Definition

Use **all** training data to compute gradient:

$$\nabla \mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\hat{y}^{(i)}, y^{(i)})$$

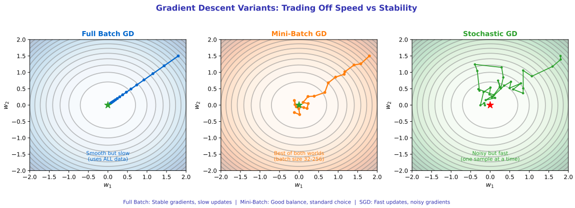
Then update weights once.

Advantages:

+ Stable gradient estimate
+ Deterministic updates
+ Guaranteed descent direction

Disadvantages:

- Slow for large datasets
- Must load all data in memory
- One update per full pass



../.

batch_vs_stochasti

Compute gradient using the entire dataset

Stochastic Gradient Descent (SGD)

Definition
Update after **each** single example:

$$\nabla \mathcal{L} \approx \nabla \ell(\hat{y}^{(i)}, y^{(i)})$$

One sample = one update.

Advantages:

+ Very fast updates
+ Can handle huge datasets
+ Noise helps escape local minima
+ Online learning possible

Disadvantages:

- Noisy gradient estimate
- Erratic convergence
- May not settle at minimum

Why "Stochastic"?

Random sampling of training examples introduces randomness into gradient.

Expected Value:

$$\mathbb{E}[\nabla \ell^{(i)}] = \nabla \mathcal{L}$$

On average, SGD points in the right direction.

Variance:
Individual updates are noisy, but noise can help exploration.

Update after each single example

Mini-Batch: The Sweet Spot

Definition
Use small batches of B examples:

$$\nabla \mathcal{L} \approx \frac{1}{B} \sum_{i=1}^{B} \nabla \ell(\hat{y}^{(i)}, y^{(i)})$$

Typical B: 32, 64, 128, 256

Advantages:

+ Reduced variance vs SGD
+ GPU parallelization
+ Reasonable memory usage
+ Frequent updates

The Modern Default

Batch Size Trade-offs

Size	Noise	Speed
1 (SGD)	High	Fast updates
32-256	Medium	Best practice
Full batch	Low	Slow updates

Large Batch Issues:

- May converge to sharp minima
- Worse generalization
- Need learning rate scaling

Balance between efficiency and noise

Epochs: Full Passes Through Data

Definition
Epoch = one complete pass through all training data.

With Mini-Batches:

- 10,000 samples
- Batch size 100
- 100 updates per epoch

Typical Training:

- 10-1000 epochs
- Monitor loss curve
- Stop when converged

Training Timeline

Stage	Behavior
Early epochs	Loss drops quickly
Middle epochs	Progress slows
Late epochs	Diminishing returns

When to Stop?

- Loss stops improving
- Validation loss increases (overfitting!)
- Resource constraints

Training typically requires multiple epochs

Training Curves

What to Plot

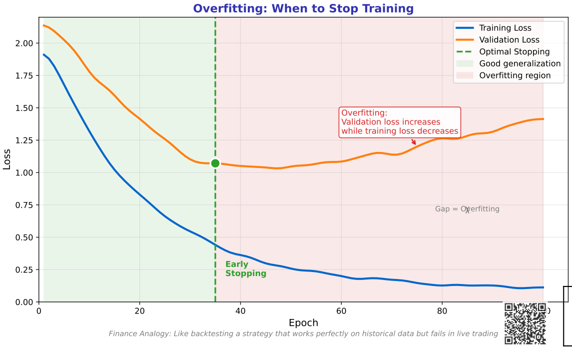
- Training loss vs. epoch
- Validation loss vs. epoch
- Learning rate schedule
- Gradient norms (debugging)

Healthy Training:

- Both losses decrease
- Validation tracks training
- Smooth convergence

Warning Signs:

- Training drops, validation rises
- Loss oscillates wildly
- Loss becomes NaN



../.

overfitting_curve

Monitoring progress during training

Worked Example: One Training Step

Simple 2-2-1 Network

Given:

- Input: $\mathbf{x} = (0.5, 0.8)^T$
- Target: $y = 1$
- Current weights (simplified)

Forward Pass:

$$
\begin{aligned}
z^{(1)} &= W^{(1)}\mathbf{x} + b^{(1)} \\
a^{(1)} &= \sigma(z^{(1)}) \\
z^{(2)} &= W^{(2)}a^{(1)} + b^{(2)} \\
\hat{y} &= \sigma(z^{(2)}) = 0.62
\end{aligned}
$$

Loss and Backward

Loss:

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.62)^2 = 0.072$$

Backward Pass:

$$
\begin{aligned}
\delta^{(2)} &= (0.62 - 1) \cdot 0.62(1 - 0.62) \\
&= -0.089
\end{aligned}
$$

Weight Gradient:

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \delta^{(2)} \cdot a^{(1)}$$

Update:

$$W^{(2)} \leftarrow W^{(2)} - 0.1 \cdot \nabla W^{(2)}$$

Following the numbers through one training step

The Vanishing Gradient Problem

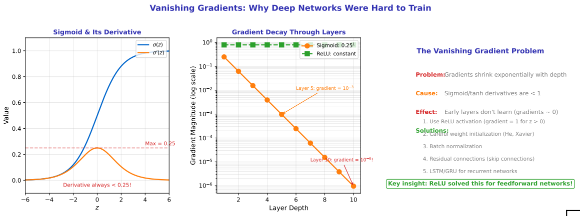
The Problem
Gradients shrink as they flow backward:

$$\delta^{(l)} \propto \prod_{k=l}^{L-1} \sigma'(z^{(k)})$$

For sigmoid: $\sigma'(z) \leq 0.25$
Through 10 layers: gradient $\times 10^{-6}$

Symptoms:

- Early layers don't learn
- Deep networks fail to train
- Loss plateaus quickly



../.

vanishing_gradient_dem

Deep networks: gradients can become vanishingly small

Full Mathematical Derivation

This Module: Intuition
We covered:

- Why backprop works (chain rule)
- How errors flow backward
- Update rule intuition
- Training dynamics

What We Skipped:

- Full mathematical derivation
- Matrix calculus details
- Vectorized implementations
- Automatic differentiation theory

Appendix B Contains:

1. Chain rule setup
2. Output layer error derivation
3. Hidden layer recursion formula
4. Complete gradient equations
5. Weight and bias gradients
6. Algorithm pseudocode

For the mathematically curious:
The appendix provides the rigorous derivation with all matrix calculus steps.

See Appendix B for complete backpropagation derivation

What Is Overfitting?

Definition

Overfitting: When a model learns the training data too well, including its noise, and fails to generalize.

Analogy:

A student who memorizes exam answers but doesn't understand the material.

Symptoms:

- Training loss: very low
- Test loss: high
- Model is "too confident"

Why It Happens

Model Complexity:

- Too many parameters
- Can fit any training data perfectly
- Including noise

Limited Data:

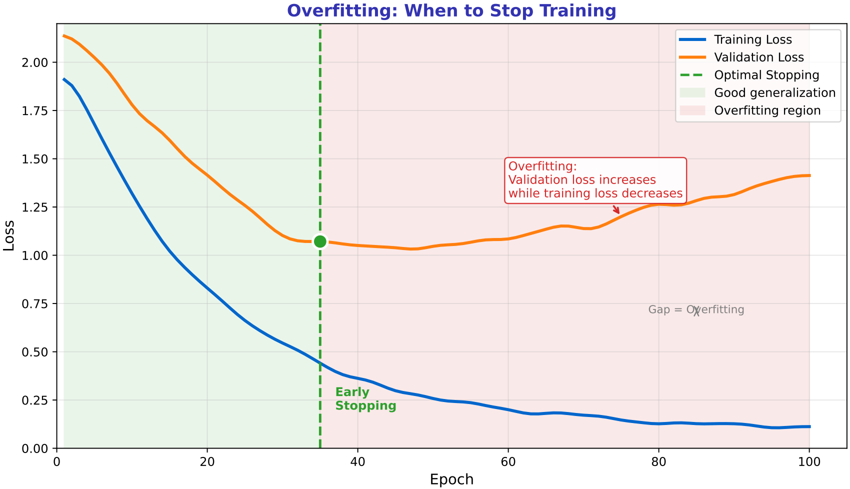
- Not enough examples
- Training set not representative
- Noise gets learned as signal

Training Too Long:

- Model eventually memorizes
- Needs early stopping

When your model memorizes instead of learns

Training vs Validation Loss



Finance Analogy: Like backtesting a strategy that works perfectly on historical data but fails in live trading

../.

overfitting_curve

Training loss decreases but validation increases

Finance: The Backtest Trap

The Trap
Every trading strategy looks good on historical data – that's how you found it!

The Process:

1. Try many strategies
2. Keep the one that worked best
3. By construction, it fits the past
4. Future performance? Unknown.

Multiple Testing:

- Try 1000 random strategies
- Best one has Sharpe 2.0
- Is it skill or luck?

Why Finance Overfits Easily

1. **Limited Data**
   - 20 years = 5000 trading days
   - Few independent observations
2. **Low Signal-to-Noise**
   - Markets are noisy
   - Easy to fit noise
3. **Non-Stationarity**
   - Regimes change
   - Past may not predict future
4. **Look-Ahead Bias**
   - Using future information
   - Subtle but deadly

"Every strategy looks good on historical data"

Why Finance Overfits So Easily

Data Limitations

Domain	Samples
ImageNet	1,200,000
MNIST	60,000
Stock returns (daily, 10y)	2,520
Stock returns (monthly, 50y)	600
Market crashes	$\sim$10

The Problem:
Neural networks have thousands of parameters but only thousands of data points.

Signal vs Noise

Image Classification:

- A cat is always a cat
- Signal is strong and consistent
- R^2 can reach 99%+

Stock Prediction:

- Returns are mostly random
- Signal is weak and changing
- R^2 of 1% is excellent!

Implication:
Standard ML practices don't directly transfer to finance.

Limited data, high noise, non-stationary markets

Detecting Overfitting

Train/Validation/Test Split

1. **Training Set** (60-80%)
   - Used to fit weights
2. **Validation Set** (10-20%)
   - Used to tune hyperparameters
   - Monitor for overfitting
3. **Test Set** (10-20%)
   - Final evaluation only
   - Touch only once!

Key Rule:
Never use test data for decisions.

Warning Signs

Overfitting Indicators:

- Training loss $\ll$ validation loss
- Validation loss starts increasing
- Model predictions are "too confident"
- Performance degrades out-of-sample

For Finance:

- Backtest Sharpe $\gg$ live Sharpe
- Strategy "stops working"
- Drawdowns worse than expected

Always monitor out-of-sample performance

Discussion Question

"How would you know if your stock prediction model is overfitting? What specific symptoms would you look for?"

Consider:

In Training:

- Training/validation gap
- Validation loss trend
- Prediction confidence

In Production:

- Live vs. backtest performance
- Regime sensitivity
- Transaction cost impact

Best Practice: Always maintain a truly out-of-sample test set that you evaluate only once.

Think-Pair-Share: 3 minutes

Preview: Fighting Overfitting

Solutions (Module 4)

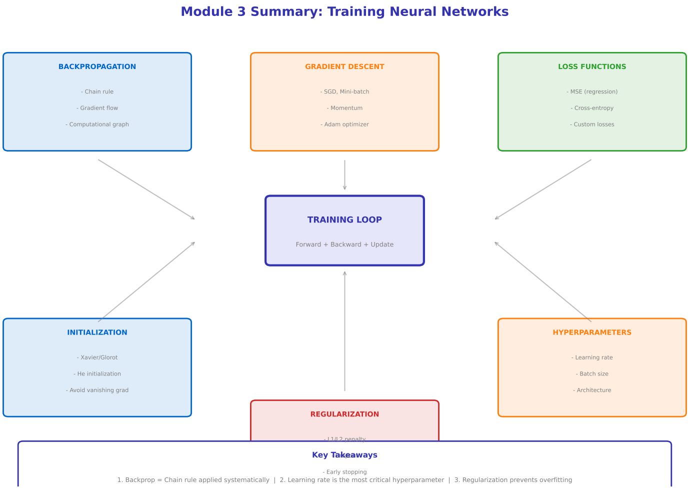
1. **L1/L2 Regularization**
   - Penalize large weights
   - Simpler models
2. **Dropout**
   - Randomly disable neurons
   - Ensemble effect
3. **Early Stopping**
   - Stop before overfitting
   - Use validation loss
4. **Data Augmentation**
   - Create more training data
   - Finance: bootstrap?

Finance-Specific

1. **Walk-Forward Validation**
   - Respect time ordering
   - Rolling windows
2. **Cross-Validation Variants**
   - Purged CV
   - Combinatorial CV
3. **Ensemble Methods**
   - Average multiple models
   - Reduce variance

Module 4 will cover these in detail.

Module 4 will cover solutions: regularization, dropout, early stopping



../.

module3_summary_diagra

The complete neural network training process

Module 3: Key Takeaways

What We Learned

1. **Loss Functions**
   - Measure prediction error
   - MSE, cross-entropy
   - Define what "good" means
2. **Gradient Descent**
   - Follow the slope downhill
   - Learning rate matters
   - Batch vs stochastic
3. **Backpropagation**
   - Chain rule for credit assignment
   - Error flows backward
   - Enables efficient gradient computation
4. **Training Dynamics**
   - Epochs and batches
   - Monitoring with curves
   - Vanishing gradients
5. **Overfitting**
   - Memorizing vs learning
   - Train/val/test split
   - Finance-specific challenges

The Big Picture:

We can now train neural networks. But making them work well requires more...

From measuring error to updating weights

What We've Built So Far

Modules 1-3 Foundation

1. **Module 1: Architecture**
   - Perceptron basics
   - Linear decision boundaries
   - Limitations (XOR)
2. **Module 2: MLPs**
   - Hidden layers
   - Non-linear activation
   - Universal approximation
3. **Module 3: Training**
   - Gradient descent
   - Backpropagation
   - Overfitting awareness

You Can Now:

- Explain how neural networks compute
- Understand the training process
- Recognize overfitting
- Follow the math (or know where to look)

What's Missing:

- Practical regularization
- Real-world applications
- Finance case studies
- Modern developments

Modules 1-3: The complete neural network foundation

Key Questions for Reflection

Think about these as you move to Module 4:

1. **Loss vs. Profit:**
   Why might minimizing MSE not maximize trading profit? What loss function would better align with trading goals?
2. **Overfitting in Finance:**
   With only 20 years of daily data, how many parameters can we safely learn? What's the ratio of samples to parameters you'd be comfortable with?
3. **Non-Stationarity:**
   If market regimes change, what does that mean for our training strategy? Should we weight recent data more heavily?
4. **The Efficient Market Hypothesis:**
   If markets are efficient, can neural networks find persistent patterns? What would success look like?

Reflect on the learning process

Preview: Module 4

"Theory meets practice. How do we actually use neural networks in finance?"

Coming Up:

- Regularization techniques
  - L1/L2, dropout, early stopping
- Financial data challenges
  - Non-stationarity, noise
- Complete case study
  - Stock prediction end-to-end

Mathematical details: See Appendix B-D for derivations

Also:

- Modern architectures overview
  - CNN, RNN, Transformers
- Limitations and ethics
  - Black-box decisions
  - Regulatory concerns
- Future directions
  - Where the field is heading

Next: Regularization, case studies, and modern developments

Training Dynamics and Regularization

Neural Networks for Finance

Neural Networks for Finance

BSc Lecture Series

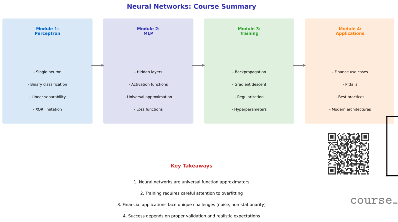
November 30, 2025

The Journey So Far

What We've Covered:

- **Module 1:** The Perceptron
  - Single neuron, decision boundaries
  - XOR limitation → AI Winter
- **Module 2:** Multi-Layer Perceptrons
  - Hidden layers, activation functions
  - Universal Approximation Theorem
- **Module 3:** Training
  - Gradient descent, backpropagation
  - Overfitting warning signs

The Foundation is Complete



../.

course_summar

Perceptron → MLP → Training: The complete foundation

The Final Question

"How do we actually use this for stock prediction?"

From theory to practice:

- How do we prevent overfitting in finance?
- What makes financial data different?
- Does this actually work?
- What are the ethical considerations?

Theory meets practice

Module 4 Roadmap

1. **Historical Context** (2012-Present)
   - The deep learning revolution
2. **Regularization Techniques**
   - L1/L2, dropout, early stopping
3. **Financial Data Challenges**
   - Non-stationarity, regime changes, biases
4. **Case Study: Stock Prediction**
   - S&P 500 direction prediction (realistic assessment)
5. **Modern Architectures**
   - CNN, RNN, Transformer overview
6. **Limitations and Ethics**
   - What neural networks can and cannot do

From theory to real-world applications

The Reality Check

Theory is Clean:

- Data is stationary
- Training set represents test set
- Patterns persist
- No transaction costs
- Unlimited computing power

Finance is Messy:

- Markets change constantly
- Past may not predict future
- Regime changes happen
- Costs eat into profits
- Latency matters

Warning: Paper profits $\neq$ Real profits

"Theory is clean. Finance is messy."

Recall from Module 3:

- Model learns training data too well
- Memorizes noise instead of patterns
- Fails on new, unseen data

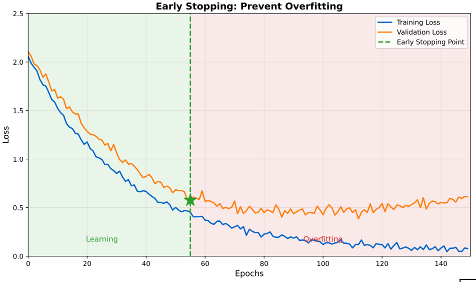
In Finance, This Is Critical:

- Backtest shows 40% annual returns
- Live trading shows -15%
- This happens constantly

Why Module 4 Focuses on This:

- Overfitting is the #1 failure mode
- Financial data is especially prone
- Must master regularization techniques

The Overfitting Gap:



../.

early_stoppin

Overfitting: The greatest challenge in financial ML

Why Finance Overfits So Easily

Limited Data:

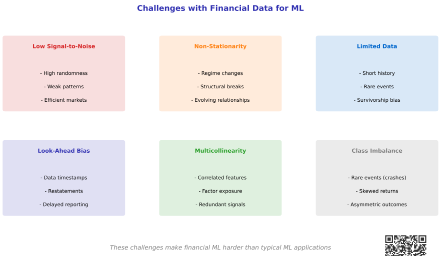
- 20 years of daily data = 5,000 samples
- Compare to ImageNet: 14,000,000 images
- Regime changes reduce effective samples further

High-Dimensional Features:

- 50 technical indicators × 10 lookbacks = 500 features
- More parameters than data points = guaranteed overfitting

Low Signal-to-Noise:

- Daily stock returns: 95%+ noise
- Real patterns are tiny



../.

financial_data_challenge

Limited data, high noise, changing regimes

The Idea: Add penalty for large weights

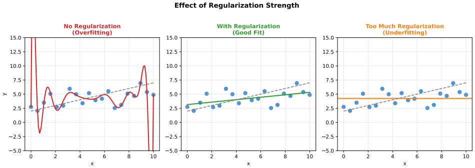
$$\mathcal{L}_{reg} = \mathcal{L} + \frac{\lambda}{2}\|\mathbf{W}\|_2^2 = \mathcal{L} + \frac{\lambda}{2}\sum_i w_i^2$$

Effect on Optimization:

- Original gradient: $\nabla_w \mathcal{L}$
- With L2: $\nabla_w \mathcal{L} + \lambda w$
- Weights decay toward zero each update
- Also called "weight decay"

Hyperparameter λ:

- $\lambda = 0$: No regularization
- λ large: All weights $\to 0$
- Typical: 10^{-4} to 10^{-2}



../.

regularization_effec

Push weights to be small

L2 Intuition

Why Does Penalizing Large Weights Help?

Mathematical View:

- Large weights $\rightarrow$ extreme predictions
- Small changes in input $\rightarrow$ big output changes
- High sensitivity $=$ memorization
- L2 forces smoother functions

Bayesian View:

- L2 $=$ Gaussian prior on weights
- Prior belief: weights should be small
- More data $\rightarrow$ prior matters less

Finance Analogy:

- Large weight on one feature $=$ "betting everything on one stock"
- Risky: what if that feature stops working?
- L2 forces diversification across features
- No single feature dominates the prediction

Key Insight:

- L2 doesn't eliminate features
- Just reduces their influence
- All features contribute, but moderately

Don't let any single feature dominate

The Idea: Penalty proportional to absolute value

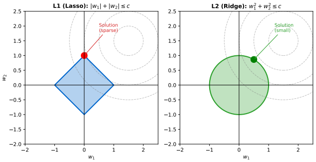
$$\mathcal{L}_{reg} = \mathcal{L} + \lambda \|\mathbf{W}\|_1 = \mathcal{L} + \lambda \sum_i |w_i|$$

Key Difference from L2:

- L1 pushes weights to **exactly zero**
- Creates sparse models (feature selection)
- Automatically identifies irrelevant features

Why Sparsity?

- L1 gradient is $\pm\lambda$ (constant)
- Small weights get pushed to zero
- L2 gradient is λw (proportional)
- Small weights shrink slowly, never reach zero



L1 vs L2 Comparison

Property	L1 (Lasso)	L2 (Ridge)
Constraint	Diamond	Circle
Sparsity	Yes (zeros)	No
Feature Selection	Automatic	No
Correlated Features	Picks one	Shrinks all
Computational	Harder	Easier

../.

l1_vs_l

Push some weights to exactly zero: feature selection

L1 vs L2: Comparison

Property	**L1 (Lasso)**	**L2 (Ridge)**
Penalty term	$\lambda \sum \lvert w_i \rvert$	$\frac{\lambda}{2} \sum w_i^2$
Effect on weights	Some become exactly 0	All shrink toward 0
Feature selection	Yes (automatic)	No
Correlated features	Picks one arbitrarily	Shares weight among them
Sparsity	Sparse solutions	Dense solutions
Computation	Non-differentiable at 0	Smooth, differentiable
Use when	Few features matter	All features may matter

Elastic Net: Combine both: $\lambda_1 \|W\|_1 + \lambda_2 \|W\|_2^2$
Best of both worlds for correlated features

L1 for sparsity, L2 for shrinkage

Dropout: Random Deactivation

The Idea (Hinton et al., 2012):

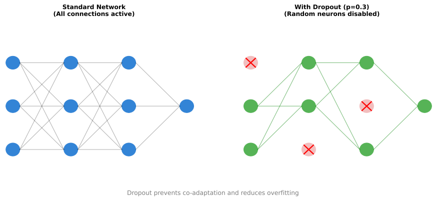
- During training: randomly “drop” neurons
- Each neuron has probability p of being set to 0
- Typically $p = 0.5$ for hidden, $p = 0.2$ for input

Training:

- Each mini-batch sees different network
- Forces redundancy in learned features
- No neuron can become a “crutch”

Inference:

- Use all neurons (no dropout)
- Scale outputs by $(1 - p)$ or use “inverted dropout”



../.

dropout_visualizatio

“No single neuron becomes a crutch”

Discussion Question

"How is dropout like diversifying a portfolio?"

- What happens if you bet everything on one stock?
- What happens if a neural network relies on one neuron?
- How does diversification protect against failure?
- How does dropout force the network to diversify?

Think-Pair-Share: 3 minutes

Dropout Intuition: Ensemble Learning

Ensemble Interpretation:

- Network with n neurons has 2^n possible subnetworks
- Dropout trains all subnetworks simultaneously
- Each mini-batch samples a different subnetwork
- Final prediction: average of all subnetworks

Why Ensembles Work:

- Different models make different errors
- Averaging reduces variance
- More robust to noise

Finance Parallel:

- One analyst: high variance predictions
- Committee of analysts: more stable
- Dropout = "committee of networks"

Practical Notes:

- Dropout slows convergence
- Needs more epochs to train
- Don't use with batch normalization (debate)
- Less common in CNNs today

Dropout approximates training an ensemble of networks

The Simplest Regularization:

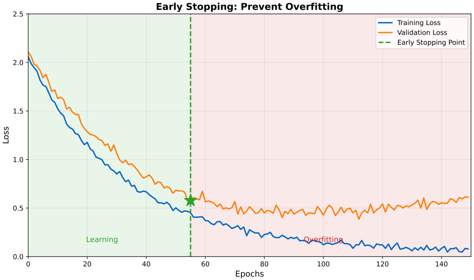
- Monitor validation loss during training
- Stop when validation loss stops improving
- Use the model from the best epoch

Implementation:

- Track best validation loss
- Patience: wait k epochs before stopping
- Save checkpoint at each improvement
- Restore best checkpoint at end

Why It Works:

- Early epochs: learning real patterns
- Later epochs: memorizing training noise
- Sweet spot: generalization peak



Typical patience: 5-20 epochs

../.

early_stoppin

Stop training when validation loss stops improving

Walk-Forward Validation for Time Series

Standard Cross-Validation: WRONG for Time Series

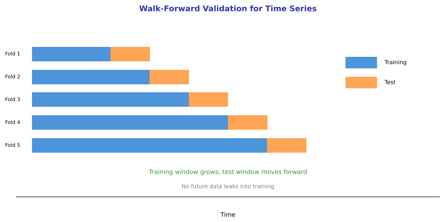
- Random splits leak future information
- Model sees 2024 data, predicts 2023
- Guaranteed overfitting

Walk-Forward Validation:

- Train on [2010-2015], validate on [2016]
- Train on [2010-2016], validate on [2017]
- Train on [2010-2017], validate on [2018]
- Always: train on past, validate on future

Anchored vs Rolling Window:

- Anchored: always start from same date
- Rolling: fixed window slides forward



../.

walk_forward_validatio

Train on past, validate on future (never the reverse)

Fighting Overfitting: Summary

Technique	Mechanism	When to Use
L2 (Ridge)	Penalize large weights	Always (as baseline)
L1 (Lasso)	Push weights to zero	Feature selection needed
Dropout	Random neuron deactivation	Deep networks
Early Stopping	Stop before overfitting	Always (free)
Walk-Forward	Time-respecting validation	Time series only

Practical Recommendation for Finance:

1. Always use walk-forward validation
2. Start with L2 regularization
3. Add early stopping (patience=10)
4. Try dropout (0.2-0.5) for deep networks
5. Use L1 if you need interpretable feature importance

Multiple defenses against overfitting