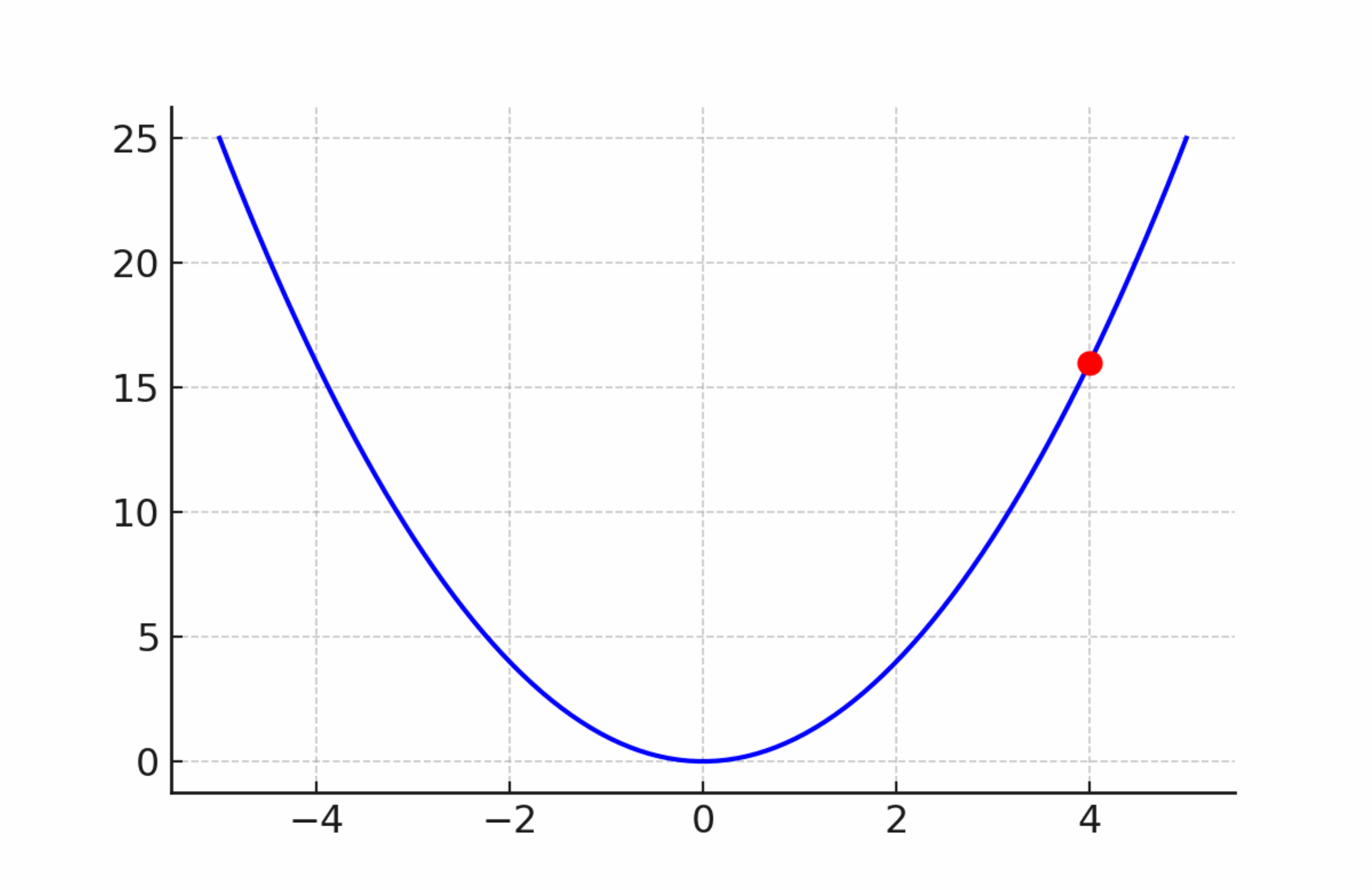
Deep Learning

Lecture 3

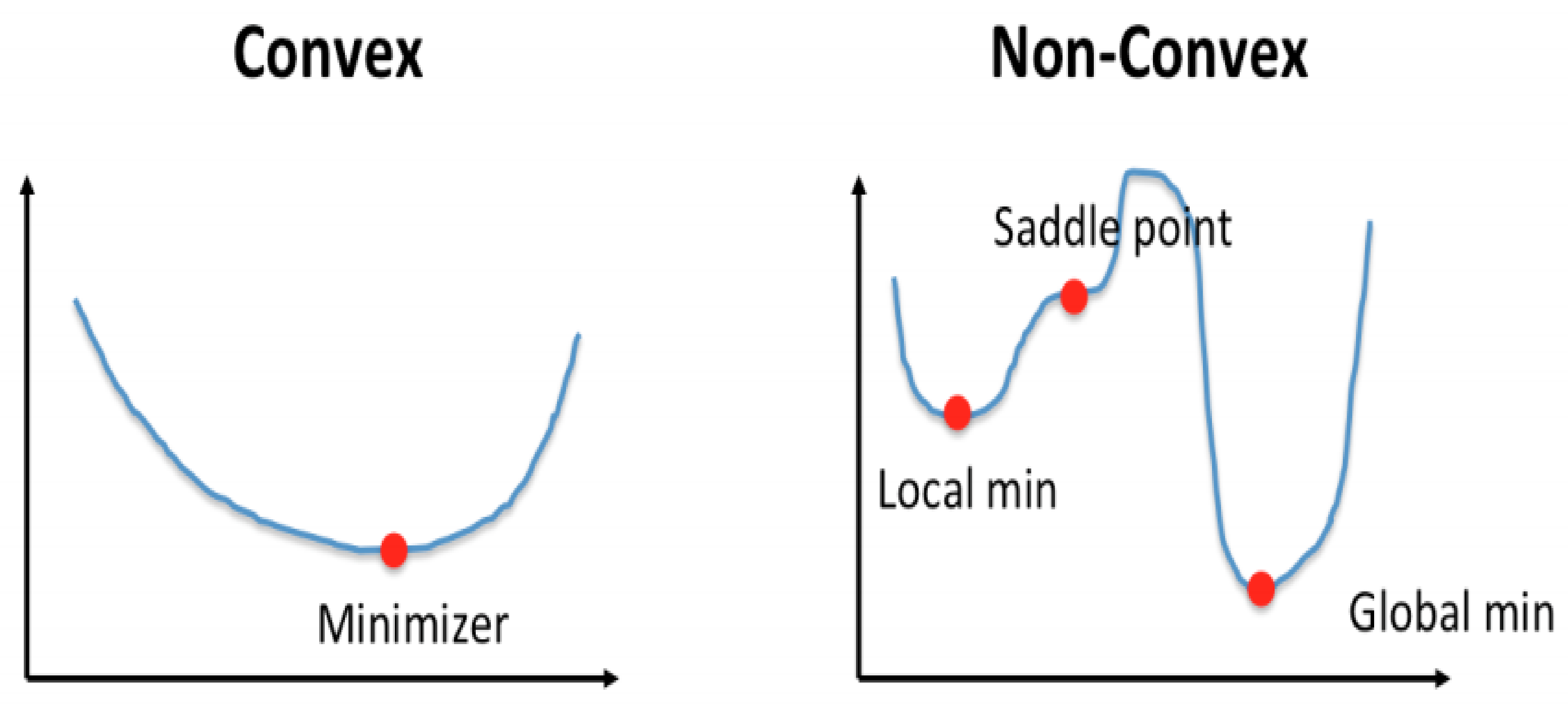
Gradient Descent

- It is an optimization algorithm used in machine learning and deep learning to minimize a cost function by iteratively updating model parameters (weights and biases)
- The goal is to find the best parameters that reduce prediction errors.

Gradient Descent



Convex VS Non-Convex



https://ai.plainenglish.io/navigating-the-terrain-convex-vs-non-convex-functions-in-optimization-86812e9a1989

How It Works (Intuition)

Say you're at the top of a mountain and want to reach the lowest point (global minimum).

- Each step you take is based on the **steepness of the slope (gradient)** at your current position.
- A large step (**high learning rate**) might cause you to jump over the minimum
- A small step (**low learning rate**) will make progress slow but steady

Components of GD

1. **Cost Function ($J(\theta)$)** – Measures the error between predicted and actual values.
2. **Gradient ($\nabla J(\theta)$)** – Direction and magnitude of change in the cost function.
3. **Learning Rate (α)** – Controls the step size for updates.

Mathematical Intuition

1. Cost Function (J(θ))

Gradient Descent optimizes a cost function **J(θ)** to minimize the error. For example, in linear regression:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Mathematical Intuition

where:

- $h_\theta(x) = \theta_0 + \theta_1 x$ (hypothesis/prediction)
- m = number of training samples
- y_i = actual value
- $h_\theta(x_i)$ = predicted value

➨ Our goal is to find **θ (theta values) that minimize J(θ)**.

Example: Finding the Minimum of a Simple Function

- We use the function:

J(θ)=$(\boldsymbol{\theta} - \mathbf{3})^{\mathbf{2}}$

The **global minimum** is at θ=3.

We start with a **random initial θ value.**

We update θ using the **Gradient Descent formula**.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\alpha}\frac{\boldsymbol{dj}}{\boldsymbol{d\theta}}$$

Example: Finding the Minimum of a Simple Function

Where:

$$\frac{dj}{d\theta}=2(\theta - 3)$$

➨Let's assume:

- **Initial θ=−5**
- **Learning rate a=0.1**
- **We run for 5 iterations**

Example: Finding the Minimum of a Simple Function

Iteration	θ	Gradient $2(\theta - 3)$	Update $\theta - 0.1 \times \text{Gradient}$	Cost $(\theta - 3)^2$
0	**-5**	$2(-5-3) = -16$	$-5 - 0.1(-16) = -3.4$	$(-5-3)^2 = 64$
1	**-3.4**	$2(-3.4-3) = -12.8$	$-3.4 - 0.1(-12.8) = -2.12$	$(-3.4-3)^2 = 42.25$
2	**-2.12**	$2(-2.12-3) = -10.24$	$-2.12 - 0.1(-10.24) = -1.096$	$(-2.12-3)^2 = 25.92$
3	**-1.096**	$2(-1.096-3) = -8.192$	$-1.096 - 0.1(-8.192) = -0.2768$	$(-1.096-3)^2 = 16.11$
4	**-0.2768**	$2(-0.2768-3) = -6.5536$	$-0.2768 - 0.1(-6.5536) = 0.37856$	$(-0.2768-3)^2 = 10.65$

Observations

1. **θ moves towards 3** in each iteration.
2. The **gradient gets smaller** as θ approaches the minimum.
3. The **cost decreases** in each step.

Variants of Gradient Descent

1. **Batch Gradient Descent (BGD)** – Uses all training data at once (slow but stable)
2. **Stochastic Gradient Descent (SGD)** – Updates parameters for each data point (faster but noisy)
3. **Mini-Batch Gradient Descent** – A balance between BGD and SGD (uses small batches)

Batch Gradient Descent (BGD)

- Batch Gradient Descent (Slow but Stable)
- For example, you want to **lose weight** and reach your **ideal body weight** of **70 kg.** You **track your weight for a month**, calculate your **average weight loss**, and **then adjust** your diet and exercise. This is **slow but accurate** because you make **big adjustments based on all data at once.**

Stochastic Gradient Descent (Fast but Noisy)

- You **weigh yourself every day** and **immediately adjust** your diet based on **just that day's weight**
- This is **faster**, but sometimes **random daily fluctuations (like eating extra one day)** might cause **overreactions**

Advanced Optimization Algorithms

1. Momentum-Based GD
2. RMSProp (Root Mean Square Propagation)
3. Adam (Adaptive Moment Estimation)

Why Momentum-Based GD

In deep learning, we face non-convex optimization.

Consistent Gradient **Noisy Gradient**

Momentum-Based GD

- Instead of just updating using the current gradient, we **accumulate** past gradients to add momentum.

For example, if you want to move from point A to B, you don't know where destination B is located, you ask four persons, and all of them tell you that B is located north. So, in between you increase your speed towards destination (B) by gaining confidence.

- Faster convergence avoids oscillations.
- Can overshoot if momentum is too high

Momentum-Based GD Mathematics

- Instead of just using the current gradient, Momentum GD also considers the past gradients
- This helps in smoother updates and avoids zig-zagging

$$v_t = \beta v_{t-1} + (1-\beta)\nabla J(\theta)$$

$$\theta = \theta - \eta v_t$$

Momentum-Based GD

$$\theta = \theta - \alpha \cdot \frac{dJ}{d\theta}$$

Standard GD

Momentum-Based GD Mathematics

Where:

- v_t is the velocity (running average of gradients).
- β is a momentum term (usually 0.9).
- η is the learning rate

Example: Ball Rolling Down a Hill

Think of a **ball rolling down a hill**, it gains speed gradually instead of taking small, slow steps. This helps **faster convergence** and reduces oscillations.

RMSProp (Root Mean Square Propagation)

- RMSProp adapts the learning rate for each parameter based on the **past gradients** to avoid **oscillations.**

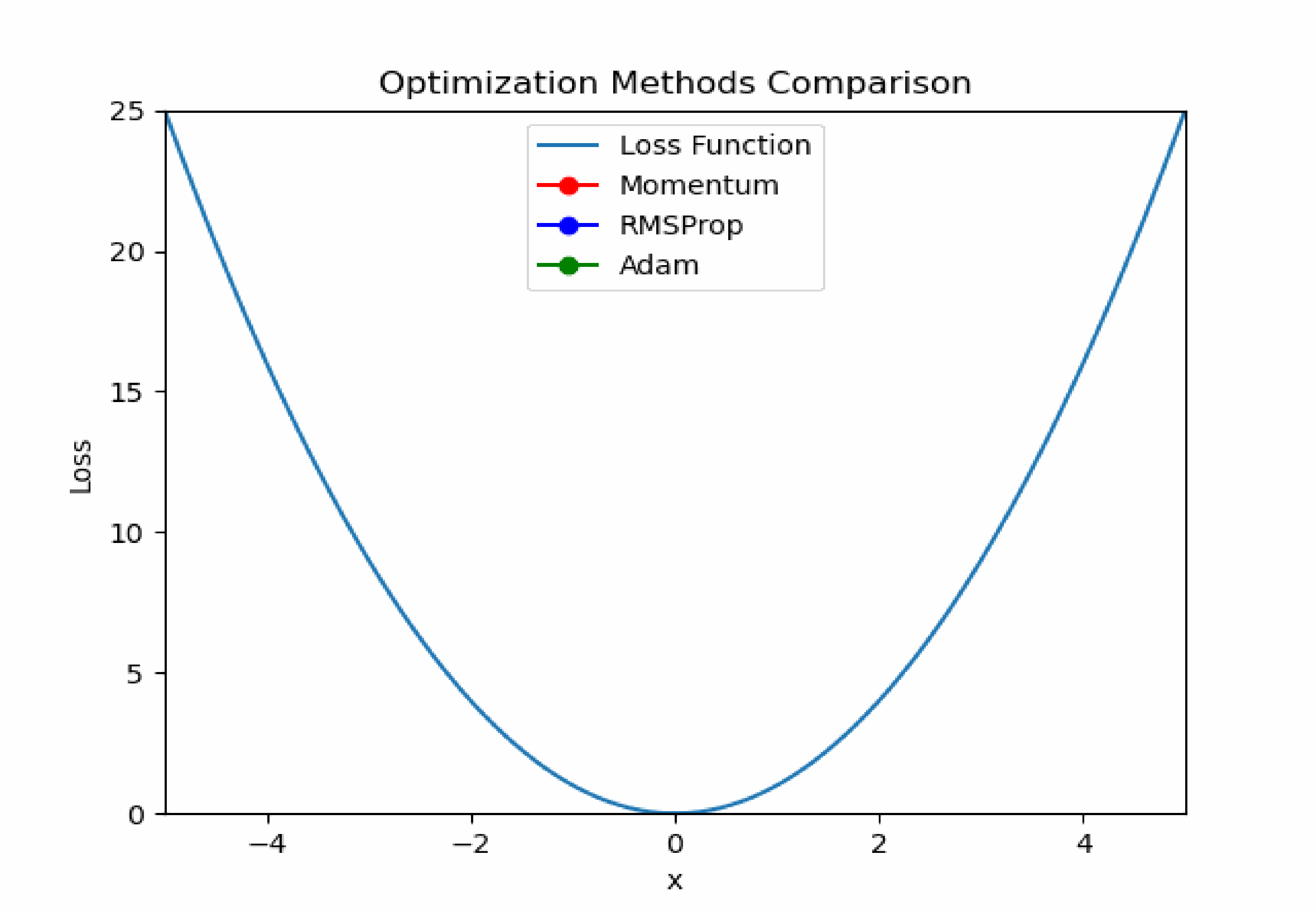
For example, you are **hiking on a terrain surface**. If you step too aggressively, you may fall. Instead, you take **small careful steps** where the ground is unstable (steep gradients) and **bigger steps** where the ground is flat (small gradients).

Adam (Adaptive Moment Estimation)

- Adam **combines Momentum and RMSProp**

Comparisons

Optimizer	Uses Past Gradients?	Adaptive Learning Rate?	Convergence Speed
Gradient Descent	❌ No	❌ No	🚶 Slow
Momentum	✅ Yes	❌ No	🏃 Faster
RMSProp	❌ No	✅ Yes	🏃 Faster
Adam	✅ Yes	✅ Yes	🚀 Fastest


Optimization Methods Comparison
Loss Function
Momentum
RMSProp
Adam
Loss
x
25
20
15
10
5
0
−4
−2
0
2
4

Early Stopping, and Dropout

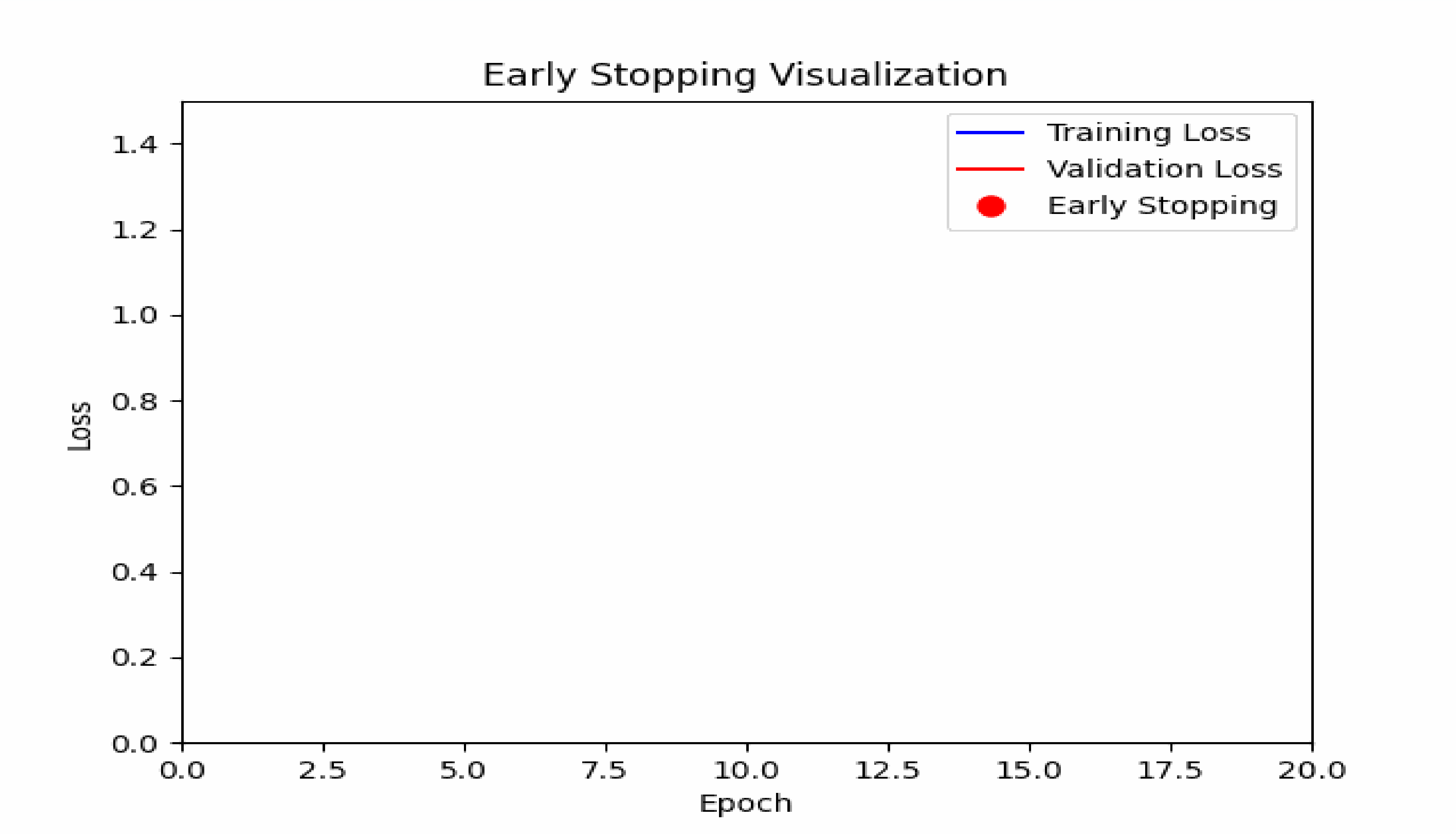
- When training neural networks, we aim to find a model that generalizes well to unseen data
- However, deep networks often suffer from **overfitting**, where they perform well on training data but poorly on test data
- To overcome this, we use **regularization techniques** such as **Early Stopping, and Dropout.**

Why Early Stopping?

- As training progresses, the model starts memorizing the training data rather than learning general patterns
- If we stop too early, the model underfits; if we train too long, it overfits

How does it work

- Monitor the model’s performance on a **validation set**
- If validation loss stops improving for a defined number of epochs, stop training
- Saves computational resources and prevents overfitting.

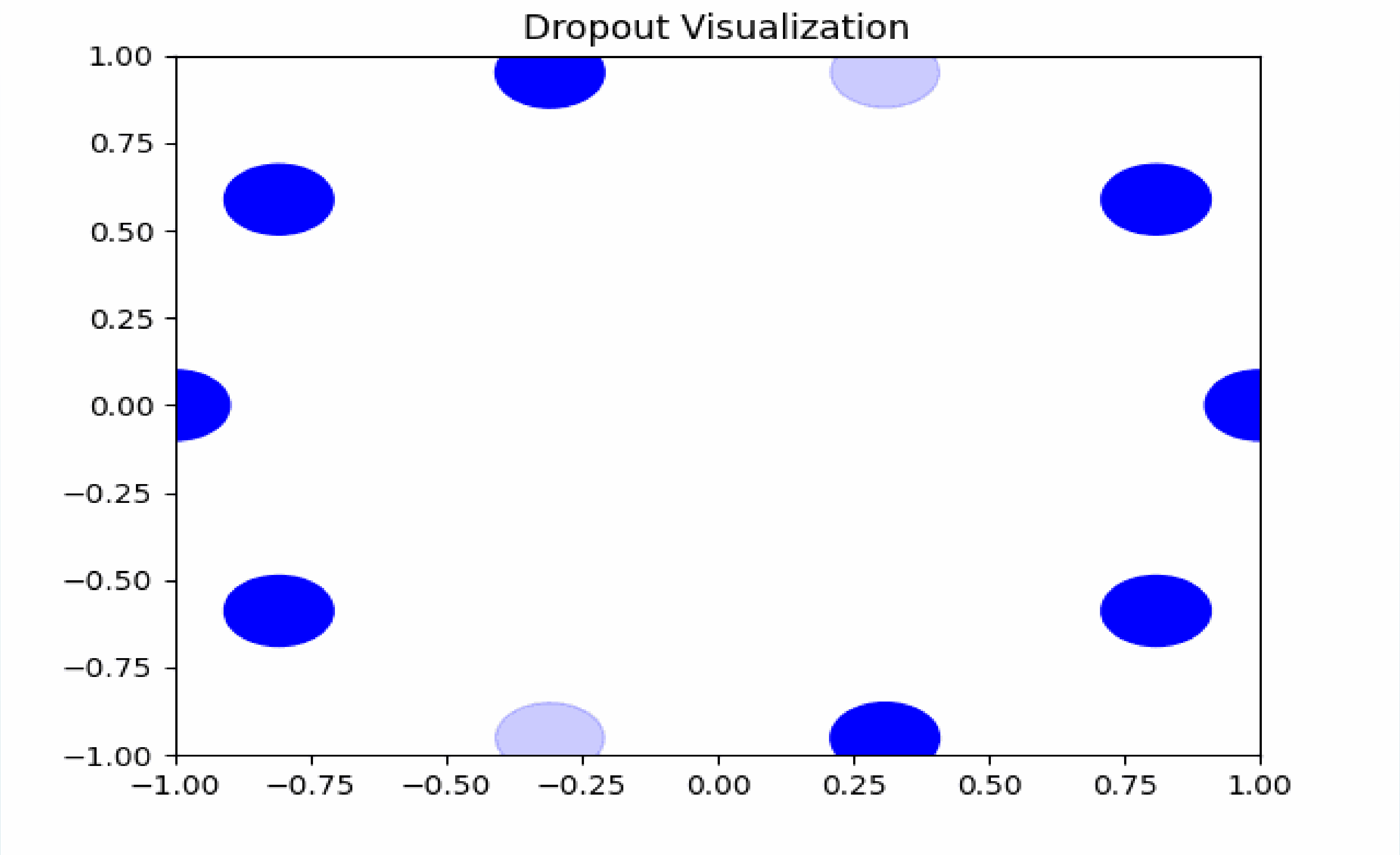

Early Stopping Visualization
Training Loss
Validation Loss
Early Stopping
Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

Dropout

- In deep networks, neurons develop **dependencies** on each other, reducing their ability to generalize.
- Dropout forces the network to **learn redundant and independent features**

How does it work

- Randomly **deactivate** (set to zero) a fraction of neurons during training
- Ensures that the network doesn’t rely on specific neurons too much


Dropout Visualization
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00