# GRADIENT DESCENT

The Key to **Learning** in Neural Networks

# Why Do We Need Gradient Descent?

Neural networks learn by trial and error. They start with random weights and adjust them over time to **reduce mistakes.**

But how do they know what to change and by how much?

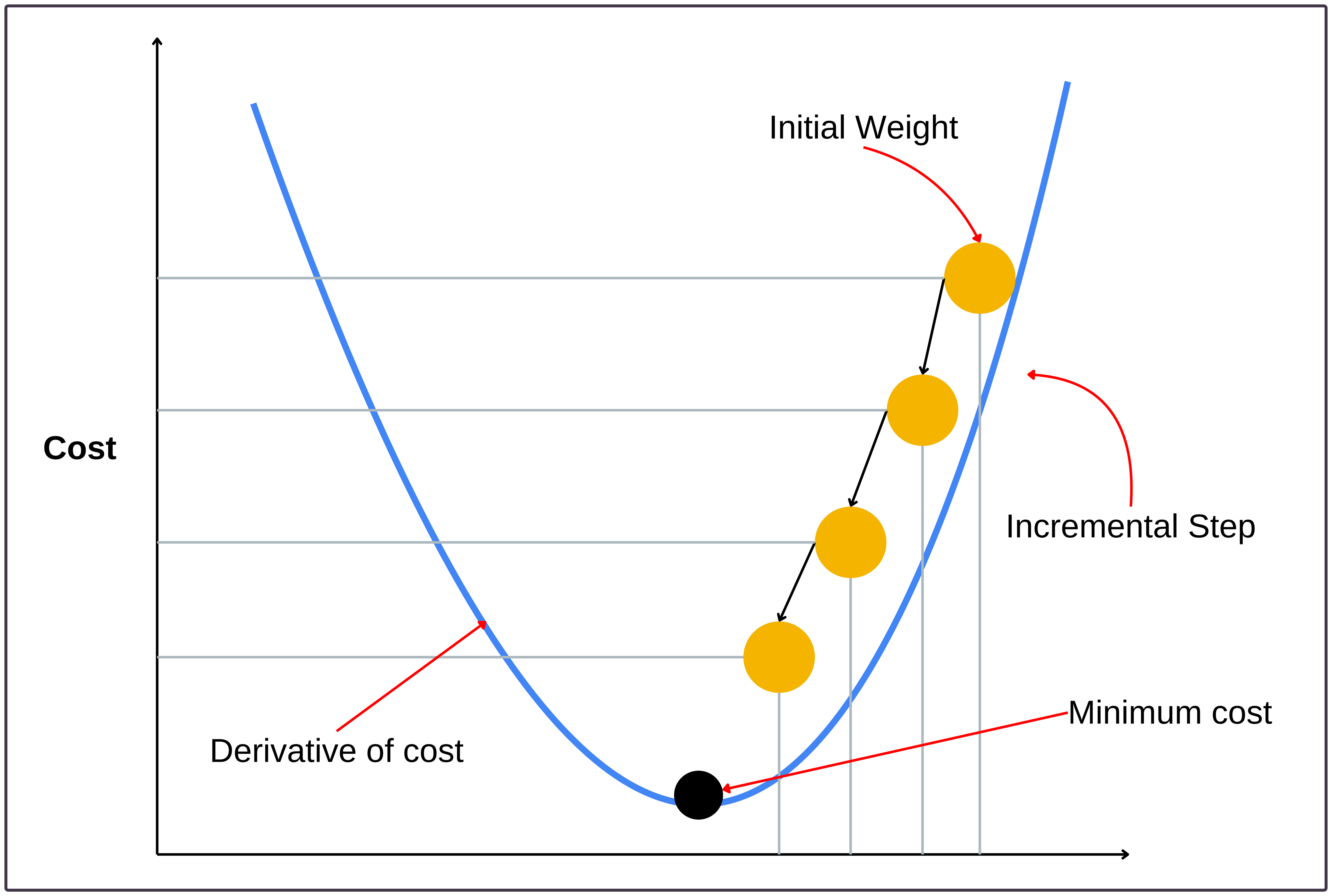
That's where **Gradient Descent** comes in—it's like a **GPS for optimization**, guiding the model toward the best possible predictions.

# The Core Idea Behind Gradient Descent

Imagine you're **hiking down** a **foggy mountain** but can't see the bottom. What do you do?

- Take **small steps downhill** based on the steepness.
- If it's steep, step carefully. If it's flat, adjust slowly.
- Keep moving until you reach the lowest point.

That's exactly how Gradient Descent works—it takes small steps in the direction that reduces error, **refining** the model's predictions.



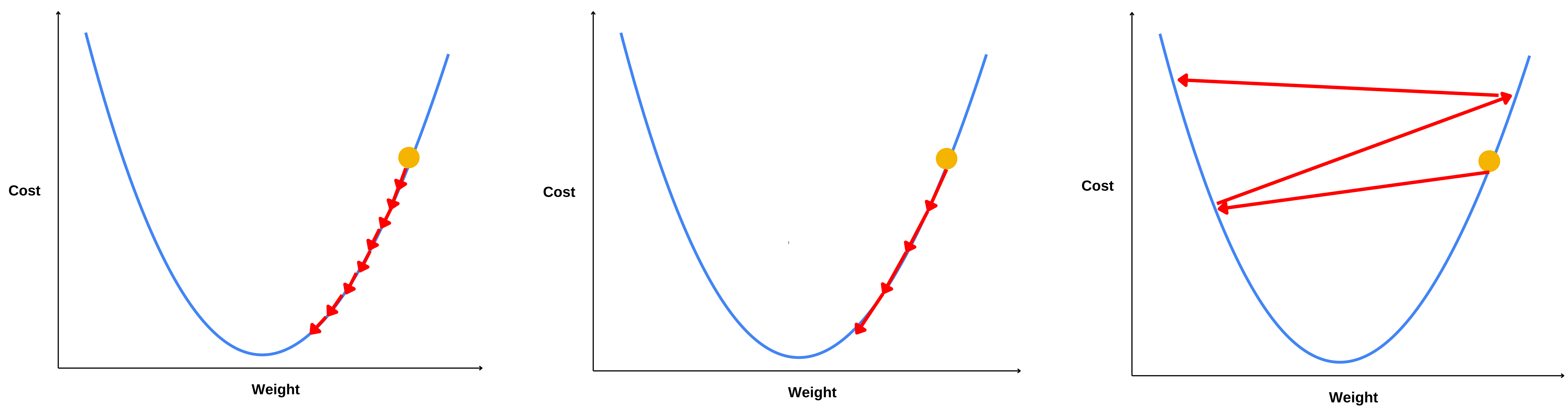

# How Gradient Descent Works?

1. **Calculate the Error** – Measure how far the model's predictions are from the actual values using a loss function.

2. **Find the Gradient** – Compute the slope (direction of steepest decrease)

3. **Adjust Weights** – Take a small step in the opposite direction of the gradient to reduce error.

4. **Repeat Until Convergence** – Keep updating until the model stops improving.




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

The **optimal** learning rate swiftly reaches the optimal point

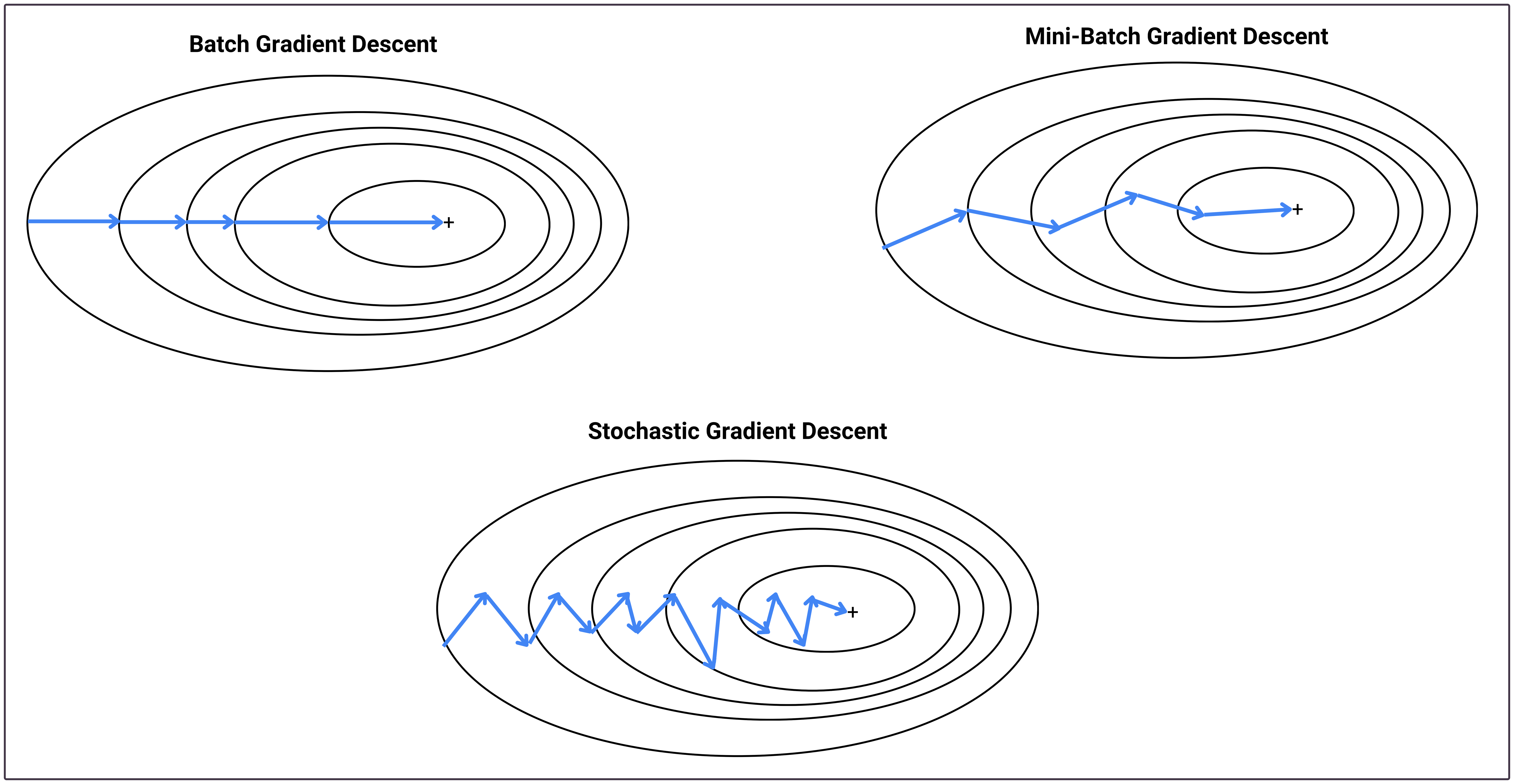
Too **large** of a learning rate causes drastic updates leading to divergent behaviours

# The Learning Rate – A Critical Choice

The **learning rate** determines **how big each step is:**

- **Too small** → Learning is slow and may never converge.
- **Too large** → The model might jump around and never settle.
- **Just right** → The model steadily improves and finds the optimal solution.

Choosing the right learning rate is crucial for efficient training!




# Types of Gradient Descent

There are different ways to apply Gradient Descent:

**1. Batch Gradient Descent**

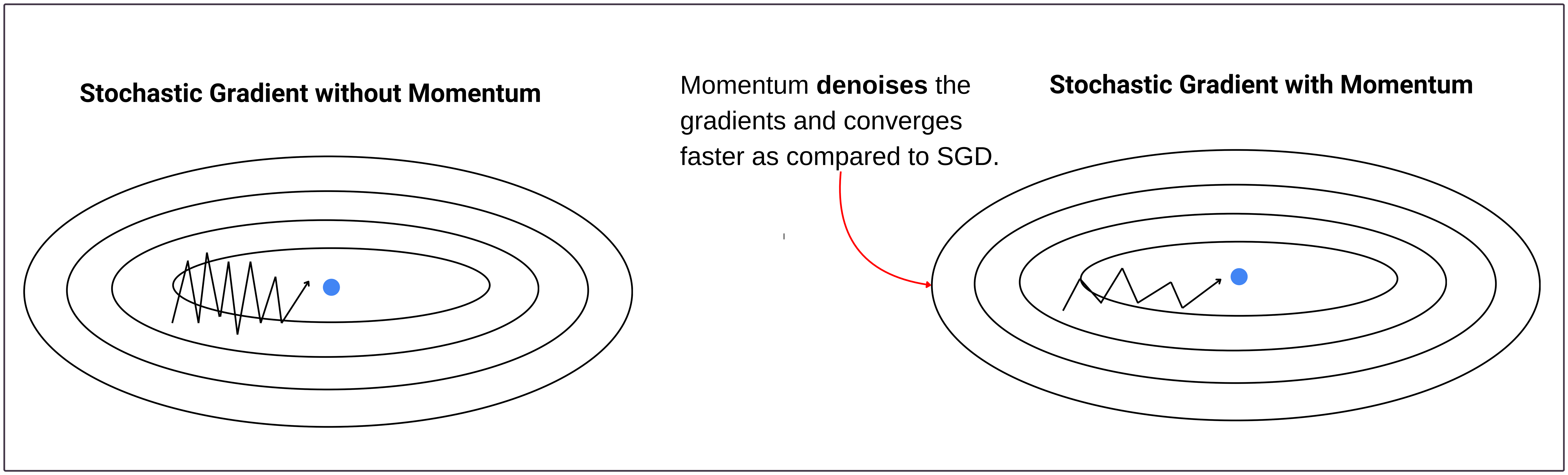
- Uses the entire dataset to compute the gradient.
- Stable but slow for large datasets.

**2. Stochastic Gradient Descent (SGD)**

- Updates weights after each sample instead of the whole dataset.
- Faster but noisier, leading to more fluctuations.

**3. Mini-Batch Gradient Descent**

- A balance between Batch and SGD, updating after a small group of samples.

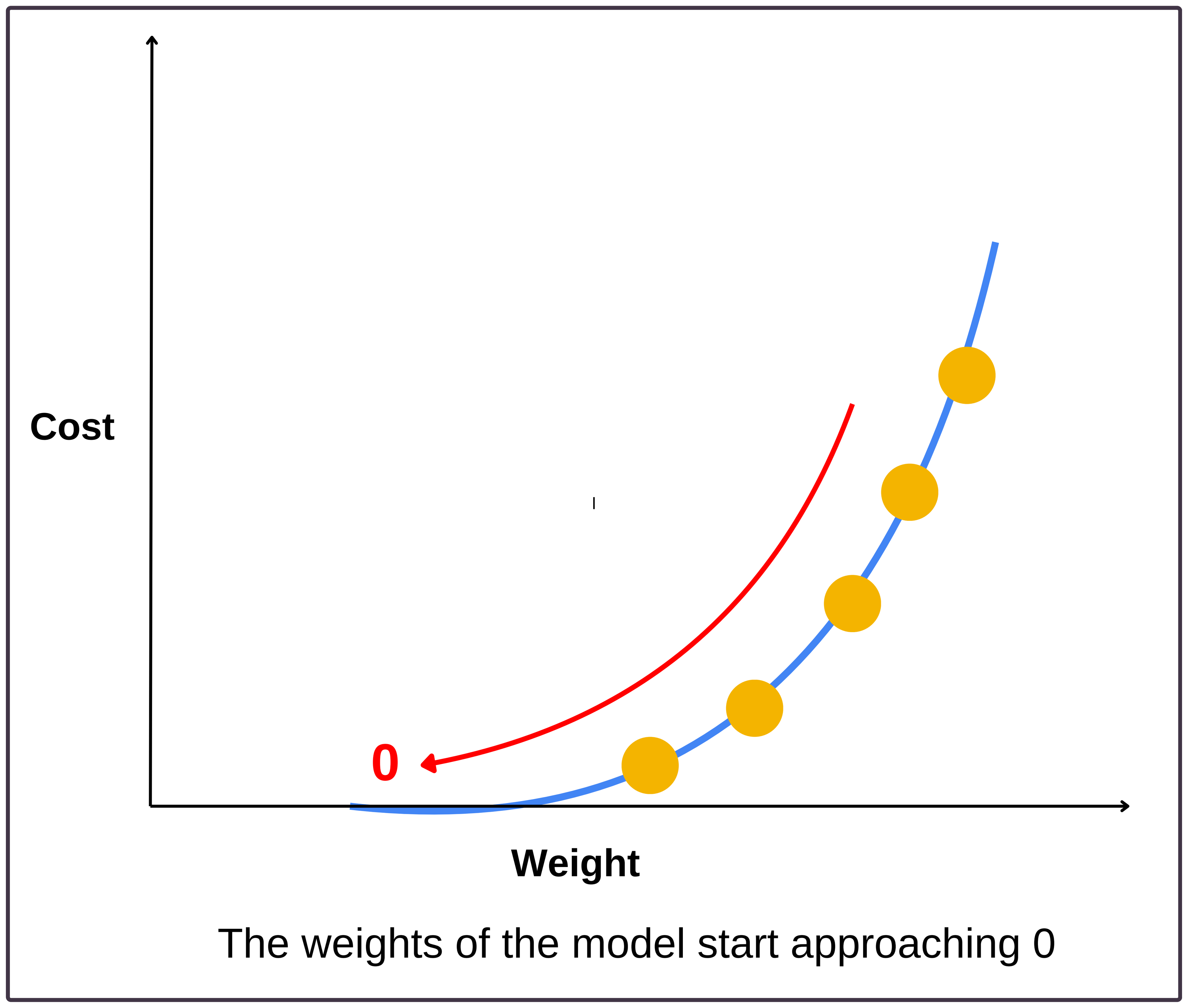



# The Role of Optimizers

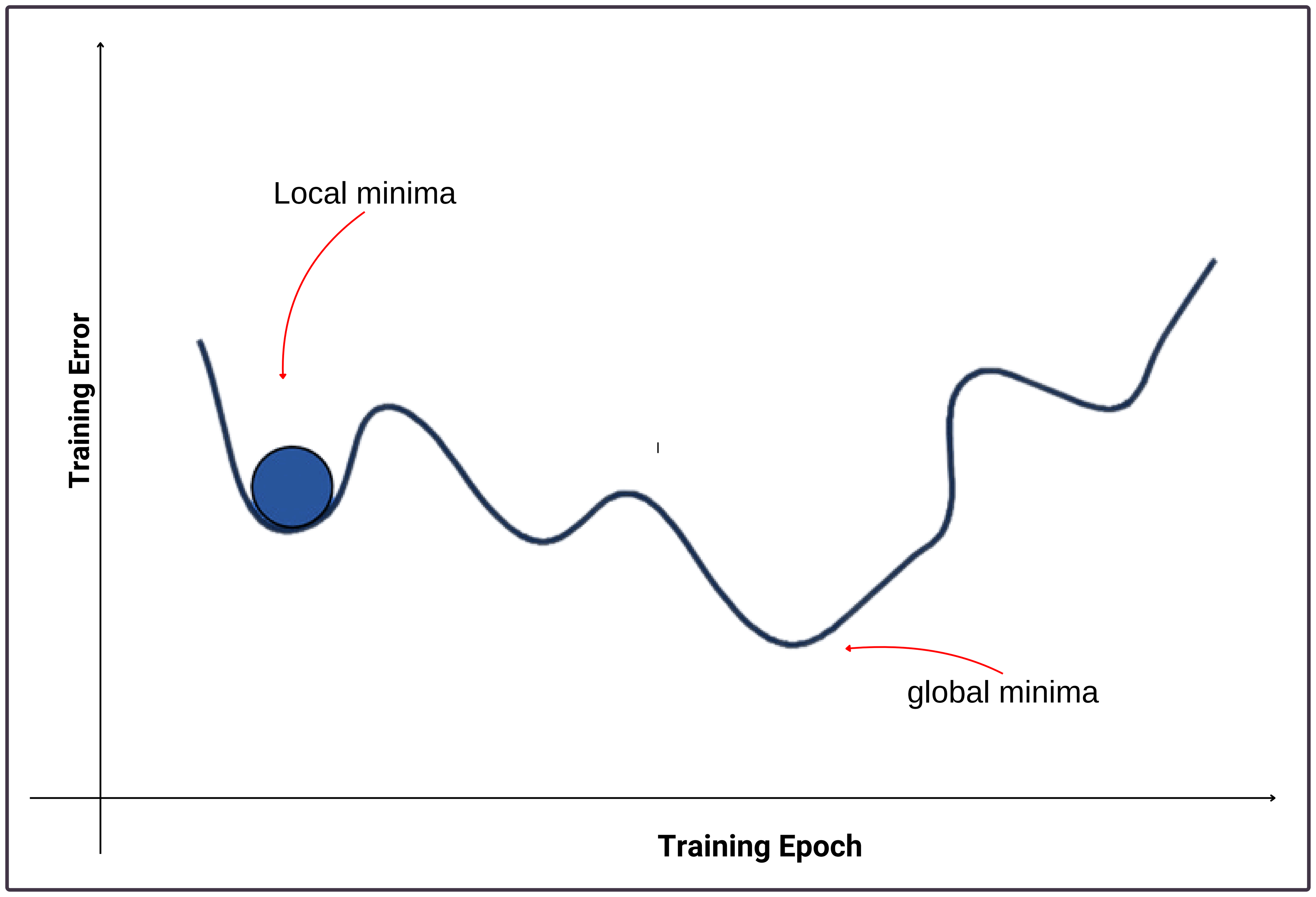
Gradient Descent is great, but sometimes we need smart optimizations to improve **speed and accuracy.**

Popular optimizers built on Gradient Descent:

- **Momentum** – Speeds up learning by considering past gradients.
- **RMSprop** – Adapts learning rates for different parameters.
- **Adam** – Combines Momentum and RMSprop for faster, more stable training (widely used).




Vanishing Gradient




Gradient stuck in local minima

# Challenges & Improvements in Gradient Descent

Gradient Descent isn't perfect. It can:

- Get stuck in **local minima** (bad solutions).
- Struggle with **vanishing gradients** in deep networks.
- Be sensitive to **bad learning rate choices.**

To improve it, techniques like **adaptive optimizers, learning rate scheduling, and better initialization methods** are used.

# Why Gradient Descent Matters

Without Gradient Descent, **neural networks wouldn't learn**. It's the key process that allows AI to:

- Improve its predictions over time.
- Adapt to new data.
- Solve complex problems in vision, NLP, and more.

It's the **engine that powers modern deep learning!**

# Final Thoughts

- Gradient Descent **optimizes neural networks** by adjusting weights to minimize errors.
- **Different types** (Batch, SGD, Mini-Batch) balance speed and stability.
- **Optimizers** like Adam help improve performance.

Mastering Gradient Descent is a **fundamental step** toward understanding deep learning!