

Lesson 33: Perceptron

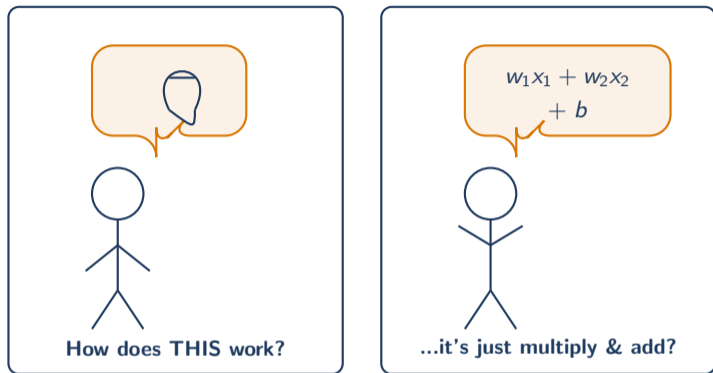
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

The Quest for Artificial Intelligence



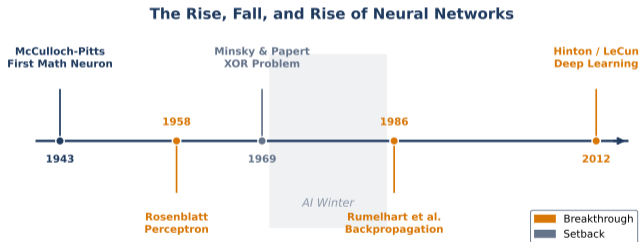
In 1943, McCulloch & Pitts asked: Can we build a mathematical brain?

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

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

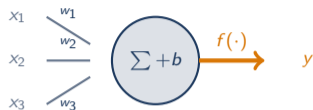
Finance Application: Foundation for deep learning models in quantitative finance

From Biology to Math: The Neuron

Biology



Math



Biology

Dendrites receive signals
Synapse strength
Soma sums and thresholds
Axon fires or not

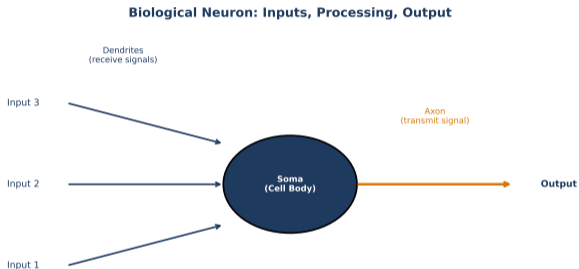
Math

Inputs x_1, x_2, \dots
Weights w_1, w_2, \dots
 $z = \sum w_i x_i + b$
 $y = f(z)$

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



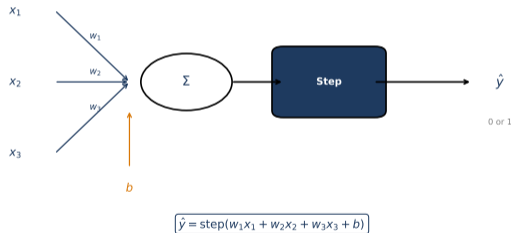
Perceptron: simplified mathematical model of biological neuron

The Perceptron: One Simple Equation

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

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

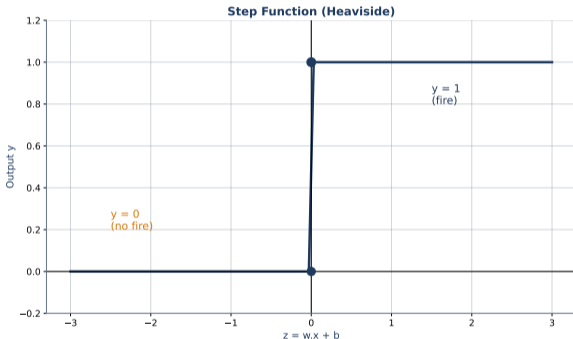
Perceptron: Weighted Sum + Step Function



Single perceptron = logistic regression reframed as a “neuron”

Step Function: The Original Activation

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

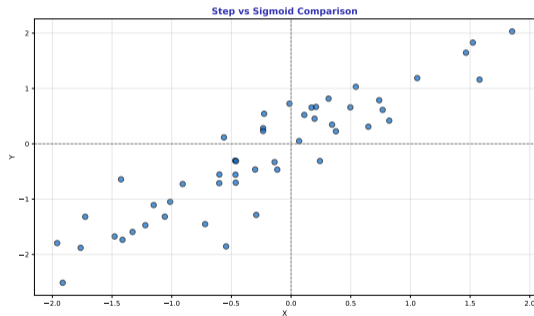


Original perceptron: output 1 if weighted sum > 0 , else 0

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)

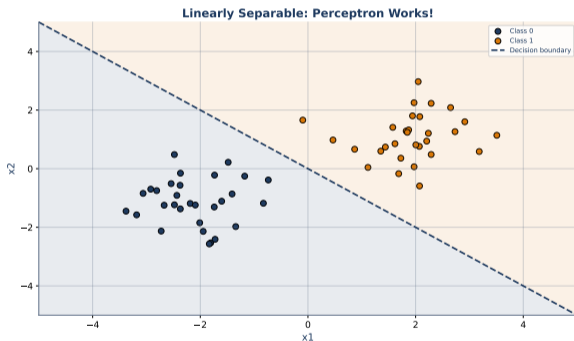


Sigmoid enables gradient-based learning; step does not

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.

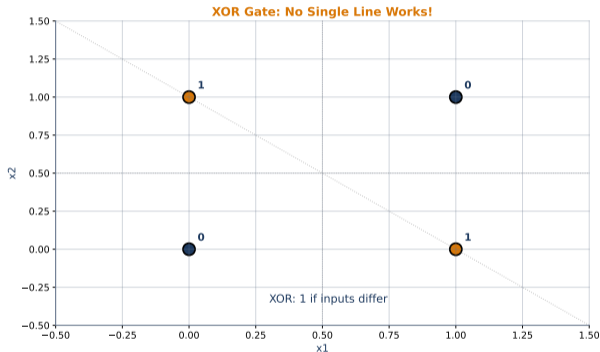
The XOR Problem: The Crisis

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

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

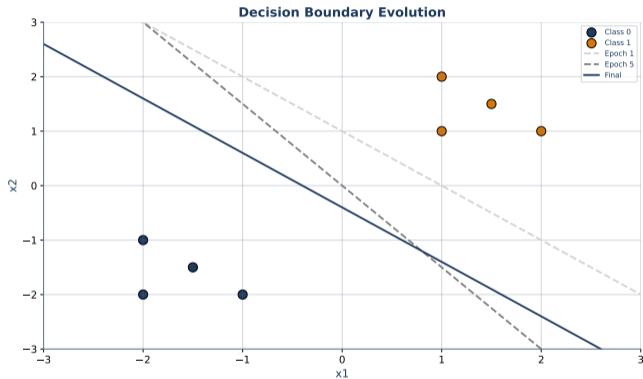
Intuition: Push the decision boundary toward misclassified points.

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

Simple rule: only update weights when the prediction is **WRONG**

Watching the Boundary Learn

Decision boundary evolves over training epochs:

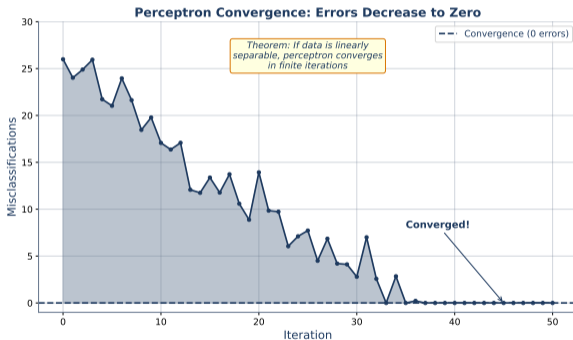


Watch the boundary adjust with each misclassified point

Convergence: When Does Learning Stop?

Convergence Theorem (Novikov, 1962):

- Linearly separable \rightarrow **always** converges
- Not separable \rightarrow oscillates forever



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

Checkpoint: Test Your Understanding

Q1: Can a single perceptron solve XOR?

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

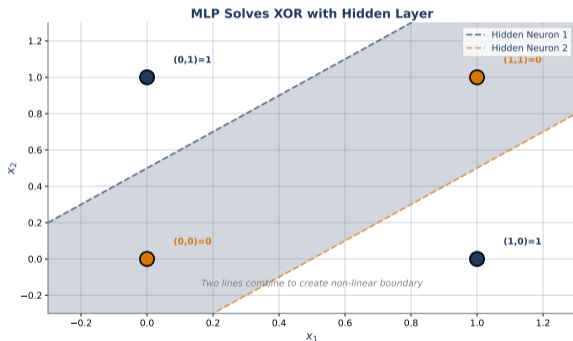
Q3: A perceptron with sigmoid activation is equivalent to which sklearn model?

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:

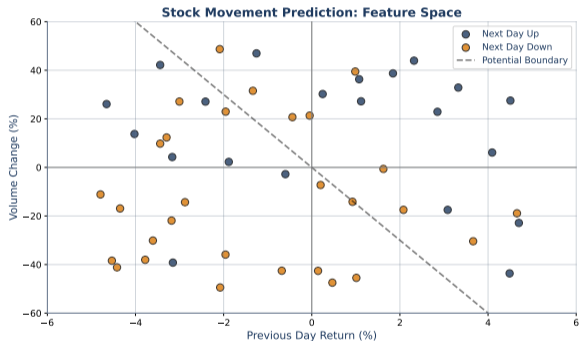
- Weights: `model.coef_`
- Bias: `model.intercept_`

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`

Finance: Market Direction Prediction

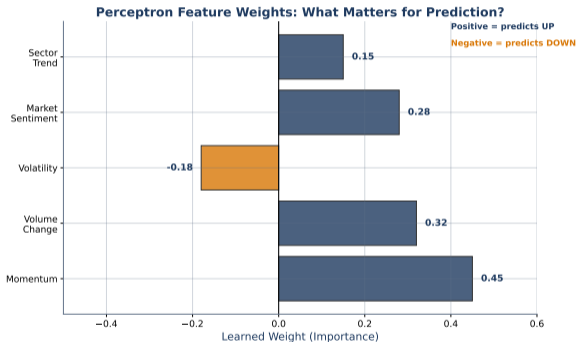
- Inputs: momentum and volatility features
- Output: up (1) or down (0) next-day prediction



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

Hands-On Exercise (25 min)

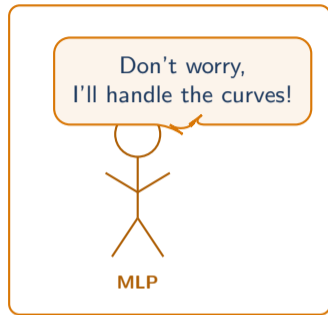
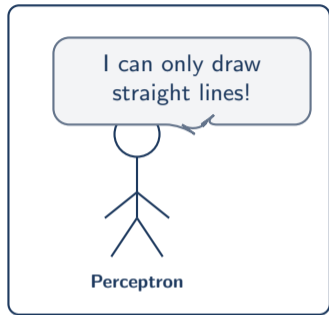
Task: Implement Perceptron from Scratch

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** → AI Winter → 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