上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 16: Training Deep Networks

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

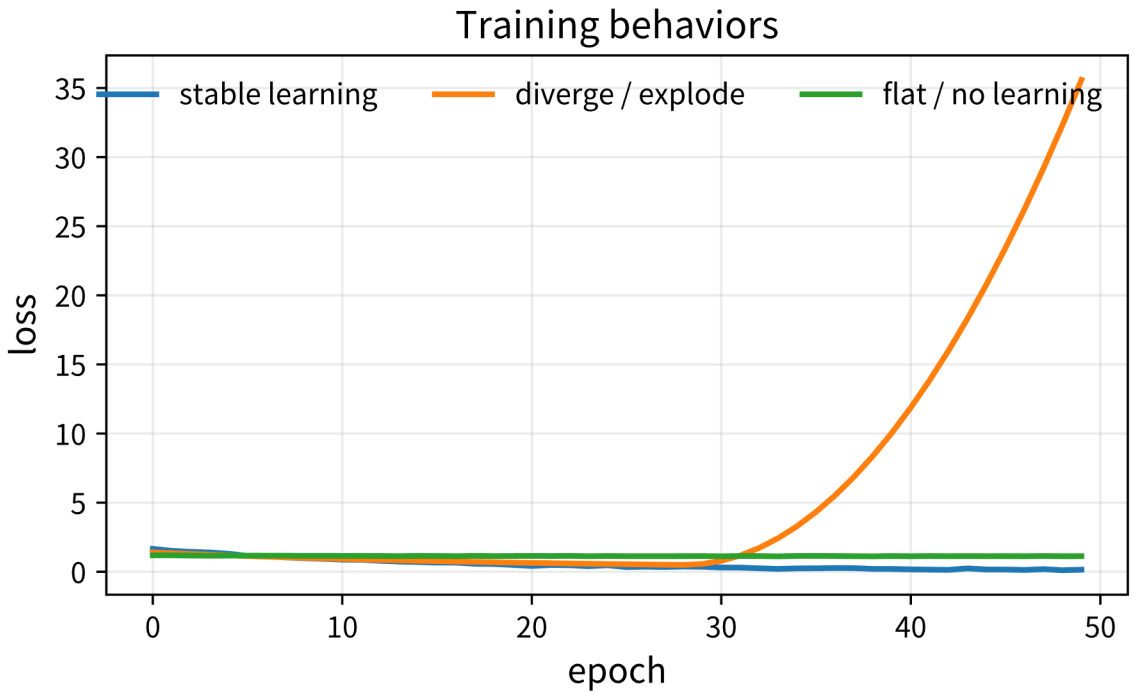
If Backprop Works, Why Is Training Still Hard?

We now have:

- MLPs
- Backpropagation
- Gradient descent

But deep nets still fail:

- No learning / flat loss
- Unstable / diverging loss
- Overfitting



Question:

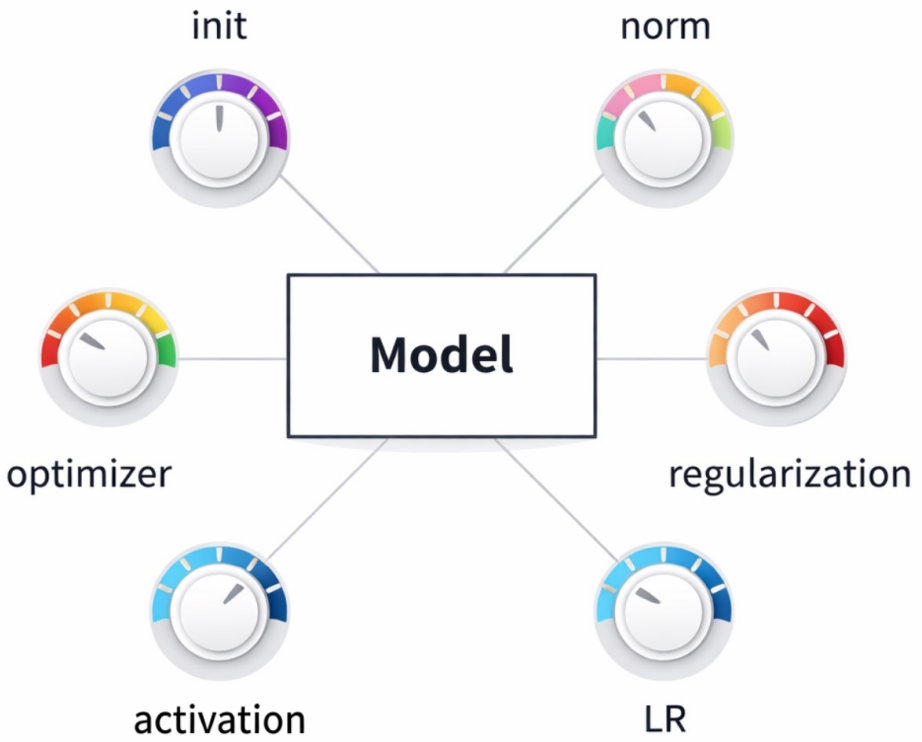
- Why does exact gradient computation not guarantee good training?

Training Is a Systems Problem

Same architecture, different outcomes.

Why?

- Initialization
- Activation
- Normalization
- Optimizer
- Learning rate
- Regularization



Message:

- Training depends on interacting design choices.

Objectives

By the end of this lecture, you should be able to:

- **Explain** why deep networks are hard to train, even with backpropagation.
- **Analyze** how initialization, normalization, optimizer choice, and learning rate affect optimization stability.
- **Design** a basic training recipe for deep networks and diagnose common failure modes.

1. Why Deep Training Fails

Three Failure Modes

1. Gradients vanish
2. Gradients explode
3. Model overfits

Two fundamental axes:

- Optimization
- Generalization

Vanishing Gradients

In deep chains:

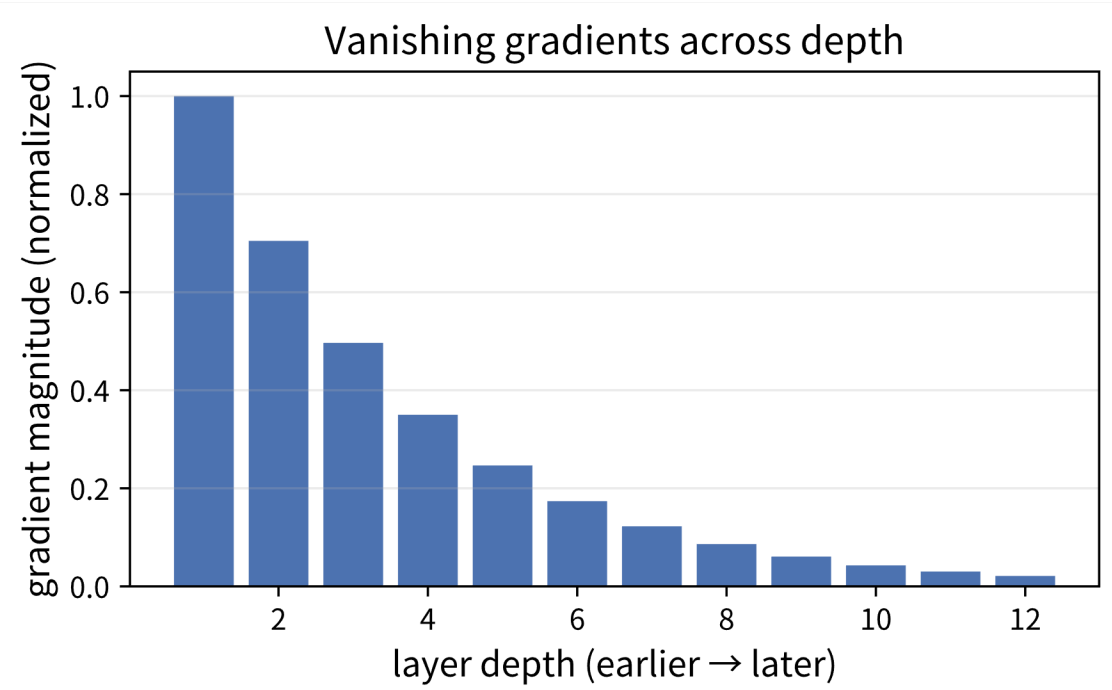
$$\frac{\partial \mathcal{L}}{\partial x} = \prod_{l} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

If many factors are < 1:

- Gradients shrink exponentially
- Early layers learn slowly
- Deep optimization stalls

Typical causes:

- Saturation
- Poor initialization
- Too much depth without control



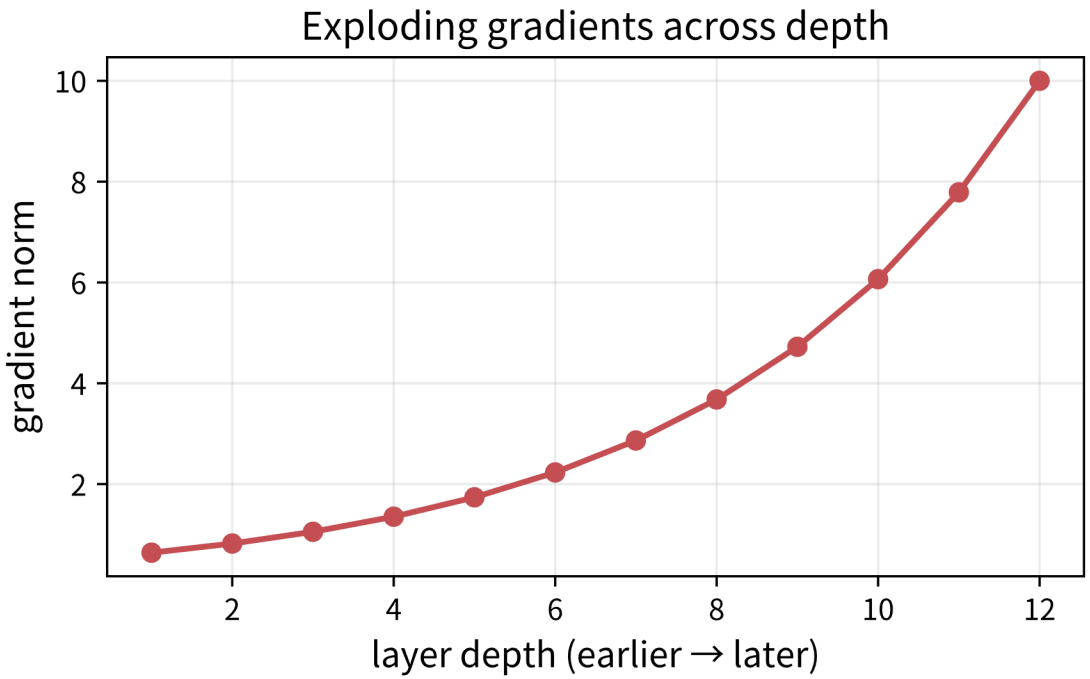
Exploding Gradients

If many factors are > 1:

- Gradients grow exponentially
- Updates become unstable
- Loss may diverge
- Activations may blow up

Symptoms:

- Huge updates
- Oscillation
- NaNs (Not a Number)



Depth Multiplies Scale Errors

Backprop repeatedly applies:

- Weight matrices
- Activation derivatives

Small scale bias compounds.

Key idea:

- Deep optimization is a signal propagation problem.

Which Activation Is More Fragile?

Compare hidden layers using:

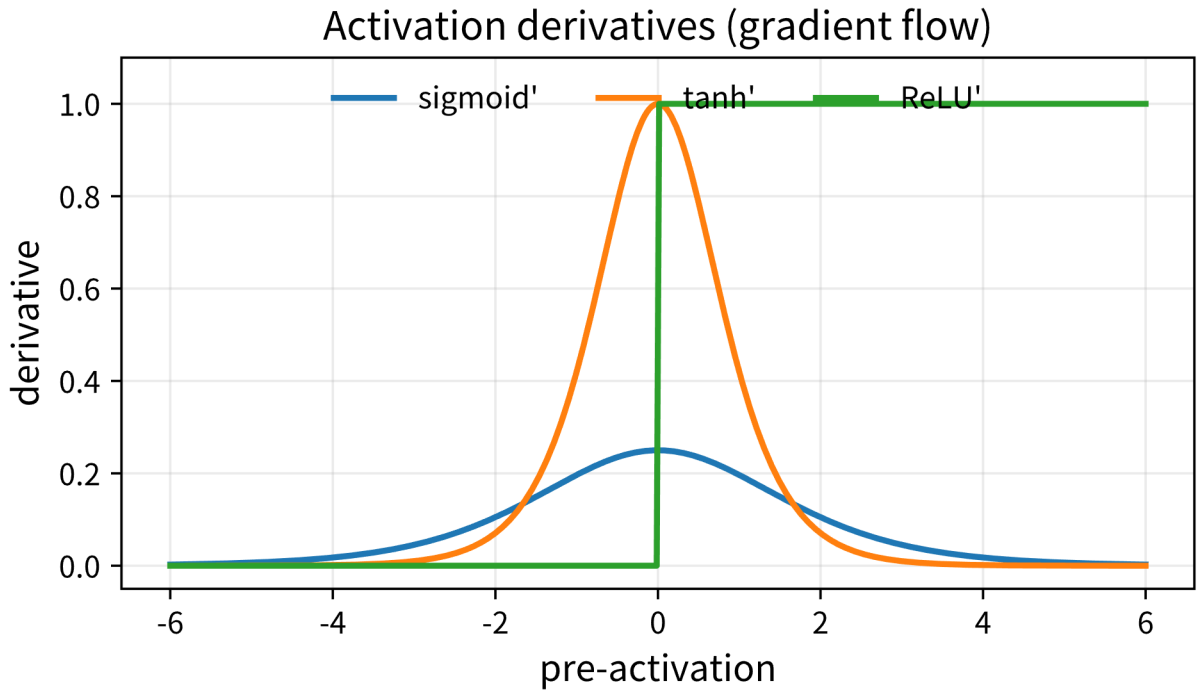
- Sigmoid
- Tanh
- ReLU

Question:

- Which is most likely to cause vanishing gradients? Why?

Answer:

- Sigmoid is worst; tanh is better but still saturates; ReLU preserves gradients on the positive side.



Optimization Success ≠ Generalization Success

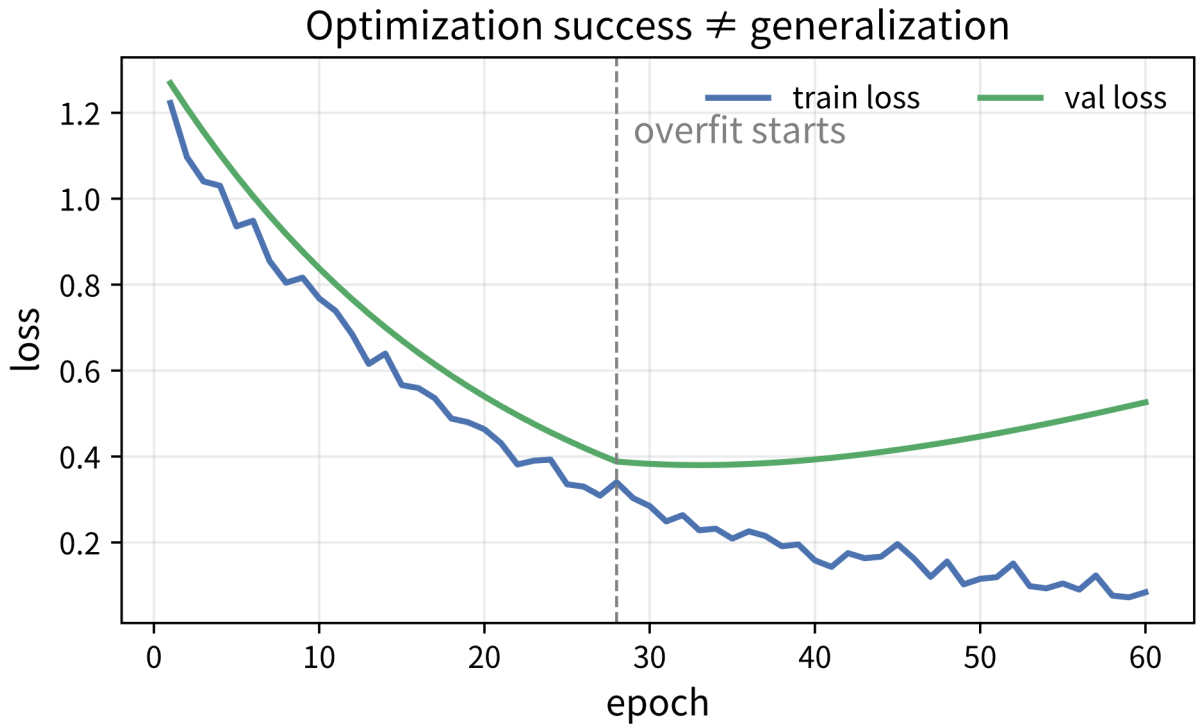
A model may fit training data well but fail on test data.

Why?

- High capacity
- Many parameters
- Limited data
- Memorization

Training has two goals:

- Reduce loss
- Generalize

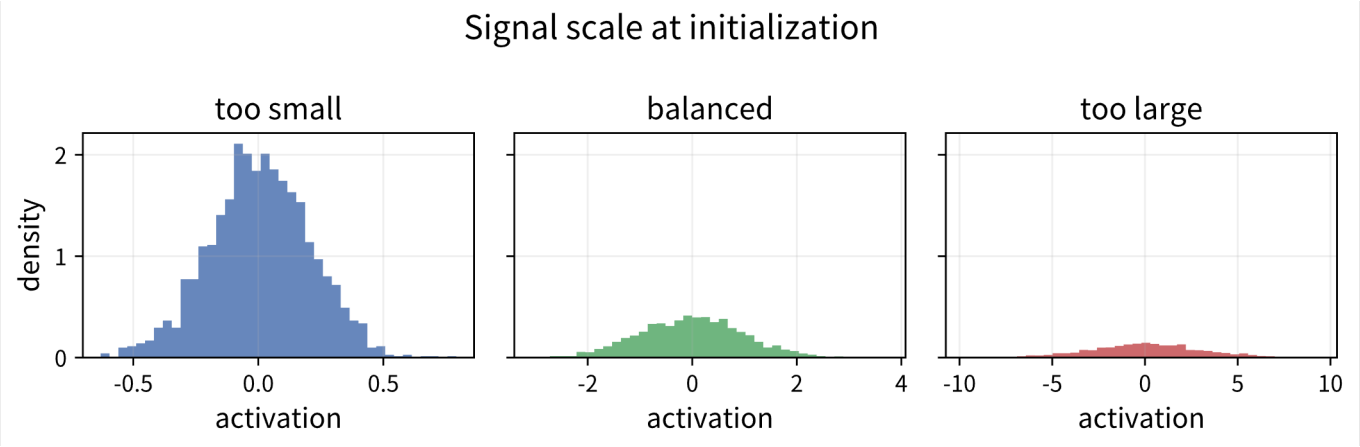


2. Stabilizing Signal Flow

Initialization Is Not a Detail

Bad initialization can cause:

- Activation explosion
- Activation collapse
- Vanishing gradients
- Exploding gradients
- Symmetry problems



Goal:

- Keep forward and backward signals well-scaled.

Why Not Initialize All Weights to Zero?

If neurons start identical:

- Same output
- Same gradient
- Same update

Then they stay **identical**.

Need:

- Random weights.

Practice:

- Biases: often zero
- Weights: not identical

Xavier Initialization

For tanh-like activations:

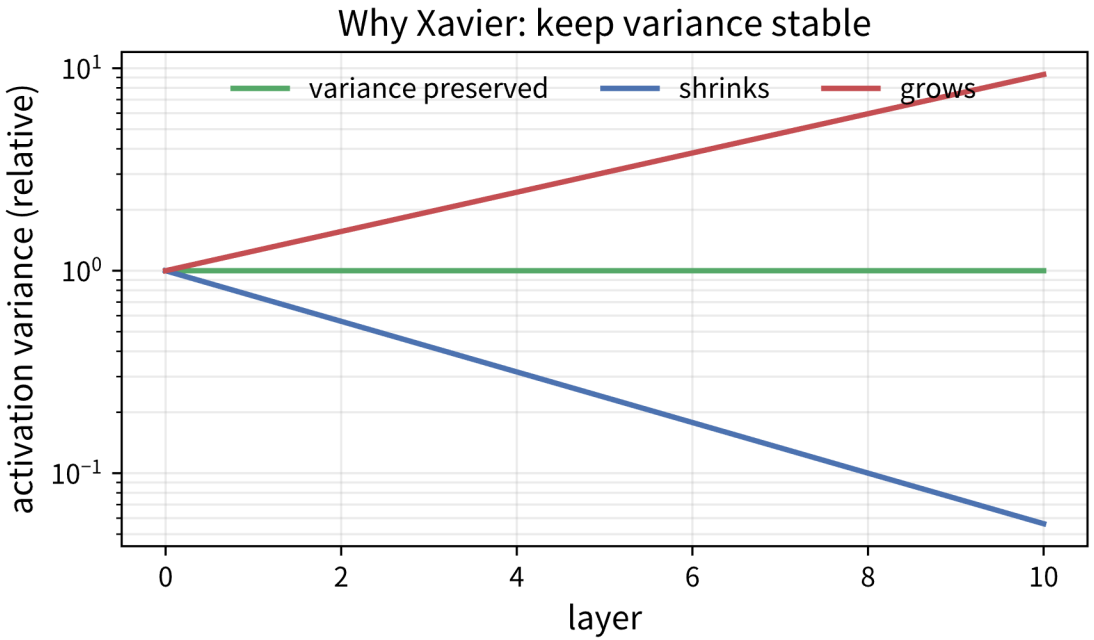
Choose weight scale to preserve variance.

Common form:

$$W_{ij} \sim \mathcal{U}\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right]$$

Idea:

Prevent systematic growth or shrinkage through layers.



Kaiming Initialization

For ReLU-like activations:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

Why larger than Xavier?

ReLU zeros out half of the signal, so we double the variance to compensate.

Rule of thumb:

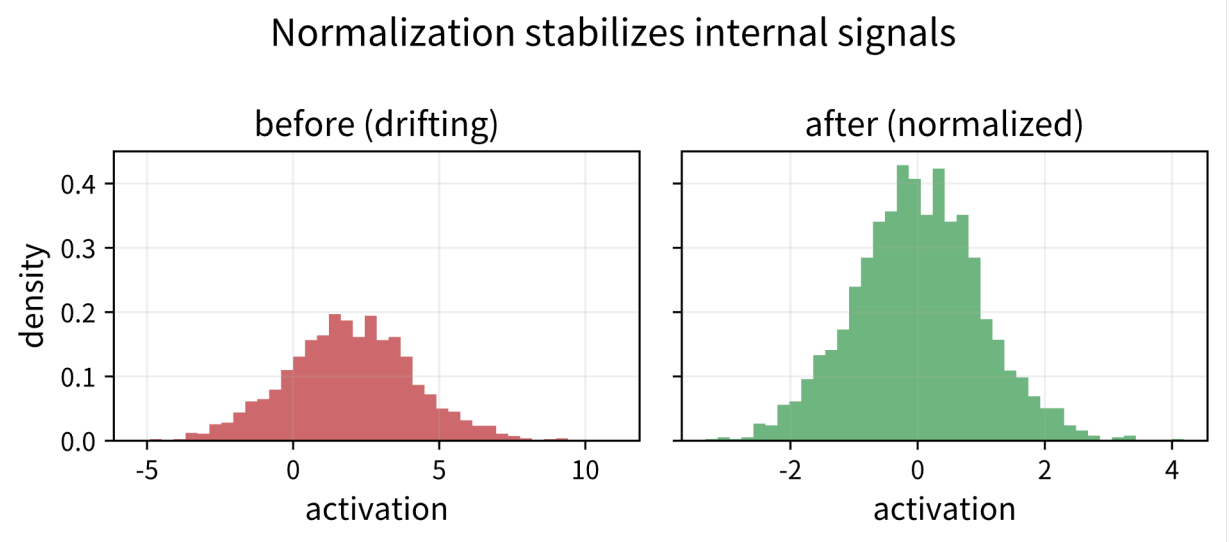
Activation	Initialization	Intuition
tanh, sigmoid	**Xavier**	Linear around zero
ReLU, LeakyReLU	**Kaiming**	Zeros half the signal

Normalization Stabilizes Internal Signals

Deep nets train more easily when activations stay well-scaled.

Normalization helps:

- Stabilize distributions
- Smooth optimization
- Improve gradient flow
- Allow larger learning rates



Most famous example:

- Batch Normalization (BatchNorm)

BatchNorm

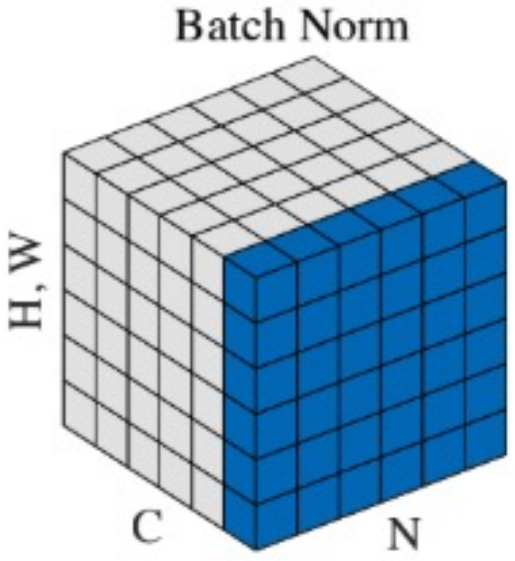
For mini-batch activations x:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

μ : mean σ: standard deviation

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

γ β: learnable scale & shift



Effect:

- Normalize
- Then learn scale and shift

Benefits:

- Faster training
- More stable gradients
- Some regularization

BatchNorm Helps, But It Is Not Magic

It does not:

- Fix all optimization problems
- Replace good initialization
- Eliminate learning-rate tuning

Also (Training vs Test):

- Uses batch statistics in training
- Uses running averages at test time

3. Optimizing Better

Why Plain SGD Is Not Enough

SGD:

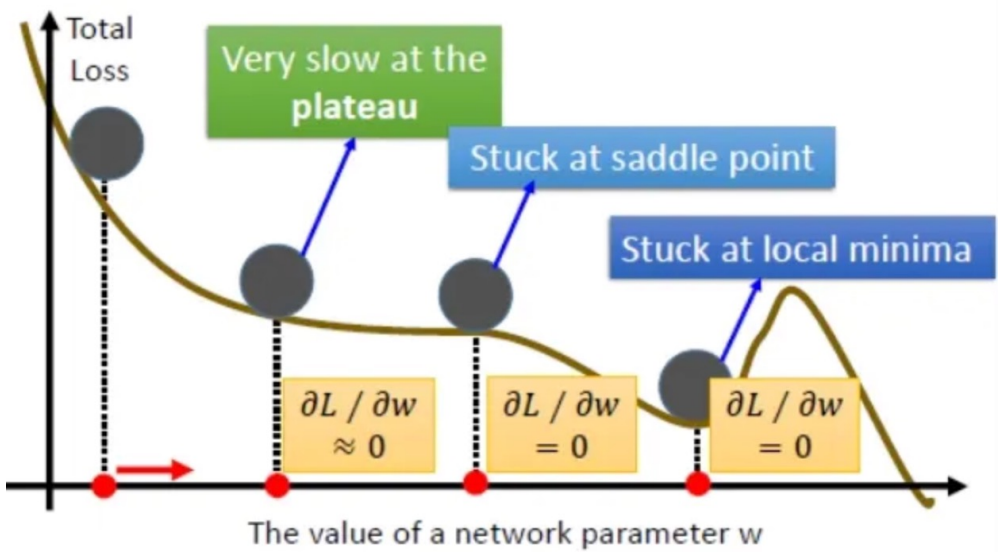
$$\theta \leftarrow \theta - \eta g$$

η: learning rate

Problems:

- Noisy updates
- Sensitive learning rate
- Slow in ill-conditioned valleys
- Oscillation

Need better update dynamics.



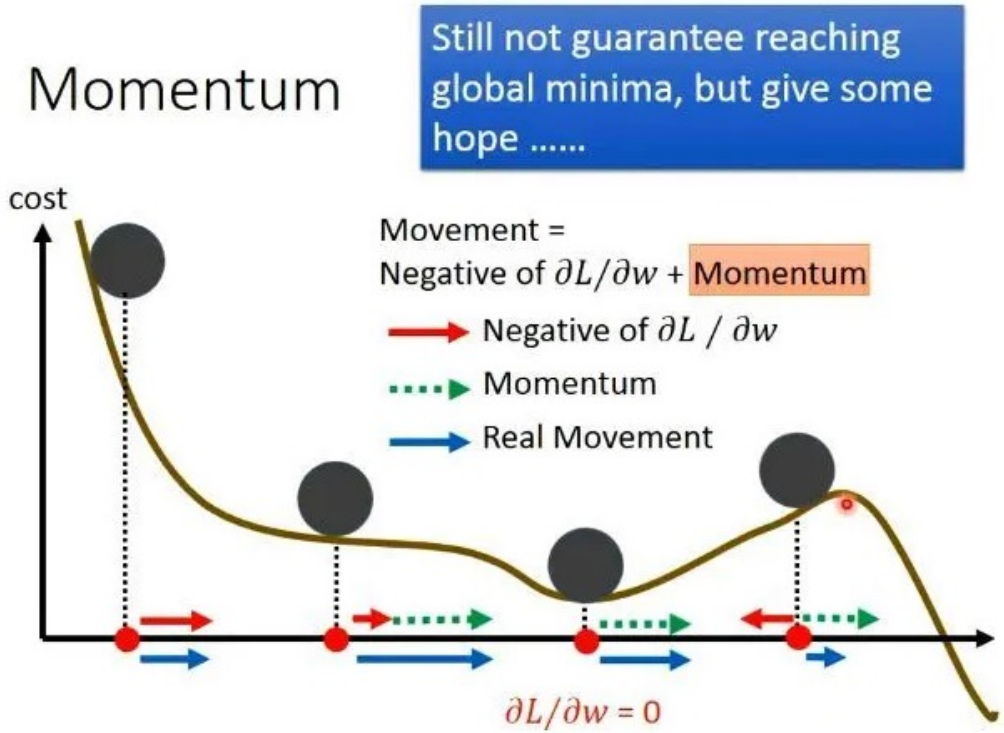
Momentum

Update rule:

$$v_t = \beta v_{t-1} + g_t$$

$$\theta_t = \theta_{t-1} - \eta v_t$$

β: momentum factor



Effect:

- Smooth noise
- Accelerate in consistent directions
- Reduce zig-zagging

RMSProp and Adam

Adaptive methods rescale updates per parameter.

Adam combines:

- Momentum
- Adaptive second moment

High-level update:

$$\theta \leftarrow \theta - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Why useful:

- Robust default
- Good for large models
- Less manual tuning than plain SGD

Optimizer	Strength	Use Case
SGD	Simple, well-understood generalization	Often used in vision (with Momentum)
Momentum	Smooths noise, accelerates progress	Standard for CNNs
Adam	Adaptive rates, robust to hyperparams	NLP, Transformers, huge models

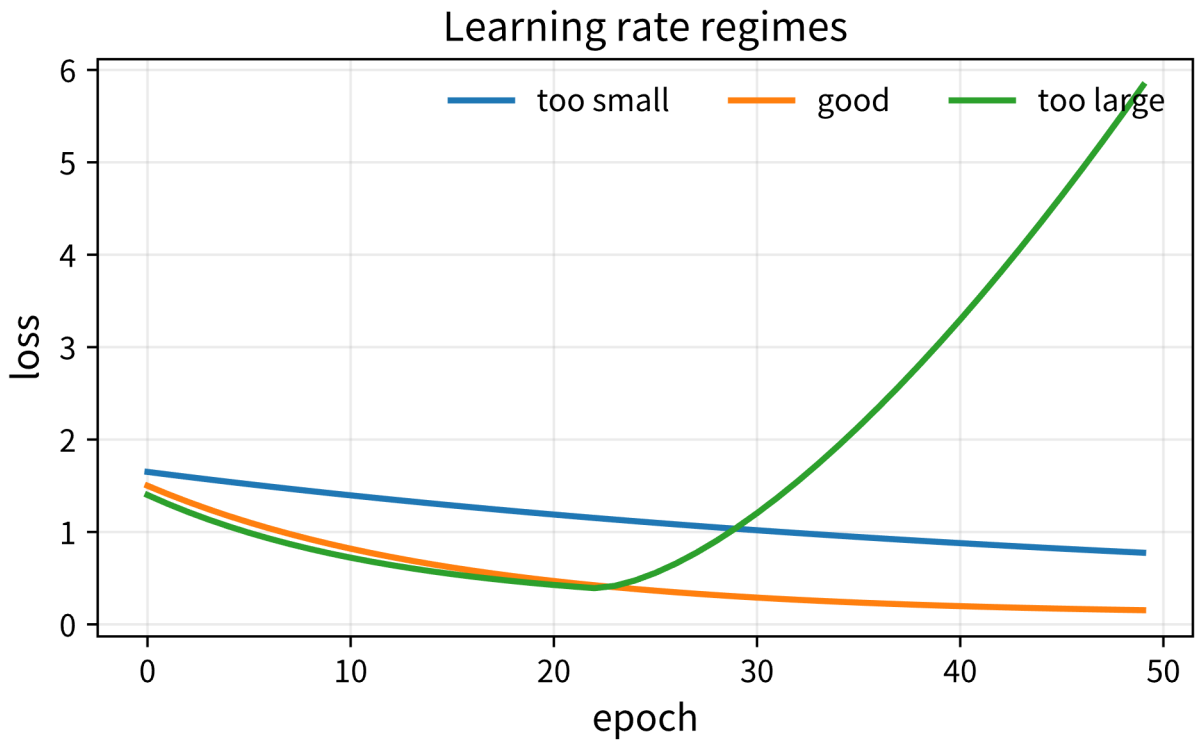
Learning Rate Is Often the Master Knob

Too small:

- Slow learning
- Apparent stagnation

Too large:

- Oscillation
- Instability
- Divergence

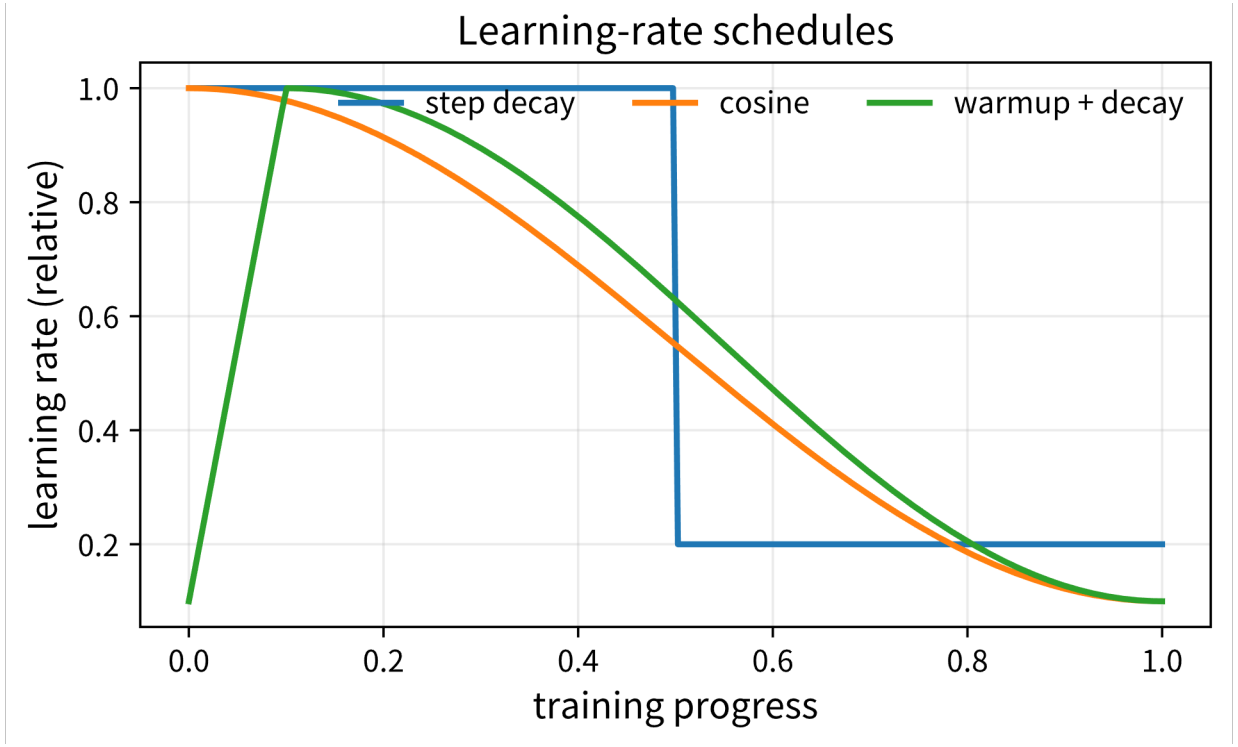


Often more important than many other hyperparameters.

Do Not Keep Learning Rate Fixed Forever

Common schedules:

- Step decay
- Cosine decay
- Warmup + decay



Why decay?

- Large steps early
- Small steps late

Warmup:

- Helps when early training is unstable.

Diagnose the Learning-Rate Problem

Case A:

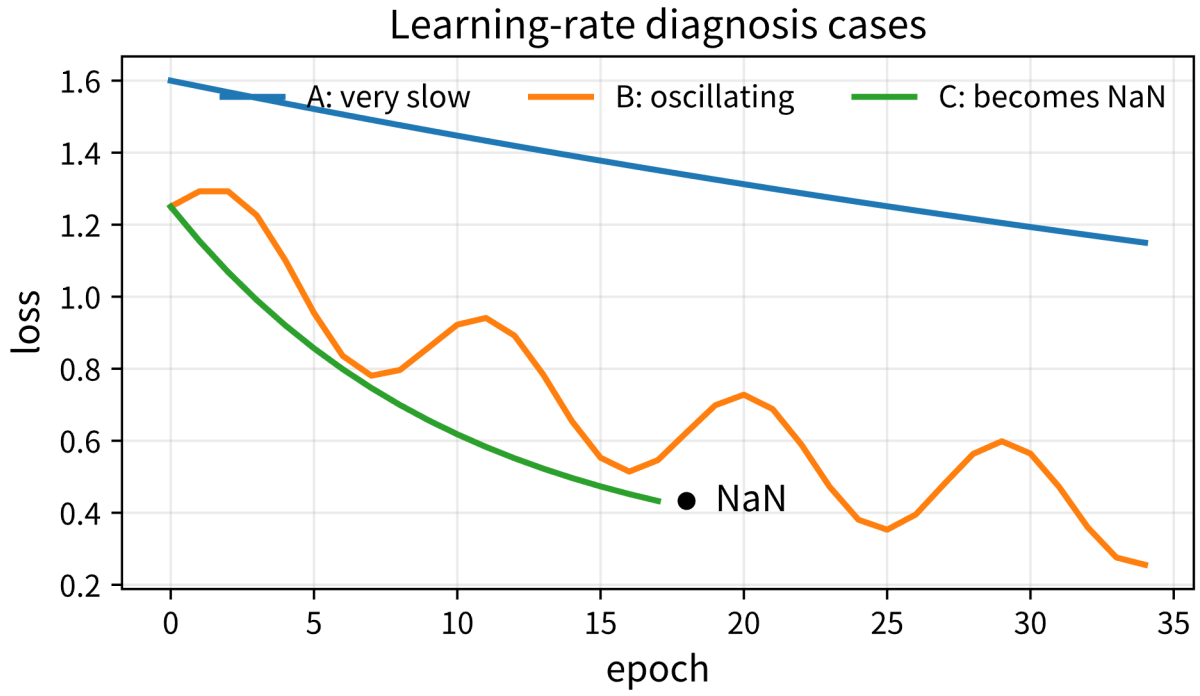
Loss decreases very slowly.

Case B:

Loss oscillates wildly.

Case C:

Loss becomes NaN.



Question:

Which cases suggest LR is too small? Which suggest LR is too large?

4. Regularization

Weight Decay (L2 Regularization)

Penalty term:

$$\mathcal{L}_{total} = \mathcal{L}_{data} + \lambda \|\theta\|_2^2$$

λ: weight decay factor

Effect:

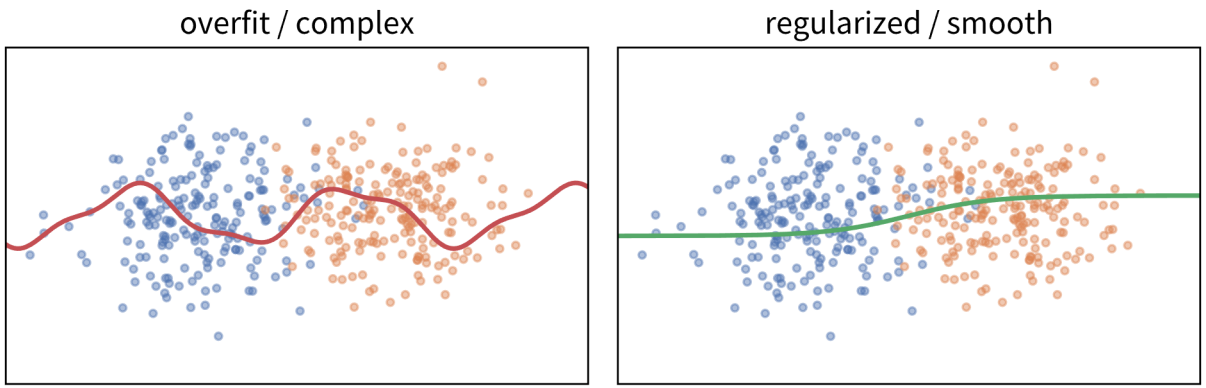
- Discourages large weights
- Smooths the solution
- Often improves generalization

Interpretation:

- Prefer simpler parameter configurations.

Why large weights lead to overfitting?

Weight decay encourages smoother solutions



Dropout

During training:

- Randomly zero some activations.

Effect:

- Reduces co-adaptation
- Adds stochastic regularization
- Encourages robust features

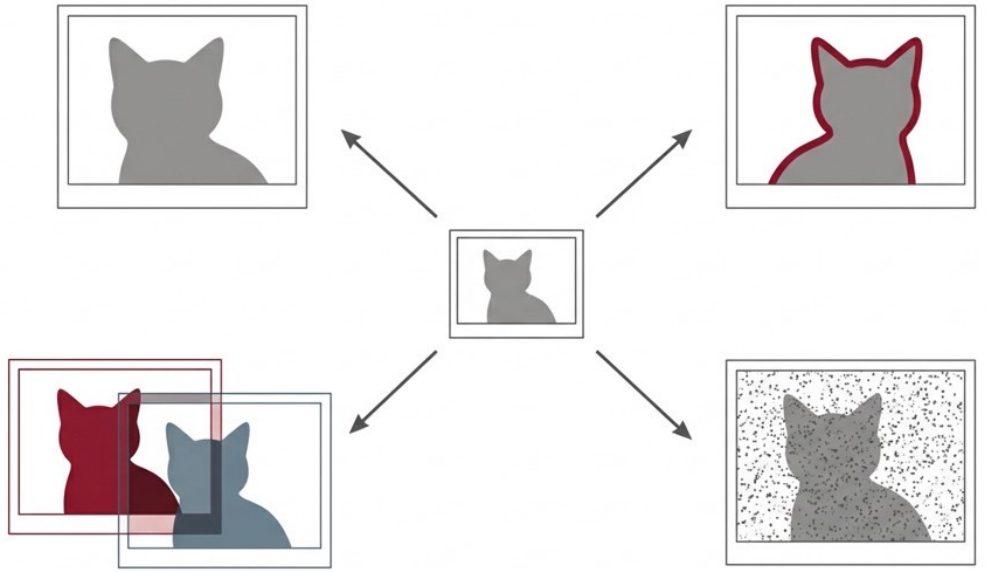
At test time:

- Use full network with appropriate scaling.

The Best Regularizer Is Often More Data

Vision examples:

- Crop
- Flip
- Color jitter
- Noise



General idea:

Augmentation injects invariances and diversity.

Regularization can come from data, not only penalties.

Early Stopping

Concept:

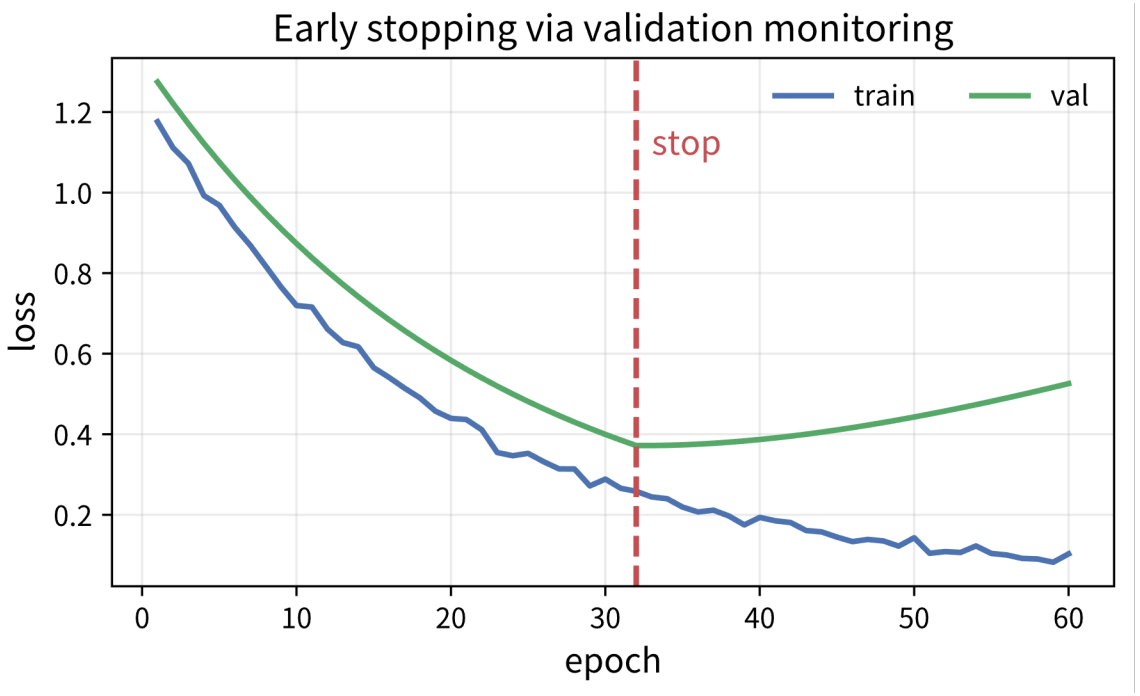
- Monitor validation performance.
- Stop before overfitting becomes severe.

Why it works:

- Training time itself is a regularization knob.

Practical role:

- Simple and effective.



A Standard Deep-Training Recipe

Typical recipe:

- ReLU network
- Kaiming initialization
- Normalization (e.g., BN)
- Adam or SGD+Momentum
- LR schedule
- Weight decay / Dropout
- Validation monitoring

Message:

- Deep learning works through combinations, not one trick.

Extension: Changes in Modern Large Model Training

Typical recipe:

- ~~ReLU network~~ (often use GELU/SiLU with better differentiations)
- Kaiming initialization
- Normalization (e.g., BN)
- ~~Adam or SGD+Momentum~~ (Typically use AdamW)
- LR schedule (LR decay becomes less important with AdamW)
- Weight decay / Dropout (No dropout, large WD for LLMs, yet no weight decay for diffusion models)
- Validation monitoring

Message:

- Deep learning works through combinations, not one trick.

Summary

Three key points:

- Backprop gives gradients, not good training by itself.
- Deep training depends on signal flow, optimizer dynamics, and regularization.
- Stable deep learning comes from coordinated design (init + norm + opt + reg).

Next:

Can architecture reduce the burden on optimization?

Week 7: CNNs and structured inductive bias.

Coursework 3

- Hyperparameter exploration (≥ 3 hyperparameters) with tables and plots.

2.3 Learning Rate

[Test several learning rates spanning a wide range.]

Analysis: *[What happens with too-high learning rates (oscillation, divergence)? What about too-low rates (slow convergence, underfitting)? How does momentum interact with learning rate?]*

Table 3: Effect of learning rate on performance.

Learning Rate	Val Acc	Test Acc	Notes
_____	__._ %	__._ %	_____
_____	__._ %	__._ %	_____
_____	__._ %	__._ %	_____