Two Historical Neural-Net Machines — A Full-Size, Homemade Build Guide

Build a real Rosenblatt Perceptron and a real Minsky SNARC from household + hobby-electronics parts. Honest engineering, full scale, no toy demos. — ALMAWARE

---

0. Read This First — What "Full-Size" Actually Means

These are not the two-resistor desk demos. You are building:

- **Machine A — Perceptron:** a **16-cell light "retina"** (LDR photoresistor grid) wired into one trainable neuron that learns to tell light-patterns apart (a bright bar vs. a dark bar, an "L" vs. a "T", left-lit vs. right-lit). Sixteen real, independently adjustable weights, auto-tuned by the learning rule.
- **Machine B — SNARC:** a **40-synapse reinforcement learner** — the same scale as Minsky & Edmonds' 1951 machine — that learns a behavior by reward, with a deliberate random (exploration) element, orchestrated so it actually solves a small maze / learns an action policy.

The honest truth up front: full size is genuinely bigger than a demo. Forty synapses is *forty* of everything in the synapse cell. The single biggest decision is **how you store and adjust a weight**, because you do it 16 or 40 times. Get that cell right, make it a module, and replicate it: **design one cell, build it modularly (boards of 8), and let muxes + one microcontroller scale it.**

Two recurring tricks make "full" tractable instead of insane:

1. **Analog multiplexers (CD74HC4067):** one 16-channel mux lets a single MCU pin read 16 sensors or address 16 weights — how you avoid running out of pins.
2. **Digital potentiometers (MCP4131) on a shared SPI bus:** each is a chip-in-a-cell that *is* the weight. The MCU sets all of them over three shared wires — turns a weekend of soldering foil capacitors into an afternoon.

You'll get the purist homemade options too (pencil-graphite weights, aluminum-foil capacitors) — they're real and they work — presented honestly as the harder road.

PART A — THE FULL ROSENBLATT PERCEPTRON

A1. What It Is

Rosenblatt's Perceptron (1957–58) **learned to classify patterns** projected onto a grid of light sensors. Light hits sensors -> each signal is multiplied by an adjustable **weight** -> all weighted signals are **summed** -> if the sum crosses a **threshold**, the output neuron "fires" (class 1), else stays quiet (class 0). Training nudges the weights until the firing is correct.

Our version: **Retina** = 16 LDRs in a 4x4 grid (extend the same method to 4x5 = 20). **Weights** = 16 + 1 bias, each independently set. **Neuron** = an analog summer (op-amp) + threshold (comparator), OR the MCU does sum-and-threshold in software while analog hardware does the sensing (recommended for clean auto-training of 16 weights).

A2. Architecture

```
      4x4 LDR RETINA (light pattern shines on this)
LDR0..LDR15 -> each LDR + fixed resistor = voltage divider (0..5V)
      |  16 analog sensor lines
```

```
     v
CD74HC4067 16:1 MUX  <- MCU picks which LDR to read (4 select pins)
     |  one analog line
     v
Arduino A0 (ADC)  -- reads x0..x15 one at a time --+
                                                   |
THE WEIGHTS: 16x MCP4131 digital pots (shared SPI) <- MCU sets w0..w15
     |                                              |
     v                                              v
ANALOG SUMMING NEURON (optional):        MCU computes S = Sum(wi*xi)+bias
op-amp inverting summer -> LM393          y = 1 if S>0 else 0  -> LED fires
comparator -> LED "NEURON FIRES"          (software neuron = training brain)
```

Two valid neurons: *Software neuron* (recommended for learning) — the MCU reads the 16 LDRs, knows its 16 weights, computes `S = Sum(wi*xi)+b`, decides fire/not, lights an LED. *Analog neuron* (authentic) — the pot wipers feed an op-amp inverting summer + LM393 comparator; the sum & decision happen in hardware. Build the software neuron first, add the analog summer as a "make it physically real" phase. Weights + learning code are identical.

A3. Three Ways to Store a Weight

Option	What	Pros	Cons	Verdict
MCP4131 digital pot *(recommended)*	10k pot, 129 taps, set by SPI	MCU sets in microseconds; perfect for auto-training 16; ~$1.2 ea	bipolar needs a trick; CS wrangling	**Build this**
Motorized fader	sliding pot a motor moves	you *see* weights move; bipolar-friendly	~$8-15 ea; bulky; needs driver	gorgeous showpiece (do 4-8)
Pencil graphite	graphite line + wiper	zero-cost, true homemade spirit	not MCU-settable; noisy	one hero cell

Build all 16 as MCP4131; optionally one pencil-graphite + one motorized fader as showpieces.

A4. The LDR Retina

Each LDR = a voltage divider: `+5V -[LDR]- node -[10k]- GND`, node -> mux channel. Bright -> voltage up; dark -> down. Do it 16x; all outputs to one CD74HC4067; common out -> Arduino A0. MCU sets the 4 select pins 0..15 and reads A0 each time -> it has x0..x15. Mount 16 LDRs in a clean 4x4 grid behind a cardboard mask with a 4x4 window.

A5. The Weight Bank (16x MCP4131)

All share SCK/MOSI (SPI). Each needs a CS — use a Mega (16 direct CS pins, simplest) or a second CD74HC4067 as a 1-to-16 CS selector. Software neuron: the MCU remembers what it wrote, so the pot value IS the weight (also feeds the analog summer if built).

A6. Bipolar Weights (the one gotcha)

Weights can be negative; a plain pot is 0..+. Fixes: (1) **software neuron** — store signed in code, map to pot only for the analog summer (easiest); (2) **bias the midpoint** — tap 64 = "zero", feed wiper relative to a 2.5V reference; (3) **two rails** — route input to both inverting & non-inverting nodes (most analog-pure).

A7. The Math — Perceptron Learning Rule

```
S = b + Sum(wi*xi)          for i=0..15
y = 1 if S>0 else 0         (prediction)
e = t - y                   (error, t=target in {0,1}, e in {-1,0,+1})
wi <- wi + eta * e * xi      for all i
b  <- b  + eta * e
```

Right -> e=0, nothing changes. Should-have-fired-but-didn't (e=+1) -> raise weights of lit pixels. Fired-but-shouldn't (e=-1) -> lower them. **The line of code that IS the learning:**

```
new_weight[i] = old_weight[i] + ETA * (target - output) * pixel[i]
```

...for i=0..15, then the MCU writes each new weight to its MCP4131 over SPI. Sixteen pots physically changing resistance = the machine learning, made of matter.

A8. Training Procedure

1. Task (binary, visual): bright-left vs bright-right, or bar vs blank, or "L" vs "T" on the 4x4. 2. Make example cards/stencils. 3. Init 16 weights + bias to small values. 4. Loop over examples: MCU reads 16 LDRs -> computes S,y -> compares to label t -> applies update -> writes new weights. 5. Repeat epochs (10-50) to zero errors. For linearly-separable tasks convergence is **guaranteed** (Perceptron Convergence Theorem). 6. You'll see the LED start firing only for class 1; on a serial plot you watch the weights settle.

Honest limit (teach-it-to-fail): a single perceptron only separates **linearly-separable** patterns — it will *never* learn XOR. That's the Minsky-Papert limitation; try teaching it XOR, watch it fail to converge, and you've reproduced a pivotal AI moment by hand.

A9. Perceptron BOM

#	Part	Qty	~Unit	~Total
1	LDR / photoresistor (GL5528)	20	$0.15	$3
2	Fixed resistor 10k	20	$0.02	$0.40
3	MCP4131 digital pot 10k	17	$1.20	$20
4	CD74HC4067 16-ch mux	2	$1.50	$3
5	Arduino Mega 2560	1	$15	$15
6	TL074 op-amp *(opt summer)*	1	$0.60	$0.60
7	LM393 comparator *(opt)*	1	$0.40	$0.40
8	LED + 330R	2	$0.10	$0.20
9	Motorized fader *(showpiece)*	0-4	$10	$0-40
10	5V 2A supply	1	$6	$6
11	Breadboard/perfboard + jumpers	—	—	$12
	PERCEPTRON TOTAL (software neuron)			**~$60**

A10. Build & Calibration

1. Build & test ONE LDR divider (A0 swings clean). 2. Build the 4x4 retina; outputs -> mux #1; mount mask. 3. Mux scan test; calibrate each cell's bright/dark voltage. 4. Wire one MCP4131, prove SPI set; replicate to 16. 5. Software neuron first; set weights by hand to "detect left half"; confirm LED logic. 6. Add the learning rule; train; watch it converge. 7. (Opt) add the analog summer — now the decision is analog.

PART B — THE FULL MINSKY SNARC

B1. What It Is

In 1951 Minsky & Edmonds built SNARC (Stochastic Neural Analog Reinforcement Calculator): ~40 synapses, each a **capacitor whose charge was the weight**, a motor/clutch to adjust it, and reinforcement. It modeled a rat learning a maze: actions partly random (the **stochastic** exploration), and on success the synapses active on the winning path were **strengthened**. Learning by reward — the first hardware reinforcement learner.

Our version keeps every essential piece: 40 synapses; capacitor-as-memory option (foil/wax-paper) AND a practical digital-pot option; a stochastic action selector; a reinforcement update; an Arduino Mega + muxes orchestrating all 40; a concrete task it visibly learns.

B2. Architecture — How 40 Synapses Form a Brain

40 synapses wire into a small network of **action-neurons**. Clean layout: **8 state lines** (which maze cell am I in) x **4 action-neurons** (N/E/S/W) = **32 synapses** + ~8 bias/eligibility cells = **~40**. Each synapse stores "how good is action a in state s" — a tabular **Q-value in hardware**.

```
STATE LINES  s0 s1 s2 s3 s4 s5 s6 s7
GO NORTH  <- synapses across all 8 states (8 weights)
GO EAST   <- synapses across all 8 states (8 weights)
GO SOUTH  <- synapses across all 8 states (8 weights)
GO WEST   <- synapses across all 8 states (8 weights)   = 32  (+8 bias = ~40)

Each synapse = [analog weight memory] + [reinforcement update circuit]
               (foil-cap charge  OR  MCP4131 tap)

ACTION SELECTION per step:
  1. read the 4 action sums for the current state
  2. STOCHASTIC pick: usually the strongest, but with prob e take RANDOM  <- exploration
  3. move the rat; reach the cheese -> REWARD
  4. REINFORCE: strengthen synapses active on the path taken
```

Modular build rule: make a **board of 8 synapses** as your unit. Five identical boards = 40. Debug one, clone four.

B3. The Synapse Cell — Two Options

Option 1 (recommended to finish): digital-pot synapse. Each synapse = one MCP4131 tap = the weight, held in MCU memory + the pot. Reinforcement = MCU nudges the tap up/down. 40 MCP4131s on shared SPI, CS via CD74HC4067 muxes. This path turns SNARC from a multi-weekend epic into a finished, learning machine.

Option 2 (true Minsky spirit): capacitor-memory synapse. Each = a capacitor whose voltage is the weight. Homemade cap = foil/wax-paper sandwich. *Strengthen* = charge it (pulse through R). *Forget* = bleed via a high-value leak resistor (beautifully lifelike). *Read* via a high-Z op-amp buffer into the mux/ADC. Eligibility = only path-active caps get the reward pulse.

```
CAP SYNAPSE CELL:
  reward pulse -[R_charge]-+-------+- to high-Z buffer -> (read weight)
    (only if eligible)     |       |
                         [Cap]   [R_leak]  <- slow forget
                          GND     GND
```

	Cap-memory (foil)	Digital-pot (MCP4131)
Faithful to 1951	***	*
Effort for 40	high (40 analog cells)	moderate (40 chips, shared bus)
Reliability	drifts/leaks (charming, fiddly)	rock-solid
Verdict	**hero showpiece / a few cells**	**build all 40 this way to finish**

B4. The Math — SNARC Reinforcement Update

No teacher — only reward. Each synapse holds `w[s,a]` = value of action a in state s.

```
ACTION (e-greedy):  with prob (1-e): a = argmax_a w[s,a]   (exploit)
                    with prob  e   : a = random            (EXPLORE = stochastic)
ELIGIBILITY: mark synapse (s,a) used this step as active.
ON REWARD r (+1 at goal, small step penalty):
   for each active (s,a):  w[s,a] <- w[s,a] + ALPHA * r * eligible[s,a]
```

Reward raises the weights of moves that led to success -> next time they win the argmax -> the rat repeats the good path. A small decay (`w <- w*0.99` or capacitor leak) prevents runaway + implements forgetting. **The learning, one line:**

```
new_w[s][a] = old_w[s][a] + ALPHA * reward * eligible[s][a]
```

Fuller temporal-difference (credit every step) = textbook **Q-learning**:

```
new_w[s][a] = old_w[s][a] + ALPHA*(reward + GAMMA*max_a' w[s'][a'] - old_w[s][a])
```

The SNARC is its physical ancestor — your hardware runs it.

B5. The Demonstrable Task — Solve a Maze

A 3x3 / 8-cell maze with start, cheese goal, a wall or two. The "rat" = a position the MCU tracks (show it as a lit LED moving on a 3x3 LED grid). Each episode: rat starts; each step reads the 4 action-weights for its cell, picks e-greedy, moves (wall = stay + penalty); reach cheese -> reward -> reinforce path. **What you'll see:** trial 1 wanders randomly; over ~20-50 trials the path-weights climb, the route shortens, until the rat **beelines to the cheese**. Decay e over time. On an LED maze you watch a blinking dot learn the route — the SNARC learning, in front of you. *One-evening alt:* a 2-bit action policy (2 switches = situation, learn by reward which of 4 actions is right).

B6. SNARC BOM

#	Part	Qty	~Unit	~Total
1	Arduino Mega 2560	1	$15	$15
2	MCP4131 digital pot 10k (synapse)	40	$1.20	$48
3	CD74HC4067 16-ch mux	4	$1.50	$6
4	74HC595 shift reg (CS, opt)	5	$0.40	$2
5	TL074 quad op-amp (buffers/summers)	3	$0.60	$1.80
6	LED grid maze viz + resistors	~9	$0.10	$1
7	Indicator LEDs	8	$0.10	$0.80
8	Buttons / DIP switches (reward/state)	4	$0.20	$0.80
9	Protoboards — 5x "8-synapse" modules	5	$2	$10
10	5V 3A supply	1	$8	$8
11	Jumper wire / headers / sockets	—	—	$15
	SNARC TOTAL (40 digital-pot synapses)			**~$108**
Cap-memory showpiece tier: foil+wax (homemade caps) $0-3; film caps 40x $4; 2N3904/CD4066 eligibility gates ~$6; 1-10M bleed resistors $0.80				**+~$15 & much labor**

B7. Build Steps

1. Build ONE 8-synapse module (8 MCP4131 on shared SPI, CS selectable); prove set+read all 8 — **debug fully before cloning.** 2. Clone to 5 modules = 40; share SCK/MOSI; route 40 CS via muxes/shift-regs. 3. Read path: each module -> CD74HC4067 -> buffer -> Mega ADC. 4. Wire the LED maze + reward button (+ switches for the 2-bit task). 5. Code the orchestrator: state encoding -> read the 4 action-weights -> e-greedy pick -> move -> on goal apply the update + write new weights. 6. Run episodes; watch convergence; tune ALPHA, e-decay, step penalty. 7. (Opt) replace 4-8 synapses with capacitor cells for authentic 1951 analog memory.

B8. Honest Scope & Scaling

40 synapses is real work — digital-pot path ~ a **focused weekend** to working + tuning; cap-memory path = a **multi-weekend craft project** (~40 analog cells), worth it for the authentic machine. Smartest way to "full" without chaos: **modular boards of 8**; **digital pots over foil caps** for the working machine (foil caps as hero cells); **muxes + shared SPI** so one Mega controls all; **software keeps the source-of-truth weights**, hardware embodies them (a bug is a code fix, not a re-solder). **It genuinely learns** — the Perceptron converges on linearly-separable visual patterns; the SNARC's rat learns its maze by reward. You're rebuilding 1951 and 1958 on your bench, honestly and at full scale.

Appendix — Shared Wisdom

- **Common power & ground:** tie all grounds; beefy 5V for 40 chips + 0.1uF decoupling near each cluster.
- **Mux discipline:** a few us settle time after changing select lines before ADC-reading.
- **Bipolar weights:** keep the signed truth in MCU memory; map to the unipolar pot/cap only at the hardware layer.
- **Serial plotting:** stream the 16/40 weights to the Arduino Serial Plotter — *seeing* them move during training is the most convincing proof (and thrilling).
- **Start small, prove the cell, then replicate** — the habit that makes "full-size" finishable.

Two real learning machines — from light sensors, pots, op-amps, a microcontroller, and (if you choose) foil-and-wax capacitors — the honest, full-scale descendants of the Perceptron and the SNARC. — ALMAWARE