Deep Learning

Optimizers

Gananath Bhuyan
Assistant Professor
School of Computer Engineering
KIIT DU, Bhubaneswar

What are Optimizers in Deep Learning?

- In deep learning, an optimizer is a crucial element that fine-tunes a neural network's parameters during training. Its primary role is to minimize the model's error or loss function, enhancing performance.
- These specialized algorithms facilitate the learning process of neural networks by iteratively refining the weights and biases based on the feedback received from the data.
- Eg: **Stochastic Gradient Descent (SGD)**, **Mini-Batch Gradient Descent**, **Adam**, AdaDelta, and RMSprop, etc., each equipped with distinct update rules, learning rates, and momentum strategies.

Choosing the Right Optimizer

- **Epoch** – The number of times the algorithm runs on the whole training dataset.
- **Sample** – A single row of a dataset.
- **Batch** – It denotes the number of samples to be taken to for updating the model parameters.
- **Learning rate** – It is a parameter that provides the model a scale of how much model weights should be updated.
- **Cost Function/Loss Function** – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.
- **Weights/ Bias** – The learnable parameters in a model that controls the signal between two neurons.

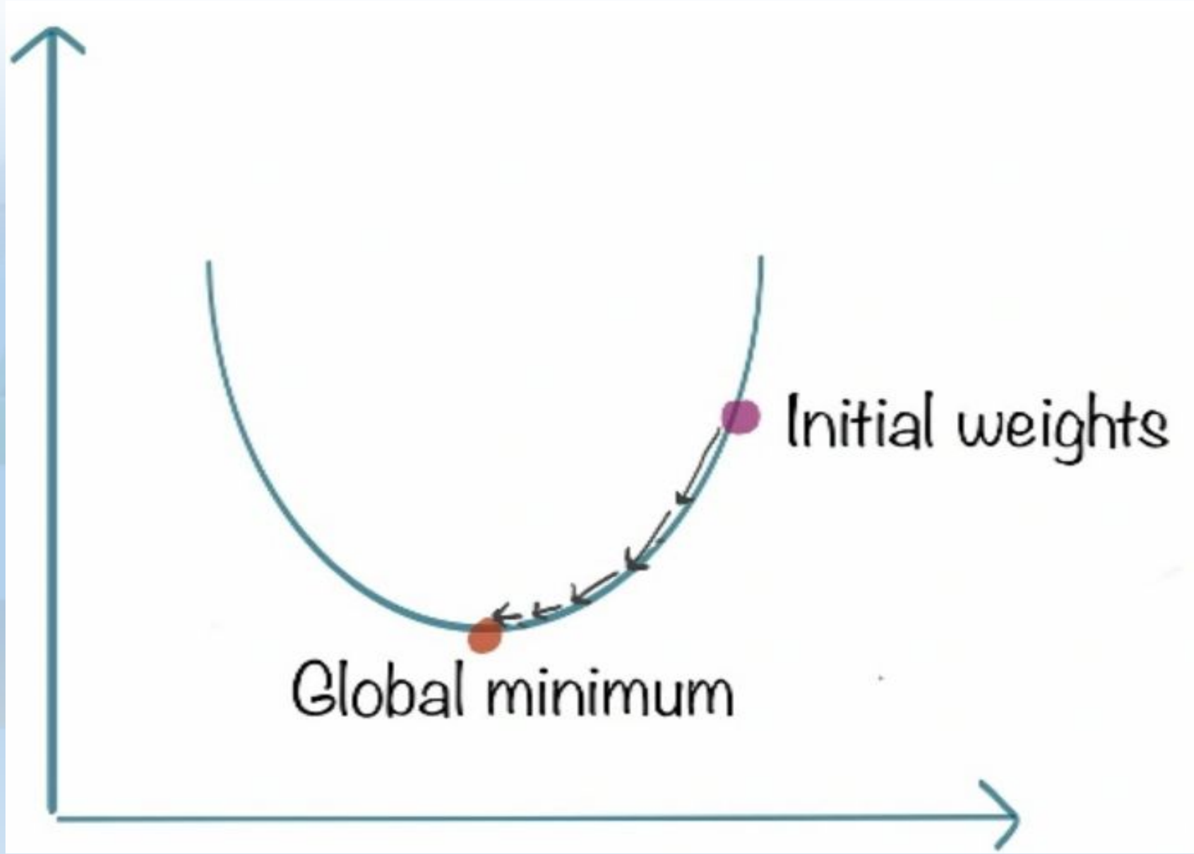
Gradient Descent

- This optimization algorithm uses calculus to consistently modify the values and achieve the local minimum.

1. Initialize Coefficients: Start with initial coefficients.
2. Evaluate Cost: Calculate the cost associated with these coefficients.
3. Search for Lower Cost: Look for a cost value lower than the current one.
4. Update Coefficients: Move towards the lower cost by updating the coefficients' values.
5. Repeat Process: Continue this process iteratively.
6. Reach Local Minimum: Stop when a local minimum is reached, where further cost reduction is not possible.

Gradient Descent

- It is expensive to calculate the gradients if the size of the data is huge.
- Gradient descent works well for convex functions, but it doesn’t know how far to travel along the gradient for nonconvex functions.



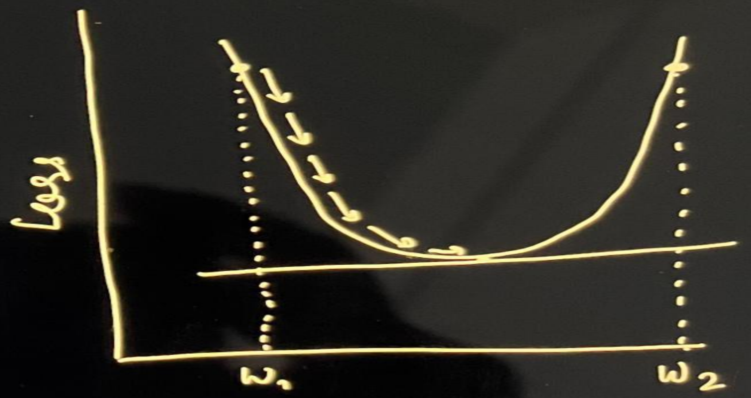
Stochastic Gradient Descent

Back

Aa

Suggested Title: New Note Edit
29 July 2025 at 12:12 AM

$$\boxed{w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}}$$



* All the 'n' data points.

=> Gradient Descent.

$$\boxed{Loss = \frac{1}{n}\sum_{i=1}^{n}(y - y')^2}$$

* Only 1 point at a time

=> Stocastic Gradient Descent (SGD)

$$\boxed{Loss = (y - y')^2}$$

Stochastic Gradient Descent

* A sample of 'K' data points.

=> Mini Batch SGD.

$$\boxed{\text{Loss} = \frac{1}{K}\sum_{i=1}^{K}(y-y')^2}$$

Stochastic Gradient Descent

$$\left[\frac{\partial L}{\partial W_{old}}\right]_{MBSGD} \approx \left[\frac{\partial L}{\partial W_{old}}\right]_{GD}$$

Sample. Population

* The zig-zag movement in SGD is due to the noise. which can be dealt with SGD with momentum.

Stochastic Gradient Descent

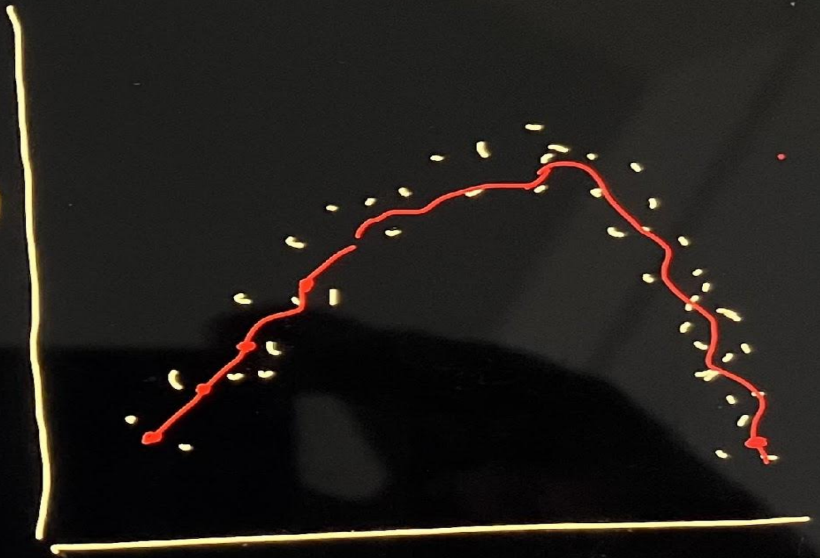
* Exponential weighted moving average :

$$\boxed{V_t = \beta * V_{t-1} + (1-\beta)\,\theta_t}$$

V_t : Current E.W.M.A. (timestamp)

V_{t-1} : previous E.W.M.A.

θ_t : Current data point



$$V_0 = 0$$

$$V_1 = \cancel{\beta \cdot V_0}^{\,0} + (1-\beta)\,\theta_1$$

$$= (1-\beta)\,\theta_1$$

Stochastic Gradient Descent

$$v_2 = \beta \cdot v_1 + (1-\beta)\,\theta_2$$

$$= \beta \cdot (1-\beta)\,\theta_1 + (1-\beta)\,\theta_2$$

$$= (1-\beta)\,(\beta \cdot \theta_1 + \theta_2)$$

$$v_3 = \beta \cdot v_2 + (1-\beta)\,\theta_3$$

$$= \beta \cdot (1-\beta)\,(\beta \cdot \theta_1 + \theta_2) + (1-\beta)\,\theta_3$$

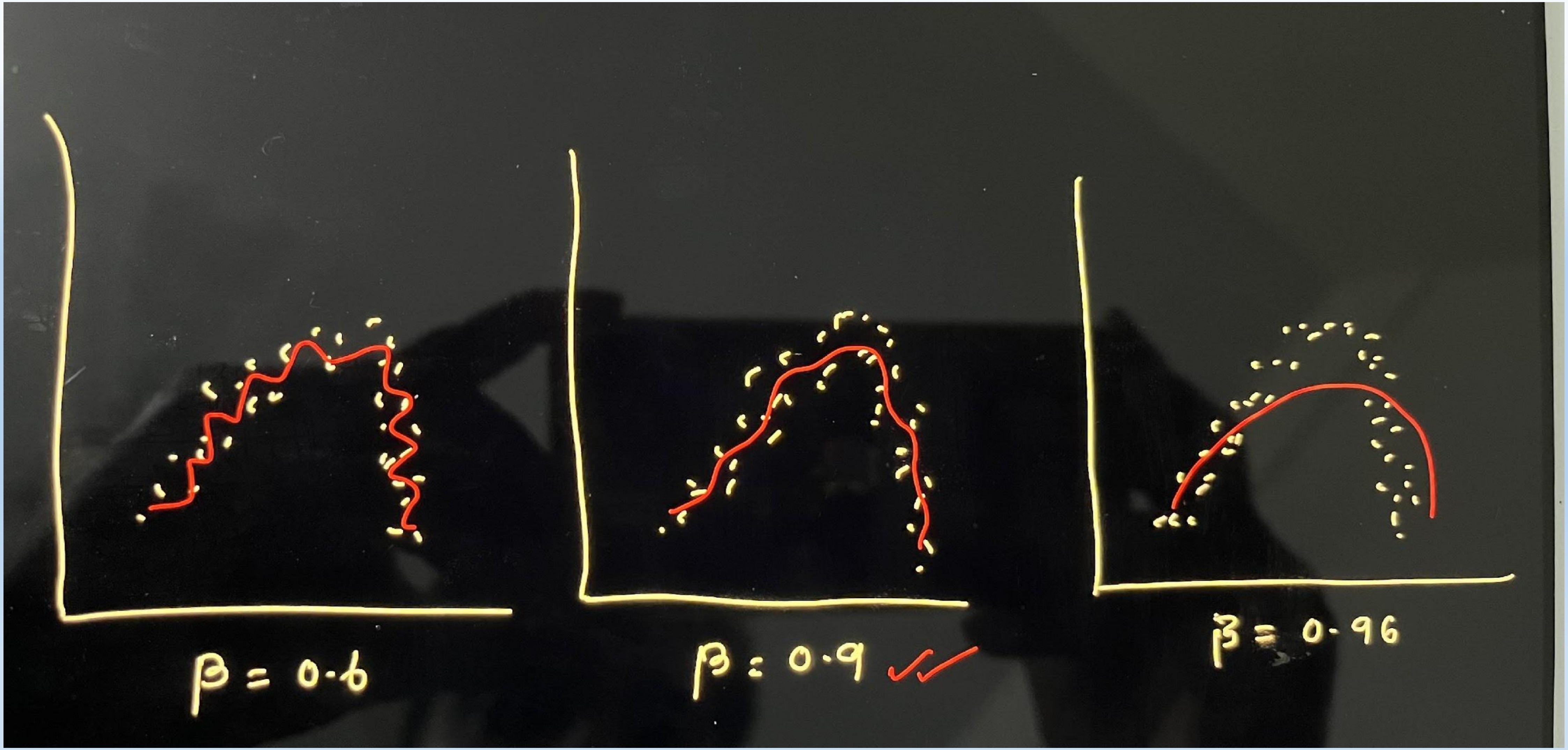
$$= (1-\beta)\,(\beta^2 \theta_1 + \beta \theta_2 + \theta_3)$$

$$\vdots$$

$$\boxed{v_n = (1-\beta)\,(\beta^{n-1} \theta_1 + \beta^{n-2} \theta_2 + \dots + \beta \theta_{n-1} + \theta_n)}$$

$$0 < \beta < 1$$

Stochastic Gradient Descent



Stochastic Gradient Descent