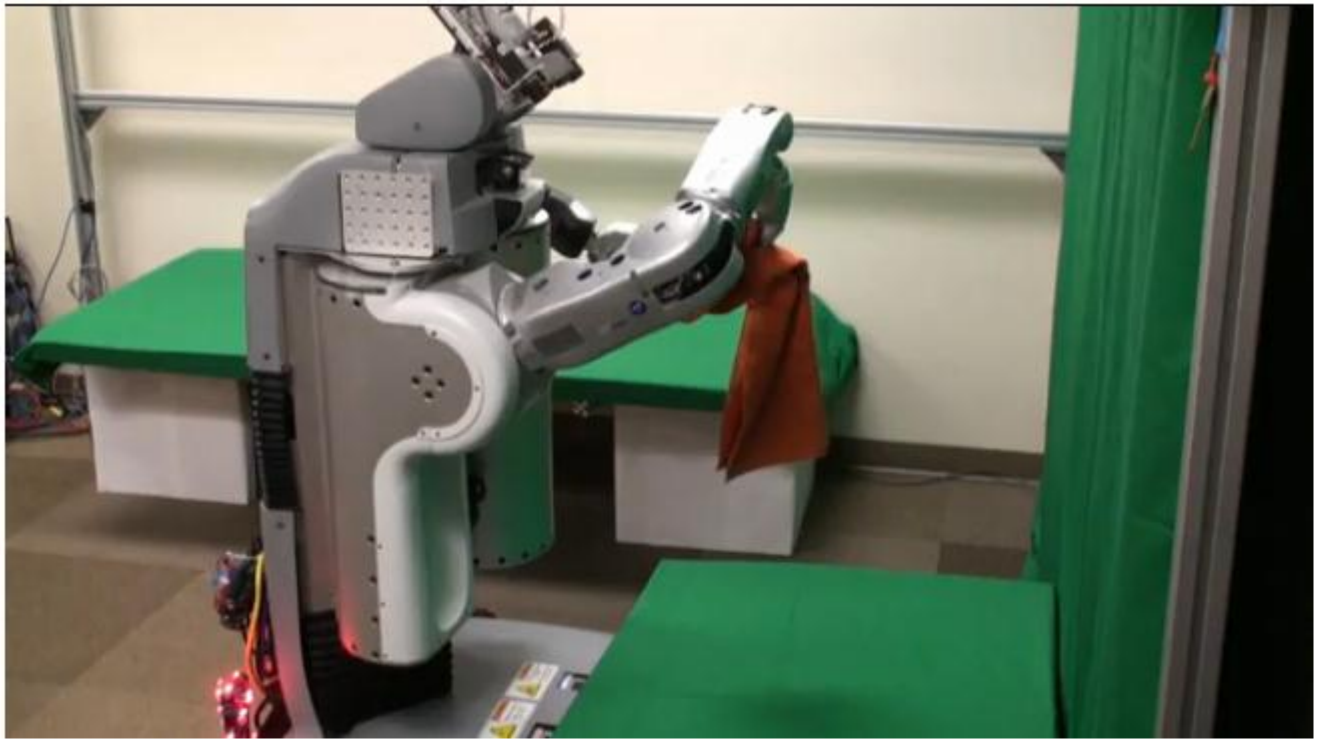
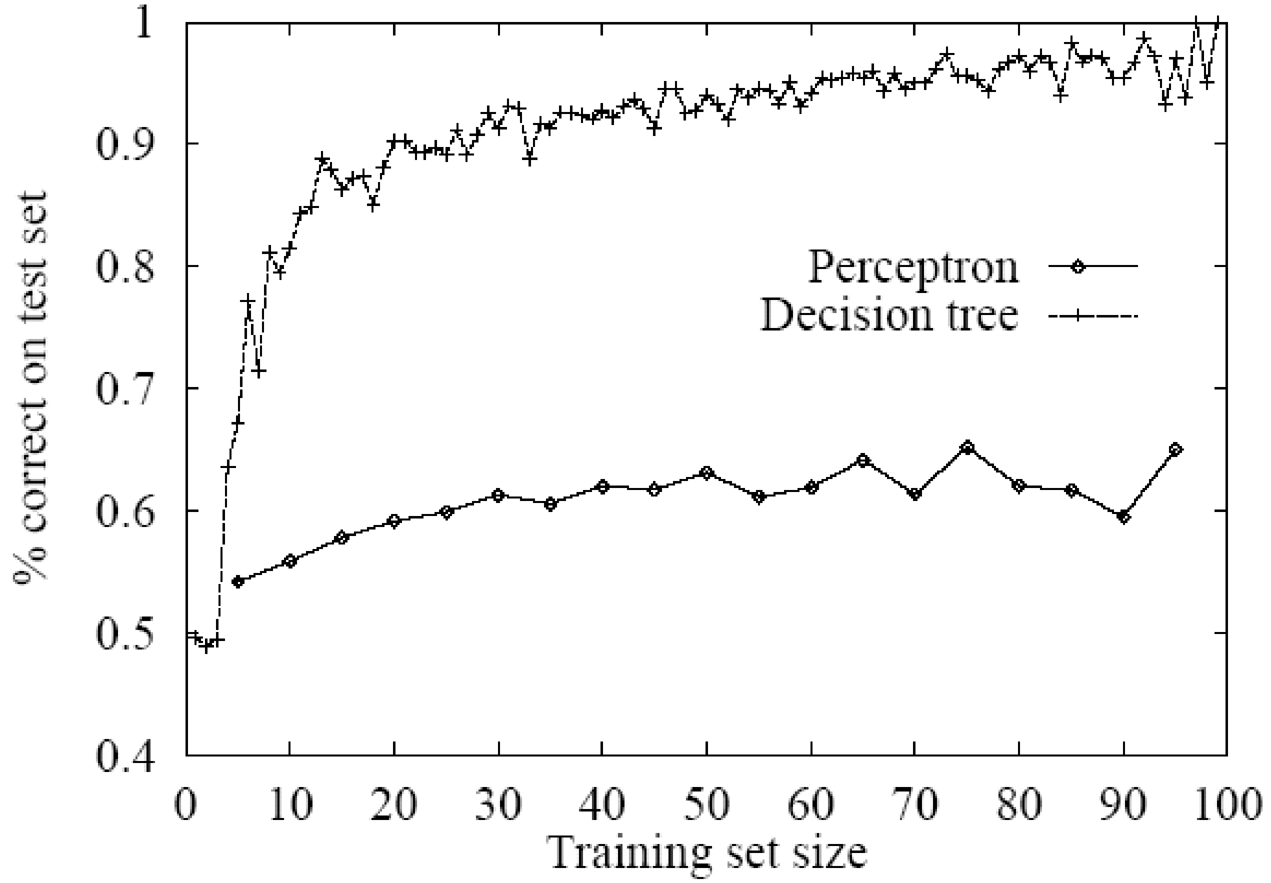
Artificial Neural Networks

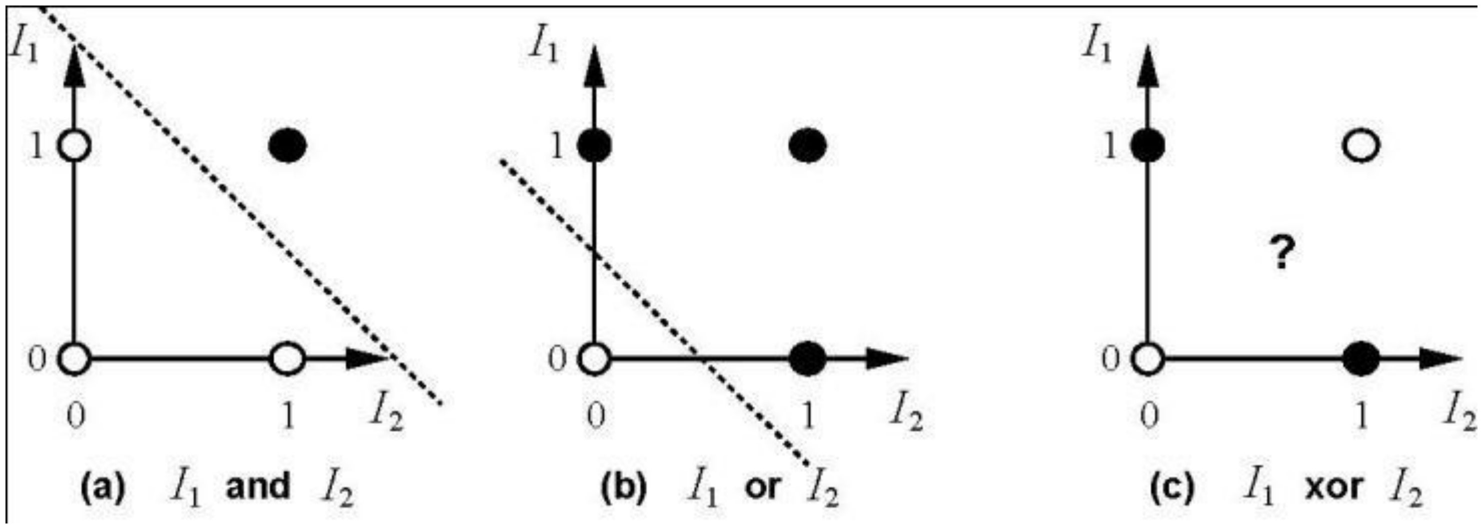
The future of AI



Restaurant Data Set



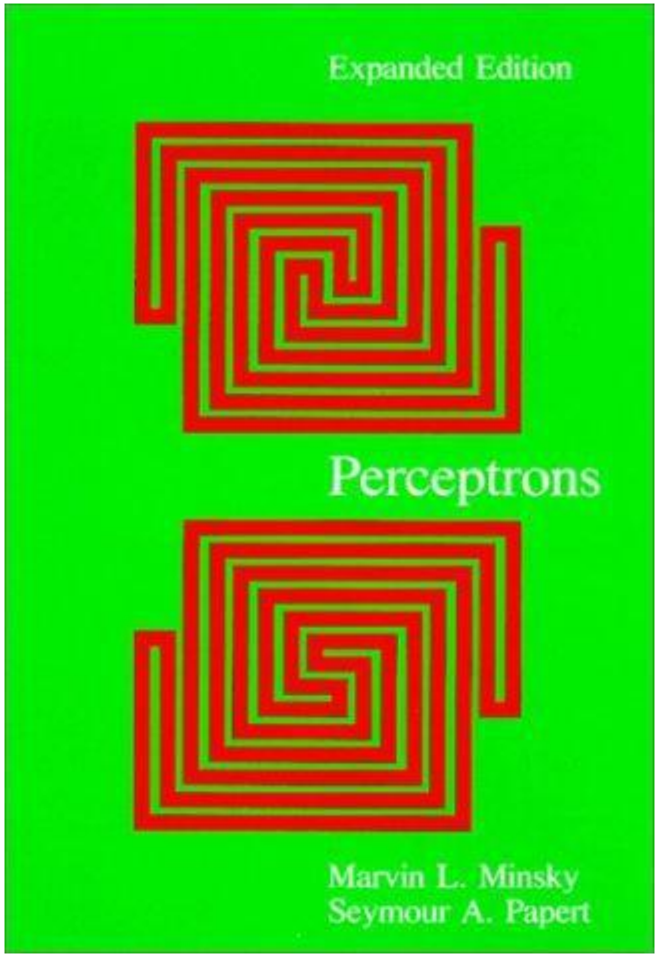
Limited Expressiveness of Perceptrons



(a) I_1 and I_2 (b) I_1 or I_2 (c) I_1 xor I_2

The XOR affair

- Minsky and Papert (1969) showed certain simple functions cannot be represented (e.g. Boolean XOR). Killed the field!
- Mid 80th: Non-linear Neural Networks (Rumelhart et al. 1986)

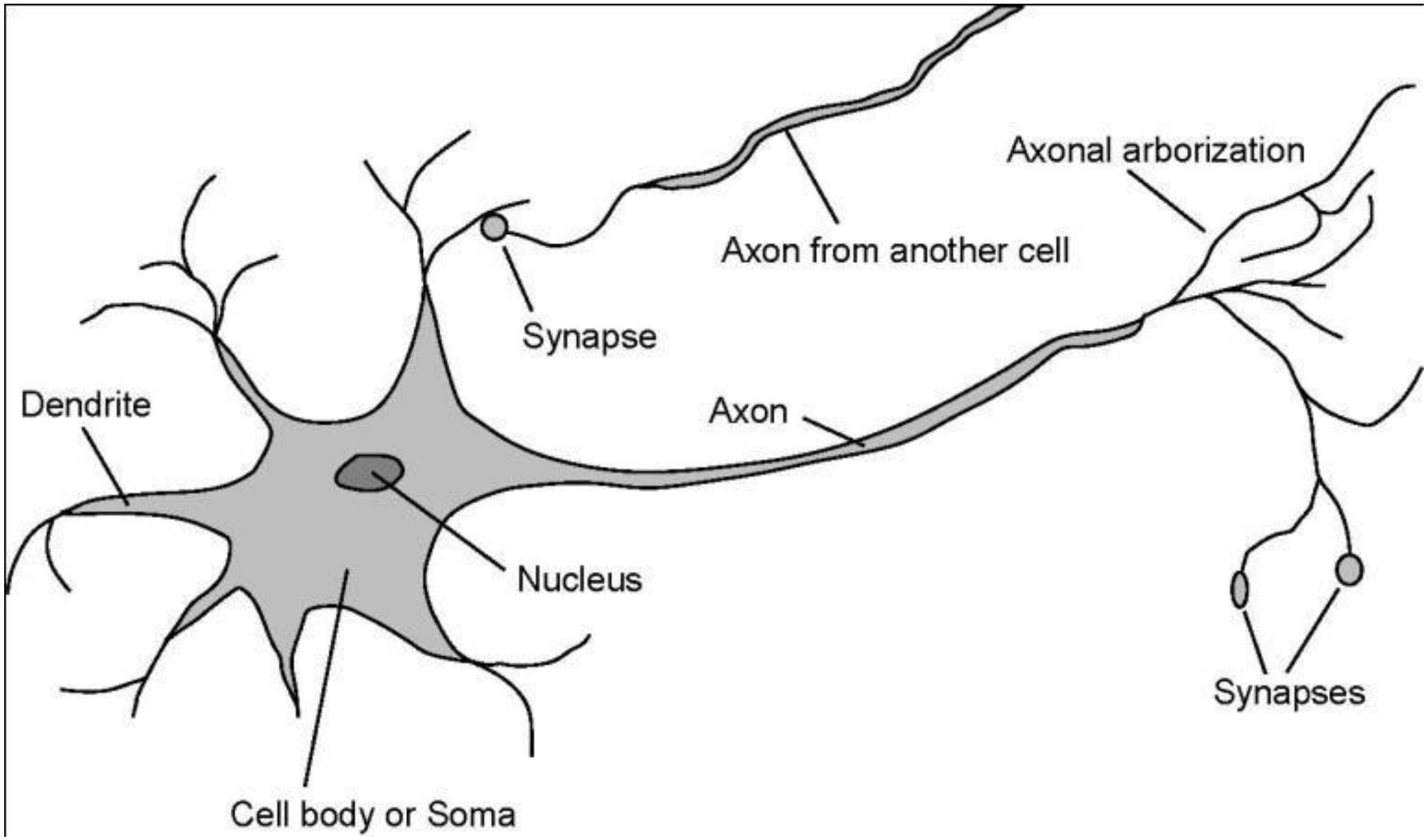


The XOR affair

Neural Networks

- Rich history, starting in the early forties (McCulloch and Pitts 1943).
- Two views:
  - **Modeling the brain**
  - **“Just” representation of complex functions** (Continuous; contrast decision trees)
- Much progress on both fronts.
- Drawn interest from: *Neuroscience, Cognitive science, AI, Physics, Statistics, and CS/EE.*

Neuron


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Neural Structure

1. Cell body; one axon (delivers output to other connect neurons); many dendrites (provide surface area for connections from other neurons).

2. Axon is a single long fiber. 100 or more times the diameter of cell body. Axon connects via **synapses** to **dendrites** of other cells.

3. Signals propagated via complicated electrochemical reaction.

4. Each neuron is a “threshold unit”. Neurons do nothing unless the collective influence from all inputs reaches a threshold level.

5. Produces full-strength output. “fires”. Stimulation at some **synapses** encourages neurons to fire; some discourage from firing.

6. Synapses can increase (**excitatory**) or decrease (**inhibitory**) potential (signal

Why Neural Nets?

Motivation:

Solving problems under the constraints similar to those of the brain may lead to solutions to AI problems that would otherwise be overlooked.

- Individual neurons operate very slowly

  But the brain does complex tasks fast: $\rightarrow$ *massively parallel algorithms*

- Neurons are failure-prone devices

  But brain is reliable anyway $\rightarrow$ *distributed representations*

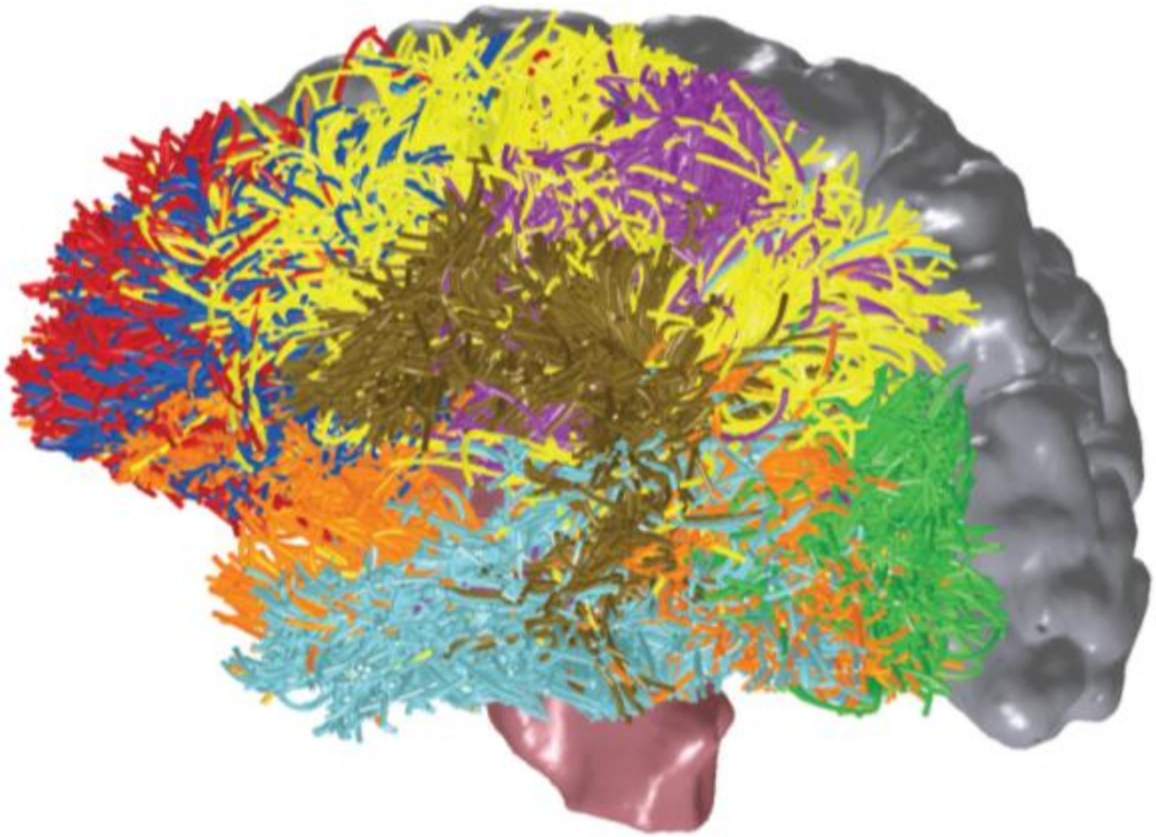
- Neurons promote approximate matching

  less brittle $\rightarrow$ *learnable*

Brains and Bytes

Computational neuroscientists are learning that the brain is like a computer, except when it isn't.

The current puzzle is to understand how a brain built from fundamentally unreliable components can reliably perform tasks that digital computers have barely begun to crack.



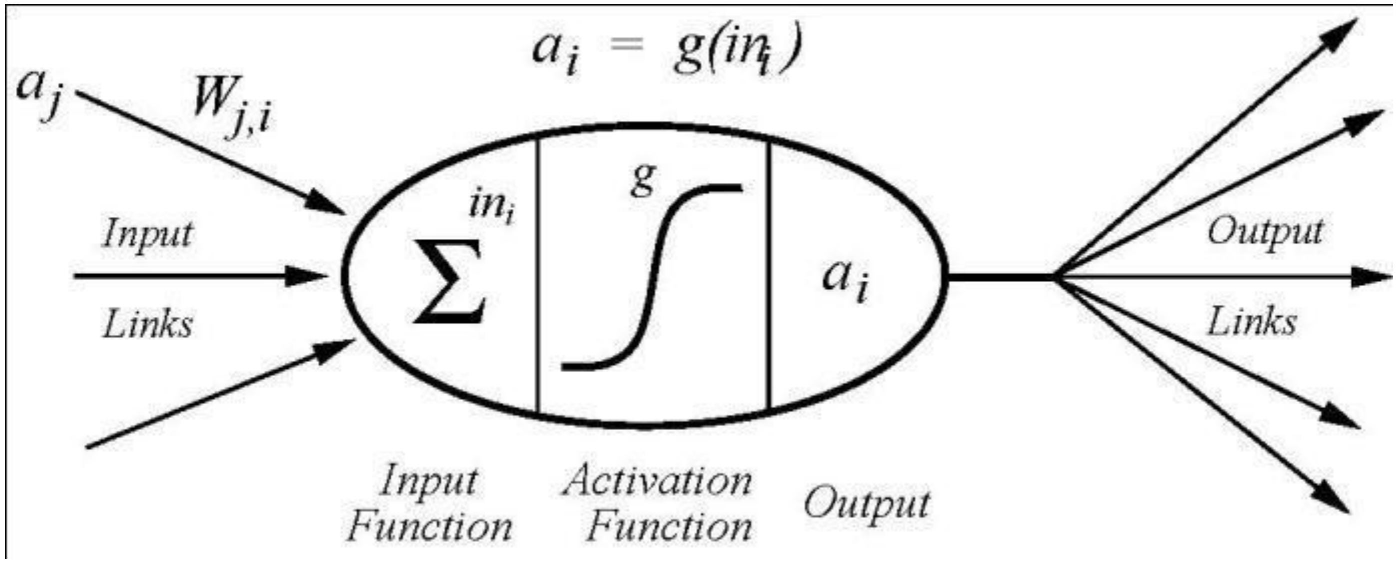
In collaboration with researchers from Stanford University, IBM scientists have developed an algorithm that uses the Blue Gene supercomputing architecture to noninvasively measure and map the connections between all cortical and sub-cortical locations within the human brain.

Connectionist Models of Learning

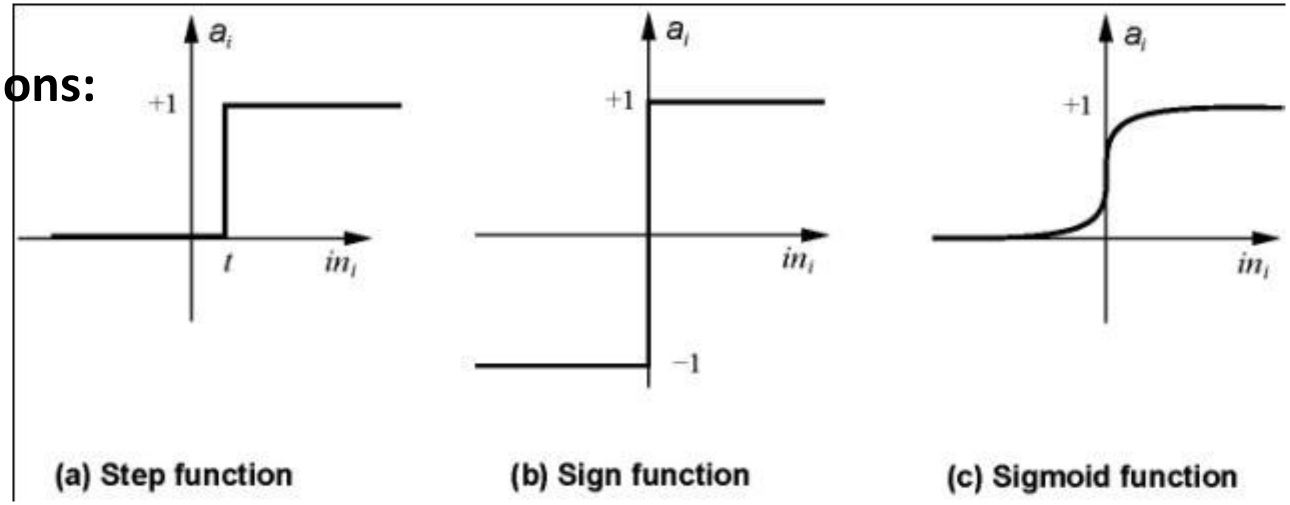
Characterized by:

- A large number of very simple neuron-like processing elements.
- A large number of weighted connections between the elements.
- Highly parallel, distributed control.
- An emphasis on learning internal representations automatically.

Artificial Neurons



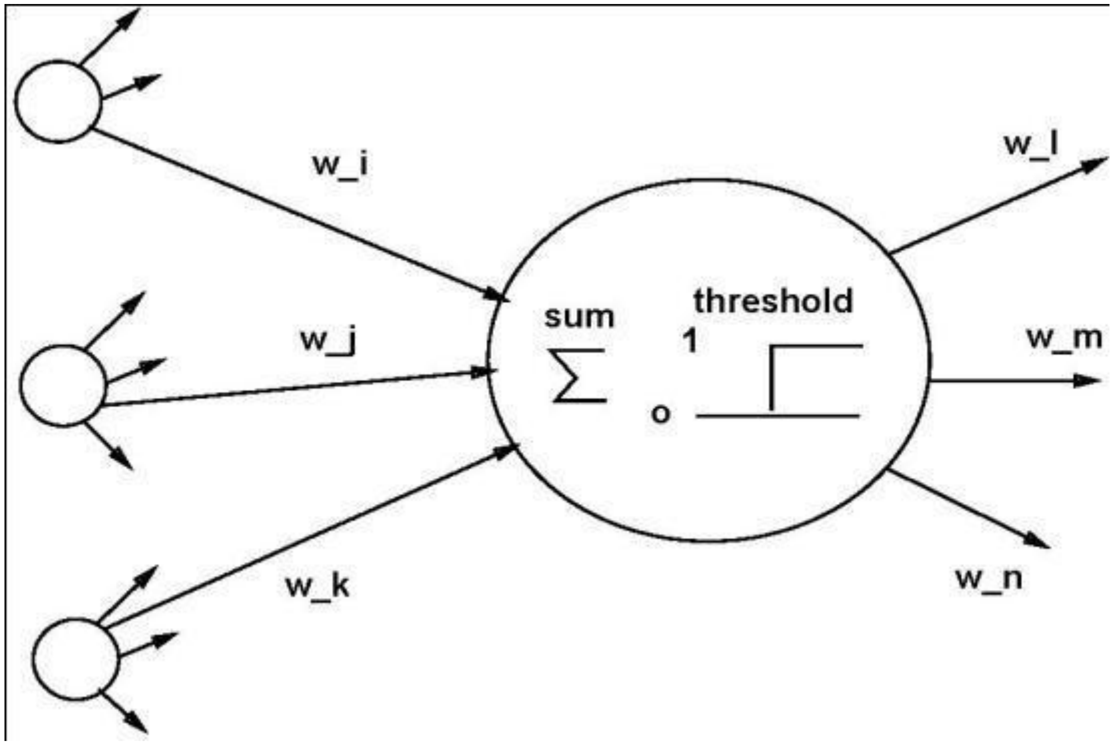
Activation Functions:



(a) Step function **(b) Sign function** **(c) Sigmoid function**

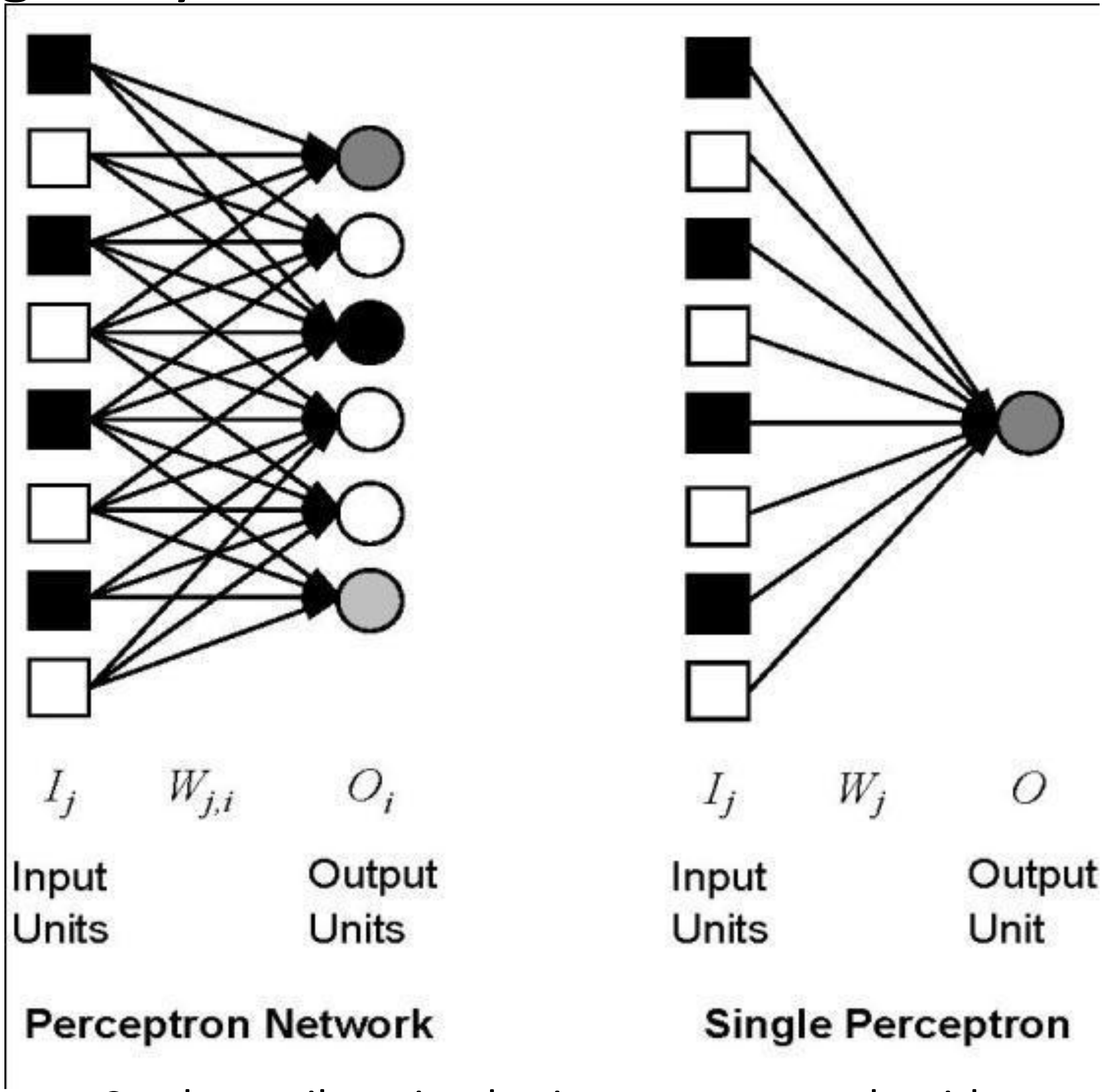
$\text{step}_t(x) = 1$, if $x \geq t$; otherwise 0. $\text{sign}(x) = +1$, if $x \geq 0$; otherwise -1 $\text{sigmoid}(x) = 1/(1+e^{-x})$

Example: Perceptron



Perceptrons

Single Layer Feed Forward Neural Networks



Can be easily trained using perceptron algorithm

2-Layer Feedforward Networks

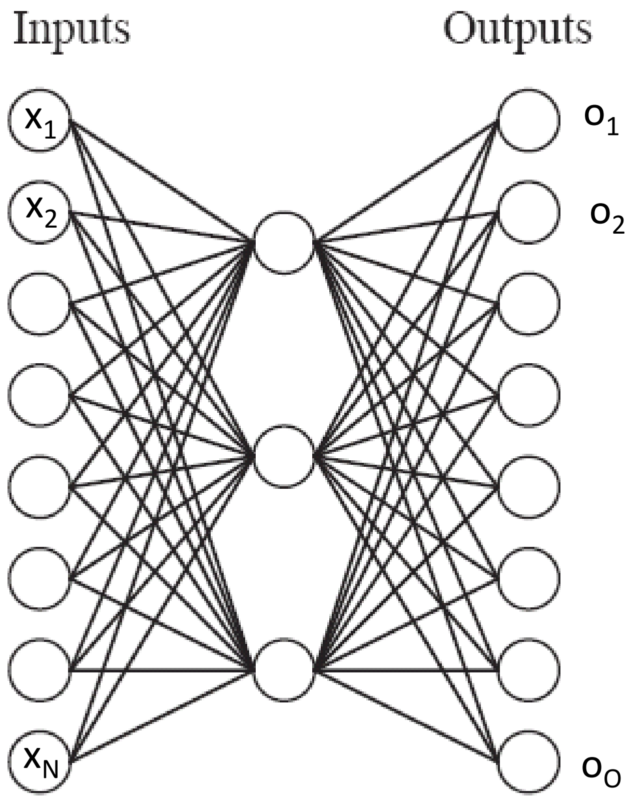
Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- But might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]

Any function can be approximated to arbitrary accuracy by a network with tw hidden layers [Cybenko 1988].

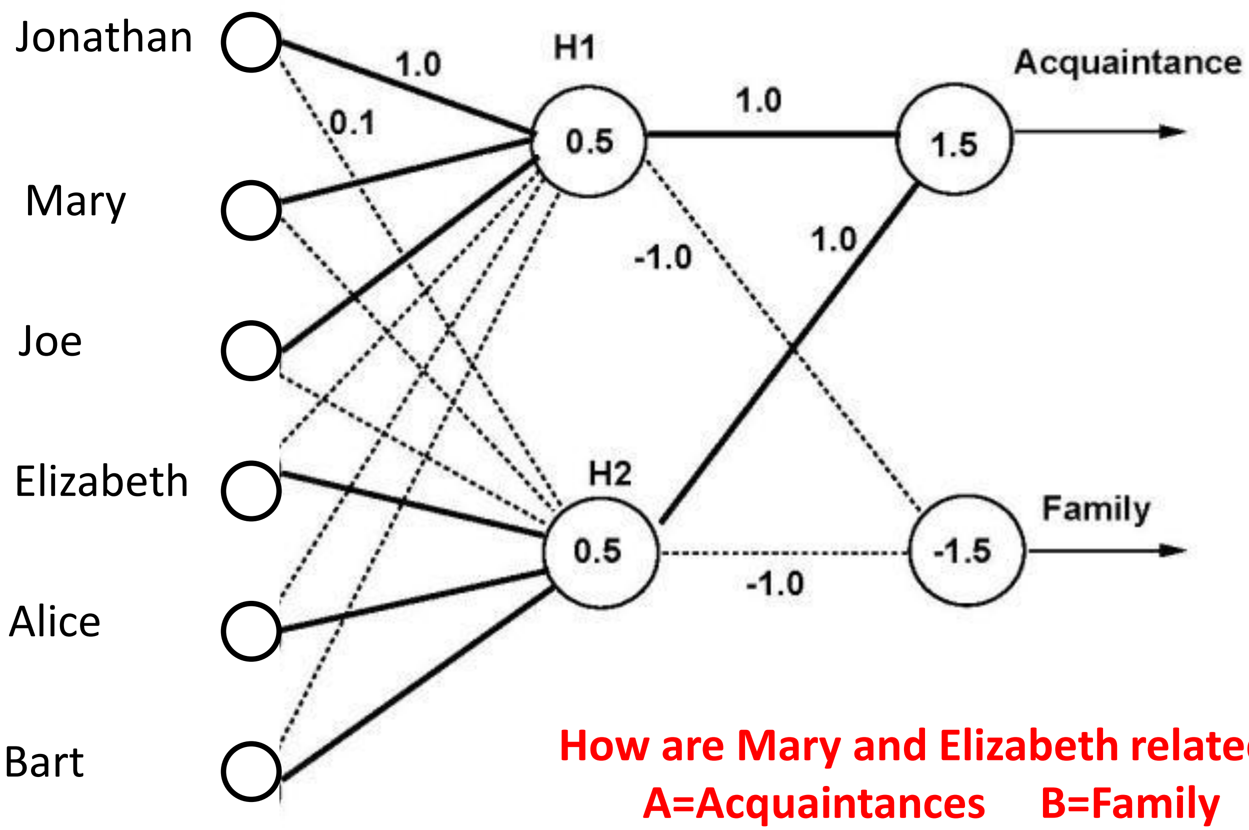


$$o_i = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_j\right)\right)$$

Multi-Layer Nets

- Fully connected, two layer, feedforward

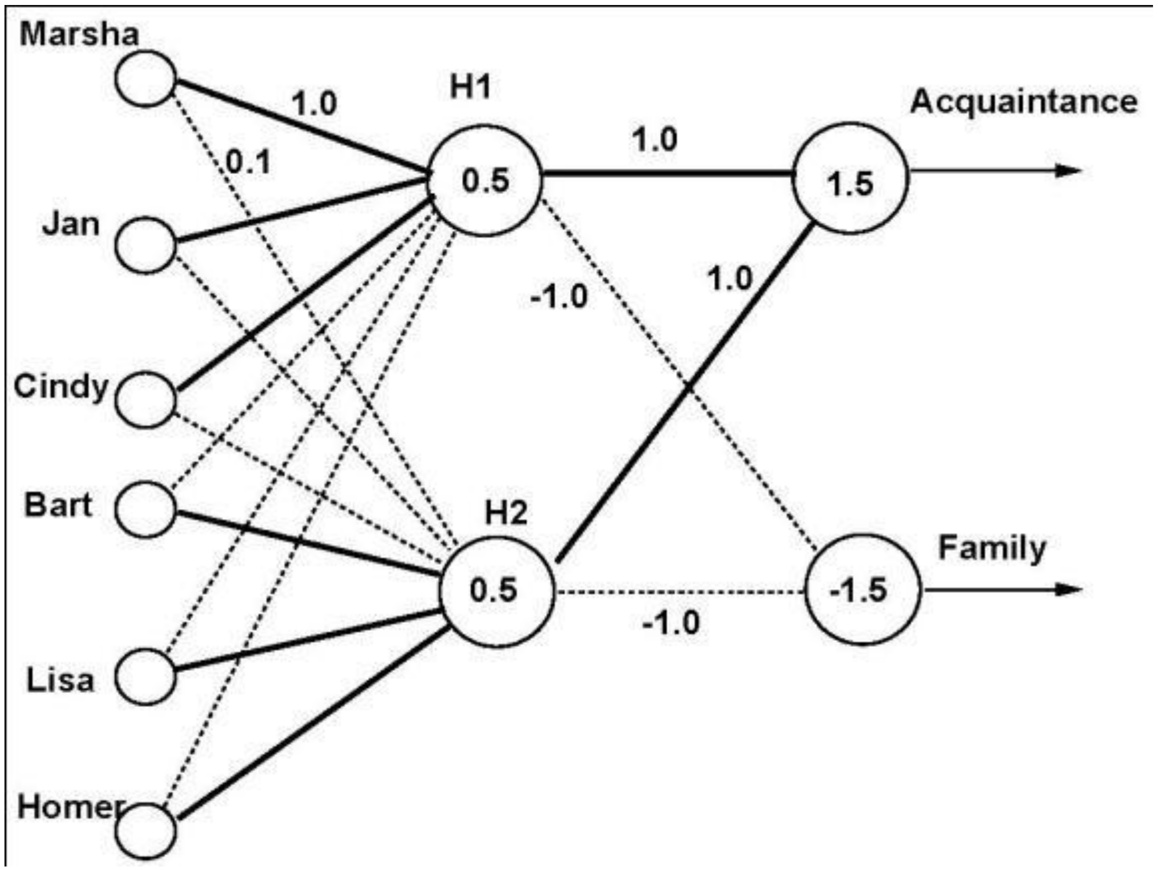
Activation function: g(x) = (1 if greater than threshold, 0 otherwise)

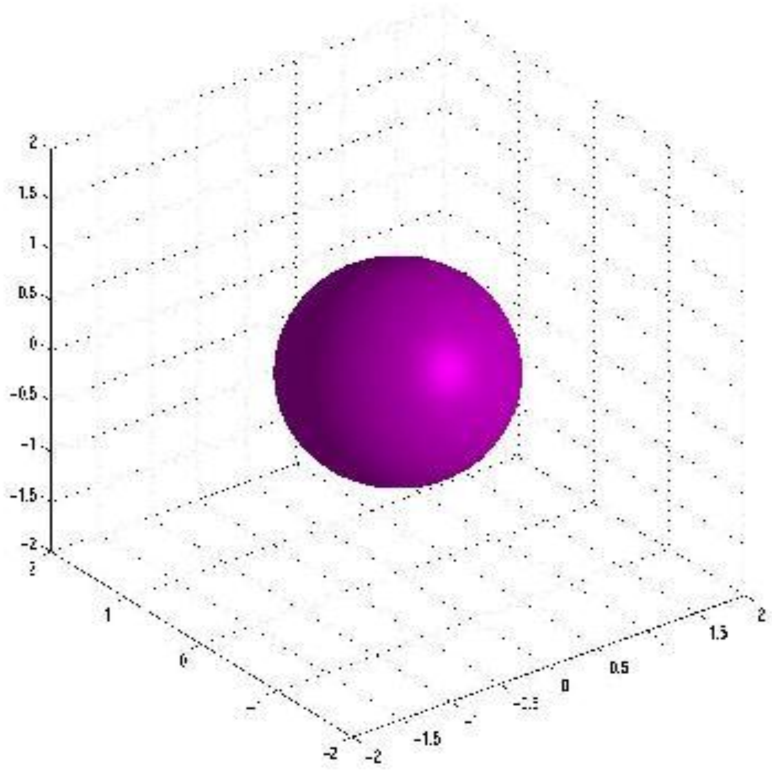


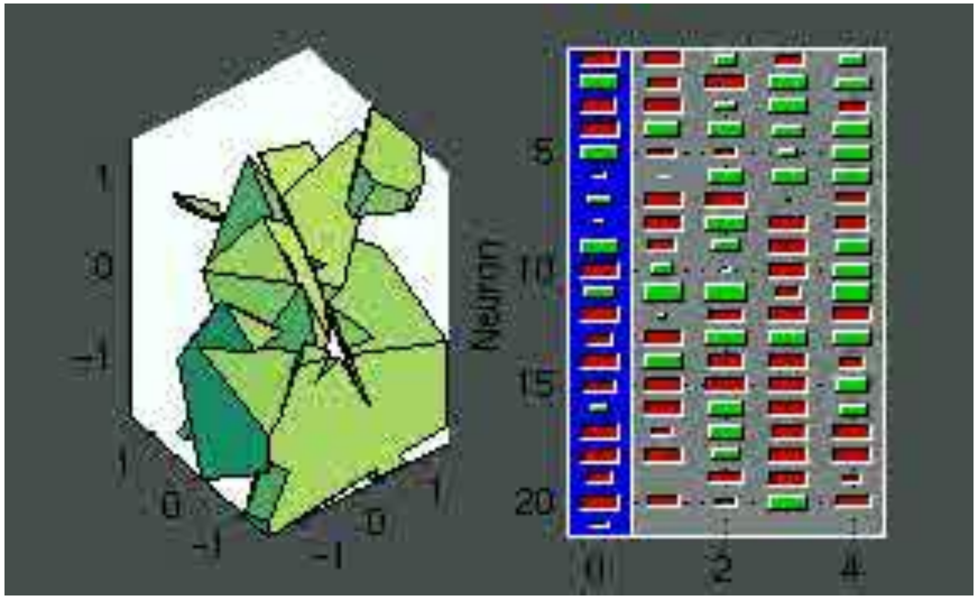
How are Mary and Elizabeth related?
A=Acquaintances B=Family

Multi-Layer Nets

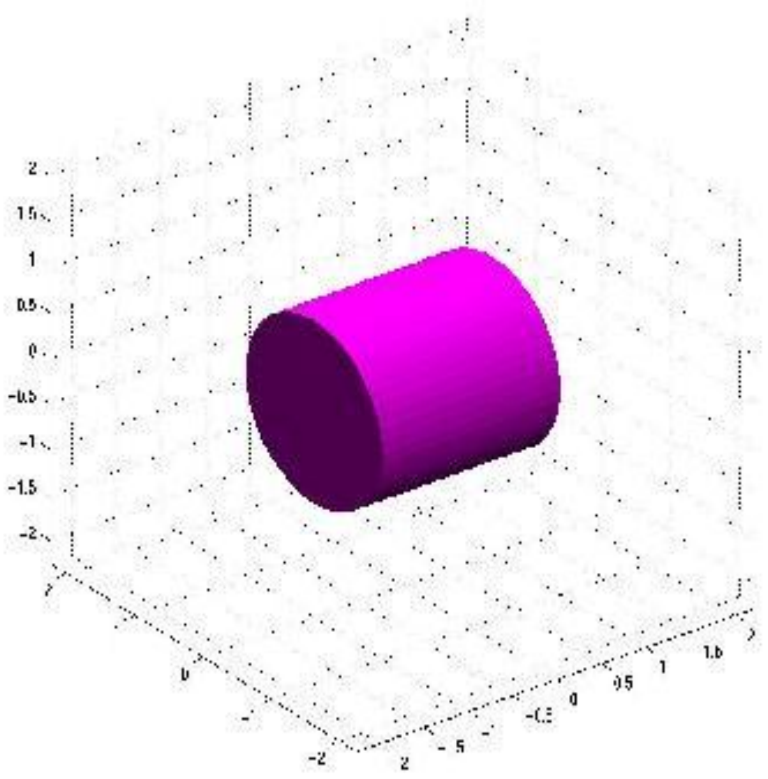
- Fully connected, two layer, feedforward

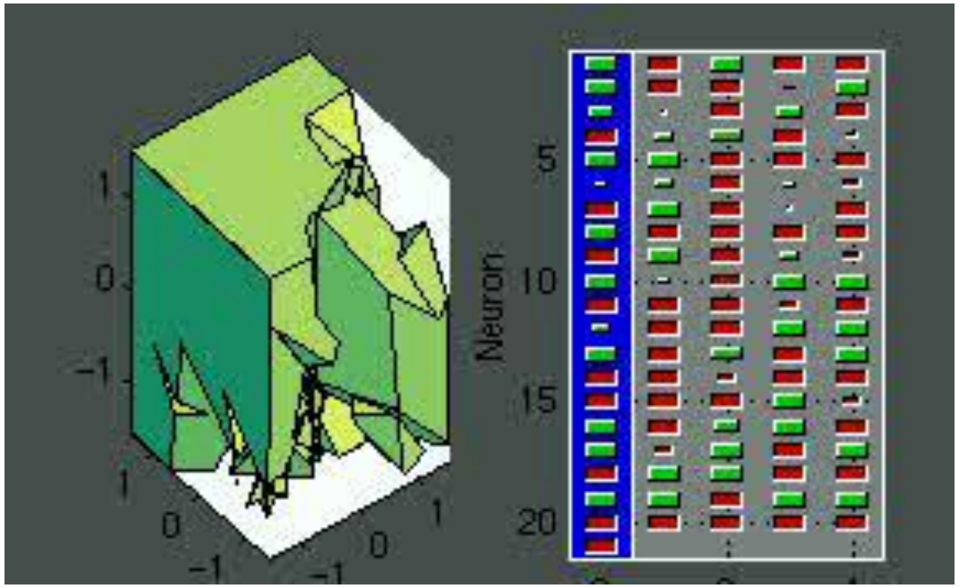




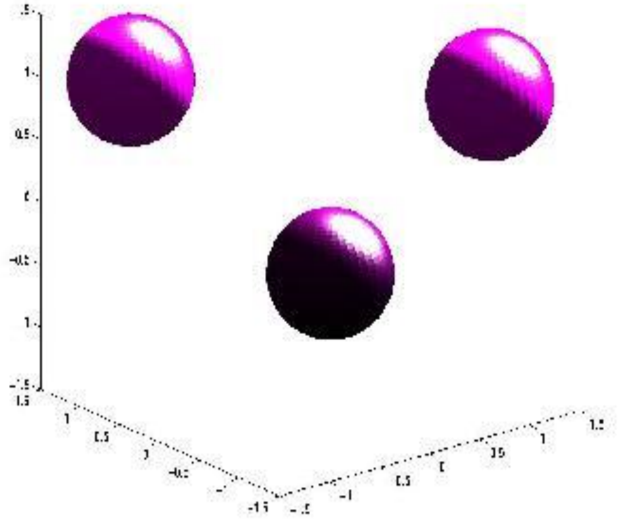


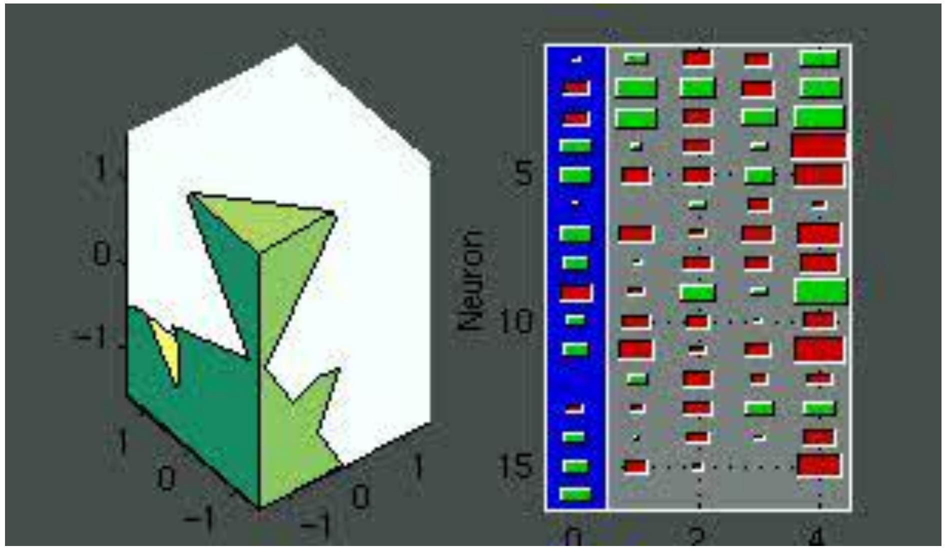
Ofer Melnik, http://www.demo.cs.brandeis.edu/pr/DIBA





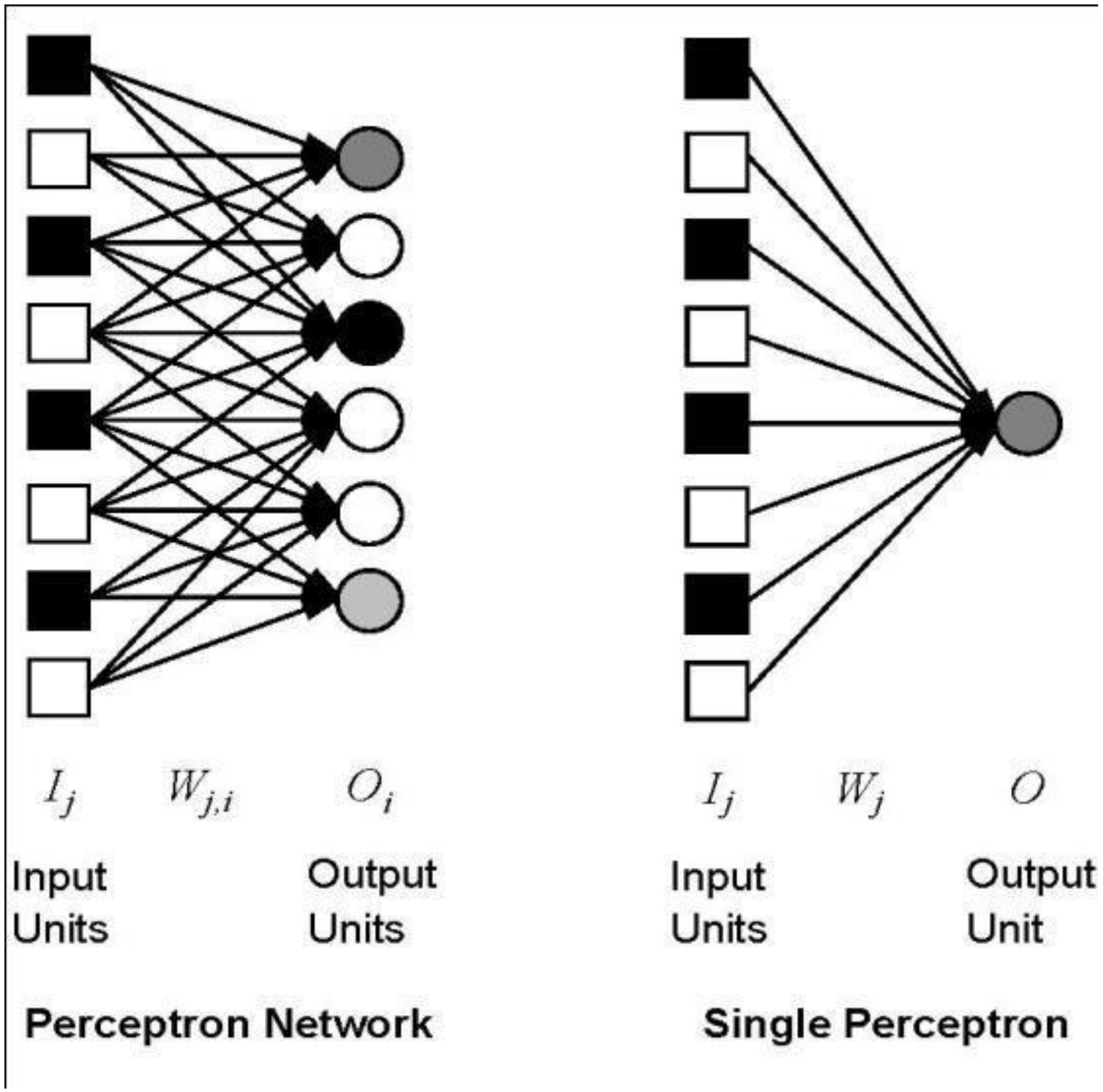
Ofer Melnik, http://www.demo.cs.brandeis.edu/pr/DIBA





Ofer Melnik, http://www.demo.cs.brandeis.edu/pr/DIBA

How can we train perceptrons?



Hebbian learning

- D. O. Hebb:
  - The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." (Hebb 1949, p. 70)
  - "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell." (Hebb 1949, p. 63)
- *Cells that fire together, wire together*
  - If error is small, increase magnitude of connections that contributed.
  - If error is large, decrease magnitude of connections that contributed.

Backpropagation

- Classical measure of error
  - Sum of square errors $E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$
  - $h_w(x)$ is output on perceptron on x.
- Gradient decent using partial derivatives

$$\begin{aligned}
\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\
&= Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right) \\
&= -Err \times g'(in) \times x_j\ ,
\end{aligned}$$

- Update weights $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$

Backpropagation Training (Overview)

Training data:

- $(x_1,y_1),\ldots,(x_n,y_n)$, with target labels $y_z \in \{0,1\}$

Optimization Problem (single output neuron):

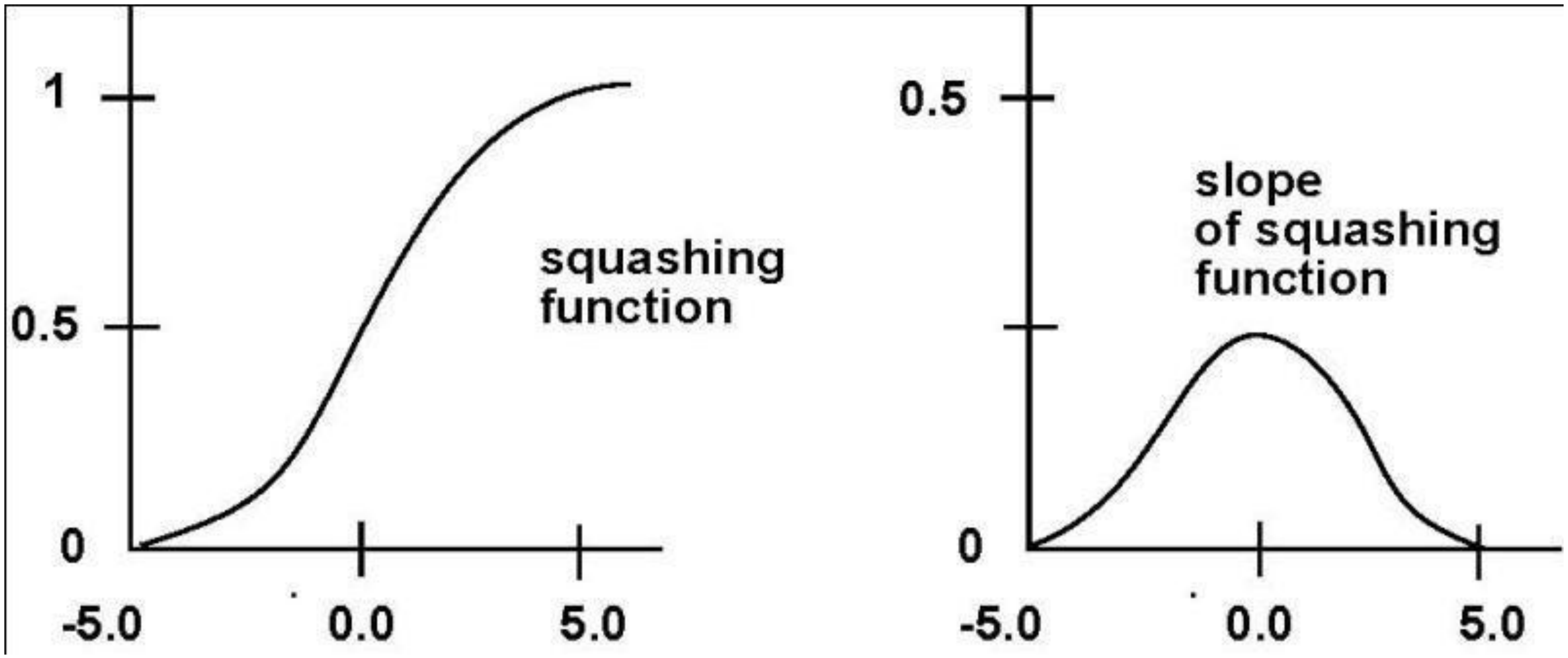
- Variables: network weights $w_{i\rightarrow j}$
- Obj.:$E=\min_w \sum_{z=1..n}(y_z - o(x_z))^2$, $o_i(x_z) = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_{z,j}\right)\right)$
- Constraints: none

Algorithm: local search via gradient descent.

- Randomly initialize weights.
- Until performance is satisfactory,
  - Compute partial derivatives $(\partial E / \partial w_{i\rightarrow j})$ of objective function E for each weight $w_{i\rightarrow j}$
  - Update each weight by $w_{i\rightarrow j}$ Ã $w_{i\rightarrow j} + \alpha\,(\partial E / \partial w_{i\rightarrow j})$

Smooth and Differentiable Threshold Function

- Replace sign function by a differentiable activation function $g(x) = \frac{1}{1+e^{-x}}$
  → sigmoid function:



Slope of Sigmoid Function

$$f(x) = \frac{1}{1+e^{-x}}$$

Slope: $\frac{df(x)}{dx} = \frac{d}{dx}(\frac{1}{1+e^{-x}})$

$$= (1+e^{-x})^{-2} e^{-x}$$

$$= \frac{e^{-x}}{(1+e^{-x})(1+e^{-x})}$$

$$= f(x)\frac{e^{-x}}{(1+e^{-x})}$$

$$= f(x)(1-f(x))$$

View in terms of output at node:

$$= o_j(1-o_j)$$

Backpropagation Training (Detail)

- Input: training data $(x_1,y_1),\ldots,(x_n,y_n)$, learning rate parameter α.
- Initialize weights.
- Until performance is satisfactory
  - **For each training instance,**
    - **Compute the resulting output**
    - **Compute $\beta_z = (y_z - o_z)$ for nodes in the output layer**
    - **Compute $\beta_j = \sum_k w_{j \rightarrow k}\, o_k\, (1 - o_k)\, \beta_k$ for all other nodes.**
    - **Compute weight changes for all weights using**

$$\Delta w_{i \rightarrow j}(l) = o_i\, o_j\, (1 - o_j)\, \beta_j$$

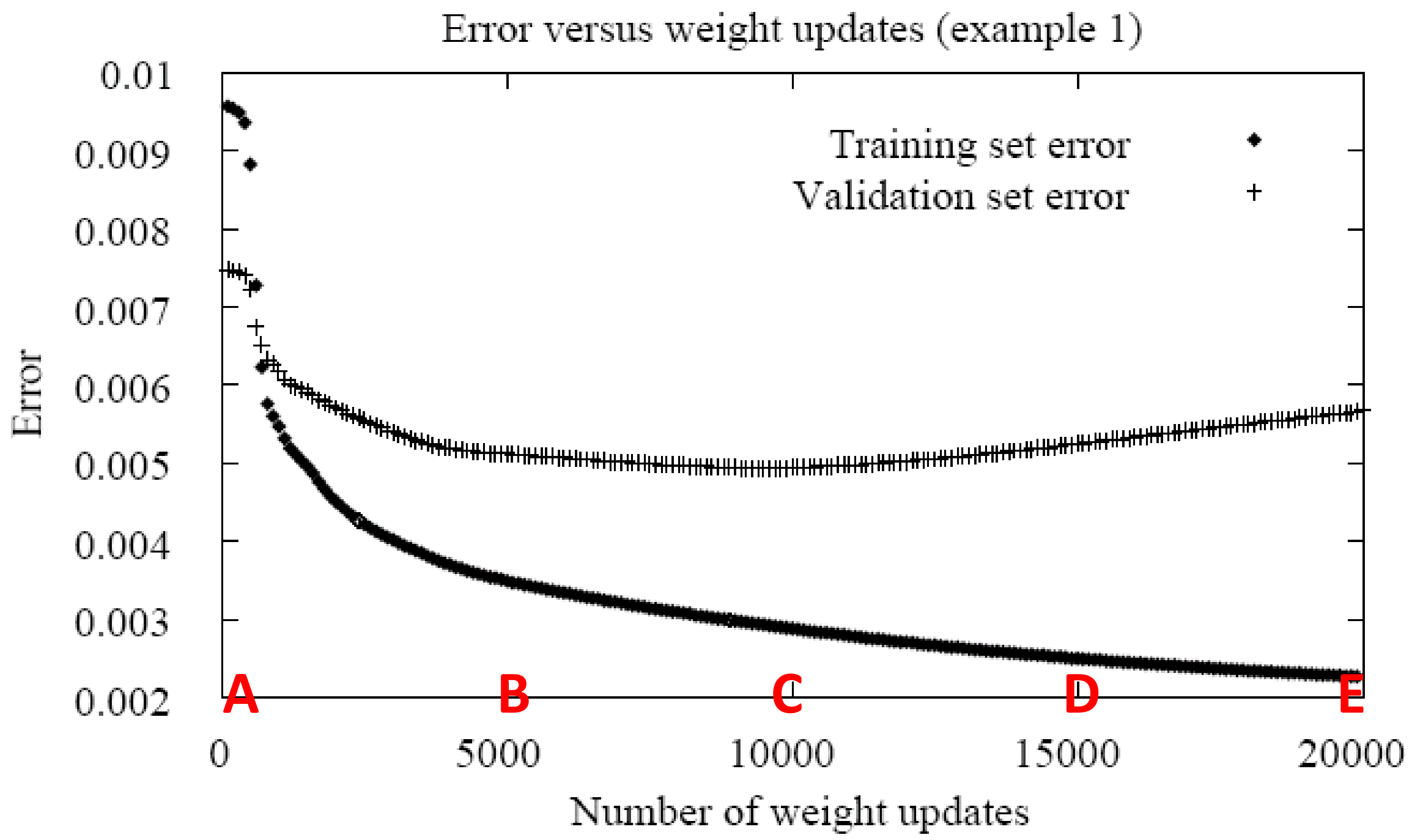
  - **Add up weight changes for all training instances, and update the weights accordingly.**

$$w_{i \rightarrow, j} \leftarrow w_{i \rightarrow, j} + \alpha \sum_l \Delta w_{i \rightarrow, j}(l)$$

Summary: Hidden Units

- Hidden units are nodes that are situated between the input nodes and the output nodes.
- Hidden units allow a network to learn non-linear functions.
- Hidden units allow the network to represent combinations of the input features.
- Given too many hidden units, a neural net will simply memorize the input patterns (overfitting).
- Given too few hidden units, the network may not be able to represent all of the necessary generalizations (underfitting).

How long should you train the net?



When would you stop training?

How long should you train the net?

- The goal is to achieve a balance between correct responses for the training patterns and correct responses for new patterns.
  - That is, a balance between memorization and generalization)
- If you train the net for too long, then you run the risk of overfitting.
  - Select number of training iterations via cross-validation on a holdout set.

Regularization

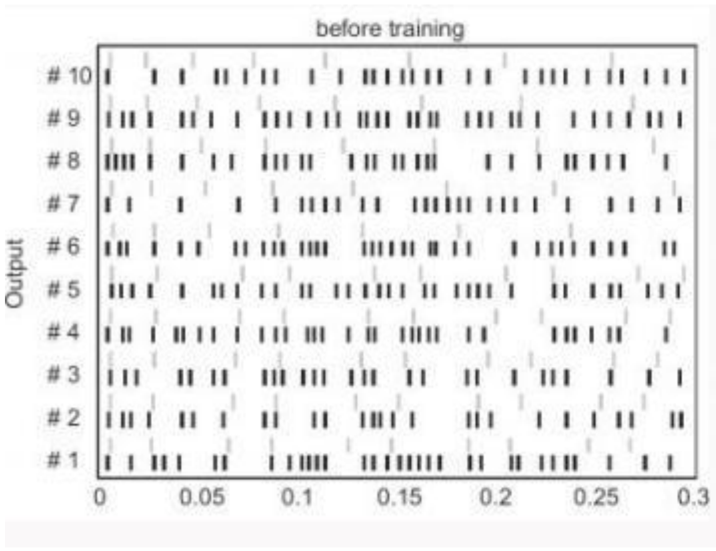
- Simpler models are better
- NN with smaller/fewer weights are better
  - Add penalty to total sum of absolute weights
  - Pareto optimize

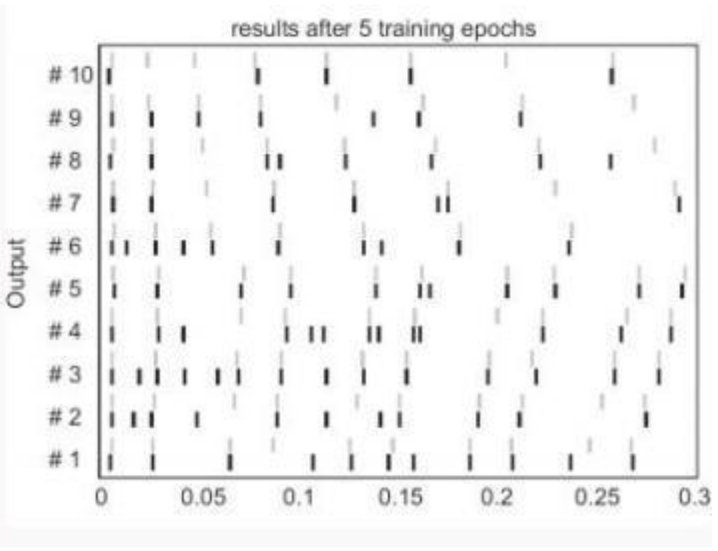
Design Decisions

- Choice of learning rate α
- Stopping criterion – when should training stop?
- Network architecture
  - How many hidden layers? How many hidden units per layer?
  - How should the units be connected? (Fully? Partial? Use domain knowledge?)
- How many restarts (local optima) of search to find good optimum of objective function?

Spiking Nets

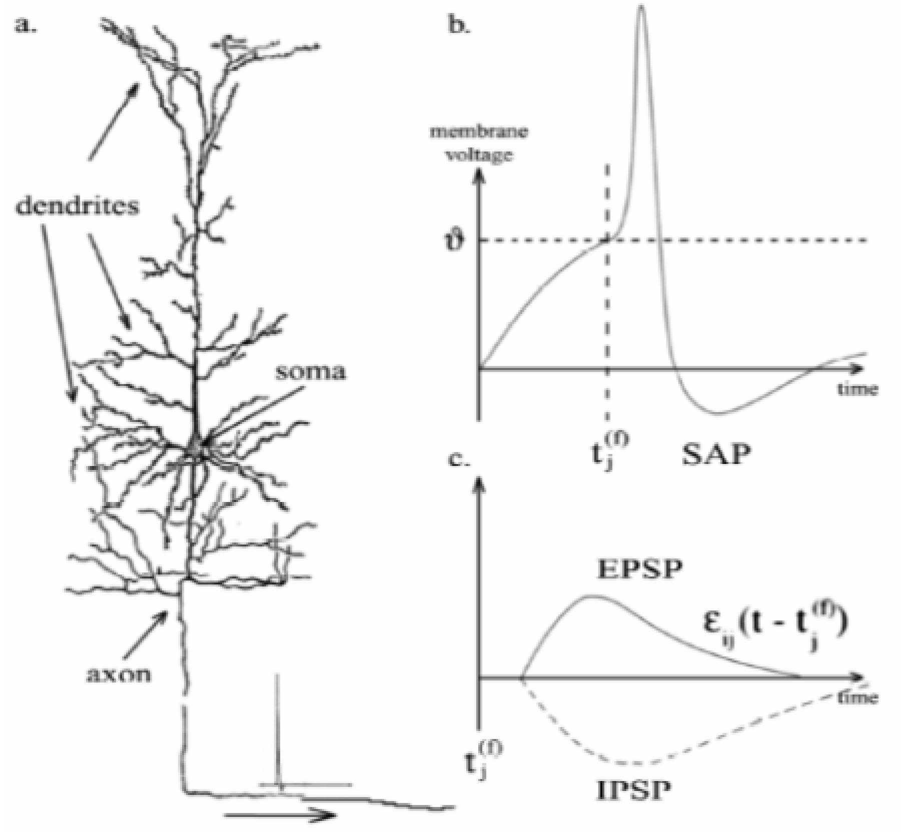
- Represent continues values using rates
  - Output spike if # of incoming spikes > threshold
  - Leaky counter

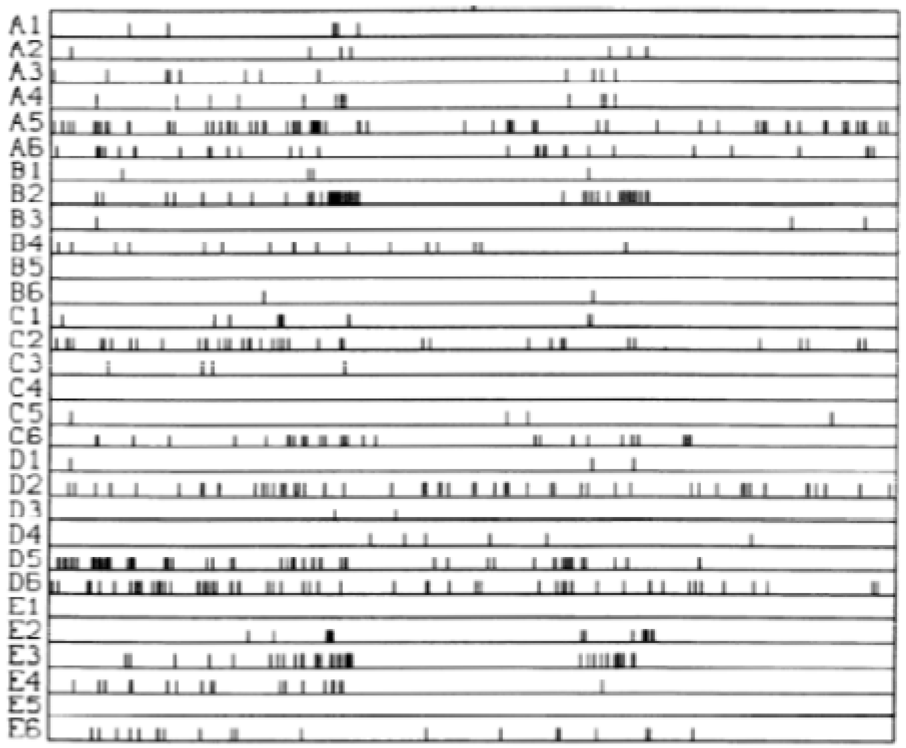




http://www.ine-news.org

Spiking

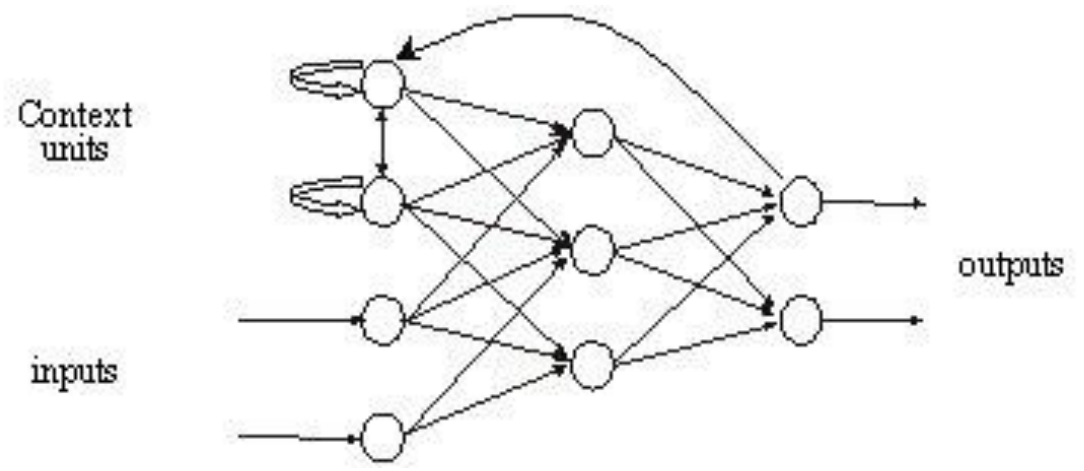




From http://www.cs.uu.nl/research/techreps/repo/CS-2003/2003-008.pdf

Recurrent networks

- Nodes connect
  - Laterally
  - Backwards,
  - To themselves
- Complex behavio
  - Dynamics, Memory



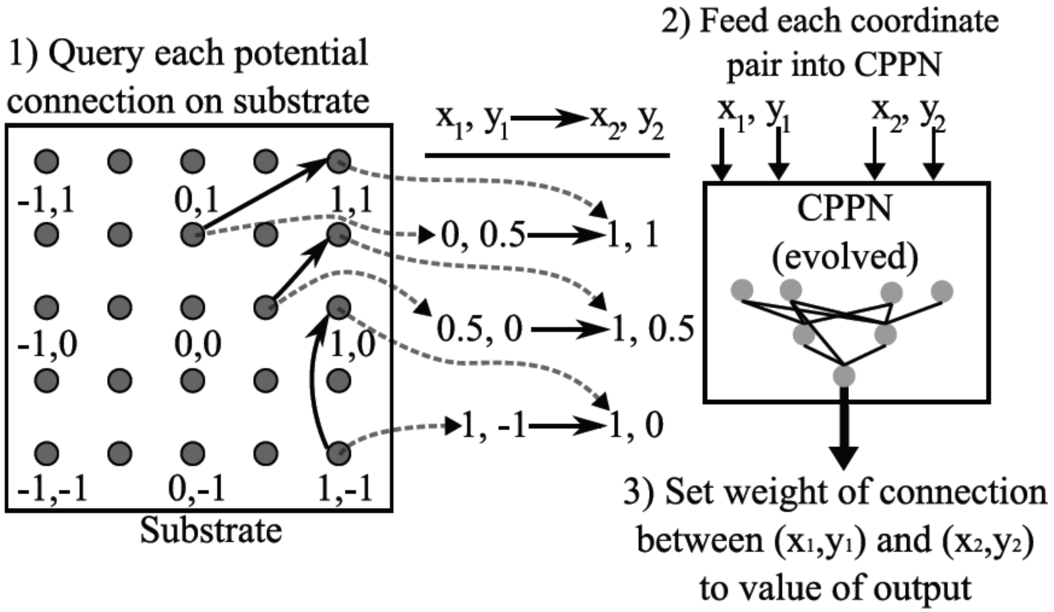
www.stowa-nn.ihe.nl/ANN.htm

Learning Network Topology

- **Optimal Brain Damage algorithm**
  - Trains a fully connected network
  - Removes connections and nodes that contribute least to the performance
    - Using information-theoretic criteria
  - Repeats until performance starts decreasing
- **Tiling algorithm: Grows networks**
  - Start with a small network that classifies many examples
  - Repeatedly add more nodes to classify remaining examples

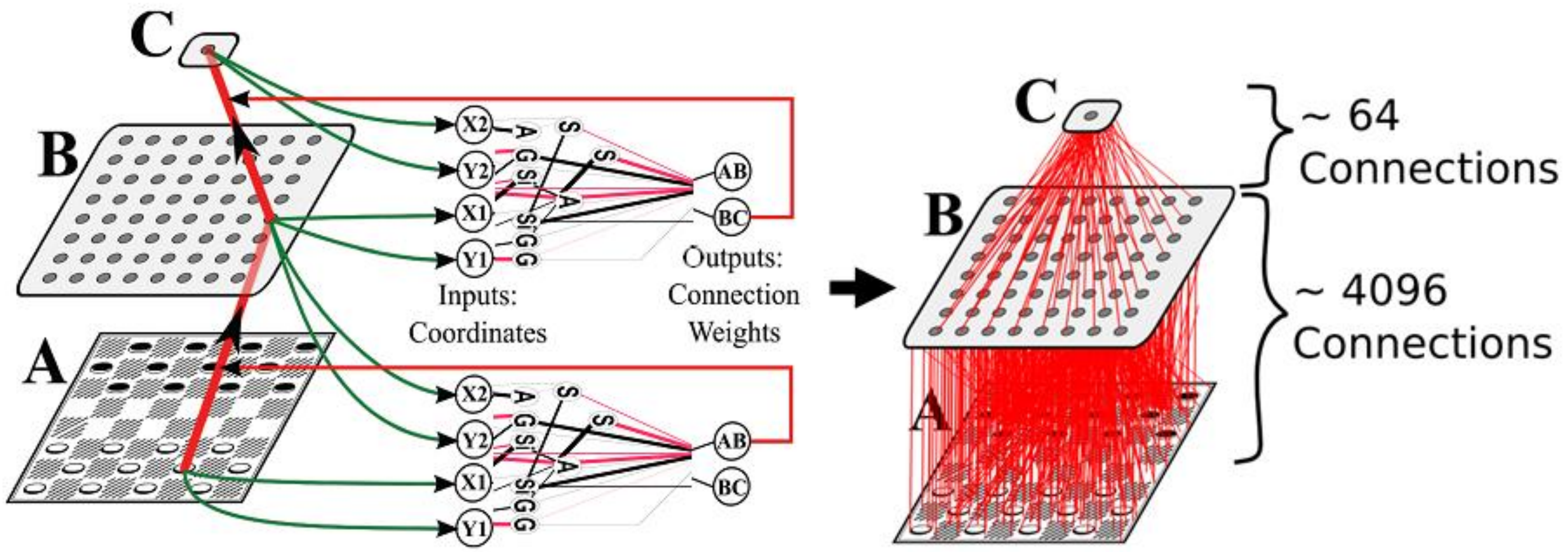
Hyper-Networks

- Use a network to generate a network
  - E.g. to determine connection $\boldsymbol{w_{ij}}$ use network that takes in ***i*** and ***j*** and produces ***w.***
  - *In 2D:*


1) Query each potential connection on substrate
-1,1
0,1
1,1
-1,0
0,0
1,0
-1,-1
0,-1
1,-1
Substrate
$x_1, y_1 \rightarrow x_2, y_2$
0, 0.5 → 1, 1
0.5, 0 → 1, 0.5
1, -1 → 1, 0
2) Feed each coordinate pair into CPPN
x_1, y_1
x_2, y_2
CPPN (evolved)
3) Set weight of connection between (x_1,y_1) and (x_2,y_2) to value of output

Ken Stanley, eplex.cs.ucf.edu

Hyper-Networks



Ken Stanley, eplex.cs.ucf.edu