Given a cost function $J(\theta)$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(h_\theta(x^i), y^i)$$

m is the number of traning examples,
L is the loss function, h_θ is the hypothesis function, $x^{(i)}$ is the input features, and $y^{(i)}$ is the target output

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \rightarrow \text{MSE}$$

The update rule, $\theta = \theta - \alpha \nabla_\theta J(\theta)$
α = learning rate

Mini-Batch Gradient Descent:

$$J_{batch}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(h_\theta(x^{(i)}, y^{(i)}))$$

$$\theta = \theta - \alpha \nabla_\theta J_{batch}(\theta)$$

Example:

First mini-Batch: (1,2), (2, 2.5)

$$\theta_0 = \theta_0 - \alpha \tfrac{1}{2} [h_\theta(1) - 2) + (h_\theta(2) - (2.5))]$$
$$\theta_1 = \theta_1 - \alpha \tfrac{1}{2} [h_\theta(1-2).1 + (h_\theta(2-2.5).2]$$

Second mini-Batch: (3, 3.5)

$$\theta_0 = \theta_0 - \alpha \cdot \tfrac{1}{1} [h_\theta(3-3.5)]$$
$$\theta_1 = \theta_1 - \alpha.1 [h_\theta(3-3.5).3]$$

Total sample: 1000
Number of mini-Batches: 16

To find the number of iterations per epoch,

$$\text{mini-Batch size} = \frac{\text{Total samples}}{\text{Number of mini-Batches}}$$
$$= \frac{1000}{16}$$
$$= 62.5$$

Total iteration = Number of Epochs × N-of mini-Batches

= $(n) \rightarrow 10 \times 16$

assume

= $\underline{160}$

Stochastic Gradient : Using only one data point at a time.

$$\theta = \theta - \alpha \nabla_\theta L(h_\theta(x^{i}, y^{(i)}))$$

Example →

$\theta_0 = \theta_0 - \alpha(h_\theta(1) - 2)$

$\theta_1 = \theta_1 - \alpha(h_\theta(1) - 2) \cdot 1$

update process for a single data point (3,5) for SGD

So, $x = 3$, $y = 5$, $\theta_0 = 0$, $\theta_1 = 1$, $\alpha = 0.1$

For $x = 3$,

① $h_\theta(3) = \theta_0 + \theta_1 \cdot 3$

hypothesis function
$h_\theta(x) = \theta_0 + \theta_1 x$

$= 0 + 1 \cdot \times 3$

$= 3$

② calculate the error:

Error $= h_\theta(3) - 5$

$= 3 - 5 = -2$

Error $= h_\theta(x - y)$

update, $\theta_0 = \theta_0 - \alpha(\text{Error})$

$= 0 - 0.1 \times (-2)$

$= 0.2$

update, $\theta_1 = \theta_1 - \alpha(\text{Error}) \cdot x$

$= 1 - 0.1 \times (-2) \cdot 3$

$= 1 + 0.6$

$= 1.6$

Gradient Descent with momentum

There are two main steps involved in applying momentum to the gradient descent optimization process:

1. Velocity update
2. Parameter update.

momentum = Negative of Gradient + momentum

The main point of is how momentum helps to avoid getting stuck in local minima and can push the optimization process pass these points.

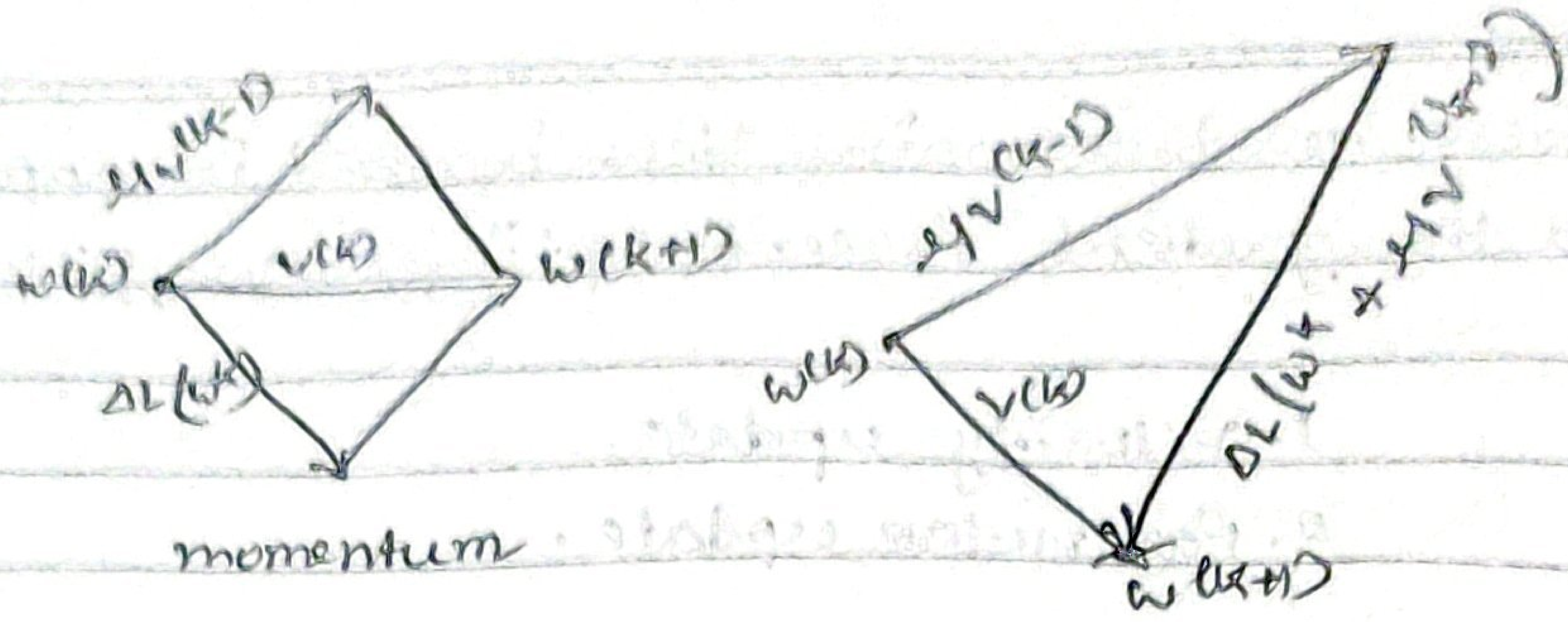
1. Velocity update: $v^{(k)} = \gamma v^{(k-1)} - \eta \nabla L(w^{(k)})$

$v^{(k)}$ → represent the velocity at iteration k
γ → is the momentum coefficient.
η → is the learning rate.
$\nabla L(w^{k})$ → is the gradient of the loss function with respect w at iteration k
$\gamma v^{(k-1)}$ → $\gamma = 0.9$ to 0.99
$- \eta \nabla L(w^{(k)})$ → current gradient scaled by the learning rate η.

Parameter update: $w^{(k+1)} = w^{(k)} + v^{(k)}$

→ $w^{(k+1)}$ → the new parameters for the next iteration
→ w^{k} → The current parameter.
→ wk → The updated velocity

Nesterov Accelerated Gradient / Nesterov momentum in ML



Traditional momentum: uses the current gradient at the current position $w^{(k)}$ along with an accumulated momentum term derived from past gradient.

Nesterov momentum: Computes the gradient at a future anticipated position $w^{(k)} + \mu v^{(k-1)}$, providing a more forwarded-looking approach that can lead to better convergence properties.

Works on future position:
The updated equations are.

$$v^{(k)} = \mu v^{(k-1)} - \eta \nabla L(w^{(k)} + \mu v^{(k-1)})$$
$$w^{(k+1)} = w^{(k)} + v^{(k)}$$

The gradient $\nabla L(w^{(k)} + \mu v^{(k-1)})$ is computed at the anticipated future position $w^{(k)} + \mu v^{(k-1)}$