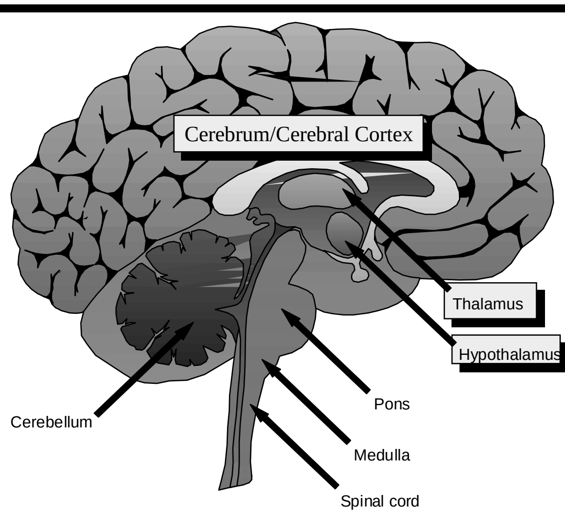
CSE 473 Guest Lecture (Raj Rao): Neural Networks

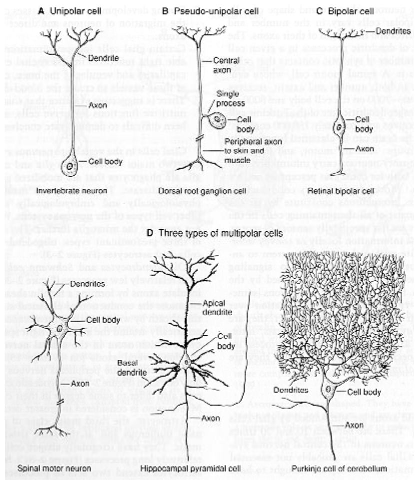
- Outline:
  - The 3-pound universe
  - Those gray cells…
  - Input-Output transformation in neurons
  - Modeling neurons
  - Neural Networks
  - Learning Networks
  - Applications

- Corresponds to Chapter 19 in Russell and Norvig

The 3-pound universe we "live" in

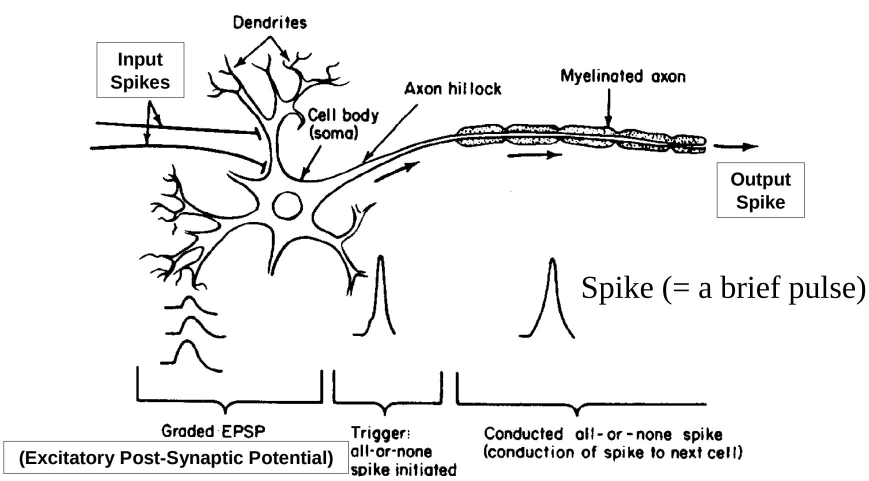


Those gray cells…Neurons



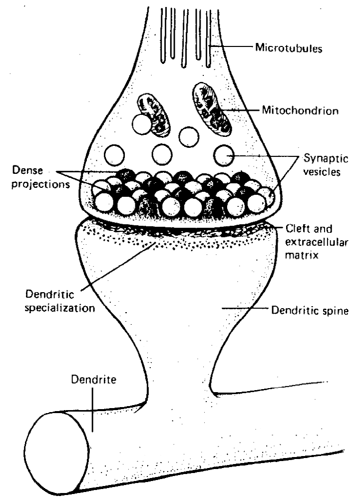
From Kandel, Schwartz, Jessel, Principles of Neural Science, 3rd edn., 1991, pg. 21

Basic Input-Output Transformation in a Neuron

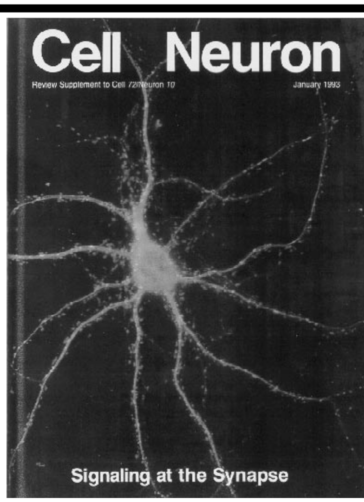


Communication between neurons: Synapses

- Synapses: Connections between neurons
  - Electrical synapses (gap junctions)
  - Chemical synapses (use neurotransmitters)
- Synapses can be excitatory or inhibitory
- Synapses are integral to memory and learning



Distribution of synapses on a real neuron…

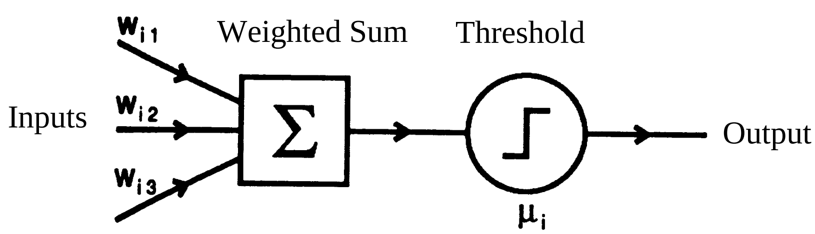


McCulloch–Pitts artificial "neuron" (1943)

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

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

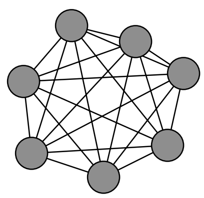
$\Theta(x) = 1$ if $x \geq 0$ and 0 if $x < 0$



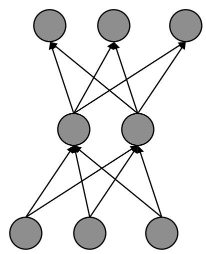
Properties of Artificial Neural Networks

- High level abstraction of neural input-output transformation:
  Inputs → weighted sum of inputs → nonlinear function → output
- Often used where data or functions are uncertain
  - Goal is to *learn* from a set of training data
  - And *generalize* from learned instances to new unseen data
- Key attributes:
  1. Massively parallel computation
  2. Distributed representation and storage of data (in the synaptic weights and activities of neurons)
  3. Learning (networks adapt themselves to solve a problem)
  4. Fault tolerance (insensitive to component failures)

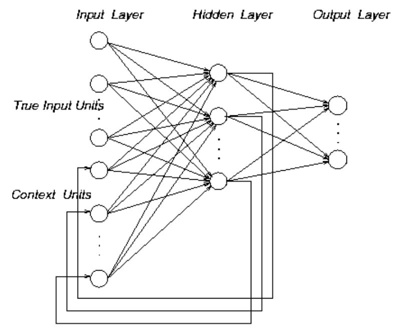
Topologies of Neural Networks



completely connected



feedforward (directed, acyclic)



recurrent (feedback connections)

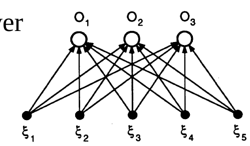
Networks Types

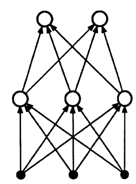
- Feedforward versus recurrent networks
  - Feedforward: No loops, input → hidden layers → output
  - Recurrent: Use feedback (positive or negative)
- Continuous versus spiking
  - Continuous networks model mean spike rate (firing rate)
- Supervised versus unsupervised learning
  - Supervised networks use a "teacher"
    - Desired output for each input is provided by user
  - Unsupervised networks find hidden statistical patterns in input data
    - Clustering, principal component analysis, etc.

Perceptrons

- Fancy name for a type of layered feedforward networks
- Uses McCulloch-Pitts type neuron: $Output_i = \Theta\left[\sum_j w_{ij}\xi_j\right]$
- Output of neuron is 1 if and only if weighted sum of inputs is greater than 0:
  $\Theta(x) = 1$ if $x \geq 0$ and 0 if $x < 0$ (a "step" function)

Single-layer



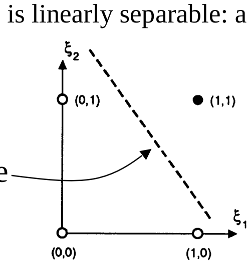


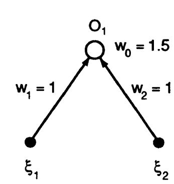
Multilayer

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
  - <u>Only functions that are linearly separable can be computed</u>
  - Example: AND is linearly separable: a AND b = 1 iff a = 1 and b = 1

Linear hyperplane

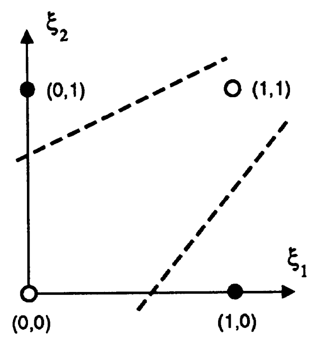




$\xi_o = -1$

Linear inseparability

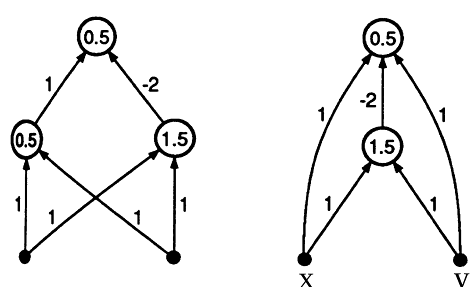
- Single-layer perceptron with threshold units fails if problem is not linearly separable
  - Example: XOR
  - a XOR b = 1 iff (a=0,b=1) or (a=1,b=0)
  - No single line can separate the "yes" outputs from the "no" outputs!



- Minsky and Papert's book showing such negative results was very influential – essentially killed neural networks research for over a decade!

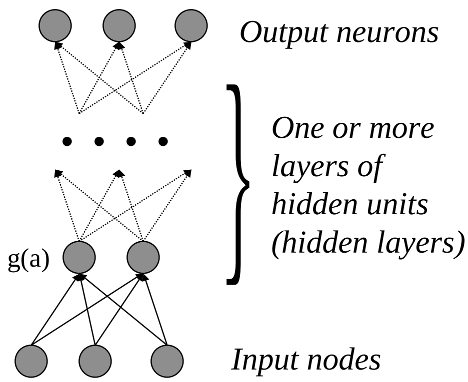
Solution in 1980s: Multilayer perceptrons

- Removes many limitations of single-layer networks
  - Can solve XOR
- Two examples of two-layer perceptrons that compute XOR



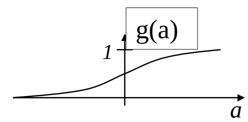
- E.g. Right side network
  - Output is 1 if and only if x + y – 2(x + y – 1.5 > 0) – 0.5 > 0

Multilayer Perceptron



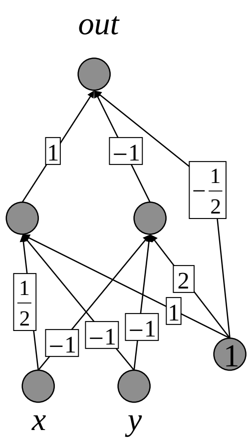
The most common activation functions:
Step function Θ or Sigmoid function:

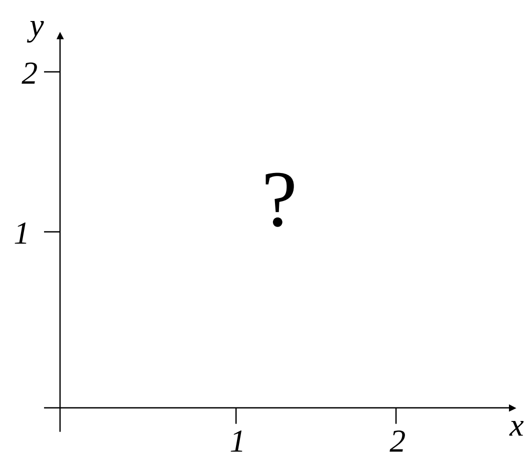
$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



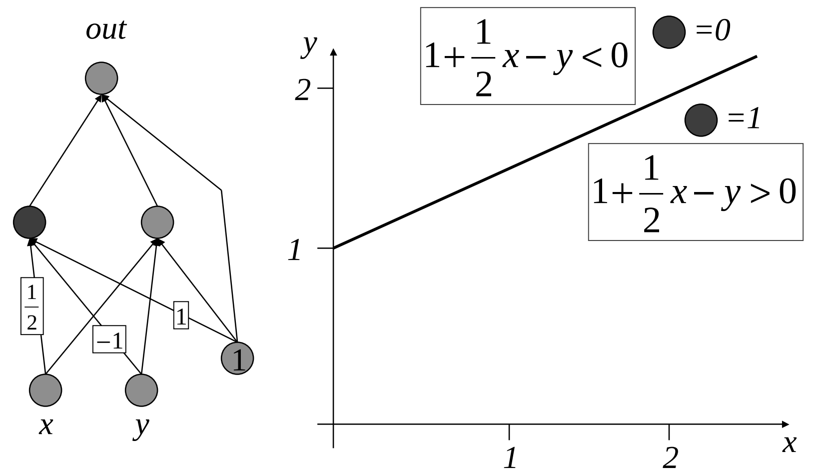
(non-linear "squashing" function)

Example: Perceptrons as Constraint Satisfaction Networks

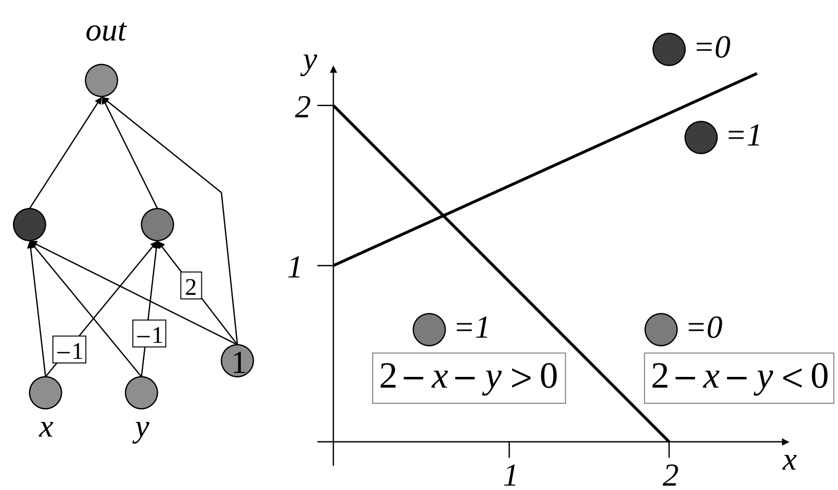




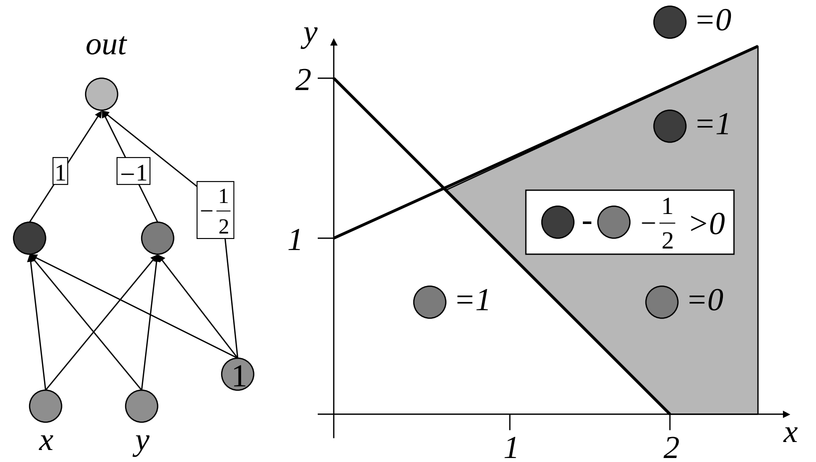
Example: Perceptrons as Constraint Satisfaction Networks



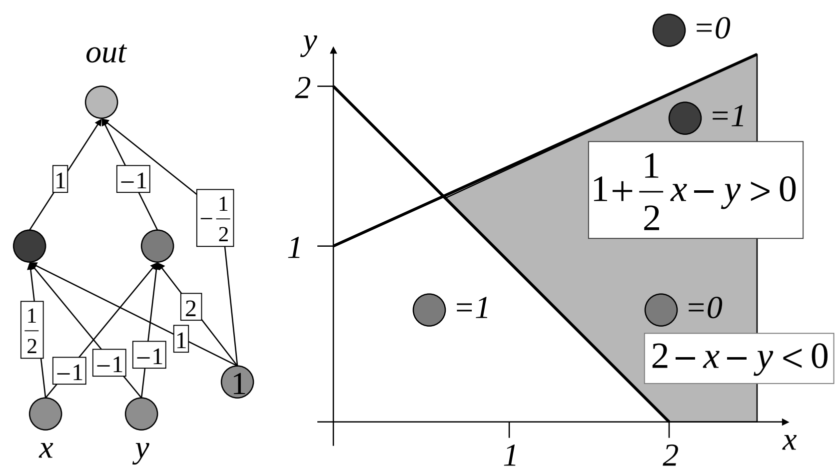
Example: Perceptrons as Constraint Satisfaction Networks



Example: Perceptrons as Constraint Satisfaction Networks

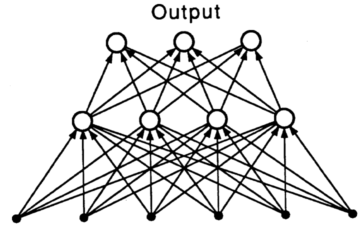


Perceptrons as Constraint Satisfaction Networks



Learning networks

- We want networks that can adapt themselves
  - Given input data, *minimize errors* between network's output and actual output by changing weights (supervised learning)
  - Can generalize from learned data to predict new outputs



Can this network adapt its weights to solve a problem?

How do we train it?

Gradient-descent learning (a la Hill-climbing)

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

$$E = \frac{1}{2}\sum_i \sum_u \left[Y_i^u - \sum_j w_{ij}\xi_j \right]^2$$

Then $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_u \left[Y_i^u - \sum_j w_{ij}\xi_j \right] \xi_j$

Cost function measures the network's squared errors 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 = Widrow-Hoff rule = LMS rule

Learning via Backpropagation of Errors

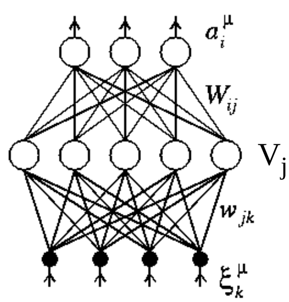
- Backpropagation is just gradient-descent learning for <u>multilayer</u> feedforward networks
- Use a *nonlinear*, differentiable activation function
  - Such as a sigmoid:

$$f \equiv \frac{1}{1+\exp(-2\eta h)} \qquad \text{where } h \equiv \sum_j w_{ij}\xi_j$$

- <u>Result</u>: Can propagate credit/blame back to internal nodes
  - Change in weights for output layer is similar to Delta rule
  - Chain rule (calculus) gives Δw_{ij} for internal "hidden" nodes

Backpropagation

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 : $\Delta W_{ij}(t+1) = -\eta \dfrac{\partial E}{\partial W_{ij}}$

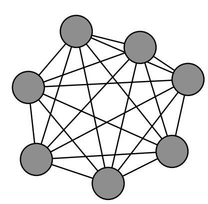
Error measure : $E = \frac{1}{2}\sum_{i\,\mu}(d_i^\mu - a_i^\mu)^2$

Backpropagation (for Math lovers' eyes only!)

- 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 <u>hidden-output connection weights</u>:
  - $\Delta W_{ij} = -\eta \partial E/\partial W_{ij} = \eta \Sigma_\mu [d_i - a_i]\, g'(A_i)\, V_j$
    $= \eta \Sigma_\mu \delta_i V_j$
- Learning rule for <u>input-hidden connection weights</u>:
  - $\Delta w_{jk} = -\eta\, \partial E/\partial w_{jk} = -\eta\, (\partial E/\partial V_j)\, (\partial V_j/\partial w_{jk})$ {*chain rule*}
    $= \eta \Sigma_{\mu,\iota} ([d_i - a_i]\, g'(A_i)\, W_{ij})\, (g'(A_j)\, \xi_k)$
    $= \eta \Sigma_\mu \delta_j \xi_k$

Hopfield networks (example of recurrent nets)

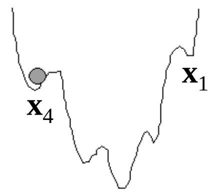
- Act as "autoassociative" memories to store patterns
  - McCulloch-Pitts neurons with outputs -1 or 1, and threshold Θ
  - All neurons connected to each other
    - Symmetric weights ($w_{ij} = w_{ji}$) and $w_{ii} = 0$
  - Asynchronous updating of outputs
    - Let s_i be the state of unit i
    -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



completely connected

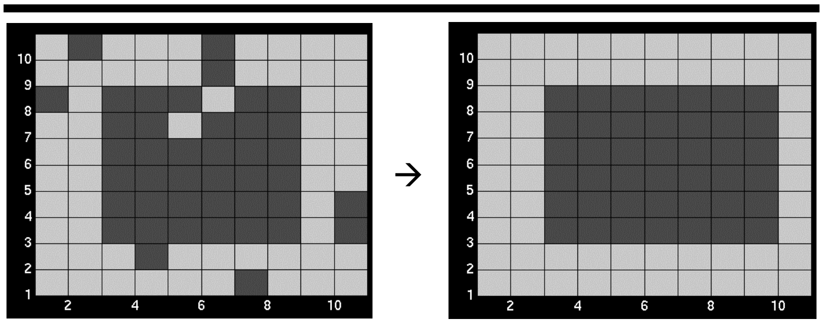
Hopfield networks

- Network converges to cost function's local minima which store different patterns

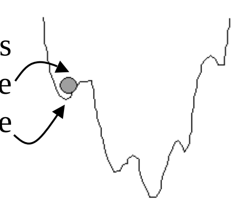


- Store p N-dimensional pattern vectors $\mathbf{x}_1, \ldots, \mathbf{x}_p$ using a "Hebbian" learning rule:
  - $w_{ji} = 1/N \, \Sigma_{m=1,\ldots,p} \; x_{m,j} \, x_{m,i}$ for all $j \neq i$; 0 for $j = i$
  - $W = 1/N \, \Sigma_{m=1,\ldots,p} \, \mathbf{x}_m \, \mathbf{x}_m^T$ (outer product of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

Pattern Completion in a Hopfield Network



Network converges
from here
to here



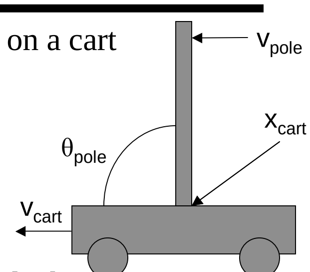
Local minimum ("attractor") of cost (or "energy") function stores pattern

Recent Trends and Applications of Neural Networks

- **Recent Trends**
  - Probabilistic approach: NNs as Bayesian networks (allows principled derivation of dynamics, learning rules, and even network structure)
  - Not-so-artificial networks: Make network more biologically realistic
  - NNs in Hardware: Ultra-fast implementation of large learning networks in silicon
- **Applications**
  - Text to speech generation (NETtalk by Sejnowski & Rosenberg)
  - Handwritten character recognition (zip codes on envelopes)
  - Autonomous driving (ALVINN at CMU – uses backprop network)
  - Control of other physical systems
    - Demos! (by Keith Grochow, as part of CSE 599, 2001)

Demos

- Neural Network learns to balance a pole on a cart
  - System:
    - 4 state variables: x_{cart}, v_{cart}, θ_{pole}, v_{pole}
    - 1 input: Force on cart
  - Backprop Network:
    - Input: State variables
    - Output: New force on cart



- NN learns to back a truck into a loading dock
  - System (Nyugen and Widrow, 1989):
    - State variables: x_{cab}, y_{cab}, θ_{cab}
    - 1 input: new $\theta_{steering}$
  - Backprop Network:
    - Input: State variables
    - Output: Force on cart