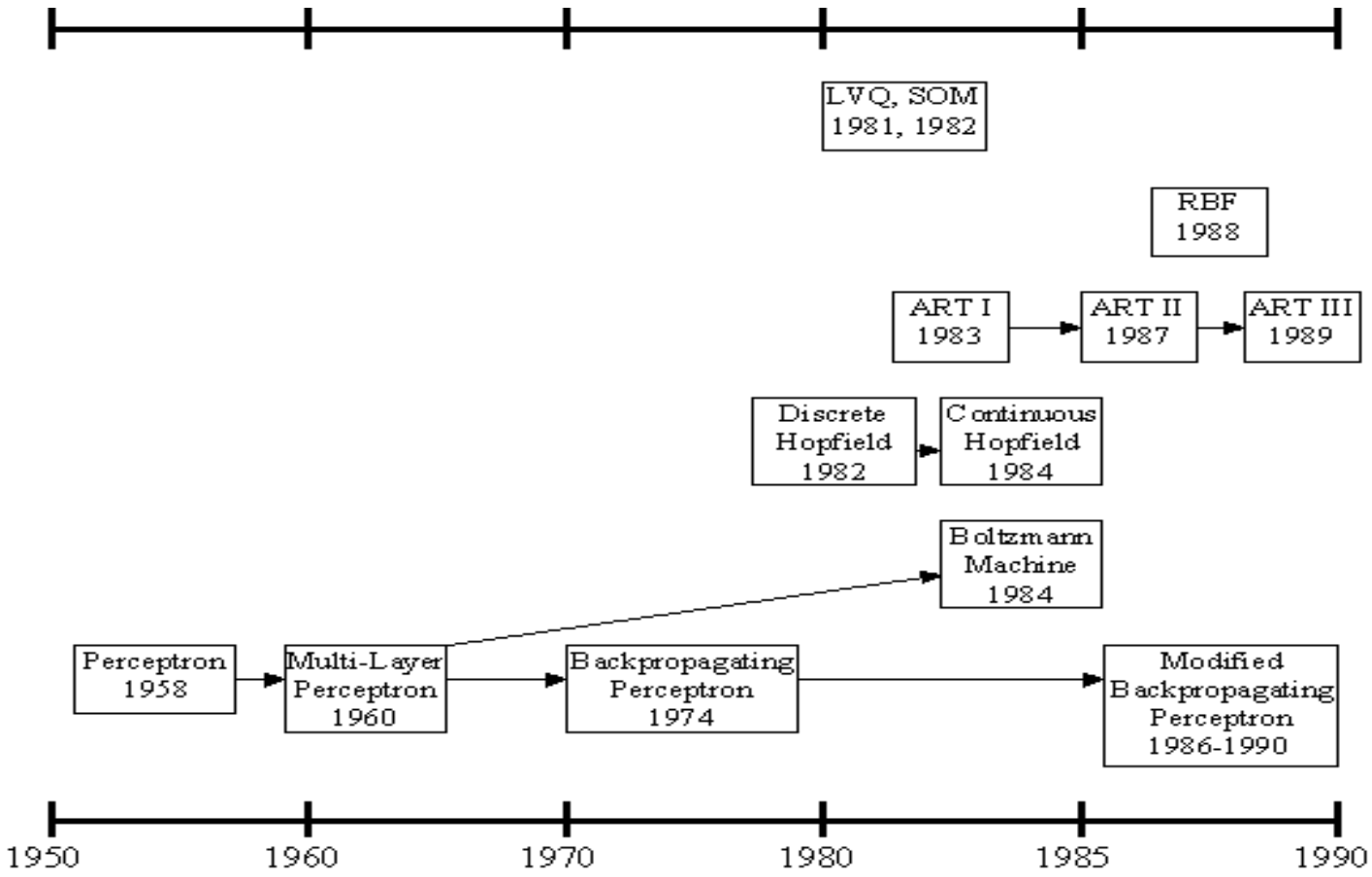
Artificial Neural Network

Lecture 23

Dr. Tamal Ghosh
Associate Professor
Department of CSE | Adamas University

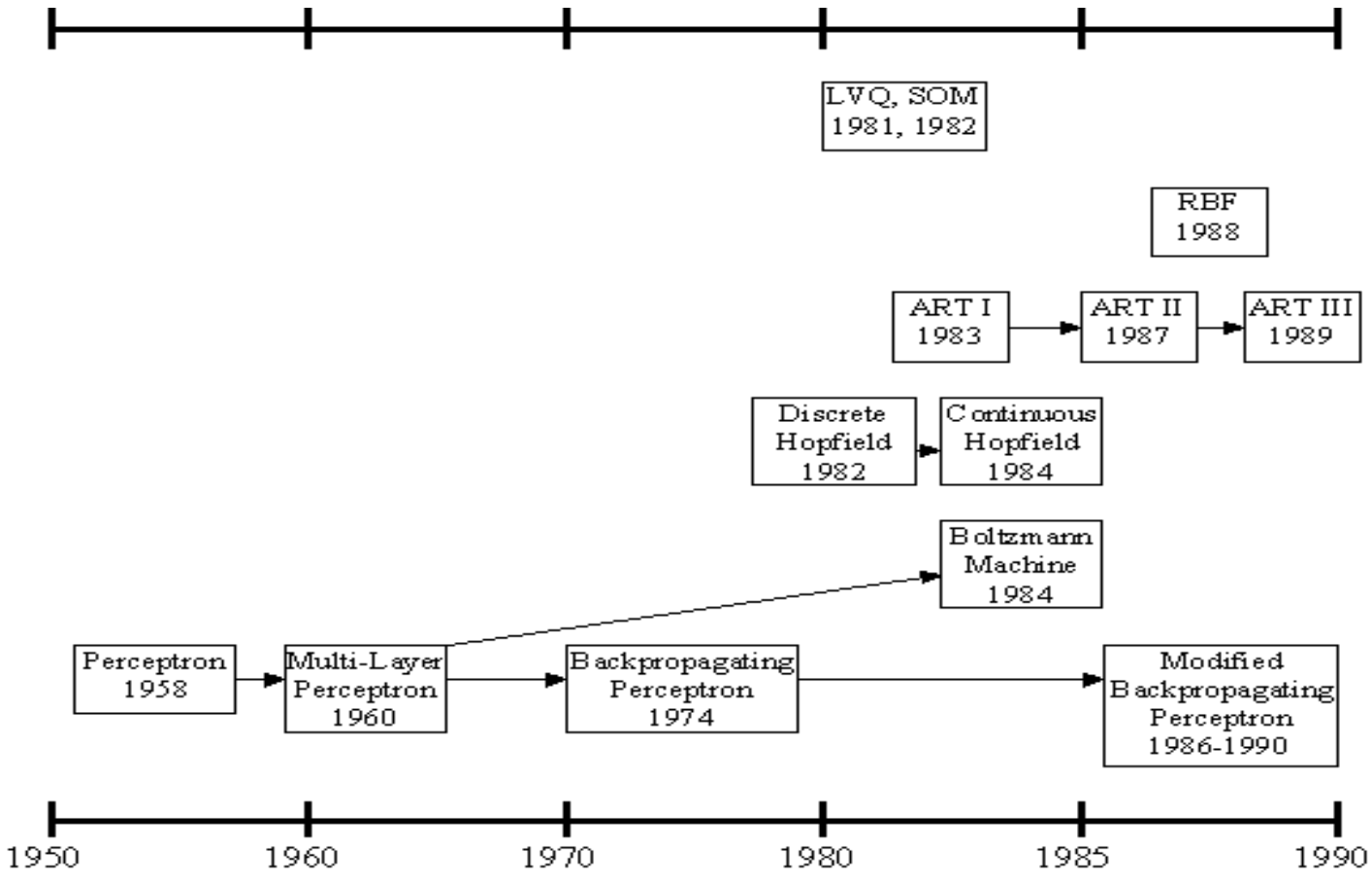
History of the Artificial Neural Networks

- history of the ANNs stems from the 1940s, the decade of the first electronic computer.
- However, the first important step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. After this, the development of ANNs has proceeded as described in *Figure*.



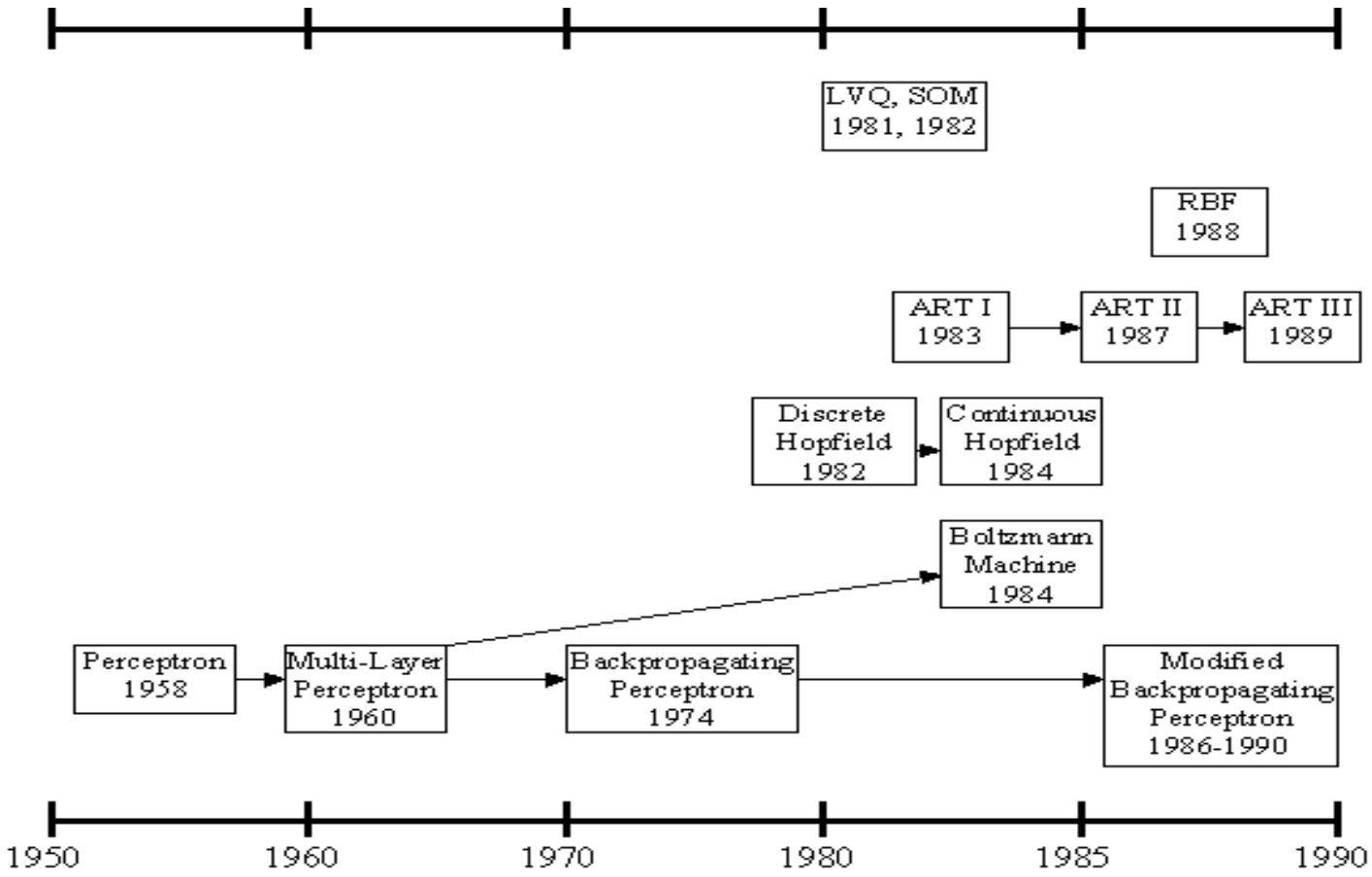
History of the Artificial Neural Networks

- in 1986, The application area of the MLP networks remained rather limited until the breakthrough when a general back propagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland.
- in 1982, Hopfield brought out his idea of a neural network. Unlike the neurons in MLP, the Hopfield network consists of only one layer whose neurons are fully connected with each other.



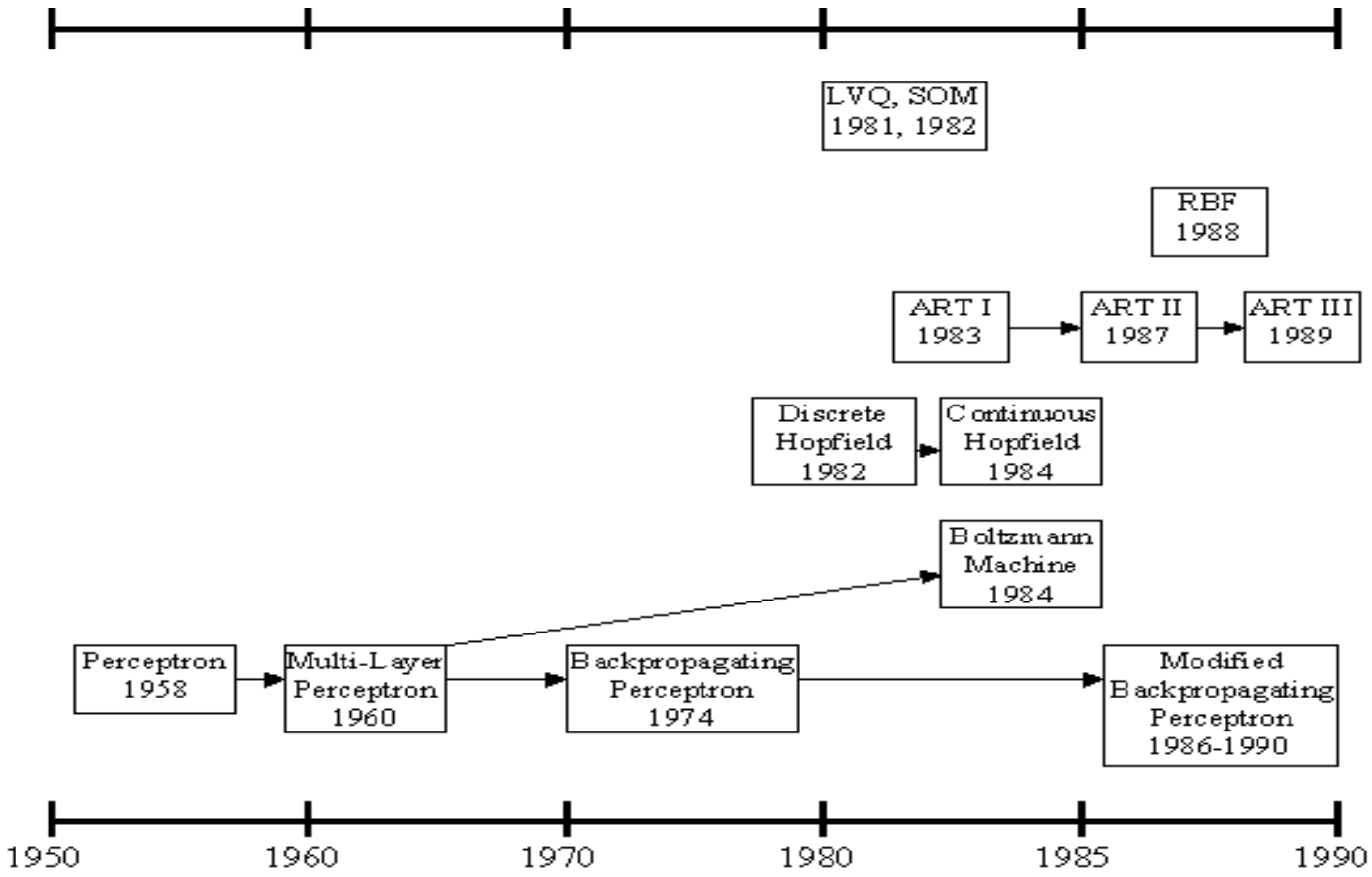
History of the Artificial Neural Networks

- history of the ANNs stems from the 1940s, the decade of the first electronic computer.
- However, the first important step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. After this, the development of ANNs has proceeded as described in *Figure*.



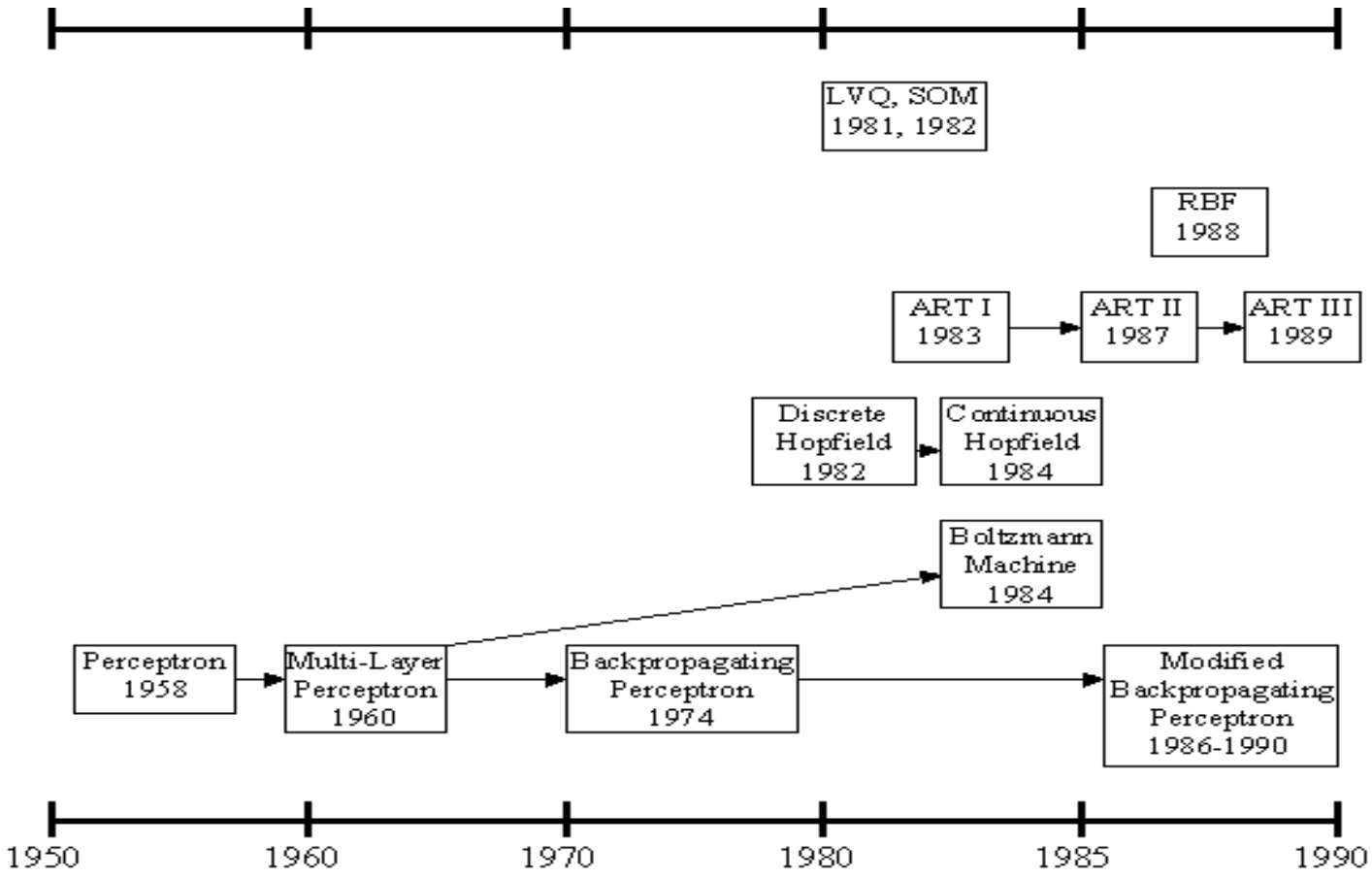
History of the Artificial Neural Networks

- Since then, new versions of the Hopfield network have been developed. The Boltzmann machine has been influenced by both the Hopfield network and the MLP.



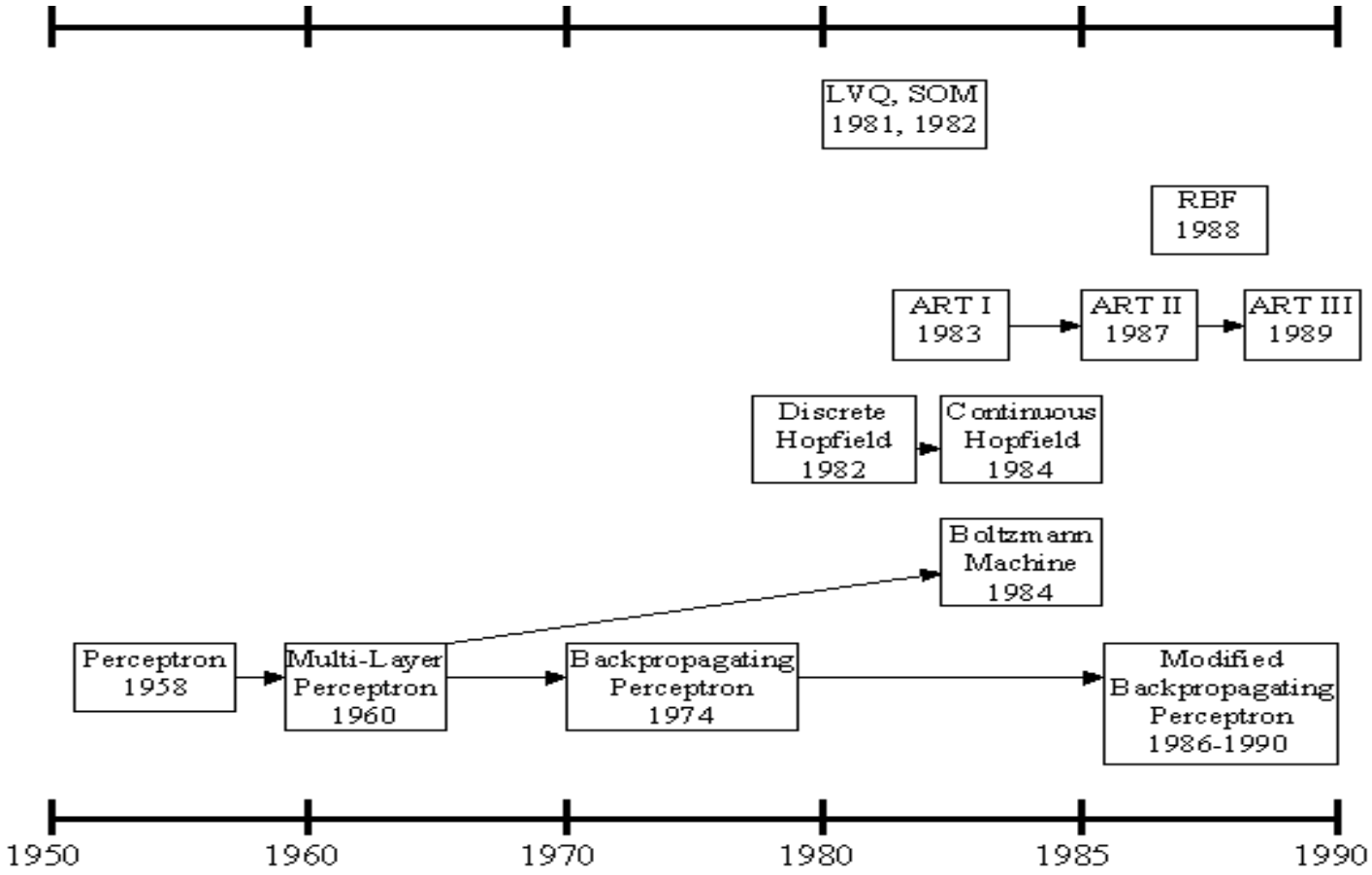
History of the Artificial Neural Networks

- in 1988, Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe. Although the basic idea of RBF was developed 30 years ago under the name method of potential function, the work by Broomhead & Lowe opened a new frontier in the neural network community.



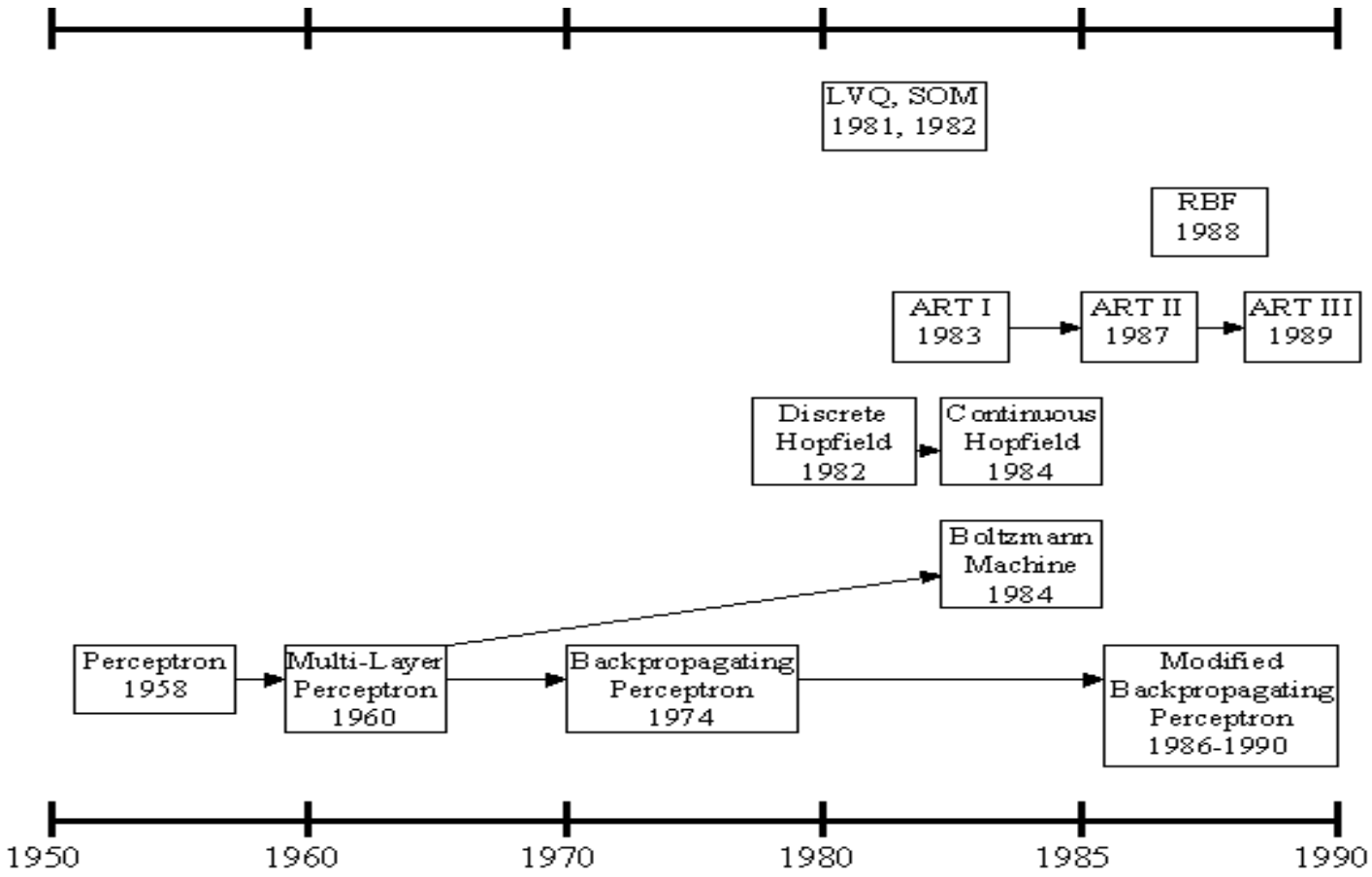
History of the Artificial Neural Networks

- in 1982, A totally unique kind of network model is the Self-Organizing Map (SOM) introduced by Kohonen. SOM is a certain kind of topological map which organizes itself based on the input patterns that it is trained with. The SOM originated from the LVQ (Learning Vector Quantization) network the underlying idea of which was also Kohonen's in 1972.



History of the Artificial Neural Networks

- Since then, research on artificial neural networks has remained active, leading to many new network types, as well as hybrid algorithms and hardware for neural information processing.



ANN

- An artificial neural network consists of a pool of simple processing units that communicate by sending signals to each other over a large number of weighted connections.

Artificial Neural Network

- A set of major aspects of a **parallel distributed model** include:
- a set of processing units (cells).
- a state of activation for every unit, which is equivalent to the output of the unit.
- Connections between the units. Generally, each connection is defined by a weight.
- a propagation rule, which determines the effective input of a unit from its external inputs.
- an activation function, which determines the new level of activation based on the effective input and the current activation.
- an external input for each unit.
- a method for information gathering (the learning rule).
- an environment within which the system must operate, providing input signals and if necessary then some error signals.

Computers vs. Neural Networks

"Standard" Computers	Neural Networks
• one CPU	highly parallel processing
•fast processing units	slow processing units
•reliable units	unreliable units
•static infrastructure	dynamic infrastructure

Why Artificial Neural Networks?

- There are two basic reasons why we are interested in building artificial neural networks (ANNs):

- **Technical viewpoint:** Some problems such as character recognition or the prediction of the future states of a system require massively parallel and adaptive processing.

- **Biological viewpoint:** ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

Artificial Neural Networks

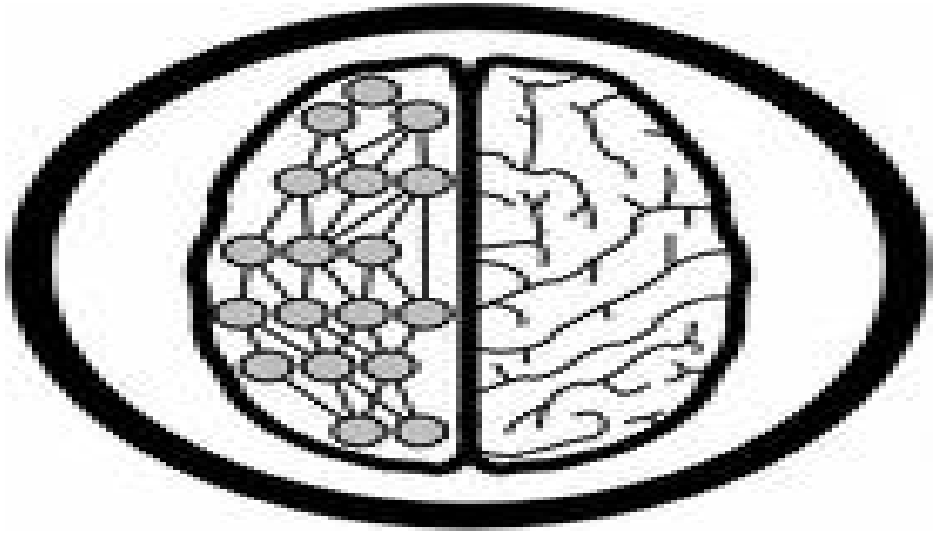
- The “building blocks” of neural networks are the **neurons**.
  - In technical systems, we also refer to them as **units** or **nodes**.
- Basically, each neuron
  - receives **input** from many other neurons.
  - changes its internal state (**activation**) based on the current input.
  - sends **one output signal** to many other neurons, possibly including its input neurons (recurrent network).

Artificial Neural Networks

- Information is transmitted as a series of electric impulses, so-called **spikes**.
- The **frequency** and **phase** of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as **10,000** other neurons.
- Usually, a neuron receives its information from other neurons in a confined area, its so-called **receptive field**.

How do ANNs work?

- An artificial neural network (ANN) is either a hardware implementation or a computer program which strives to simulate the information processing capabilities of its biological exemplar. ANNs are typically composed of a great number of interconnected artificial neurons. The artificial neurons are simplified models of their biological counterparts.
- ANN is a technique for solving problems by constructing software that works like our brains.

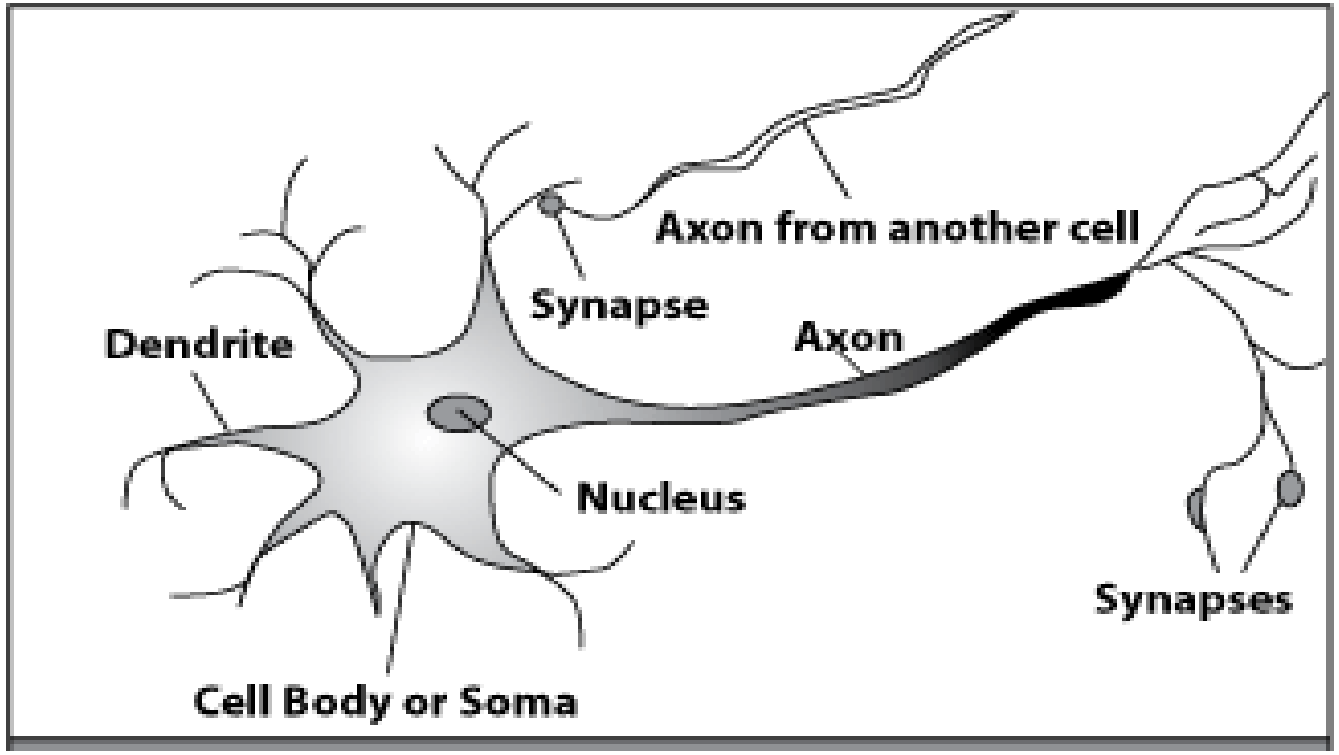


How do our brains work?

- The Brain is A massively parallel information processing system.
- Our brains are a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

How do our brains work?

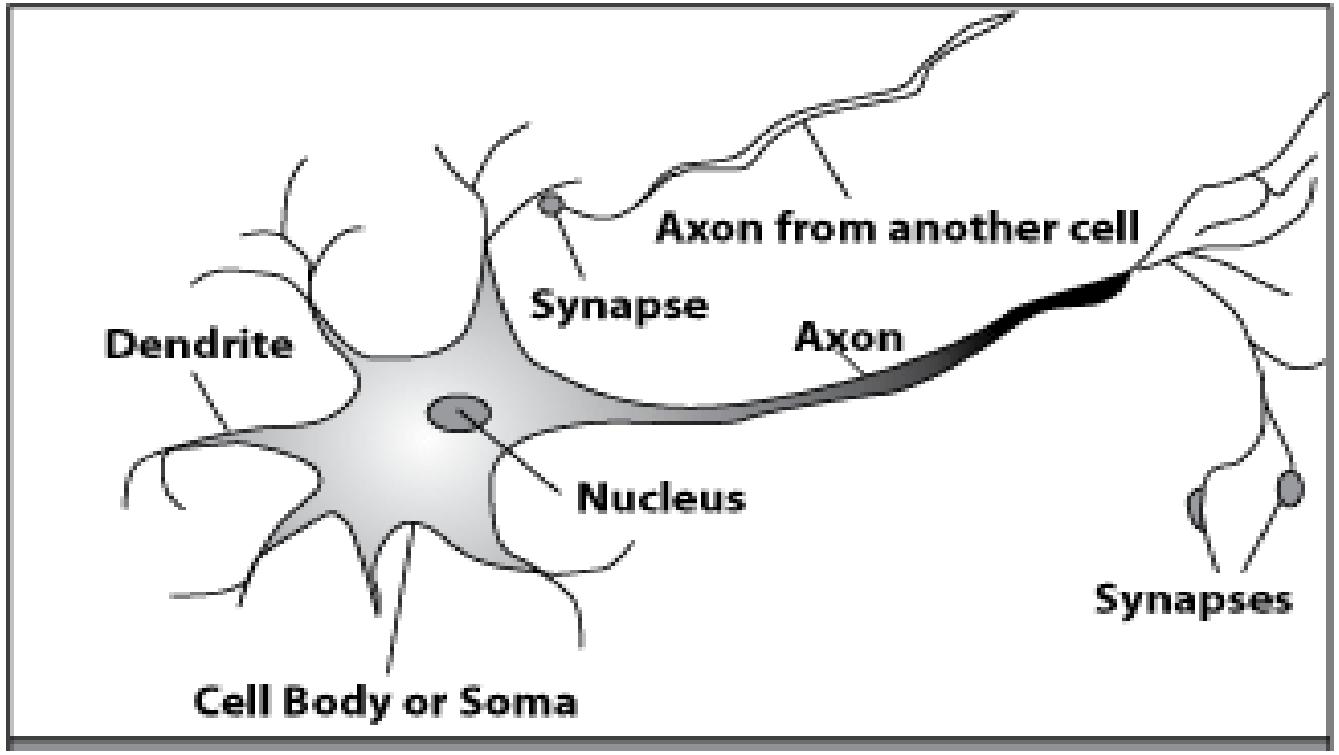
- A Neuron



Dendrites: Input
Cell body: Processor
Synaptic: Link
Axon: Output

How do our brains work?

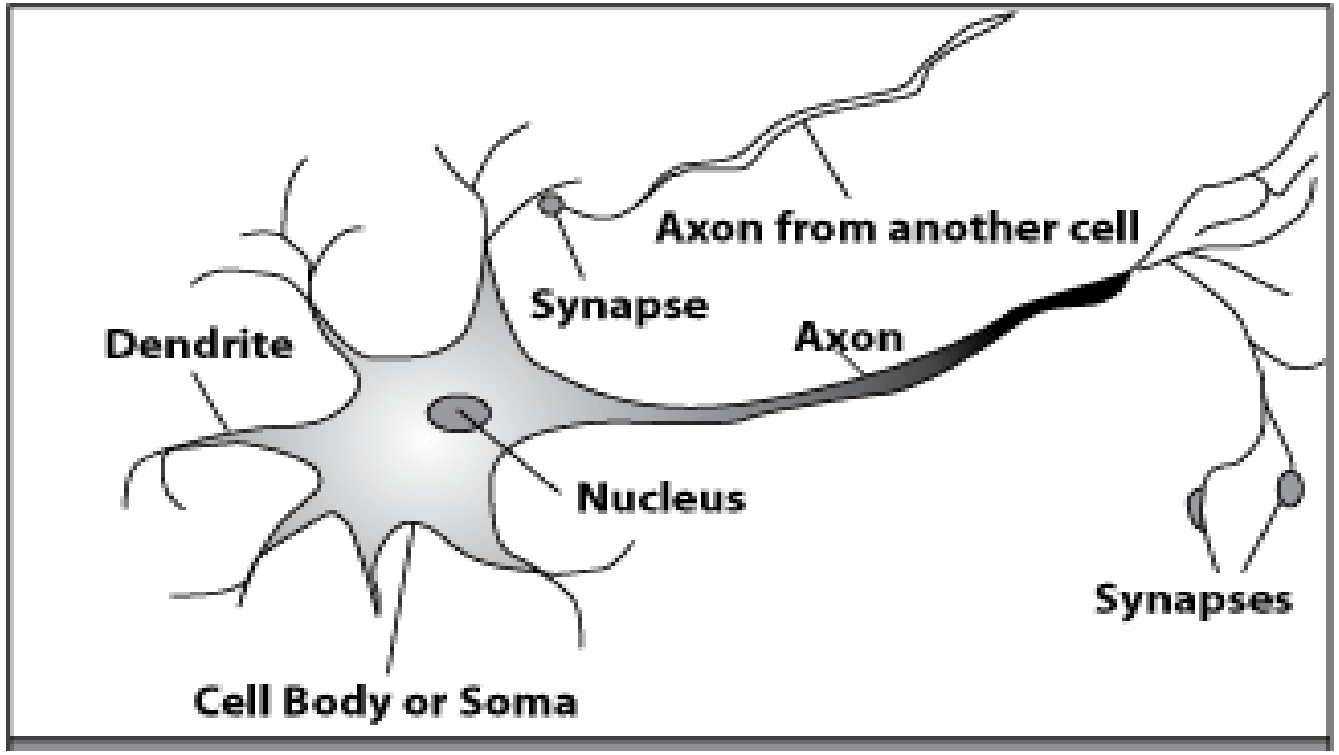
- A processing element



A neuron is connected to other neurons through about *10,000 synapses*

How do our brains work?

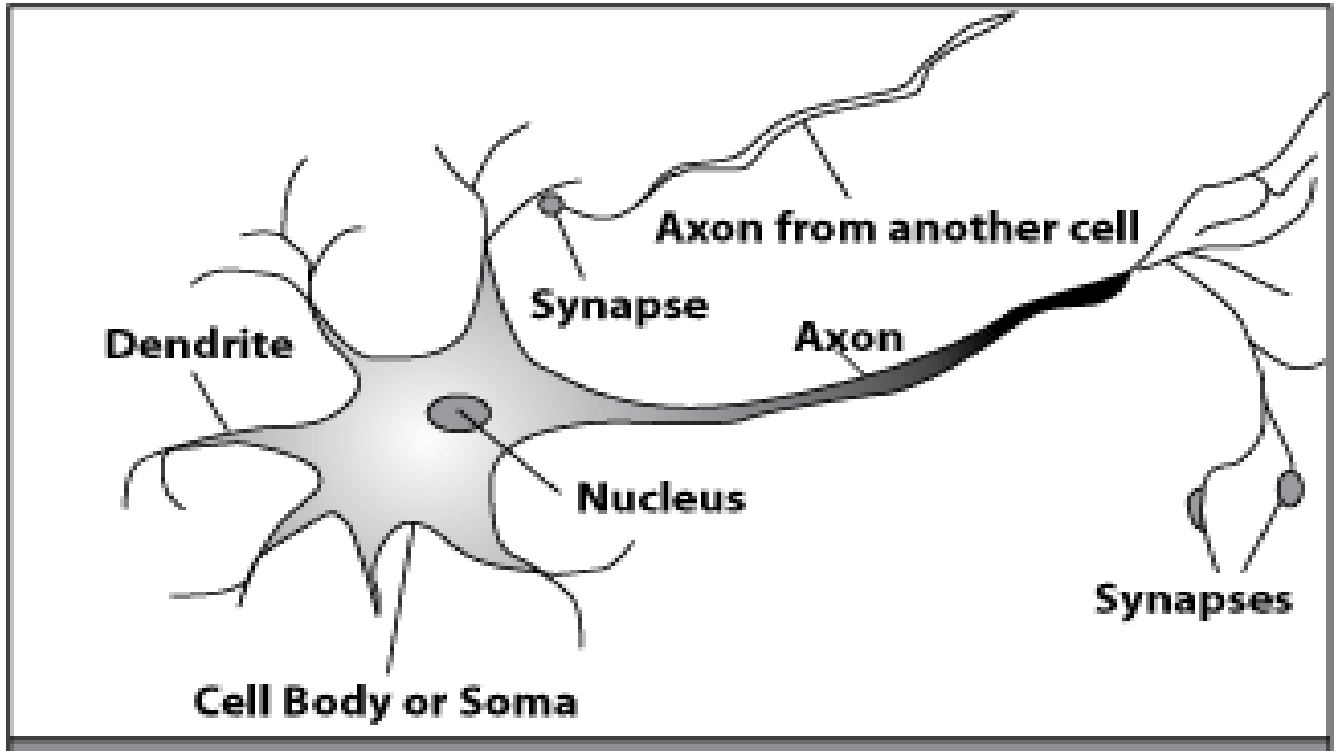
- A processing element



A neuron receives input from other neurons. Inputs are combined.

How do our brains work?

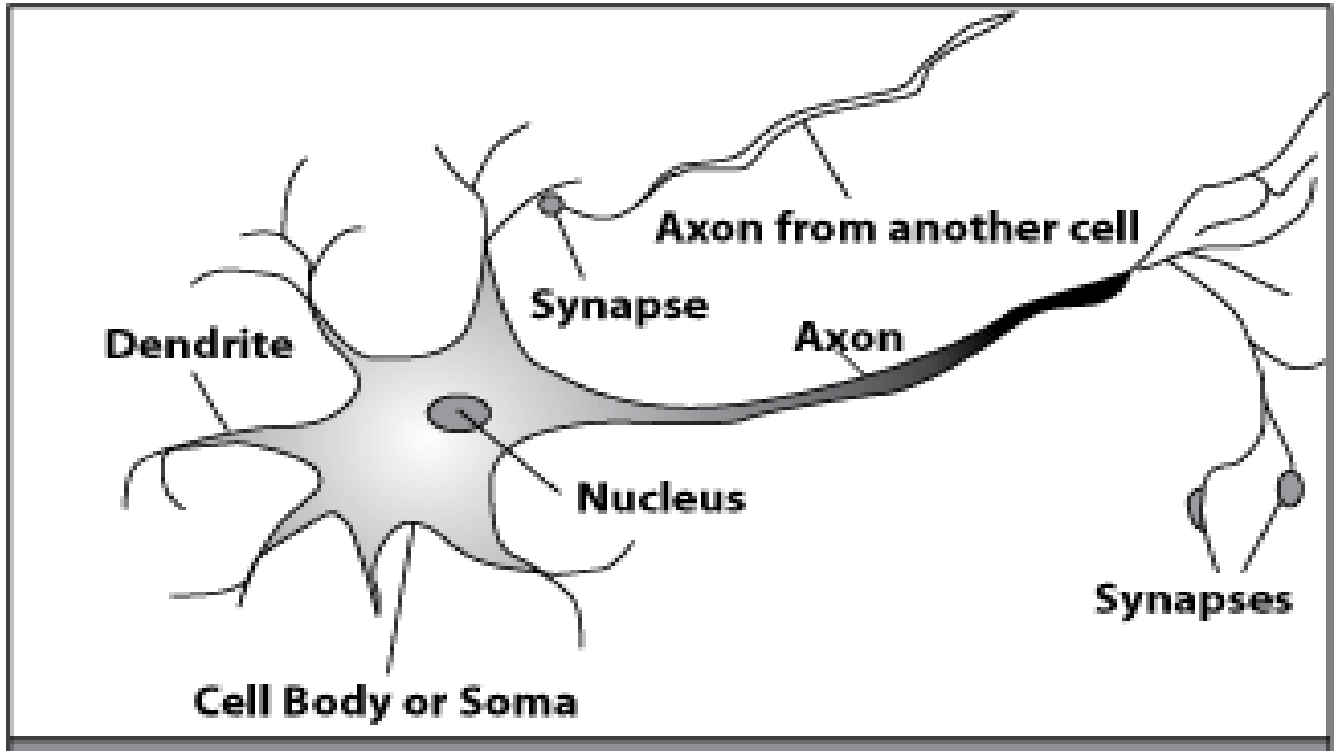
- A processing element



Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)

How do our brains work?

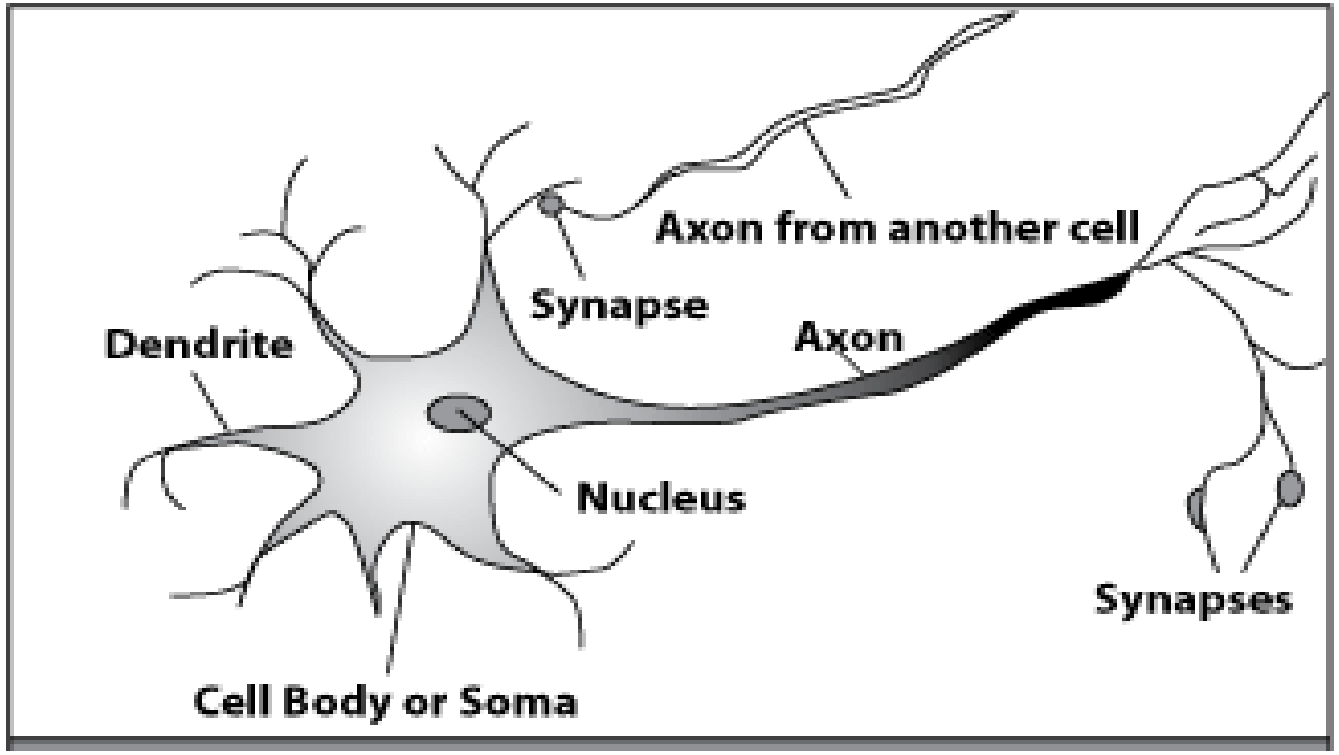
- A processing element



The axon endings almost touch the dendrites or cell body of the next neuron.

How do our brains work?

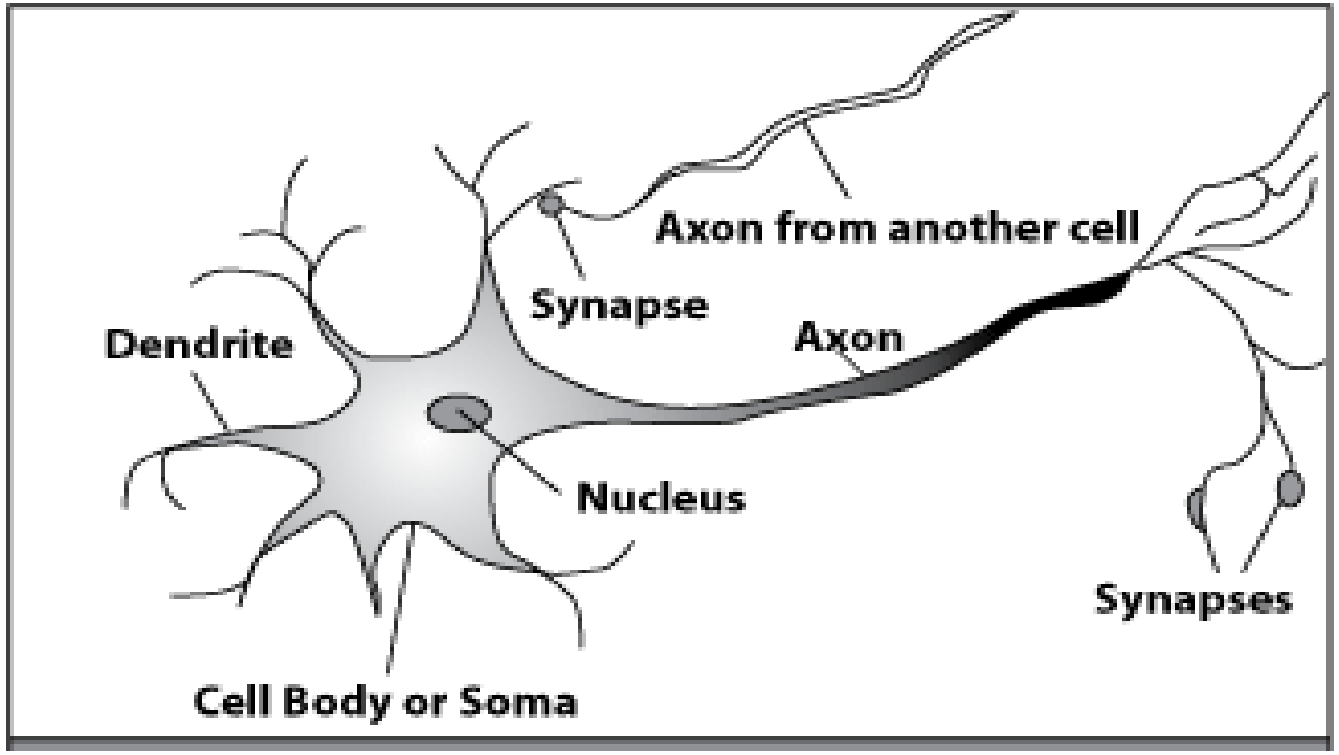
- A processing element



Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.

How do our brains work?

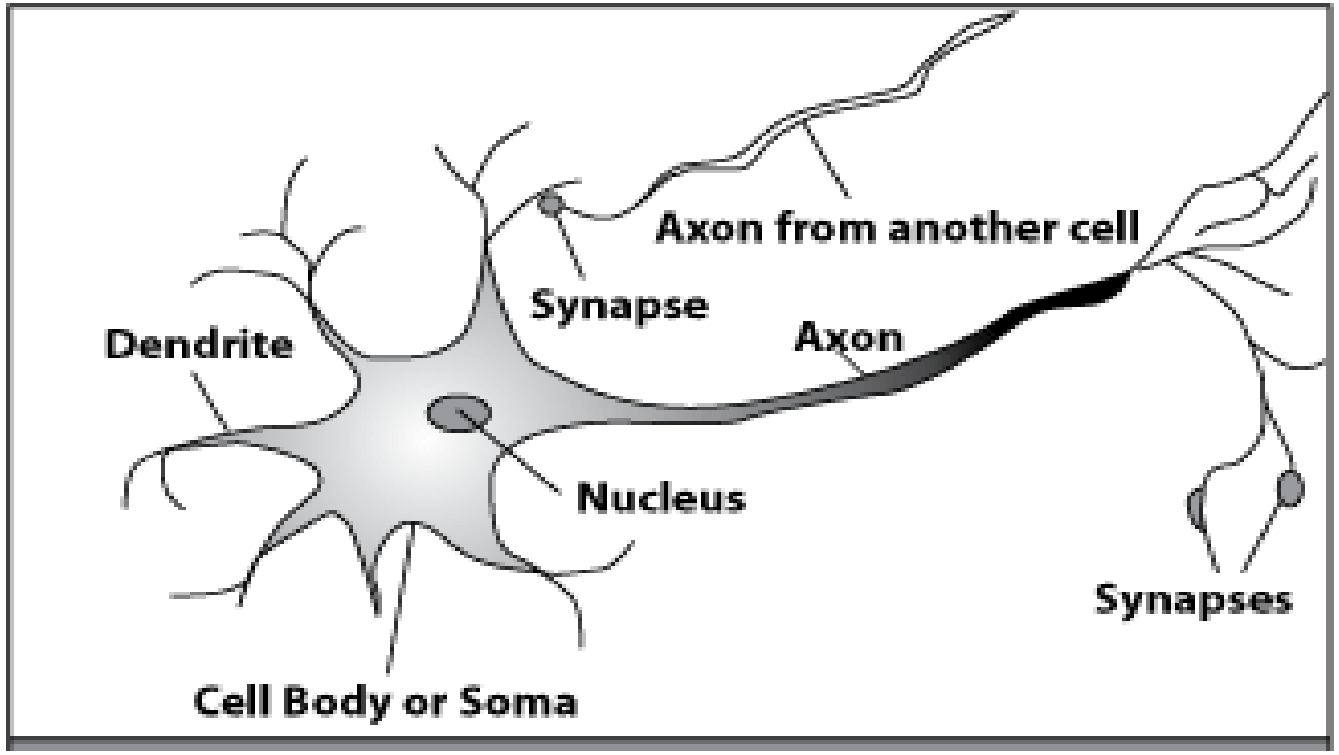
- A processing element



Neurotransmitters are chemicals which are released from the first neuron and which bind to the
Second.

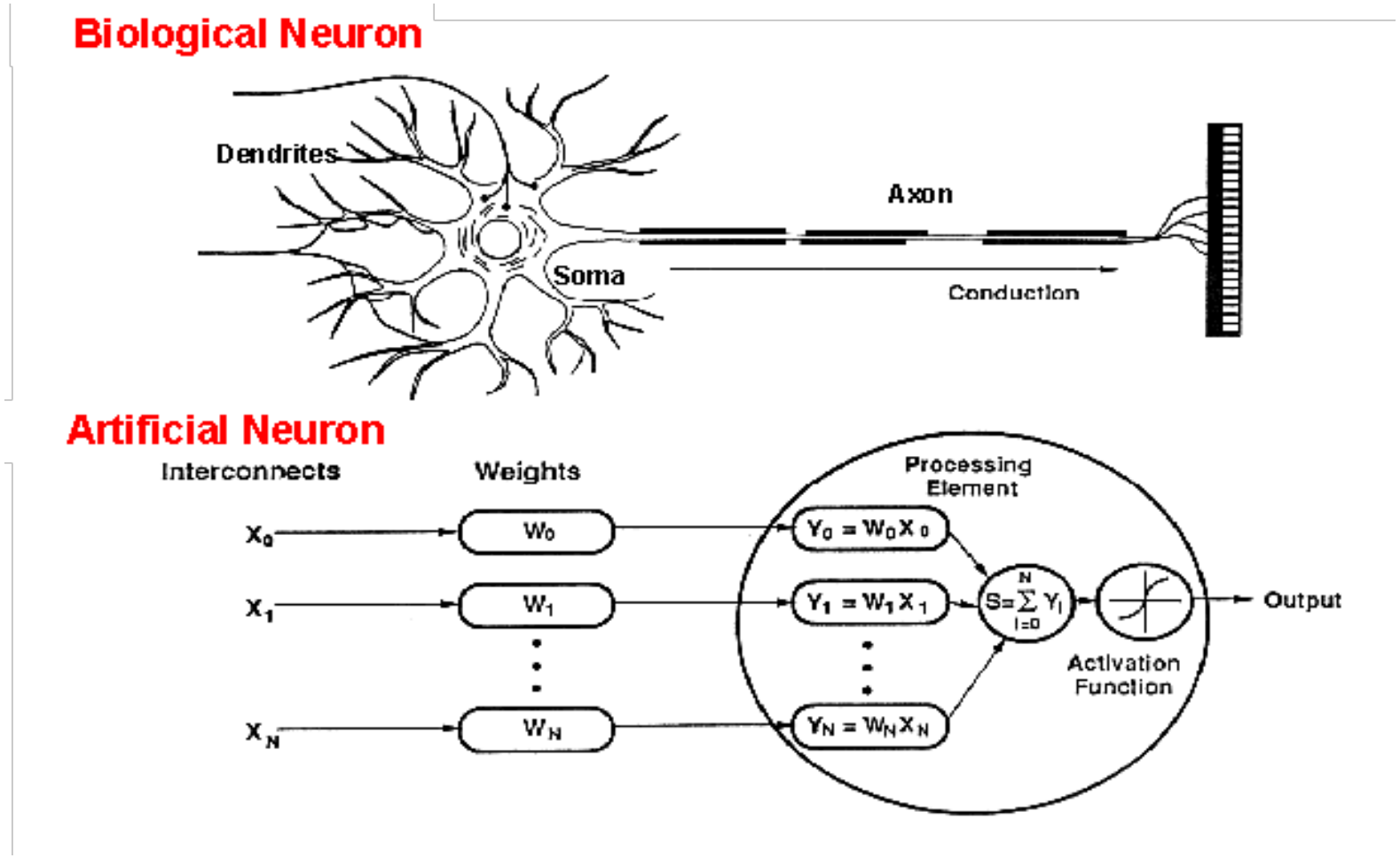
How do our brains work?

- A processing element



This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

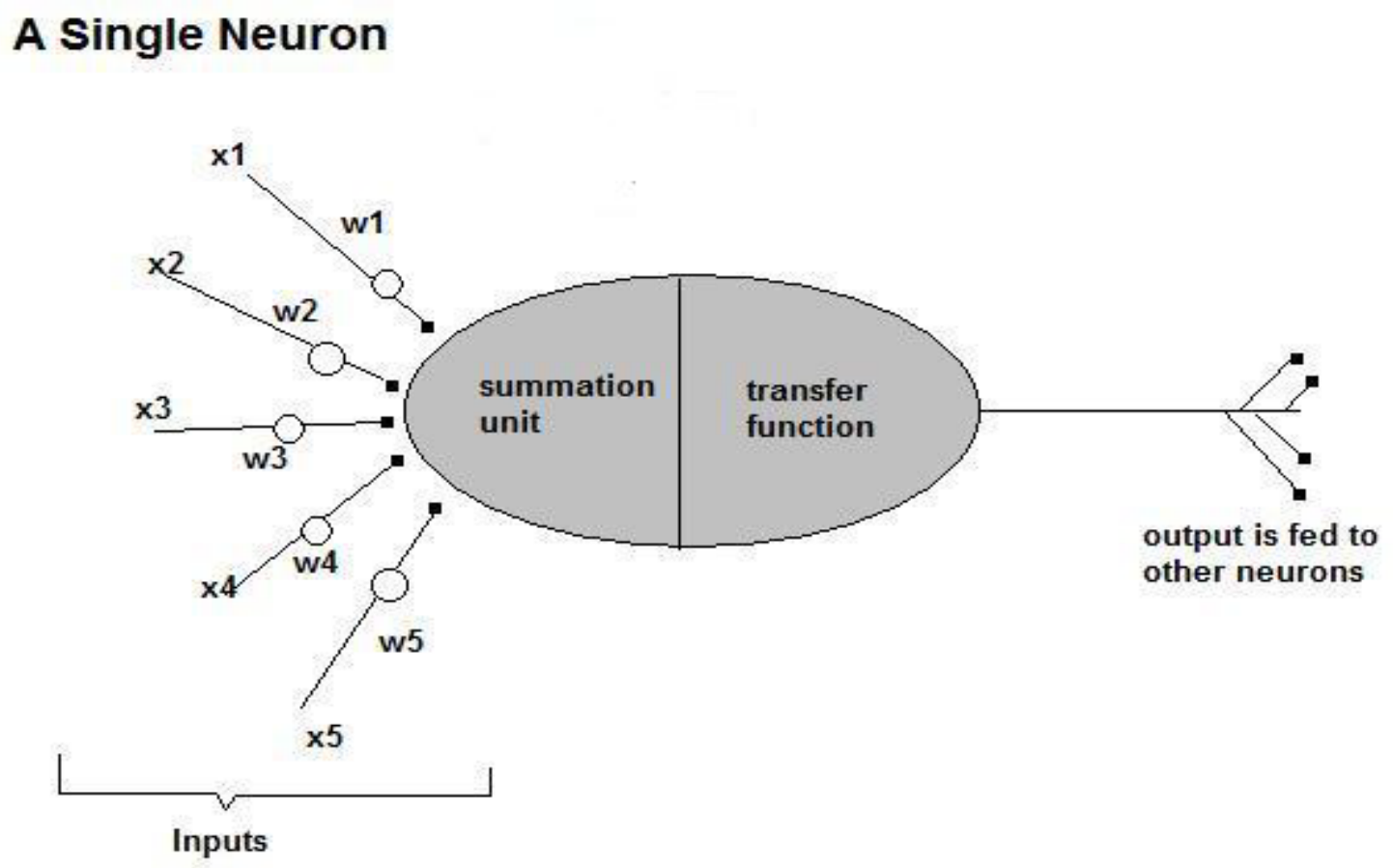
How do ANNs work?



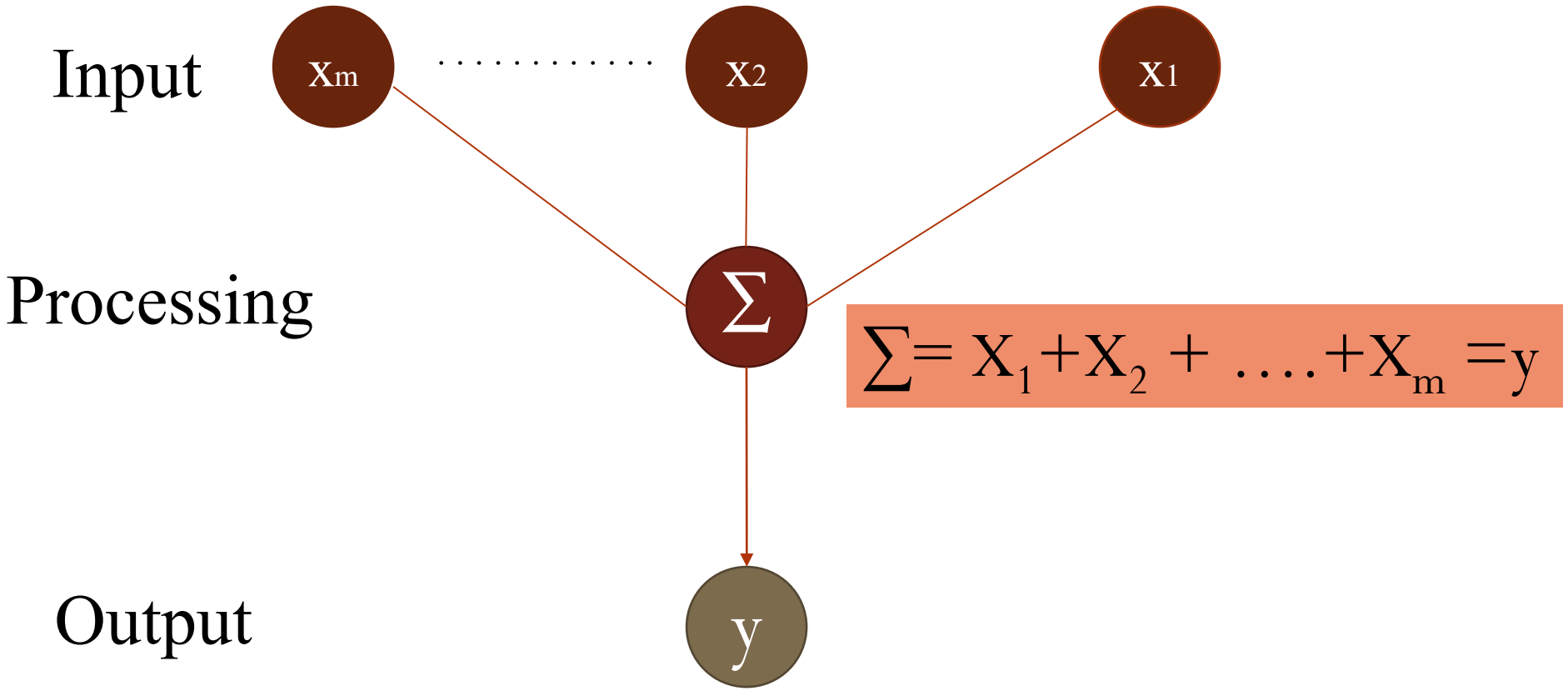
An artificial neuron is an imitation of a human neuron

How do ANNs work?

- Now, let us have a look at the model of an artificial neuron.



How do ANNs work?



How do ANNs work?

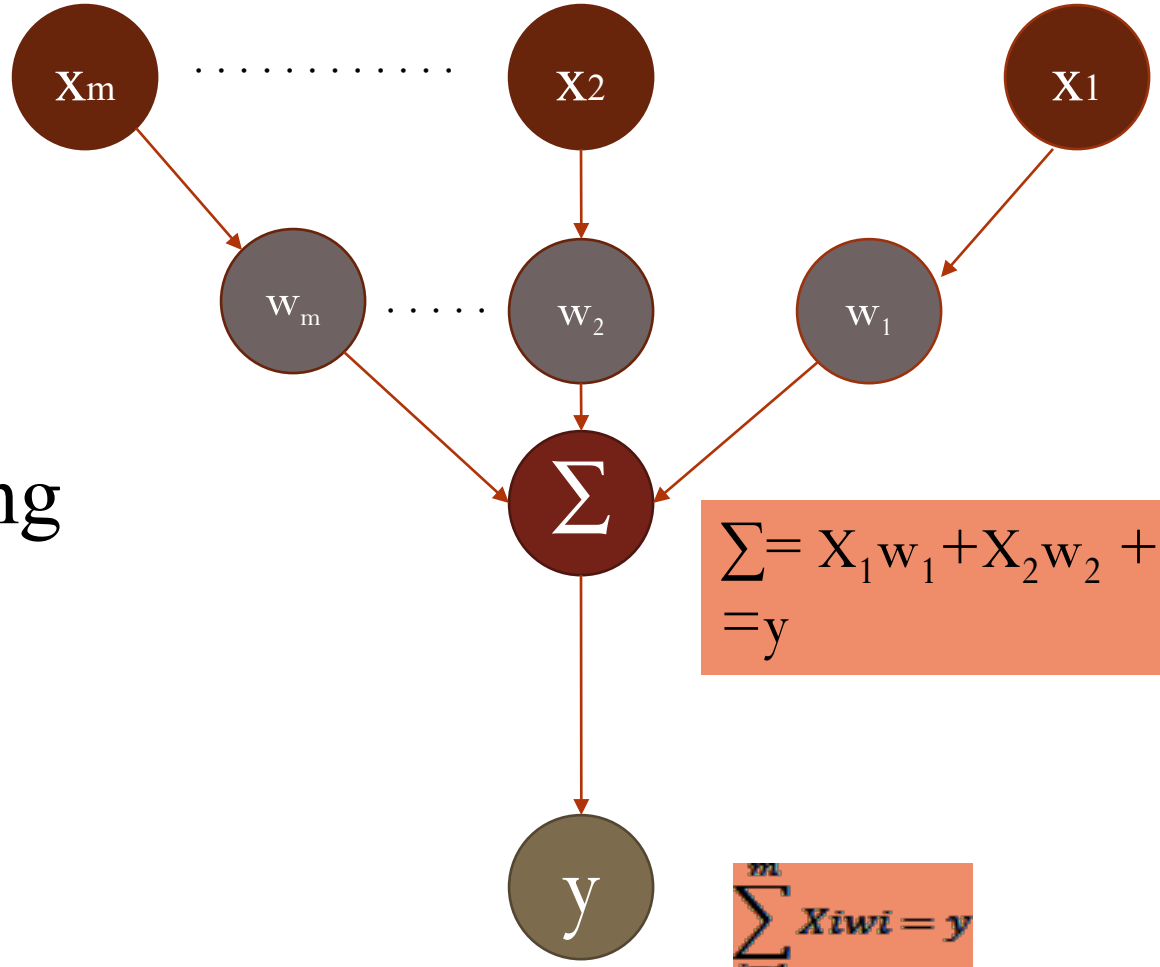
Not all inputs are equal

Input

weights

Processing

Output

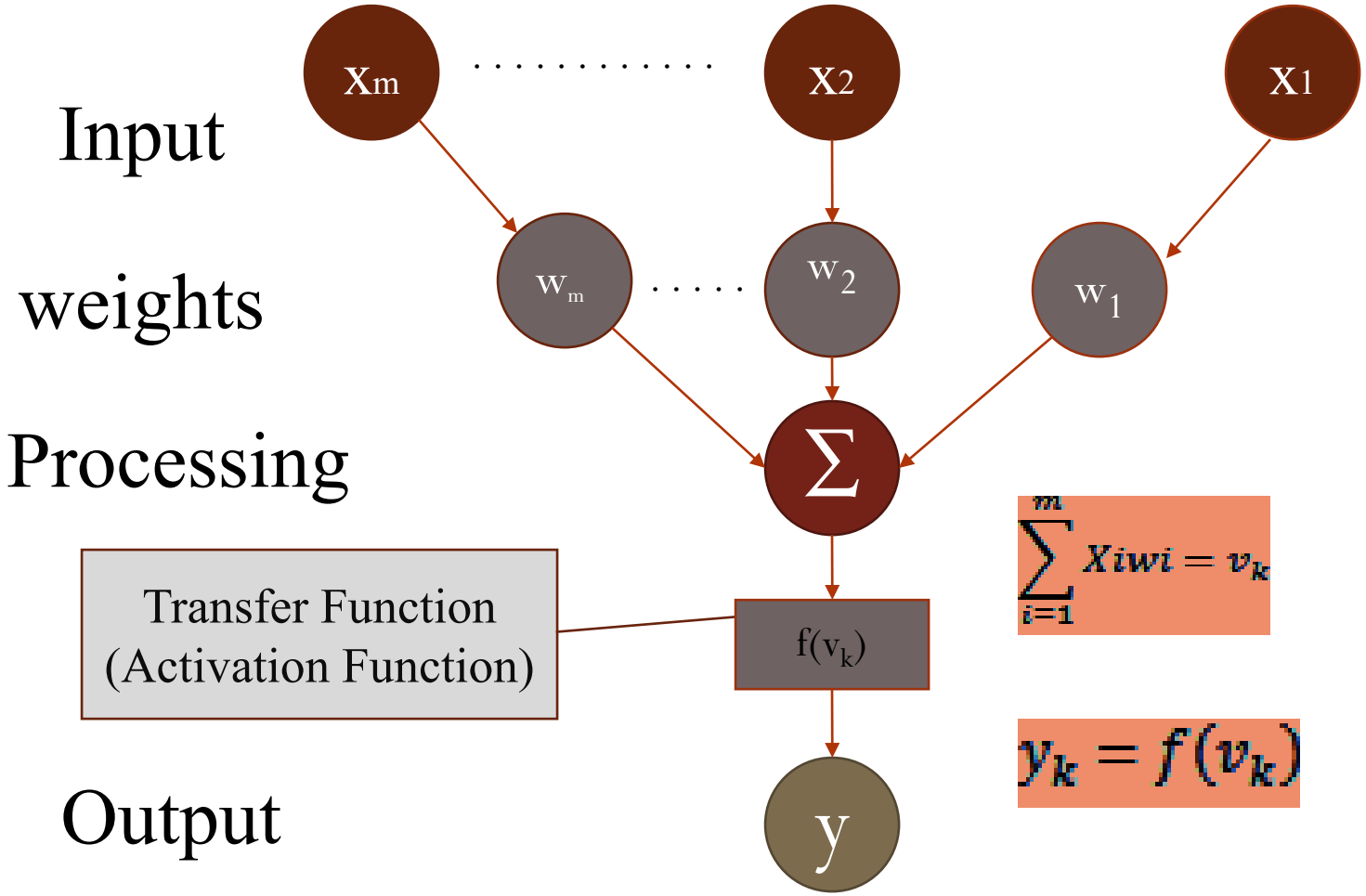


$$\sum = X_1w_1 + X_2w_2 + \ldots + X_mw_m = y$$

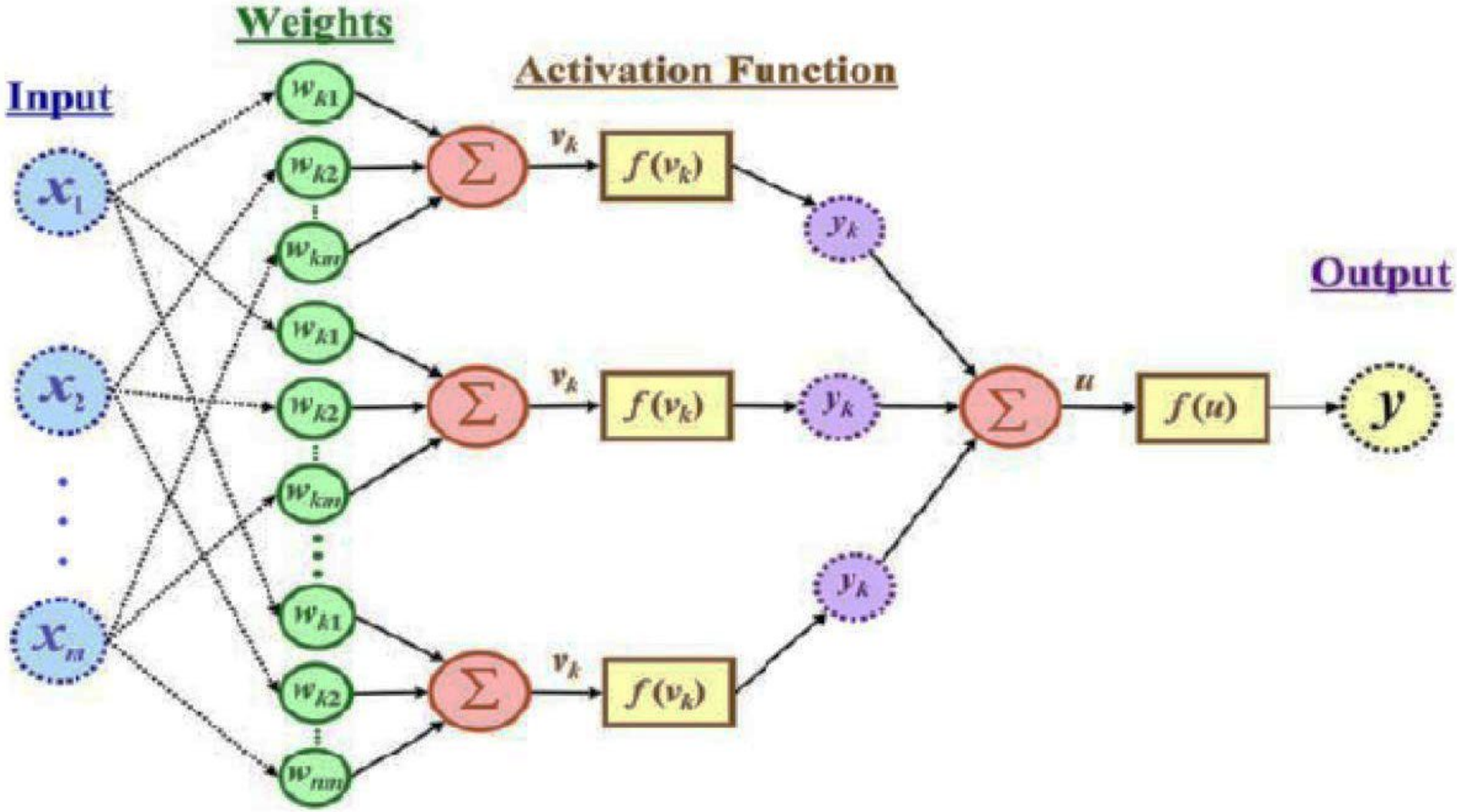
$$\sum_{i=1}^{m} Xiwi = y$$

How do ANNs work?

The signal is not passed down to the next neuron verbatim



The output is a function of the input, that is affected by the weights, and the transfer functions



Artificial Neural Networks

- An ANN can:
  1. compute *any computable* function, by the appropriate selection of the network topology and weights values.
  2. learn from experience!
     - Specifically, by trial-and-error

Learning by trial-and-error

Continuous process of:

- Trial:

  Processing an input to produce an output (In terms of ANN: Compute the output function of a given input)

- Evaluate:

Evaluating this output by comparing the actual output with the expected output.

- Adjust:

  Adjust the *weights*.

How it works?

- Set initial values of the weights randomly.
- Input: truth table of the XOR
- Do
  - Read input (e.g. 0, and 0)
  - Compute an output (e.g. 0.60543)
  - Compare it to the expected output. (Diff= 0.60543)
  - Modify the weights *accordingly.*
- Loop until a condition is met
  - Condition: certain number of iterations
  - Condition: error threshold

Design Issues

- Initial weights (small random values $\in[-1,1]$)
- Transfer function (How the inputs and the weights are combined to produce output?)
- Error estimation
- Weights adjusting
- Number of neurons
- Data representation
- Size of training set

Transfer Functions

- **Linear:** The output is proportional to the total weighted input.
- **Threshold:** The output is set at one of two values, depending on whether the total weighted input is greater than or less than some threshold value.
- **Non-linear:** The output varies continuously but not linearly as the input changes.

Error Estimation

- The **root mean square error (RMSE)** is a frequently-used measure of the differences between values predicted by a model or an estimator and the values actually observed from the thing being modeled or estimated

Weights Adjusting

- After each iteration, weights should be adjusted to minimize the error.
  - All possible weights
  - Back propagation

Back Propagation

- Back-propagation is an example of supervised learning is used at each layer to minimize the error between the layer's response and the actual data
- The error at each hidden layer is an average of the evaluated error
- Hidden layer networks are trained this way

Back Propagation

- N is a neuron.
- N_w is one of N's inputs weights
- N_{out} is N's output.
- $N_w = N_w + \Delta N_w$
- $\Delta N_w = N_{out} * (1 - N_{out}) * N_{ErrorFactor}$
- $N_{ErrorFactor} = N_{ExpectedOutput} - N_{ActualOutput}$
- This works only for the last layer, as we can know the actual output, and the expected output.

Number of neurons

- Many neurons:
  - Higher accuracy
  - Slower
  - Risk of over-fitting
    - Memorizing, rather than understanding
    - The network will be useless with new problems.
- Few neurons:
  - Lower accuracy
  - Inability to learn at all
- Optimal number.

Data representation

- Usually input/output data needs pre-processing
- Pictures
  - Pixel intensity
- Text:
  - A pattern

Size of training set

- No one-fits-all formula
- Over fitting can occur if a “good” training set is not chosen
- What constitutes a “good” training set?
  - Samples must represent the general population.
  - Samples must contain members of each class.
  - Samples in each class must contain a wide range of variations or noise effect.
- The size of the training set is related to the number of hidden neurons

Learning Paradigms

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning

- This is what we have seen so far!
- A network is fed with a set of training samples (inputs and corresponding output), and it uses these samples to learn the general relationship between the inputs and the outputs.
- This relationship is represented by the values of the weights of the trained network.

Unsupervised learning

- No desired output is associated with the training data!
- Faster than supervised learning
- Used to find out *structures within data:*
  - Clustering
  - Compression

Reinforcement learning

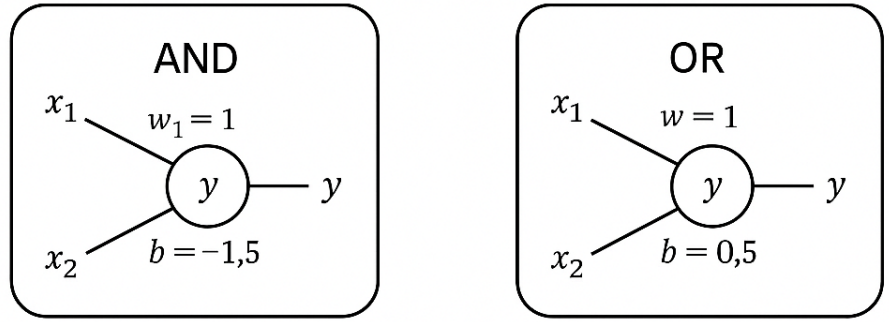
- Like supervised learning, but:
  - Weights adjusting is not directly related to the error value.
  - The error value is used to randomly, shuffle weights!
  - Relatively slow learning due to ‘randomness’.

Applications Areas

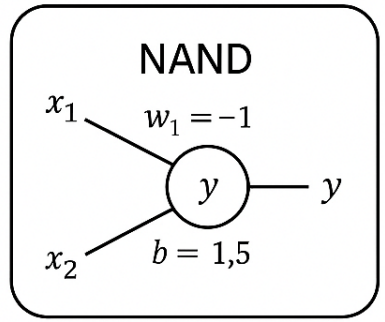
- Function approximation
  - including time series prediction and modeling.
- Classification
  - including patterns and sequences recognition, novelty detection and sequential decision making.
    - (radar systems, face identification, handwritten text recognition)
- Data processing
  - including filtering, clustering blinds source separation and compression.
    - (data mining, e-mail Spam filtering)

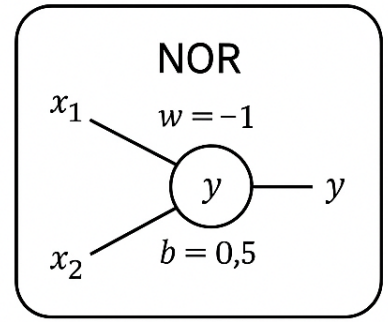
Advantages / Disadvantages

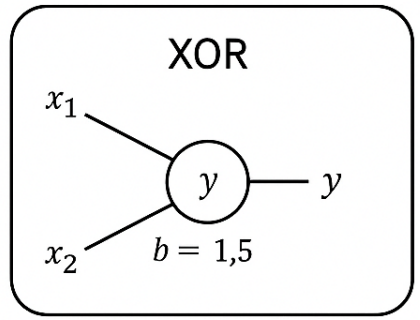
- Advantages
  - Adapt to unknown situations
  - Powerful, it can model complex functions.
  - Ease of use, learns by example, and very little user domain-specific expertise needed
- Disadvantages
  - Forgets
  - Not exact
  - Large complexity of the network structure

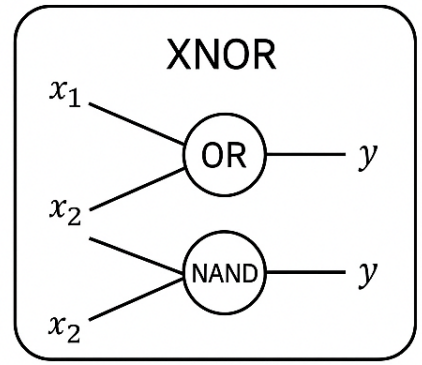

AND
x_1
$w_1 = 1$
y
y
x_2
$b = -1{,}5$

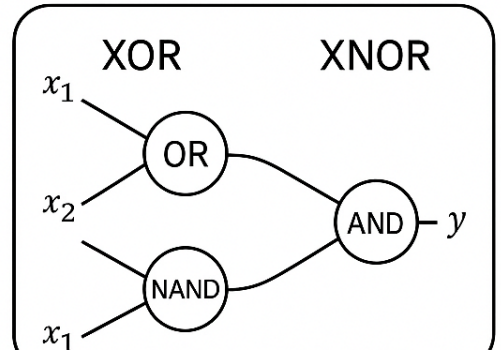
OR
x_1
$w = 1$
y
y
x_2
$b = 0{,}5$


NAND
x_1
$w_1 = -1$
y
y
x_2
$b = 1{,}5$


NOR
x_1
$w = -1$
y
y
x_2
$b = 0{,}5$


XOR
x_1
y
y
x_2
$b = 1{,}5$


XNOR
x_1
OR
y
x_2
NAND
y
x_2


XOR
XNOR
x_1
OR
x_2
AND
y
NAND
x_1

Conclusion

- Artificial Neural Networks are an imitation of the biological neural networks, but much simpler ones.
- The computing would have a lot to gain from neural networks. Their ability to learn by example makes them very flexible and powerful furthermore there is need to device an algorithm in order to perform a specific task.

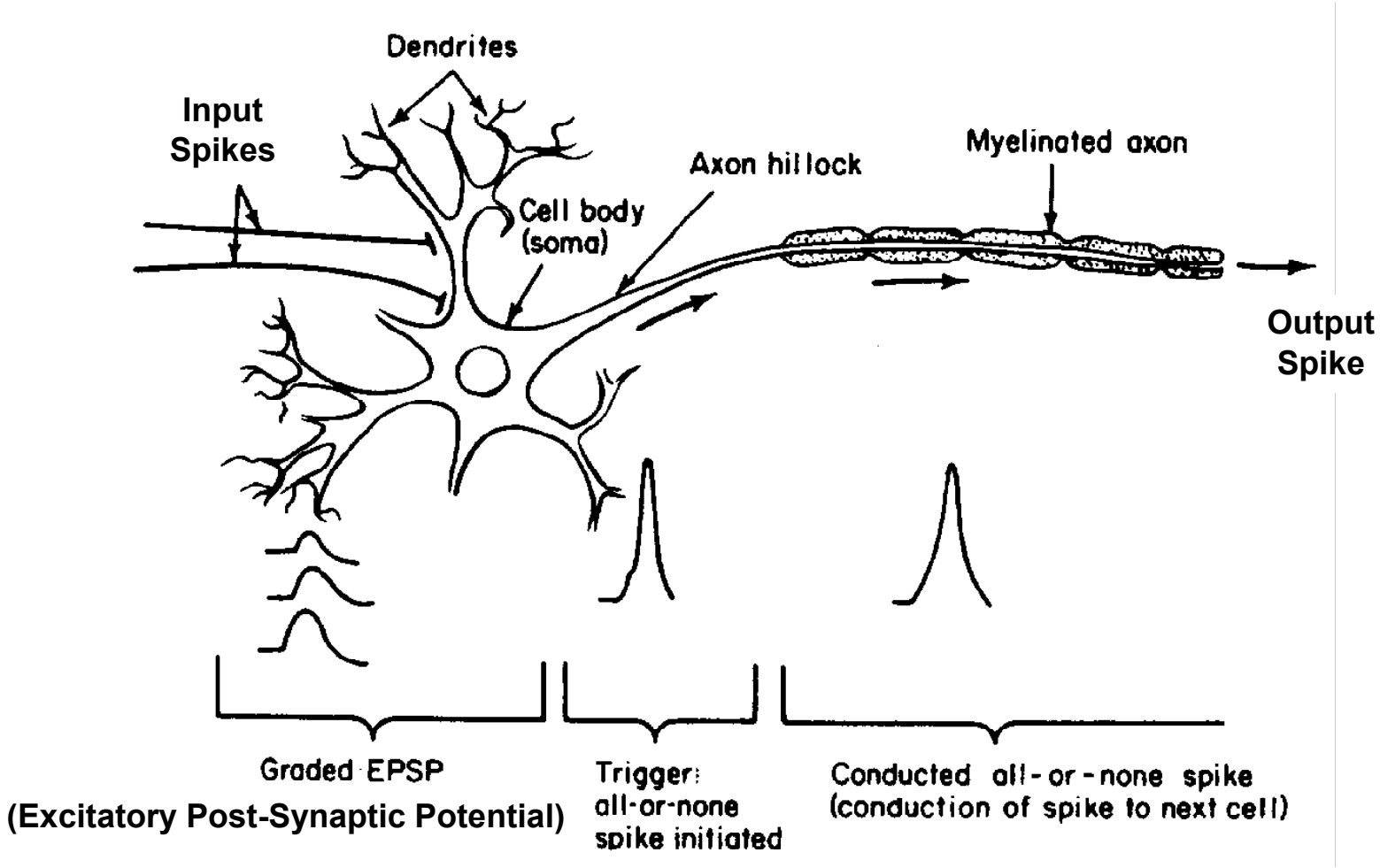
Conclusion

- Neural networks also contributes to area of research such a neurology and psychology. They are regularly used to model parts of living organizations and to investigate the internal mechanisms of the brain.
- Many factors affect the performance of ANNs, such as the transfer functions, size of training sample, network topology, weights adjusting algorithm, …

References

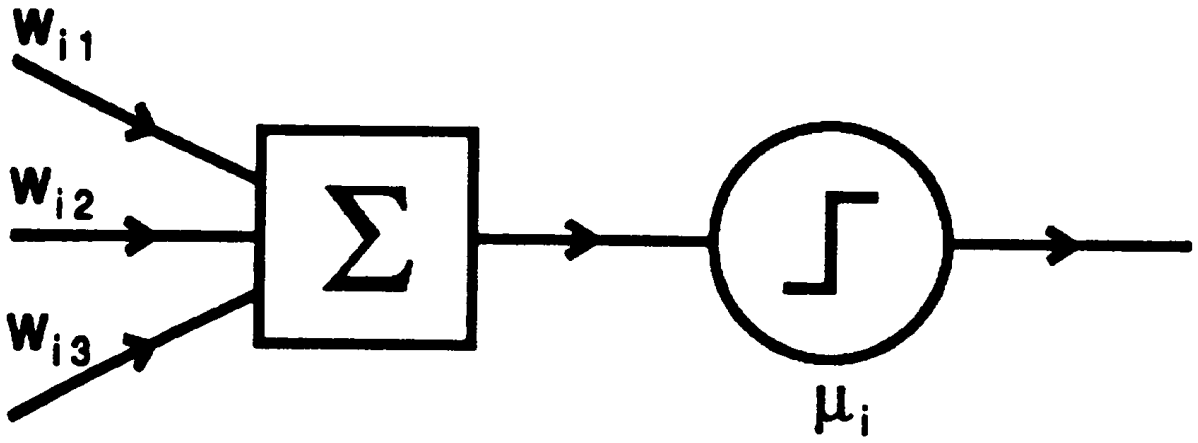
- Craig Heller, and David Sadava, *Life: The Science of Biology, fifth edition,* Sinauer Associates, INC, USA, 1998.
- Introduction to Artificial Neural Networks, Nicolas Galoppo von Borries
- Tom M. Mitchell, *Machine Learning, WCB McGraw-Hill, Boston, 1997.*

Basic Input-Output Transformation



McCulloch–Pitts “neuron” (1943)

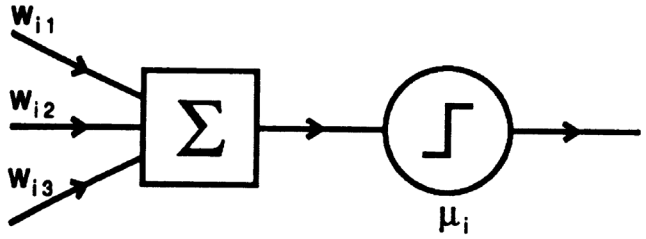
- Attributes of neuron
  - m binary inputs and 1 output (0 or 1)
  - Synaptic weights w_{ij}
  - Threshol



McCulloch–Pitts Neural Networks

- Synchronous discrete time operation
  - Time quantized in units of synaptic delay

$$n_i(t+1) = \Theta\left(\sum_j w_{ij}\, n_j(t) - \mu_i\right)$$



$n_i \equiv$ output of unit i
$\Theta \equiv$ step function
$w_{ij} =$ weight from unit j to i
$\mu_i =$ threshold

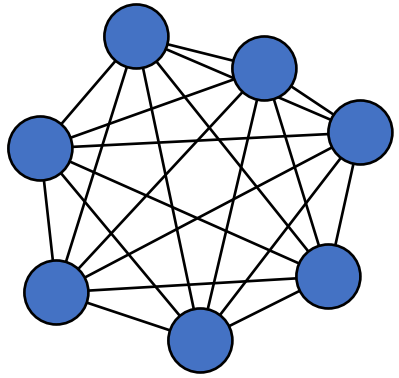
- Output is 1 if and only if weighted sum of inputs is greater than threshold
  $\Theta(x) = 1$ if $x \geq 0$ and 0 if $x < 0$

- Behavior of network can be simulated by a finite automaton
- Any FA can be simulated by a McCulloch-Pitts Network

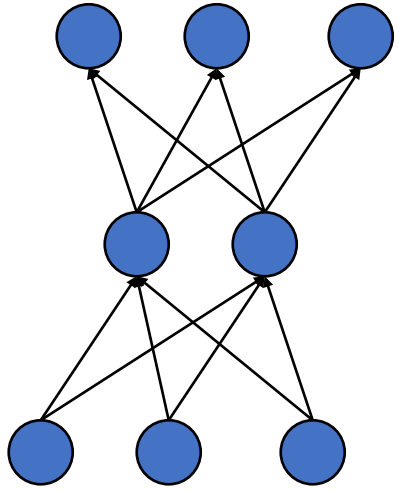
Properties of Artificial Neural Networks

- High level abstraction of neural input-output transformation
  - Inputs → weighted sum of inputs → nonlinear function → output
    - Typically no spikes
    - Typically use implausible constraints or learning rules
- Often used where data or functions are uncertain
  - Goal is to learn from a set of training data
  - And to generalize from learned instances to new unseen data
- Key attributes
  - Parallel computation
  - Distributed representation and storage of data
  - Learning (networks adapt themselves to solve a problem)
  - Fault tolerance (insensitive to component failures)

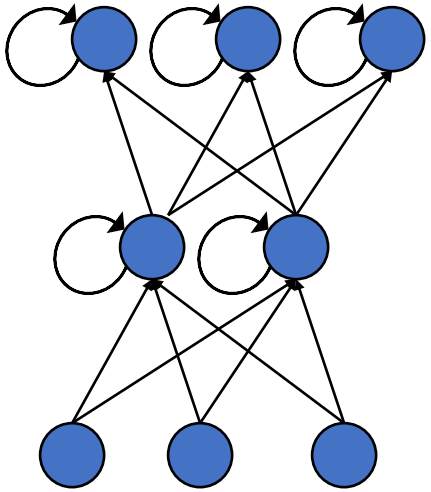
Topologies of Neural Networks



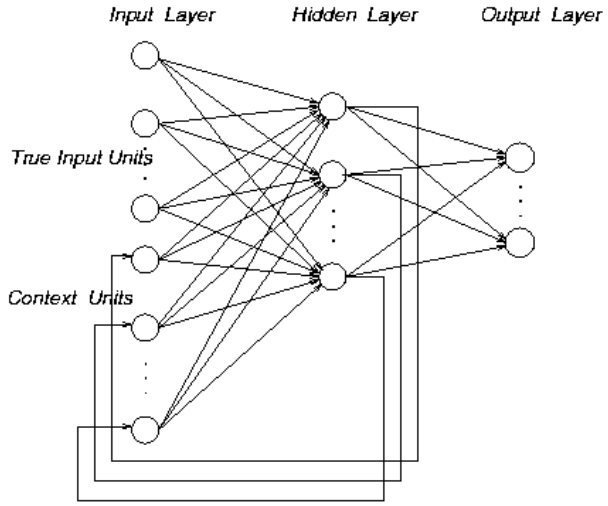
completely connected



feedforward (directed, a-cyclic)



recurrent (feedback connections)



Networks Types

- Feedforward versus recurrent networks
  - Feedforward: No loops, input → hidden layers → output
  - Recurrent: Use feedback (positive or negative)
- Continuous versus spiking
  - Continuous networks model mean spike rate (firing rate)
    - Assume spikes are integrated over time
  - Consistent with rate-code model of neural coding
- Supervised versus unsupervised learning
  - Supervised networks use a “teacher”
    - The desired output for each input is provided by user
  - Unsupervised networks find hidden statistical patterns in input data
    - Clustering, principal component analysis

History

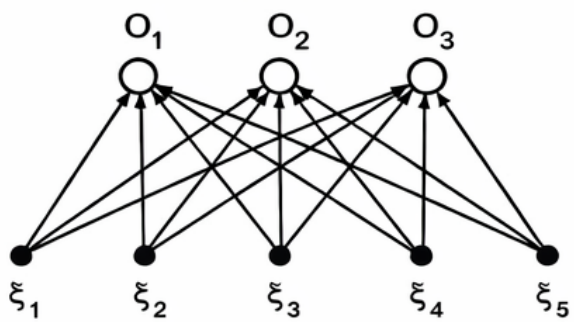
- 1943: McCulloch–Pitts “neuron”
  - Started the field
- 1962: Rosenblatt’s perceptron
  - Learned its own weight values; convergence proof
- 1969: Minsky & Papert book on perceptrons
  - Proved limitations of single-layer perceptron networks
- 1982: Hopfield and convergence in symmetric networks
  - Introduced energy-function concept
- 1986: Backpropagation of errors
  - Method for training multilayer networks
- Present: Probabilistic interpretations, Bayesian and spiking networks

Perceptrons

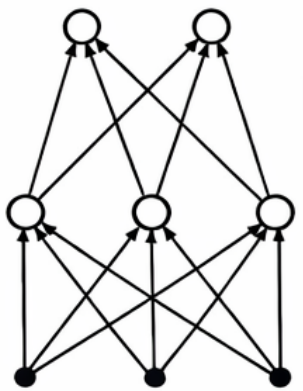
- Attributes
  - Layered feedforward networks
  - Supervised learning
    - Hebbian: Adjust weights to enforce correlations
  - Parameters: weights w_{ij}
  - Binary output = Θ(weighted sum of inputs)
    - Take w_o to be the threshold with fixed input -1.

$$Output_i = \Theta \sum w_{ij} \xi_j$$

Single-layer



Single-layer



Multilayer

Multilayer

Training Perceptrons to Compute a Function

- Given inputs ξ_j to neuron i and desired output Y_i, find its weight values by iterative improvement:
  1. Feed an input pattern
  2. Is the binary output correct?
     - $\Rightarrow$Yes: Go to the next pattern
     - $\Rightarrow$No: Modify the connection weights using error signal ($Y_i - O_i$)
     - $\Rightarrow$Increase weight if neuron didn't fire when it should have and vice versa

$$\Delta w_{ij} = \eta \, (a_i - o_i) \, \xi_j$$

and in expanded form (using the weighted sum for o_i):

$$= \eta \left(a_i - \sum_j w_{ij} \, \xi_j \right) \xi_j$$

h º learning rate

x_j º input

Y_i º desired output

O_i º actual output

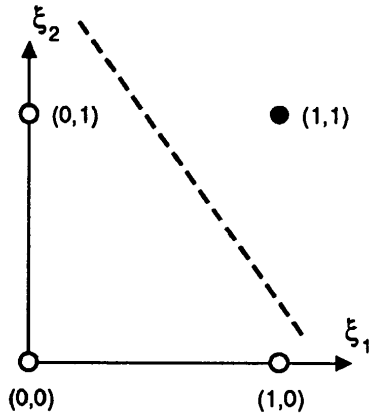
- Learning rule is Hebbian (based on input/output correlation)
  - converges in a finite number of steps if a solution exists
  - Used in ADALINE (adaptive linear neuron) networks

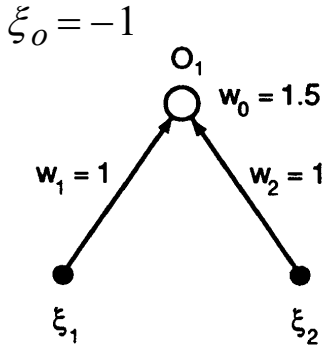
Computational Power of Perceptrons

- Consider a single-layer perceptron
  - Assume threshold units
  - Assume binary inputs and outputs
  - Weighted sum forms a linear hyperplane

$$\sum_j w_{ij}\xi_j = 0$$

- Consider a single output network with two inputs
  - Only functions that are linearly separable can be computed
  - Example: AND is linearly separable



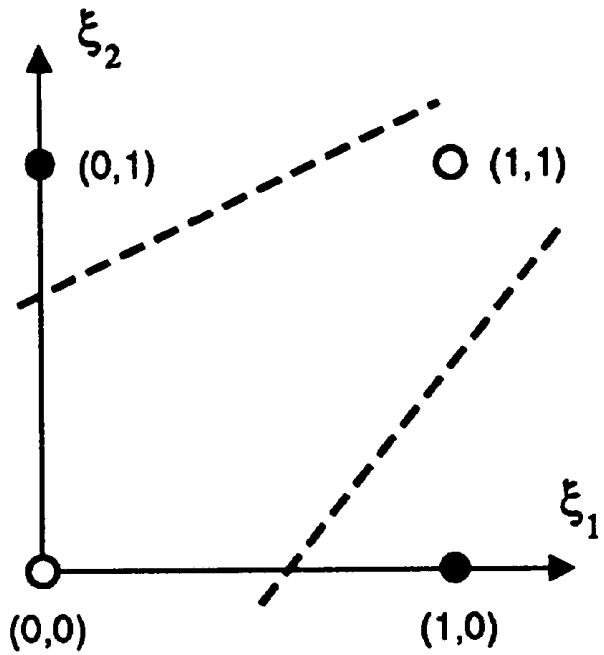


Linear inseparability

- Single-layer perceptron with threshold units fails if problem is not linearly separabl
  - Example: XOR

F Can use other tricks (e.g. complicated threshold functions) but complexity blows up

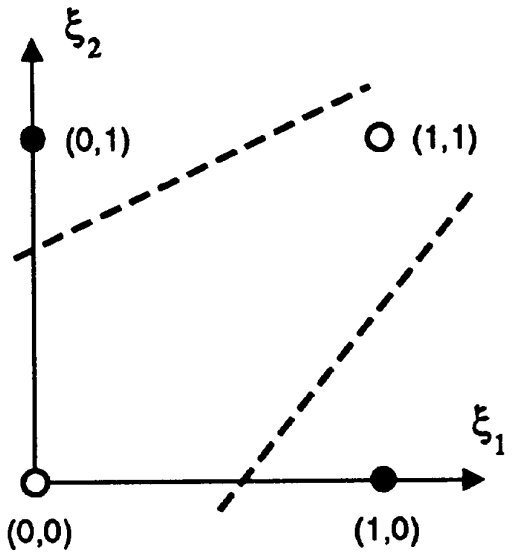
F Minsky and Papert's book showing these negative results was very influential



Solution in 1980s: Multilayer perceptrons

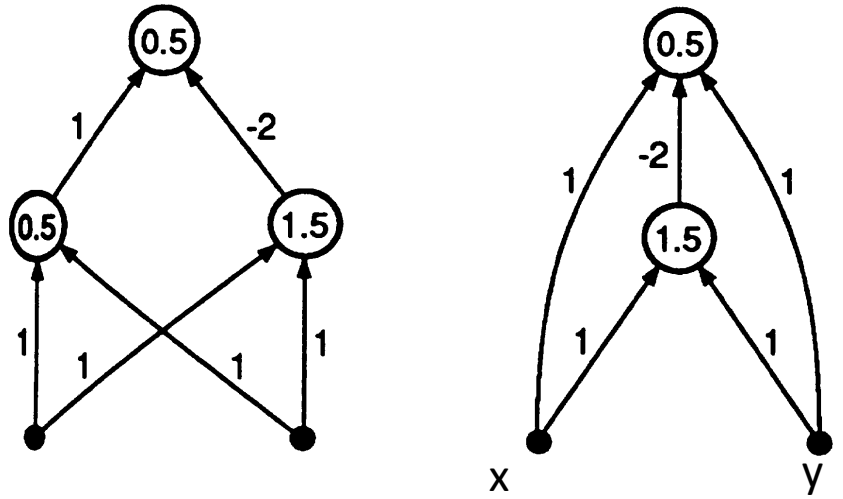
- Removes many limitations of single-layer networks
  - Can solve XOR

- Exercise: Draw a two-layer perceptron that computes the XOR function
  - 2 binary inputs ξ_1 and ξ_2
  - 1 binary output
  - One “hidden” layer
  - Find the appropriate weights and threshold



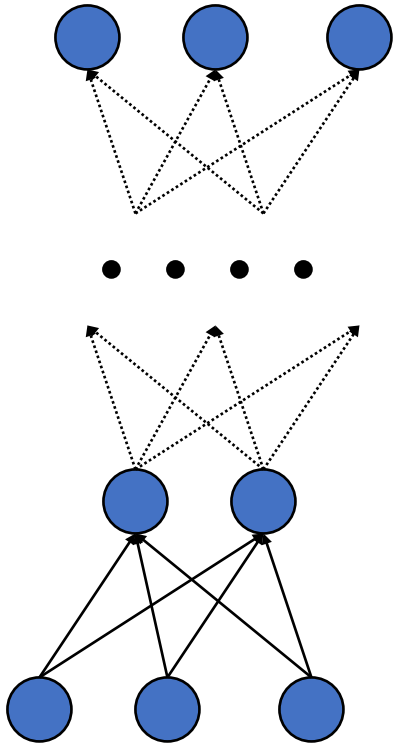
Solution in 1980s: Multilayer perceptrons

- Examples of two-layer perceptrons that compute XOR



- E.g. Right side network
  - Output is 1 if and only if $x + y - 2(x + y - 1.5 > 0) - 0.5 > 0$

Multilayer Perceptron



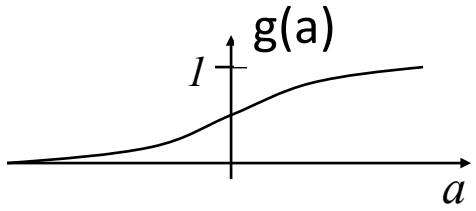
Output neurons

One or more layers of hidden units (hidden layers)

Input nodes

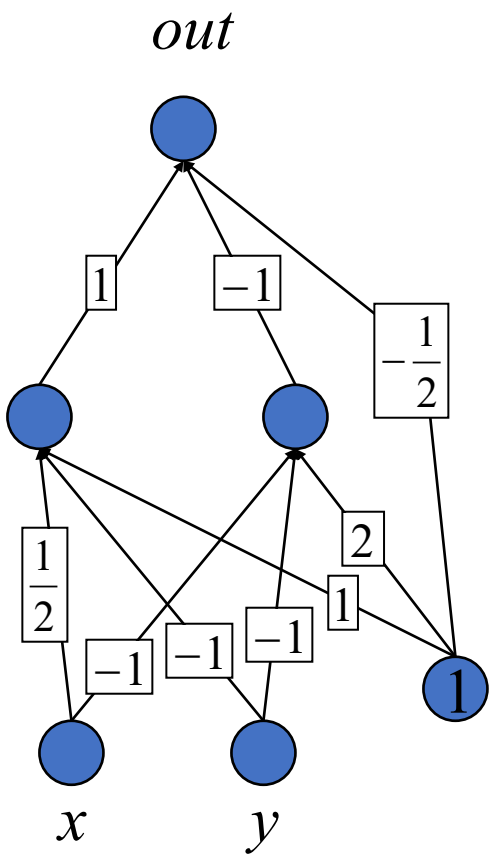
The most common output function (Sigmoid):

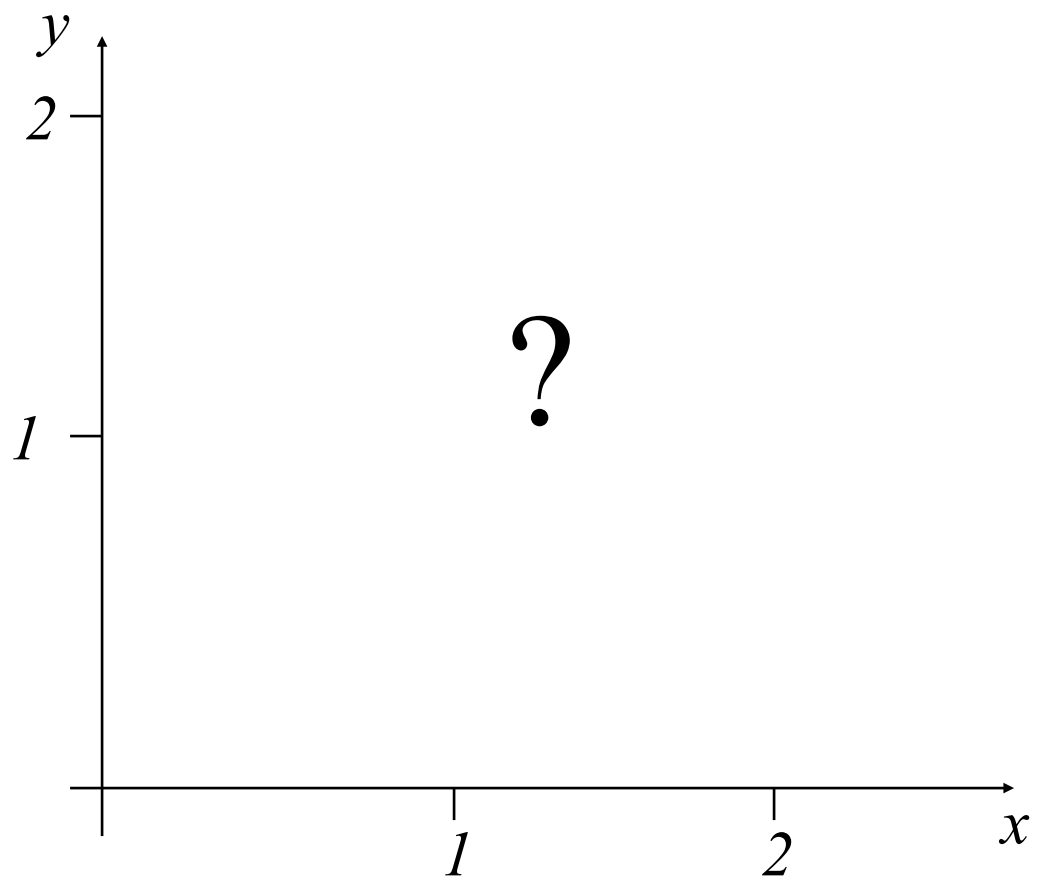
$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



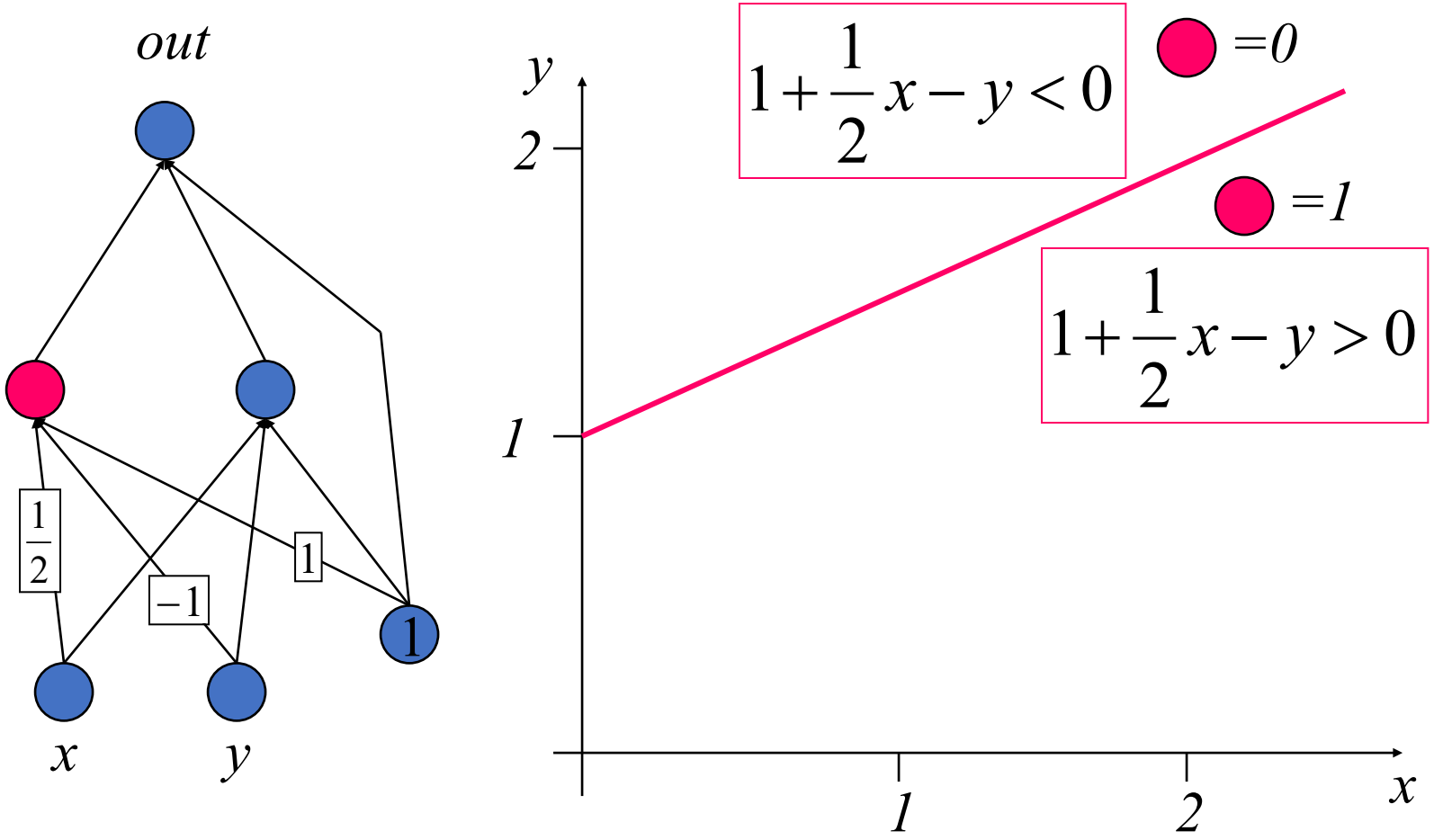
(non-linear squashing function)

Example: Perceptrons as Constraint Satisfaction Networks

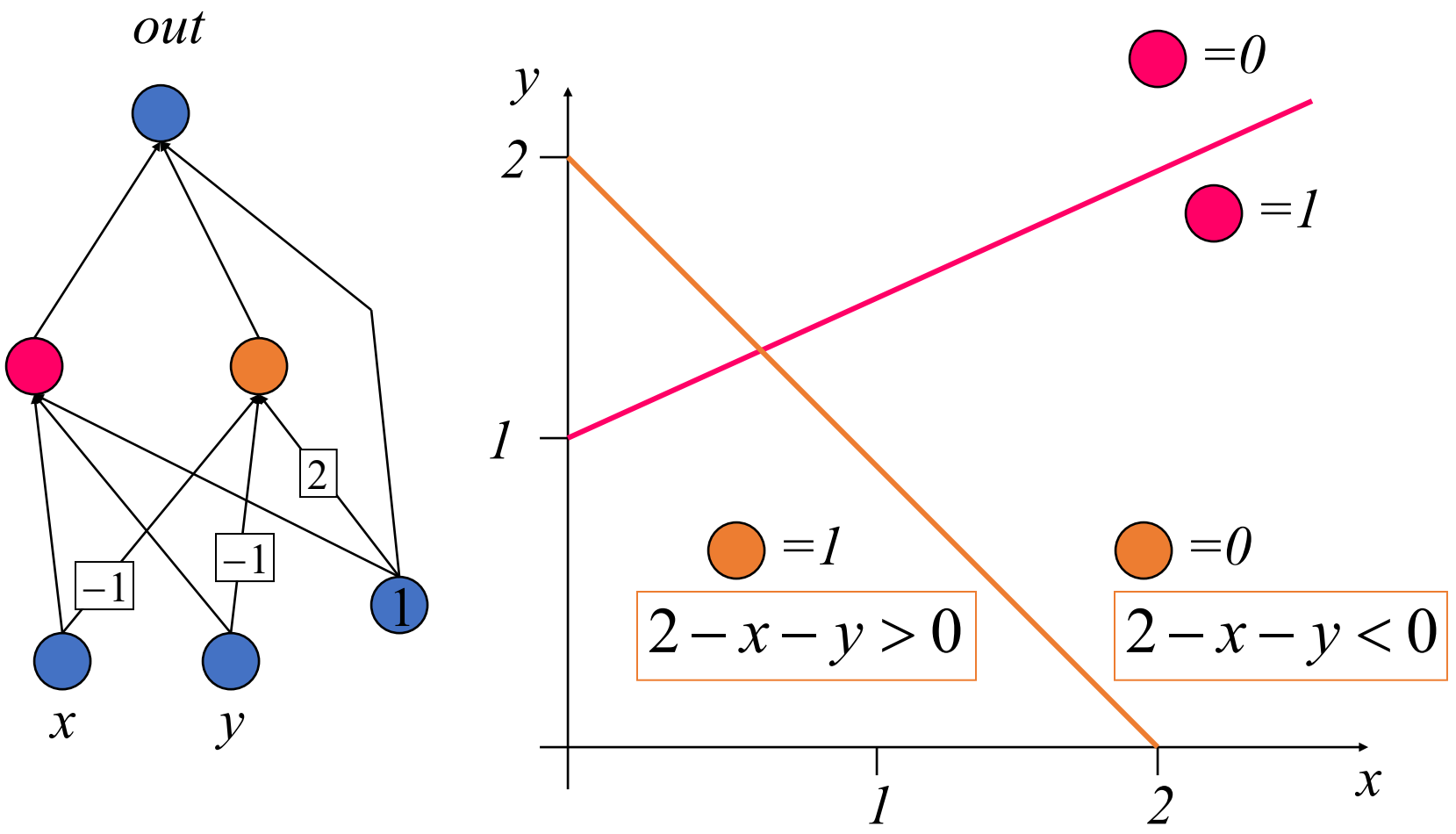




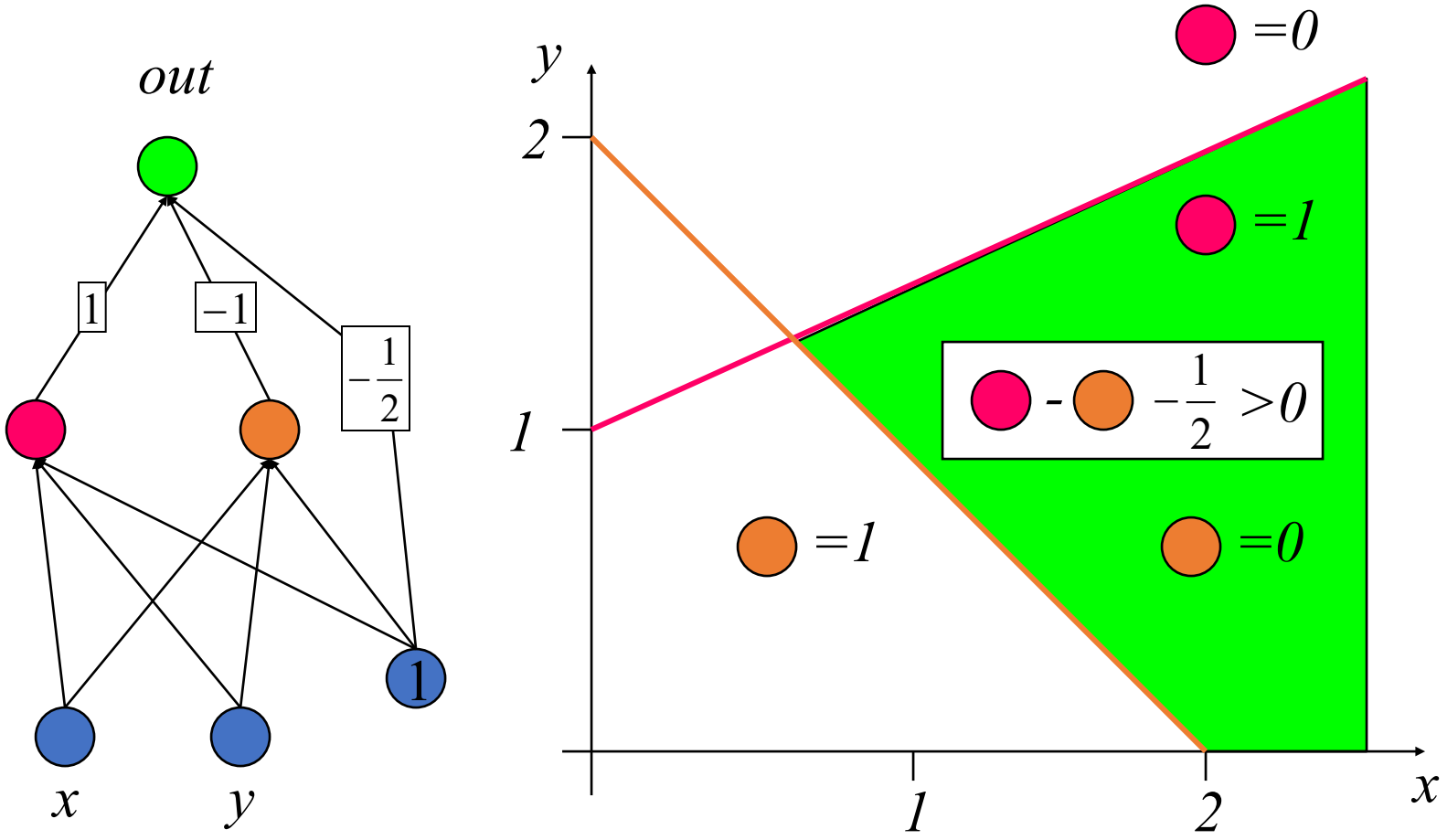
Example: Perceptrons as Constraint Satisfaction Networks



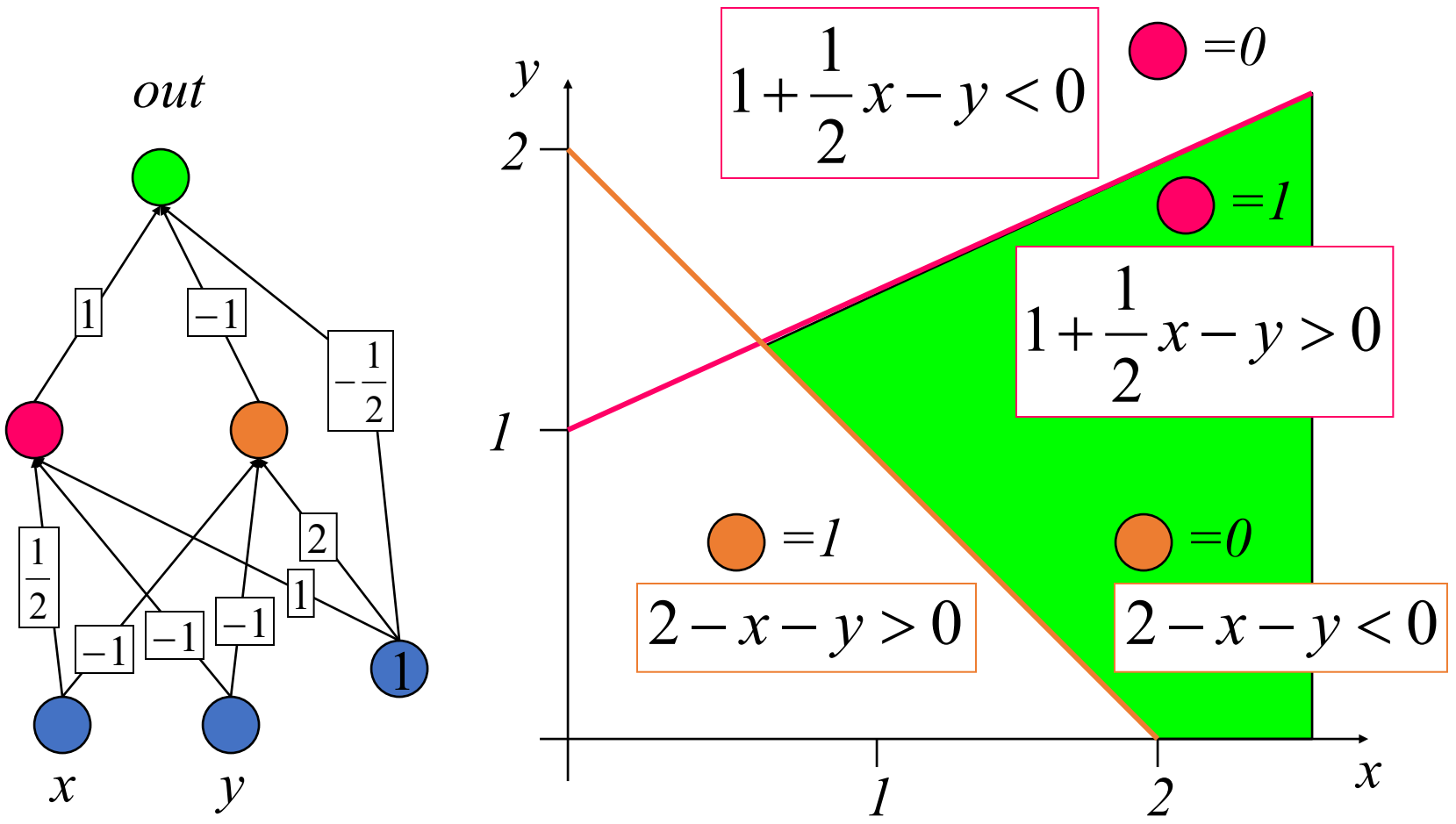
Example: Perceptrons as Constraint Satisfaction Networks



Example: Perceptrons as Constraint Satisfaction Networks

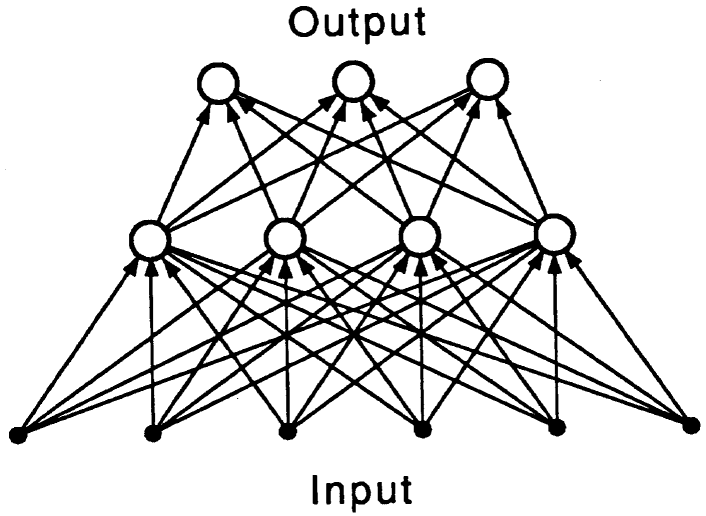


Perceptrons as Constraint Satisfaction Networks



Learning networks

- We want networks that configure themselves
  - Learn from the input data or from training examples
  - Generalize from learned data



Can this network configure itself to solve a problem?

How do we train it?

Gradient-descent learning

- Use a differentiable activation function
  - Try a continuous function *f* ( ) instead of Θ( )
    - First guess: Use a linear unit
  - Define an error function (cost function or "energy" function)

$$E = \frac{1}{2} \sum_i \sum_u \left(t_i^{(u)} - \sum_j w_{ij}\, \xi_j^{(u)} \right)^2$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_u \left(t_i^{(u)} - \sum_j w_{ij}\, \xi_j^{(u)} \right) \xi_j^{(u)}$$

Cost function measures the network's performance as a differentiable function of the weights

- Changes weights in the direction of smaller errors
  - Minimizes the mean-squared error over input patterns μ
  - Called Delta rule = adaline rule = Widrow-Hoff rule = LMS rule

Backpropagation of errors

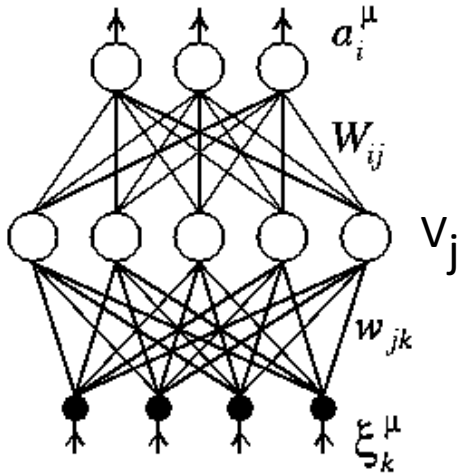
- Use a *nonlinear*, differentiable activation function
  - Such as a sigmoid

$$f = \frac{1}{1 + \exp(-2h)} \quad \text{where} \quad h = \sum_j w_{ij}\, \xi_j$$

- Use a multilayer feedforward network
  - Outputs are differentiable functions of the inputs
- Result: Can propagate credit/blame back to internal nodes
  - Chain rule (calculus) gives Δw_{ij} for internal "hidden" nodes
  - Based on gradient-descent learning

Backpropagation of errors (con't)

Multi-layer error-back-propagation (MLBP)



$$a_i^\mu = g_i\left(\sum_j W_{ij}\, g_j\left(\sum_k w_{jk}\xi_k^\mu\right)\right)$$

Back-propagation learning : $\Delta W_{ij}(t+1) = -\eta \frac{\partial E}{\partial W_{ij}} + \alpha \Delta W_{ij}(t)$

Error measure : $E = \frac{1}{2}\sum_{i\mu} (d_i^\mu - a_i^\mu)^2$

Backpropagation of errors (con't)

- Let A_i be the activation (weighted sum of inputs) of neuron i
- Let $V_j = g(A_j)$ be output of hidden unit j
- Learning rule for hidden-output connection weights:
  - $\Delta W_{ij} = -\eta \partial E/\partial W_{ij} = \eta \Sigma_\mu [d_i - a_i] g'(A_i) V_j$

    $= \eta \Sigma_\mu \delta_i V_j$
- Learning rule for input-hidden connection weights:
  - $\Delta w_{jk} = -\eta \partial E/\partial w_{jk} = -\eta (\partial E/\partial V_j) (\partial V_j/\partial w_{jk})$ {chain rule}

    $=\eta \Sigma_{\mu,\iota} ([d_i - a_i] g'(A_i) W_{ij}) (g'(A_j) \xi_k)$

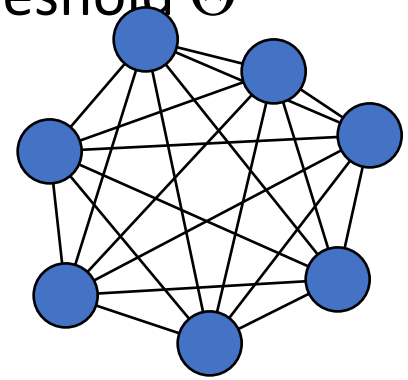
    $= \eta \Sigma_\mu \delta_j \xi_k$

Backpropagation

- Can be extended to arbitrary number of layers but three is most commonly used
- Can approximate arbitrary functions: crucial issues are
  - generalization to examples not in test data set
  - number of hidden units
  - number of samples
  - speed of convergence to a stable set of weights (sometimes a momentum term $\alpha \, \Delta w_{pq}$ is added to the learning rule to speed up learning)
- In your homework, you will use backpropagation and the delta rule in a simple pattern recognition task: classifying noisy images of the digits 0 through 9
- C Code for the networks is already given – you will only need to modify the input and output

Hopfield networks

- Act as “autoassociative” memories to store patterns
  - McCulloch-Pitts neurons with outputs -1 or 1, and threshold Θ
  - All neurons connected to each other
    - Symmetric weights ($w_{ij} = w_{ji}$) and $w_{ii} = 0$
  - Asynchronous updating of outputs
    - Let s_i be the state of unit i
    - At each time step, pick a random unit
    - Set s_i to 1 if $\sum_j w_{ij} s_j \geq \Theta$; otherwise, set s_i to -1



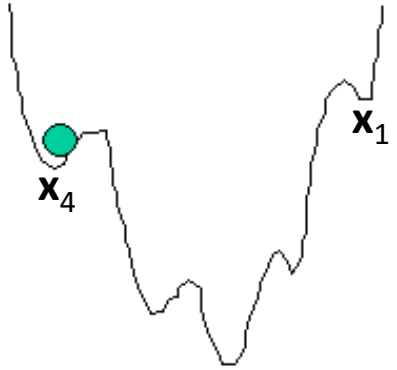
completely connected

Hopfield networks

- Hopfield showed that asynchronous updating in symmetric networks minimizes an “energy” function and leads to a stable final state for a given initial state
- Define an energy function (analogous to the gradient descent error function)
  - $E = -1/2 \sum_{i,j} w_{ij} s_i s_j + \sum_i s_i \Theta_i$
- Suppose a random unit i was updated: E always decreases!
  - If s_i is initially –1 and $\sum_j w_{ij} s_j > \Theta_i$, then s_i becomes +1
    - Change in $E = -1/2 \sum_j (w_{ij} s_j + w_{ji} s_j) + \Theta_i = - \sum_j w_{ij} s_j + \Theta_i < 0$!!
  - If s_i is initially +1 and $\sum_j w_{ij} s_j < \Theta_i$, then s_i becomes -1
    - Change in $E = 1/2 \sum_j (w_{ij} s_j + w_{ji} s_j) - \Theta_i = \sum_j w_{ij} s_j - \Theta_i < 0$!!

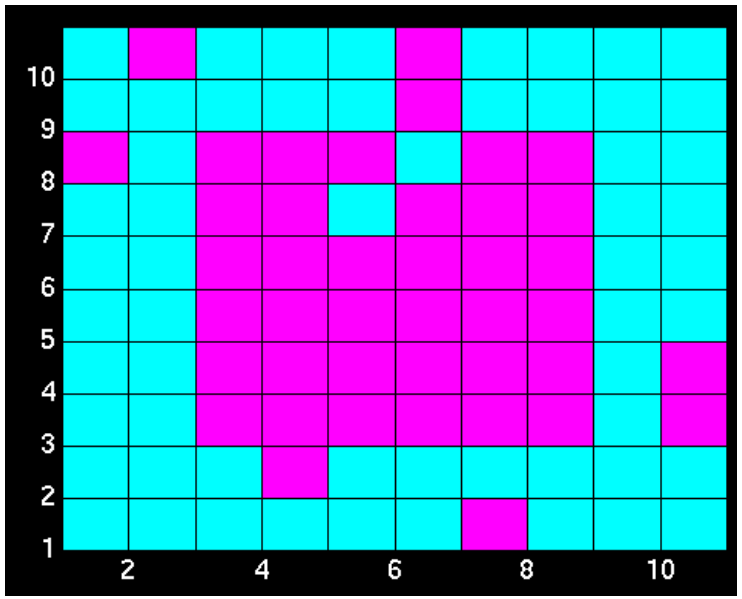
Hopfield networks

- Note: Network converges to local minima which store different patterns.

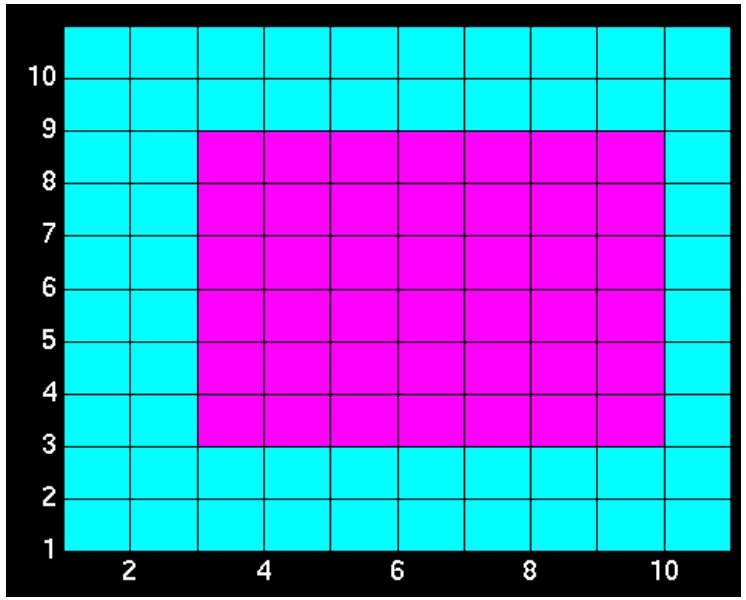


- Store p N-dimensional pattern vectors $\mathbf{x}_1, \ldots, \mathbf{x}_p$ using Hebbian learning rule:
  - $w_{ji} = 1/N \sum_{m=1,..,p} x_{m,j} x_{m,i}$ for all $j \neq i$; 0 for $j = i$
  - $W = 1/N \sum_{m=1,..,p} \mathbf{x}_m \mathbf{x}_m^T$ (outer product of vectors; diagonal zero)
    - T denotes vector transpose

Pattern Completion in a Hopfield Network

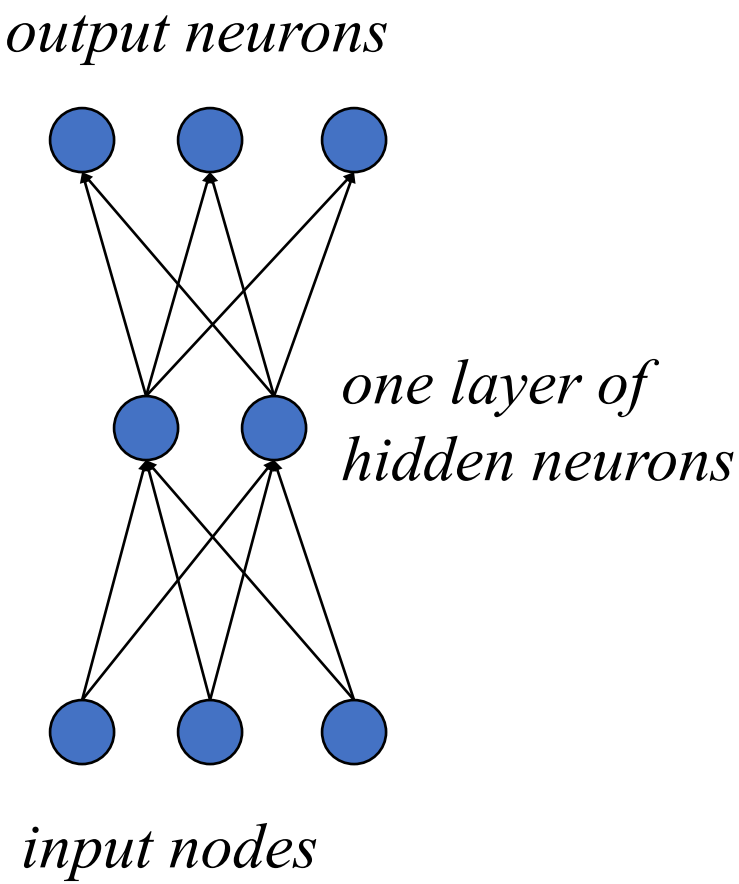


→

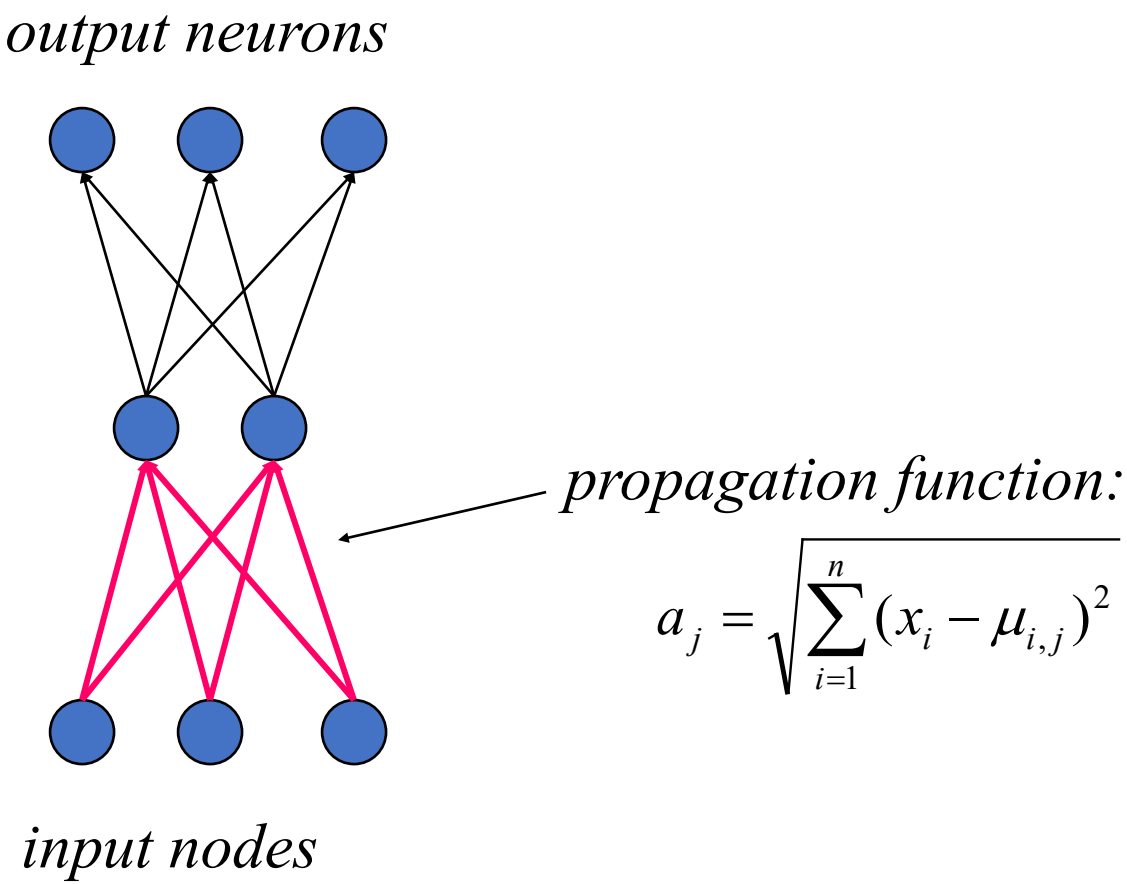


Local minimum
(“attractor”)
of energy function
stores pattern

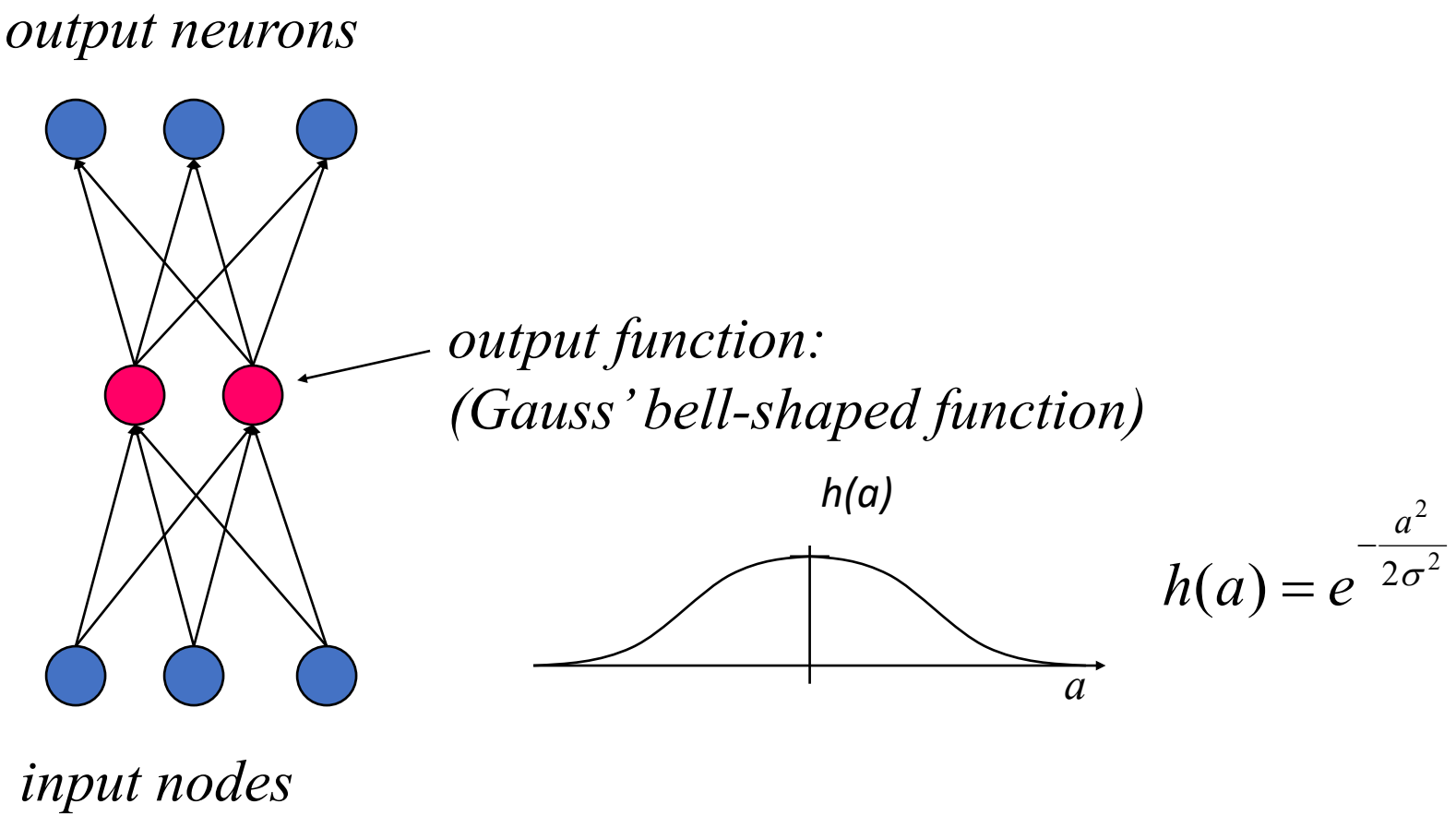
Radial Basis Function Networks



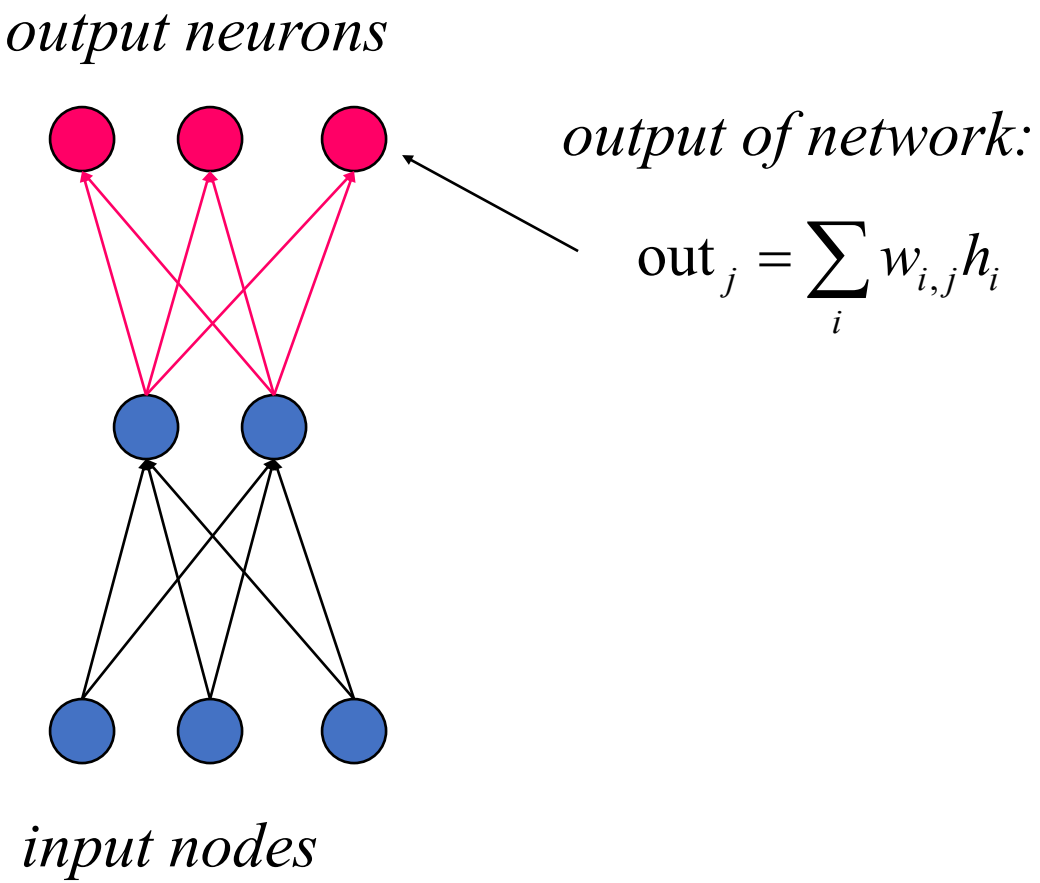
Radial Basis Function Networks



Radial Basis Function Networks

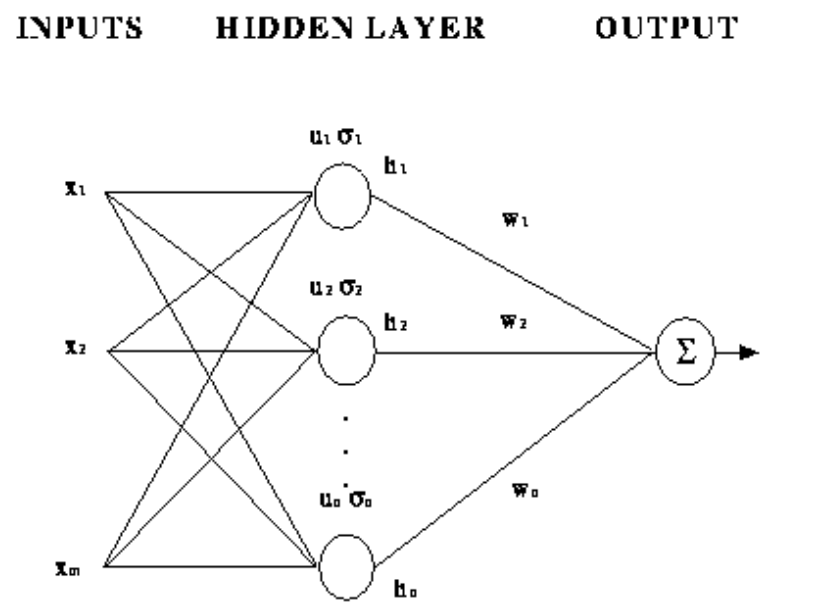


Radial Basis Function Networks



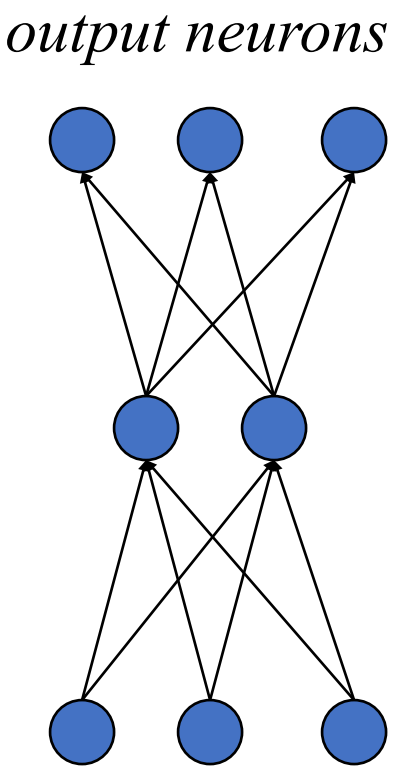
RBF networks

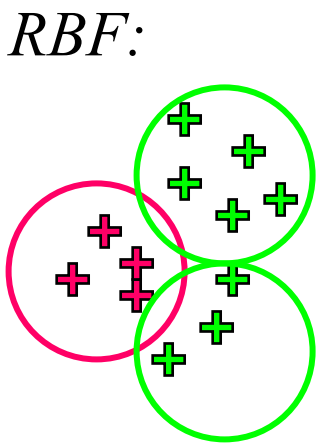
- Radial basis functions
  - Hidden units store means and variances
  - Hidden units compute a Gaussian function of inputs $x_1,\ldots x_n$ that constitute the input vector **x**
- Learn weights w_i, means μ_i, and variances σ_i by minimizing squared error function (gradient descent learning)

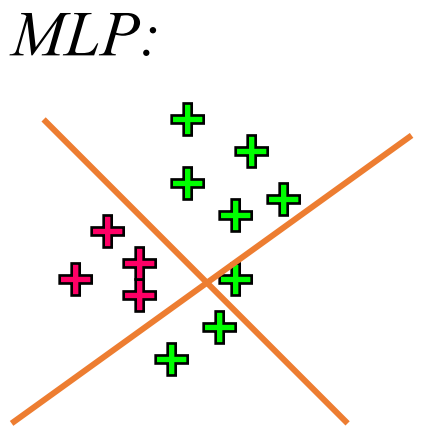


$$h_i = exp[-\frac{(\mathbf{x}-\mathbf{u}_i)^T(\mathbf{x}-\mathbf{u}_i)}{2\sigma^2}],\ y = \sum_i h_i w_i$$

RBF Networks and Multilayer Perceptrons







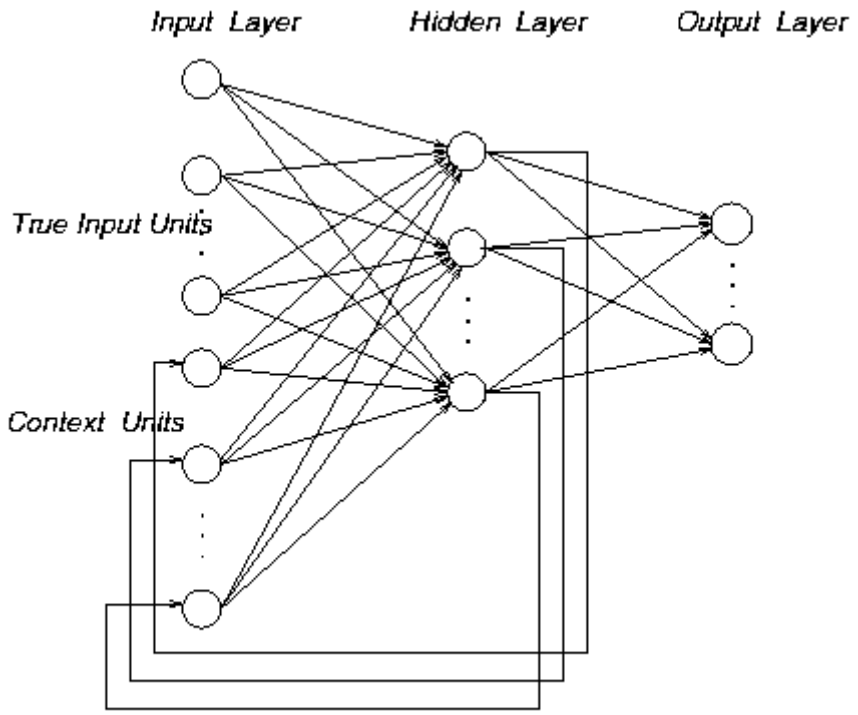
Recurrent networks

- Employ feedback (positive, negative, or both)
  - Not necessarily stable
    - Symmetric connections can ensure stability
- Why use recurrent networks?
  - Can learn temporal patterns (time series or oscillations)
  - Biologically realistic
    - Majority of connections to neurons in cerebral cortex are feedback connections from local or distant neurons
- Examples
  - Hopfield network
  - Boltzmann machine (Hopfield-like net with input & output units)
  - Recurrent backpropagation networks: for small sequences, unfold network in time dimension and use backpropagation learning

Recurrent networks (con't)

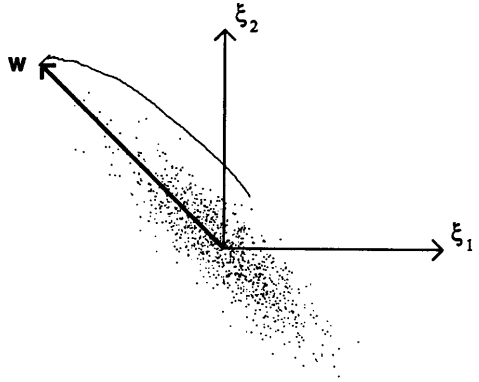
- Example
  - Elman networks
    - Partially recurrent
    - Context units keep internal memory of part inputs
    - Fixed context weights
    - Backpropagation for learning
    - E.g. Can disambiguate A→B→C and C→B→A

Elman network



Unsupervised Networks

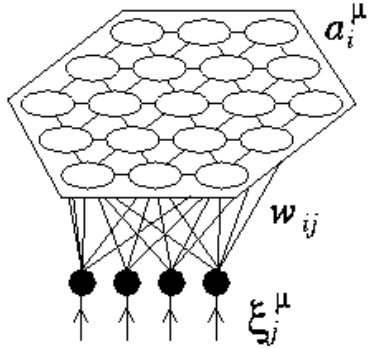
- No feedback to say how output differs from desired output (no error signal) or even whether output was right or wrong
- Network must discover patterns in the input data by itself
  - Only works if there are redundancies in the input data
  - Network self-organizes to find these redundancies
    - Clustering: Decide which group an input belongs to
      - Synaptic weights of one neuron represents one group
    - Principal Component Analysis: Finds the data covariance matrix
    - Hebb rule performs PCA! (Oja, 1982)
      - $\Delta w_i = \eta \, \xi_i y$
      - Output $y = \Sigma_i \, w_i \, \xi_i$



Self-Organizing Maps (Kohonen Maps)

- Feature maps
  - Competitive networks
  - Neurons have locations
  - For each input, winner is the unit with largest output
  - Weights of winner and nearby units modified to resemble input pattern
  - Nearby inputs are thus mapped topographically
- Biological relevance
  - Retinotopic map
  - Somatosensory map
  - Tonotopic map

Self Organized Map (SOM)



$$a_i^\mu = \begin{cases} 1 & \text{if } i = i^* \\ 0 & \text{if } i \neq i^* \end{cases}$$

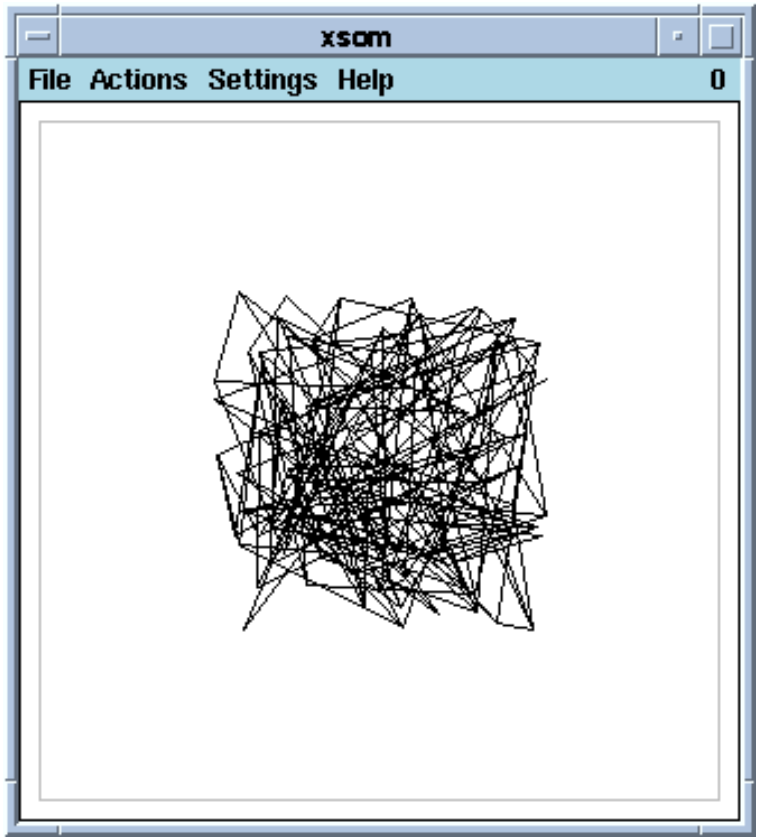
$$h_i^\mu = \sum_j w_{ij}\xi_j \qquad h_{i^*}^\mu \geq h_i^\mu \text{ for all } i$$

Kohonen learning : $\Delta w_{ij} = \eta \Lambda(i,i^*)(\xi_j^\mu - w_{ij})$

Neighborhood function : $\Lambda(i,i^*) = e^{-\frac{|\mathbf{r}_i - \mathbf{r}_{i^*}|^2}{2\sigma^2}}$

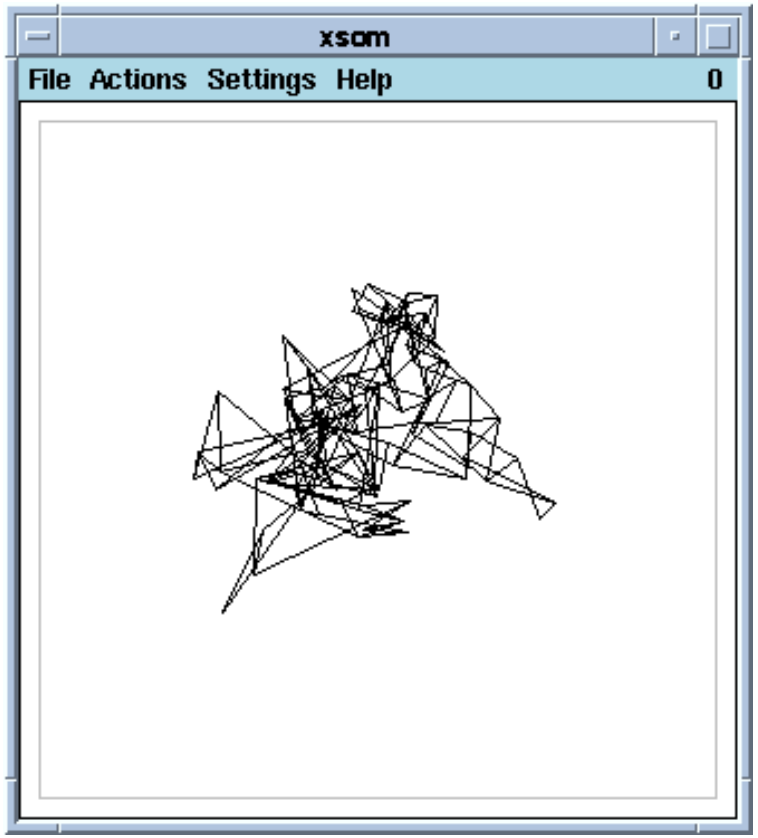
Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Initial weights w1 and w2 random as shown on right; lines connect neighbors



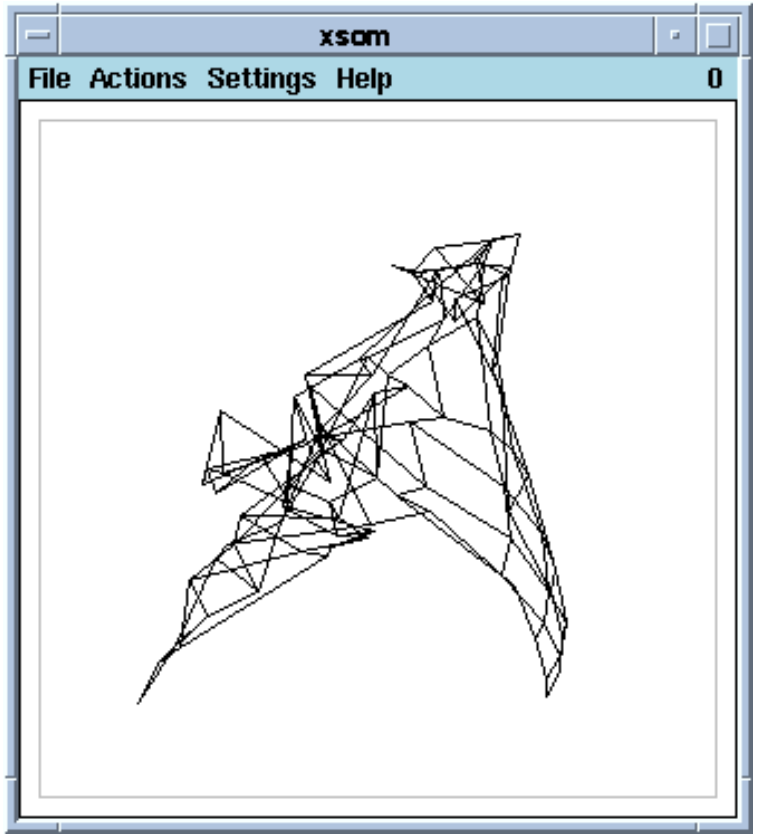
Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Weights after 10 iterations



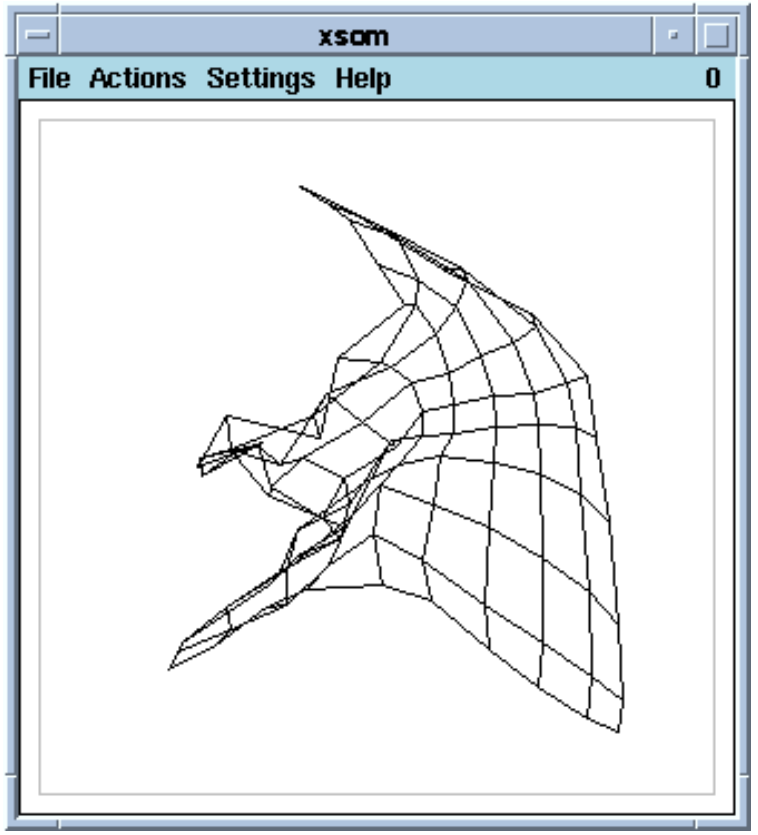
Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Weights after 20 iterations



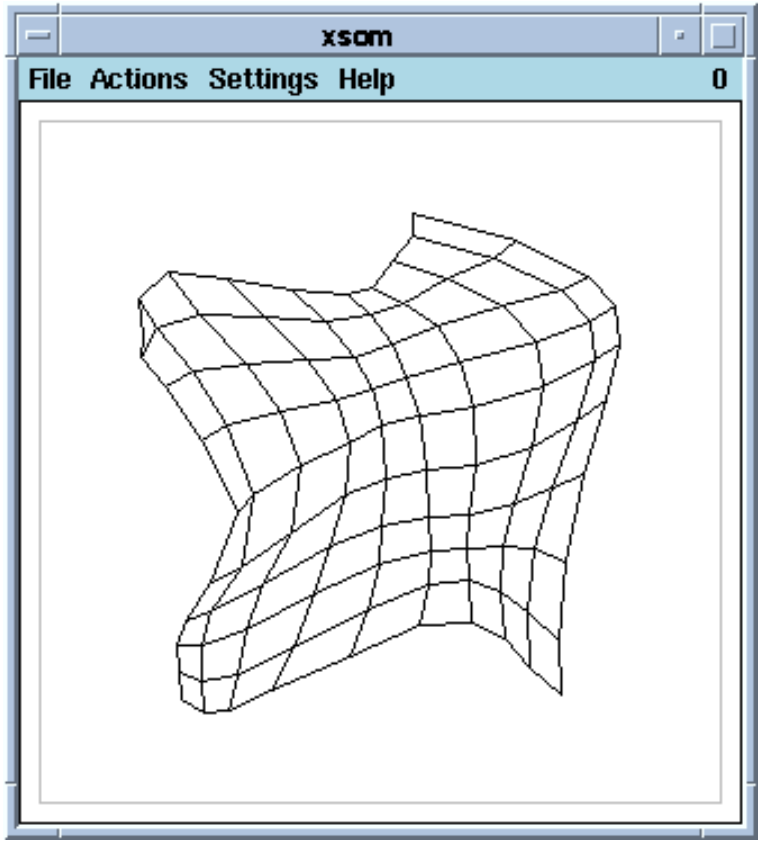
Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Weights after 40 iterations



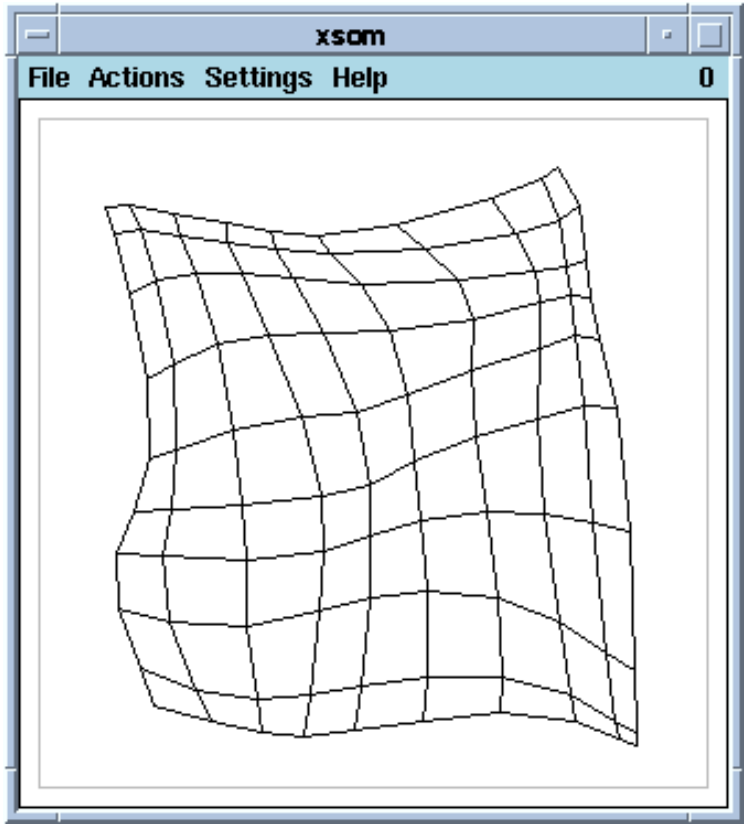
Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Weights after 80 iterations



Example of a 2D Self-Organizing Map

- 10 x 10 array of neurons
- 2D inputs (x,y)
- Final Weights (after 160 iterations)
- Topography of inputs has been captured



Summary: Biology and Neural Networks

- So many similarities
  - Information is contained in synaptic connections
  - Network learns to perform specific functions
  - Network generalizes to new inputs
- But NNs are woefully inadequate compared with biology
  - Simplistic model of neuron and synapse, implausible learning rules
  - Hard to train large networks
  - Network construction (structure, learning rate etc.) is a heuristic art
- One obvious difference: Spike representation
  - Recent models explore spikes and spike-timing dependent plasticity
- Other Recent Trends: Probabilistic approach
  - NNs as Bayesian networks (allows principled derivation of dynamics, learning rules, and even structure of network)
  - Not clear how neurons encode probabilities in spikes

Thank You