

Lecture 9:

Initialization and Normalization

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

Part 1: Initialization Strategies

Strategies for setting **weights and biases** before training begins. Proper initialization prevents gradient vanishing/exploding problems, accelerates training speed, and ensures better convergence. (e.g., Xavier, He initialization)

Part 2: Normalization Techniques

Techniques for normalizing the **input distribution** to each layer during training. Reduces Internal Covariate Shift, stabilizes training, and enables the use of higher learning rates. (e.g., Batch Norm, Layer Norm, Group Norm)

Part 3: Normalization and Generalization

Explores how normalization techniques affect model **generalization performance**. Covers theoretical and empirical analysis of how normalization prevents overfitting, improves test performance, and enhances model robustness.

Part 1/3:

Initialization Strategies

- 1.** Why is Initialization Important?
- 2.** Problems with Zero Initialization
- 3.** Random Initialization and Breaking Symmetry
- 4.** Gradient Vanishing/Exploding
- 5.** Xavier/Glorot Initialization
- 6.** He Initialization (for ReLU)
- 7.** LSUV Initialization
- 8.** Leveraging Pre-trained Weights
- 9.** Comparison of Initialization Strategies

Why is Initialization Important?

Initialization Strategy

Sets starting point for
gradient descent

Affects **convergence speed** & performance

Prevents
vanishing/exploding
gradients

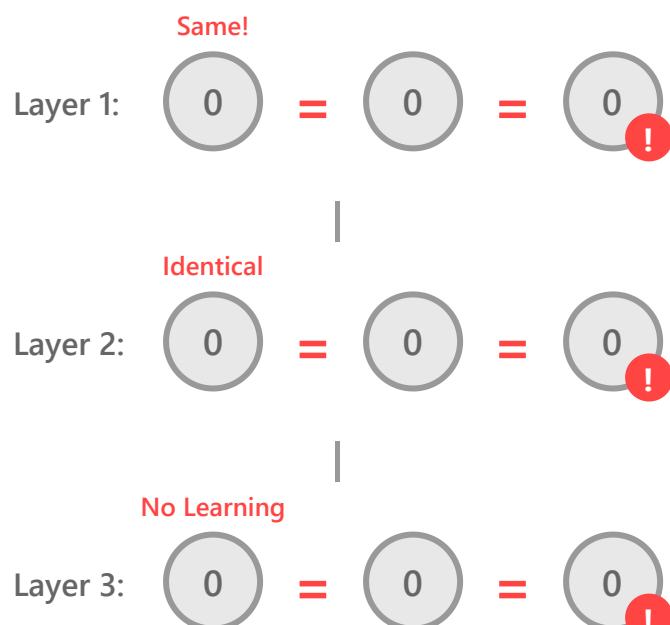
Determines learning
effectiveness

Ensures **training stability**

Critical for **deep networks**

Influences **final accuracy**

Problems with Zero Initialization



- 1 Symmetry problem: All neurons compute identical outputs
- 2 Gradients are identical for all neurons in same layer
- 3 Network fails to break symmetry during backpropagation
- 4 Results in redundant neurons learning same features
- 5 No learning occurs - weights remain constant
- 6 Effectively reduces network to single neuron per layer

Random Initialization and Breaking Symmetry

X Zero Initialization



All neurons identical
Symmetry problem



✓ Random Initialization



Each neuron unique
Symmetry broken!

Common Distributions

$N(0, 0.01)$ or $U(-0.01, 0.01)$

Key Benefits

Breaks symmetry between neurons with small random values

Each neuron learns **different features** independently

Enables **diverse representation** learning across layers

Scale matters: Too large → saturation, Too small → vanishing gradients

Foundation for **all modern methods**

Practical Example: Why Zero Initialization Fails



✗ Zero Initialization

$$w_{1\ 1} = w_{1\ 2} = w_{2\ 1} = w_{2\ 2} = 0$$

$$\begin{aligned} h_1 &= w_{1\ 1} \cdot x_1 + w_{1\ 2} \cdot x_2 \\ &= 0 \cdot 1 + 0 \cdot 2 = 0 \end{aligned}$$

$$\begin{aligned} h_2 &= w_{2\ 1} \cdot x_1 + w_{2\ 2} \cdot x_2 \\ &= 0 \cdot 1 + 0 \cdot 2 = 0 \end{aligned}$$

$h_1 = h_2 = 0$ (Identical!)

Gradient Update:

$$\partial L / \partial w_{1\ 1} = \partial L / \partial w_{1\ 2} \quad (\text{same})$$

$$\partial L / \partial w_{2\ 1} = \partial L / \partial w_{2\ 2} \quad (\text{same})$$

$w_{1\ 1}$ and $w_{1\ 2}$ stay identical

$w_{2\ 1}$ and $w_{2\ 2}$ stay identical

⚠ Symmetry Never Broken

✓ Random Initialization

$$w_{1\ 1} = 0.5, w_{1\ 2} = -0.3$$

$$w_{2\ 1} = 0.2, w_{2\ 2} = 0.4$$

$$\begin{aligned} h_1 &= w_{1\ 1} \cdot x_1 + w_{1\ 2} \cdot x_2 \\ &= 0.5 \cdot 1 + (-0.3) \cdot 2 \\ &= 0.5 - 0.6 = -0.1 \end{aligned}$$

$$\begin{aligned} h_2 &= w_{2\ 1} \cdot x_1 + w_{2\ 2} \cdot x_2 \\ &= 0.2 \cdot 1 + 0.4 \cdot 2 \\ &= 0.2 + 0.8 = 1.0 \end{aligned}$$

$h_1 = -0.1 \neq h_2 = 1.0$ (Different!)

Gradient Update:

$$\partial L / \partial w_{1\ 1} \neq \partial L / \partial w_{1\ 2} \quad (\text{different})$$

$$\partial L / \partial w_{2\ 1} \neq \partial L / \partial w_{2\ 2} \quad (\text{different})$$

Each weight learns
unique features independently

✓ Symmetry Broken Successfully

Gradient Vanishing/Exploding Problems

↓ Vanishing Gradients

L4

L3

L2

L1

Gradients become **extremely small** in early layers, preventing effective learning in deep networks.

↑ Exploding Gradients

L1

L2

L3

L4

Gradients grow **exponentially large** during backpropagation, causing training instability.

Key Causes & Effects

Cause: Repeated multiplication of small/large values during backpropagation

Susceptible activations: Sigmoid/tanh particularly vulnerable

Impact: Prevents effective learning in deep networks

Consequence: Slow convergence or complete training failure

✓ Solution

Proper initialization helps maintain gradient magnitude throughout the network

Xavier/Glorot Initialization

Designed for Activations

sigmoid

tanh

Var

Var

Var

Layer 1

Layer 2

Layer 3

Variance maintained ≈ 1.0

Key Properties

Variance maintained across layers during forward/backward pass

Keeps activation variance approximately **1.0**

Prevents **saturation** in early training stages

Standard for **fully connected networks**

Optimal for **tanh/sigmoid** activation functions

Uniform Distribution

$$W \sim U(-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})})$$

Gaussian Distribution

$$W \sim N(0, 2/(n_{in} + n_{out}))$$

Calculation Examples

Example 1: Uniform Distribution

Input neurons (n_{in}): **784**

Output neurons (n_{out}): **128**

Weight matrix size: **784×128**

- 1 Calculate sum

$$n_{in} + n_{out} = 784 + 128 = 912$$

- 2 Calculate fraction

$$\begin{aligned} 6 / (n_{in} + n_{out}) &= 6 / 912 \\ &= 0.006579 \end{aligned}$$

- 3 Calculate square root

$$\sqrt{0.006579} = 0.0811$$

Example 2: Gaussian Distribution

Input neurons (n_{in}): **256**

Output neurons (n_{out}): **64**

Weight matrix size: **256×64**

- 1 Calculate sum

$$n_{in} + n_{out} = 256 + 64 = 320$$

- 2 Calculate variance

$$\begin{aligned} \sigma^2 &= 2 / (n_{in} + n_{out}) = 2 / 320 \\ &= 0.00625 \end{aligned}$$

- 3 Standard deviation (σ)

$$\sigma = \sqrt{0.00625} = 0.0791$$

$$W \sim U(-0.0811, +0.0811)$$

Weight samples (partial)

```
[-0.0523  0.0701 -0.0198  0.0445]  
[ 0.0312 -0.0789  0.0621 -0.0356]  
[-0.0678  0.0234  0.0789 -0.0512]  
[ 0.0456 -0.0123  0.0654  0.0289]
```

$$W \sim N(0, \sigma=0.0791)$$

Weight samples (partial)

```
[-0.0423  0.0612 -0.0891  0.0234]  
[ 0.0756 -0.0189  0.0523 -0.0678]  
[-0.0345  0.0801 -0.0267  0.0445]  
[ 0.0589 -0.0734  0.0123  0.0690]
```

Example 3: Small Network

Input neurons (n_{in}): **10**

Output neurons (n_{out}): **5**

Weight matrix size: **10×5**

1 Uniform Distribution calculation

$$n_{in} + n_{out} = 10 + 5 = 15$$

$$\sqrt{6/15} = \sqrt{0.4} = 0.6325$$

$$W \sim U(-0.6325, +0.6325)$$

Full weight matrix (10×5)

Example 4: Large Network Comparison

Input neurons (n_{in}): **2048**

Output neurons (n_{out}): **1024**

Weight matrix size: **2048×1024**

1 Uniform Distribution

$$\sqrt{6/(2048+1024)} = \sqrt{6/3072}$$

$$= \sqrt{0.001953} = 0.0442$$

2 Gaussian Distribution

```
[-0.3421  0.5234 -0.1823  0.4512 -0.2901]  
[ 0.2134 -0.4823  0.3912  0.1567 -0.5123]  
[-0.4567  0.2901  0.5421 -0.3234  0.1789]  
[ 0.3789 -0.2156  0.4234 -0.5678  0.2912]  
[-0.1234  0.4567 -0.3812  0.2345  0.5234]  
[ 0.4912 -0.3567  0.1234 -0.4321  0.3678]  
[-0.2678  0.5012 -0.4156  0.1923  0.3456]  
[ 0.3234 -0.1789  0.4678 -0.2567  0.5123]  
[-0.4123  0.2678  0.3567 -0.5234  0.1456]  
[ 0.5678 -0.3012  0.2134  0.4789 -0.3901]
```

$$\sigma = \sqrt{2/3072} = \sqrt{0.000651}$$
$$= 0.0255$$

Uniform: $W \sim U(-0.0442, +0.0442)$

Gaussian: $W \sim N(0, \sigma=0.0255)$

 **Observation:** As the network grows larger, the initialization range becomes smaller. This prevents the sum of many inputs from becoming too large.

He Initialization (for ReLU)

ReLU Effect: Half Neurons Output 0



Approximately **50% neurons killed** by ReLU

He Initialization Formula

$$W \sim N(0, \frac{2}{n_{in}})$$

Variance factor of **2** compensates for ReLU's effect

Key Features

Specifically designed for **ReLU activation** functions

Accounts for ReLU **killing half** the neurons (output 0)

Maintains **signal strength** in deep ReLU networks

Significantly improves training of **very deep networks**

✓ Default Choice

Standard initialization for modern **CNN architectures** and deep learning models with ReLU



Practical Calculation Example

Example 1: Layer with 4 inputs → 3 outputs

Step 1: Calculate Standard Deviation

Example 2: Layer with 128 inputs → 64 outputs

Step 1: Calculate Standard Deviation

$$n_{in} = 4$$

$$\sigma = \sqrt{2/n_{in}} = \sqrt{2/4} = \sqrt{0.5}$$

$$\sigma \approx 0.707$$

Step 2: Generate Weight Matrix (3×4)

Sample from $N(0, 0.707^2)$

```
W = [[ 0.423, -0.891, 0.156, 0.734]
[-0.612, 0.289, -0.445, 0.821]
[ 0.534, -0.178, 0.693, -0.356]]
```

Verification

Variance ≈ 0.5 ✓

Compensates for ReLU killing ~50%
neurons

$$n_{in} = 128$$

$$\sigma = \sqrt{2/n_{in}} = \sqrt{2/128}$$

$$\sigma = \sqrt{0.015625}$$

$$\sigma \approx 0.125$$

Step 2: Generate Weight Matrix (64×128)

Sample from $N(0, 0.125^2)$

```
W[0,:5] = [ 0.089, -0.134, 0.112, -0.078,
0.156]
W[1,:5] = [-0.091, 0.145, -0.123, 0.067,
-0.089]
W[2,:5] = [ 0.134, -0.056, 0.178, -0.145,
0.103]
...
(Much smaller values due to larger nin)
```

Key Insight

Larger n_{in} → Smaller weights

Prevents activation explosion 



Comparison: Xavier vs He Initialization

Layer Size (n _{in})	Xavier: $\sigma = \sqrt{1/n_{in}}$	He: $\sigma = \sqrt{2/n_{in}}$	Ratio (He/Xavier)
4	0.500	0.707	$\sqrt{2} \approx 1.41\times$

16	0.250	0.354	$\sqrt{2} \approx 1.41 \times$
64	0.125	0.177	$\sqrt{2} \approx 1.41 \times$
256	0.0625	0.088	$\sqrt{2} \approx 1.41 \times$

Key Takeaway

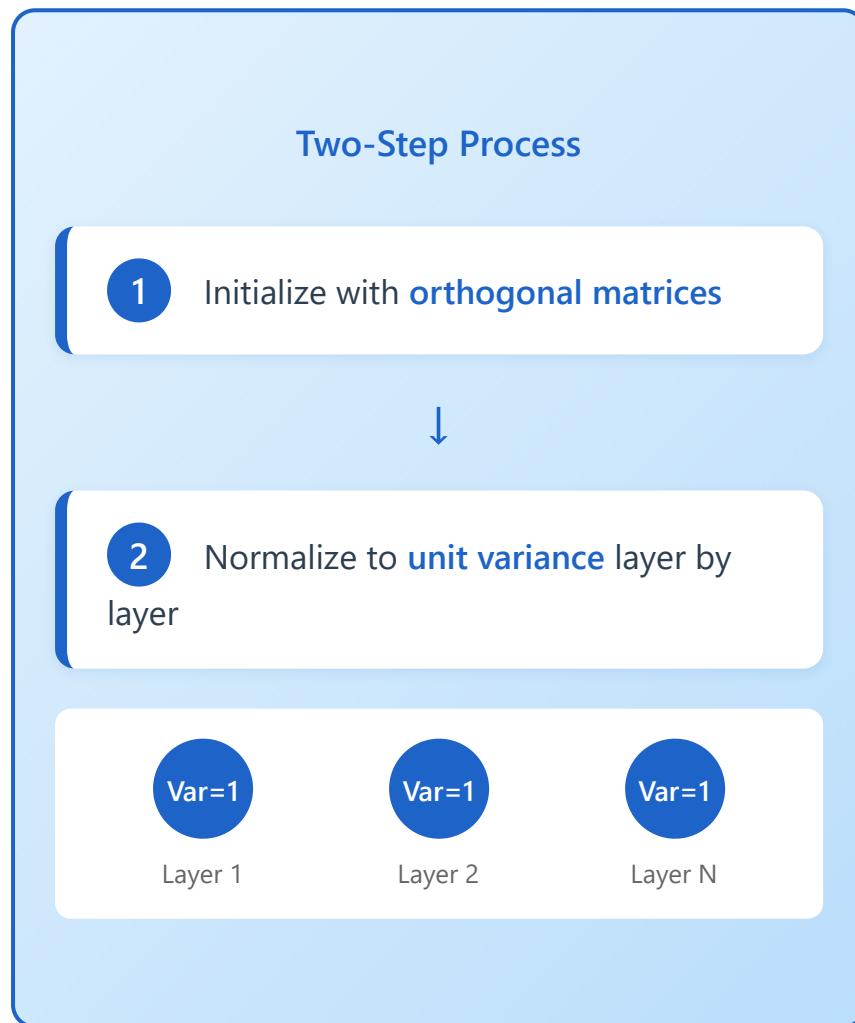
He initialization uses **$\sqrt{2}$ times larger** weights than Xavier

This compensates for ReLU zeroing out half the neurons

Result: **Stable gradient flow** in deep ReLU networks

LSUV Initialization

Layer-Sequential Unit-Variance Initialization



Key Characteristics

Sequential process: Initializes and normalizes one layer at a time

Orthogonal initialization: Provides better starting point than random

Data-driven: Uses actual data to estimate variance

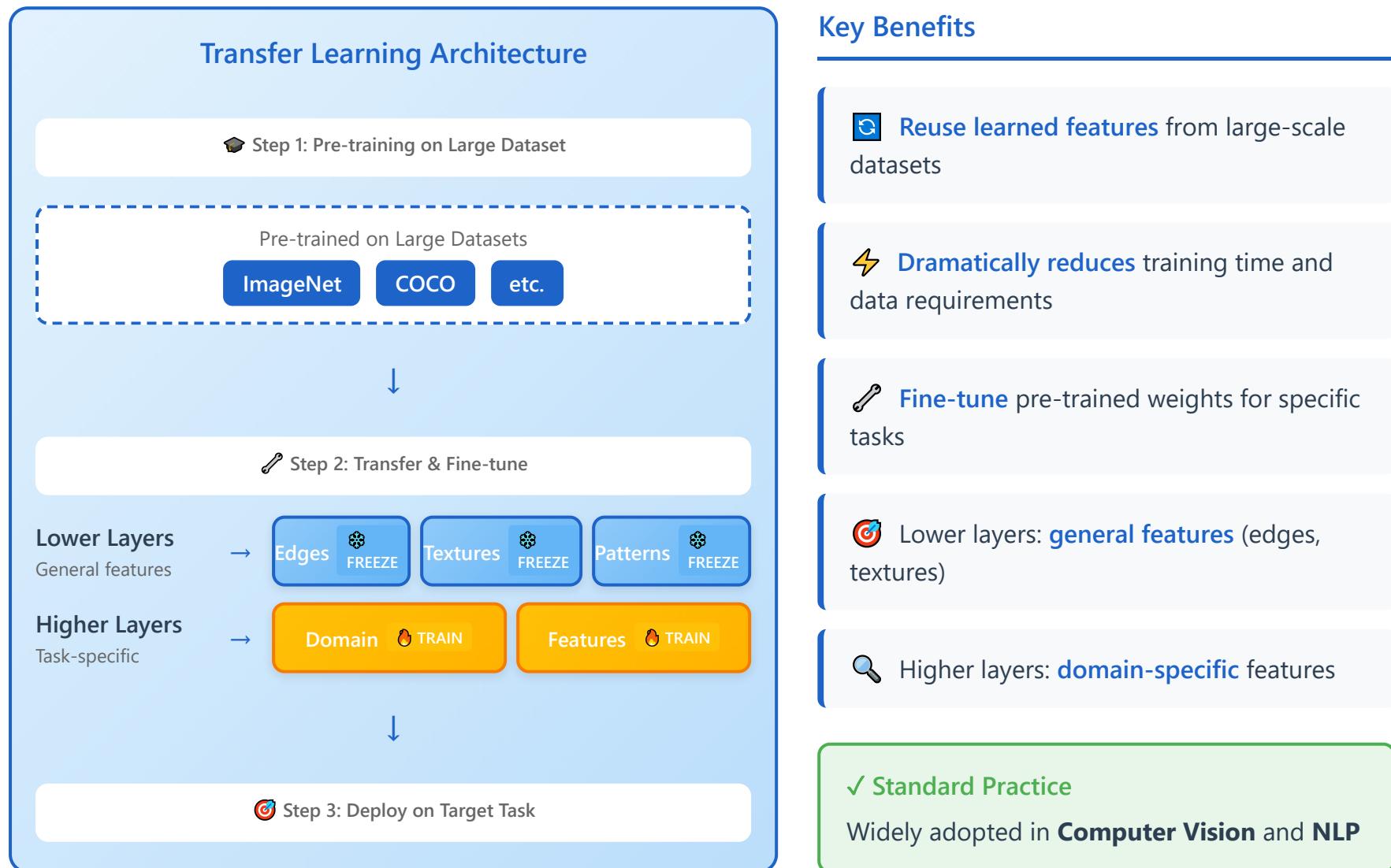
More sophisticated: Advanced compared to Xavier/He methods

✓ **Particularly Effective**
Excels in **very deep networks** with **100+ layers**

Uses **small batch of data** for variance estimation

 Additional Resource: DeepLearning.AI - Weight Initialization Guide

Leveraging Pre-trained Weights



Comparison of Initialization Strategies

Method	Best Use Case	Recommendation
Zero Init	Never use - causes symmetry problem	 Avoid
Random Small Values	Basic approach, often insufficient for deep networks	 Basic
Xavier/Glorot	Best for tanh/sigmoid activations	 Recommended
He Initialization	Best for ReLU and variants (LeakyReLU, PReLU)	 Recommended
LSUV	Best for extremely deep networks (100+ layers)	 Advanced
Pre-trained Weights	Best when transfer learning is applicable	 Preferred

Key Takeaway: Choice depends on network architecture and activation functions. Modern practice favors He

initialization for ReLU networks and pre-trained weights when available.

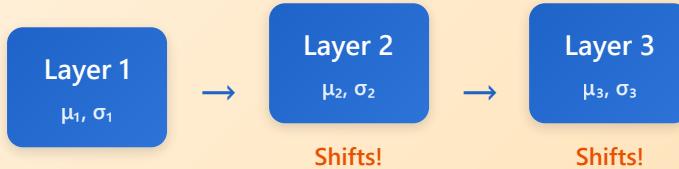
Part 2/3:

Normalization Techniques

- 10.** Internal Covariate Shift
- 11.** Batch Normalization (Batch Norm)
- 12.** Layer Normalization (Layer Norm)
- 13.** Instance Normalization
- 14.** Group Normalization
- 15.** Weight Normalization
- 16.** Spectral Normalization
- 17.** Comparison of Normalization Techniques
- 18.** When to Use Which Normalization?

Internal Covariate Shift

Distribution Shifts During Training



⚠️ Training Impact

Requires lower learning rates for stable training

Key Problems

Distribution changes as previous layers update during training

Forces layers to **continuously adapt** to new distributions

Slows down training and reduces stability

More **pronounced in deeper** networks

✓ Solution

Normalization techniques address this problem directly by stabilizing distributions

Mathematical Explanation

Input Distribution Changes per Layer

$$x^{(l)} = f(x^{(l-1)}; \theta^{(l-1)})$$

Batch Normalization Formula

$$\mu_B = (1/m) \sum x_i$$

Input of layer l depends on parameters $\theta^{(l-1)}$ of previous layer

$$\theta^{(l-1)} \rightarrow \theta^{(l-1)} + \Delta\theta$$

When parameters update, input distribution $P(x^{(l)})$ changes

Quantifying Covariate Shift

$$D_{KL}(P_{\text{old}}(x) \parallel P_{\text{new}}(x))$$

Measure shift magnitude using **KL-Divergence** between old and new distributions

Calculate batch mean

$$\sigma^2_B = (1/m) \sum (x_i - \mu_B)^2$$

Calculate batch variance

$$\hat{x}_i = (x_i - \mu_B) / \sqrt{(\sigma^2_B + \epsilon)}$$

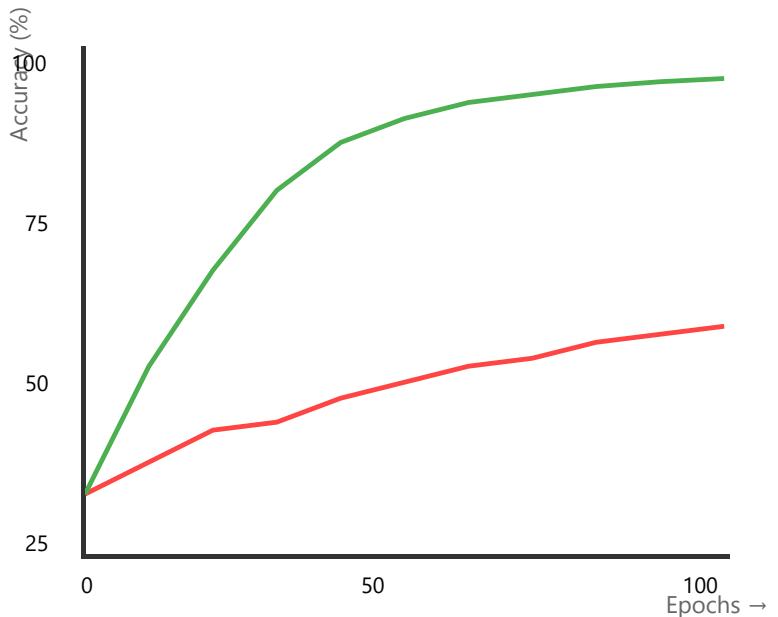
Normalized value (mean 0, variance 1)

$$y_i = \gamma \hat{x}_i + \beta$$

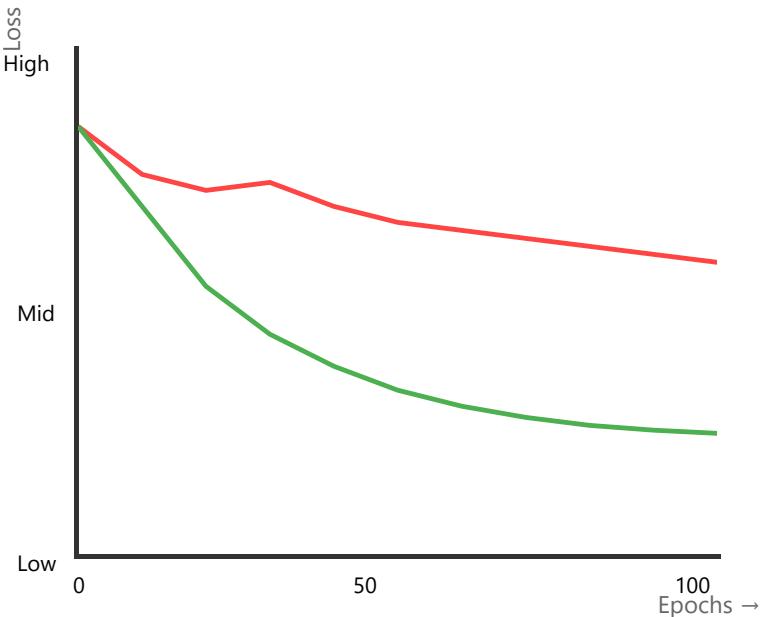
Scale and shift with learnable γ, β parameters

Experimental Results

Training Accuracy Over Epochs



Loss Over Epochs



Without
Normalization

With Batch
Normalization

Without
Normalization

With Batch
Normalization

Before vs After Comparison

✗ Without Normalization

Learning Rate **0.001 (Very Low)**

Convergence Time **~200 epochs**

Training Stability **Unstable**

Gradient Flow **Poor**

Distribution Behavior



✓ With Batch Normalization

Learning Rate **0.01-0.1 (Higher)**

Convergence Time **~50 epochs**

Training Stability **Stable**

Gradient Flow **Excellent**

Distribution Behavior



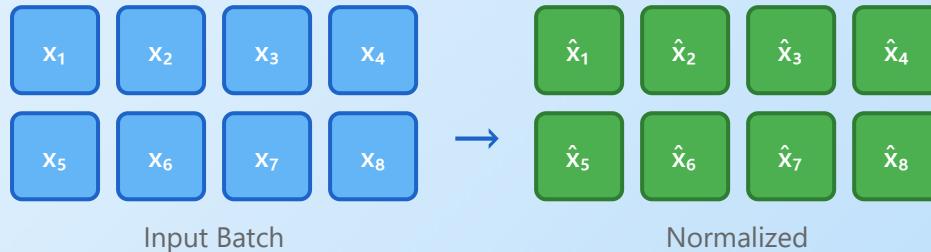
4 Pages Total | Scroll down to see more →

💡 Key Insight

Batch Normalization allows **10-100x higher learning rates** and reduces training time by **~75%**

Batch Normalization (Batch Norm)

Normalizes Across Batch Dimension



Batch Norm Formula

$$\hat{x} = (x - \mu_{\text{batch}}) / \sqrt{\sigma^2_{\text{batch}} + \epsilon}$$

$$y = \gamma \cdot \hat{x} + \beta$$

Learnable: γ (scale) & β (shift)

Key Benefits

Reduces internal covariate shift significantly

Enables higher learning rates and faster convergence

Acts as regularizer, reduces need for dropout

Normalizes inputs across the batch dimension

Learnable parameters (γ, β) maintain expressiveness

✓ Most Widely Used

Standard normalization technique in **Convolutional Neural Networks (CNNs)**



How Batch Normalization Works



Internal Covariate Shift Problem

As neural networks get deeper, the **data distribution changes** progressively through each layer, causing problems.

Sigmoid Activation Function

Steepest Region:
-2 to 2

Early Layers

Data in -2~2 range

Fast Learning

Deep Layers

Data drifts away

Slow Learning (Vanishing Gradient)



Batch Normalization Solution

Re-normalize data at each layer to maintain optimal distribution throughout the network.

Applied at Every Layer:

- 1 Calculate batch mean and variance
- 2 Normalize data to -2~2 range
- 3 Scale with γ and shift with β
- 4 Pass to next layer

Without Normalization



Slow Convergence

With Normalization



Fast Convergence



Key Insight 1: Alleviates Vanishing Gradient

Keeps data in the steepest region of sigmoid (-2~2), significantly reducing the vanishing gradient problem.



Key Insight 2: Faster Training

Transforms the loss function from a valley shape to a bowl shape, finding the optimum much faster.



Key Insight 3: Regularization Effect

Normalizing with slightly different mean/variance per batch naturally adds noise, providing regularization similar to Dropout.



Numerical Calculation Example

1 Input Batch Data

Batch Size: **4 samples**

Input values: $x = [1, 3, 5, 7]$

2 Calculate Mean

$$\mu = (1 + 3 + 5 + 7) / 4$$

$$\mu_{\text{batch}} = 4.0$$

3 Calculate Variance

$$\sigma^2 = [(1-4)^2 + (3-4)^2 + (5-4)^2 + (7-4)^2] / 4$$

$$\sigma^2 = [9 + 1 + 1 + 9] / 4$$

$$\sigma^2_{\text{batch}} = 5.0$$

4 Normalize

$$\hat{x} = (x - \mu) / \sqrt{\sigma^2 + \varepsilon}$$

$\varepsilon = 0.001$ (for stability)

$$\hat{x}_1 = (1 - 4) / \sqrt{5.001} = -1.34$$

$$\hat{x}_2 = (3 - 4) / \sqrt{5.001} = -0.45$$

$$\hat{x}_3 = (5 - 4) / \sqrt{5.001} = 0.45$$

$$\hat{x}_4 = (7 - 4) / \sqrt{5.001} = 1.34$$

5 Apply Scale & Shift

Learnable Parameters:

$$\gamma \text{ (scale)} = 2.0$$

Final Formula:

$$y = \gamma \cdot x^+ + \beta$$

β (shift) = 1.0

$y = 2.0 \cdot x^+ 1.0$

⭐ Final Output Values

y_1

-1.68

y_2

0.10

y_3

1.90

y_4

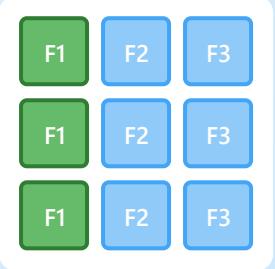
3.68

Original: [1, 3, 5, 7] → After Normalization: [-1.68, 0.10, 1.90, 3.68]

Layer Normalization (Layer Norm)

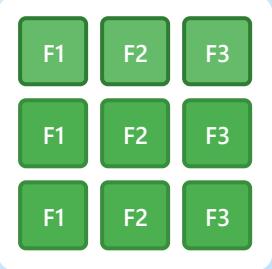
Batch Norm vs Layer Norm

Batch Norm



Across Batch ↓

Layer Norm



Across Features →

Normalizes across **feature dimension** per sample

Key Features

Independent of batch size during training/inference

Each sample normalized **independently**

No dependence on **batch statistics**

Essential for **RNNs and Transformers** (sequential models)

✓ Standard in NLP

Used in modern NLP architectures:
BERT, GPT, and Transformers

✓ Independent of Batch Size

Works with batch size = 1

Each sample normalized independently



Layer Normalization Formula

$$\mu = (1/H) \sum x_i$$

Mean: Average value of all features in the layer

$$\sigma^2 = (1/H) \sum (x_i - \mu)^2$$

Variance: Average of squared deviations from the mean

$$x^{\hat{}} = (x - \mu) / \sqrt{\sigma^2 + \epsilon}$$

Normalization: Transform to mean 0 and variance 1

$$y = \gamma \odot x^{\hat{}} + \beta$$

Scale & Shift: Adjust with learnable parameters

γ (**gamma**): Scale parameter (learnable)

β (**beta**): Shift parameter (learnable)

ϵ (**epsilon**): Small constant for numerical stability (e.g., 10^{-5})

H : Number of hidden units in the layer



What is Internal Covariate Shift?

Internal Covariate Shift refers to the phenomenon where the distribution of inputs to each layer continuously changes during neural network training. As the parameters of previous layers are updated, the statistical characteristics of the inputs received by the current layer change with each iteration.

⚠️ Problems that Arise

- Training instability: Each layer must constantly adapt to changing input distributions
- Slow convergence: Requires smaller learning rates
- Gradient problems: Risk of vanishing/exploding gradients
- Worse in deep networks: Cumulative effect increases with depth

✓ Layer Normalization Solution

- Stabilizes input distribution: Normalizes each sample independently
- Faster training: Enables larger learning rates
- Improved gradient flow: Stabilizes gradients during backpropagation
- Consistent representations: Model can learn stable features



Why Layer Norm is Used in Transformers

1 Independent Sequence Processing

Transformers process each input sequence independently. Rather than sharing statistics across batches like Batch Normalization, normalization needs to be applied independently for each sample (sentence).

2 Variable-Length Sequence Handling

Sentences vary in length, and each token carries different meanings. Layer Norm normalizes across the feature dimension of each token, operating stably regardless of sequence length.

3 Self-Attention 안정화

Self-Attention 메커니즘에서 각 토큰은 시퀀스의 모든 토큰을 참조합니다. Layer Normalization은 이러한 attention 연산 전후에 적용되어 표현을 일관되게 유지하고 학습 수렴을 빠르게 합니다.

4 배치 크기 독립성

추론(inference) 시 배치 크기가 1일 수 있습니다. Layer Norm은 배치 크기에 의존하지 않으므로 학습과 추론에서 동일하게 동작하여 안정적입니다.



주요 NLP 모델에서의 Layer Norm 적용

BERT

GPT-2/3/4

T5

RoBERTa

BART

Instance Normalization

Normalizes Per Instance, Per Channel



Each **spatial map** normalized **separately**

Key Features

Normalizes each sample and channel independently

Removes **instance-specific contrast** information

Preserves **content** while normalizing style

Better than Batch Norm for **style-related tasks**

Style vs Content



Normalizes
Style

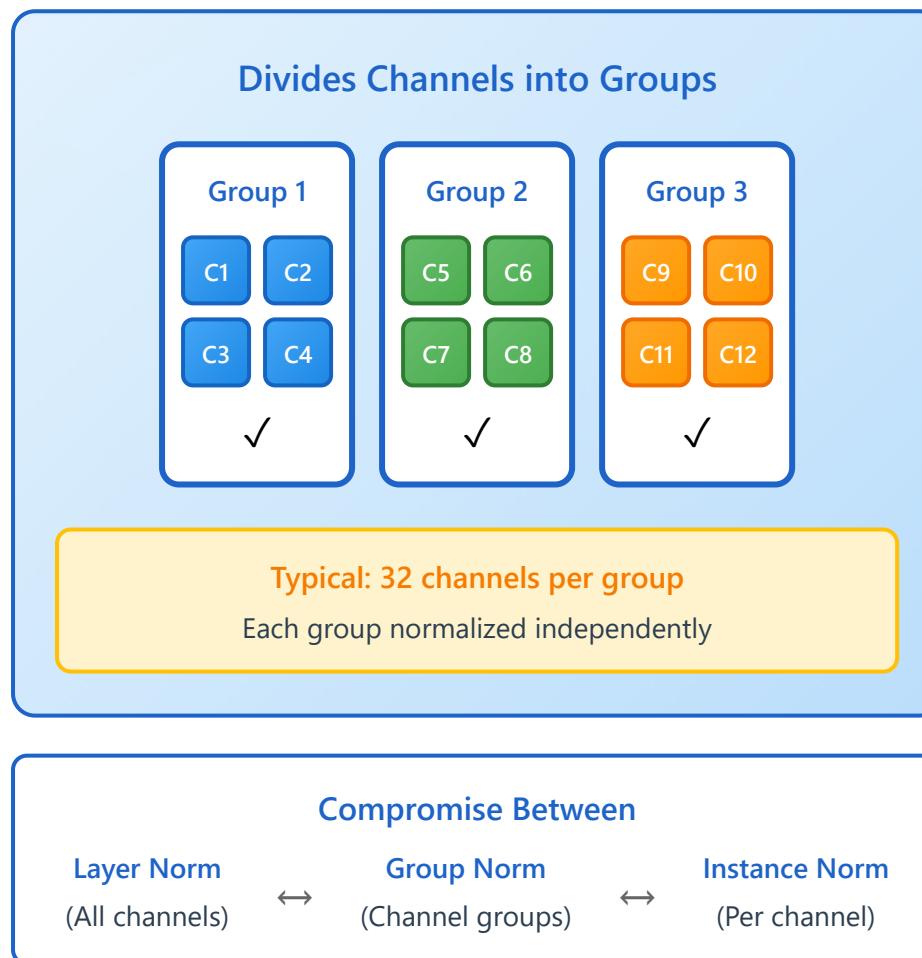


Preserves
Content

✓ Primary Applications

Used in **Style Transfer** and **GANs** for image generation tasks

Group Normalization



Key Features

Independent of batch size like Layer Norm

Better than Layer Norm for visual tasks (CNNs)

Stable performance across different batch sizes

Compromise between Layer Norm and Instance Norm

✓ Ideal For

Small batch training in tasks like object detection and segmentation

Weight Normalization

Reparameterization

$$w = g \cdot (v / \|v\|)$$

g

Magnitude
Scale

v / ||v||

Direction
Normalized



Weight Norm vs Batch Norm

Key Features

Decouples magnitude (**g**) and direction (**v**) of weights

Normalizes **weights** rather than activations

Reduces **parameters** compared to Batch Norm

Faster training than Batch Norm in some cases

✓ Works Well With

RNNs and **Reinforcement Learning** applications

Weight Norm

Normalizes weights

Batch Norm

Normalizes activations

Fewer params

g and v only

More params

γ , β + statistics

Less commonly used in modern architectures

Spectral Normalization

Constrains Spectral Norm

$$\tilde{W}_{SN} = W / \sigma(W)$$

Divides weights by **largest singular value** (σ)

✓ Ensures Lipschitz Continuity

Controls gradient flow through network layers

Critical for GAN Training

Generator

Creates images

Discriminator

Spectral Norm
Evaluates images

Key Features

Constrains **spectral norm** of weight matrices

Ensures **Lipschitz continuity** of network layers

Prevents **gradient explosion** in discriminator

Improves training stability **without batch statistics**

✓ Standard in GANs

Used in modern GAN architectures: **BigGAN**,
StyleGAN

Comparison of Normalization Techniques

Technique	Best Use Cases			Key Characteristics
Batch Norm	CNNs	Large batches		Normalizes across batch dimension
Layer Norm	RNNs	Transformers	Small batches	Independent of batch size
Instance Norm	Style Transfer	GANs		Per instance, per channel
Group Norm	Small batch vision	Detection	Segmentation	Groups channels, batch-independent
Weight Norm	RNNs	Reinforcement Learning		Normalizes weights, not activations
Spectral Norm	GAN discriminators	BigGAN	StyleGAN	Ensures Lipschitz continuity

Key Insight: Each normalization technique addresses different aspects of training stability. Choose based on your architecture, batch size constraints, and specific task requirements.

When to Use Which Normalization?



Large Batch CNNs
Training CNNs with large batch sizes



Batch Normalization



Transformers / RNNs
Sequential models, language tasks



Layer Normalization



Style Transfer

Artistic tasks, style manipulation



Small Batch Detection

Object detection, segmentation



Instance Normalization

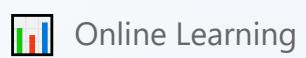


Discriminator stability in GANs



Spectral Normalization

Group Normalization



Single sample or streaming data



Layer / Group Norm

⚠ Key Considerations



Batch Size
Constraints



Computational
Cost



Architecture
Type



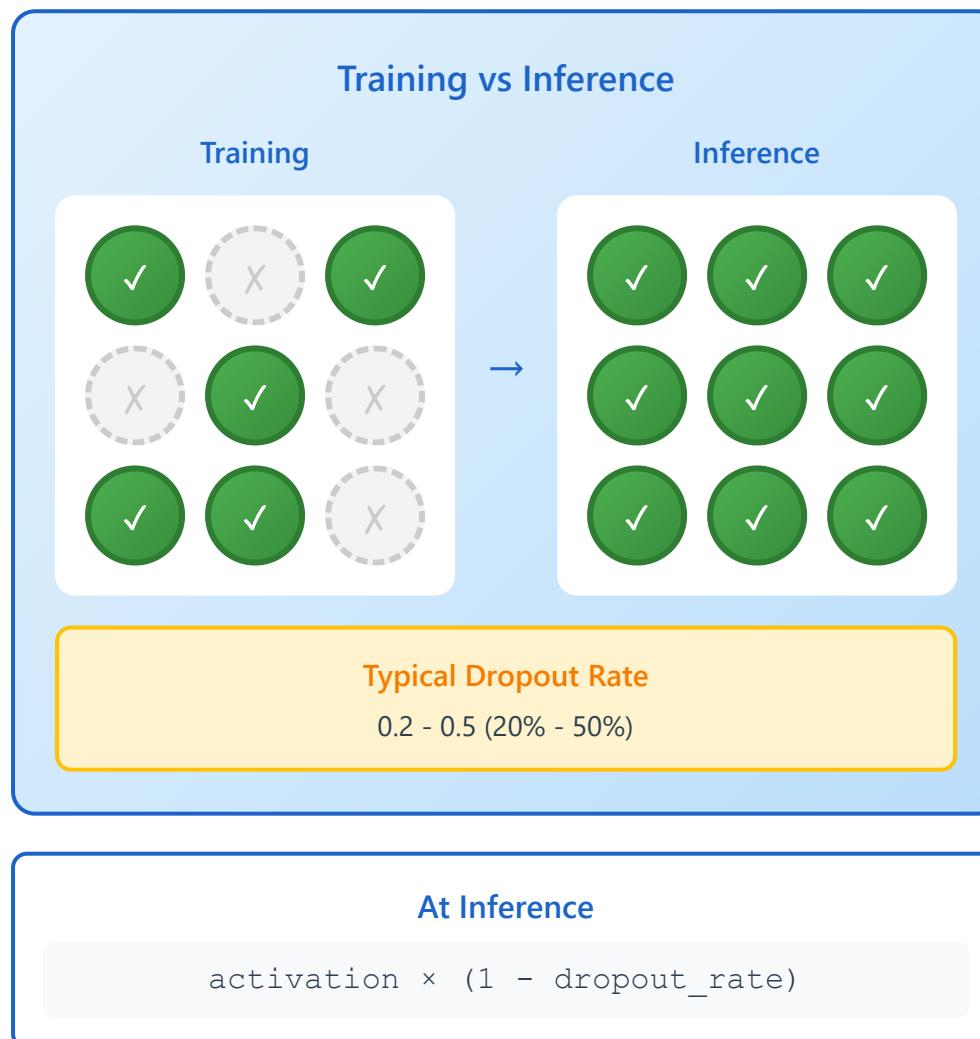
Task
Requirements

Part 3/3:

Normalization and Generalization

- 19.** Principles of Dropout
- 20.** Dropout Variants (DropConnect, DropBlock)
- 21.** Stochastic Depth
- 22.** Data Augmentation Strategies
- 23.** Mixup and CutMix
- 24.** Early Stopping
- 25.** Ensemble Methods

Principles of Dropout



Key Principles

Randomly drops neurons during training
(output set to 0)

Prevents **co-adaptation** of neurons

Trains **ensemble of sub-networks**
simultaneously

Powerful regularization technique

✓ Main Benefit

Reduces **overfitting significantly** in large networks

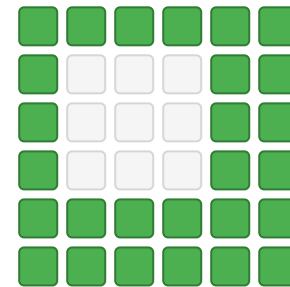
Dropout Variants (DropConnect, DropBlock)

DropConnect



Drops **weights** instead of activations

DropBlock

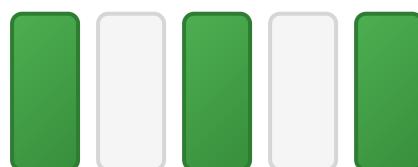


Drops **contiguous regions** in feature maps

✓ Better for CNNs

Prevents info leaking

Spatial Dropout



Drops entire **feature channels**

DropPath

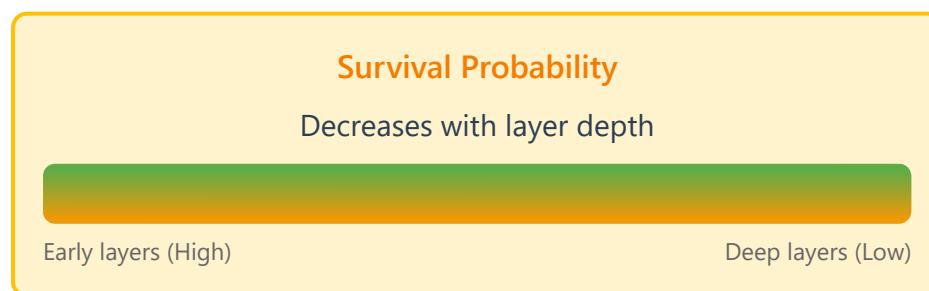
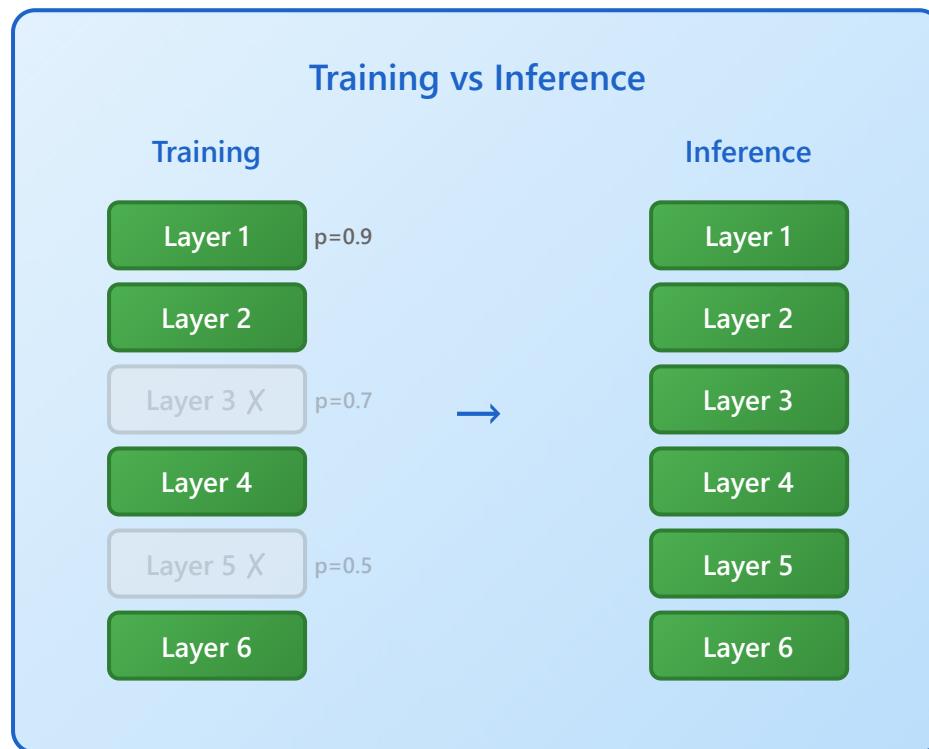
Branch 1

Branch 2 X

Branch 3

Drops entire **paths** in multi-branch architectures

Stochastic Depth



Key Benefits

Randomly drops **entire layers** during training

Creates **shorter networks** during training

Improves **gradient flow** to early layers

Reduces **training time** significantly

✓ **Enables Extreme Depth**

Training of **extremely deep networks** (1000+ layers)

Data Augmentation Strategies



Geometric



Flips, rotations, crops



Color Jittering



Brightness, contrast, saturation



Cutout



Randomly mask patches



Rotation



Random angle rotations



Random Crop



Crop different regions



Random Erasing



Erase random regions

✓ Key Benefits



Increases training diversity



Improves generalization



No additional data needed

Mixup and CutMix

Mixup

Linear Interpolation



Formula

$$x_{\text{mix}} = \lambda \cdot x_i + (1-\lambda) \cdot x_j$$

$$y_{\text{mix}} = \lambda \cdot y_i + (1-\lambda) \cdot y_j$$

✓ Blends entire images

✓ Smooth interpolation

CutMix

Cut and Paste Regions



Mixing Ratio

$$\lambda \sim \text{Beta}(\alpha, \alpha)$$

Paste cut region to another

✓ Retains local features

✓ More natural appearance

Shared Benefits

Smoother decision boundaries

Improved robustness

Better calibration

Early Stopping



⌚ Patience Parameter

Wait N epochs before stopping

Key Features

- Monitors validation performance during training
- Stops when validation loss **stops improving**
- Prevents **overfitting** to training data
- Saves **best model** based on validation metric

✓ Simple & Effective

Widely used regularization technique, often combined with learning rate scheduling

Ensemble Methods



Averaging



Mean Prediction

Averages predictions from multiple models



Voting



Majority Vote

Takes majority vote for classification



Stacking



Meta-Model

Trains meta-model on base predictions



Typical Size



3-10 models in ensemble

✓ Key Benefits



Reduces variance across predictions



Improves robustness of the model



Trades computational cost for higher accuracy

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com