UNIT - 4

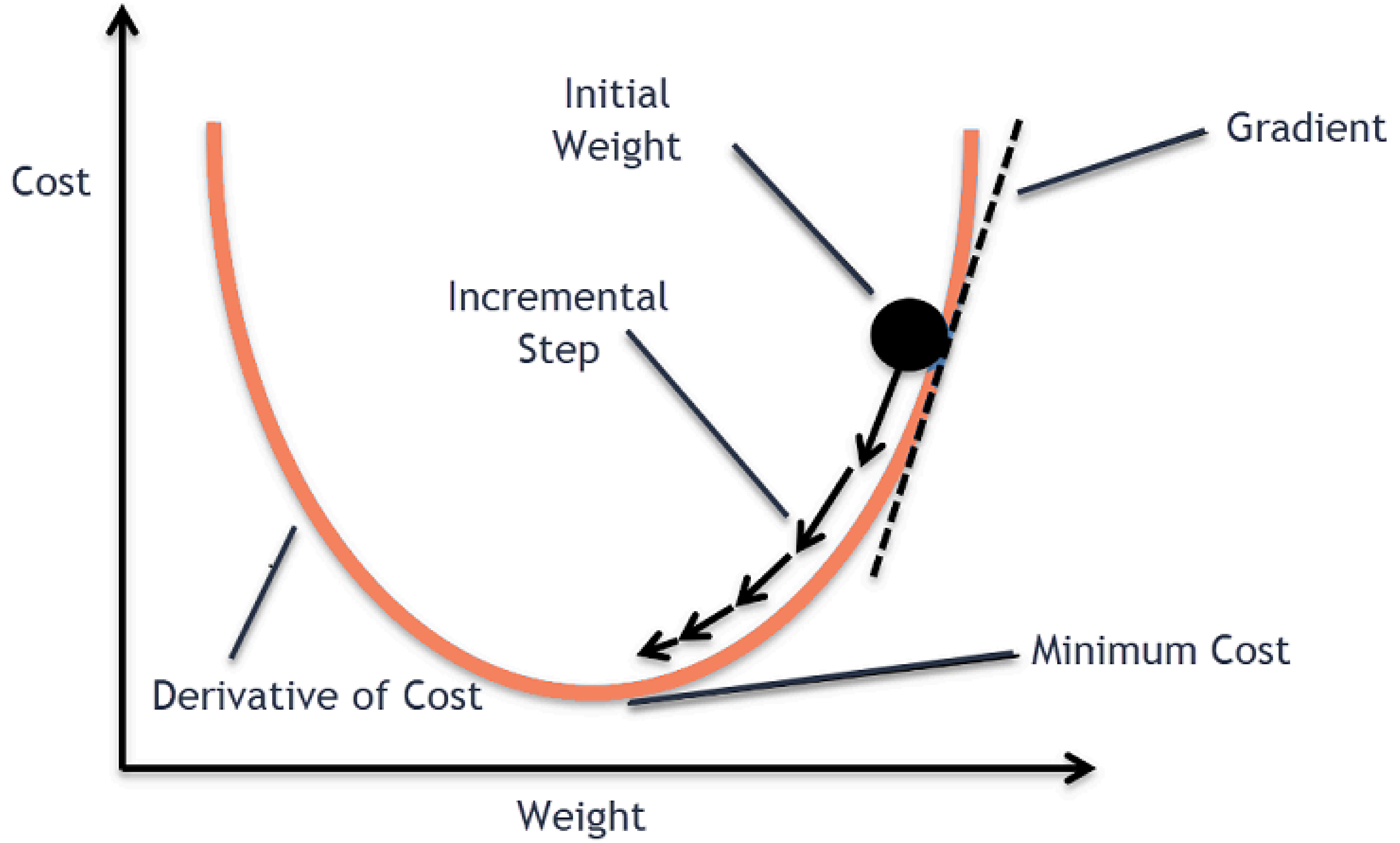
NEURAL NETWORKS

UNIT IV NEURAL NETWORKS 9

Multilayer perceptron, activation functions, network training – gradient descent optimization – Stochastic gradient descent, error backpropagation, from shallow networks to deep networks – Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout

Gradient Descent Optimization

- Gradient descent is an optimization algorithm used to train machine learning models and neural networks by minimizing errors between predicted and actual results (Cost Function).
- It is used to minimize the cost function by iteratively adjusting parameters (Weights & bias) in the direction of the negative gradient, aiming to find the optimal set of parameters.


Initial Weight
Gradient
Cost
Incremental Step
Minimum Cost
Derivative of Cost
Weight

Gradient Descent Optimization

- Before we dive into gradient descent, let us review linear regression. The formula for the slope of a line is y = mx + b, where m represents the slope and b is the intercept on the y-axis.
- Finding the line of best fit, which required calculating the error between the actual output and the predicted output (y-hat) using the mean squared error formula.
- The gradient descent algorithm behaves similarly, but it is based on a convex function.

Gradient Descent Optimization

- The algorithm operates by calculating the gradient of the cost function, which indicates the direction and magnitude of the steepest ascent.
- However, since the objective is to minimize the cost function, gradient descent moves in the opposite direction of the gradient, known as the negative gradient direction.
- By iteratively updating the model’s parameters in the negative gradient direction, gradient descent gradually converges towards the optimal set of parameters that yields the lowest cost.

Gradient Descent Optimization

- Let’s say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?
- The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

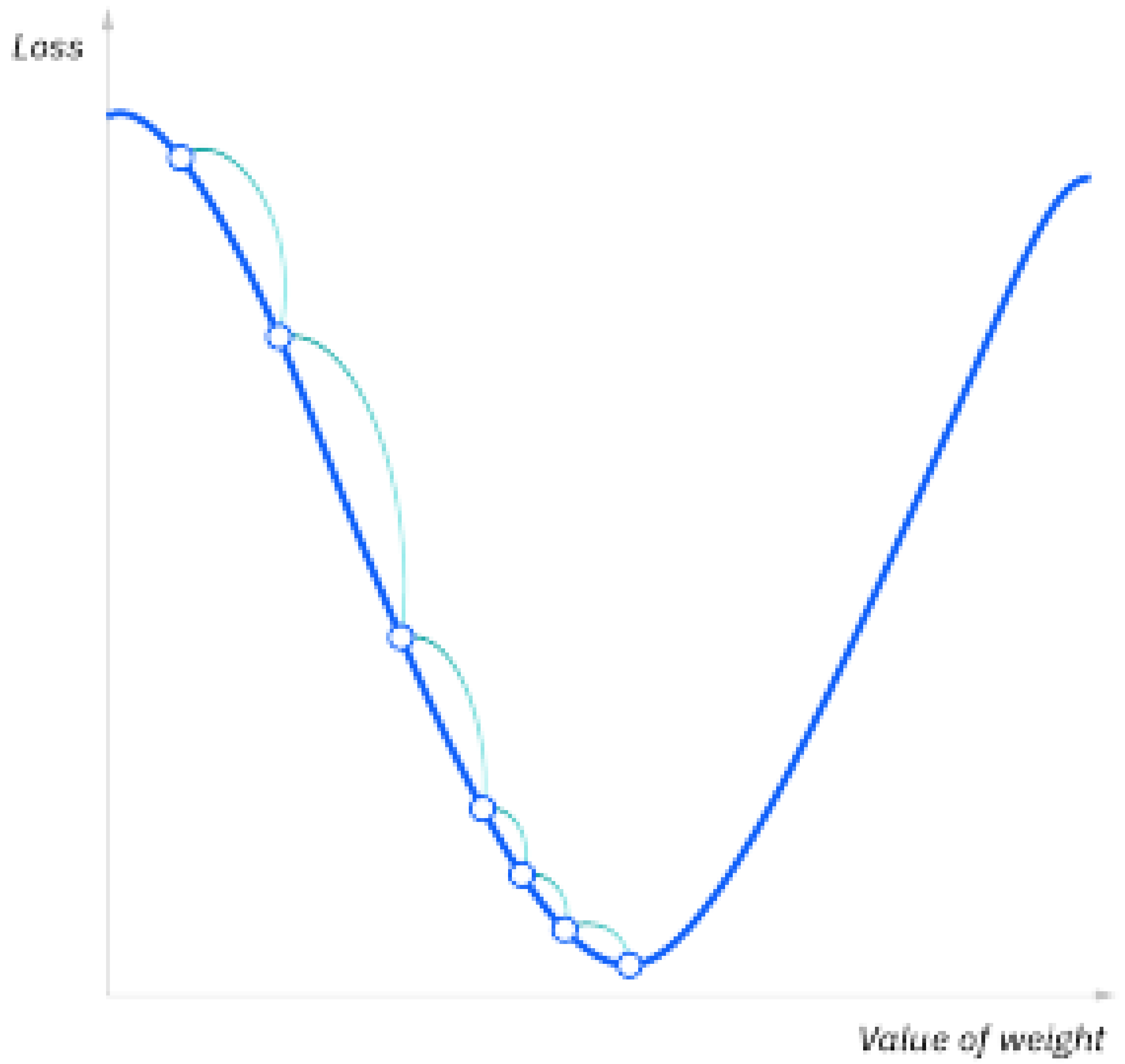
Gradient Descent Optimization

- The goal of the gradient descent algorithm is to minimize the given function (say cost function).
- To achieve this goal, it performs two steps iteratively:
- Compute the gradient (slope), the first-order derivative of the function at that point
- Make a step (move) in the direction opposite to the gradient, opposite direction of slope increase from the current point by alpha (Learning rate) times the gradient at that point

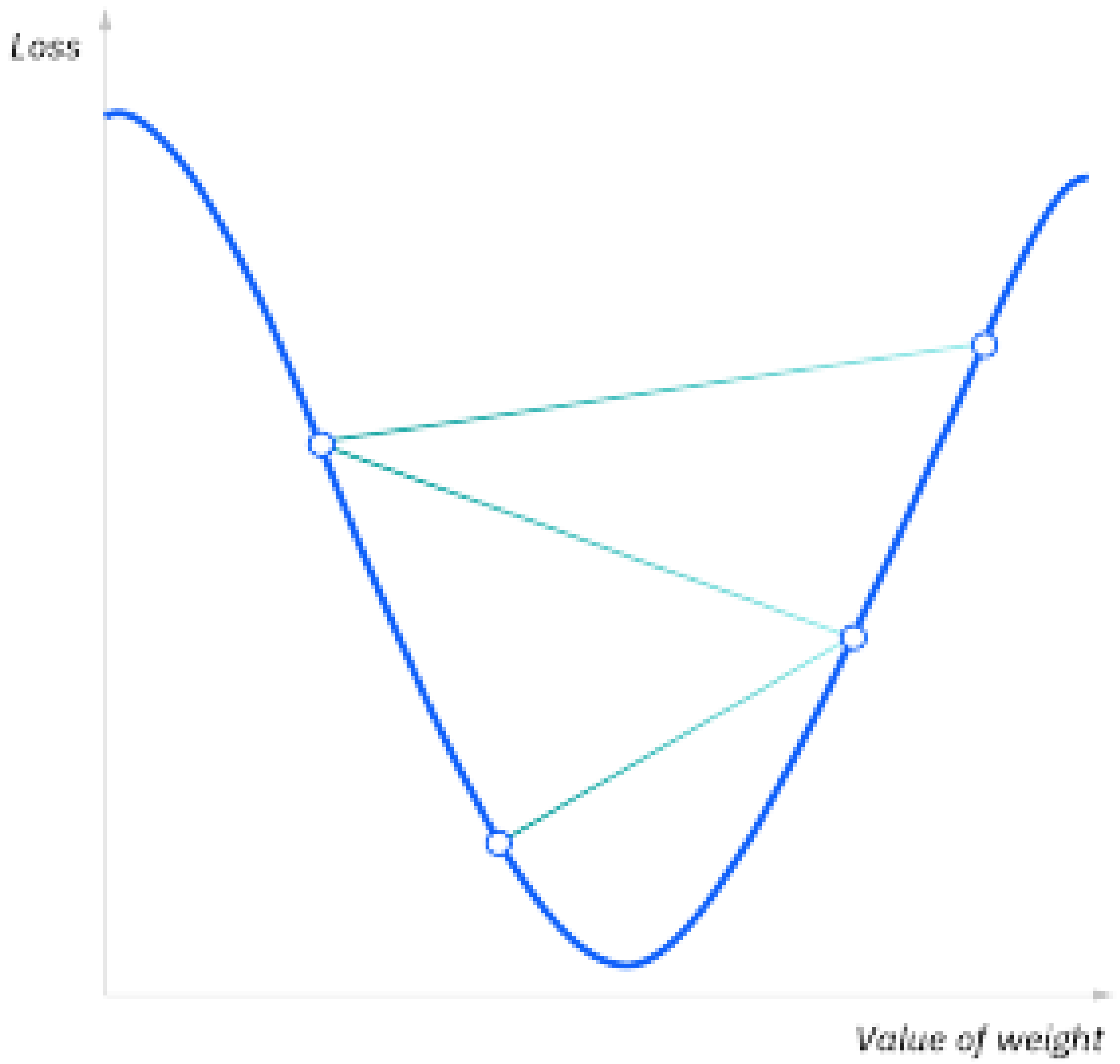
Learning Rate

- The learning rate, a hyperparameter, determines the step size (length of the steps) taken in each iteration, influencing the speed and stability of convergence.
- High learning rates result in larger steps but risks overshooting the minimum.
- Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

Small learning rate



Large learning rate



Gradient Descent

```
Repeat until converge {
        w = w - $\alpha \left[ \frac{\partial Loss}{\partial w} \right]$
        b = b - $\alpha \left[ \frac{\partial Loss}{\partial b} \right]$
}
```

Types of Gradient Descent

- ✓ Batch Gradient Descent
- ✓ Stochastic Gradient Descent
- ✓ Mini-batch Gradient Descent

Batch Gradient Descent

- Batch gradient descent updates the model’s parameters using the gradient of the entire training set.
- It calculates the average gradient of the cost function for all the training examples and updates the parameters in the opposite direction.
- Batch gradient descent guarantees convergence to the global minimum, but can be computationally expensive and slow for large datasets.

Stochastic Gradient Descent

- Stochastic gradient descent updates the model’s parameters using the gradient of one training example at a time.
- It randomly selects a training example, computes the gradient of the cost function for that example, and updates the parameters in the opposite direction.
- Stochastic gradient descent is computationally efficient and can converge faster than batch gradient descent.
- However, it can be noisy and may not converge to the global minimum.

Mini-Batch Gradient Descent

- **Mini-batch gradient descent updates the model’s parameters using the gradient of a small subset of the training set, known as a mini-batch.**
- **It calculates the average gradient of the cost function for the mini-batch and updates the parameters in the opposite direction.**
- **Mini-batch gradient descent algorithm combines the advantages of both batch and stochastic gradient descent, and is the most commonly used method in practice.**
- **It is computationally efficient and less noisy than stochastic gradient descent, while still being able to converge to a good solution.**

ERROR BACKPROPAGATION

Error Propagation

- Error Backpropagation or simply Backprop (BP) is the algorithm used along with an optimization algorithm such as Gradient Descent (GD) to learn the parameters of an NN model.
- BP produces gradients which are then used in optimization.
- The BP stage has the following steps
    - *Evaluate the error signal for each layer*
    - *Use the error signal to compute error gradients*
    - *Update layer parameters using the error gradients with an optimization algorithm such as GD.*
- Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network.

Understanding BackPropagation

- This gradient is then used by an optimization algorithm, such as GD, to adjust the weights to minimize the loss.
- Backpropagation is the chain rule from calculus, which is used to calculate the partial derivatives of the loss function with respect to each weight in the network.
- The term "backpropagation" reflects the way these derivatives are computed backward, from the output layer to the input layer.
- The process involves two main phases: a forward pass, where the input data is passed through the network to compute the output, and a backward pass, where the gradients are computed by propagating the error backward through the network.

Forward Pass

- In the forward pass, the input data is fed into the network, and operations defined by the network architecture are performed layer by layer to compute the output.
- This output is then used to calculate the loss, which measures the difference between the network's prediction and the true target values.

Backward Pass

- The backward pass starts with the computation of the gradient of the loss function with respect to the output of the network.
- Then, by applying the chain rule, the algorithm calculates the gradient of the loss with respect to each weight by propagating the error information back from the output layer to the input layer. This process involves the following steps:
- Compute the derivative of the loss function with respect to the activations of the output layer.

Backward Pass

- For each layer, starting from the last hidden layer and moving to the first, compute the gradients of the loss with respect to the layer's inputs, which are the activations of the previous layer.
- Compute the gradients of the loss with respect to the weights by considering the gradients with respect to the layer's inputs and the derivatives of the layer's activations with respect to its weights.
- These gradients tell us how much a change in each weight would affect the loss, allowing the optimization algorithm to adjust the weights in a direction that minimizes the loss.

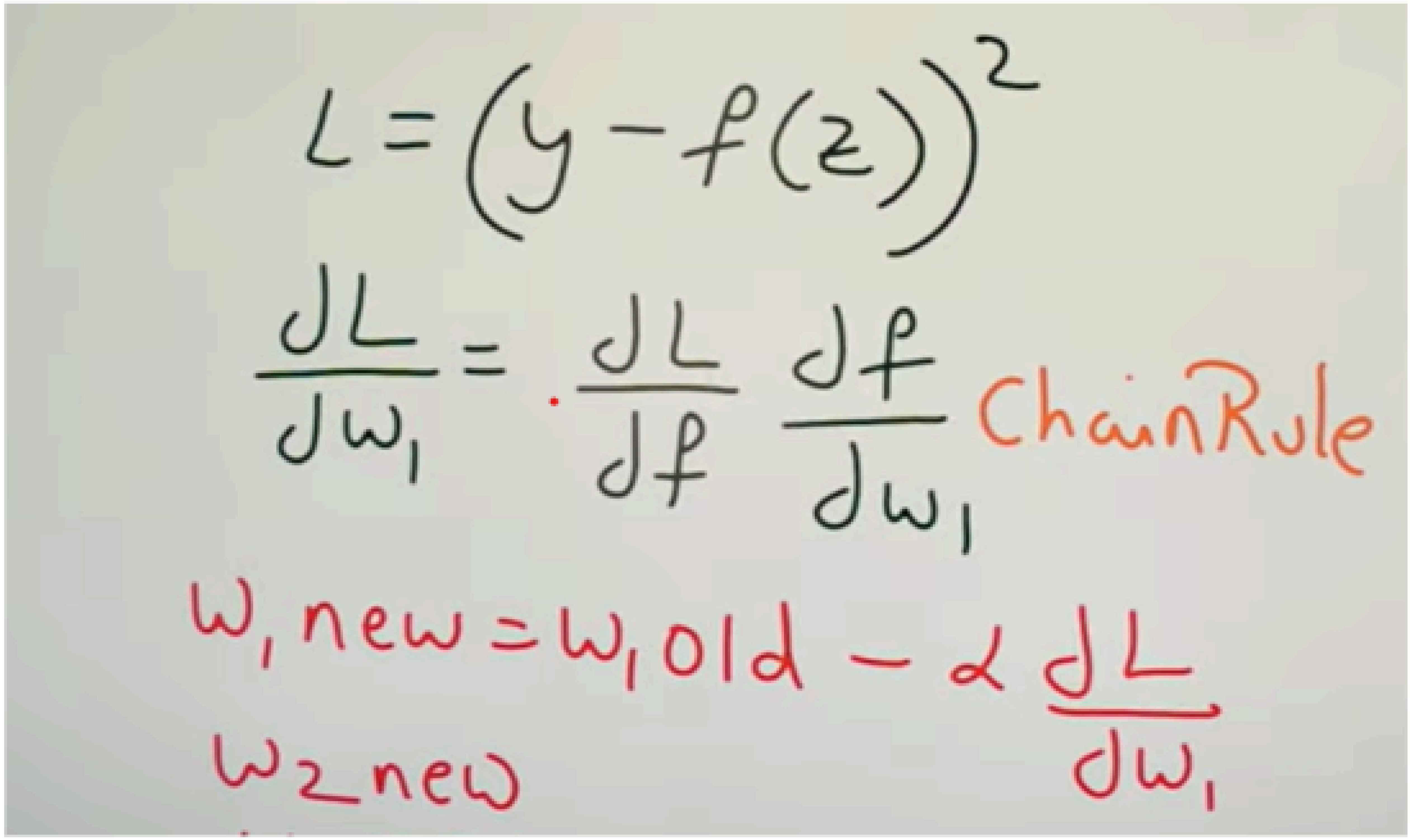
Updating Weights

- **Once the gradients are computed, the weights are updated typically using a gradient descent optimization algorithm. The weight update rule is generally of the form:**

  new_weight = old_weight - learning_rate * gradient

- **where the learning rate is a <u>hyperparameter</u> that controls the size of the weight updates.**

- Updating weights, w values – Back propagation
-



$$L=(y-f(z))^2$$

$$\frac{dL}{dw_1}=\frac{dL}{df}\frac{df}{dw_1}$$ ChainRule

$$w_1\text{new}=w_1\text{old}-\alpha\frac{dL}{dw_1}$$

$w_2\text{new}$

Challenges with Backpropagation

- **Vanishing Gradients:** In deep networks, the gradients can become very small or even approximate to zero, effectively preventing the weights in the earlier layers from changing significantly. This can slow down or even halt training.
- **Exploding Gradients:** Conversely, gradients can also grow exponentially, causing large weight updates that can destabilize the learning process.
- **Non-Convex Loss Functions:** Neural networks typically have non-convex loss functions, which means there can be multiple local minima. Gradient descent methods can get stuck in these local minima instead of finding the global minimum.

Improvements and Variants

- **To address these challenges, various improvements and variants of the backpropagation algorithm have been developed:**
- **Activation Functions:**
  - **Functions like ReLU and its variants help mitigate the vanishing gradient problem.**
- **Weight Initialization: Techniques like Xavier and He initialization set the initial weights to values that help prevent vanishing or exploding gradients.**
- **Gradient Clipping:**
  - **This technique prevents gradients from becoming too large, addressing the exploding gradient problem.**
- **Optimization Algorithms:**
  - **Advanced optimizers like Adam, RMSprop, and AdaGrad adapt the learning rate during training to improve convergence.**