Module 1: The Birth of Neural Computing

From Biological Inspiration to the Perceptron (1943-1969)

Neural Networks for Finance

BSc Lecture Series

November 26, 2025

The Investment Committee

How Does a Committee Make Decisions?

Imagine an investment committee evaluating a stock:

- **Analyst A**: "Strong earnings growth" (+1 vote)
- **Analyst B**: "High debt levels" (-1 vote)
- **Analyst C**: "Good momentum" (+1 vote)
- **Senior Partner**: "Market risk is elevated" (-2 votes)

The Decision Process:

1. Gather evidence from each analyst
2. Weight opinions by seniority/expertise
3. Sum the weighted votes
4. If total > threshold: **Buy**

Weighted Voting

Analyst	Vote	Weight
Analyst A	+1	1.0
Analyst B	-1	1.0
Analyst C	+1	1.0
Senior Partner	-1	2.0
Weighted Sum		**-1.0**

Decision: Don't Buy

Finance Hook: This is exactly how a perceptron works!

What If Machines Could Decide?

The Central Question
In 1943, scientists asked:

"Can we build a machine that learns to make decisions like a brain?"

Why This Matters for Finance:

- Humans are slow and biased
- Markets process millions of data points
- Pattern recognition at scale
- Consistent, emotionless decisions

The Promise
If we could capture how neurons compute:

- Automatic stock screening
- Risk assessment at scale
- Pattern detection in market data
- Learning from historical decisions

The Challenge
How do we translate biological processes into mathematical operations?

This module tells the story of how scientists attempted this translation.

The fundamental question that started neural network research

Module 1 Roadmap

The Complete Journey (4 Modules)

1. **The Perceptron (Today)**
   - Single neuron foundations
   - 1943-1969 history
2. Multi-Layer Perceptrons
   - Stacking layers, activation functions
3. Training Neural Networks
   - Backpropagation, optimization
4. Applications in Finance
   - Stock prediction case study

Today's Module Structure

1. **Historical Context** (1943-1969)
   - McCulloch-Pitts, Hebb, Rosenblatt
2. **Biological Inspiration**
   - From neurons to mathematics
3. **The Perceptron**
   - Intuition, then math
4. **Learning Algorithm**
   - How it adjusts weights
5. **Limitations**
   - XOR problem, AI Winter

Your journey through neural network fundamentals

Learning Objectives

By the end of this module, you will be able to:

1. **Understand biological inspiration**
   - How real neurons inspired artificial ones
   - What we kept and what we simplified
2. **Master the perceptron model**
   - Inputs, weights, sum, activation
   - The decision-making unit
3. **Interpret decision boundaries**
   - Geometric meaning of weights
   - Linear separability concept
4. **Apply the learning algorithm**
   - Weight update rule
   - Convergence conditions
5. **Recognize limitations**
   - XOR problem
   - Why single layers are not enough

Finance Connection: Throughout, we'll use stock classification as our running example.

By the end of this module, you will be able to...

1943: The Mathematical Neuron

Warren McCulloch & Walter Pitts
In 1943, a neurophysiologist and a logician asked:

"Can we describe what neurons do using mathematics?"

Their paper: "A Logical Calculus of Ideas Immanent in Nervous Activity"

Key Insight:

- Neurons have binary states (fire or not)
- This is like TRUE/FALSE in logic
- Networks of neurons can compute any logical function

charts/mcculloch_pitts_diagram/mcculloch_pitts_dia

Warren McCulloch and Walter Pitts: "A Logical Calculus of Ideas Immanent in Nervous Activity"

What McCulloch & Pitts Proposed

The brain performs computation through:

1. **Binary Signals**
   - Neurons either fire (1) or don't (0)
   - Like bits in a computer
2. **Threshold Logic**
   - Sum of inputs exceeds threshold $\rightarrow$ fire
   - Otherwise $\rightarrow$ stay quiet
3. **Network Composition**
   - Complex behaviors from simple units
   - AND, OR, NOT gates from neurons

Logical Operations with Neurons

AND Gate (threshold $= 2$):

- Both inputs $= 1 \rightarrow$ output $= 1$
- Otherwise $\rightarrow$ output $= 0$

OR Gate (threshold $= 1$):

- Any input $= 1 \rightarrow$ output $= 1$
- All inputs $= 0 \rightarrow$ output $= 0$

Implication: If neurons compute logic, and computers compute logic, then we can build artificial brains!

If neurons compute, can we build artificial ones?

1949: Hebbian Learning

Donald Hebb's Insight
McCulloch-Pitts neurons were fixed. But how does the brain *learn*?

Hebb's Rule (1949):

> *"Neurons that fire together, wire together."*

In Plain Terms:

- If neuron A consistently activates neuron B
- The connection A $\rightarrow$ B grows stronger
- Repeated patterns reinforce pathways

Finance Analogy:
An analyst who repeatedly identifies winning stocks gains more influence in the committee.

`charts/hebb_learning_visualization/hebb_learning_v`

Donald Hebb: "Neurons that fire together, wire together"

1958: The Perceptron is Born

Frank Rosenblatt at Cornell
Combined McCulloch-Pitts neurons with Hebbian learning into a machine that could *learn from examples*.

The Perceptron:

- A single artificial neuron
- Adjustable connection weights
- Learns to classify patterns
- Implemented in hardware (Mark I)

Key Innovation:
Not just fixed logic gates, but a system that **learns** the right weights from training data.

charts/mark1_perceptron_diagram/mark1_perceptron_d

The New York Times Headline

July 8, 1958 - The New York Times

"New Navy Device Learns By Doing; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser"

The Promises Made:

- Machines that recognize faces
- Automatic translation of languages
- Systems that "perceive" like humans
- The Navy predicted: walking, talking, self-reproducing machines

The Reality:
The perceptron could classify simple patterns, but the gap between promise and capability was vast.

Lessons for Today

Sound Familiar?

- "AI will replace all jobs"
- "Machines will be smarter than humans by 20XX"
- "This changes everything"

Pattern:

1. Genuine breakthrough
2. Media amplification
3. Overpromising
4. Disappointment
5. "AI Winter"

History repeats...

"New Navy Device Learns By Doing" - The hype cycle begins

Timeline: The Early Years

charts/timeline_1943_1969/timeline_1943_1969.pdf

Discussion Question

"The perceptron was funded by the US Navy for military applications. How does funding source shape research direction? Are there parallels in modern AI development?"

Consider:

- Military vs. commercial vs. academic funding
- What problems get prioritized?
- Open vs. closed research
- Today: Tech giants fund most AI research
- Government initiatives (CHIPS Act, etc.)
- Startup ecosystem influence

Think-Pair-Share: 3 minutes

The Biological Neuron

Anatomy of a Real Neuron

1. **Dendrites** (Input)
   - Tree-like branches
   - Receive signals from other neurons
   - Thousands of connections
2. **Cell Body (Soma)** (Processing)
   - Integrates incoming signals
   - Contains the nucleus
   - Determines if neuron fires
3. **Axon** (Output)
   - Long fiber carrying output signal
   - Connects to other neurons
   - All-or-nothing signal

How It Works

1. Signals arrive at dendrites
2. Soma sums the inputs
3. If sum exceeds threshold: neuron **fires**
4. Action potential travels down axon
5. Signal reaches next neurons

Key Numbers:

- Human brain: $\sim$86 billion neurons
- Each neuron: $\sim$7,000 connections
- Total synapses: $\sim$100 trillion

Dendrites receive, soma processes, axon transmits

The Artificial Neuron

Mathematical Abstraction

1. **Inputs** $(x_1, x_2, \dots, x_n)$
   - Numerical values (features)
   - Replace dendrites
2. **Weights** $(w_1, w_2, \dots, w_n)$
   - Importance of each input
   - Replace synapse strength
3. **Weighted Sum**
   - $z = \sum_{i=1}^{n} w_i x_i + b$
   - Replace soma integration
4. **Activation Function**
   - $y = f(z)$
   - Replace firing decision

The Complete Model

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Components:

- x_i: Input features
- w_i: Learnable weights
- b: Bias (threshold adjustment)
- f: Activation function
- y: Output (prediction)

Key Point: The weights are what the network *learns*.

From biology to mathematics: the abstraction trade-off

Biological vs. Artificial: Side by Side

charts/biological_vs_artificial_neuron/biological_vs_artificial_neuron.pdf

Finance Analogy: The Analyst

A Financial Analyst as a Neuron

Biology	Finance
Dendrites	Market data feeds
Synapses	Data reliability weights
Soma	Analyst's judgment
Threshold	Conviction level
Axon	"Buy" recommendation

The Process:

1. Receive multiple data points
2. Weight by source quality
3. Aggregate into overall view
4. If conviction > threshold: recommend

Example: Stock Screening

Inputs (Data):

- x_1: P/E ratio $= 15$
- x_2: Revenue growth $= 20\%$
- x_3: Debt/Equity $= 0.5$

Weights (Importance):

- $w_1 = 0.3$ (value focus)
- $w_2 = 0.5$ (growth priority)
- $w_3 = -0.2$ (debt penalty)

Decision:

$$z = 0.3(15) + 0.5(20) - 0.2(0.5) = 14.4$$

If $z > 10$: **Buy**

Inputs (data) -¿ Weights (importance) -¿ Decision (output)

What We Gained from Abstraction

Benefits of Simplification

1. **Mathematical Tractability**
   - We can write equations
   - Analyze behavior formally
   - Prove theorems
2. **Computability**
   - Easy to implement in code
   - Fast computation
   - Scales to millions of units
3. **Trainability**
   - Can adjust weights systematically
   - Gradient-based optimization
   - Learn from data

What We Can Now Do

- Define learning algorithms
- Compute exact outputs
- Train on historical data
- Make predictions on new data
- Analyze decision boundaries

Scale Comparison:

	Brain	**GPU**
Operations/sec	10^{16}	10^{15}
Power	20W	300W
Training time	Years	Hours

Different trade-offs, different capabilities.

Simplification enables computation

What We Lost from Abstraction

Biological Complexity We Ignored

1. **Temporal Dynamics**
   - Real neurons have timing
   - Spike patterns carry information
   - We use static activations
2. **Structural Complexity**
   - Dendrites have local computation
   - Different neuron types
   - We use uniform units
3. **Neurochemistry**
   - Neurotransmitters vary
   - Modulatory systems
   - We use simple multiplication

Implications

What ANNs Cannot Do (Well):

- Energy efficiency of brain
- One-shot learning
- Continuous adaptation
- Common sense reasoning

The Trade-off:

Tractability $\uparrow$ Realism $\downarrow$

Artificial neurons are inspired by biology, not copies of it.

The brain does far more than our models capture

The Simplest Decision Maker

What is a Perceptron?
The simplest possible neural network:

- One artificial neuron
- Multiple inputs, one output
- Binary decision: Yes or No

Think of it as:

- A filter for data
- A simple classifier
- A linear decision maker

Finance Application:
Stock screener that outputs "Buy" or "Don't Buy" based on financial metrics.

Real-World Examples

Email Spam Filter:

- Inputs: word frequencies
- Output: spam or not spam

Loan Approval:

- Inputs: income, credit score, debt
- Output: approve or reject

Stock Screening:

- Inputs: P/E, momentum, volume
- Output: buy or pass

All these are binary classification problems that a perceptron can solve (if the data is linearly separable).

A single perceptron is a stock screening filter

charts/perceptron_architecture/perceptron_architecture.pdf

Finance Scenario: Buy or Sell?

Problem Setup

You want to build a simple stock screener:

- **Goal**: Decide Buy or Pass
- **Data**: Historical financial metrics
- **Method**: Perceptron classifier

Available Features:

1. P/E Ratio (valuation)
2. 6-month momentum (%)
3. Average daily volume
4. Debt-to-Equity ratio
5. Earnings surprise (%)

The Question

Given these features for a new stock, should we add it to our portfolio?

Example Stock:

- P/E $= 18$
- Momentum $= +12\%$
- Volume $= 2$M shares
- D/E $= 0.8$
- Surprise $= +5\%$

Traditional Approach:

Analyst manually weighs factors and decides.

Perceptron Approach:

Learn the weights from historical winners/losers.

Given financial indicators, should we buy this stock?

Inputs: The Raw Data

What Are Inputs?
Each input x_i is a numerical feature:

- A measurement
- A statistic
- A signal

In Finance:

- Price-based: returns, volatility
- Fundamental: P/E, ROE, debt ratios
- Technical: RSI, moving averages
- Sentiment: news scores, analyst ratings

Key Requirement:
All inputs must be **numerical**. Categorical data needs encoding.

Notation

For a stock with n features:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Example (n=3):

$$\mathbf{x} = \begin{pmatrix} 18 \\ 0.12 \\ 0.8 \end{pmatrix} = \begin{pmatrix} \text{P/E} \\ \text{Momentum} \\ \text{D/E} \end{pmatrix}$$

Note: Features often need **normalization** (covered in Module 3).

What data feeds into our decision?

What Are Weights?

Each weight w_i represents:

- Importance of input x_i
- Direction of influence
- Learned from data

Interpretation:

- $w_i > 0$: Higher x_i pushes toward "Buy"
- $w_i < 0$: Higher x_i pushes toward "Sell"
- $|w_i|$ large: Strong influence
- $|w_i|$ small: Weak influence

charts/weighted_sum_visualization/weighted_sum_vis

"Not all data is equally important" - weights encode importance

Discussion Question

"If you could only look at 3 metrics for a stock, which would you choose and why? How would you weight them?"

Consider:

Value Investor Might Choose:

- P/E ratio ($w = 0.5$)
- Book value ($w = 0.3$)
- Dividend yield ($w = 0.2$)

Growth Investor Might Choose:

- Revenue growth ($w = 0.5$)
- Momentum ($w = 0.3$)
- Market share ($w = 0.2$)

Key Insight: Different investors would assign different weights. The perceptron *learns* these weights from historical performance.

Think-Pair-Share: 3 minutes

The Weighted Sum: Adding Up Evidence

Computing the Weighted Sum

$$z = \sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

What This Means:

- Multiply each input by its weight
- Sum all the products
- Add the bias term b
- Result: a single "score"

The Bias b:

- Shifts the decision threshold
- Like a "base rate" or prior
- Can be thought of as $w_0 \cdot x_0$ where $x_0 = 1$

Worked Example

Inputs:

- $x_1 = 0.8$ (normalized P/E)
- $x_2 = 0.6$ (normalized momentum)

Weights:

- $w_1 = 0.5$
- $w_2 = 0.7$
- $b = -0.3$

Calculation:

$$\begin{aligned} z &= w_1 x_1 + w_2 x_2 + b \\ &= (0.5)(0.8) + (0.7)(0.6) + (-0.3) \\ &= 0.4 + 0.42 - 0.3 \\ &= \mathbf{0.52} \end{aligned}$$

Combine all weighted inputs into a single score

Analogy: The Voting Committee

The Perceptron as a Committee

Member	Vote	Weight	Contribution
P/E analyst	+1	0.5	+0.5
Momentum	+1	0.7	+0.7
Bias (skeptic)	-1	0.3	-0.3
Total			**+0.9**

If Total > 0: Committee recommends **Buy**

Key Insight:
The perceptron is just a weighted voting system where the weights are learned from data.

Why This Works

Traditional Committee:

- Human experts set weights
- Based on experience/intuition
- May have biases
- Hard to scale

Perceptron Committee:

- Weights learned from data
- Based on historical performance
- Consistent application
- Scales to any volume

Trade-off: Data-driven weights may not capture regime changes or rare events.

Some votes count more than others

The Threshold: Making the Call

The Activation Function
After computing z, we need a final decision.

Step Function:

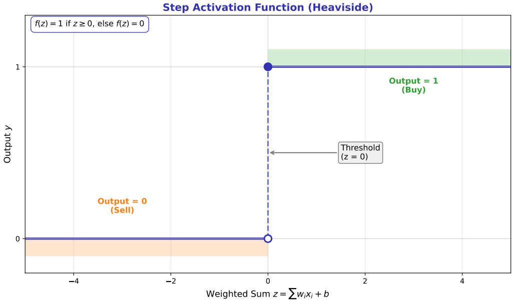
$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Interpretation:

- $z \geq 0$: Evidence favors "Buy" $\rightarrow$ output 1
- $z < 0$: Evidence favors "Sell" $\rightarrow$ output 0

Why Step Function?

- Binary classification needs binary output
- Mimics neuron firing (all-or-nothing)
- Simple to implement



Above threshold = Buy, Below threshold = Sell

The Complete Perceptron Flow

The Pipeline

1. **Input**: Receive features $\mathbf{x}$
2. **Weight**: Multiply by $\mathbf{w}$
3. **Sum**: Add all products + bias
4. **Activate**: Apply step function
5. **Output**: Return prediction

Compact Notation:

$$y = f(\mathbf{w}^T \mathbf{x} + b)$$

where $\mathbf{w}^T \mathbf{x} = \sum_i w_i x_i$

charts/perceptron_architecture/perceptron_architecture.pdf

Now Let's Formalize

What You Already Know

From the intuition section:

- Inputs are weighted
- Weights encode importance
- Sum is compared to threshold
- Output is binary

What's Next

- Precise mathematical notation
- Geometric interpretation
- Foundation for learning algorithm

Why Math Matters

Without Math:

- "The network kind of learns"
- "Adjust weights somehow"
- "It works, probably"

With Math:

- Precise learning rules
- Convergence guarantees
- Understanding of limitations

The next 8 slides formalize what you already understand intuitively.

You understand the intuition. Let's write it precisely.

The Perceptron Equation

Scalar Form

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where f is the step function:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Vector Form

$$y = f(\mathbf{w}^T \mathbf{x} + b)$$

where:

- $\mathbf{w} = (w_1, \ldots, w_n)^T$
- $\mathbf{x} = (x_1, \ldots, x_n)^T$

Alternative Notation

We can absorb the bias into weights:

$$\tilde{\mathbf{w}} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_n \end{pmatrix}, \quad \tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Then:

$$y = f(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}})$$

Note: This "bias trick" simplifies notation but they are equivalent.

The complete mathematical model

Unpacking the Mathematics

Term by Term

Symbol	Meaning
x_i	Input feature i
w_i	Weight for feature i
b	Bias (threshold shift)
z	Weighted sum (pre-activation)
f	Activation function
y	Output prediction
n	Number of features

Dimensions:

- $\mathbf{x} \in \mathbb{R}^n$
- $\mathbf{w} \in \mathbb{R}^n$
- $b, z, y \in \mathbb{R}$

What Gets Learned?

Learned (trainable):

- Weights $w_1, \ldots, w_n$
- Bias b

Fixed (architecture):

- Number of inputs n
- Activation function f

Given (data):

- Input values $x_1, \ldots, x_n$
- Target labels (for training)

Total Parameters: $n + 1$
(For a 3-feature perceptron: 4 parameters)

Each symbol has a meaning

The Bias Term

What Does Bias Do?

Without bias ($b = 0$):

$$z = \mathbf{w}^T \mathbf{x}$$

The decision boundary passes through origin.

With bias ($b \neq 0$):

$$z = \mathbf{w}^T \mathbf{x} + b$$

The decision boundary can be anywhere.

Interpretation:

- $b > 0$: Default toward "Buy"
- $b < 0$: Default toward "Sell"
- Like a prior belief

Finance Analogy

Without Bias:
"I have no opinion until I see data"

With Positive Bias:
"I'm generally bullish; you need to convince me to sell"

With Negative Bias:
"I'm skeptical by default; you need strong evidence to buy"

Key Point: Bias shifts the "bar" that evidence must clear. It's learned from data just like weights.

Bias shifts the decision threshold

The Step Activation Function

Formal Definition
The Heaviside step function:

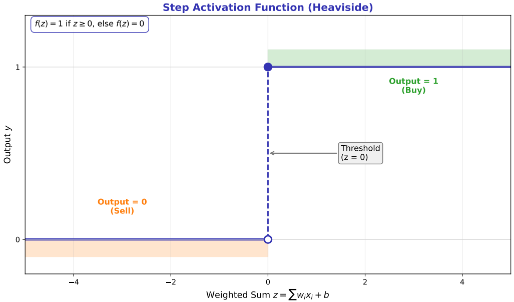
$$f(z) = \mathbf{1}_{z \geq 0} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Properties:

- Output $\in \{0, 1\}$
- Discontinuous at $z = 0$
- Not differentiable (problem for gradient-based learning!)

Variants:

- Sign function: outputs $\{-1, +1\}$
- Same idea, different labels



Preview: The non-differentiability of the step function is why we'll need smoother activations (sigmoid, ReLU) in later modules.

Binary output: yes or no

The Perceptron as a Hyperplane

The equation $\mathbf{w}^T\mathbf{x} + b = 0$ defines a hyperplane:

- In 2D: a line
- In 3D: a plane
- In nD: a hyperplane

Regions:

- $\mathbf{w}^T\mathbf{x} + b > 0$: Class 1 (Buy)
- $\mathbf{w}^T\mathbf{x} + b < 0$: Class 0 (Sell)
- $\mathbf{w}^T\mathbf{x} + b = 0$: Decision boundary

Weight Vector Direction:

$\mathbf{w}$ is perpendicular to the decision boundary, pointing toward the positive class.

charts/decision_boundary_2d/decision_boundary_2d.pdf

Finance Example: Classifying Stocks

Two-Feature Stock Screener

Features:

- x_1: P/E ratio (normalized)
- x_2: 6-month momentum (%)

Classes:

- Green: Outperformed (Buy)
- Red: Underperformed (Sell)

Goal:

Find w_1, w_2, b such that:

$$w_1 \cdot \mathrm{P/E} + w_2 \cdot \mathrm{Momentum} + b = 0$$

separates the classes.

charts/stock_features_scatter/stock_features_scatter.pd

The Decision Boundary Formula

In 2D: The Line Equation

From $w_1 x_1 + w_2 x_2 + b = 0$:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

This is a line with:

- Slope: $-\frac{w_1}{w_2}$
- Intercept: $-\frac{b}{w_2}$

Example:

If $w_1 = 2, w_2 = 1, b = -3$:

$$x_2 = -2x_1 + 3$$

Stocks above this line: Buy
Stocks below this line: Sell

charts/finance_decision_boundary/finance_decision_b

The line that separates buy from sell

How Does the Perceptron Learn?

The Learning Problem
Given:

- Training data: $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$
- Each $\mathbf{x}^{(i)}$: feature vector
- Each $y^{(i)} \in \{0, 1\}$: true label

Find:

- Weights $\mathbf{w}$
- Bias b
- Such that predictions match labels

The Approach:
Start with random weights, then iteratively adjust based on mistakes.

The Core Idea

If prediction is correct:
Do nothing. Weights are fine.

If prediction is wrong:
Adjust weights to make this example more likely to be correct next time.

Repeat:
Keep cycling through training data until no mistakes (or convergence).

Key Insight: Learning = adjusting weights based on errors.

Learning = adjusting weights based on mistakes

Two Types of Errors

False Negative ($\hat{y} = 0$, $y = 1$):

- Predicted Sell, should be Buy
- The score z was too low
- Need to *increase* score for this $\mathbf{x}$
- Solution: Add $\mathbf{x}$ to $\mathbf{w}$

False Positive ($\hat{y} = 1$, $y = 0$):

- Predicted Buy, should be Sell
- The score z was too high
- Need to *decrease* score for this $\mathbf{x}$
- Solution: Subtract $\mathbf{x}$ from $\mathbf{w}$

Visual Intuition

Before update:
Point is on wrong side of boundary.

After update:
Boundary moves to include the point on the correct side.

The Update Rule:

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + (y - \hat{y}) \cdot \mathbf{x}$$

Check:

- If $y = 1, \hat{y} = 0$: add $\mathbf{x}$
- If $y = 0, \hat{y} = 1$: subtract $\mathbf{x}$
- If $y = \hat{y}$: no change

Each mistake is a learning opportunity

Why Adding x Works

For a false negative (missed Buy):

- Current: $\mathbf{w}^T\mathbf{x} + b < 0$
- After adding $\mathbf{x}$ to $\mathbf{w}$:
- New score: $(\mathbf{w} + \mathbf{x})^T\mathbf{x} + b$
- $= \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x} + b$
- $= \mathbf{w}^T\mathbf{x} + \|\mathbf{x}\|^2 + b$

Since $\|\mathbf{x}\|^2 > 0$, the new score is higher!

Geometrically:

Adding $\mathbf{x}$ rotates the decision boundary toward classifying $\mathbf{x}$ correctly.

charts/perceptron_learning_animation/perceptron_learni

The Update Equations

For each training example $(\mathbf{x}, y)$:

Weight update:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y - \hat{y})\mathbf{x}$$

Bias update:

$$b \leftarrow b + \eta(y - \hat{y})$$

where:

- $\eta > 0$ is the learning rate
- $\hat{y} = f(\mathbf{w}^T\mathbf{x} + b)$ is prediction
- y is true label

The Complete Algorithm

1. Initialize $\mathbf{w} = \mathbf{0}$, $b = 0$
2. **repeat**:
   a. For each $(\mathbf{x}^{(i)}, y^{(i)})$ in training set:
   b. Compute $\hat{y}^{(i)} = f(\mathbf{w}^T\mathbf{x}^{(i)} + b)$
   c. If $\hat{y}^{(i)} \neq y^{(i)}$:
      $\mathbf{w} \leftarrow \mathbf{w} + \eta(y^{(i)} - \hat{y}^{(i)})\mathbf{x}^{(i)}$
      $b \leftarrow b + \eta(y^{(i)} - \hat{y}^{(i)})$
3. **until** no errors (or max iterations)

The mathematical update rule

The Learning Rate

What is η?
The learning rate controls step size:

- How much weights change per update
- Typical values: 0.01 to 1.0
- For perceptron: often $\eta = 1$

Effects:
η **too small:**

- Very slow learning
- Many iterations needed
- But stable

η **too large:**

- May overshoot
- Oscillate around solution
- But faster initially

For the Perceptron

Good news:
For linearly separable data, the perceptron converges regardless of $\eta > 0$.

Why?
The convergence theorem (next slides) guarantees finding a solution if one exists.

In Practice:
$\eta = 1$ is common for perceptron. Learning rate matters more for:

- Gradient descent (Module 3)
- Non-separable data
- Multi-layer networks

Step size matters: too big or too small both cause problems

Worked Example: Stock Classification

Setup
Two stocks, two features:

- $\mathbf{x}^{(1)} = (0.5, 0.8)$, $y^{(1)} = 1$ (Buy)
- $\mathbf{x}^{(2)} = (0.2, 0.3)$, $y^{(2)} = 0$ (Sell)

Initialize: $\mathbf{w} = (0, 0)$, $b = 0$, $\eta = 1$

Iteration 1: Example 1

- $z = 0 \cdot 0.5 + 0 \cdot 0.8 + 0 = 0$
- $\hat{y} = f(0) = 1$ (threshold at 0)
- $y = 1$, correct! No update.

Iteration 1: Example 2

- $z = 0$, $\hat{y} = 1$
- $y = 0$, wrong!
- $\mathbf{w} \leftarrow (0, 0) + 1(0 - 1)(0.2, 0.3) = (-0.2, -0.3)$
- $b \leftarrow 0 + 1(0 - 1) = -1$

Iteration 2: Example 1

- $z = -0.2(0.5) - 0.3(0.8) - 1 = -1.34$
- $\hat{y} = 0$
- $y = 1$, wrong!
- $\mathbf{w} \leftarrow (-0.2, -0.3) + (0.5, 0.8) = (0.3, 0.5)$
- $b \leftarrow -1 + 1 = 0$

Iteration 2: Example 2

- $z = 0.3(0.2) + 0.5(0.3) + 0 = 0.21$
- $\hat{y} = 1$, $y = 0$, wrong!
- $\mathbf{w} \leftarrow (0.3, 0.5) - (0.2, 0.3) = (0.1, 0.2)$
- $b \leftarrow 0 - 1 = -1$

Continue until convergence...

Following the math with real numbers

Convergence: Does It Always Work?

The Perceptron Convergence Theorem
Theorem (Rosenblatt, 1962):
If the training data is **linearly separable**, the perceptron learning algorithm will find a separating hyperplane in a **finite** number of updates.

Key Conditions:

- Data must be linearly separable
- Learning rate $\eta > 0$
- Cycling through all examples

Bound on Updates:

$$\text{mistakes} \leq \frac{R^2}{\gamma^2}$$

where R = max norm, γ = margin

charts/convergence_plot/convergence_plot.pdf

Discussion Question

"What happens when data isn't linearly separable in financial markets? Can you think of examples?"

Consider:

Examples of Non-Separable Data:

- High P/E growth stocks AND low P/E value stocks both outperform
- Medium-risk investments underperform both conservative and aggressive
- "Buy the rumor, sell the news" patterns

What Happens to the Perceptron?

- Never converges
- Oscillates forever
- Best we can do: minimize errors
- Need something more powerful...

Foreshadowing: This is exactly why we need **multi-layer** networks (Module 2).

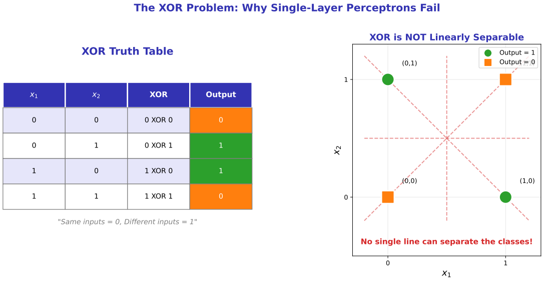
Think-Pair-Share: 3 minutes

The Exclusive OR Function

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

In Words:
Output is 1 if inputs are *different*, 0 if inputs are *same*.

The Challenge:
Try to draw a single line that separates the 1s from the 0s...



Some patterns cannot be separated by a single line

Why XOR Cannot Be Solved

Geometric Impossibility

Perceptron decision boundary:
$w_1x_1 + w_2x_2 + b = 0$
This is always a **straight line**.

XOR requires:
A boundary that curves or has multiple segments.

Linear vs Non-Linear
Linearly Separable:

- AND, OR, NAND, NOR
- One line can separate

Not Linearly Separable:

- XOR, XNOR
- No single line works

charts/linear_vs_nonlinear_patterns/linear_vs_nonlinear

1969: The Critique That Changed Everything

Minsky and Papert's Book
"Perceptrons: An Introduction to Computational Geometry" (1969)

Key Arguments:

1. Single-layer perceptrons cannot compute XOR
2. Many important functions are non-linear
3. No known training algorithm for multi-layer networks
4. Scaling limitations

The Impact:
The book was rigorous and influential. It convinced funding agencies that neural networks were a dead end.

The Controversy

Valid Points:

- Single layers *are* limited
- XOR problem is real
- No training algorithm existed (then)

Overstated Points:

- "Neural networks can't work"
- Implied multi-layer networks wouldn't help
- Discouraged research for 15+ years

Lesson: Valid criticism of current methods shouldn't stop research into future improvements.

Marvin Minsky and Seymour Papert: "Perceptrons" book

The First AI Winter Begins

The Collapse
After 1969:

- Funding dried up
- Researchers left the field
- "Neural networks don't work"
- Symbolic AI took over

Duration: 1969 to $\sim$1982

What Survived:

- A few dedicated researchers
- Theoretical work continued quietly
- Hopfield networks (1982)
- Backpropagation (1986)

charts/ai_winter_timeline/ai_winter_timeline.pdf

Module 1: Key Takeaways

What We Learned

1. **Historical Foundation**
   - McCulloch-Pitts (1943): neurons compute
   - Hebb (1949): learning strengthens connections
   - Rosenblatt (1958): perceptron learns
2. **The Perceptron Model**
   - Weighted sum + threshold
   - Linear decision boundary
   - Learns from mistakes
3. **Limitations**
   - Only linearly separable problems
   - XOR is impossible
   - Led to AI Winter

`charts/module1_summary_diagram/module1_summary_diagram`

"What if we stack multiple perceptrons?"

The Problem We Face
Single perceptrons can only solve linearly separable problems. Real financial data is rarely that simple.

The Solution Preview:

- Add "hidden" layers
- Non-linear activation functions
- Multi-Layer Perceptrons (MLPs)

Coming in Module 2:

- How XOR gets solved
- MLP architecture
- Activation functions (sigmoid, ReLU)
- Universal Approximation Theorem
- Loss functions

Spoiler: Adding just one hidden layer changes everything.

Mathematical details for this module: See Appendix A (Perceptron Convergence Proof)

Next: Solving XOR with Multi-Layer Perceptrons