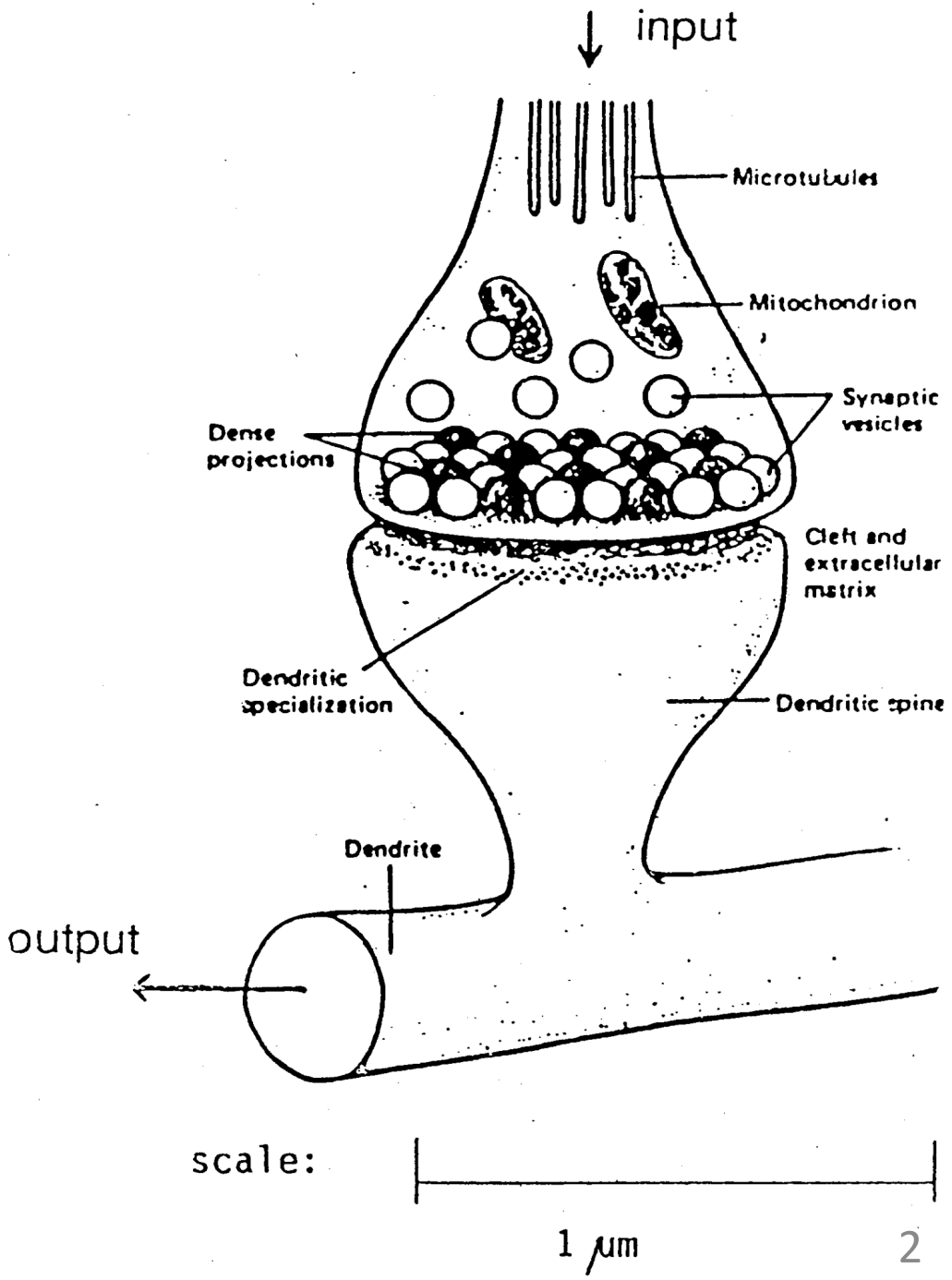
Computational model of a single neuron - Perceptron

- Biological synapse
- McCulloch-Pitts abstraction
  - Some McCulloch-Pitts cells
  - Gating network with memory
  - Stimulus-response decoder
- Rosenblatt perceptron machine
- Computational model of perceptron
- Eliminating thresholds
- Perceptrons problems - Minsky & Papert

A typical synapse

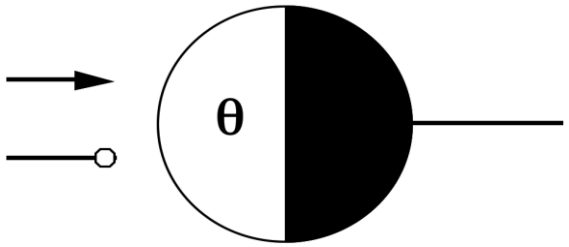


McCulloch & Pitts (1943)

Abstract neuron description

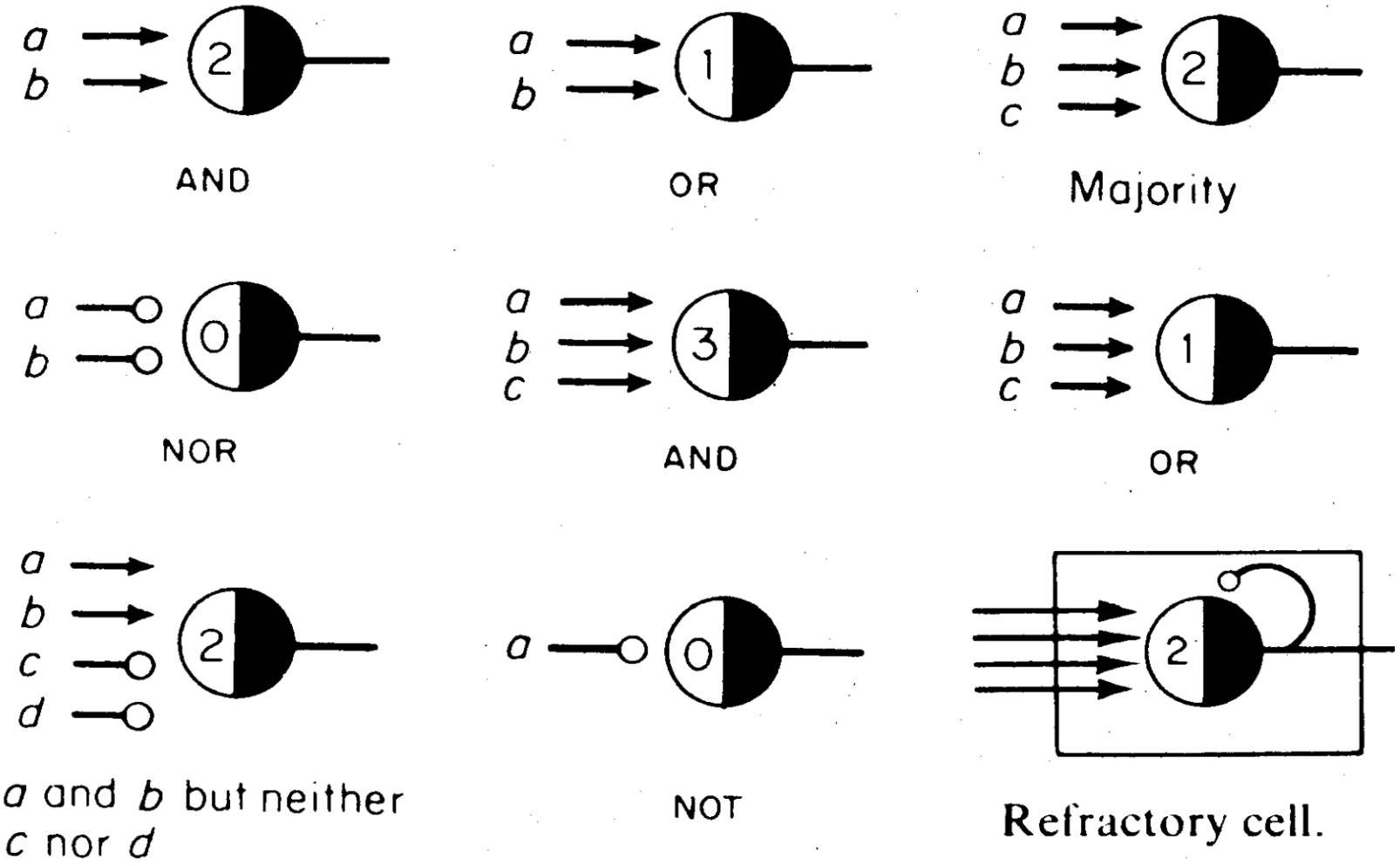
- N inputs $x_1, x_2, ..., x_N$ where $N \geq 1$, $x_i \in \{0, 1\}$
- 1 output y, where $y \in \{0, 1\}$
- threshold θ, where θ is an integer
- N weights $w_1, w_2, ..., w_N$
  where w_i is the weight of x_i
- $w_i \in \{-1, 1\}$ inhibitory -1, shown as a circle
  or excitatory 1, shown as an arrow

- $y(x) = 1$
  iff
  $\sum_i (x_i * w_i) \geq \theta$

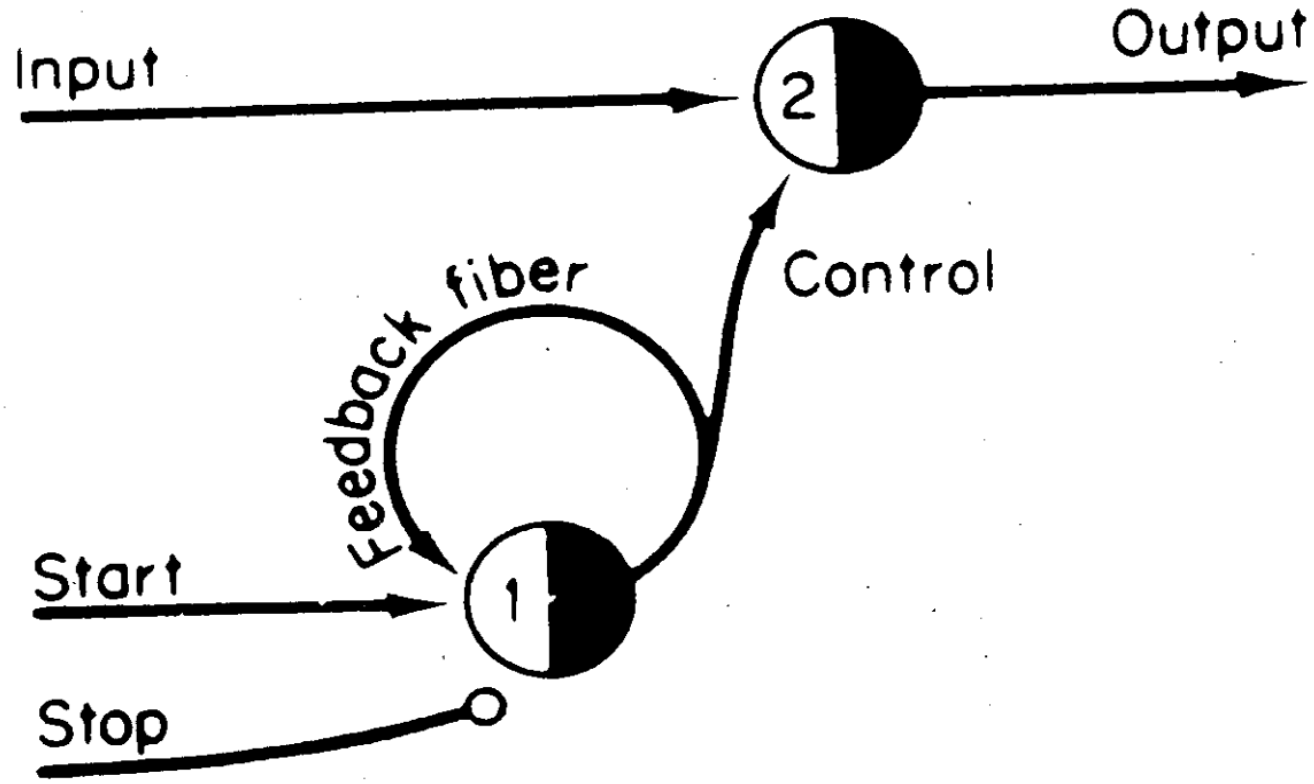


x1 and (not x2)

Some McCulloch-Pitts cells



Gating network with memory

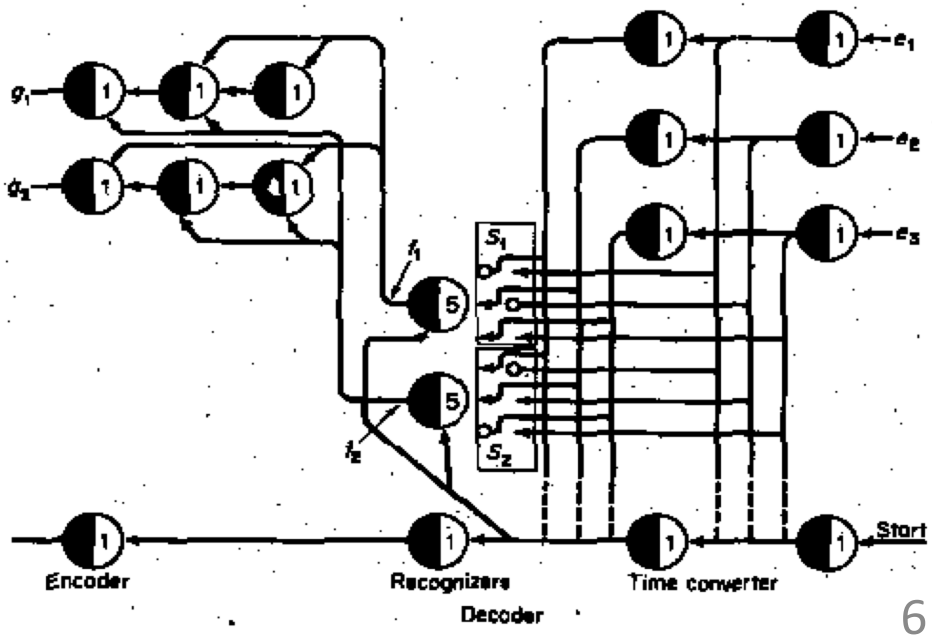


Stimulus-response decoder

	e_1		x	should yield	g_1		x	x	R_1
S_1	e_2	x			g_2	x		x	
	e_3	x	x						

and

	e_1	x		should yield	g_1	x	x		R_2
S_2	e_2	x	x		g_2		x	x	
	e_3		x						

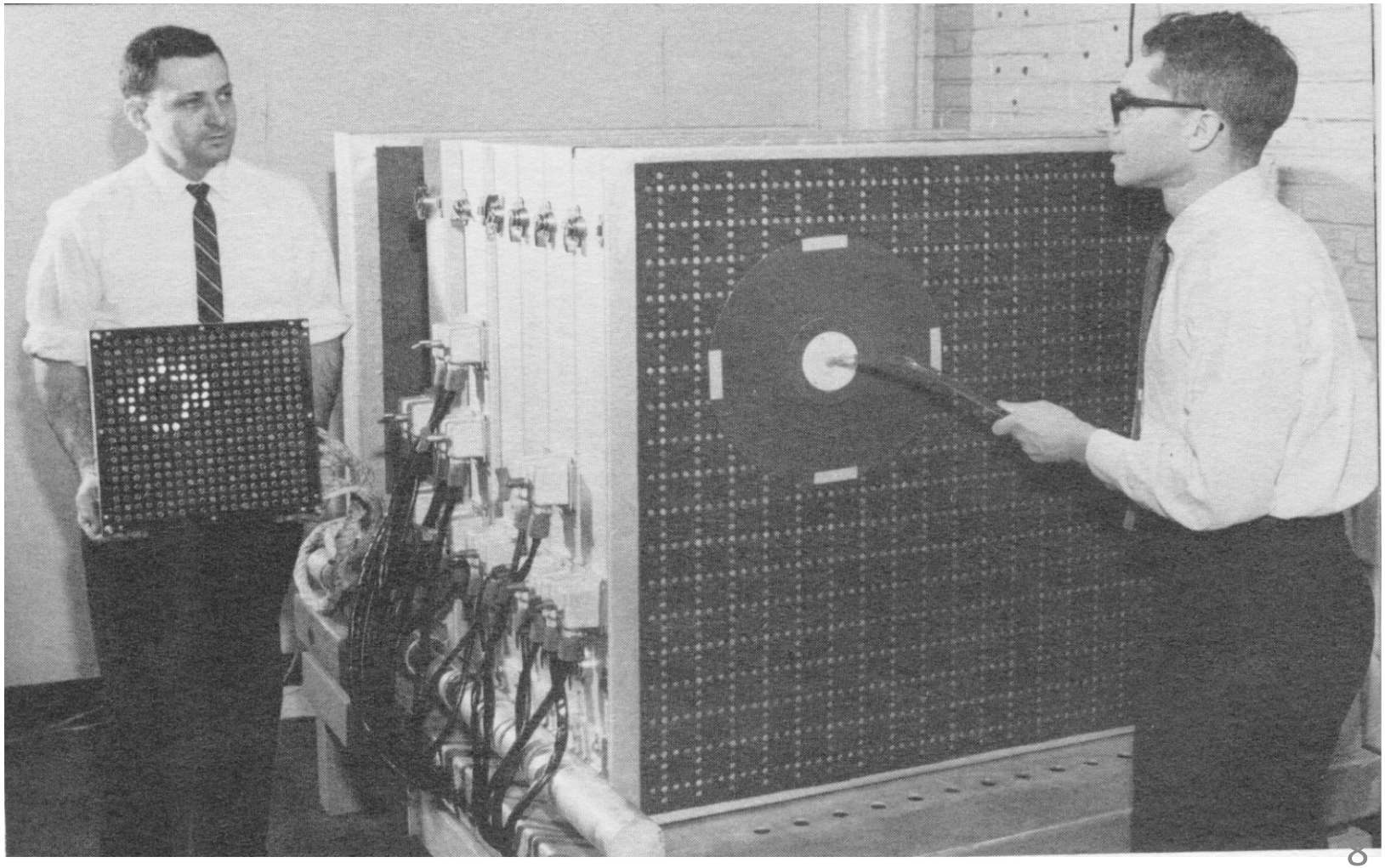


Rosenblatt – 1950's – Perceptron

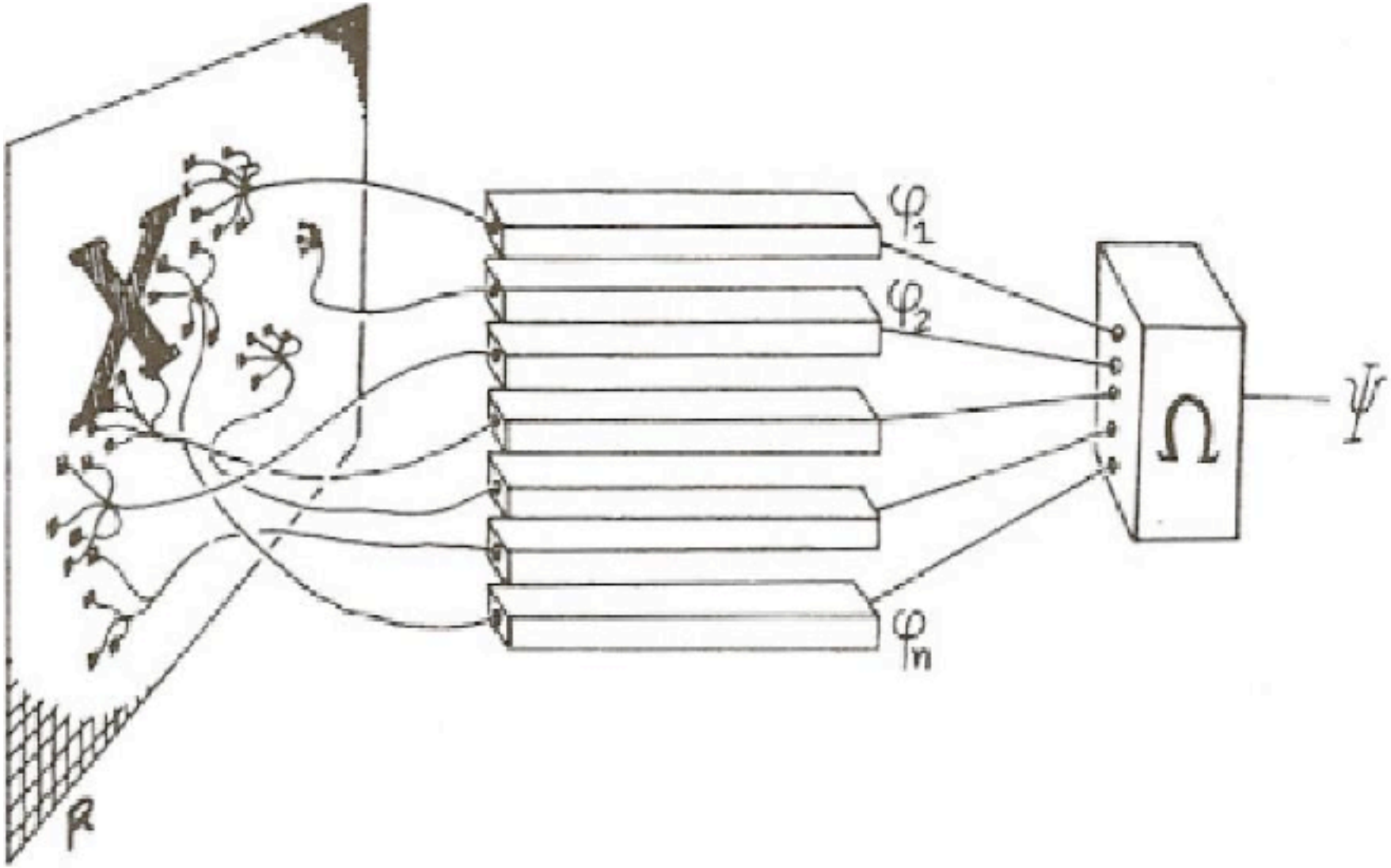
- Major diffs to McCulloch-Pitts cells
  - weights not fixed
  - random interconnections
  - network changes with time – learn from experience

- Perceptron
  - Sensory grid – Associator units – Response units.
  - Retinal pre-processing produces a set of *n real numbers called a pattern.*
  - Pattern recognition device not for specific patterns, but having the ability to recognise a set of patterns after a finite number of trials.
  - A pattern is a vector in *n-dimensional (Euclidean) pattern space.*
  - Points in pattern space are grouped into regions separated by decision surfaces.
    - Needs the assumption that nearby points in the pattern space belong to the same category (this is valid for most physical systems).

Rosenblatt and his Perceptron

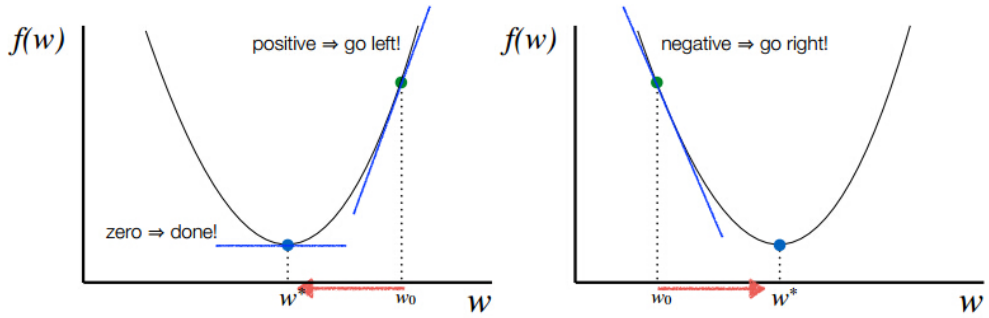


Rosenblatt's Perceptron



Gradient descent

- For a function f($\boldsymbol{w}$) the partial derivative $\frac{\partial}{\partial w_i}$f($w_i$) tells us how $f(\boldsymbol{w})$ changes as only the variable w_i changes at point $\boldsymbol{w}$
- This tells us how to change w_i to minimise $f(\boldsymbol{w})$
- Decrease $f(\boldsymbol{w})$ by moving in the opposite direction (downhill) of the partial derivative
- Gradient descent updates the parameter vector $\boldsymbol{w}$, using the partial derivatives of the error function: $\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E\left(\boldsymbol{x}, \boldsymbol{w}^{(\tau)}\right)$
- Stochastic gradient descent: update $\boldsymbol{w}$ after every input vector $\boldsymbol{x}$

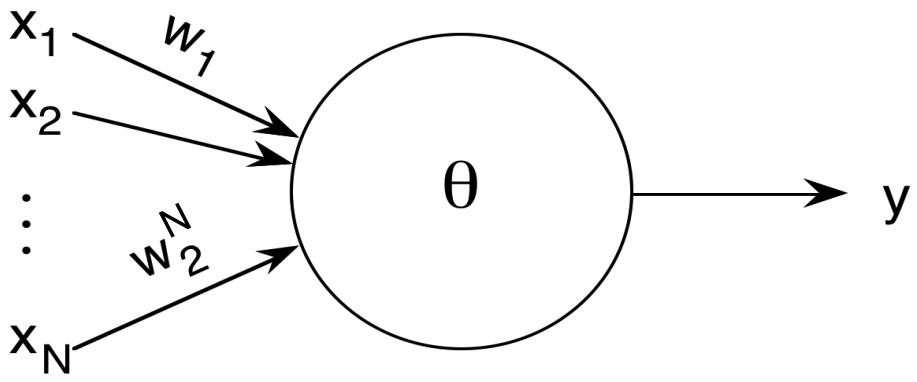


Computational model of Perceptron

- Weights are real values and learned from the data
- Targets $t \in \{-1, 1\}$ $y = -1$ iff $\sum_{i \epsilon N}(w_i x_i) < \theta, 1$ otherwise
- Loss function based on misclassified patterns only to eliminate the non-differentiable step in the threshold function:

$$E(\boldsymbol{x}, \boldsymbol{w}) = -\left(\sum_{i \epsilon N}(w_i x_i) - \theta\right) t$$

$$\frac{\partial}{dw_i} E(\boldsymbol{x}, \boldsymbol{w}) = -x_i t$$



Computational model of Perceptron

Algorithm

1. Set $w^{(\tau)}$, $\theta^{(\tau)}$ at time $\tau = 0$, to small random values

2. Present input

   X = $x_1, x_2, \ldots, x_N$

   & target output t

3. Calculate actual output $y = \sum_{i \epsilon N}(w_i x_i) - \theta$

4. Calculate error $E(\boldsymbol{x}, \boldsymbol{w}) = -\left(\sum_{i \epsilon N}(w_i x_i) - \theta\right) t$

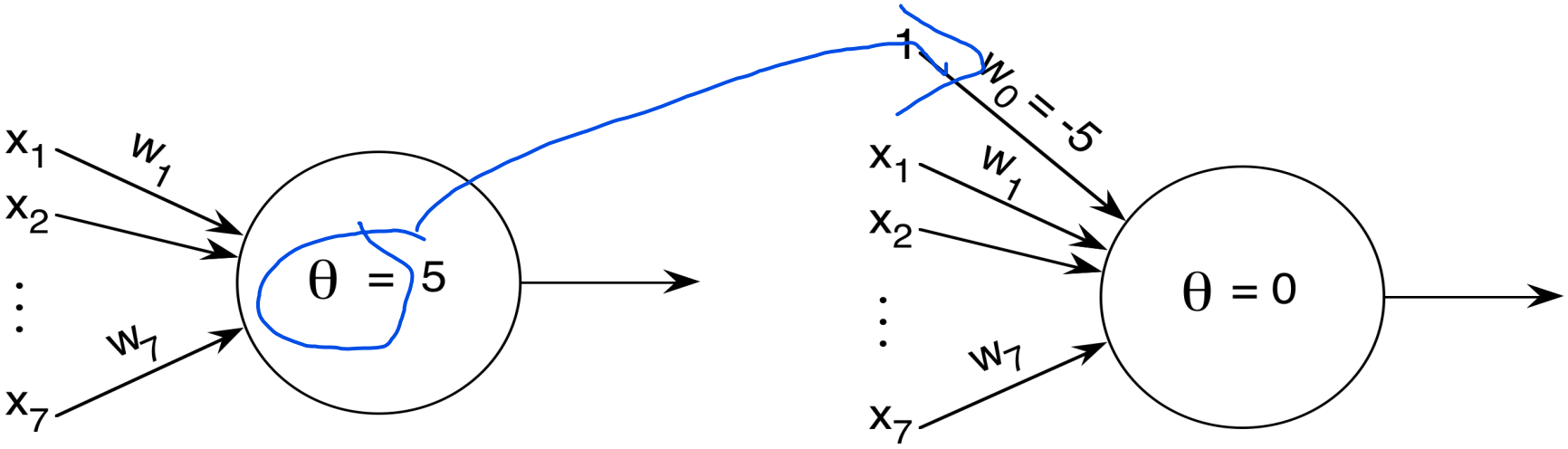
5. Adapt weights using stochastic gradient descent

   (τ is timesteps) $w_i^{(\tau+1)} = w_i^{(\tau)} - \eta(-x_i t)$ 更新值

6. Go to step 2. $0 < \eta < 1$ (η usually small, e.g. 0.01)

Eliminating thresholds

- These 2 systems behave identically in use:



- Threshold: –ve weight for input always on
  - Benefit: during training the threshold (or *bias*) *learns just like the rest of the weights.*

PERCEPTRONS

Minsky, M &Papert, S, MIT Press, 1969. Pg 231:

> "The perceptron has shown itself to be worthy of study despite (and even because of) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting 'learning theorem' for the mulitlayered machine will be found."

What Perceptrons can not do

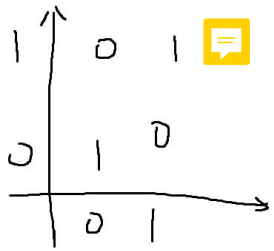
- Input Desired

  (0, 0) 1

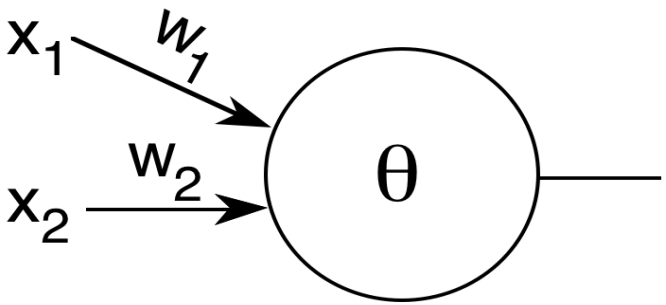
  (0, 1) 0

  (1, 0) 0

  (1, 1) 1



- Each component of the input taken separately provides no evidence about the correct output
- Each input-output case leads to an inequality constraint on the weights:



- It is impossible to satisfy these inequalities, hence no set of weights can be found to do this task.

$$0 + 0 > \theta$$

$$0 + w_2 < \theta$$

$$w_1 + 0 < \theta$$

$$w_1 + w_2 > \theta$$