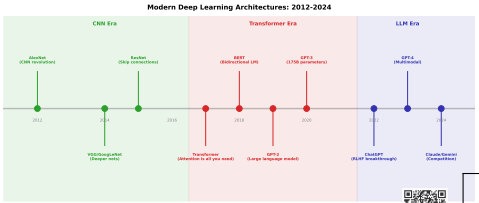
2012: The Deep Learning Revolution

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"



../.

modern_architectures_timelin

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?"

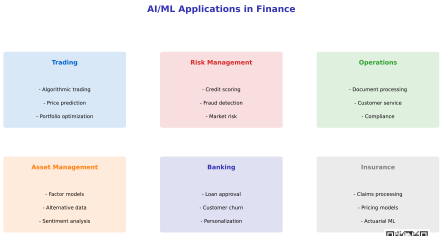
- Was it just computing power?
- What role did data play?
- Were the algorithms fundamentally different?
- Could we have predicted this breakthrough?

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



../../ai_applications_financ

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 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”

../.

regime_change

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

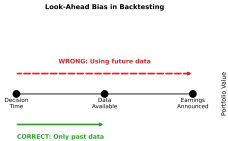
Definition: Using information that wasn't available at decision time.

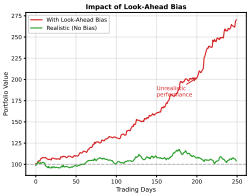
Common Mistakes:

- 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:

- Point-in-time data
- Rolling normalization
- Careful feature engineering

../.

look_ahead_bia

The silent killer of backtests

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

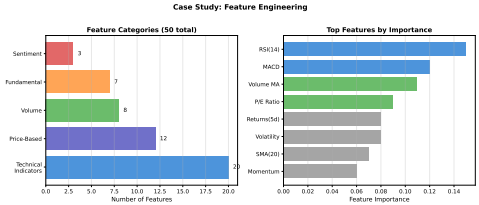
Technical Indicators (15 features):

- 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

Preprocessing: Rolling z-score (252-day window), clip at ± 3



Total: 20 input features

../.

case_study_feature

15 technical indicators + 5 market factors

Architecture Decision

Network: 20-16-8-1

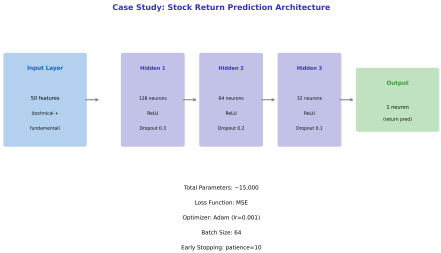
- Input: 20 features
- Hidden 1: 16 neurons (ReLU)
- Hidden 2: 8 neurons (ReLU)
- Output: 1 neuron (Sigmoid)

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



../.

case_study_architectur

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

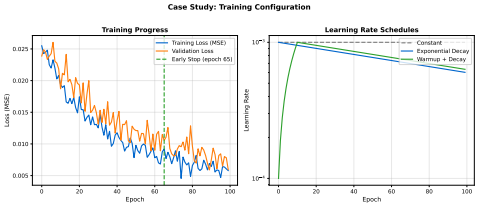
Results: Training Progress

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%



../.

case_study_trainin

Monitoring the training process

Results: Accuracy

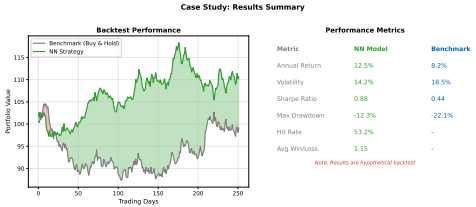
Out-of-Sample Results (2012-2023):

- 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...



../.

case_study_result

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:

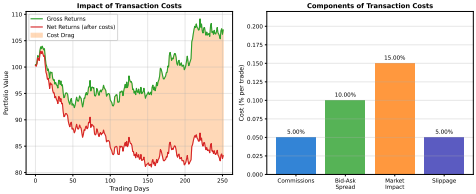
- Lower return than buy-hold
- But much lower risk
- Better risk-adjusted performance
- Before costs!

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%**

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

../.

transaction_cost

Costs can eliminate paper profits entirely

The Efficient Market Hypothesis

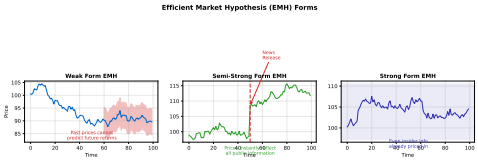
EMH (Fama, 1970):
"Prices fully reflect all available information"

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 research, so they couldn't be efficient.

../.

emh_visualizatio

Markets are (mostly) efficient

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