Lesson 33: Perceptron

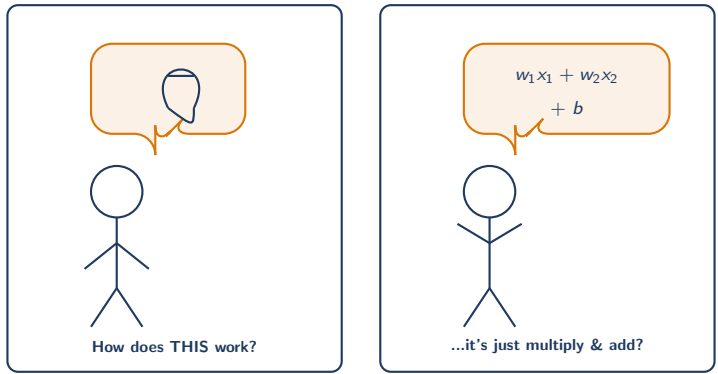
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

The Quest for Artificial Intelligence



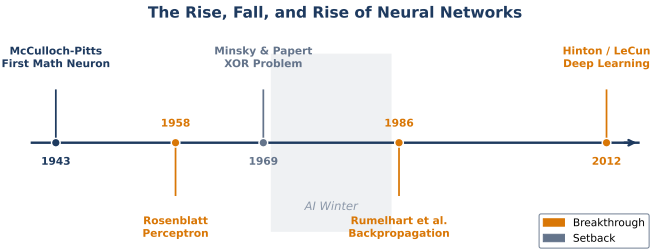
In 1943, McCulloch & Pitts asked: Can we build a mathematical brain?

One equation started a revolution – and three boom-bust cycles

A Brief History of Neural Networks

Three boom-bust cycles in 70 years:

- **1943–1969:** Birth and first death (XOR crisis)
- **1986–2006:** Backpropagation revival, then stagnation
- **2012–now:** Deep learning revolution (GPU + data + scale)

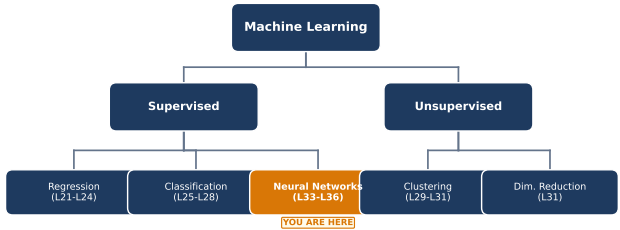


Pattern: each "winter" ended when compute caught up with theory

Where Deep Learning Fits

Neural networks still do regression AND classification

- Same goals as L21–L28, but with **learned features**
- Traditional ML: you engineer features; DL: the model learns them
- The perceptron is the simplest possible neural network



Deep learning = ML with automatic feature extraction via stacked layers

Learning Objectives

The Problem: Traditional ML uses hand-crafted features. Can a model **learn its own**? Understanding the perceptron is the first step toward deep learning.

After this lesson, you will be able to:

1. Explain the biological inspiration and its mathematical abstraction
2. Build and train a single perceptron from scratch
3. Recognize the linear separability limitation (and why XOR broke everything)
4. Connect the perceptron to multi-layer networks that overcome it

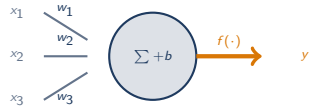
---

Finance Application: Foundation for deep learning models in quantitative finance

From Biology to Math: The Neuron

Biology

Math

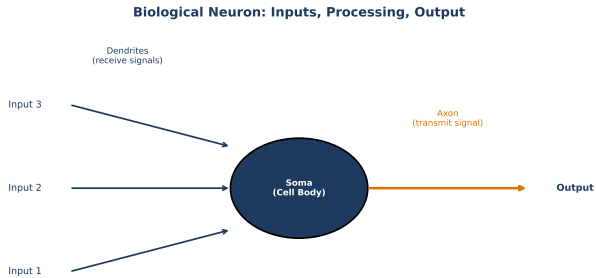


Biology	Math
Dendrites receive signals	Inputs $x_1, x_2, \ldots$
Synapse strength	Weights $w_1, w_2, \ldots$
Soma sums and thresholds	$z = \sum w_i x_i + b$
Axon fires or not	$y = f(z)$

We simplify radically – keep only the essential math

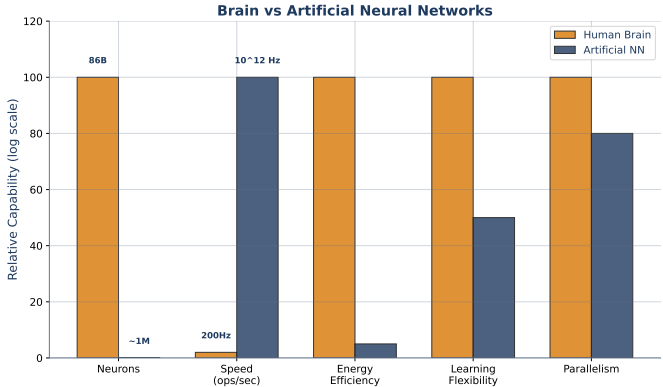
The Biological Inspiration

- Receives signals, sums them, fires if threshold exceeded
- We keep this idea but replace biology with linear algebra



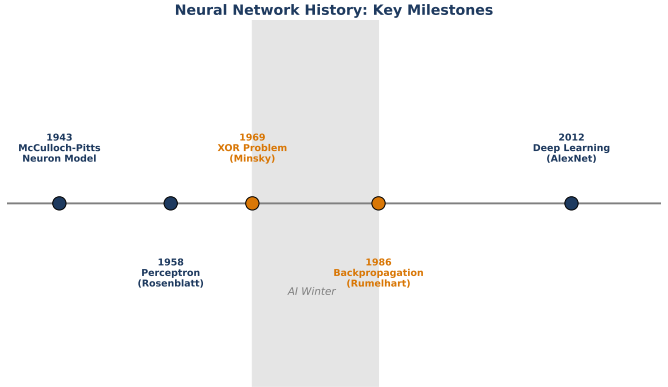
Perceptron: simplified mathematical model of biological neuron

Brain vs Artificial Neural Networks



ANNs are inspired by but not identical to biological neural networks

Neural Network History (Detailed)



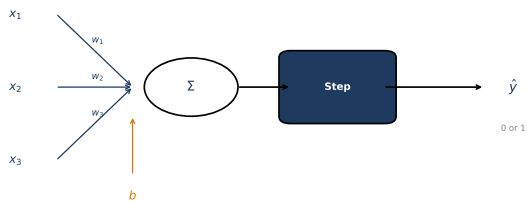
From 1943 McCulloch-Pitts to modern deep learning revolution

The Perceptron: One Simple Equation

Core: $y = \text{step}\left(\sum_{i=1}^{n} w_i x_i + b\right)$

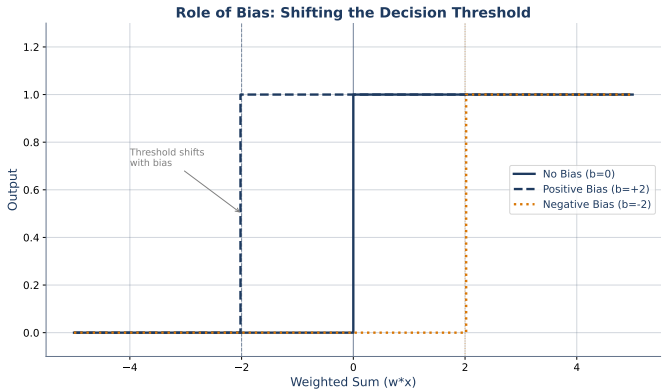
- Identical to logistic regression (L25) with step instead of sigmoid
- Weights w_i control importance; bias b shifts the threshold

Perceptron: Weighted Sum + Step Function



Single perceptron = logistic regression reframed as a "neuron"

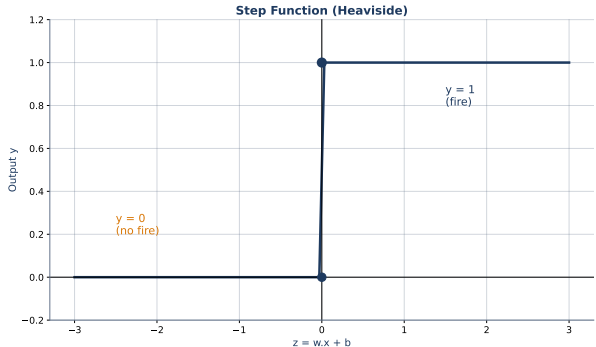
Role of Bias



Bias shifts the decision boundary – controls activation threshold

Step Function: The Original Activation

- Output 1 if weighted sum > 0, else 0
- Gradient is **zero everywhere** – cannot learn via gradient descent

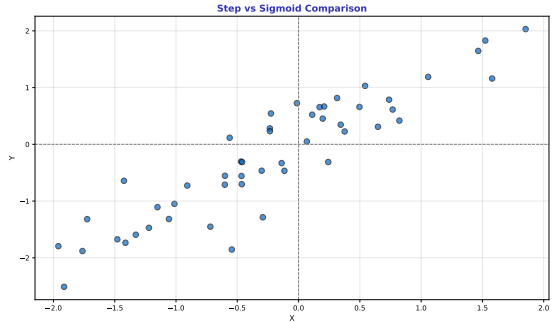


Original perceptron: output 1 if weighted sum > 0, else 0

Why Sigmoid Is Better

Problem: Step function gradient is zero everywhere.

- **Smooth:** $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- **Range** $[0, 1]$**:** interpretable as probability
- **Non-zero gradient:** enables backpropagation (L35)

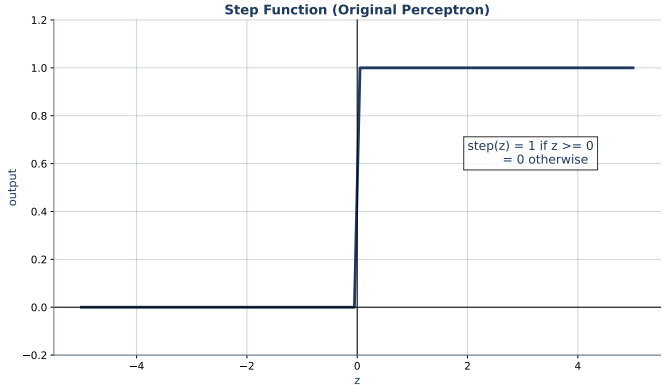


Sigmoid enables gradient-based learning; step does not

Step Function (Original)

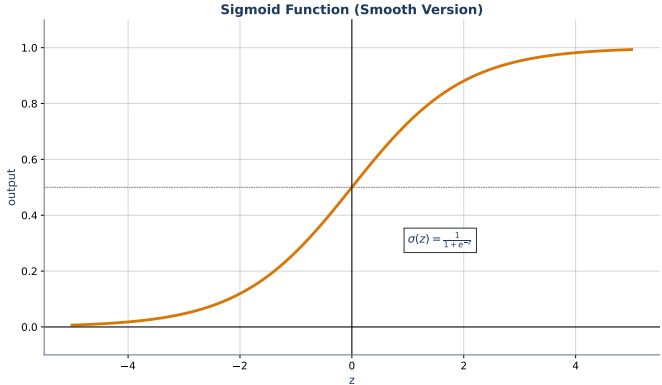
Binary Activation

- Output 1 if weighted sum > 0, else 0
- Simple but not differentiable at threshold



Step function: original perceptron activation

Sigmoid Function (Smooth)

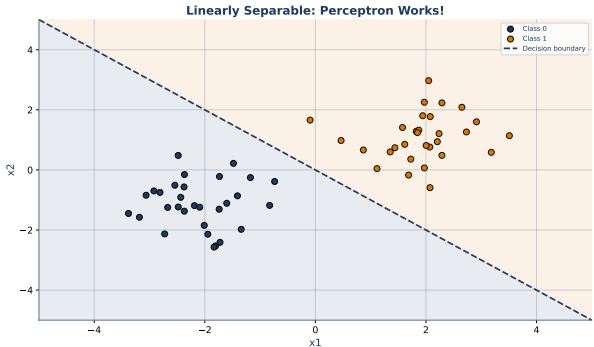


Sigmoid: smooth, differentiable, outputs probabilities (0–1)

What Can a Perceptron Learn?

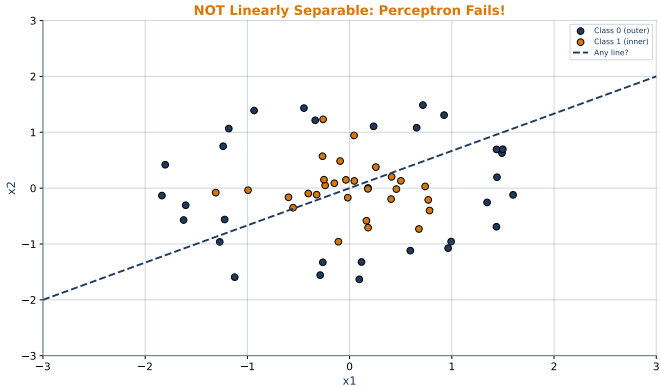
Linear separability is the hard constraint

- Single perceptron = one straight line (hyperplane in nD)
- Same limitation as logistic regression – because it **is** logistic regression



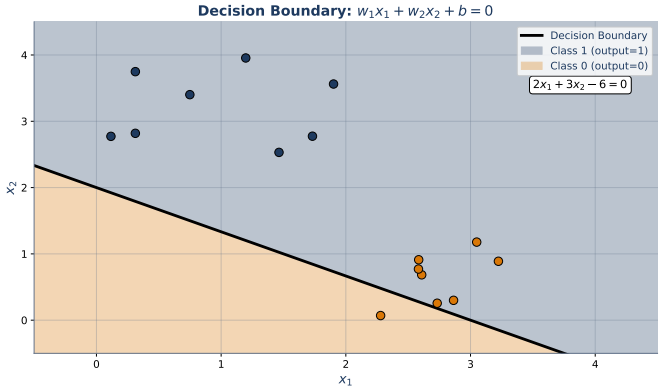
Perceptron = linear classifier. Can separate AND, OR but not XOR.

Not Linearly Separable



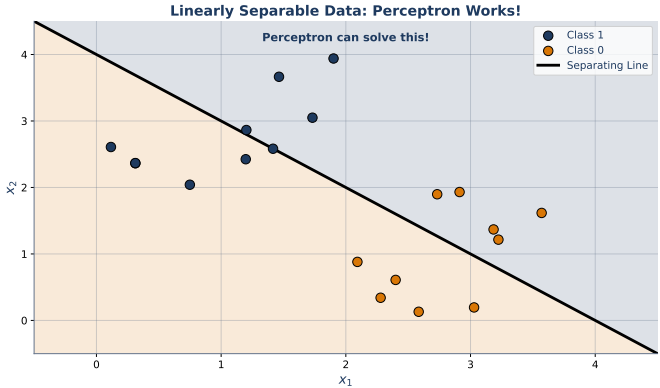
No single line can separate these classes – perceptron fails

Decision Boundary Math



Boundary: $w_1x_1 + w_2x_2 + b = 0$ – a line in 2D, hyperplane in nD

Linear Separability Concept



Fundamental concept: can classes be separated by a hyperplane?

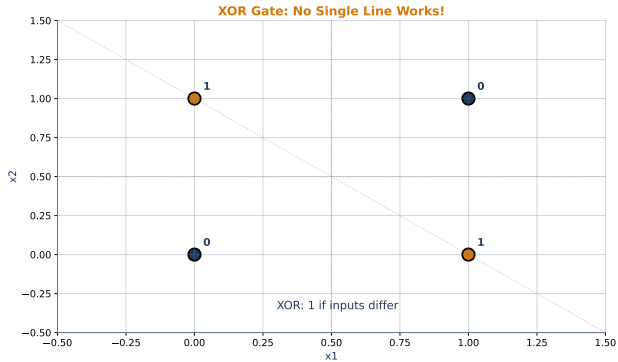
The XOR Problem: The Crisis

1969: Minsky & Papert PROVED a single perceptron CANNOT solve XOR.

XOR Truth Table:

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

No line divides the 1s from 0s.



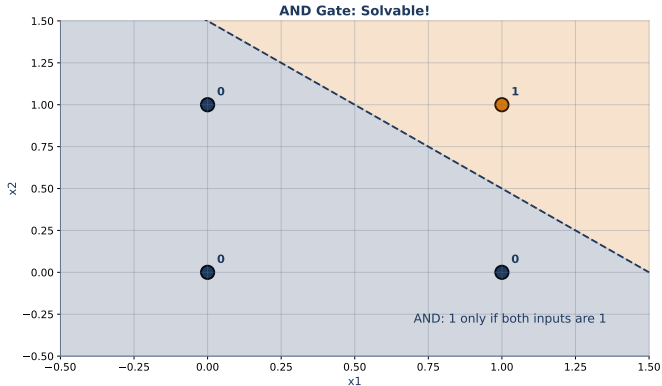
Consequence: Funding dried up. The **AI Winter** began (1970s–1980s).

Lesson: never dismiss an idea because of limitations in its simplest form

AND Gate (Solvable)

Logic Gates as Classification

- AND: output 1 only if both inputs are 1
- Linearly separable – perceptron can learn it



AND gate: single perceptron works perfectly

The Perceptron Learning Rule

Algorithm:

1. **Initialize:** $\mathbf{w} = \mathbf{0}$, $b = 0$
2. **For each sample** (x, y_{true}):
   - Predict: $\hat{y} = \text{step}(\mathbf{w} \cdot \mathbf{x} + b)$
   - If correct ($\hat{y} = y_{\text{true}}$): **do nothing**
   - If wrong: $\mathbf{w} \leftarrow \mathbf{w} + \eta\,(y_{\text{true}} - \hat{y})\,\mathbf{x}$
3. **Repeat** until no errors or max epochs

Intuition: Push the decision boundary toward misclassified points.

Learning rate η: Typically 1.0 for perceptron (or 0.1 for finer steps).

Simple rule: only update weights when the prediction is WRONG

Learning Rule: Full Derivation

Why does the update rule work?

Consider a misclassified point where $y_{\text{true}} = 1$ but $\hat{y} = 0$:

- Error: $e = y_{\text{true}} - \hat{y} = 1 - 0 = +1$
- Update: $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \eta \cdot (+1) \cdot \mathbf{x}$
- Effect: $\mathbf{w} \cdot \mathbf{x}$ **increases**, making it more likely > 0 next time

Now consider $y_{\text{true}} = 0$ but $\hat{y} = 1$:

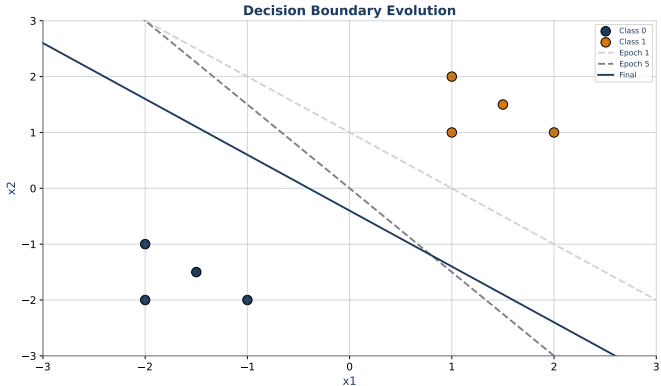
- Error: $e = 0 - 1 = -1$
- Update: $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \eta \cdot (-1) \cdot \mathbf{x}$
- Effect: $\mathbf{w} \cdot \mathbf{x}$ **decreases**, pushing output below threshold

Bias update: $b \leftarrow b + \eta\,(y_{\text{true}} - \hat{y})$ shifts the threshold.

Each update moves the boundary to correctly classify the misclassified point

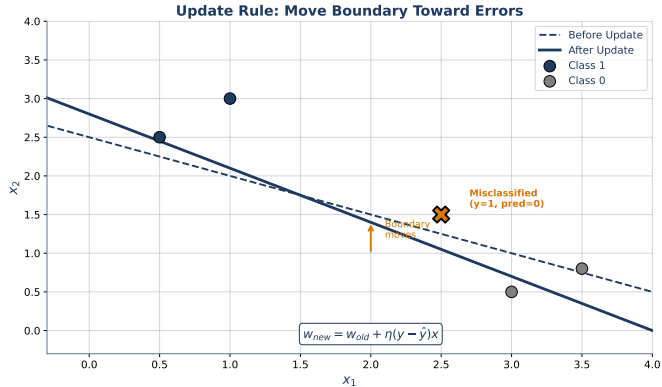
Watching the Boundary Learn

Decision boundary evolves over training epochs:



Watch the boundary adjust with each misclassified point

Update Rule Intuition

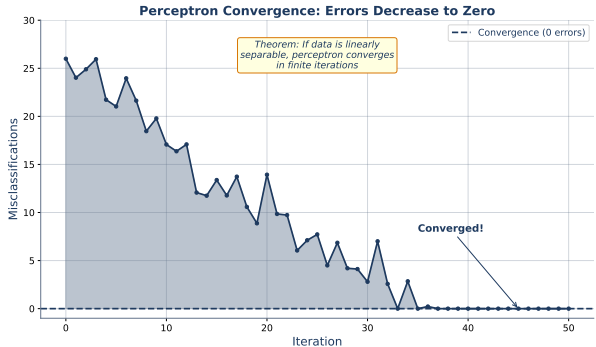


Push boundary toward misclassified points

Convergence: When Does Learning Stop?

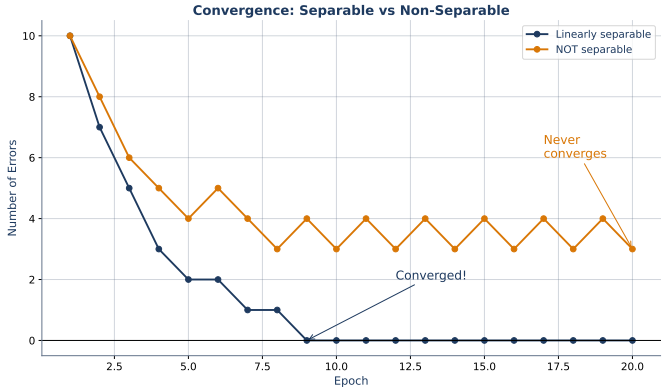
Convergence Theorem (Novikov, 1962):

- Linearly separable → **always** converges
- Not separable → oscillates forever



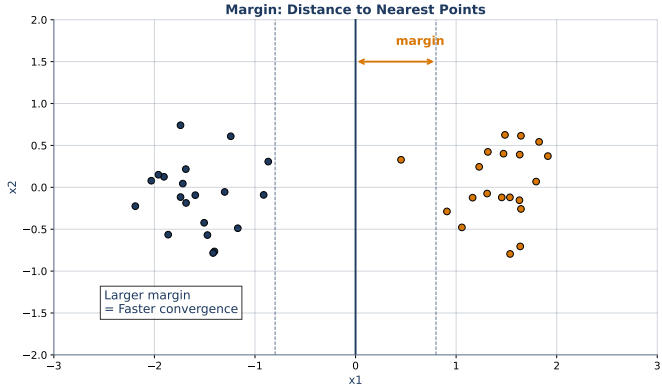
Guaranteed convergence for separable problems – but most real data is not separable

Convergence: Separable vs Non-Separable



Separable: converges. Non-separable: oscillates indefinitely

Margin (Distance to Boundary)



Larger margin = faster convergence. SVM maximizes margin.

Checkpoint: Test Your Understanding

Q1: Can a single perceptron solve XOR?

Q2: What does the learning rule do when the prediction is **correct**?

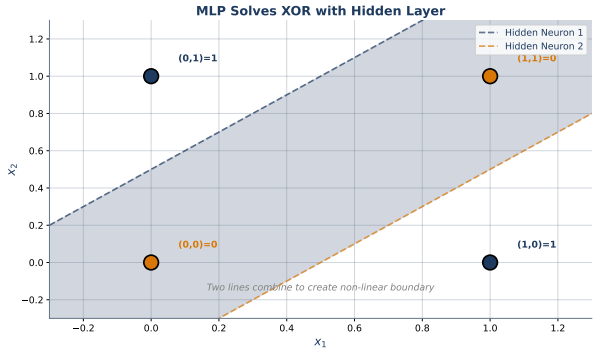
Q3: A perceptron with sigmoid activation is equivalent to which sklearn model?

Answers: Q1: No (not linearly separable). Q2: Nothing (no update). Q3: LogisticRegression.

The Solution: Add Hidden Layers

What Minsky and Papert missed:

- Multi-layer networks **can** solve XOR
- Each hidden neuron learns a different boundary; combined: non-linear regions



Solution: add hidden layers (multi-layer perceptron) for non-linear boundaries

sklearn: Perceptron in Practice

Using sklearn's Perceptron:

- `from sklearn.linear_model import Perceptron`
- `model = Perceptron(max_iter=1000, random_state=42)`
- `model.fit(X_train, y_train)`
- `y_pred = model.predict(X_test)`

Access learned parameters:

- Weights: `model.coef_`
- Bias: `model.intercept_`

Note: The perceptron is a **building block**, not a production model. Use `LogisticRegression` or `MLPClassifier` for real tasks.

Same sklearn API pattern: fit, predict, score

Python Implementation: From Scratch

Perceptron Class Structure:

- **__init__:** Set `learning_rate` and `n_epochs`
- **fit(X, y):** Initialize `self.w = np.zeros(n_features)`, `self.b = 0`
- **Training loop:** For each sample, if `error = y - y_pred != 0`:
  - `self.w += lr * error * x`
  - `self.b += lr * error`
- **predict_one(x):** `return 1 if np.dot(x, self.w) + self.b >= 0 else 0`
- **predict(X):** Apply `predict_one` to all samples

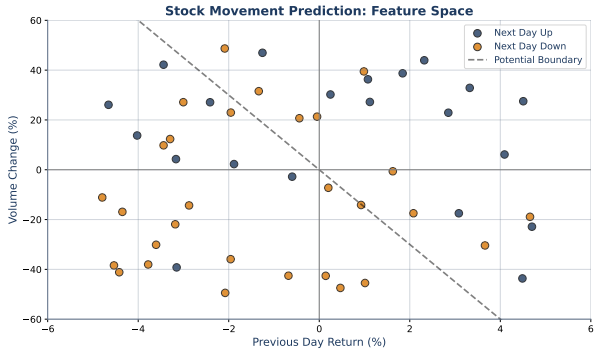
Usage:

```
p = Perceptron(lr=0.1, epochs=100)
p.fit(X_train, y_train)
predictions = p.predict(X_test)
```

Simple loop: predict, check error, update weights if wrong

Finance: Market Direction Prediction

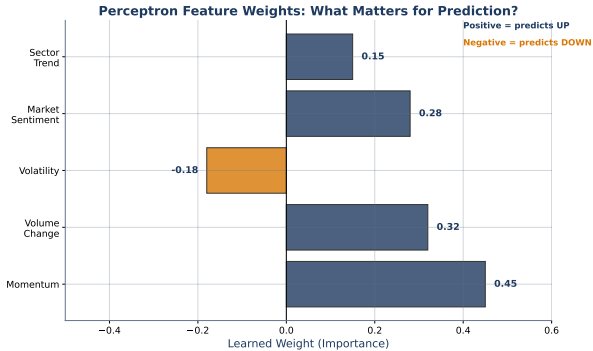
- Inputs: momentum and volatility features
- Output: up (1) or down (0) next-day prediction



Reality: financial patterns are rarely linearly separable

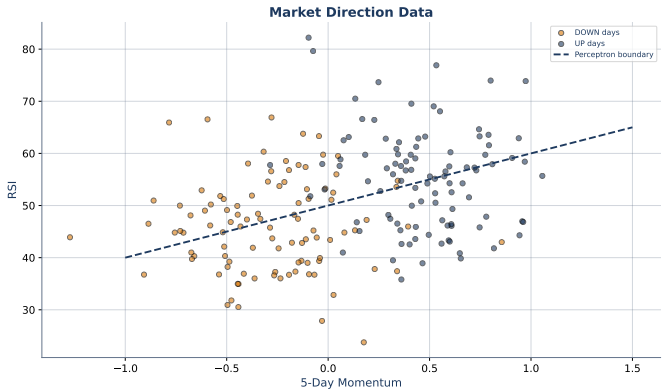
Results: Why Perceptrons Aren't Enough

- Linear boundary captures some signal, but accuracy is limited
- Non-linear regimes need deeper architectures



This motivates everything in L34–L36: MLPs, activations, backpropagation

Market Direction Data



Feature space visualization of market direction problem

Hands-On Exercise (25 min)

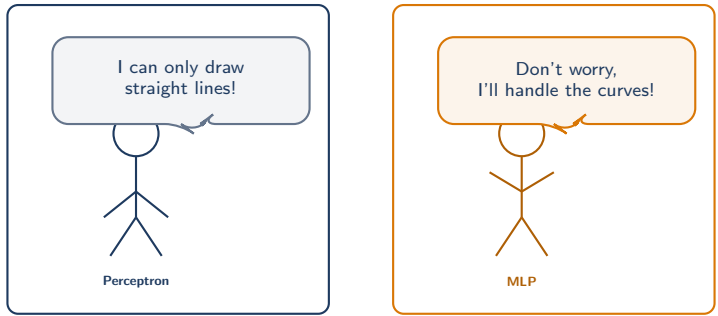
Task: Implement Perceptron from Scratch

1. Create synthetic linearly separable 2D data
2. Implement perceptron learning rule in Python (no sklearn)
3. Train and plot decision boundary at each epoch
4. Try on XOR data – observe failure to converge
5. Compare your implementation to sklearn's `Perceptron`

Deliverable: Boundary evolution animation + XOR failure plot.

Extension: modify learning rate – how does convergence speed change?

The Artificial Neuron: What We Learned



One neuron = one line. Stack neurons into layers = learn ANY pattern.

Key Takeaways

What you should remember:

1. **Perceptron = logistic regression reframed** as a "neuron"
2. **Linear separability** is the hard limit – one perceptron draws one line
3. **XOR broke everything** $\rightarrow$ AI Winter $\rightarrow$ but MLPs saved the day
4. This single neuron is the **foundation for all deep learning**

Connection to previous lessons:

- L25 (Logistic Regression): same math, different name
- L27 (Classification Metrics): same evaluation applies
- L28 (Class Imbalance): same challenges apply

Memory: perceptron = single neuron = linear boundary. XOR needs hidden layers.

Preview: Stacking Legos

One neuron draws a line.
Stack neurons into layers → learn **any** pattern.

Coming next in L34–L36:

- **L34:** MLPs and Activation Functions – ReLU, tanh, and why they matter
- **L35:** Backpropagation – how networks actually learn
- **L36:** Overfitting Prevention – dropout, early stopping, regularization

The progression:

Single Neuron (L33) → Stacked Layers (L34) → Learning Algorithm (L35) → Robust Training (L36)

Next lesson: L34 MLPs and Activations – adding hidden layers changes everything