Introduction to Neural Networks

From Brain to Business: How Machines Learn to Predict

Neural Networks for Business Applications

November 23, 2025

By the End of This Lecture, You Will Be Able To:

- **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 for a business problem
- **Trace** information flow through forward propagation
- **Describe** how networks learn by minimizing prediction errors
- **Evaluate** when neural networks are appropriate for business predictions
- **Assess** the ethical implications of automated prediction systems

The Prediction Challenge: Can We Predict Markets?

The Business Question:

- Can we predict if a stock price will rise or fall tomorrow?
- Traditional methods: Statistical analysis, expert intuition, rule-based systems
- Challenge: Markets are complex, non-linear systems
- Many interacting factors: price history, volume, sentiment, volatility, interest rates

Why This Matters:

- Better investment decisions ($\rightarrow$ higher returns)
- Risk management ($\rightarrow$ protect capital)
- Portfolio optimization ($\rightarrow$ balanced exposure)
- Automated trading strategies ($\rightarrow$ scalability)

Our journey begins with understanding how nature solved similar prediction problems

What We Need: A Learning System

Requirements for Market Prediction System

A system that can:

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 market conditions

Inspiration from Nature

The human brain solves complex pattern recognition tasks every day by learning from experience.

Can we mimic this for business predictions?

▶ Let's explore how brains work...

Next: Understanding biological neurons as the foundation

Nature's Computer: How Your Brain Makes Predictions

The Biological Neuron Structure:

- **Dendrites:** Receive signals from other neurons (like input channels)
- **Soma (cell body):** Integrates incoming signals with different weights
- **Axon:** Transmits output to next neurons (like output wire)
- **Synapses:** Connection points with varying strengths

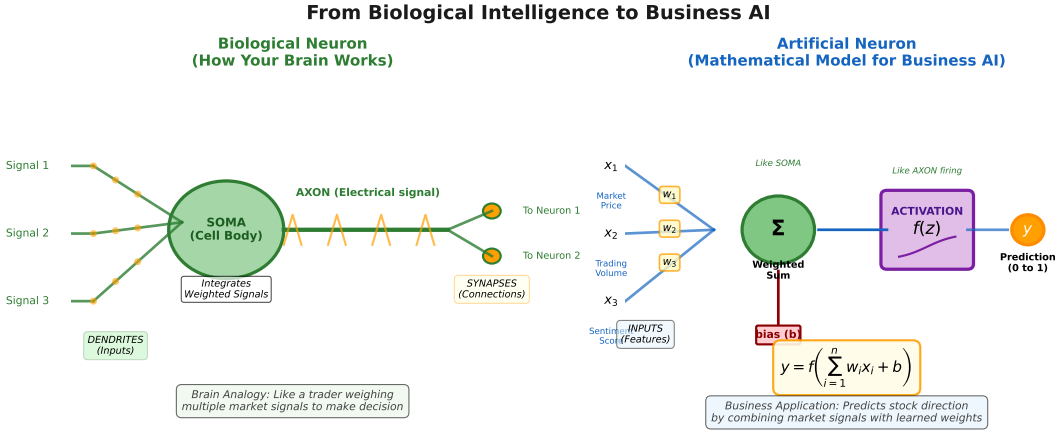
Key Insights for Business AI:

1. Multiple inputs combined → Consider many market factors simultaneously
2. Weighted connections → Some factors matter more than others
3. Non-linear activation → Threshold effects (tipping points)
4. Layered processing → Abstract reasoning emerges from simple units

The Bridge to Mathematics

Just as your brain learned to recognize patterns through experience, we can create mathematical models that learn the same way!

Next: See the visual comparison of biological vs artificial neurons



The artificial neuron mathematically mimics biological signal integration and activation

The Artificial Neuron: Mathematical Model

How an Artificial Neuron Works:

Step 1: Weighted Sum (Mimics Soma Integration)

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

- x_i: Input features (market data: price, volume, sentiment)
- w_i: Weights (learned from data, not programmed!)
- b: Bias term (baseline adjustment)

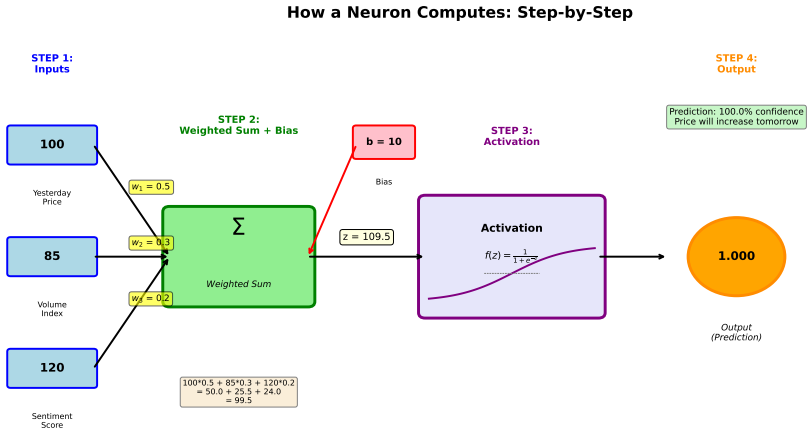
This is like dendrites weighting incoming signals differently

Step 2: Activation Function (Mimics Axon Firing)

$$y = f(z) = \frac{1}{1 + e^{-z}} \quad \text{(Sigmoid function)}$$

- f: Activation function (adds crucial non-linearity)
- Output: probability between 0 and 1

Single Neuron Computation: Step-by-Step Example



With market inputs (price=100, volume=85, sentiment=120), the neuron predicts 100% probability of price increase

Activation Functions: Why Non-Linearity Matters

The Problem with Linear Functions:

- Without activation functions, neural networks would just be fancy linear regression
- Real business relationships are non-linear!

Real-World Non-Linearity Examples in Business:

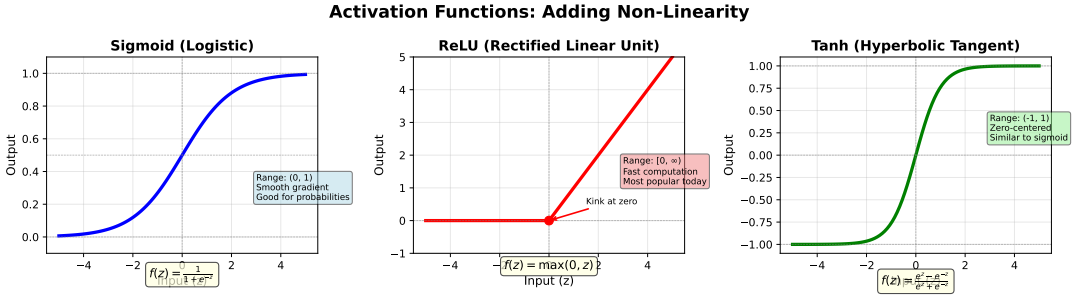
1. **Diminishing returns:** Doubling marketing spend doesn't double sales
2. **Threshold effects:** Market sentiment must reach tipping point to trigger buying
3. **Saturation:** Customer engagement plateaus beyond certain point
4. **Network effects:** Value increases non-linearly with user base

Three Common Activation Functions:

- **Sigmoid:** Smooth (0,1) range - perfect for probabilities
- **ReLU:** Fast, efficient - most popular in modern networks
- **Tanh:** Zero-centered (-1,1) - alternative to sigmoid

Next: Visual comparison of these three activation functions

Activation Functions: Visual Comparison



Each activation function has different properties: Sigmoid for probabilities, ReLU for speed, Tanh for zero-centered outputs

The Limitation: Why One Neuron Is Not Enough

What One Neuron Can Do:

- Draw a single straight line (hyperplane) in feature space
- Separate linearly separable patterns
- Example: "Buy if (price $>$ 100 AND volume $>$ 50)"

Business Analogy: One simple rule for decision-making

What One Neuron Cannot Do:

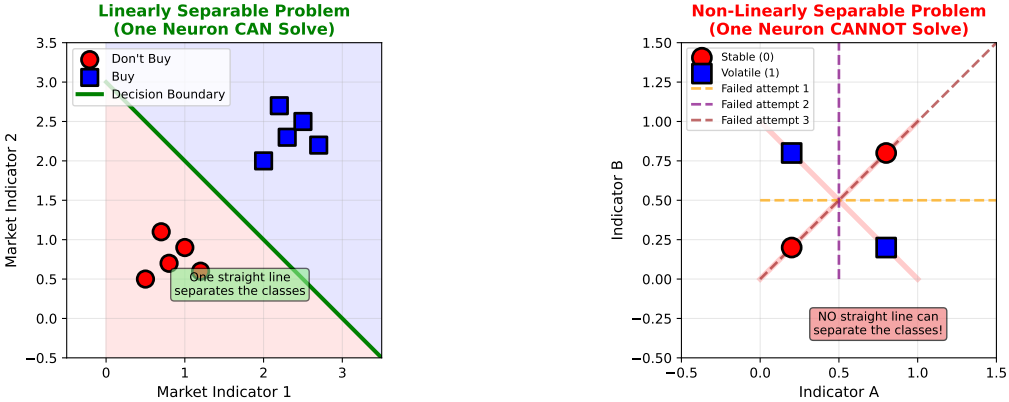
- Complex, curved decision boundaries
- XOR-like patterns: "Buy when (high price XOR high volume) BUT NOT both"
- Real-world market patterns with interactions!

Business Reality: Multiple interacting factors, non-linear relationships, conditional dependencies

Solution: Use Multiple Layers of Neurons!

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

Left: Linearly separable (one neuron works). Right: XOR pattern (one neuron fails, need hidden layers)

Building the Network: Layers of Intelligence

The Multi-Layer Architecture:

Input Layer
- Raw market features
- No computation
- Like sensory neurons

Hidden Layer(s)
- Pattern detection
- Feature combinations
- Like association cortex

Example: Learns "high volume + rising price = momentum"

Output Layer
- Final prediction
- Probability output
- Like motor neurons

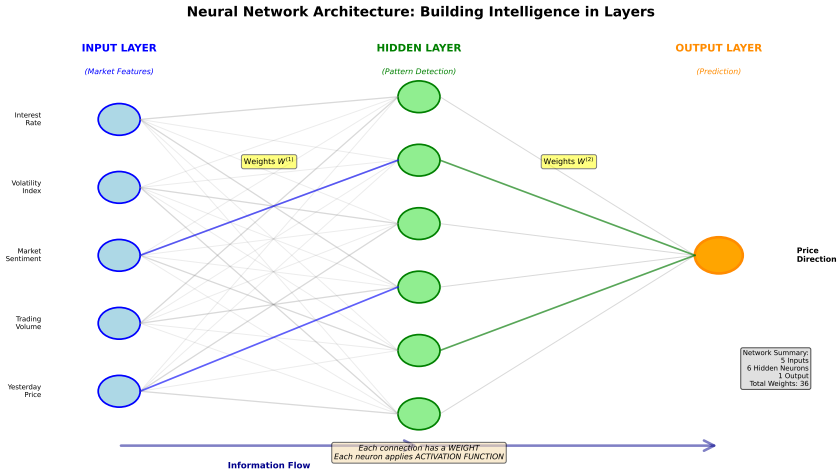
Result: Buy/Sell decision

Key Principle: Hierarchical Learning

- Layer 1: Detects simple patterns ("price rising", "volume high")
- Layer 2: Combines into complex patterns ("strong momentum", "weak support")
- Layer 3: Makes strategic decisions ("high probability buy signal")

Next: See the full network architecture with all connections

Neural Network Architecture Diagram



5 inputs → 6 hidden neurons → 1 output. Total: 36 weights to learn from data

Forward Propagation: How Networks Make Predictions

The Forward Pass Process:

1. **Input:** Feed today's market features into input layer
   - Example: price=105.2, volume=0.75, sentiment=0.62
2. **Hidden Layer Computation:** Each neuron processes inputs

$$a^{(1)} = \sigma(W^{(1)}x + b^{(1)})$$

   - Detects patterns like "rising trend" or "high volatility"
3. **Output Layer Computation:** Final prediction

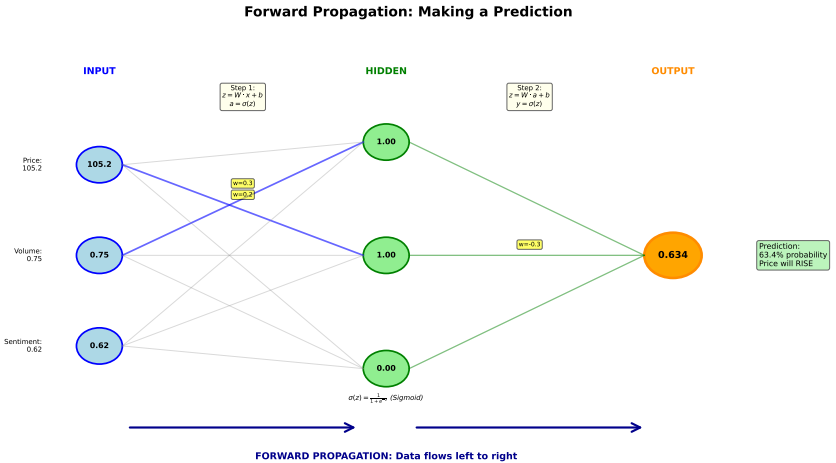
$$y = \sigma(W^{(2)}a^{(1)} + b^{(2)}) = 0.742$$

   - Interpretation: 74.2% confidence price will rise
4. **Decision:** Convert probability to action
   - If $y > 0.5 \rightarrow$ **BUY**
   - If $y < 0.5 \rightarrow$ **SELL**

Efficiency: All neurons in a layer compute in parallel (matrix operations)!

Next: See forward propagation with actual numbers and calculations

Forward Propagation: Detailed Example



Data flows left to right: inputs (105.2, 0.75, 0.62) → hidden activations → output (0.742) = 74% buy confidence

Learning from Mistakes: The Training Process

The Critical Question:

How does the network learn the RIGHT weights?

Learning Process (Like Human Learning):

1. **Make prediction with random weights** (initial guess)
   - Network predicts: 55% probability price will rise
2. **Measure error (Loss Function)**

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \text{(Mean Squared Error)}$$

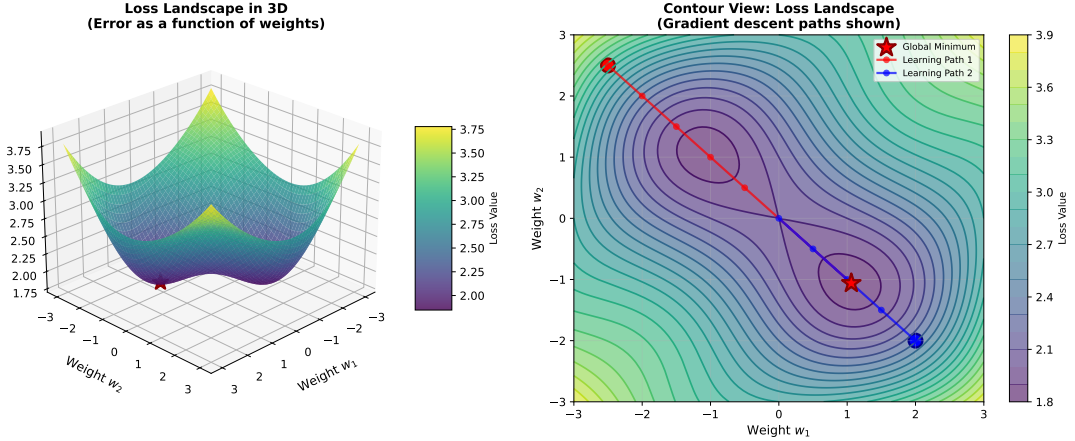
   - Actual: price fell (y=0), Predicted: 0.55, Error: $(0 - 0.55)^2 = 0.30$
3. **Adjust weights to reduce error (Gradient Descent)**

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

   - η = learning rate (how fast we learn from mistakes)
   - Move weights in direction that reduces error
4. **Repeat thousands of times until convergence**

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

Loss Landscape: The Error Surface



Goal: Find weights (red star) that minimize loss. Different starting points converge to same optimum through gradient descent

Gradient Descent: Learning by Stepping Downhill

The Gradient Descent Algorithm:

Algorithm Steps:

1. **Calculate gradient:** Which direction increases error?

$$\frac{\partial L}{\partial w} = \text{slope of loss function}$$

2. **Step in opposite direction:** Move toward lower error

$$w_{new} = w_{old} - \eta \times \text{gradient}$$

3. **Repeat until convergence:** Keep stepping until error stops decreasing

Learning Rate (η) Trade-offs:

- Too small: Slow learning, may never converge
- Too large: May overshoot minimum, unstable
- Just right: Steady progress toward optimum

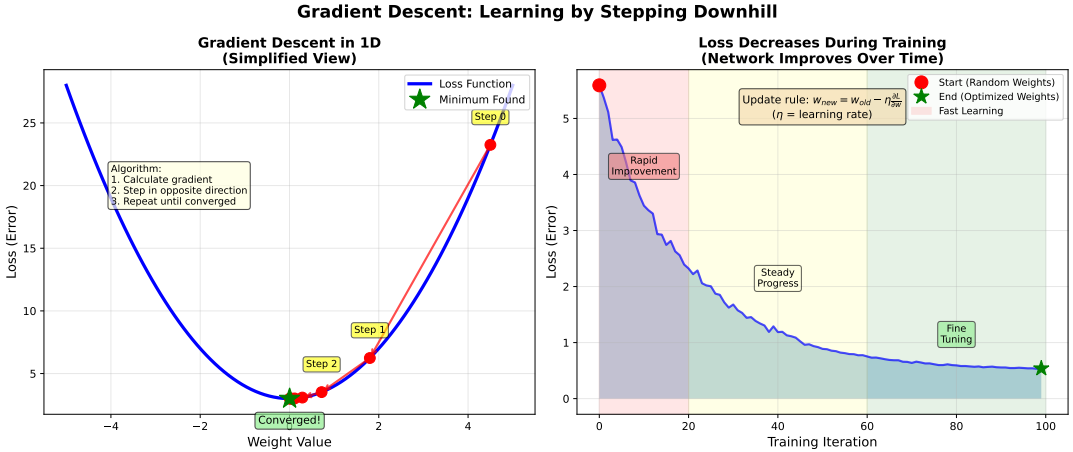
Business Analogy:

Like a trader learning from past mistakes:

- Fast learning phase: Rapid improvement from obvious patterns
- Steady progress: Fine-tuning strategy
- Convergence: Optimal trading rules learned

Next: See how loss decreases over training iterations

Gradient Descent: Optimization in Action



Left: 1D visualization showing steps toward minimum. Right: Loss decreasing over 100 training iterations

Putting It Together: Market Prediction Case Study

The Business Application:

- **Goal:** Predict if stock price will rise or fall tomorrow
- **Data:** 60 days of historical market data
- **Features:** 4 input variables per day

Input Features (Network Inputs):

1. **Historical Stock Price:** Yesterday's closing price
2. **Trading Volume:** How much stock was traded (normalized)
3. **Market Sentiment:** News/social media sentiment score (0-1)
4. **Volatility Index:** Measure of market uncertainty (0-1)

Target Variable (Network Output):

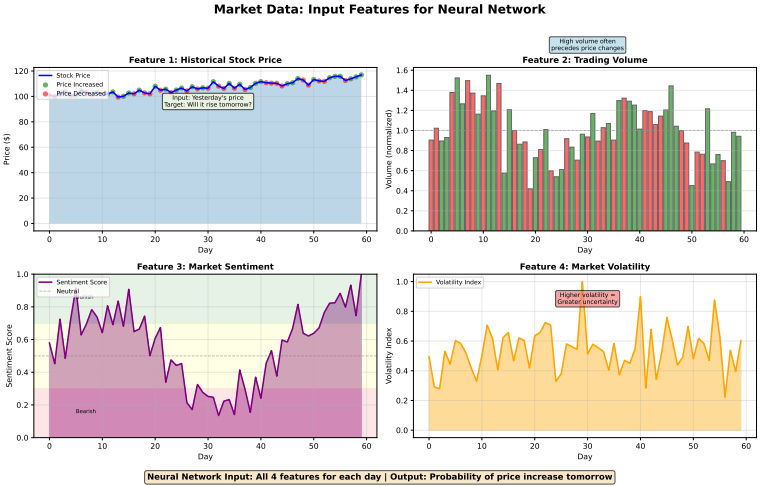
- Binary: 1 = price increased, 0 = price decreased
- Network outputs probability: p(price rise)

Training Setup:

- Training data: First 45 days (learn patterns)
- Test data: Last 15 days (evaluate performance)
- Network: 4 inputs $\rightarrow$ 6 hidden $\rightarrow$ 1 output

Next: See the actual market data used for training

Market Data: Input Features for Neural Network



60 days of market data: price trend (up/down markers), volume bars, sentiment score, volatility index

Training Results: Before vs After

The Experiment:

- **Before training:** Random weights → random predictions (coin flip)
- **After training:** Learned weights → intelligent predictions
- **Test set:** 30 days of unseen data

Key Results:

- **Before training:** ≈50% accuracy (random guessing)
- **After training:** ≈70% accuracy (learned patterns!)
- **Improvement:** +20 percentage points through learning

What the Network Learned:

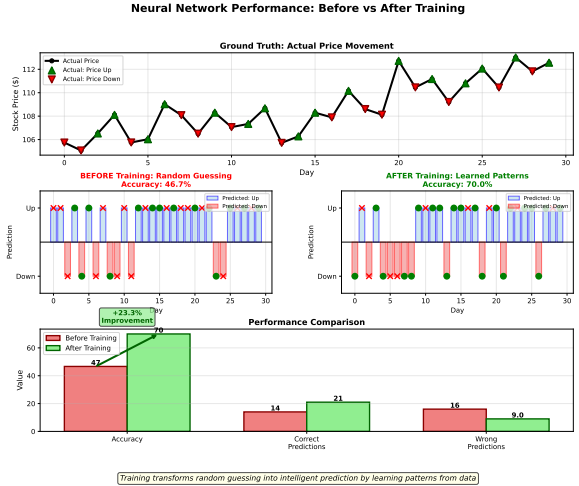
- High volume + rising price + positive sentiment = likely continued rise
- Low volume + high volatility = uncertain, avoid prediction
- Sentiment lags price but confirms trends
- *Network discovered these patterns from data alone!*

Reality Check:

- 70% is good for financial markets
- 100% is impossible (markets are inherently unpredictable)
- Still useful: 70% accuracy over many trades = profit

Next: See detailed before/after comparison with prediction accuracy

Prediction Results: Before vs After Training



Top: Actual prices. Middle: Before training (random, 50% accuracy). After training (learned patterns, 70% accuracy). Bottom: Metrics comparison

Summary: From Neurons to Predictions

Our Journey (Concept → Visualization):

1. Biology: Neurons integrate weighted signals → Chart
2. Mathematics: $y = f(\sum w_i x_i + b)$ → Chart
3. Non-linearity: Activation functions enable complexity → Chart
4. Limitation: One neuron fails on XOR → Chart
5. Architecture: Layers learn hierarchical patterns → Chart
6. Forward prop: Making predictions → Chart
7. Learning: Loss landscape → Chart
8. Optimization: Gradient descent → Chart
9. Application: Market data → Chart
10. Results: 50% → 70% accuracy → Chart

Key Insight:
Neural networks **learn from data**, not explicit programming!

The separation of concept and visualization helped you see both theory and practice

When to Use Neural Networks

Use Neural Networks When:

- Large dataset (thousands+ examples)
- Complex patterns (non-linear relationships)
- Difficult to specify rules (too many cases)
- Pattern recognition (images, speech, text)
- Acceptable black-box (don't need to explain every decision)
- Computational resources available

Example Applications:

- Customer churn prediction
- Fraud detection
- Recommendation systems
- Image recognition
- Natural language processing

Do NOT Use When:

- Small dataset (few hundred examples)
- Simple linear relationships
- Need interpretability (regulatory requirements)
- Rules are known (use rule-based system instead)
- High stakes, low data (medical diagnosis with small samples)
- Real-time constraints (millisecond decisions)

Better Alternatives:

- Linear/logistic regression (interpretable)
- Decision trees (explainable)
- Expert systems (known rules)
- Statistical models (small data)

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

Important Limitations & Ethical Responsibilities

Technical Limitations:

- **Data hungry:** Need thousands of examples
- **Black box:** Hard to explain decisions to regulators
- **Overfitting:** May memorize training data, fail on new data
- **No guarantees:** Markets are inherently unpredictable
- **Computational cost:** Training requires GPUs, time, energy
- **Hyperparameter sensitivity:** Architecture choices matter

Ethical Responsibilities:

- **Fairness:** Biased data $\rightarrow$ biased predictions
  - Historical hiring data may encode discrimination
  - Loan approval systems may disadvantage minorities
- **Transparency:** Can you explain decisions to stakeholders?
  - GDPR requires "right to explanation"
  - Use interpretability tools (SHAP, LIME)
- **Accountability:** Who is responsible when AI fails?
  - Flash crash: Automated trading gone wrong
  - Wrong medical diagnosis: Who pays?
- **Societal impact:** Unintended consequences
  - Job displacement through automation
  - Algorithmic trading destabilizes markets

With great predictive power comes great responsibility!

Always consider ethical implications before deploying AI systems in real-world business contexts

Appendix: Mathematical Details (Backpropagation)

How Gradient Descent Works: The Chain Rule
For a simple 2-layer network, we compute gradients layer-by-layer working backwards:

Output Layer Gradient:

$$\begin{aligned}\frac{\partial L}{\partial w^{(2)}} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}} \\ &= (\hat{y} - y) \cdot \sigma'(z^{(2)}) \cdot a^{(1)}\end{aligned}$$

Hidden Layer Gradient (using chain rule):

$$\begin{aligned}\frac{\partial L}{\partial w^{(1)}} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial w^{(1)}} \\ &= (\hat{y} - y) \cdot \sigma'(z^{(2)}) \cdot w^{(2)} \cdot \sigma'(z^{(1)}) \cdot x\end{aligned}$$

Common Loss Functions:

- **Mean Squared Error (Regression):** $L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
- **Binary Cross-Entropy (Classification):** $L = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$

Backpropagation efficiently computes how each weight contributed to the final error

Appendix: Practical Tips & Resources

Practical Tips for Building Business Neural Networks:

- **Start simple:** Try linear/logistic regression first as baseline
- **Feature engineering matters:** Good inputs > complex architecture
- **Avoid overfitting:** Use validation sets, regularization (L1/L2), dropout
- **Hyperparameter tuning:** Learning rate, architecture, batch size, epochs
- **Interpretability tools:** SHAP values, attention weights, feature importance
- **Monitor training:** Plot loss curves, check for convergence
- **Cross-validation:** Don't rely on single train/test split

Recommended Resources:

Books:

- Deep Learning (Goodfellow et al.)
- Neural Networks and Deep Learning (Nielsen)
- Hands-On Machine Learning (Geron)

Courses:

- Andrew Ng - Machine Learning (Coursera)
- Fast.ai - Deep Learning for Coders
- MIT 6.S191 - Intro to Deep Learning

Tools & Frameworks:

- PyTorch (flexible, research-friendly)
- TensorFlow/Keras (production-ready)
- scikit-learn (simpler models)

Practice Datasets:

- Kaggle competitions
- Yahoo Finance API
- UCI ML Repository

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

Appendix: Practice Problem for Business Students

Design Challenge: You are a data scientist at a retail company.

Problem: Predict customer churn (will customer leave next month?)

Available Data:

- Customer demographics (age, location, income)
- Purchase history (frequency, recency, monetary value)
- Customer service interactions (calls, complaints, resolutions)
- Website engagement (visits, time spent, pages viewed)

Your Tasks (Work in Groups):

1. Design a neural network architecture: How many layers? How many neurons per layer?
2. What would be your input features? Raw data or engineered features?
3. What activation functions would you use and where?
4. What loss function is appropriate for this problem?
5. How would you evaluate model performance? (accuracy, precision, recall, F1?)
6. What are potential ethical concerns with automated churn prediction?
7. How would you explain predictions to business stakeholders?
8. When would you NOT use a neural network for this problem?

Discuss in groups and present your design - there's no single right answer!