Early Neural Network Models — A Short Tutorial

McCulloch–Pitts · Hebb Net · Perceptron · ADALINE · MADALINE · Delta Rule *@SSRoy*

The thread through all of them

A single neuron draws **one straight line** and labels '+' on one side, '−' on the other. Problems a line can split are **linearly separable** (AND, OR); **XOR is not**. This one fact explains what each single-unit model can do, why XOR needs **MADALINE**, and how they learn their line — by hand (MP), by correlation (Hebb), by error-correction (Perceptron), or by least-squares (ADALINE, via the **delta rule**).

1. McCulloch–Pitts (MP) neuron

- Oldest model (1943). **Binary** inputs; weights and threshold **θ fixed by hand** — **no learning**.
- Fires (outputs 1) if $net = \Sigma\ w_i x_i \geq \theta$, else 0. A single inhibitory input can veto firing.

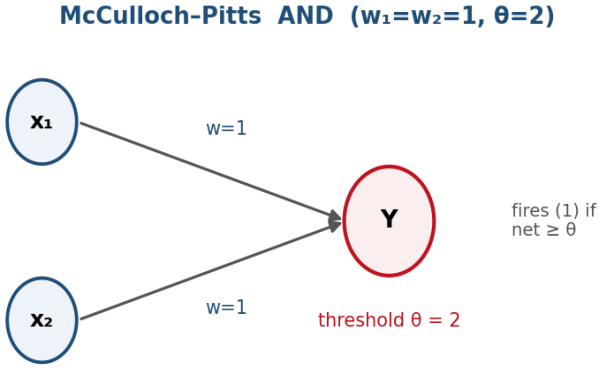

Figure 1. MP neuron wired as AND ($w_1=w_2=1$, $\vartheta=2$).

AND (θ = 2) and OR (θ = 1)

x_1	x_2	net	AND (θ=2)	OR (θ=1)
1	1	2	**1**	**1**
1	0	1	0	**1**
0	1	1	0	**1**
0	0	0	0	0

Limit

One MP neuron realises only linearly separable gates. AND, OR ✓; **XOR needs a second layer**.

2. Hebb Net

The earliest **learning** rule (Hebb, 1949): 'neurons that fire together, wire together.' A weight grows when input and target are both active. One pass over the data (**no iteration**).

Hebb algorithm

```
Step 0. w_i = 0 (i = 1..n),  b = 0.
Step 1. for each training pair s:t:
   Step 2. x_i = s_i
   Step 3. y = t
   Step 4. w_i(new) = w_i(old) + x_i·y
           b (new) = b (old) + y
```

Worked: AND (bipolar inputs & targets) — Δw = x·t, one pass

Pattern (x_1,x_2,1)	t	Δ(w_1,w_2,b) = (x_1t, x_2t, t)	new (w_1,w_2,b)
start	—	—	(0, 0, 0)
(1, 1, 1)	+1	(1, 1, 1)	(1, 1, 1)
(1, −1, 1)	−1	(−1, 1, −1)	(0, 2, 0)
(−1, 1, 1)	−1	(1, −1, −1)	(1, 1, −1)
(−1, −1, 1)	−1	(1, 1, −1)	(2, 2, −2)

Final weights (2, 2, −2) → separating line $x_1 + x_2 - 1 = 0$, which classifies AND correctly. Bipolar coding is essential here.

Limits of Hebb

With **binary** data it cannot learn when a target is 0 (no update happens). Even in bipolar form it is a one-shot correlational rule and can **fail on some linearly separable problems** — which is exactly why iterative, error-driven rules (Perceptron, ADALINE) were developed.

3. Linear separability (the bridge to trainable nets)

Every single unit fires on one side of the line $b + w_1x_1 + w_2x_2 = 0$ (its **decision boundary**). It can solve a problem only if **one straight line** separates the '+' points from the '−' points.

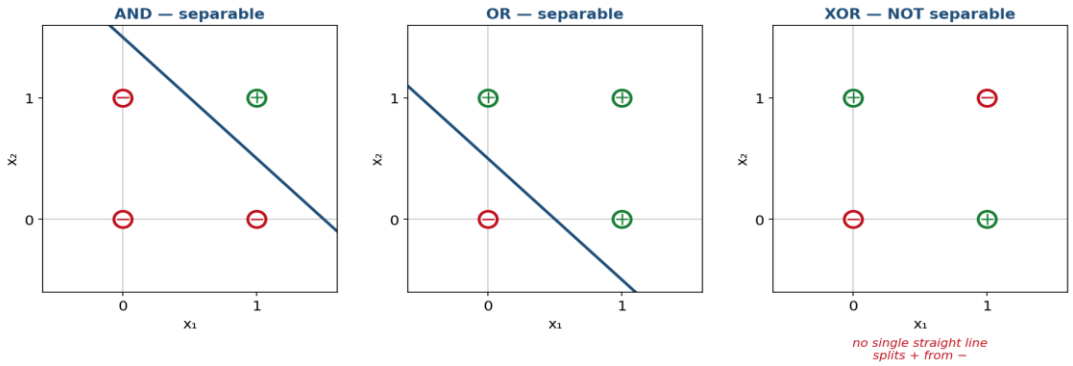

Figure 2. AND and OR are linearly separable; XOR is not — no single line splits its + and − corners.

- **Separable (AND, OR)** → a single MP / Hebb / Perceptron / ADALINE **can** solve it.
- **Not separable (XOR)** → no single-line model can (no weights exist) — a **structural** limit, not a training fault.
- Fix = **more layers** (several lines): this is why **MADALINE** and multilayer nets exist.
- The **bias `b`** frees the line from passing through the origin — often the difference between solvable and not.

4. Perceptron (algorithm only)

The perceptron keeps the single-line geometry but **learns by error correction**: it updates weights **only when it misclassifies** a pattern, and is **guaranteed to converge if the data are linearly separable**. Algorithm below; full treatment to follow.

```
Step 0. init weights & bias to 0;  set α (0 < α ≤ 1).
Step 1. while stopping condition is false:
   Step 2. for each training pair s:t:
      Step 3. xᵢ = sᵢ
      Step 4. y_in = b + Σ xᵢ wᵢ
              y = 1 if y_in > θ ; 0 if −θ ≤ y_in ≤ θ ; −1 if y_in < −θ
      Step 5. if y ≠ t:  wᵢ += α·t·xᵢ ;  b += α·t     (else no change)
   Step 6. stop when no weight changes in a full pass.
```

Placeholder

Detailed perceptron examples, the θ-band, and the convergence theorem are to be added separately.

5. ADALINE (Adaptive Linear Neuron)

Learns by the **delta rule (Widrow–Hoff / LMS)**. Training uses the **linear** net input `y_in`; a threshold is applied only afterwards to classify.

ADALINE (delta-rule) algorithm

```
Step 0. init weights (small);  set α.
Step 1. while not converged:
   Step 2. for each bipolar pair s:t:
      Step 3. x_i = s_i
      Step 4. y_in = b + Σ x_i w_i          (LINEAR)
      Step 5. w_i += α(t − y_in) x_i ;  b += α(t − y_in)
   Step 6. stop when weight changes are tiny.
```

Worked: AND (bipolar), α = 0.1, start (0,0,0) — epoch 1

Pattern (x_1,x_2)	t	y_in	t − y_in	new (w_1, w_2, b)
start	—	—	—	(0, 0, 0)
(1, 1)	+1	0	+1.00	(0.100, 0.100, 0.100)
(1, −1)	−1	0.100	−1.10	(−0.010, 0.210, −0.010)
(−1, 1)	−1	0.210	−1.21	(0.111, 0.089, −0.131)
(−1, −1)	−1	−0.331	−0.669	(0.178, 0.156, −0.198)

Over many epochs the weights settle at the least-squares solution **$w_1 = w_2 = ½$, $b = −½$** (line $x_1 + x_2 - 1 = 0$). Unlike the perceptron, it updates on **every** pattern via the residual `t − y_in`.

6. MADALINE (Many ADALINEs)

Two **hidden** ADALINEs feed one **output** unit fixed to perform **OR**. Combining two lines lets MADALINE carve a **non-separable** region — so it **solves XOR**.

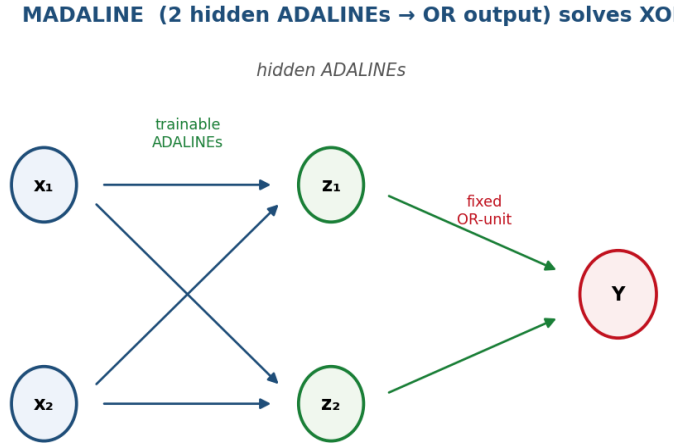

Figure 3. MADALINE: two trainable hidden ADALINEs → fixed OR output → XOR solvable.

MRI algorithm (original MADALINE rule)

```
Step 0. output weights FIXED so Y = OR(z_1,z_2); init hidden weights; set α.
Step 1. while not converged, for each bipolar pair s:t:
   Step 2. forward: z_in_j = b_j + Σ x_i w_ij ; z_j = f(z_in_j) ; y = OR(z_1,z_2)
   Step 3. if y = t : no change.
```

```
   Step 4. if t = +1: train the hidden unit whose net is CLOSEST to 0
           toward +1 (delta rule).
   Step 5. if t = −1: train EVERY hidden unit with positive net
           toward −1 (delta rule).
Step 6. stop when weights settle.
```

Why it works

Each hidden ADALINE draws its own line; the OR output fires if either region is active — two lines bound the two '+' corners of XOR that no single line could isolate. MRI uses a **minimal-disturbance** heuristic ('don't rock the boat').

7. Delta Rule — definition

Definition

The **delta rule** (also **Widrow–Hoff** or **LMS**, Least Mean Squares) adjusts each weight in proportion to the difference between the **target** and the **linear net input**: $\Delta w_i = \alpha(t - y\_in) \cdot x_i$, with $\Delta b = \alpha(t - y\_in)$. It is obtained by **gradient descent** on the squared error $E = (t - y\_in)^2$, so it minimises mean-squared error. It is the training rule of the **ADALINE** and the foundation of **backpropagation** (its multilayer, nonlinear generalisation).

8. Roadmap — networks coming next

The models above are single-/simple-layer supervised nets. The next set (to be added later) covers associative memory, competition, self-organisation and adaptive resonance:

Associative memory nets

- **Heteroassociative Memory NN** — maps input patterns to *different* stored output patterns.
- **Autoassociative Network** — recalls a stored pattern from a noisy or partial version of *itself*.
- **Iterative Autoassociative Network** — feeds output back to input repeatedly until it settles on a stored pattern.
- **BAM (Bidirectional Associative Memory)** — two-layer memory that recalls in *both* directions (input ↔ output).

Competitive nets

- **Maxnet** — finds the maximum by mutual inhibition (one winner survives).
- **Mexican Hat Network** — on-centre / off-surround lateral links; sharpens contrast.
- **Hamming Network** — classifies by the closest stored exemplar (minimum Hamming distance).

Self-organising & hybrid

- **Kohonen SOM** — unsupervised topological map; nearby neurons respond to similar inputs.
- **LVQ (Learning Vector Quantization)** — supervised competitive classifier refining class prototypes.
- **Counterpropagation Network** — Kohonen + Grossberg layers for fast approximate mapping.

Adaptive Resonance Theory

- **ART1** — stable category learning for *binary* inputs, controlled by a vigilance parameter.
- **ART2** — the ART family extended to *continuous / analog* inputs.