Module 4: From Theory to Practice

Neural Networks in Finance and Modern Developments (2012-Present)

Neural Networks for Finance

BSc Lecture Series

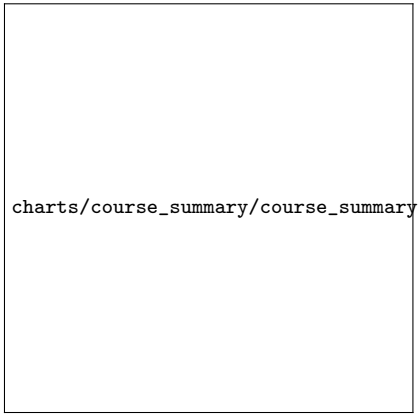
November 26, 2025

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



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."

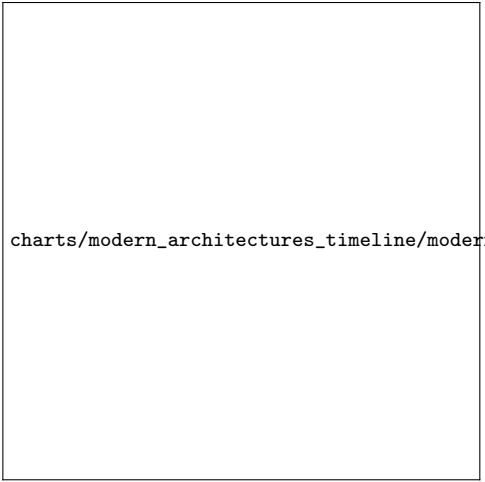
2012: The Deep Learning Revolution

AlexNet (Krizhevsky et al., 2012):

- ImageNet competition: 1.2M images, 1000 classes
- **Error rate: 15.3%** (vs. 26.2% second place)
- Deep convolutional neural network (8 layers)

Why This Mattered:

- 10+ percentage points better than alternatives
- Proved deep learning works at scale
- GPU training (2x NVIDIA GTX 580)
- Started the deep learning "gold rush"



AlexNet: When deep learning proved its superiority

What Made Deep Learning Work?

Three Factors Converged in the 2010s:

1. Big Data
- ImageNet: 14M+ images
- Internet scale data
- Labeled datasets
- In finance: tick data, alternative data

2. GPU Computing
- Parallel matrix operations
- 100x speedup vs CPU
- CUDA programming
- Cloud GPU access

3. Better Algorithms
- ReLU activation
- Dropout regularization
- Batch normalization
- Better optimizers (Adam)

All three were necessary; none was sufficient alone

The convergence of data, compute, and algorithms

2017: Attention Is All You Need

The Transformer Architecture (Vaswani et al., 2017):

- Originally for machine translation
- Key innovation: **Self-attention mechanism**
- No recurrence needed $\rightarrow$ parallelizable

Self-Attention Intuition:

- Each position "attends" to all other positions
- Learns which inputs are relevant to each other
- "The cat sat on the mat because *it* was tired"
- Attention reveals that "it" refers to "cat"

Attention Formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Q: Query (what am I looking for?)
- K: Key (what do I have?)
- V: Value (what do I return?)

Vaswani et al.: The architecture that changed everything

2020+: The GPT Era

Scaling Laws and Foundation Models:

Key Developments:

- GPT-2 (2019): 1.5B parameters
- GPT-3 (2020): 175B parameters
- GPT-4 (2023): rumored 1T+ parameters
- ChatGPT: Conversational interface

Scaling Discovery:

- Performance scales predictably with:
  - Model size
  - Dataset size
  - Compute budget

Impact on Finance:

- Sentiment analysis from news/social media
- Document understanding (10-K filings)
- Natural language queries for data
- Automated research summarization

But: LLMs don't predict stock prices

- Different problem domain
- Time series $\neq$ language patterns

From GPT-2 to GPT-4 and beyond

Discussion Question

"Why did neural networks succeed in 2012 but not in 1990?

What changed?"

- Was it just computing power?
- What role did data play?
- Were the algorithms fundamentally different?
- Could we have predicted this breakthrough?

Think-Pair-Share: 3 minutes

AI in Finance Today

Major Players:

- **Renaissance Technologies**
  - Medallion Fund: 66% avg. return (1988-2018)
  - Highly secretive, physics/math PhDs
- **Two Sigma**
  - $60B+ AUM
  - Heavy ML/AI focus
- **Citadel**
  - Market making + hedge fund
  - ML for high-frequency trading

Common Applications: Signal generation, portfolio optimization, risk management, alternative data analysis

Renaissance, Two Sigma, Citadel: Industry adoption

The Current Landscape

What's Actually Working:

- Risk management and fraud detection
- High-frequency market making
- Alternative data processing
- Portfolio optimization
- Credit scoring
- Sentiment analysis

What's Mostly Hype:

- "AI that beats the market consistently"
- Perfect stock price prediction
- Fully automated trading for retail
- "Guaranteed returns" from AI

Red Flag: If someone claims their AI consistently beats the market, ask why they're selling it instead of using it.

Separating reality from marketing

The Overfitting Problem Revisited

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:

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

Limited data, high noise, changing regimes

L2 Regularization (Ridge)

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}

charts/regularization_effect/regularization_e

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

L1 Regularization (Lasso)

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

charts/l1_vs_l2/l1_vs_l2.pdf

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):

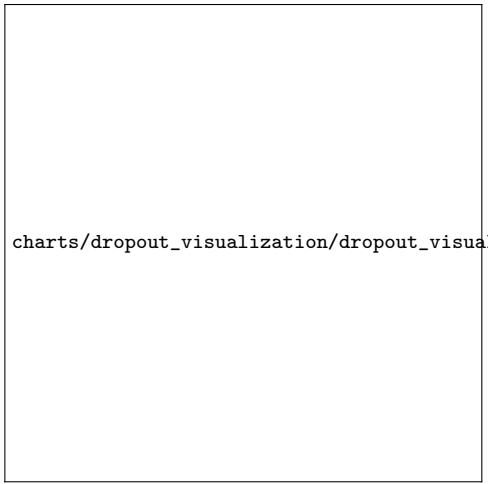
- During training: randomly "drop" neurons
- Each neuron has probability p of being set to 0
- Typically $p = 0.5$ for hidden, $p = 0.2$ for input

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"



"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

Early Stopping

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

Stop training when validation loss stops improving

Walk-Forward Validation for Time Series

Standard Cross-Validation: WRONG for Time Series

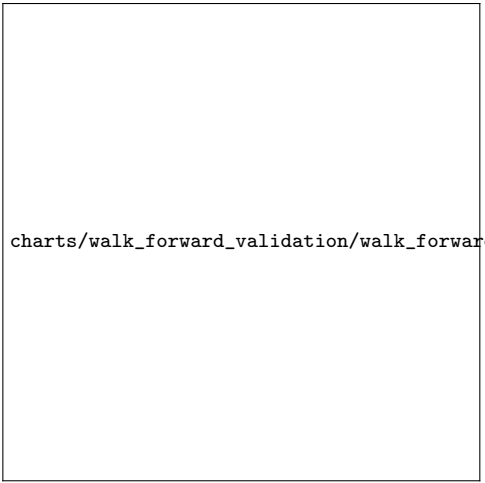
- Random splits leak future information
- Model sees 2024 data, predicts 2023
- Guaranteed overfitting

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



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

The Nature of Financial Data

Financial Data is Fundamentally Different:

Images/Text:

- Patterns are stable over time
- Cat in 2020 looks like cat in 2010
- English grammar doesn't change daily
- High signal-to-noise ratio
- Abundant labeled data

Financial Markets:

- Patterns change constantly
- Strategies that work get arbitraged away
- Regime changes (bull/bear/crisis)
- Extremely low signal-to-noise
- Limited history, no "labels" for future

Key Insight: Success in image recognition doesn't translate to finance.
The problems are fundamentally different.

Financial data is fundamentally different from images or text

Non-Stationarity

Definition:

- Statistical properties change over time
- Mean, variance, correlations all shift
- Model trained on past may fail on future

Causes in Finance:

- Central bank policy changes
- Market structure evolution (HFT, ETFs)
- Regulatory changes
- Technology disruption
- Global events (pandemics, wars)

Implication:

- Models have "shelf life"
- Need regular retraining
- "What worked" $\neq$ "what will work"

charts/regime_changes/regime_changes.pdf

The patterns that worked yesterday may not work tomorrow

Regime Changes

Markets Switch Between Fundamentally Different Behaviors:

Bull Market:

- Upward trend
- Low volatility
- Mean reversion works
- Risk-on behavior
- Correlations low

Bear Market:

- Downward trend
- High volatility
- Momentum works
- Risk-off behavior
- Correlations spike

Crisis:

- Extreme moves
- "All correlations go to 1"
- Historical patterns break
- Liquidity disappears
- Fat tails dominate

Challenge: You don't know which regime you're in until it's over.
Solution: Train separate models or use regime detection.

Markets switch between fundamentally different behaviors

Noise vs Signal

Signal-to-Noise Ratio (SNR):

- Daily stock returns: SNR $\approx$ 0.05
- Speech recognition: SNR $\approx$ 10-20
- 200-400x harder!

What This Means:

- 95%+ of price movement is random
- True patterns are tiny
- Easy to find spurious patterns
- Need massive data to detect signal

Example:

- Average daily return: 0.04%
- Daily standard deviation: 1%
- Signal = return / std = 0.04

Implications for ML:

- Models will find patterns in noise
- Backtests look amazing
- Live performance disappoints
- Need extreme skepticism

Reality Check:

- If returns were 50% predictable, you'd be a billionaire in months
- Markets are efficient enough that small edges are huge
- 55% accuracy is actually impressive

Most price movement is noise, not signal

Look-Ahead Bias

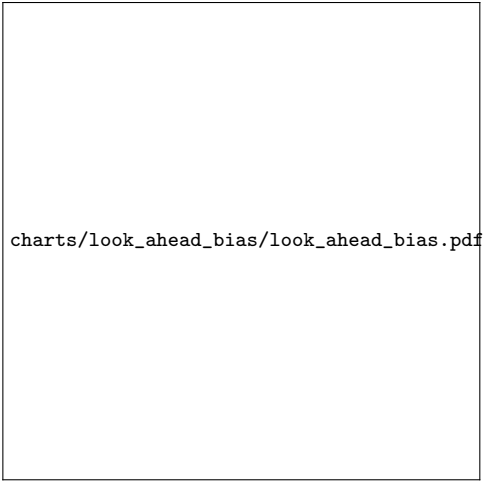
Definition: Using information that wasn't available at decision time.

Common Mistakes:

- Using today's adjusted close to trade at today's open
- Normalizing with full dataset statistics
- Including stocks that didn't exist yet
- Using restated (revised) financial data
- Feature engineering with future data

Example:

- Train on 2020-2023
- Normalize: subtract mean, divide by std
- Problem: Mean includes 2023!
- In 2020, you didn't know 2023 stats



Prevention:

Survivorship Bias

Definition: Only successful companies remain in the dataset.

The Problem:

- S&P 500 today has survivors
- Enron, Lehman, Bear Stearns are gone
- Your model never sees failures
- Learns patterns of survivors only

Impact:

- Overstates historical returns
- "Average stock returned 10%/year"
- Actually includes only winners

Real Example:

- Study: "Value stocks beat growth"
- Used current value stocks list
- Many value stocks went bankrupt
- True effect was much smaller

Solution:

- Point-in-time constituent lists
- Include delisted companies
- Use survivorship-bias-free databases
- Account for delisting returns

Your dataset doesn't include the failures

Discussion Question

"What makes financial prediction harder than image recognition?"

- A cat is always a cat. Is a bull market always a bull market?
- ImageNet has 14 million labeled images. How many "market crashes" exist?
- If everyone uses the same model, what happens?
- Does finding patterns in finance make them disappear?

Think-Pair-Share: 3 minutes

Data Preprocessing for Finance

Essential Preprocessing Steps:

Normalization:

- Z-score: $\frac{x-\mu}{\sigma}$
- Use rolling window (e.g., 252 days)
- Never use future data!

Missing Data:

- Forward fill (most common)
- Linear interpolation
- Drop if too many missing
- Never: backward fill

Outlier Handling:

- Winsorize at 1%/99% percentile
- Or use robust statistics (median)
- Don't remove outliers blindly!
- Crashes are real data

Feature Engineering:

- Returns not prices (stationarity)
- Log returns for mathematical convenience
- Technical indicators as features
- Lag features appropriately

Proper preprocessing is essential

Case Study: S&P 500 Direction Prediction

A Realistic Example from Start to Finish

Goal:

- Predict S&P 500 next-day direction
- Binary: Up or Down?
- Use only information available at market close

Why This Problem:

- Simple, well-defined target
- Abundant data
- Common industry problem
- Illustrates key challenges

Our Approach:

1. Define features and target
2. Choose architecture
3. Set up walk-forward validation
4. Train and evaluate
5. Reality check the results

Spoiler: Results will be modest.
That's the honest truth about financial ML.

A realistic example from start to finish

Problem Definition

Target Variable:

$$y_t = \begin{cases} 1 & \text{if } R_{t+1} > 0 \\ 0 & \text{if } R_{t+1} \leq 0 \end{cases}$$

where $R_{t+1} = \frac{P_{t+1} - P_t}{P_t}$ is next-day return.

Baseline:

- Random guess: 50% accuracy
- Actual: S&P 500 up 53% of days (long-term)
- "Always predict up": 53% accuracy

Goal:

- Beat 53% consistently
- Out-of-sample (not just backtest)
- After transaction costs

Data:

- Period: 2000-2023 (24 years)
- Frequency: Daily
- Samples: $\sim$6,000 trading days

Important Notes:

- This is harder than it sounds
- Small edge = big money
- Markets are highly efficient
- Most published research overfits

Binary classification: Up or Down?

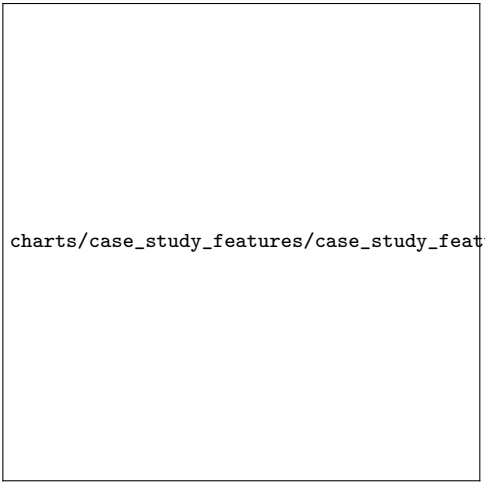
Input Features

Technical Indicators (15 features):

- Returns: 1-day, 5-day, 20-day
- Moving averages: 10/50/200-day ratios
- Volatility: 20-day rolling std
- RSI (14-day), MACD
- Bollinger Band position
- Volume ratio (vs 20-day avg)

Market Factors (5 features):

- VIX level and change
- Treasury yield (10Y)
- Credit spread
- Put/Call ratio



Total: 20 input features

Architecture Decision

Network: 20-16-8-1

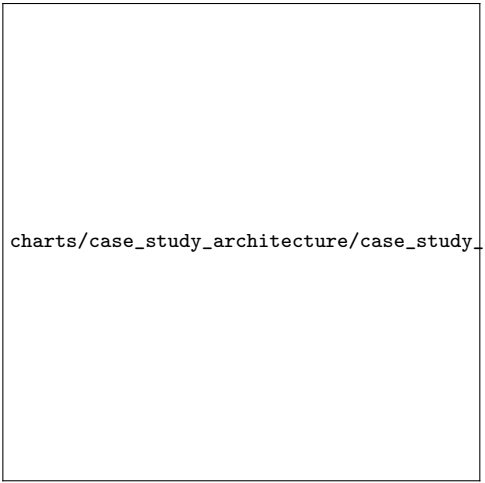
- Input: 20 features
- Hidden 1: 16 neurons (ReLU)
- Hidden 2: 8 neurons (ReLU)
- Output: 1 neuron (Sigmoid)

Why This Architecture?

- Relatively shallow (avoid overfitting)
- Decreasing width (funnel shape)
- Total parameters: $\sim$500
- Parameters $<<$ samples (6,000)

Regularization:

- L2: $\lambda = 0.001$
- Dropout: 0.2 (after each hidden layer)
- Early stopping: patience=10



Balancing model capacity with overfitting risk

Training Setup

Walk-Forward Validation:

Data Split:

- Training: 10 years (2,500 days)
- Validation: 2 years (500 days)
- Test: 2 years (500 days)
- Roll forward by 1 year, retrain

Training Details:

- Optimizer: Adam (lr=0.001)
- Loss: Binary cross-entropy
- Batch size: 64
- Max epochs: 200
- Early stopping: patience=10

Walk-Forward Windows:

Train	Valid	Test
2000-09	2010-11	2012-13
2001-10	2011-12	2013-14
2002-11	2012-13	2014-15
...	...	...
2010-19	2020-21	2022-23

Total: 10 test windows

10 years training, 2 years validation, 2 years test

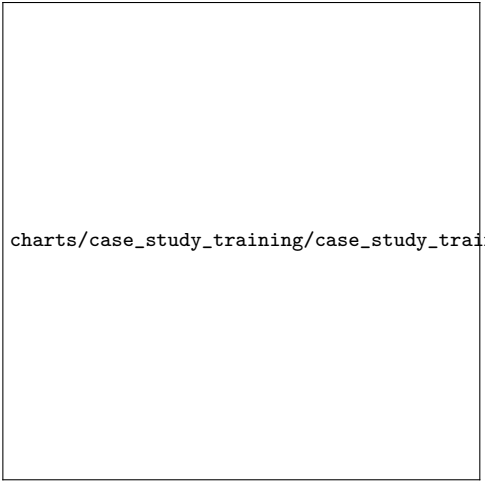
Results: Training Progress

Typical Training Run (2010-2019 → 2022-23):

- Training loss decreases smoothly
- Validation loss: decreases, then flat
- Early stopping at epoch 45-80
- Gap between train/val loss: moderate

Observations:

- Good: Not severe overfitting
- Good: Validation loss improves
- Moderate: Some train-val gap
- Training accuracy: 58-62%
- Validation accuracy: 54-56%



Monitoring the training process

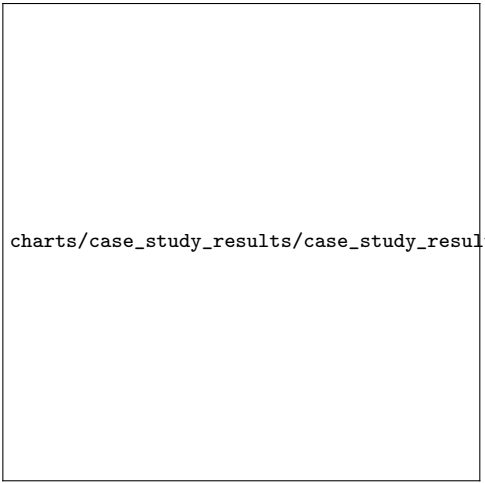
Results: Accuracy

Out-of-Sample Results (2012-2023):

- Average test accuracy: **54.2%**
- Range across windows: 51.8% - 56.7%
- Baseline (always up): 53.1%
- Edge over baseline: +1.1%

By Year:

- Best: 2017 (56.7%) - low volatility
- Worst: 2020 (51.8%) - COVID crash
- Average bull market: 55.1%
- Average bear market: 52.4%



Is 54.2% good? It depends on costs and execution...

54.2% accuracy - is this good?

Discussion Question

"If a model is 54% accurate at predicting direction,

is it profitable?"

- What if each trade costs 0.1% in fees and slippage?
- What if you trade once per day vs once per month?
- Does accuracy equal profitability?
- What other metrics matter?

Think-Pair-Share: 3 minutes

Beyond Accuracy: Risk-Adjusted Returns

Accuracy $\neq$ Profitability

What Accuracy Misses:

- Size of wins vs losses
- 54% accuracy with small wins, large losses = loss
- Timing of predictions
- Risk taken to achieve returns

Better Metrics:

- **Sharpe Ratio**: $\frac{\text{Return} - R_f}{\text{Volatility}}$
- **Max Drawdown**: Largest peak-to-trough loss
- **Win/Loss Ratio**: Avg win / Avg loss

Our Case Study:

- Annual return: 8.2% (vs 9.5% buy-hold)
- Volatility: 12.1% (vs 18.2% buy-hold)
- Sharpe: 0.68 (vs 0.52 buy-hold)
- Max drawdown: 18% (vs 34% buy-hold)

Interpretation:

- Lower return than buy-hold
- But much lower risk
- Better risk-adjusted performance
- Before costs!

Accuracy is not the same as profitability

Reality Check: Transaction Costs

Types of Costs:

- Commission: \$0-10 per trade (retail)
- Bid-ask spread: 0.01%-0.1%
- Market impact: depends on size
- Slippage: execution vs expected price

Our Strategy:

- Trades: 252 days/year (daily)
- Round-trip cost: 0.1% (conservative)
- Annual cost: $252 \times 0.1\% = 25.2\%$
- Gross return: 8.2%
- **Net return: -17%**

charts/transaction_costs/transaction_costs.pd

Lesson: Daily trading requires extremely high accuracy to be profitable.

Costs can eliminate paper profits entirely

The Efficient Market Hypothesis

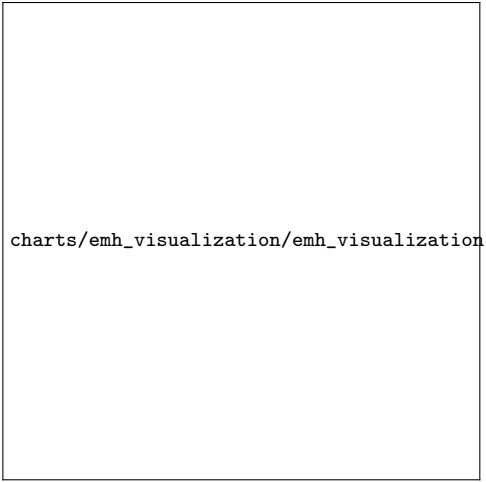
EMH (Fama, 1970):
"Prices fully reflect all available information"

Three Forms:

- **Weak:** Can't profit from past prices
- **Semi-strong:** Can't profit from public info
- **Strong:** Can't profit from any info

Implications for ML:

- If EMH true: all patterns are noise
- If EMH false: patterns exist but are small
- Reality: markets are "mostly efficient"
- Small, temporary inefficiencies exist



Grossman-Stiglitz Paradox:
If markets were perfectly efficient, no one would do

What Works and What Doesn't

What Works (Maybe):

- Risk management and hedging
- Alternative data processing
- High-frequency market making
- Factor model enhancement
- Portfolio optimization
- Regime detection

Where NNs Add Value:

- Complex non-linear relationships
- High-dimensional feature spaces
- Alternative data (satellite, NLP)
- Execution optimization

What Doesn't Work:

- "Predicting stock prices" (directly)
- Black-box trading systems
- Complex models on small data
- Ignoring transaction costs
- Overfitting to backtests

Honest Expectations:

- Small edges are valuable
- 55% accuracy is impressive
- Risk management $>$ alpha generation
- Domain knowledge essential

Setting appropriate expectations for neural networks in finance

Beyond MLPs: Modern Architectures

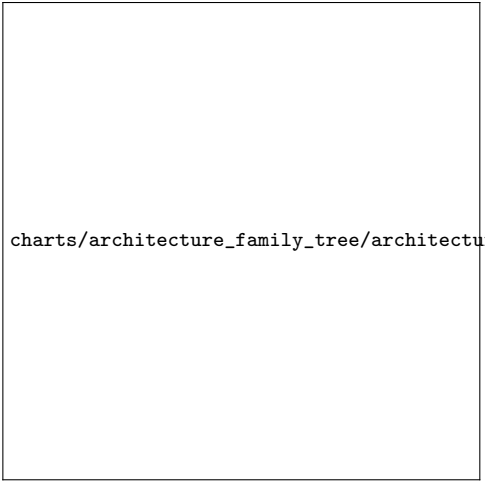
The MLP Foundation:

- Everything we learned applies to modern architectures
- Backpropagation: same algorithm
- Activation functions: same choices
- Regularization: same techniques

Key Modern Architectures:

1. **CNN**: Convolutional Neural Networks
2. **RNN/LSTM**: Recurrent Networks
3. **Transformer**: Attention-based

Common Thread:
All contain feedforward (MLP) components!



MLPs are the foundation for everything that followed

Convolutional Neural Networks

Key Idea: Learnable pattern detectors

- Convolutional filters slide over input
- Detect local patterns (edges, shapes)
- Weight sharing reduces parameters
- Hierarchical feature learning

For Time Series:

- 1D convolutions over time
- Detect patterns in price/volume sequences
- Filter learns what to look for
- E.g., "head and shoulders" pattern

Architecture:

$$\text{Conv1D} \rightarrow \text{ReLU} \rightarrow \text{Pool}$$

$$\rightarrow \text{Conv1D} \rightarrow \text{ReLU} \rightarrow \text{Pool}$$

$$\rightarrow \text{Flatten} \rightarrow \text{MLP} \rightarrow \text{Output}$$

Finance Use Cases:

- Technical pattern recognition
- Order book analysis
- Multi-asset correlation patterns

CNNs: Finding patterns with learnable filters

Recurrent Neural Networks

Key Idea: Memory for sequences

- Process sequences one step at a time
- Maintain hidden state (memory)
- Output depends on current + past inputs
- Natural for time series

RNN Update:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

Problem: Vanishing gradients over long sequences

LSTM Solution (1997):

- Forget gate: what to discard
- Input gate: what to add
- Output gate: what to reveal
- Cell state: long-term memory

Finance Use Cases:

- Time series forecasting
- Sequence-to-sequence (prices)
- Combining with attention

RNNs and LSTMs: Designed for sequences

Transformers and Attention

Key Innovation: Self-attention

- Each position attends to all others
- No recurrence needed
- Parallelizable (fast training)
- Captures long-range dependencies

Components:

- Multi-head attention
- Feedforward layers (MLPs!)
- Layer normalization
- Positional encoding

Attention Formula:

$$\text{Attn} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Finance Use Cases:

- News/sentiment analysis (NLP)
- Document understanding
- Multi-asset attention
- Temporal attention for prices

The architecture behind GPT and modern NLP

The MLP Foundation

Every Modern Architecture Contains MLPs:

CNN:

- Conv layers: shared MLPs
- Final classifier: MLP
- Same activation functions

RNN/LSTM:

- Gate computations: MLPs
- Output layer: MLP
- Same backprop algorithm

Transformer:

- FFN after attention: MLP
- Position-wise: MLP
- 2/3 of parameters in MLPs!

What you learned in this course is the foundation for all of deep learning.

Perceptron $\rightarrow$ MLP $\rightarrow$ CNN/RNN/Transformer

Every modern architecture contains feedforward components

Discussion Question

"If you were building a financial AI startup today,

what architecture and problem would you focus on?"

- Direct price prediction vs risk management?
- Traditional features vs alternative data?
- Simple MLP vs complex Transformer?
- Retail product vs institutional tool?

Think-Pair-Share: 3 minutes

The Black Box Problem

The Interpretability Challenge:

- Neural networks: millions of parameters
- No simple explanation for decisions
- "Why did you sell?" - "Because weight 47,823 was 0.0032"

Why This Matters in Finance:

- Regulatory requirements (explainability)
- Risk management needs understanding
- Client trust requires explanation
- Debugging requires insight

Partial Solutions:

- SHAP values, LIME
- Attention visualization
- Simpler models where possible

Trade-off:

Simple	Complex
Interpretable	Black box
Linear	Non-linear
Stable	May overfit
Lower accuracy	Higher accuracy

Question:
Is 1% more accuracy worth losing all interpretability?

Neural networks are often difficult to interpret

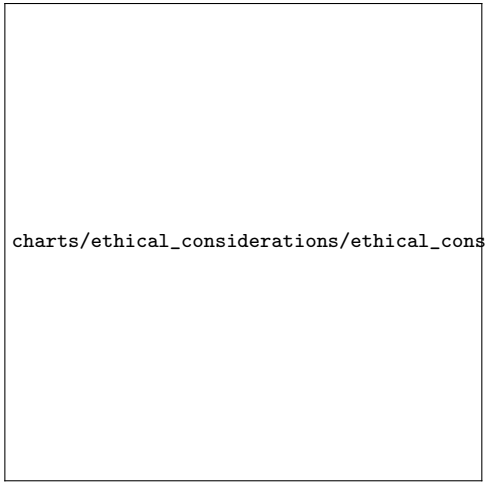
Regulatory Requirements

Key Regulations:

- **MiFID II** (EU): Best execution, transparency
- **GDPR**: Right to explanation for automated decisions
- **SR 11-7** (US): Model risk management
- **Basel III**: Capital requirements, risk models

Explainability Mandates:

- Credit decisions must be explainable
- Trading algorithms need documentation
- Model validation required
- Audit trails essential



Trend: Increasing regulation of algorithmic decision-making

Regulations increasingly demand explainable AI

Systemic Risk from Correlated AI

What if everyone uses similar models?

The Problem:

- Similar training data
- Similar architectures
- Similar features
- $\Rightarrow$ Similar predictions
- $\Rightarrow$ Correlated trades
- $\Rightarrow$ Amplified market moves

Historical Example:

- August 2007: Quant meltdown
- Many funds used similar strategies
- All deleveraged simultaneously
- Massive losses in days

Flash Crash Risk:

- May 6, 2010: Dow dropped 1000 points in minutes
- Algorithmic trading implicated
- Feedback loops between systems

Mitigations:

- Circuit breakers
- Position limits
- Diversity requirements
- Human oversight
- Stress testing for crowding

Correlated AI trading could amplify market instability

Model Risk Management

"All models are wrong, some are useful" - George Box

Types of Model Risk:

- **Specification risk**: Wrong model type
- **Implementation risk**: Coding bugs
- **Data risk**: Bad inputs
- **Usage risk**: Misapplication

Famous Failures:

- LTCM (1998): Model assumptions failed
- Knight Capital (2012): $440M in 45 minutes
- London Whale (2012): VAR model issues

Governance Framework:

- Independent model validation
- Documentation requirements
- Regular backtesting
- Stress testing
- Change management
- Clear ownership

Key Principle:
Never deploy a model you don't understand well enough to know when it might fail.

Responsible deployment requires proper oversight

Historical Lesson: Responsible Innovation

AI Has Seen Hype Cycles Before:

The Pattern:

1. Breakthrough discovery
2. Excessive optimism/funding
3. Over-promising
4. Failure to deliver
5. "AI Winter" backlash
6. Quiet progress
7. Next breakthrough...

Examples:

- 1960s: "Machines will think in 20 years"
- 1980s: Expert systems will replace experts
- 2010s: "Deep learning solves everything"
- Today: "AGI is imminent"

Lesson:
Hype damages the field. Responsible claims and honest assessment help it grow sustainably.

Your Responsibility: Be honest about what neural networks can and cannot do.

Hype cycles damage the field; responsible claims help it grow

Where Neural Networks Add Value

Realistic Assessment of Neural Networks in Finance:

High Value Applications:

- **Risk Management**
  - Fraud detection
  - Credit scoring
  - Anomaly detection
- **Alternative Data**
  - Satellite imagery analysis
  - News sentiment
  - Social media signals
- **Execution**
  - Optimal order routing
  - Market making
  - Transaction cost analysis

Lower Value (Often Overhyped):

- Direct price prediction
- "AI-powered" retail trading apps
- Fully automated strategies
- Complex models on limited data

Key Insight:
Neural networks work best when:

- Abundant data available
- Clear signal exists
- Domain expertise integrated
- Proper validation done

Risk management, alternative data, market making

Neural Networks: 1943-2024

charts/full_timeline_1943_2024/full_timeline_1943_2024.pdf

Course Summary: Key Takeaways

Module 1 - Perceptron:

- Neuron as weighted voting
- Linear separability limits
- XOR problem $\rightarrow$ AI Winter

Module 2 - MLPs:

- Hidden layers solve XOR
- Activation functions enable non-linearity
- Universal Approximation Theorem

Module 3 - Training:

- Gradient descent finds minimum
- Backprop: efficient gradient computation
- Overfitting is the main enemy

Module 4 - Practice:

- Regularization fights overfitting
- Financial data is uniquely challenging
- Honest assessment of capabilities

The Foundation: Everything in modern AI builds on these concepts.

The essential concepts from all four modules

Where to Go from Here

Suggested Learning Path:

Theory:

- Deep Learning (Goodfellow et al.)
- Neural Networks and Deep Learning (Nielsen) - free online
- Stanford CS231n (CNNs)
- Stanford CS224n (NLP)

Practice:

- PyTorch or TensorFlow tutorials
- Kaggle competitions
- Personal projects
- Open-source contributions

Finance-Specific:

- Advances in Financial ML (de Prado)
- Machine Learning for Asset Managers
- QuantConnect, Zipline (backtesting)
- Academic papers (SSRN, arXiv q-fin)

Key Advice:

- Build things!
- Start simple, add complexity
- Focus on fundamentals
- Be skeptical of claims
- Domain knowledge matters

Suggested resources for continued learning

Thank You

Neural Networks for Finance
BSc Lecture Series

Questions?

See Mathematical Appendix for full derivations