

Introduction to Neural Networks

From Brain to Business: How Machines Learn to Predict

Neural Networks for Business Applications

November 26, 2025

Learning Objectives

- **Explain** how biological neurons inspire artificial neural networks
- **Calculate** the output of an artificial neuron given inputs and weights
- **Design** a simple multilayer network architecture
- **Trace** information flow through forward propagation
- **Describe** how networks learn by minimizing prediction errors
- **Evaluate** when neural networks are appropriate for business

The Prediction Challenge: Can We Predict Markets?

The Business Question

Can we predict if a stock price will rise or fall tomorrow?

- Traditional: Statistical analysis, expert intuition
- Challenge: Markets are **complex, non-linear**
- Many factors: price, volume, sentiment, volatility

The Limitation

Rule-based systems cannot capture all interactions

Why This Matters

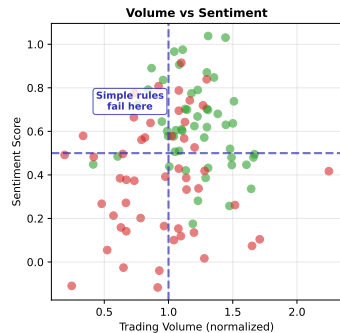
- Better investment decisions
- Risk management
- Portfolio optimization
- Automated trading strategies

What We Need

A system that learns patterns from data, not explicit rules

Our journey begins with understanding how nature solved similar prediction problems

Why Simple Rules Fail: Market Data Complexity



Observe: Can you draw a single line that separates green (up) from red (down) in any panel?



System Requirements

1. Process multiple inputs simultaneously
2. Learn patterns from historical data
3. Handle **non-linear** relationships
4. Improve predictions over time
5. Generalize to new conditions

Key Insight

We need a system that learns, not one we program

Inspiration from Nature

The human brain solves complex pattern recognition every day

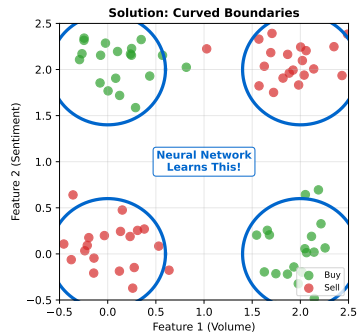
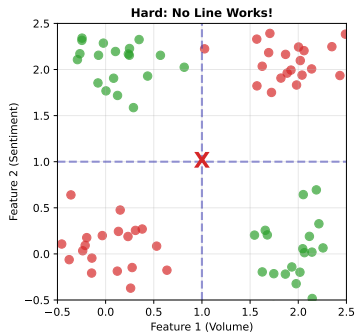
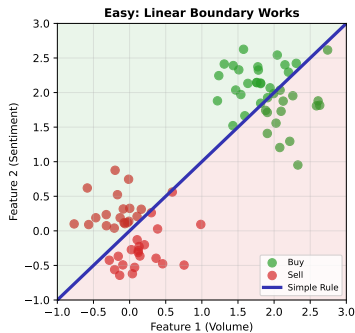
Brain Capabilities

- Processes millions of inputs
- Learns from experience
- Handles ambiguity
- Generalizes to new situations

Can we mimic this for business predictions?

Next: Understanding biological neurons as the foundation

The Goal: Learn Complex Decision Boundaries



Observe: The rightmost panel shows what neural networks can learn - curved boundaries that adapt to data



Part 1: Foundations

From biological neurons to artificial intelligence

Let's begin with the inspiration from nature

[1] – [2] – [3] – [4] – [5]

Biological Neuron Structure

- **Dendrites:** Receive signals
- **Soma:** Integrates weighted signals
- **Axon:** Transmits output
- **Synapses:** Variable connection strengths

Key Principle

Fire when weighted sum exceeds threshold

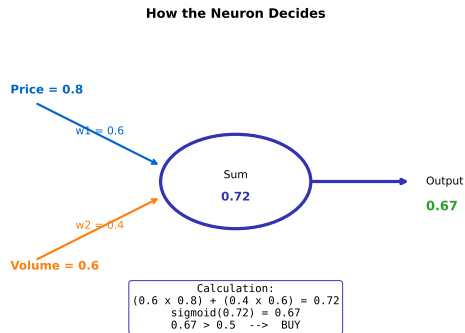
Business AI Insights

1. Multiple inputs combined
2. Weighted connections (importance)
3. Non-linear activation (thresholds)
4. Layered processing (abstraction)

Mathematical models can learn the same way!

Next: See the visual comparison of biological vs artificial neurons

From Concept to Computation: Neuron as Decision Maker

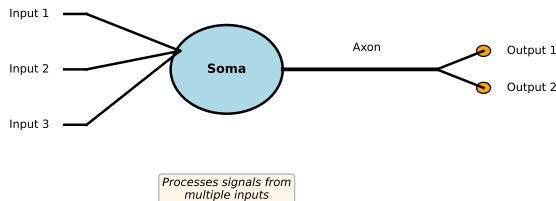


Observe: The decision boundary (purple line) divides the space into BUY and SELL zones based on weighted inputs

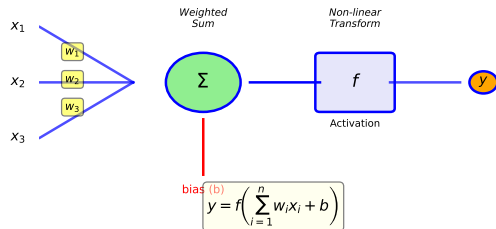


From Biology to Artificial Intelligence

Biological Neuron



Artificial Neuron



Observe: Which biological components map directly to mathematical operations?



Step 1: Weighted Sum

$$z = \sum_{i=1}^n w_i x_i + b$$

- x_i : Inputs (market data)
- w_i : Weights (**learned**)
- b : Bias (baseline)

Step 2: Activation

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- Adds non-linearity
- Output: probability (0 to 1)
- Mimics neuron firing

Complete: $y = \sigma(\sum w_i x_i + b)$

Next: See a concrete example with real market numbers

Practice: Calculate a Neuron's Output

Given Values

- Inputs: $x_1 = 1.2$, $x_2 = 0.8$
- Weights: $w_1 = 0.3$, $w_2 = 0.5$
- Bias: $b = -0.2$

Step 1: Weighted Sum

$$z = w_1x_1 + w_2x_2 + b$$

$$z = (0.3)(1.2) + (0.5)(0.8) + (-0.2)$$

$$z = 0.36 + 0.40 - 0.20 = \mathbf{0.56}$$

Step 2: Apply Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-0.56}}$$

$$\sigma(0.56) = \frac{1}{1+0.571} = \frac{1}{1.571} = \mathbf{0.636}$$

Interpretation

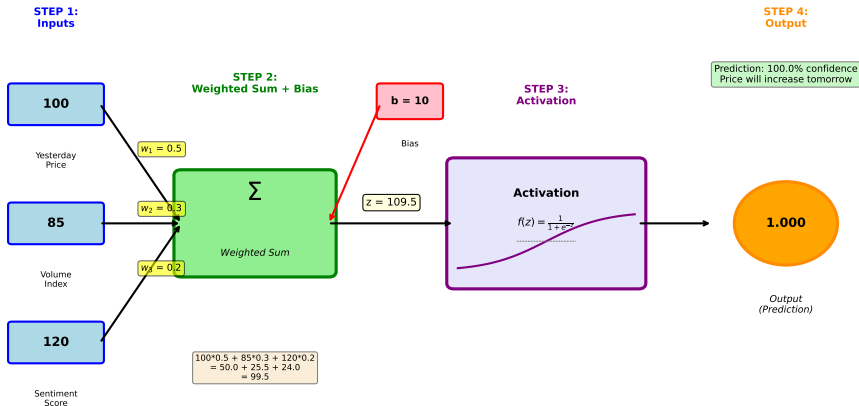
63.6% confidence: price will rise

Your Turn: What if $w_1 = 0.6$?

Work through this calculation – it's the foundation of all neural network predictions

Single Neuron Computation: Step-by-Step Example

How a Neuron Computes: Step-by-Step



Think – Pair – Share

What other business processes might benefit from 'learning from data' instead of following explicit rules?

1. Think (1 min)

Reflect individually on the question

2. Pair (2 min)

Discuss with a neighbor

3. Share (2 min)

Share insights with class

Part 2: Building Blocks

Activation functions and their role in learning

Now that we understand neurons, let's explore what makes them powerful

[1] – [2] – [3] – [4] – [5]

Activation Functions: Why Non-Linearity Matters

The Problem

Without activation functions:

- Networks = linear regression
- Cannot learn complex patterns

Three Common Functions

- **Sigmoid:** $(0,1)$ for probabilities
- **ReLU:** Fast, efficient
- **Tanh:** Zero-centered $(-1,1)$

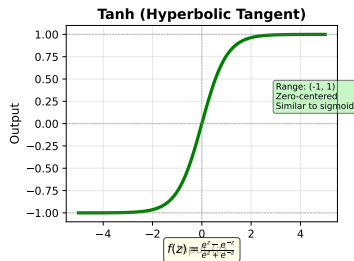
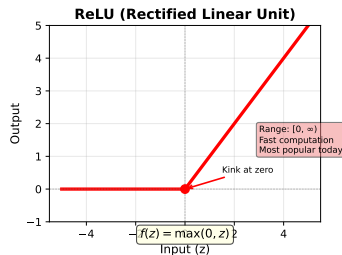
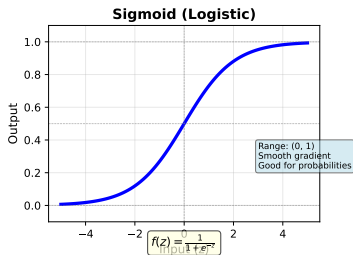
Business Non-Linearity

1. Diminishing returns
2. Threshold effects
3. Saturation points
4. Network effects

Activation functions capture these patterns!

Next: Visual comparison of these three activation functions

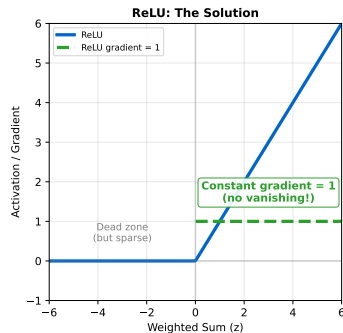
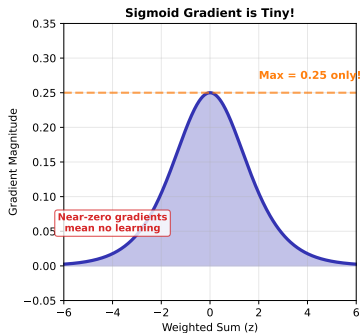
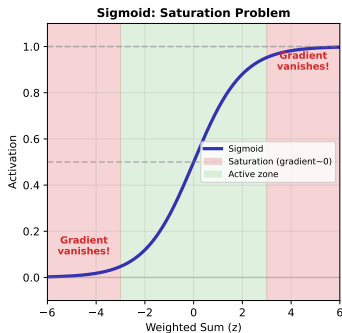
Activation Functions: Adding Non-Linearity



Observe: Where does each function's output change most rapidly? Why does this matter?



Advanced: The Vanishing Gradient Problem



Advanced insight: Sigmoid's tiny gradients in saturation zones slow learning – ReLU solves this in deep networks



The Limitation: Why One Neuron Is Not Enough

What One Neuron Can Do

- Single straight decision boundary
- Separate linearly separable patterns
- Simple rules only

Analogy: One rule for decisions

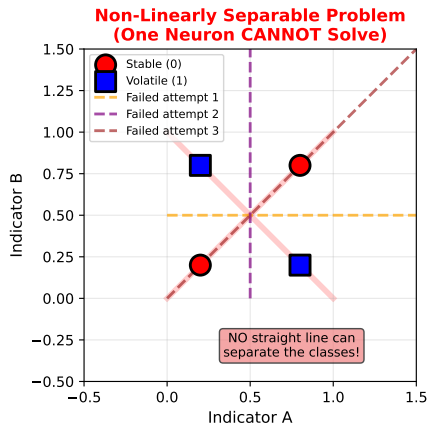
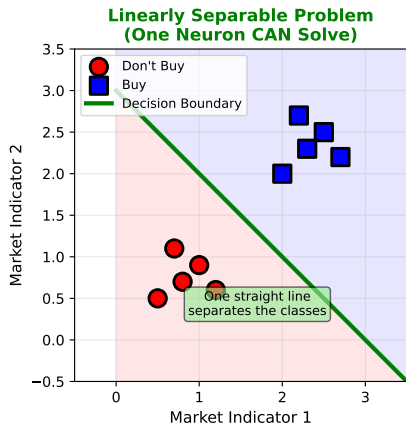
What One Neuron Cannot Do

- Complex, curved boundaries
- XOR-like patterns
- Real-world market interactions

Solution: Multiple Layers!

Next: See the XOR problem that proves one neuron's limitation

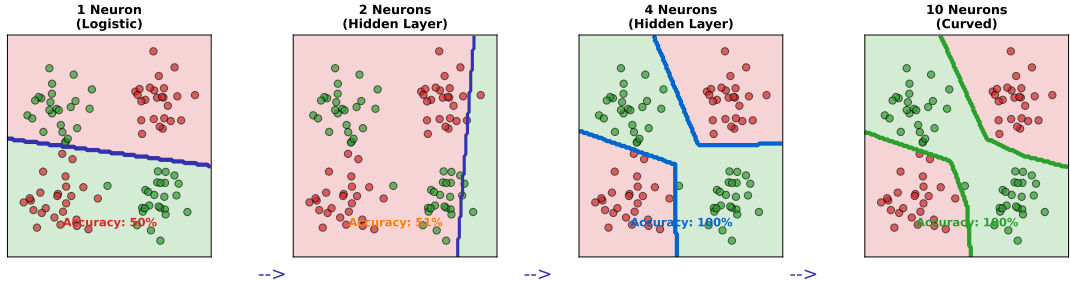
Why One Neuron Is Not Enough



Solution: Use Multiple Layers (Hidden Layers) to Create Non-Linear Decision Boundaries



Solution: How Adding Neurons Creates Curved Boundaries



Key insight: More neurons = more flexibility. Each neuron adds a decision line; combined, they form complex shapes



Think – Pair – Share

Can you think of a business metric that shows diminishing returns or threshold effects?

1. Think (1 min)

Reflect individually on the question

2. Pair (2 min)

Discuss with a neighbor

3. Share (2 min)

Share insights with class

Part 3: Network Architecture

Building layers of intelligence

With building blocks ready, let's construct full networks

[1] – [2] – **[3]** – [4] – [5]

Multi-Layer Architecture

- **Input:** Raw features (no computation)
- **Hidden:** Pattern detection
- **Output:** Final prediction

Result: Buy/Sell decision

Hierarchical Learning

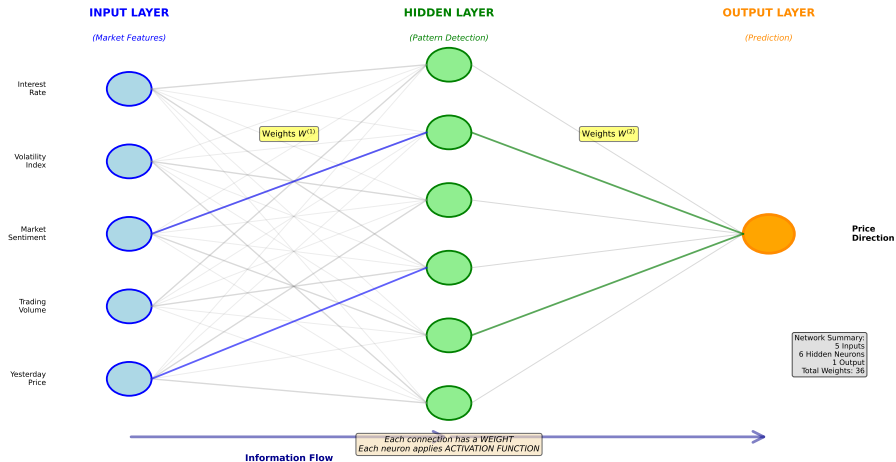
- **Layer 1:** Simple patterns
- **Layer 2:** Complex patterns
- **Layer 3:** Strategic decisions

Each layer builds on previous abstractions

Next: See the full network architecture with all connections

Neural Network Architecture Diagram

Neural Network Architecture: Building Intelligence in Layers



Feature Hierarchy (Part 1): From Raw Data to Patterns

Input Layer: Raw Numbers

The network receives:

- Stock prices: [102.3, 103.1, ...]
- Trading volume: [1.2M, 0.9M, ...]
- Sentiment scores: [0.6, 0.5, ...]

Just numbers – no meaning yet!

Hidden Layer 1: Simple Patterns

The first layer detects:

- Upward trends
- Downward trends
- Momentum shifts
- Volume spikes

Finds basic features in the noise

First transformation: Raw numbers become recognizable patterns

Hidden Layer 2: Complex Patterns

Combines simple patterns:

- Trend + High volume = **Bullish**
- Trend + Low volume = **Bearish**
- Support/resistance levels
- Multi-day patterns

Strategic insights emerge!

Output Layer: Trading Decision

Final decision:

- **BUY**: 68% confidence
- **SELL**: 32% confidence

Since 68% > 50% threshold:

ACTION: BUY

Final transformation: Patterns become actionable trading signals

Forward Propagation: How Networks Make Predictions

The Forward Pass

1. **Input:** Feed market features
2. **Hidden:** $a = \sigma(Wx + b)$
3. **Output:** $y = \sigma(Wa + b)$

All neurons compute in parallel!

Example

Input: price=105.2, volume=0.75

Output: $y = 0.742$

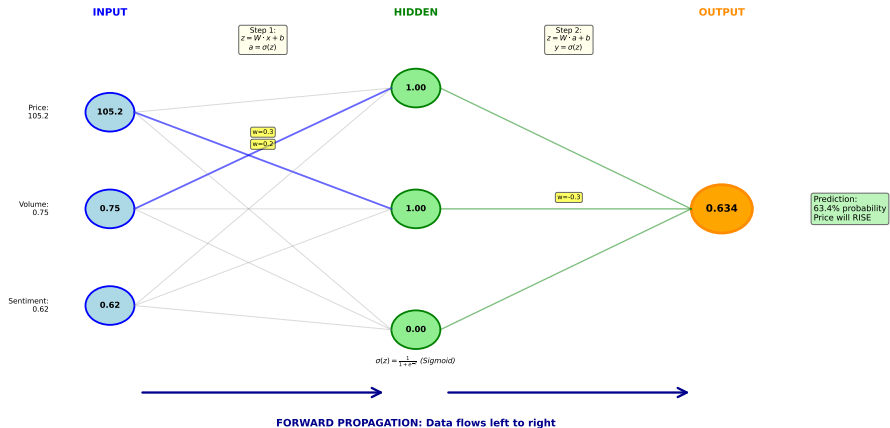
Interpretation:

- 74.2% confidence price rises
- $y > 0.5$: **BUY**
- $y < 0.5$: **SELL**

Next: See forward propagation with actual numbers and calculations

Forward Propagation: Detailed Example

Forward Propagation: Making a Prediction



Think – Pair – Share

For your industry, what would be the 'inputs' and 'outputs' of a useful neural network?

1. Think (1 min)

Reflect individually on the question

2. Pair (2 min)

Discuss with a neighbor

3. Share (2 min)

Share insights with class

Part 4: Learning Process

How networks learn from mistakes

We can make predictions – now let's learn how to improve them

[1] – [2] – [3] – [4] – [5]

Learning Steps

1. **Predict** with random weights
2. **Measure error:**

$$L = \frac{1}{n} \sum (y - \hat{y})^2$$

3. **Adjust weights:**

$$w_{new} = w_{old} - \eta \nabla L$$

4. **Repeat** until convergence

Example

Predicted: 55% rise, Actual: fell

Error: $(0 - 0.55)^2 = 0.30$

Learning:

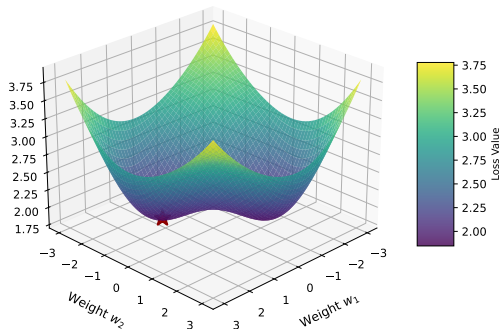
- Calculate gradient direction
- Move weights to reduce error

Like a trader learning from mistakes

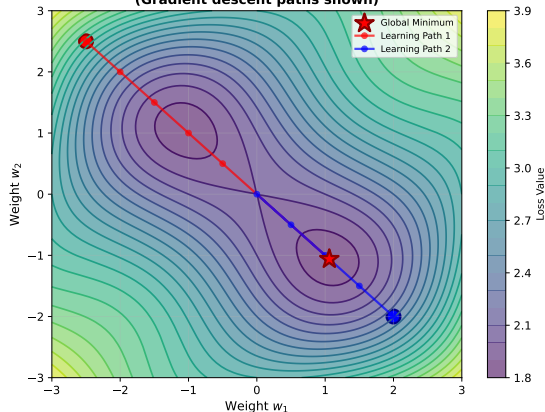
Next: Visualize the loss landscape that we're trying to navigate

Loss Landscape: The Error Surface

Loss Landscape in 3D
(Error as a function of weights)



Contour View: Loss Landscape
(Gradient descent paths shown)



Goal: Find the weights that minimize the loss
(The red star shows the optimal solution)

Observe: What happens if we start from different random initial weights?



Algorithm

1. Calculate gradient (slope)
2. Step opposite direction
3. Repeat until convergence

Learning Rate Trade-offs

- **Too small:** Slow
- **Too large:** Unstable
- **Just right:** Steady

Business Analogy

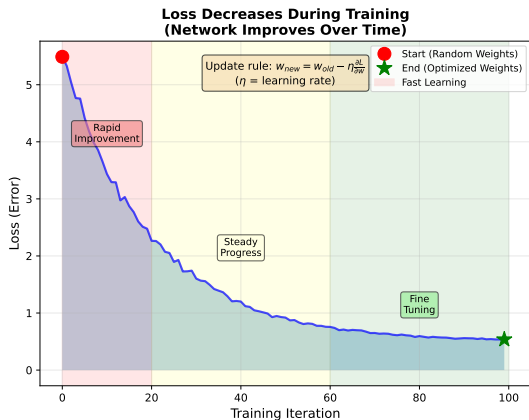
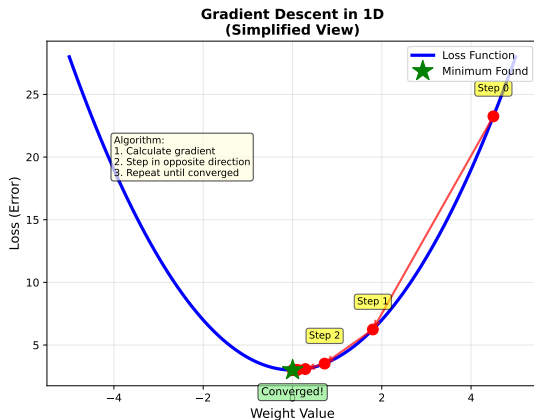
Like a trader learning:

- Fast learning from obvious patterns
- Steady fine-tuning
- Convergence to optimal rules

Gradient shows fastest error reduction

Next: See how loss decreases over training iterations

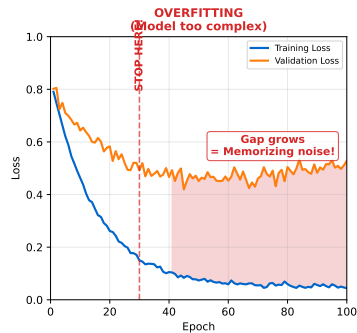
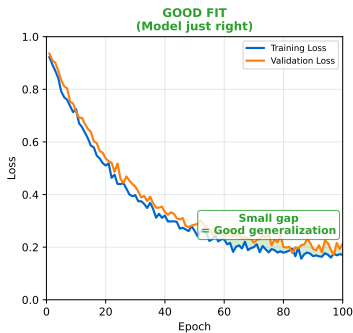
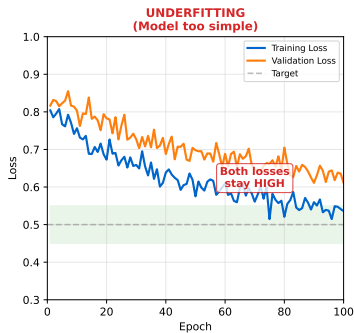
Gradient Descent: Learning by Stepping Downhill



Observe: How does the step size (learning rate) affect how quickly we reach the minimum?



Critical Concept: Overfitting vs Underfitting



Key practical skill: Watch for diverging training/validation loss – that's when to stop training!



Think – Pair – Share

How is gradient descent similar to how businesses optimize through trial and error?

1. Think (1 min)

Reflect individually on the question

2. Pair (2 min)

Discuss with a neighbor

3. Share (2 min)

Share insights with class

Part 5: Application

Putting it all together with market prediction

Theory complete – let's apply everything to a real case

[1] – [2] – [3] – [4] – [5]

Business Application

- **Goal:** Predict price direction
- **Data:** 60 days market data

Input Features

1. Stock Price
2. Trading Volume
3. Market Sentiment
4. Volatility Index

Target Variable

Binary: 1 = up, 0 = down

Network outputs: $p(\text{rise})$

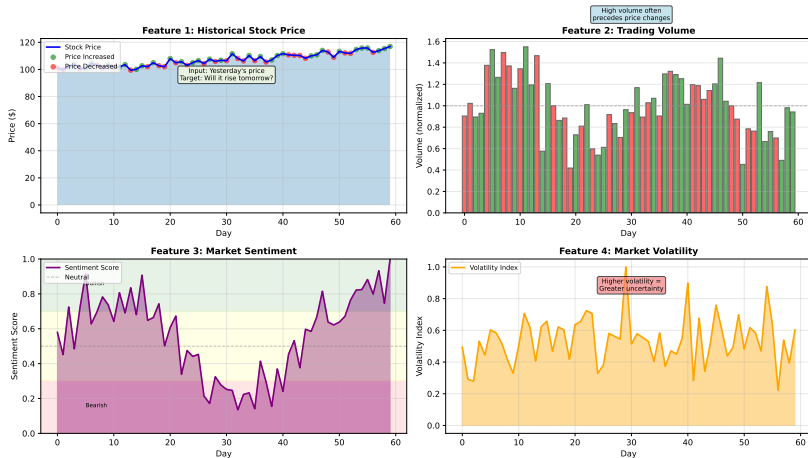
Setup

- Train: 45 days
- Test: 15 days
- Network: 4-6-1

Next: See the actual market data used for training

Market Data: Input Features for Neural Network

Market Data: Input Features for Neural Network



Neural Network Input: All 4 features for each day | Output: Probability of price increase tomorrow



The Experiment

- Before: Random weights (coin flip)
- After: Learned weights
- Test: 30 days unseen data

Results

- **Before:** 50% accuracy
- **After:** 70% accuracy
- **Gain:** +20 points

What Network Learned

- Volume + price + sentiment patterns
- Volatility indicates uncertainty
- Sentiment confirms trends

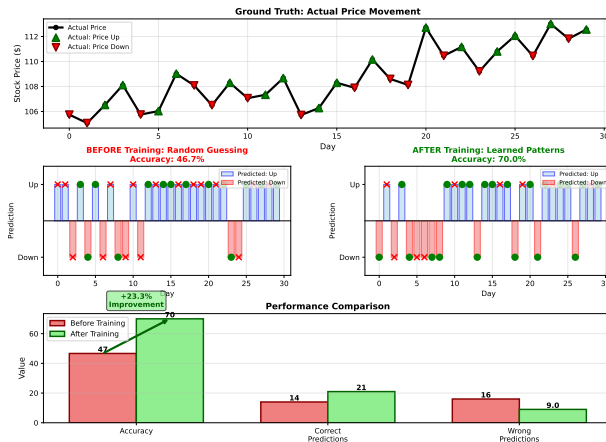
Discovered from data alone!

70% is good for markets (100% impossible)

Next: See detailed before/after comparison with prediction accuracy

Prediction Results: Before vs After Training

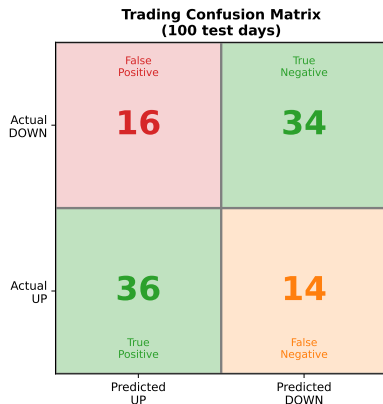
Neural Network Performance: Before vs After Training



Training transforms random guessing into intelligent prediction by learning patterns from data



Understanding Model Performance: Confusion Matrix



Performance Metrics Explained

ACCURACY
70%

PRECISION
69%

RECALL
72%

F1 SCORE
0.71

Accuracy:	Overall correct predictions	70/100
Precision:	When we say BUY, how often right?	36/52
Recall:	Of all UP days, how many caught?	36/50
F1 Score:	Balance of precision & recall	harmonic mean

Trading Insight: 69% precision means ~1/3 of BUY signals are wrong!

Business insight: 70% accuracy means different things for trading – precision determines false BUY rate



Think – Pair – Share

What data would you need to predict customer behavior in your domain?

1. Think (1 min)

Reflect individually on the question

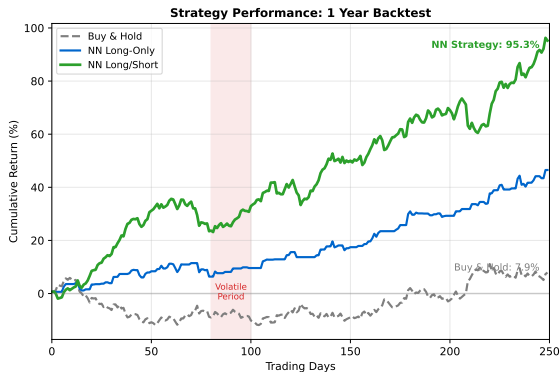
2. Pair (2 min)

Discuss with a neighbor

3. Share (2 min)

Share insights with class

The Business Case: Strategy Backtest Results



PERFORMANCE COMPARISON

Metric	Buy & Hold	NN Strategy
Total Return	7.9%	95.3%
Sharpe Ratio	0.40	3.76
Max Drawdown	-16.8%	-9.2%
Win Rate	52%	66%

Key Insight: 70% accuracy translates to significant alpha!

(Backtest only - past performance does not guarantee future results)

Bottom line: 70% accuracy translates to meaningful outperformance – this is why neural networks matter for business



Summary: Three Key Insights

Definition

1. Neurons Compute Weighted Sums

Each neuron:

$$y = \sigma \left(\sum w_i x_i + b \right)$$

- Weighted inputs
- Non-linear activation
- Parallel processing

2. Networks Learn from Errors

Gradient descent:

$$w_{new} = w_{old} - \eta \nabla L$$

- Measure prediction error
- Adjust weights
- Minimize loss function

3. Patterns Emerge from Data

- No explicit rules
- Discovers relationships

Business Example

Market Prediction

Input: Price (0.8), Volume (0.6)

Computation:

- $(0.6 \times 0.8) + (0.4 \times 0.6) = 0.72$
- $\sigma(0.72) = 0.67$
- $0.67 > 0.5 \rightarrow \text{BUY}$

Learning Process

- Predicted: 55% rise, Actual: fell
- Error: $(0 - 0.55)^2 = 0.30$
- Adjust: Reduce weights on price
- Next time: Better prediction

Discovered Patterns

- Volume + sentiment \rightarrow direction
- 70% accuracy (no manual rules!)
- Outperforms buy-and-hold

Quick Check: Test Your Understanding

Q1: What does the activation function do?

- (a) Stores the input data
- (b) **Adds non-linearity to enable complex patterns**
- (c) Calculates the learning rate

Q2: Why do we need multiple layers?

- (a) To make training faster
- (b) To use more data
- (c) **To learn hierarchical, complex patterns**

Q3: What does gradient descent minimize?

- (a) The number of neurons
- (b) **The prediction error (loss function)**
- (c) The training time

Check Your Answers

Answer Key

- Q1: (b) Non-linearity
- Q2: (c) Hierarchical patterns
- Q3: (b) Loss/error

Scoring

- 3/3: Excellent grasp!
- 2/3: Review that topic
- 1/3: Revisit core concepts

If any answer surprised you, go back and review that section

Use Neural Networks When

- Large dataset (thousands+ examples)
- Complex patterns
- Difficult to specify rules
- Pattern recognition tasks
- Black-box acceptable

Applications

Churn, fraud, recommendations, images, NLP

Do NOT Use When

- Small dataset
- Simple relationships
- Need interpretability
- Rules are known
- Real-time constraints

Alternatives

Regression, decision trees, expert systems

Choose the right tool - neural networks are powerful but not always appropriate

Technical Limitations

- Data hungry
- Black box decisions
- Overfitting risk
- No guarantees
- Computational cost

Ethical Concerns

- **Fairness:** Biased data leads to biased predictions
- **Transparency:** GDPR requires explanations
- **Accountability:** Who is responsible?
- **Impact:** Job displacement, market stability

With great predictive power comes great responsibility!

Always consider ethical implications before deploying AI systems

Output Layer Gradient

$$\frac{\partial L}{\partial w^{(2)}} = (\hat{y} - y) \cdot \sigma'(z) \cdot a$$

Hidden Layer Gradient

$$\frac{\partial L}{\partial w^{(1)}} = \delta^{(2)} \cdot w^{(2)} \cdot \sigma'(z^{(1)}) \cdot x$$

Loss Functions

MSE (Regression):

$$L = \frac{1}{n} \sum (y - \hat{y})^2$$

Cross-Entropy (Classification):

$$L = - \sum [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Backpropagation efficiently computes how each weight contributed to the error

Practical Tips

- Start simple (baseline first)
- Feature engineering matters
- Avoid overfitting (validation, dropout)
- Tune hyperparameters
- Monitor training curves

Books

Goodfellow (Deep Learning), Nielsen, Geron

Courses

- Andrew Ng (Coursera)
- Fast.ai
- MIT 6.S191

Tools

PyTorch, TensorFlow, scikit-learn

Practice

Kaggle, Yahoo Finance, UCI Repository

Best way to learn: Build real projects with real data!

Design Challenge

You are a data scientist at a retail company.

Problem: Predict customer churn

Data Available:

- Demographics
- Purchase history
- Service interactions
- Website engagement

Your Tasks

1. Design network architecture
2. Select input features
3. Choose activation functions
4. Select loss function
5. Define evaluation metrics
6. Identify ethical concerns
7. Plan stakeholder explanation

Discuss in groups - there's no single right answer!