Lecture 8:

Loss, Optimization and Scheduling

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

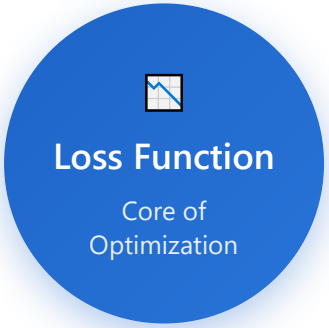
Part 1/3:

Loss Function Design

1. Role and Importance of Loss Functions
2. Regression Losses - MSE, MAE, Huber
3. Classification Loss - Cross-Entropy
4. Hinge Loss and SVM
5. Focal Loss - Class Imbalance
6. Contrastive Loss
7. Triplet Loss
8. Regularization Terms (L1, L2, L1+L2)
9. Custom Loss Function Design

Loss Function: Role & Importance

Understanding the Core Component of Neural Network Training



1 Quantification

Quantifies difference between predicted and actual values

2 Gradient Direction

Guides optimization process by providing gradient direction

3 Task-Specific Design

Different loss functions needed per task (regression vs classification)

4 Performance Impact

Direct impact on model performance and behavior

5 Objective Function

Acts as the objective function that the model seeks to minimize

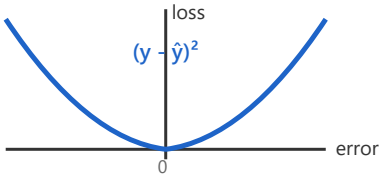
6 Convergence & Accuracy

Affects convergence speed and final model accuracy

Regression Loss Functions Comparison

MSE

Mean Squared Error (L2)



✓ Strong penalty for large errors

✗ Sensitive to outliers

Characteristics from squaring operation

Huber Loss

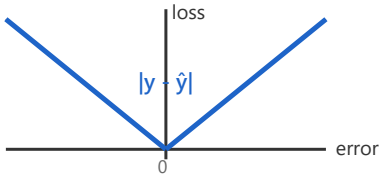
Hybrid Approach

✓ Combines MSE and MAE advantages

✓ Adjustable with delta parameter

MAE

Mean Absolute Error (L1)



✓ Robust to outliers

Uniform penalty regardless of error size

↓

Use MSE when

Outliers are important

Use Huber when

Balanced approach is needed

Use MAE when

Robustness is required

📊 Calculation Examples with 5 Data Points

Sample	Actual (y)	Predicted (ŷ)	Error (y - ŷ)
1	100	98	2
2	150	153	-3
3	200	199	1
4	180	182	-2
5	**120**	**140**	**-20**

MSE Calculation

Formula: MSE = $(1/n)\Sigma(y - \hat{y})^2$

Steps:

- $(2)^2 = 4$
- $(-3)^2 = 9$
- $(1)^2 = 1$
- $(-2)^2 = 4$
- $(-20)^2 = \mathbf{400}$

Sum = 418

MSE = 418 ÷ 5

MSE = 83.6

Huber Loss (δ=5)

Formula:

If $|error| \leq \delta$: $0.5 \times error^2$

If $|error| > \delta$: $\delta \times (|error| - 0.5\delta)$

Steps:

- $|2| \leq 5$: $0.5 \times 4 = 2$
- $|3| \leq 5$: $0.5 \times 9 = 4.5$
- $|1| \leq 5$: $0.5 \times 1 = 0.5$
- $|2| \leq 5$: $0.5 \times 4 = 2$
- $|20| > 5$: $5 \times (20-2.5) = \mathbf{87.5}$

Sum = 96.5

Huber = 96.5 ÷ 5

Huber = 19.3

MAE Calculation

Formula: MAE = $(1/n)\Sigma|y - \hat{y}|$

Steps:

- $|2| = 2$
- $|-3| = 3$
- $|1| = 1$
- $|-2| = 2$
- $|-20| = \mathbf{20}$

Sum = 28

MAE = 28 ÷ 5

MAE = 5.6

💡 **Key Insight:**

Notice sample #5 with error of -20 (outlier): **MSE = 83.6** (heavily penalized), **Huber = 19.3** (balanced penalty), **MAE = 5.6** (moderate penalty). Huber Loss provides a middle ground between MSE and MAE!

Classification Loss: Cross-Entropy

Measuring dissimilarity between predicted and true distributions

📊 Core Concept

Measures the difference between predicted probability distribution and actual labels, outputting probability distribution through Softmax activation

✓ Encourages high confidence for correct classes

✓ Larger penalty for confident wrong predictions

≡ Equivalent to maximizing log-likelihood of correct class

1 Binary Cross-Entropy

Used for binary classification problems

2 Classes (0 or 1)

2 Categorical Cross-Entropy

Used for multi-class classification problems

N Classes (Multi-class)

📝 Calculation Examples

Binary Cross-Entropy Examples

Categorical Cross-Entropy Examples

Loss = -[y·log(p) + (1-y)·log(1-p)]

True=1, Pred=0.95
Loss = 0.051

True=1, Pred=0.70
Loss = 0.357

True=0, Pred=0.10
Loss = 0.105

True=1, Pred=0.50
Loss = 0.693

True=1, Pred=0.20
Loss = 1.609

Loss = -Σ y·log(p) (only correct class)

True=[0,1,0], Pred=[0.1,0.8,0.1]
Loss = 0.223

True=[1,0,0], Pred=[0.7,0.2,0.1]
Loss = 0.357

True=[0,0,1], Pred=[0.2,0.2,0.6]
Loss = 0.511

True=[0,1,0], Pred=[0.4,0.4,0.2]
Loss = 0.916

True=[1,0,0], Pred=[0.2,0.5,0.3]
Loss = 1.609

Hinge Loss and Support Vector Machines

Maximum Margin Classification

🎯 Primary Loss Function for SVM

Loss function focused on maximizing the margin between classes

Margin-based Penalty: Penalizes only predictions within margin or misclassified

Confident Predictions: Encourages correct predictions beyond the margin

Outlier Robustness: Less sensitive to outliers compared to cross-entropy

Hinge Loss Variants

L Linear Hinge Loss

$$\max(0, 1 - y \cdot f(x))$$

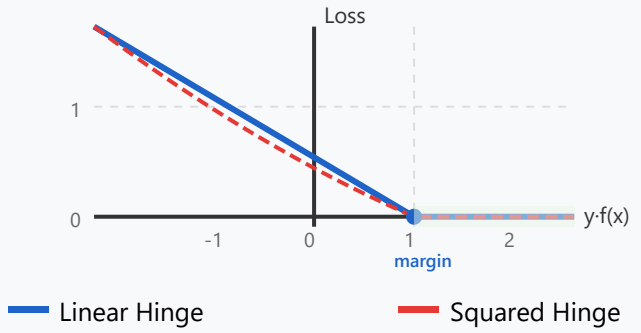
Standard hinge loss with margin-based linear penalty

S Squared Hinge Loss

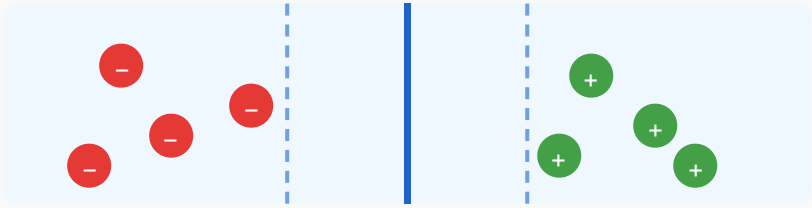
$$\max(0, 1 - y \cdot f(x))^2$$

Smoother, differentiable variant

📈 Hinge Loss Graph



🎯 SVM Margin Concept



✓ Use Cases

⚡ Key Advantages

- ✓ Binary classification problems
- ✓ Large-margin classifiers
- ✓ Text classification & document categorization

- ✓ Maximizes decision boundary margin
- ✓ Good generalization on unseen data
- ✓ Sparse solution (support vectors)

Focal Loss: Addressing Class Imbalance

Focusing on Hard Examples

🎯 Target Problem

Designed to solve severe class imbalance problems

Aligns learning with business metrics or task requirements

Focal Loss Formula

$$FL = -\alpha(1-p)^{\gamma} \log(p)$$

Working Mechanism

- ↓ Down-weights loss for well-classified examples
- ↑ Focuses learning on hard, misclassified examples
- ✓ Greatly improves minority class performance

📈 Loss Comparison: CE vs Focal Loss

Key Parameters

γ Focusing Parameter

Controls down-weighting rate

Reduces contribution of well-classified examples

Typical value: γ = 2

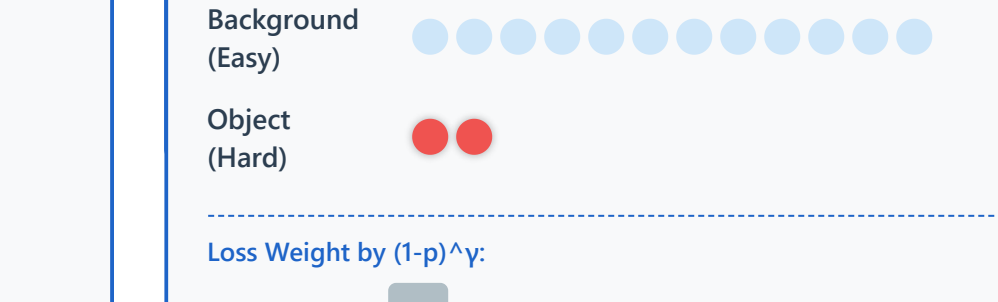
α Balancing Parameter

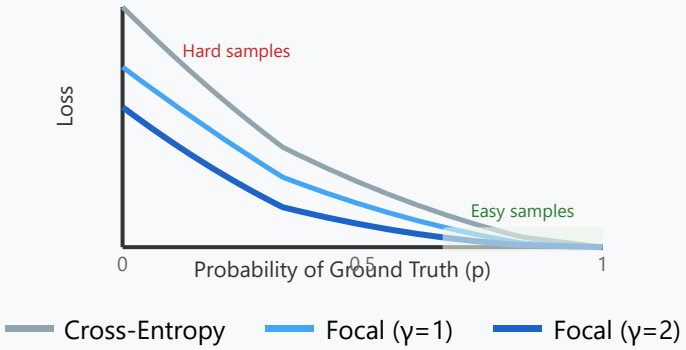
Addresses class imbalance problem

Adjusts per-class weights

Addresses class imbalance

⚖️ How Focal Loss Reweights Samples





Cross-Entropy Focal (γ=1) Focal (γ=2)

Easy (p=0.9) 0.01

Hard (p=0.2) 0.64

✓ Easy samples contribute little → Model focuses on hard examples

Contrastive Loss

Learning Similarity Metrics in Embedding Space

🎯 Core Concept

Learns meaningful representations in embedding space by training on sample pairs (similar/dissimilar)

Training Approach

Can learn meaningful representations without explicit labels

↓

Similar Pairs

Place similar samples close together in embedding space

Minimize Distance

→

Dissimilar Pairs

Place dissimilar samples apart by a margin

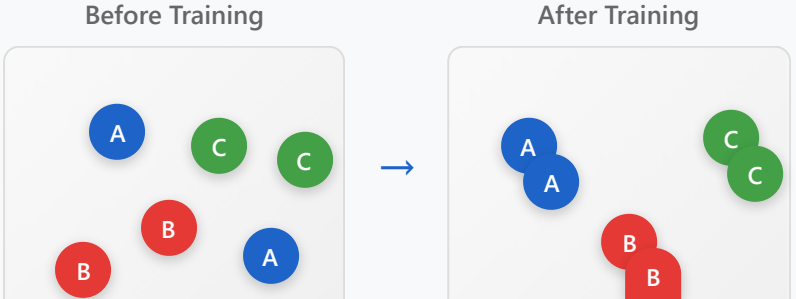
Maintain Margin

Applications

1. Face Recognition
2. Signature Verification
3. Image Retrieval

Foundation for Siamese Networks

Embedding Space Transformation

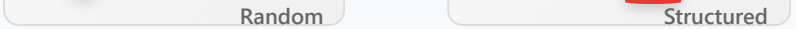


Loss Function Components

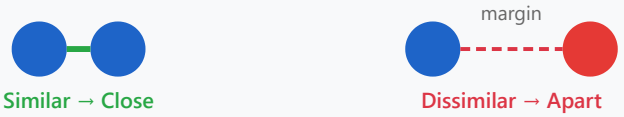
$$L = (1-Y)\cdot\tfrac{1}{2}\cdot D^2 + Y\cdot\tfrac{1}{2}\cdot\max(0, m-D)^2$$

Y=0 (Similar)
Minimize D^2
Pull together

Y=1 (Dissimilar)
Push if D < margin
Maintain distance



Similar pairs cluster together • Dissimilar pairs separate



Triplet Loss

Extension of Contrastive Loss with Triplet Structure

Triplet Components

Anchor Reference sample

Positive Sample similar to anchor

Negative Sample different from anchor

Loss Formula

$$\max(d(a,p) - d(a,n) + \text{margin}, 0)$$

Penalizes when anchor is closer to negative than to positive

Efficiency

More efficient than Contrastive Loss
(One optimization step per triplet)

Triplet Mining

Careful triplet selection strategy needed for effective learning

- **Hard Negatives:** Most difficult negative samples
- **Semi-Hard Negatives:** Medium difficulty negative samples

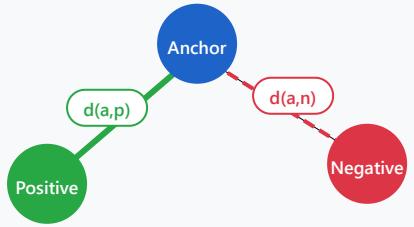
Mining Approaches

- Online Mining
- Offline Mining

Applications

1. Face Recognition (FaceNet)
2. Person Re-identification
3. Fine-grained Similarity Learning

Triplet Structure in Embedding Space



Goal: d(a,p) + margin < d(a,n)

Learning Process

d(a,p) becomes smaller • **d(a,n)** becomes larger

✓ Positive should be closer than Negative by at least a margin

Regularization Terms: L1, L2, and Elastic Net

Preventing Overfitting through Model Complexity Constraints

🎯 **Regularization prevents overfitting by constraining model complexity**

L2 Regularization

Ridge

$$\lambda \sum w^2$$

- ✓ Penalizes sum of squared weights
- ✓ Encourages small weights
- ✓ Smooth solutions
- ○ Does not enforce sparsity

L1 Regularization

Lasso

$$\lambda \sum |w|$$

- ✓ Penalizes sum of absolute weights
- ✓ Promotes sparsity
- ✓ Enables automatic feature selection
- ○ Sets some weights to zero

Elastic Net

L1 + L2 Combined

$$\lambda_1 \sum |w| + \lambda_2 \sum w^2$$

- ✓ Combines advantages of L1 and L2
- ✓ Balances sparsity and grouping effect
- ✓ Handles correlated features
- ★ **Best of Both Worlds**

λ

Lambda (λ): Regularization Strength

Hyperparameter controlling regularization strength - larger values lead to stronger regularization effect

Custom Loss Function Design

Aligning Training with Domain-Specific Objectives

Why Custom Loss?

Standard loss functions may not sufficiently reflect domain-specific objectives

Align learning with business metrics or task requirements

Key Considerations

1. **Differentiability** - Ensure differentiability
2. **Computational Efficiency** - Fast computation
3. **Numerical Stability** - Avoid overflow/underflow

Multi-term Combination

$$L_{total} = w_1 \cdot L_1 + w_2 \cdot L_2 + \ldots + w_n \cdot L_n$$

Design Process

1. Verify reflection of true optimization goal
2. Combine multiple loss terms with weighting coefficients
3. Validate performance improvement with held-out data

Domain Examples

Computer Vision

Perceptual Loss, SSIM

Object Detection

Classification + Localization + Objectness

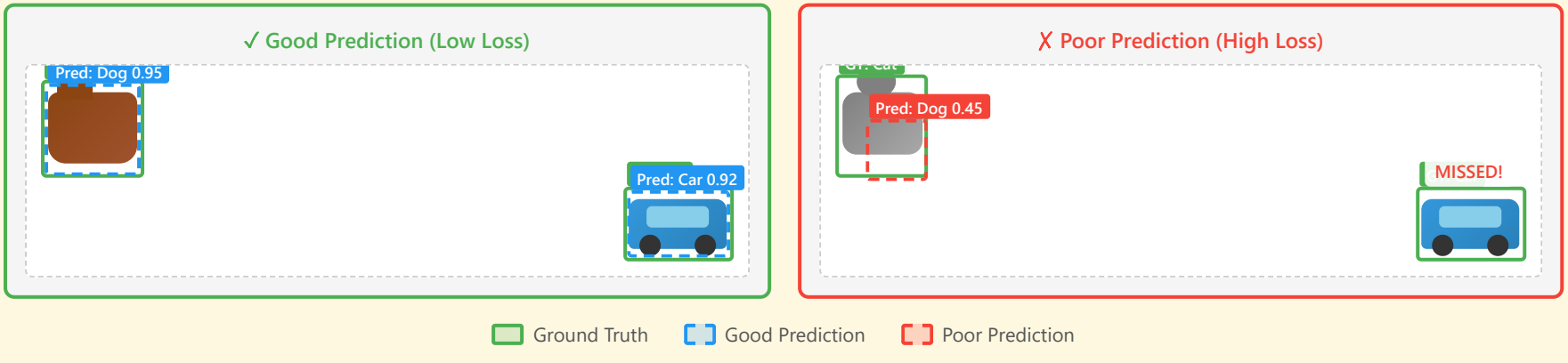
Medical Imaging

Sensitivity/Specificity balance

NLP Tasks

Semantic similarity, contextual consistency

Object Detection Loss Visualization: Success vs Failure Cases



① **Classification Loss:** Cross-entropy for class labels

$-\Sigma\, y \cdot \log(\hat{y})$

② **Localization Loss:** IoU/Smooth L1 for bbox coordinates

Smooth-L1(bbox_pred, bbox_gt)

③ **Objectness Loss:** Confidence score prediction

BCE(obj_pred, obj_gt)

④ **Miss Penalty:** False negative detection

Penalizes undetected objects

Total: $L = \lambda_1 \cdot L_{cls} + \lambda_2 \cdot L_{box} + \lambda_3 \cdot L_{obj}$

Part 2/3:

Optimization Algorithms

10. Gradient Descent Review
11. Batch vs Mini-Batch vs Stochastic
12. Momentum Method
13. Nesterov Accelerated Gradient
14. AdaGrad
15. RMSprop
16. Adam and AdamW
17. Second-Order Optimization - L-BFGS
18. Comparison of Optimization Algorithms

Gradient Descent Review

Foundation for Modern Neural Network Training

🎯 Core Concept

Iterative optimization algorithm to minimize loss function

Updates parameters in negative gradient direction

Update Rule

$$\theta = \theta - \eta \nabla L(\theta)$$

η: Learning Rate

How It Works

1. Calculate loss gradient for all parameters
2. Move parameters in opposite direction of gradient
3. Repeat until convergence

📊 Key Characteristics

Foundation of all modern neural network training algorithms

Convergence depends on learning rate, loss landscape, and initialization

⚠️ Challenges

- Can get trapped in local minima
- Saddle point problems

⚖️ Trade-off

Convergence speed ↔ Stability

Batch vs Mini-Batch vs Stochastic Gradient Descent

Balancing Speed, Stability, and Generalization

Batch GD

Entire Dataset

- ✓ Accurate gradient
- ✓ Stable convergence
- ✗ Slow speed
- ✗ Sharp minima

Mini-Batch GD

Subset of Data

- ✓ Balances speed and stability
- ✓ GPU parallel processing
- ✓ Better generalization
- ✓ Practical choice

Stochastic GD

Single Sample

- ✓ Fast updates
- ✓ Flat minima
- ✗ Noisy gradient
- ✗ Unstable convergence

📊 Typical Mini-Batch Sizes

32 64 128 256

Larger Batches

Gradients: → More stable
Hardware: → More efficient
Minima: → Sharp

Smaller Batches

Gradients: → Noisier
Generalization: → Better
Minima: → Flat

📐 Gradient Calculation Examples

Understanding How Each Method Computes Gradients

Example Setup

Dataset: N = 1000 samples | **Loss Function:** $L(\theta) = \frac{1}{2}(y - \hat{y})^2$ | **Learning Rate:** $\alpha = 0.01$

1 Batch Gradient Descent

Gradient:

$$\nabla L(\theta) = (1/N) \sum_{i=1}^{N} \nabla L(\theta, x_i, y_i)$$

Step 1: Calculate loss for all 1000 samples

Step 2: Compute average gradient: $\nabla L = (1/1000) \sum_{i=1}^{1000} \nabla L_i$

Step 3: Update parameters: $\theta \leftarrow \theta - 0.01 \times \nabla L$

Key Feature: 1 epoch = 1 parameter update (using all 1000 samples)

2 Mini-Batch Gradient Descent

Gradient (batch size B=100):

$$\nabla L(\theta) = (1/B) \sum_{i=1}^{B} \nabla L(\theta, x_i, y_i)$$

Batch 1: Use samples 1-100 → Compute ∇L_1 → $\theta \leftarrow \theta - 0.01 \times \nabla L_1$

Batch 2: Use samples 101-200 → Compute ∇L_2 → $\theta \leftarrow \theta - 0.01 \times \nabla L_2$

⋮ *Continue...*

Batch 10: Use samples 901-1000 → Compute ∇L_{10} → $\theta \leftarrow \theta - 0.01 \times \nabla L_{10}$

Key Feature: 1 epoch = 10 parameter updates (1000/100 = 10 batches)

3 Stochastic Gradient Descent

Gradient (single sample):

$$\nabla L(\theta) = \nabla L(\theta, x_i, y_i)$$

Sample 1: Use x_1, y_1 → Compute ∇L_1 → $\theta \leftarrow \theta - 0.01 \times \nabla L_1$

Sample 2: Use x_2, y_2 → Compute ∇L_2 → $\theta \leftarrow \theta - 0.01 \times \nabla L_2$

⋮ *Continue...*

Sample 1000: Use x_{1000}, y_{1000} → Compute ∇L_{1000} → $\theta \leftarrow \theta - 0.01 \times \nabla L_{1000}$

Key Feature: 1 epoch = 1000 parameter updates (update after each sample)

⚖ Method Comparison Summary

Method	Sample Size	Updates per Epoch	Computation Time	Gradient Accuracy
Batch GD	N = 1000	1	Slowest	Very accurate
Mini-Batch GD	**B = 100**	**10**	**Balanced**	**Sufficiently accurate**
Stochastic GD	B = 1	1000	Fast updates	Noisy

Momentum Method

Accelerating Gradient Descent with Velocity Accumulation

🚀 Core Concept

Accelerates gradient descent by accumulating velocity from past gradients

Update Rules

1. Velocity Update

$$v = \beta v + \nabla L(\theta)$$

2. Parameter Update

$$\theta = \theta - \eta v$$

Momentum Coefficient β: Typically 0.9
(Retains 90% of previous velocity)

✨ Key Benefits

1. Helps overcome local minima
2. Navigates ravines in loss landscape
3. Reduces oscillations in high curvature directions
4. Accelerates progress in consistent gradient directions
5. Dampens oscillations in narrow valleys

Physical Analogy

Similar to a ball rolling down a hill gaining momentum

Effect Visualization


Without Momentum
Zigzag movement
With Momentum
Smooth progress

Nesterov Accelerated Gradient (NAG)

Improved Momentum with Look-Ahead Gradient Calculation

🔍 Key Innovation

"Look-ahead" mechanism

Calculates gradient at anticipated future position

Update Rules

1. Velocity Update (Look-ahead)

$$v = \beta v + \nabla L(\theta - \beta v)$$

2. Parameter Update

$$\theta = \theta - \eta v$$

⚡ **Key Point:** Calculates gradient at θ - βv position for more accurate direction prediction

vs Standard Momentum

Momentum	Nesterov
Calculates gradient at current position	Calculates gradient at future position

✨ Advantages

- ✓ Better convergence characteristics than standard momentum
- ✓ Reduces overshooting through early trajectory correction
- ✓ Particularly effective for convex optimization problems

⚖️ **Trade-off:** Slightly higher computational cost than standard momentum

Widely Used

Widely used in major deep learning frameworks like PyTorch and TensorFlow

AdaGrad (Adaptive Gradient Algorithm)

Parameter-Specific Learning Rate Adaptation

🎯 Core Concept

Adaptively adjusts learning rate
for each parameter based on historical gradients

Update Rule

$$\theta = \theta - \eta/\sqrt{(G + \varepsilon)} \odot \nabla L(\theta)$$

G: Cumulative sum of squared gradients
⊙: Element-wise multiplication

Adaptation Mechanism

Infrequent Parameters
Larger updates

Frequent Parameters
Smaller updates

✨ Benefits

- ✓ No manual learning rate tuning needed
- ✓ Advantageous for sparse data

⚠ Main Drawbacks

- ! Learning rate monotonically decreases over time
- ! Aggressive decay can cause premature training stop

ε Epsilon Parameter

Prevents division by zero (typically 1e-8)

💡 Best Use Cases

- Sparse Data Processing
- Natural Language Processing

RMSprop (Root Mean Square Propagation)

Adaptive Learning Rate with Exponential Moving Average

Key Improvement

Solves AdaGrad's monotonically decreasing learning rate problem

Innovation

Uses exponential moving average of squared gradients

Update Rules

1. EMA of Squared Gradients

$$E[g^2] = \rho E[g^2] + (1-\rho)g^2$$

2. Parameter Update

$$\theta = \theta - \eta/\sqrt{(E[g^2] + \varepsilon)} \odot g$$

vs AdaGrad

AdaGrad	RMSprop
Monotonically decreasing LR	LR can dynamically increase/decrease

Advantages

- ✓ Learning rate can dynamically increase/decrease
- ✓ Effective for non-stationary objectives
- ✓ Well-suited for Recurrent Neural Networks

Origin

Introduced in Geoff Hinton's Coursera lectures (Unpublished paper)

Good default choice for

Decay Rate ρ: Typically 0.9
Maintains moving average of recent gradients

various optimization problems

Adam and AdamW

Most Popular and Improved Optimizers

Adam

Most popular optimization algorithm combining Momentum and RMSprop

AdamW

Improved Adam using decoupled weight decay regularization

Moment Estimates

First Moment
Mean

Second Moment
Variance

Default Parameters

- $\beta_1 = 0.9$ (Momentum)
- $\beta_2 = 0.999$ (RMSprop)
- $\varepsilon = 1e\text{-}8$ (Numerical stability)

Key Improvements

1. Fixes weight decay implementation bug in original Adam
2. Decouples L2 regularization from gradient-based optimization
3. Generally better generalization than Adam

Key Difference

Adam
Weight decay in gradient

AdamW
Weight decay decoupled

Includes **Bias Correction**
for moment estimates in early iterations

🏆

AdamW is the preferred choice in modern deep learning

Second-Order Optimization: L-BFGS

Limited-Memory Broyden-Fletcher-Goldfarb-Shanno

Core Concept

Optimization using second-order information (Hessian) with efficient inverse Hessian approximation

First-Order vs Second-Order

First-Order

- Uses gradient only
- More iterations
- Lower cost per iteration

Second-Order

- Uses Hessian
- Fewer iterations
- Higher cost per iteration

Characteristics

- ✓ Fewer iterations than first-order methods
- ✓ Deterministic method
- ✗ High computational cost per iteration ($O(n^2)$)

Performance Trade-off

Iterations	Per-iteration Cost
Fewer ✓	Higher ✗

Limitations

- ! Unsuitable for large-scale deep learning due to memory constraints
- ! Not suitable for mini-batch optimization

Suitable Applications

Small Models: Optimization of small-scale models

Scientific Computing: Scientific computing applications

Traditional ML: Logistic Regression, etc.

✗ Requires full batch evaluation

Specific Cases: When full batch evaluation is feasible

Comparison of Optimization Algorithms

Performance Characteristics and Trade-offs

Metric	SGD	Momentum	RMSprop	Adam/AdamW
Training Speed	Slow	Moderate	Fast	**Fastest**
Generalization	Good	**Best**	Moderate	Good
Memory Usage	**Lowest**	Low	Moderate	High
HP Sensitivity	High	Moderate	Low	**Lowest**

🎬 Optimizer Behavior Visualization

SGD

Slow, zigzag path through loss landscape

View Animation →

Momentum

Accelerates in consistent direction

View Animation →

RMSprop

Adaptive learning per parameter

View Animation →

Adam

Combines momentum + adaptive rates

View Animation →

SGD + Momentum

Adam/AdamW

RMSprop

Speed Ranking

Adam > RMSprop > Momentum > SGD

Simple, reliable
Good generalization
Requires tuning

Adaptive
Fast convergence
Out-of-the-box

Good for RNNs
Non-stationary
problems

Recommendation: Start with Adam/AdamW, then fine-tune with SGD+Momentum

Interactive Visualizations & Resources

- Gradient Descent Viz (GitHub)
- Interactive Optimizer Demo
- Why Momentum Works (Distill)
- Sebastian Ruder's Guide

Part 3/3:

Learning Rate Scheduling

19. Importance of Learning Rate
20. Fixed vs Adaptive Learning Rate
21. Step Decay
22. Exponential Decay
23. Cosine Annealing
24. Warm-up and Linear Schedule
25. Cyclical Learning Rates

Importance of Learning Rate

The Most Critical Hyperparameter

The most important hyperparameter affecting training dynamics and convergence

Learning Rate Impact

⚠ Too Large

- Training divergence
- Loss oscillation
- Loss explosion

⚠ Too Small

- Slow convergence
- Stuck in poor local minima

✓ Optimal (Dynamic)

Optimal learning rate changes throughout training

Training Phase Strategy

Early Training

Large learning rate for fast initial progress

Late Training

Small learning rate for fine-tuning and convergence

Key Insights

1. Different layers may benefit from different learning rates
2. Learning rate scheduling improves final model performance

Visual Comparison: How Learning Rate Affects Optimization

✗ Too Large (η = 1.0)

Too Small (η = 0.001)

✓ Optimal (η = 0.1)

Overshoots the minimum repeatedly, causing oscillation or divergence

Takes tiny steps, extremely slow progress toward the minimum

Balanced steps lead to smooth and efficient convergence

Restart Animation

Fixed vs Adaptive Learning Rate

Comparing Learning Rate Strategies

Fixed Learning Rate

Constant throughout training

- ✓ Simple implementation
- ✗ Suboptimal
- ✗ Requires careful manual tuning
- ✗ Difficult to find good value

Adaptive Learning Rate

Scheduled or metric-based variation

- ✓ Improved convergence
- ✓ Better final accuracy
- ✓ Fast early training
- ✓ Careful late refinement

Adaptive Learning Rate Types

Schedule-based

Predefined decay strategy

(Step, Exponential)

Performance-based

Reduce on validation metric plateau

(Plateau Detection)

✨ Key Benefits of Adaptive LR

- Combines benefits of fast early training and careful late refinement
- Improves both convergence speed and final model performance

Modern Practice: Always use some form of LR Scheduling

Step Decay Learning Rate Scheduling

Reducing Learning Rate at Fixed Intervals

📉 Core Concept

Reduce learning rate by a fixed factor at specific epoch intervals

Formula

$$LR = initial\_lr \times 0.1^{(epoch \,//\, step\_size)}$$

Reduce by 0.1× every step_size epochs

💡 Common Schedule

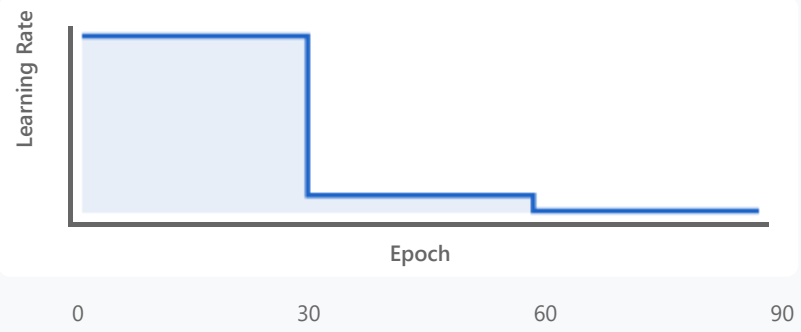
Reduce by 0.1× every 30 epochs

- **Epoch 0-29:** LR = 1.0 × initial_lr
- **Epoch 30-59:** LR = 0.1 × initial_lr
- **Epoch 60-89:** LR = 0.01 × initial_lr

⚖️ Characteristics

- ✓ Simple to implement and understand
- ✓ Effective when training duration is known
- ✓ Can align drops with learning regime changes
- ⚠ May cause sudden jumps in loss when LR drops

Learning Rate Progression



🎯 Widely Used In

Computer Vision (ResNet Training Schedule)

Exponential Decay Learning Rate Scheduling

Gradual Continuous Reduction

📉 Core Concept

Gradually decrease learning rate
following an exponential function

Formula

$$LR = initial\_lr \times decay\_rate^{(epoch\ /\ decay\_steps)}$$

Decay Rate: Typically 0.94-0.99
(per epoch or step)

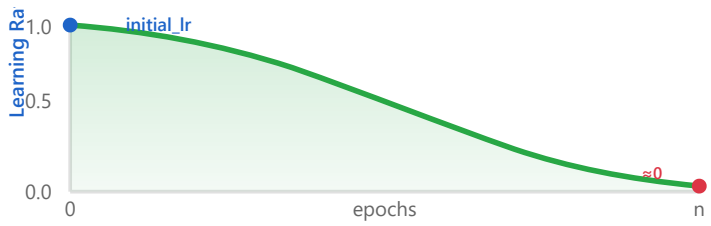
vs Step Decay

Step Decay	Exponential
Sudden changes	Smooth transition
Staircase reduction	Continuous reduction

⚖️ Characteristics

- ✓ Smoother than step decay
- ✓ Smooth transition without sudden changes
- ✓ Effective for long training runs
- ⚠ Can be too aggressive if decay rate poorly tuned

Decay Curve Visualization



🎯 Commonly Used In

- Reinforcement Learning
- NLP

Cosine Annealing

Smooth Learning Rate Scheduling with Cosine Function

🔍 Core Concept

Learning rate follows a cosine function from maximum to minimum value

Formula

$$LR = min\_lr + (max\_lr - min\_lr) \times (1 + \cos(\pi t/T)) / 2$$

t: current epoch, T: total epochs

Decay Behavior

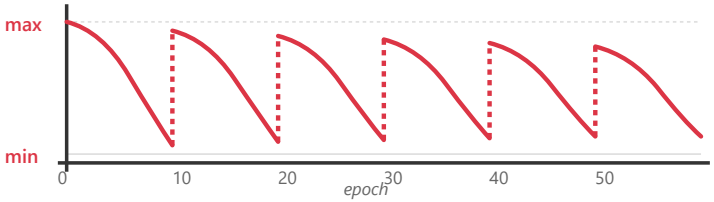
Early Phase

Fast reduction

Late Phase

Slow reduction (Fine-tuning)

Cosine Curve Visualization



✨ Key Advantages

- ✓ Smooth and gradual decay
- ✓ No hyperparameters to tune (only max/min LR)
- ✓ Improves generalization (empirically proven)

🔄 Warm Restarts (SGDR)

Can include restarts to escape local minima

🎯 Popular In

Computer Vision

Transformer Training

Warm-up and Linear Schedule

Gradual Start with Progressive Decay

1 Warm-up Phase

- ↑ Gradual increase from low value
- 🛡 Prevents large gradient updates from random initialization
- 📊 Typically first 5-10% of training

2 Linear Decay Phase

- ↓ Linear decrease from peak to minimum
- 🎯 Remaining 90-95% of training

🔗 Combined Schedule

Warm-up → Linear or Cosine Decay

📈 Schedule Visualization

✨ Key Benefits

- ✓ Stabilizes training in early phase
- ✓ Especially effective with large batch sizes
- ✓ Essential for large-scale model training

🎯 Essential For

BERT | GPT | Vision Transformers | LLMs

Cyclical Learning Rates

Oscillating Between Bounds for Better Exploration

Core Concept

Learning rate varies cyclically
between minimum and maximum values

Range Policies

Triangular: Linear increase/decrease

Triangular2: Range halves each cycle

Exponential: Exponential variation

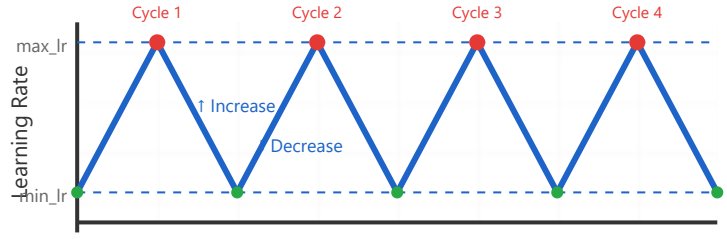
Parameters

Cycle Length: Typically 2-10 epochs per cycle

1cycle Policy

Single cycle including
warm-up and cool-down phases

Cyclical Pattern Visualization



Key Benefits

- ✓ Escape saddle points & explore loss landscape
- ✓ Better generalization performance
- ✓ Faster convergence
- ✓ Regularization effect
- ✓ Find better local minima through exploration

**Successfully used in
fast.ai training methodology**

Learn more about PyTorch LR Schedulers →

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com