

Module-3

Deep Learning (BCA-701)

Module-3

Regularization for Deep Learning:

Regularization is a technique that adds constraints or penalties to a neural network to prevent it from becoming too complex and overfitting the data.

Why Regularization is Needed in Deep Learning

- Deep networks have millions of parameters
- High risk of overfitting
- Regularization improves generalization
- Makes models more robust and stable

Common Regularization Techniques

1. L2 Regularization (Weight Decay)

- Adds a penalty proportional to the **square of weights**
- Penalizes large weights

Modified error function:

$$E_{reg} = E + \lambda \sum w^2$$

Where:

- λ = regularization parameter

✓ Encourages small weights

✓ Most commonly used

2. L1 Regularization

- Penalizes **absolute values of weights**

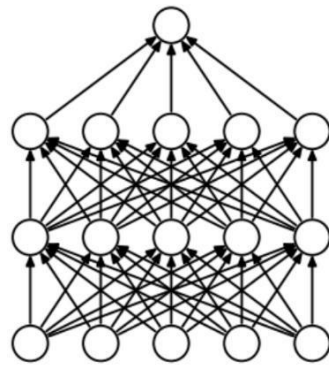
$$E_{reg} = E + \lambda \sum |w|$$

✓ Produces sparse models

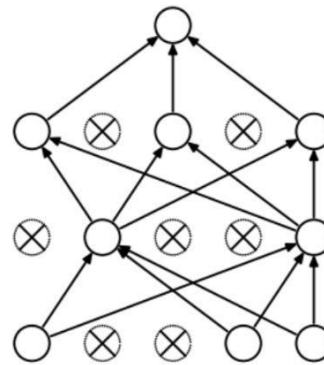
✓ Some weights become zero

3. Dropout

- Randomly disables neurons during training
- Prevents co-adaptation of neurons



(a) Standard Neural Net



(b) After applying dropout.

4. Early Stopping

- Stop training when validation error increases
- Prevents over-training

✓ Simple and effective

5. Data Augmentation

- Artificially increase training data
- Example: rotation, flipping, noise

✓ Reduces overfitting

✓ Improves robustness

6. Noise Injection

- Add noise to:
 - Inputs
 - Weights
 - Outputs

✓ Forces network to learn stable patterns

Effect of Regularization

- Reduces model complexity
- Improves test accuracy
- Controls variance
- Slightly increases bias

Comparison: With vs Without Regularization

Without Regularization	With Regularization
Overfits easily	Better generalization
Large weights	Controlled weights
Poor test accuracy	Improved test accuracy

1. Parameter Norm Penalties - L2 Parameter Regularization

L2 Parameter Regularization, also known as **Weight Decay**, is one of the most widely used **parameter norm penalty techniques** in deep learning. It helps control model complexity by **penalizing large weights**, thereby reducing overfitting.

What are Parameter Norm Penalties?

Parameter norm penalties are **regularization methods** that add a penalty term to the loss function based on the **magnitude (norm) of the parameters (weights)**.

- Large weights → complex model → overfitting
- Small weights → smoother model → better generalization

L2 Parameter Regularization (Weight Decay)

Definition

L2 regularization adds a penalty equal to the **sum of squares of the weights** to the original loss function.

Modified Loss Function

$$E_{total} = E_{data} + \lambda \sum_i w_i^2$$

Where:

- E_{data} = original loss (e.g., MSE or cross-entropy)
- w_i = network weights
- λ = regularization parameter (controls strength)

Effect on Weight Update

During gradient descent, weight update becomes:

$$\frac{\partial E}{\partial w_i} = w_i - \eta \left(\frac{\partial E}{\partial w_i} + 2\lambda w_i \right)$$

- The term $2\lambda w_i$ pulls weights toward zero.

Why it is Called Weight Decay

- Each update slightly **shrinks the weights**
- Prevents weights from growing too large
- Produces smoother decision boundaries

Geometric Interpretation

- L2 regularization constrains weights inside a **hypersphere**
- Encourages evenly distributed small weights instead of few large ones

Advantages of L2 Regularization

- Reduces overfitting
- Improves generalization
- Keeps all weights small (no abrupt zeroing)

- Easy to implement

Limitations

- Does not produce sparse models
- Choice of λ is critical

L1 vs L2 Regularization (Quick Comparison)

Aspect	L1	L2
Penalty	$(\sum w$	w
Sparsity	Yes	No
Weight behavior	Many zeros	Small non-zero
Stability	Less	More stable

2. Dataset Augmentation s Semi-Supervised Learning

These two techniques are used in deep learning to improve generalization, especially when labeled data is limited.

1. Dataset Augmentation

Dataset Augmentation is a regularization technique where the training dataset is artificially increased by applying transformations to existing data without changing their labels.

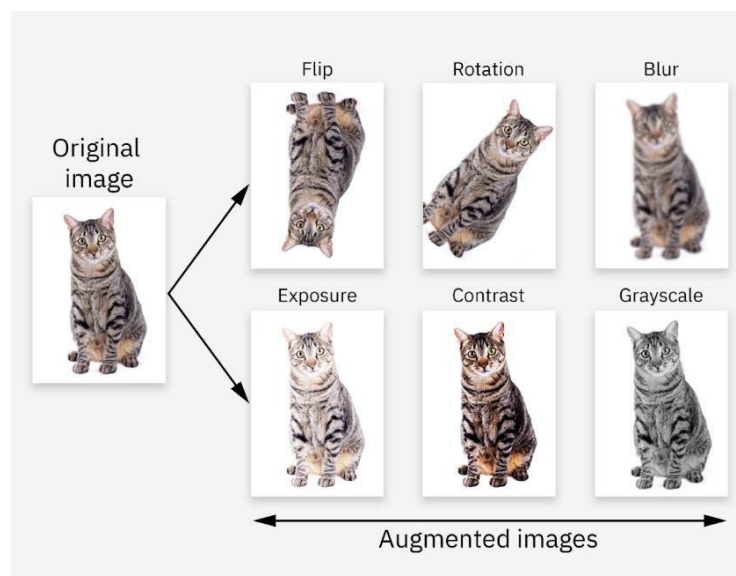
Why Dataset Augmentation is Needed

- Collecting labeled data is expensive
- Deep networks need large datasets
- Reduces overfitting
- Improves robustness

Common Augmentation Techniques

For Image Data

- Rotation
- Flipping (horizontal/vertical)
- Scaling
- Cropping
- Adding noise
- Brightness C contrast adjustment



For Text Data

- Synonym replacement
- Random insertion/deletion
- Back translation

For Audio Data

- Time shifting
- Noise injection
- Pitch variation

Advantages

- Increases dataset size
- Improves model generalization
- Reduces overfitting
- No need for new labels

Limitations

- Limited by nature of transformations
- Poor augmentation may distort data

2. Semi-Supervised Learning

Semi-Supervised Learning is a learning approach that uses **both labeled and unlabeled data** for training.

Why Semi-Supervised Learning is Used

- Labeled data is scarce and costly
- Unlabeled data is abundant
- Improves learning accuracy

Basic Idea

- Train model using small labeled dataset
- Use model predictions to label unlabeled data
- Retrain using combined data

Common Semi-Supervised Techniques**1. Self-Training**

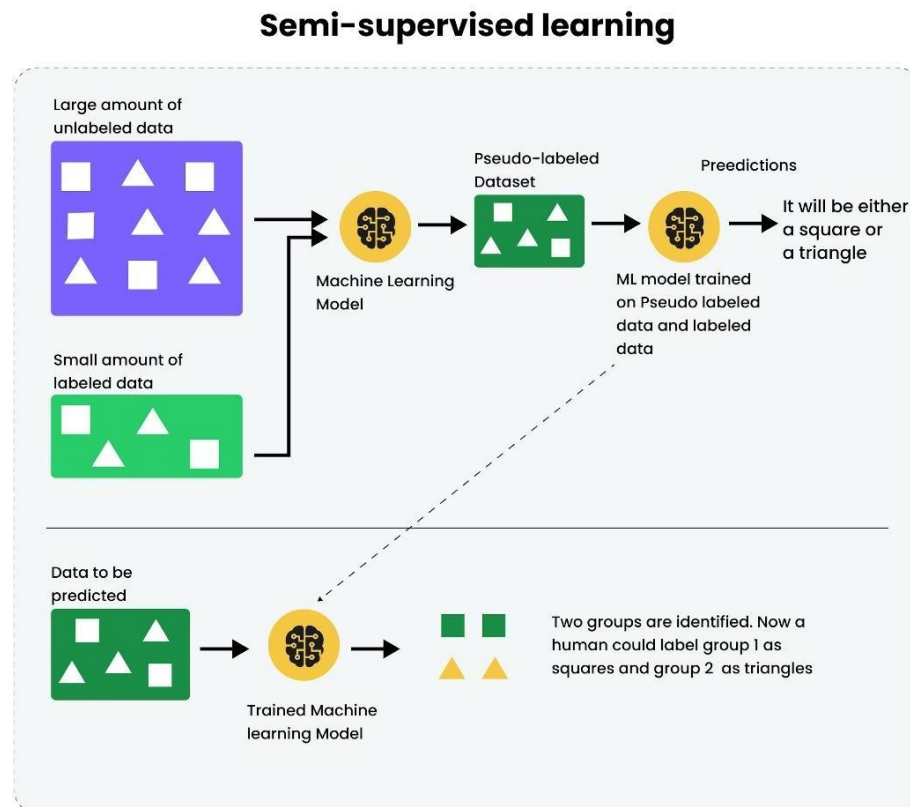
- Model labels unlabeled data itself
- High-confidence predictions are reused

2. Co-Training

- Two models train each other using different feature sets

3. Graph-Based Methods

- Data represented as graph
- Labels propagated through graph



Advantages

- Reduces labeling cost
- Improves performance
- Utilizes real-world data effectively

Limitations

- Incorrect pseudo-labels can propagate errors
- More complex training

Comparison: Dataset Augmentation vs Semi-Supervised Learning

Aspect	Dataset Augmentation	Semi-Supervised Learning
Data type	Labeled data	Labeled + unlabeled
Goal	Increase data size	Reduce labeling need
Overfitting control	Yes	Yes
Complexity	Low	Moderate

Optimization for Training Deep Models:

Optimization in deep learning refers to the process of **adjusting model parameters (weights and biases)** so that the **loss function is minimized**. Training deep models is challenging due to complex error surfaces and high-dimensional parameter spaces.

What is Optimization?

Optimization aims to find the **best set of parameters** that minimizes a loss function (θ) using iterative update rules such as **gradient-based methods**.

$$\theta \leftarrow \theta - \eta \nabla_{\theta} E(\theta)$$

Where:

- θ = model parameters
- η = learning rate
- $\nabla_{\theta} E$ = gradient of loss

Why Optimization is Difficult in Deep Learning

Deep models have:

- Millions of parameters
- Non-convex loss functions

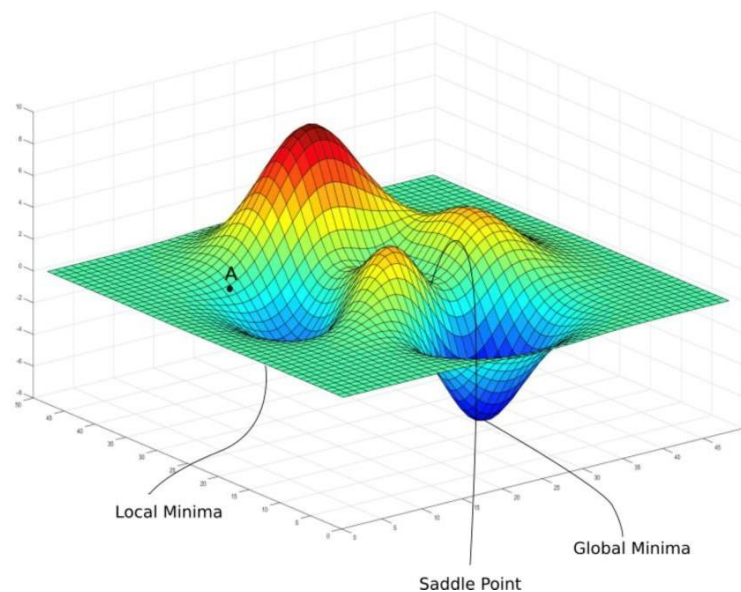
- Noisy gradients
- Long training times

These lead to several **optimization challenges**.

Major Optimization Challenges

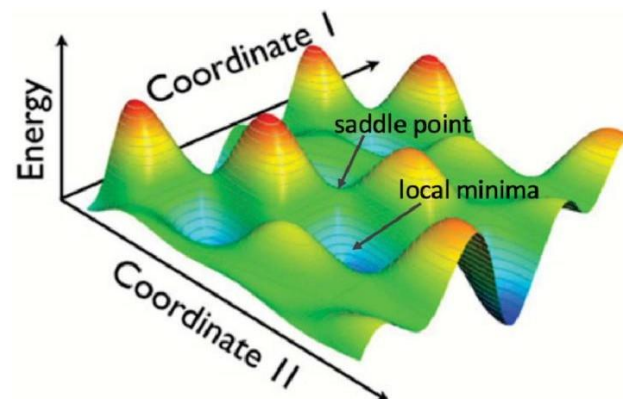
1. Local Minima

- Points where loss is low but **not globally minimal**
- Can trap optimization algorithms



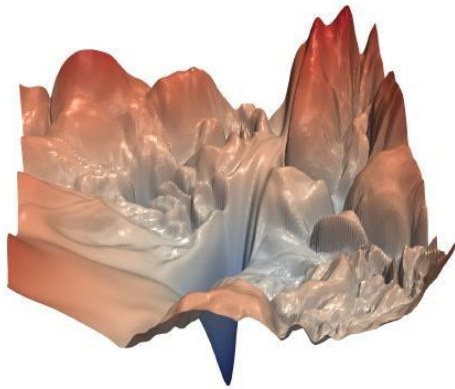
2. Saddle Points

- Gradient is zero, but point is **neither minimum nor maximum**
- Very common in high-dimensional spaces
- Slows down learning significantly

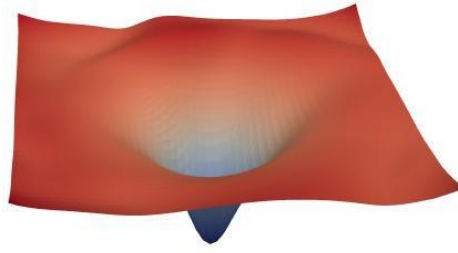


3. Plateaus (Flat Regions)

- Regions with very small gradients
- Weight updates become extremely slow
- Common with sigmoid/tanh activations



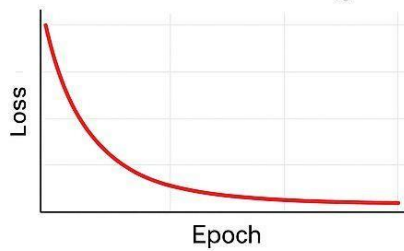
(a) without skip connections



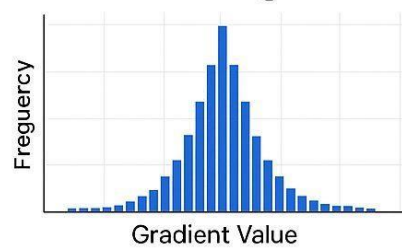
(b) with skip connections

Visualization: How to Spot Vanishing Gradients in Your Model

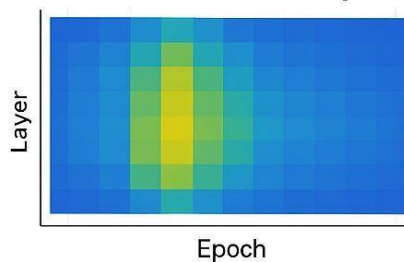
Loss Plateauing



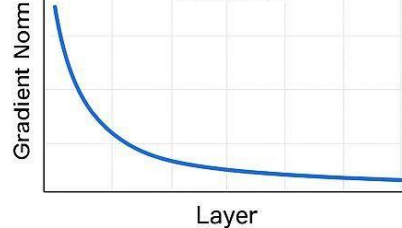
Gradient Histograms



Gradient Heatmap



Layer-wise Gradient Norms



4. Vanishing Gradient Problem

- Gradients become very small in deep networks
- Earlier layers learn very slowly
- Common with sigmoid activation

5. Exploding Gradients

- Gradients grow very large
- Causes unstable training and overflow

6. Poor Conditioning

- Loss surface has steep and flat directions
- Gradient descent oscillates or converges slowly

Common Optimization Algorithms**1. Gradient Descent Variants**

- Batch Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini-batch Gradient Descent

2. Momentum-Based Methods

- Momentum
- Nesterov Accelerated Gradient

✓ Helps escape saddle points

✓ Reduces oscillations

3. Adaptive Learning Rate Methods

- AdaGrad
- RMSProp

- Adam
- ✓ Automatically adjust learning rate
- ✓ Faster convergence

Techniques to Improve Optimization

- Proper weight initialization
- Input normalization
- Use of ReLU activation
- Batch normalization
- Gradient clipping

3. Challenges in Neural Network Optimization - Ill Conditioning

Ill conditioning is one of the major challenges faced while training deep neural networks. It refers to a situation where the **loss function surface is badly shaped**, making optimization **slow and unstable**.

What is Ill Conditioning? (Simple Definition)

Ill conditioning occurs when the **error surface has very steep slopes in some directions and very flat slopes in other directions**, causing gradient-based optimization algorithms to struggle.

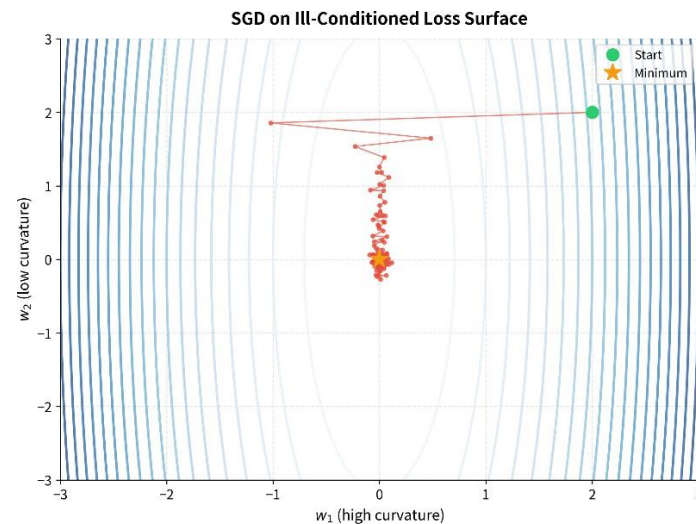
Geometric Interpretation

- The loss surface looks like a **long, narrow valley**
- Gradients point in inefficient directions
- Optimization algorithm **zig-zags** instead of moving directly to the minimum

Why Ill Conditioning Happens

- Poor weight initialization
- Highly correlated input features

- Deep networks with many layers
- Improper scaling of inputs
- Large difference in curvature of loss surface



Effect on Gradient Descent

- Slow convergence
- Oscillations during training
- Requires very small learning rate
- Training may take many iterations

Mathematical View

Ill conditioning is related to the **Hessian matrix** of the loss function:

- Large difference between **largest and smallest eigenvalues**
- High **condition number** $\Rightarrow$ severe ill conditioning

Problems Caused by Ill Conditioning

- Gradient descent overshoots in steep directions
- Barely moves in flat directions
- Wastes computational resources

- May fail to converge efficiently

How to Reduce Ill Conditioning

1. Input Normalization

- Scale inputs to zero mean and unit variance
- Reduces correlation between features

2. Batch Normalization

- Normalizes activations within the network
- Makes loss surface smoother

3. Adaptive Optimization Algorithms

- Adam
- RMSProp
- AdaGrad

✓ Automatically adjust learning rates

4. Momentum

- Reduces zig-zag behavior
- Accelerates convergence

5. Proper Weight Initialization

- Xavier initialization
- He initialization

4. Local Minima

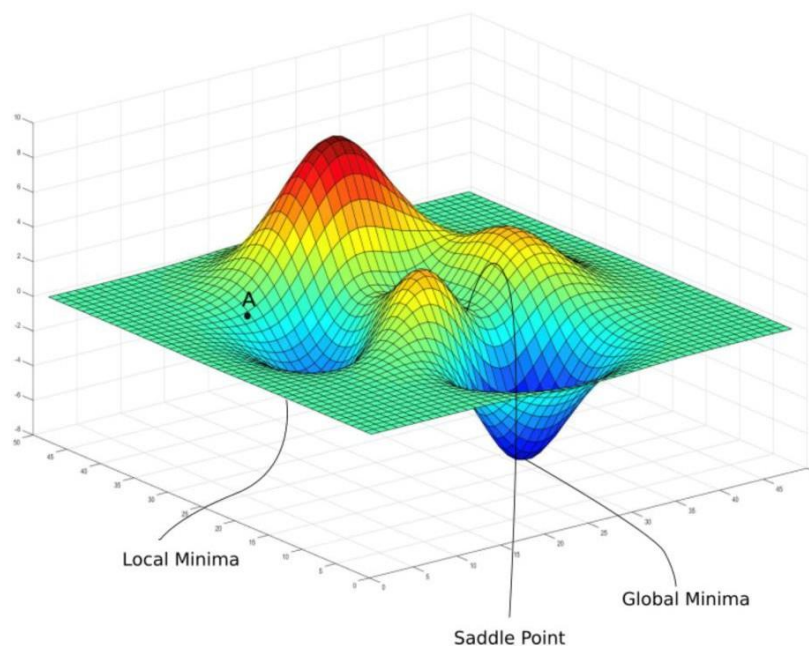
A **local minimum** is a point on the loss (error) surface where the loss is **lower than its immediate surroundings**, but **not the lowest possible value overall** (global minimum). During training, optimization algorithms like gradient descent can get **stuck** at these points.

What is a Local Minimum? (Simple Definition)

A local minimum is a solution where the gradient is zero and the loss is minimal **only within a small neighborhood**, not across the entire parameter space.

Geometric Interpretation

- The loss surface contains **many valleys**
- A local minimum is a **small valley**
- The global minimum is the **deepest valley**



Why Local Minima Occur in Neural Networks

- Highly **non-convex** loss functions
- Large number of parameters
- Complex interactions between layers
- Nonlinear activation functions

Effect on Training

- Gradient becomes **zero** at a local minimum
- Weight updates stop or slow down
- Model may converge to a **suboptimal solution**

Are Local Minima Always Bad?

In **deep neural networks**:

- Many local minima have **similar loss values**
- Some local minima perform **almost as well as the global minimum**
- Hence, local minima are **less problematic** than saddle points in practice

How to Avoid or Escape Local Minima

1. Momentum

- Adds inertia to updates
- Helps move past shallow local minima

2. Random Weight Initialization

- Different starting points reduce chance of poor minima

3. Stochastic / Mini-batch Gradient Descent

- Noisy updates help escape local minima

4. Adaptive Optimizers

- Adam, RMSProp adjust learning rates automatically

5. Plateaus

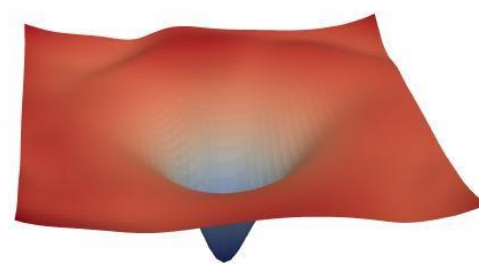
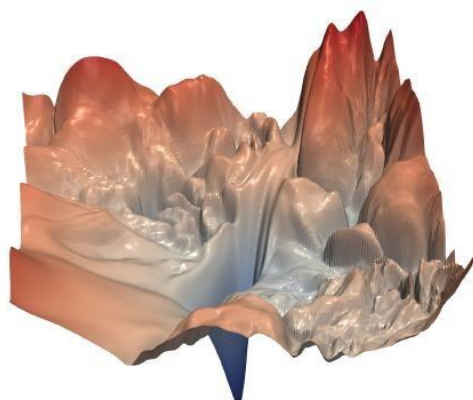
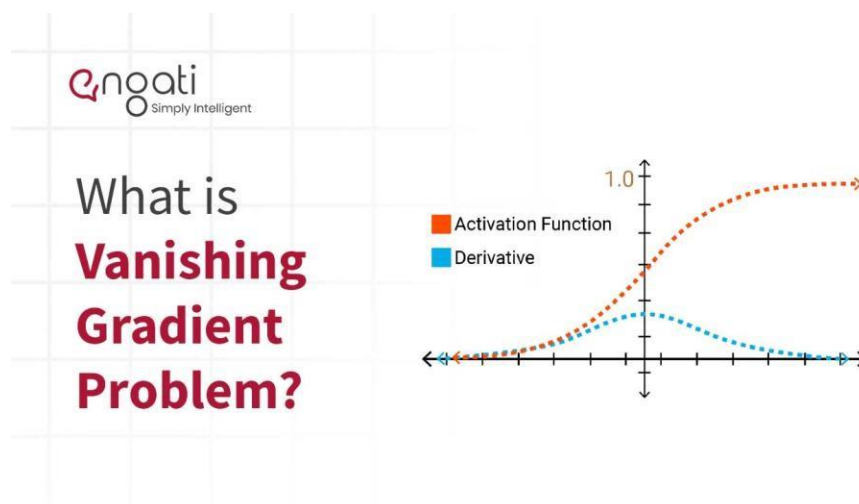
A **plateau** is a region on the loss (error) surface where the **gradient is very small or nearly zero over a wide area**. In such regions, learning becomes **extremely slow**, even though the network has not reached an optimal solution.

What is a Plateau? (Simple Definition)

A plateau is a **flat region of the error surface** where changes in weights produce **very little change in error**, causing gradient-based optimization to stall.

Geometric Interpretation

- The loss surface looks **almost flat**
- Gradients are close to zero
- Weight updates become tiny



Why Plateaus Occur

- Vanishing gradient problem
- Use of sigmoid or tanh activation functions
- Poor weight initialization
- Deep network architectures
- Saturated neurons

Effect on Training

- Very slow convergence
- Training may appear to stop
- Requires many iterations
- Wastes computational resources

Difference Between Plateaus and Local Minima

Aspect	Plateaus	Local Minima
Gradient	Very small	Zero
Shape	Flat region	Valley
Learning	Very slow	Stops
Escape	Possible	Difficult

How to Overcome Plateaus

1. Use ReLU Activation

- Avoids saturation
- Reduces vanishing gradients

2. Momentum

- Accumulates past gradients

- Helps move across flat regions

3. Adaptive Optimizers

- Adam
- RMSProp

✓ Increase effective step size in flat regions

4. Batch Normalization

- Stabilizes activations
- Improves gradient flow

5. Better Weight Initialization

- Xavier initialization
- He initialization

6. Saddle Points and Other Flat Regions

In deep neural network optimization, **saddle points and flat regions** are more serious challenges than local minima. They cause **very slow learning** and often trap gradient-based optimization algorithms.

1. Saddle Points

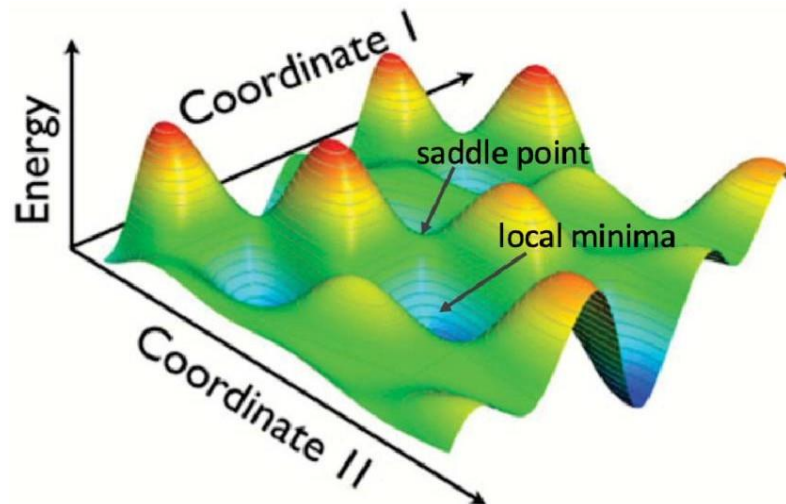
What is a Saddle Point? (Simple Definition)

A **saddle point** is a point on the loss surface where:

- Gradient is **zero**
- It is **not a minimum or maximum**
- The surface curves **upward in some directions and downward in others**

Geometric Interpretation

- Looks like a **horse saddle**
- Downhill in one direction, uphill in another
- Gradient descent may stop or slow down



Why Saddle Points Are Common

- Deep networks have **high-dimensional parameter spaces**
- Number of saddle points increases exponentially with dimensions
- Much more frequent than local minima in deep learning

Effect on Training

- Gradient becomes zero $\rightarrow$ learning stalls
- Optimizer may take long time to escape
- Appears like convergence, but is not optimal

2. Other Flat Regions

What are Flat Regions?

Flat regions are areas of the loss surface where:

- Gradients are **very small**
- Error does not change significantly

- Training progresses extremely slowly

Causes of Flat Regions

- Vanishing gradient problem
- Saturated activation functions (sigmoid, tanh)
- Deep architectures
- Poor scaling of inputs

Difference Between Saddle Points and Flat Regions

Aspect	Saddle Points	Flat Regions
Gradient	Zero	Near zero
Shape	Curved up C down	Almost flat
Learning	Stalls	Very slow
Common in deep nets	Very common	Common

Why Saddle Points Are Worse Than Local Minima

- Harder to detect
- More frequent
- Optimization may stay near saddle point for many iterations

How to Overcome Saddle Points s Flat Regions

1. Stochastic / Mini-batch Gradient Descent

- Noise helps escape saddle points

2. Momentum

- Pushes optimization through flat areas

3. Adaptive Optimizers

- Adam
- RMSProp

✓ Increase learning speed in low-gradient directions

4. ReLU and Variants

- Reduce vanishing gradients

5. Batch Normalization

- Improves gradient flow

Key Exam Points

- Saddle points have zero gradient but are not minima
- More common than local minima in deep learning
- Flat regions slow down training
- Momentum and adaptive optimizers help escape