The Perceptron

A Complete Tutorial

Based on Fausett, Fundamentals of Neural Networks, Chapter 2. Every numerical is traced against the source and independently re-computed.

1. What a Perceptron Is

A perceptron is a **single-layer, feed-forward** network that decides **membership in one class**: an output of +1 means "belongs to the class" and -1 means "does not belong." It has a layer of input units (one per pattern component), a bias unit whose signal is always 1, and a single output unit. Only the weights from the inputs (and bias) to the output are adjustable.

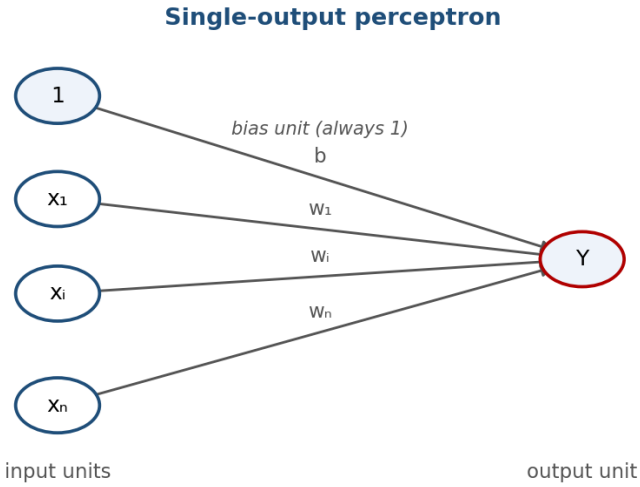

Figure 1. Architecture of a single-output perceptron.

Historical note. The original perceptrons (Rosenblatt 1962; Block 1962) had three layers — sensory, associator, response — but only the associator→response weights were trainable. Mathematically we therefore study just that one trainable layer, treating the associator units as inputs.

2. The Three Core Equations

(a) Net input

$$y\_in = b + \Sigma_i\, x_i \cdot w_i$$

(b) Activation function (note the undecided band of width 2θ)

$$y = \begin{cases} +1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

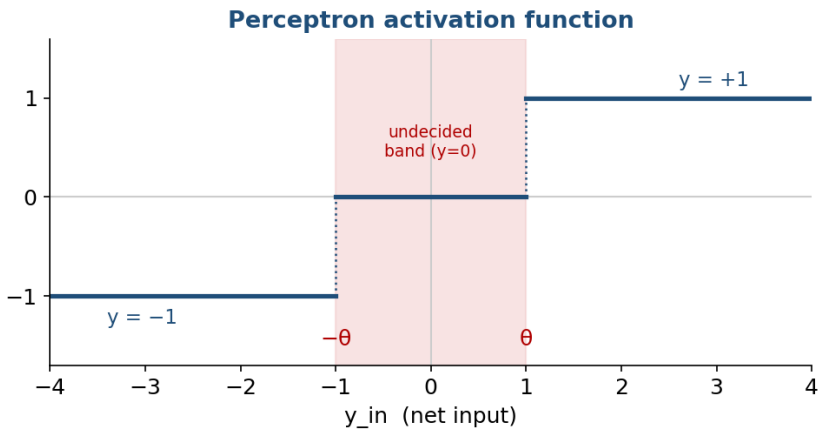

Figure 2. The perceptron activation function with its "undecided" zero-response band.

(c) Learning rule (applied only on error, only to active inputs)

$$w_i(\text{new}) = w_i(\text{old}) + \alpha \cdot t \cdot x_i$$
$$b\ (\text{new}) = b\ (\text{old}) + \alpha \cdot t$$

Here α is the learning rate $(0 < \alpha \leq 1)$ and t is the target $(+1 \text{ or } -1)$. The update fires **only when `y ≠ t`** and touches only weights whose input $x_i \neq 0$.

Why θ and the bias are NOT interchangeable here

For a plain step function a threshold and a bias can be swapped. For the perceptron they cannot, because θ sets the width of the undecided band (−θ ≤ y_in ≤ θ), not merely the position of the boundary. Changing θ changes the band width. So a perceptron needs both an adjustable bias and a threshold, and geometrically there are two boundary lines with a zero-response strip between them.

3. Algorithm — Single Output

```
Step 0.  Initialize weights and bias (0 for simplicity).
         Set learning rate α (may set α = 1).
Step 1.  While stopping condition is FALSE, do Steps 2–6.
   Step 2.  For each training pair s:t, do Steps 3–5.
      Step 3.  Set inputs:  xᵢ = sᵢ.
      Step 4.  y_in = b + Σ xᵢ wᵢ ;  apply activation → y.
      Step 5.  If y ≠ t:  wᵢ += α·t·xᵢ ;  b += α·t.
               Else: no change.
   Step 6.  If NO weight changed in Step 2 → STOP. Else continue.
```

Behaviour to remember: updates happen only on a misclassification, so as more patterns become correct, less learning occurs. Only weights on non-zero inputs change.

4. Numerical 1 — AND, Binary Inputs / Bipolar Targets

Setup: α = 1, θ = 0.2, initial (w_1, w_2, b) = (0, 0, 0). Third input component is the bias, always 1.

x_1	x_2	1	target t
1	1	1	+1
1	0	1	−1
0	1	1	−1
0	0	1	−1

Epoch 1 (row by row)

Pattern (x_1,x_2,1)	y_in	out y	t	Δ(w_1,w_2,b)	new (w_1,w_2,b)
start	—	—	—	—	(0, 0, 0)
(1,1,1)	0	0	+1	(1, 1, 1)	(1, 1, 1)
(1,0,1)	2	+1	−1	(−1, 0, −1)	(0, 1, 0)
(0,1,1)	1	+1	−1	(0, −1, −1)	(0, 0, −1)
(0,0,1)	−1	−1	−1	(0, 0, 0)	(0, 0, −1)

First row read-out: y_in = 0 lies in [−0.2, 0.2] so y = 0; since 0 ≠ +1 it is an error, and Δw = t·(x_1,x_2,1) = +1·(1,1,1).

Epoch 2 (full)

Pattern	y_in	out	t	Δ	new (w_1,w_2,b)
start	—	—	—	—	(0, 0, −1)
(1,1,1)	−1	−1	+1	(1, 1, 1)	(1, 1, 0)
(1,0,1)	1	+1	−1	(−1, 0, −1)	(0, 1, −1)
(0,1,1)	0	0	−1	(0, −1, −1)	(0, 0, −2)
(0,0,1)	−2	−1	−1	(0, 0, 0)	(0, 0, −2)

Epochs 3–10 (end-of-epoch weights)

Epoch	end (w_1,w_2,b)	Epoch	end (w_1,w_2,b)
1	(0, 0, −1)	6	(1, 2, −3)
2	(0, 0, −2)	7	(1, 2, −4)
3	(0, 1, −2)	8	(2, 2, −4)
4	(0, 1, −3)	9	(2, 3, −4)
5	(1, 1, −3)	10	(2, 3, −4) → no change: CONVERGED

Final weights (2, 3, −4). Verification with θ = 0.2: (1,1)→y_in=1→+1 ✓; (1,0)→−2→−1 ✓; (0,1)→−1→−1 ✓; (0,0)→−4→−1 ✓ **Cost of binary inputs: 10 epochs.**

5. Numerical 2 — AND, Bipolar Inputs / Targets

Setup: α = 1, θ = 0, initial (w_1, w_2, b) = (0, 0, 0).

x_1	x_2	1	t
1	1	1	+1

x_1	x_2	1	t
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

Epoch 1 (full)

Pattern (x_1,x_2,1)	y_in	out y	t	Δ(w_1,w_2,b)	new (w_1,w_2,b)
start	—	—	—	—	(0, 0, 0)
(1,1,1)	0	0	+1	(1, 1, 1)	(1, 1, 1)
(1,−1,1)	1	+1	−1	(−1, 1, −1)	(0, 2, 0)
(−1,1,1)	2	+1	−1	(1, −1, −1)	(1, 1, −1)
(−1,−1,1)	−3	−1	−1	(0, 0, 0)	(1, 1, −1)

Epoch 2 (the check pass)

Pattern	y_in	out	t	Δ	weights
(1,1,1)	1	+1	+1	(0,0,0)	(1, 1, −1)
(1,−1,1)	−1	−1	−1	(0,0,0)	(1, 1, −1)
(−1,1,1)	−1	−1	−1	(0,0,0)	(1, 1, −1)
(−1,−1,1)	−3	−1	−1	(0,0,0)	(1, 1, −1)

All Δw = 0 in epoch 2 → the net was fully trained after epoch 1. Final weights (1, 1, −1), giving the separating line $x_2 = -x_1 + 1$.

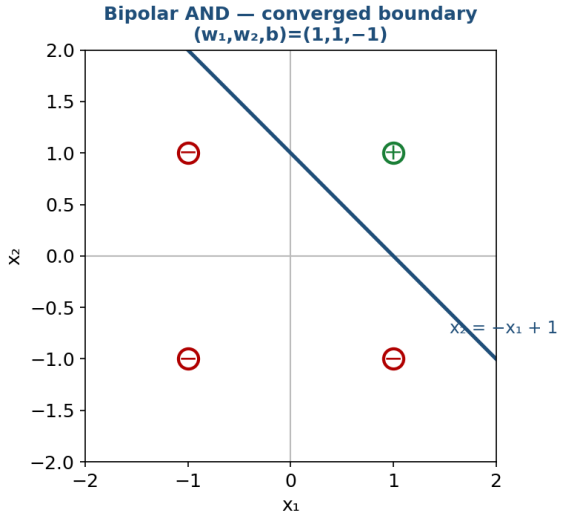

Figure 3. Converged decision boundary for bipolar AND.

The teaching punchline

Representation	Epochs to converge
Binary inputs (Numerical 1)	10
Bipolar inputs (Numerical 2)	1

Same problem, same logic. With binary inputs a 0 component cannot change its weight ($\Delta w_i = \alpha \cdot t \cdot x_i = 0$ when $x_i = 0$), so “off” inputs never learn. Bipolar −1 inputs stay active and keep supplying corrections.

6. Numerical 3 — Perceptron Beats the Hebb Rule

The Hebb rule fails on some linearly-separable problems; the perceptron does not. Mapping (a slice of 3-input parity): target +1 if no input is zero, −1 if exactly one input is zero. **Setup:** α = 1, θ = 0.1, init $(w_1,w_2,w_3,b) = (0,0,0,0)$.

Epoch 1 (full)

Pattern $(x_1,x_2,x_3,1)$	y_in	out	t	$\Delta(w_1,w_2,w_3,b)$	new (w_1,w_2,w_3,b)
start	—	—	—	—	(0, 0, 0, 0)
(1,1,1,1)	0	0	+1	(1, 1, 1, 1)	(1, 1, 1, 1)
(1,1,0,1)	3	+1	−1	(−1,−1, 0,−1)	(0, 0, 1, 0)
(1,0,1,1)	1	+1	−1	(−1, 0,−1,−1)	(−1, 0, 0,−1)
(0,1,1,1)	−1	−1	−1	(0, 0, 0, 0)	(−1, 0, 0,−1)

The perceptron keeps correcting over further epochs and converges. The lesson is existence: **whatever the single-pass Hebb rule cannot learn on a separable problem, the iterative, error-driven perceptron can.** (The source prints only selected epochs and no clean final vector, so none is invented here — epoch 1 above is fully verified.)

7. Perceptron Convergence Theorem

Statement. If the training set is linearly separable — i.e. there exists a weight vector w* with f(x(p)·w*) = t(p) for every pattern p — then from any starting vector the perceptron learning rule reaches a weight vector that classifies all patterns correctly, in a finite number of steps. The vector found need not equal w* and need not be unique.

Bound on the number of weight updates

$$k \le M \cdot \|w^*\|^2 / m^2$$

where m = min over the set { x·w* } > 0, and M = max over the set { $\|x\|^2$ }, after rewriting every pattern so all targets are +1 (flip the sign of every component of any pattern whose target was −1).

Proof sketch (why k is finite)

Starting from w(0)=0, after k error-updates:

- **Lower bound (grows like k^2):** by Cauchy–Schwarz, $\|w(k)\|^2 \ge (w(k)\cdot w^*)^2 / \|w^*\|^2 \ge (k\cdot m)^2 / \|w^*\|^2$.
- **Upper bound (grows like k):** since each errored pattern has x·w ≤ 0, $\|w(k)\|^2 \le \|w(k-1)\|^2 + \|x\|^2 \le k\cdot M$.

A quantity that is simultaneously $\ge (km)^2/\|w^*\|^2$ and $\le kM$ cannot let k grow without bound; squeezing gives $k \le M\|w^*\|^2/m^2$. Hence updates are finite. ■

What the proof does NOT require

Binary inputs are not needed; any positive learning rate α works; any fixed θ works; the exact target magnitude does not matter — only bounded pattern norms and the existence of a separating w*.

8. Several-Output Perceptron (Character Recognition)

To classify into one of several classes, use **one output unit per class**. Because the net is single-layer, the weight column for output j is trained independently of every other output — you are simply solving several one-class problems in parallel.

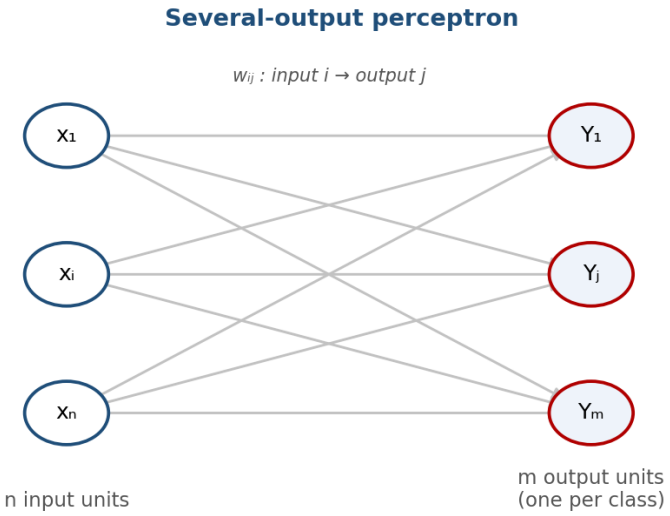

Figure 4. Several-output perceptron: each output owns an independent weight column.

Example architecture: 63 input units (a 7×9 pixel letter) → 7 output units for A, B, C, D, E, J, K. Weight w_{ij} connects input i to output j; each output has its own bias b_j.

Algorithm (multi-output, bipolar pairs, α = 1)

```
Step 0.  Initialize weights & biases (0 or small random).
Step 1.  While stopping condition FALSE:
   Step 2.  For each bipolar pair s:t:
      Step 3.  x_i = s_i
      Step 4.  y_in_j = b_j + Σ_i x_i w_ij
               y_j = +1 if y_in_j> θ ; 0 if -θ≤y_in_j≤θ ; -1 if <-θ
      Step 5.  If t_j ≠ y_j:  b_j += t_j ;  w_ij += t_j·x_i.  Else unchanged.
   Step 6.  Stop if no weight changed.
```

Target encoding: "A" → (1, −1, −1, −1, −1, −1, −1), "B" → (−1, 1, −1, −1, −1, −1, −1), and so on — one bipolar slot per class. In practice an unknown character ideally yields one +1 and six −1s; ties are resolved by comparing the y_in magnitudes of the competing outputs. The trained net also generalizes to noisy inputs (a few flipped pixels), which is its practical value.

9. Limitation — XOR Is Not Linearly Separable

A single perceptron can learn **only** linearly separable problems. XOR is the classic case it cannot solve: no single straight line puts both + points on one side and both − points on the other.

x_1	x_2	target t
1	1	−1

x_1	x_2	target t
1	−1	+1
−1	1	+1
−1	−1	−1

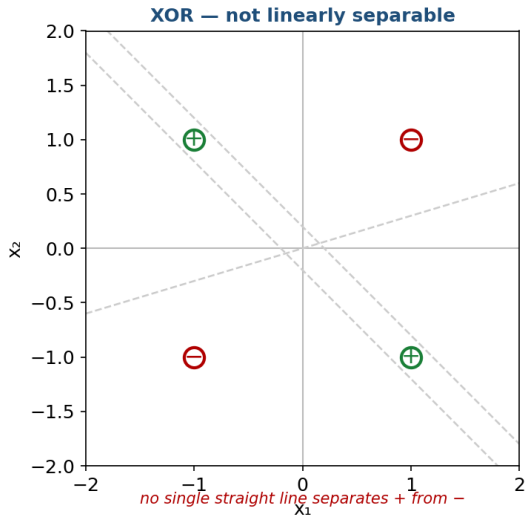

Figure 5. XOR: the + and − points cannot be split by any one line.

This is a structural limit of the single-layer perceptron, not a training failure — the convergence theorem simply does not apply, because no separating w* exists. Solving XOR requires more than one layer.

10. Note — A "Delta-Form" Update for the Perceptron

One-line answer

A delta-form update that uses the thresholded output y, $\Delta w = \alpha(t - y)\cdot x$, is valid for the perceptron: because y = t on correct patterns, (t − y) is nonzero only on error, so it collapses back to the ordinary perceptron rule $\Delta w = \alpha\cdot t\cdot x$. (The genuine delta rule instead uses the linear net input y_in, $\Delta w = \alpha(t - y\_in)\cdot x$ — that version trains the ADALINE, not the perceptron.)

11. One-Page Reference

Item	Formula / fact
Net input	$y\_in = b + \Sigma\, x_i \cdot w_i$
Activation	$+1$ if $y\_in > \theta$; 0 if $-\theta \le y\_in \le \theta$; -1 if $y\_in < -\theta$
Update (on error only)	$w_i \mathrel{+}= \alpha \cdot t \cdot x_i$; $b \mathrel{+}= \alpha \cdot t$
Convergence	finite steps iff linearly separable; $k \le M\|w^*\|^2/m^2$
Bipolar > binary	active -1 inputs keep learning; 0 inputs cannot
AND, binary inputs	converges in 10 epochs → (2, 3, −4)
AND, bipolar inputs	converges in 1 epoch → (1, 1, −1)
Several outputs	one output unit per class, trained independently
XOR	not separable → a single perceptron cannot solve it
Delta-form for perceptron	$\Delta w = \alpha(t-y) \cdot x$ with thresholded $y \equiv$ perceptron rule