

CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu

TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)

William Yeoh (wyeoh@usc.edu)

Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN

Office hours: By appointment

Class page: <http://www-rcf.usc.edu/~macskass/CS561-Spring2010/>

This class will use <http://www.uscden.net/> and class webpage

- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:

[AIMA] Artificial Intelligence: A Modern Approach,
by Stuart Russell and Peter Norvig. (2nd ed)

Distance Education Network

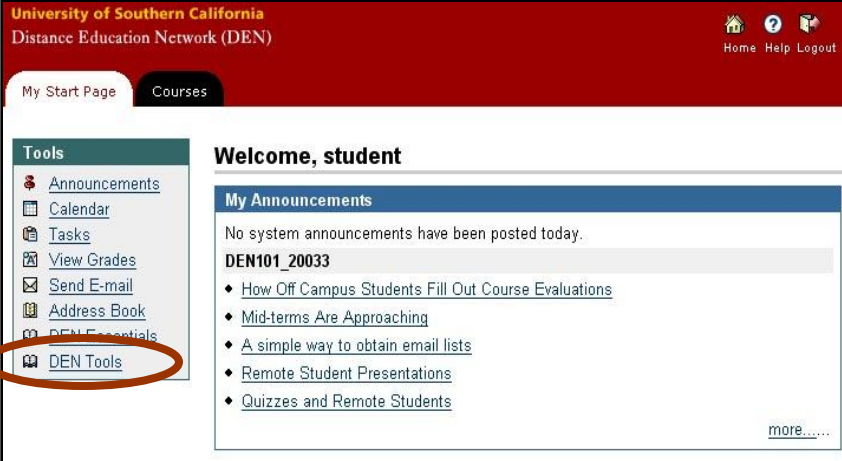
Online Course Evaluation Instructions for DEN Students

- On campus students need to fill out the paper evaluations in class
- Course evaluation period is Monday, April 12th - Friday, April 30th.

Follow these steps:

1. Log-in to the DEN portal website <http://mapp.usc.edu>.
2. From “My Start Page” click on the DEN Tools link.
3. Once the DEN Tools window pops-up, click on Course Evaluation.
4. Complete the evaluation and hit submit.

- DEN students should NOT fill out a paper evaluation!
- DEN online evaluations are completely anonymous.
- Reminder e-mails will be sent out.



The screenshot shows the DEN portal interface. At the top, it says "University of Southern California Distance Education Network (DEN)". There are navigation tabs for "My Start Page" and "Courses". A "Tools" menu is visible on the left, with "DEN Tools" circled in red. The main content area is titled "Welcome, student" and shows "My Announcements" for course "DEN101_20033". The announcements include: "No system announcements have been posted today.", "How Off Campus Students Fill Out Course Evaluations", "Mid-terms Are Approaching", "A simple way to obtain email lists", "Remote Student Presentations", and "Quizzes and Remote Students".



Statistical Learning [AIMA Ch 20]

- Part I:
 - Bayesian learning
 - Maximum *a posteriori* and maximum likelihood learning
 - Bayes net learning
 - ML parameter learning with complete data
 - linear regression
- Part II:
 - Neural Networks
 - Brains
 - Perceptrons
 - Multi-layer perceptrons
 - Applications of neural networks

Part I: Full Bayesian learning

- View learning as Bayesian updating of a probability distribution over the **hypothesis space**
- H is the hypothesis variable, values h_1, h_2, \dots , prior $P(H)$
- j^{th} observation d_j gives the outcome of random variable D_j
training data $\mathbf{d} = d_1, \dots, d_N$
- Given the data so far, each hypothesis has a posterior probability:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i)$$

where $P(\mathbf{d}|h_i)$ is called the **likelihood**

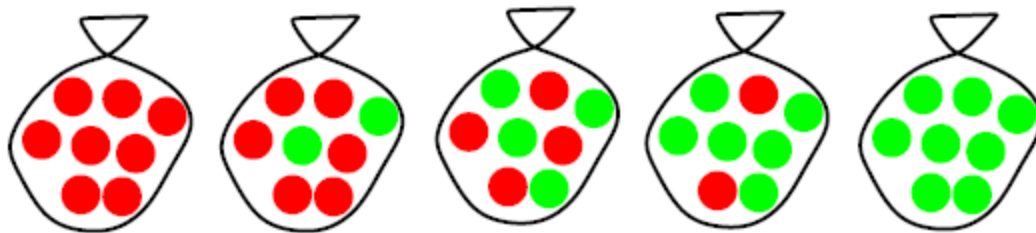
- Predictions use a likelihood-weighted average over the hypotheses:

$$P(X|\mathbf{d}) = \sum_i P(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i P(X|h_i)P(h_i|\mathbf{d})$$

- No need to pick one best-guess hypothesis!

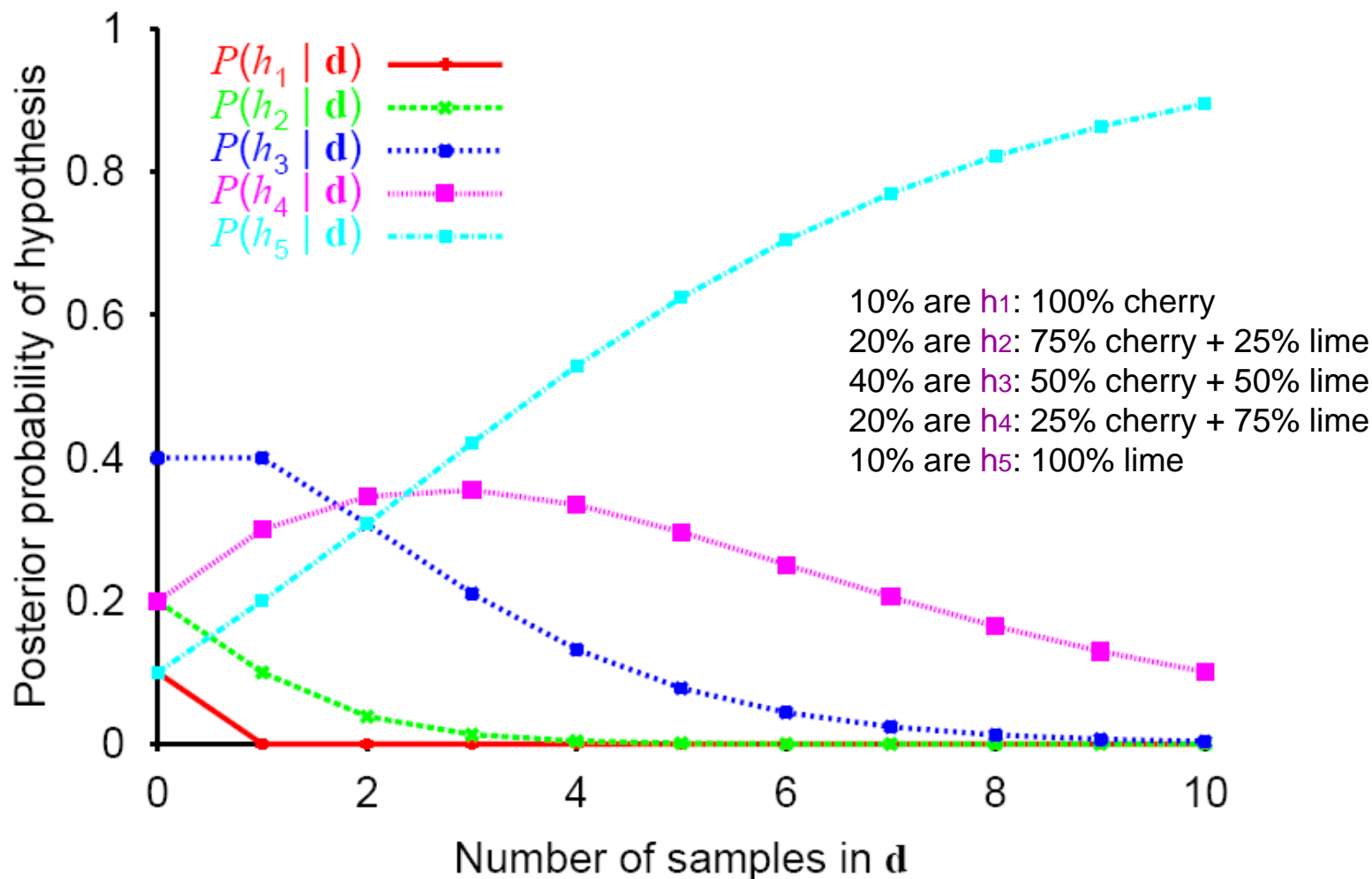
Example

- Suppose there are five kinds of bags of candies:
 - 10% are h_1 : 100% cherry candies
 - 20% are h_2 : 75% cherry candies + 25% lime candies
 - 40% are h_3 : 50% cherry candies + 50% lime candies
 - 20% are h_4 : 25% cherry candies + 75% lime candies
 - 10% are h_5 : 100% lime candies

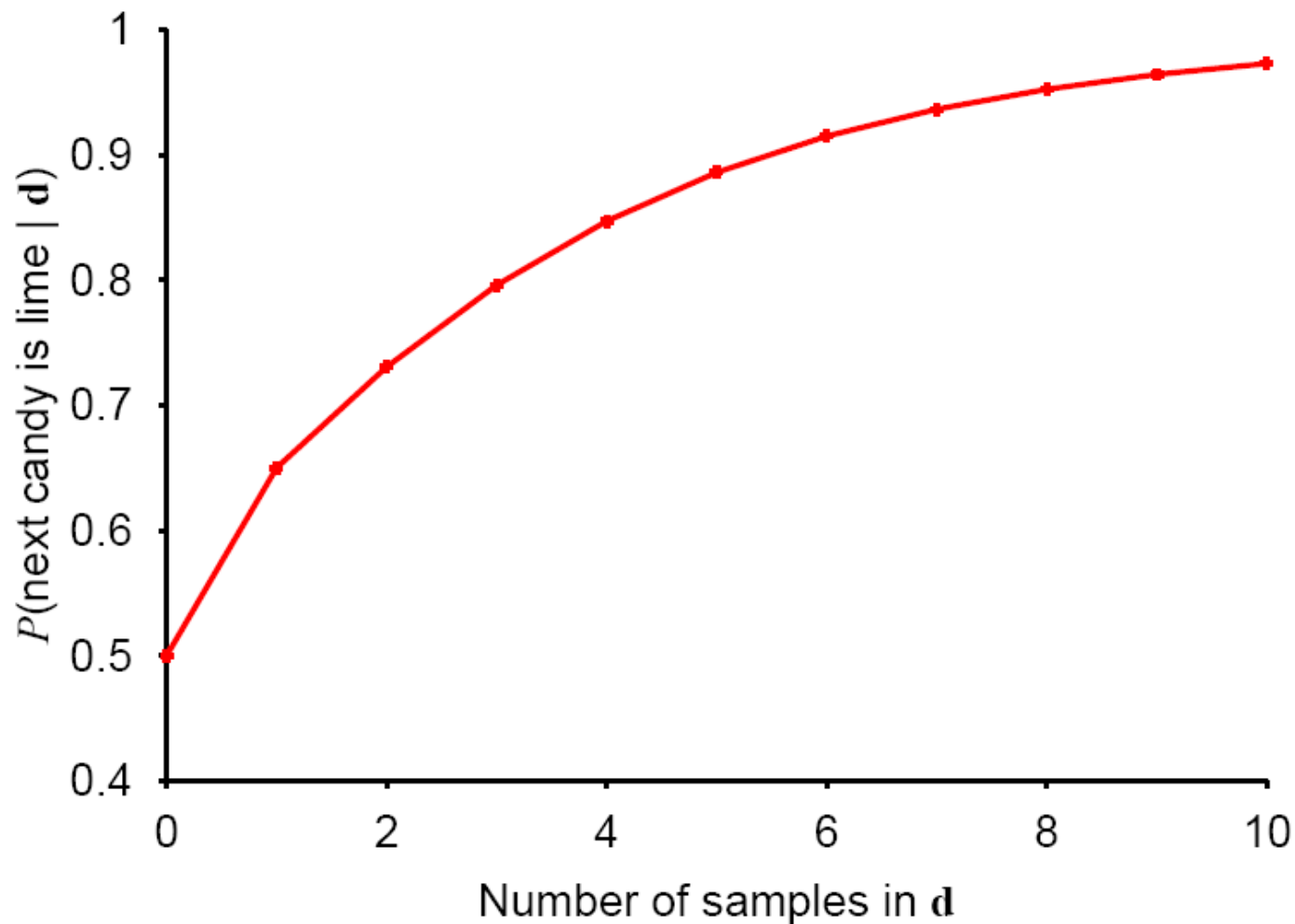


- Then we observe candies drawn from some bag: ● ● ● ● ● ● ● ● ● ●
- What kind of bag is it? What flavor will the next candy be?

Posterior probability of hypotheses



Prediction probabilities



MAP approximation

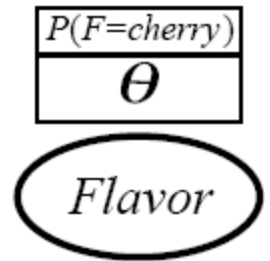
- Summing over the hypothesis space is often intractable (e.g., 18,446,744,073,709,551,616 Boolean functions of 6 attributes)
- Maximum a posteriori (MAP) learning: choose h_{MAP} maximizing $P(h_i|d)$
i.e., maximize $P(d|h_i)P(h_i)$ or $\log P(d|h_i) + \log P(h_i)$
- Log terms can be viewed as (negative of)
bits to encode data given hypothesis + bits to encode hypothesis
- This is the basic idea of minimum description length (MDL) learning
- For deterministic hypotheses, $P(d|h_i)$ is 1 if consistent, 0 otherwise
 \Rightarrow MAP = simplest consistent hypothesis (cf. science)

ML approximation

- For large data sets, prior becomes irrelevant
- **Maximum likelihood** (ML) learning: choose h_{ML} maximizing $P(d|h_i)$
- i.e., simply get the best fit to the data; identical to MAP for uniform prior (which is reasonable if all hypotheses are of the same complexity)
- ML is the “standard” (non-Bayesian) statistical learning method

ML parameter learning in Bayes nets

- Bag from a new manufacturer; fraction θ of cherry candies?
- Any θ is possible: continuum of hypotheses h_θ
- θ is a **parameter** for this simple (**binomial**) family of models



- Suppose we unwrap N candies, c cherries and $\ell=N-c$ limes
- These are **i.i.d.** (independent, identically distributed) observations, so

$$P(d | h_\theta) = \prod_{j=1}^N P(d_j | h_\theta) = \theta^c \cdot (1-\theta)^\ell$$

- Maximize this w.r.t. θ – which is easier for the **log-likelihood**:

$$L(d | h_\theta) = \log P(d | h_\theta) = \sum_{j=1}^N \log(d_j | h_\theta) = c \log \theta + \ell \log(1-\theta)$$

$$\frac{dL(d | h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

- Seems sensible, but causes problems with 0 counts!

Multiple parameters

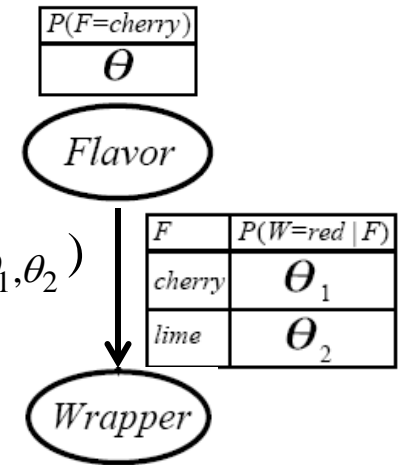
- Red/green wrapper depends probabilistically on flavor:
- Likelihood for, e.g., cherry candy in green wrapper:

$$\begin{aligned} P(F = \text{cherry}, W = \text{green} | h_{\theta, \theta_1, \theta_2}) \\ &= P(F = \text{cherry} | h_{\theta, \theta_1, \theta_2}) P(W = \text{green} | F = \text{cherry}, h_{\theta, \theta_1, \theta_2}) \\ &= \theta \cdot (1 - \theta_1) \end{aligned}$$

- N candies, r_c red-wrapped cherry candies, etc.:

$$P(d | h_{\theta, \theta_1, \theta_2}) = \theta^c (1 - \theta)^\ell \cdot \theta_1^{r_c} (1 - \theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1 - \theta_2)^{g_\ell}$$

$$\begin{aligned} L = & [c \cdot \log \theta + \ell \log(1 - \theta)] \\ & + [r_c \cdot \log \theta_1 + g_c \log(1 - \theta_1)] \\ & + [r_\ell \cdot \log \theta_2 + g_\ell \log(1 - \theta_2)] \end{aligned}$$



Multiple parameters cont'd

Derivatives of L contain only the relevant parameter:

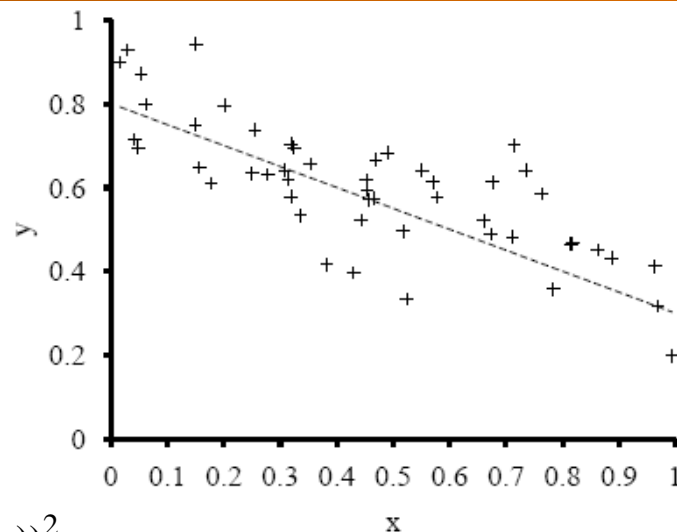
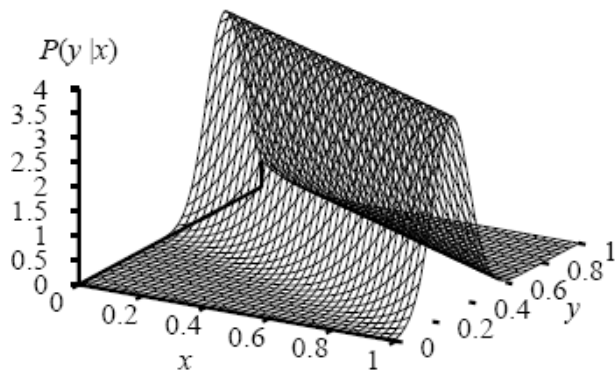
$$\frac{dL}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell}$$

$$\frac{dL}{d\theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c + g_c}$$

$$\frac{dL}{d\theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell + g_\ell}$$

With **complete data**, parameters can be learned separately

Example: linear Gaussian model



- Maximizing $P(y | x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-(\theta_1x+\theta_2))^2}{2\sigma^2}}$ w.r.t. θ_1, θ_2

$$= \text{minimizing } E = \sum_{j=1}^N (y_j - (\theta_1 x_j + \theta_2))^2$$

That is, minimizing the sum of squared errors gives the ML solution for a linear fit **assuming Gaussian noise of fixed variance**

Summary: Bayesian learning

Full Bayesian learning gives best possible predictions but is intractable

MAP learning balances complexity with accuracy on training data

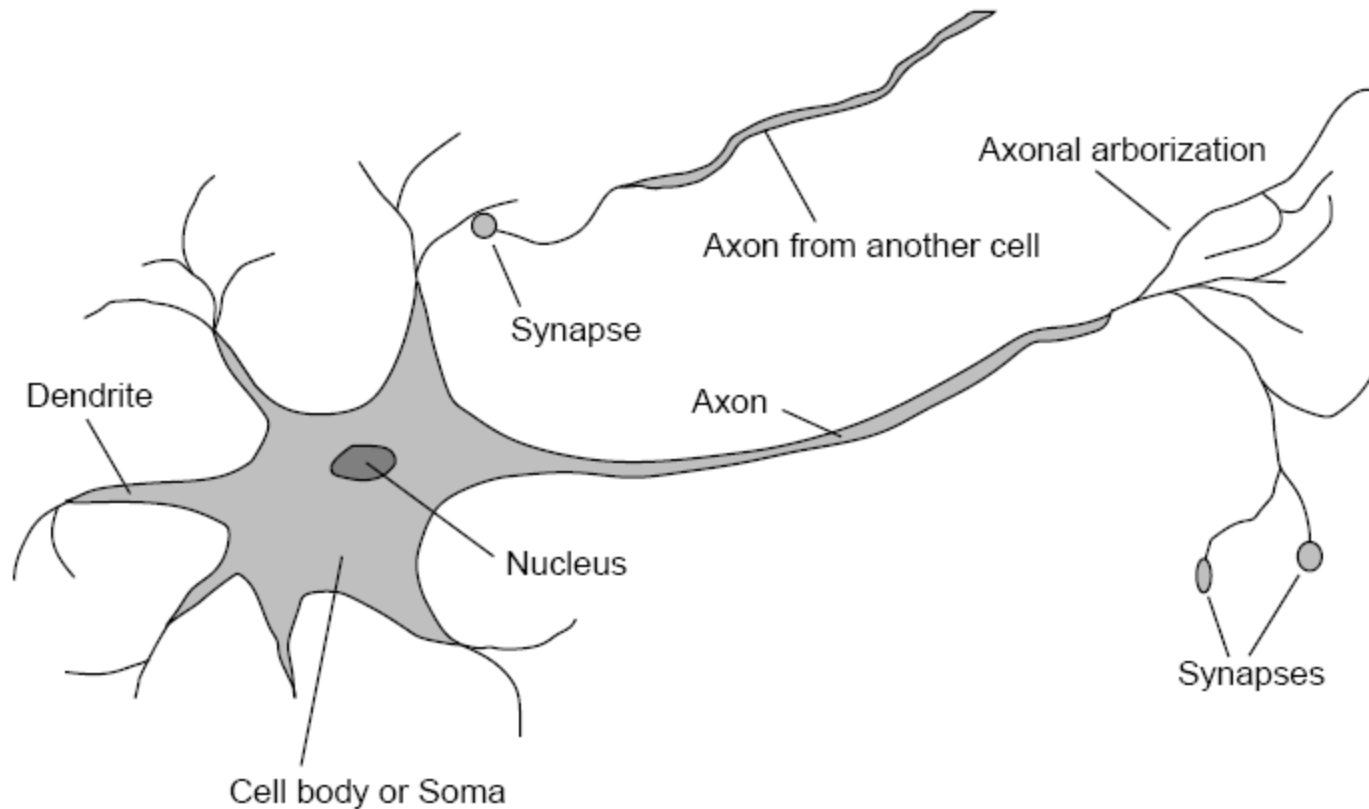
Maximum likelihood assumes uniform prior, OK for large data sets

1. Choose a parameterized family of models to describe the data
requires substantial insight and sometimes new models
2. Write down the likelihood of the data as a function of the parameters
may require summing over hidden variables, i.e., inference
3. Write down the derivative of the log likelihood w.r.t. each parameter
4. Find the parameter values such that the derivatives are zero
may be hard/impossible; modern optimization techniques help

Part II: Neural Networks

Brains

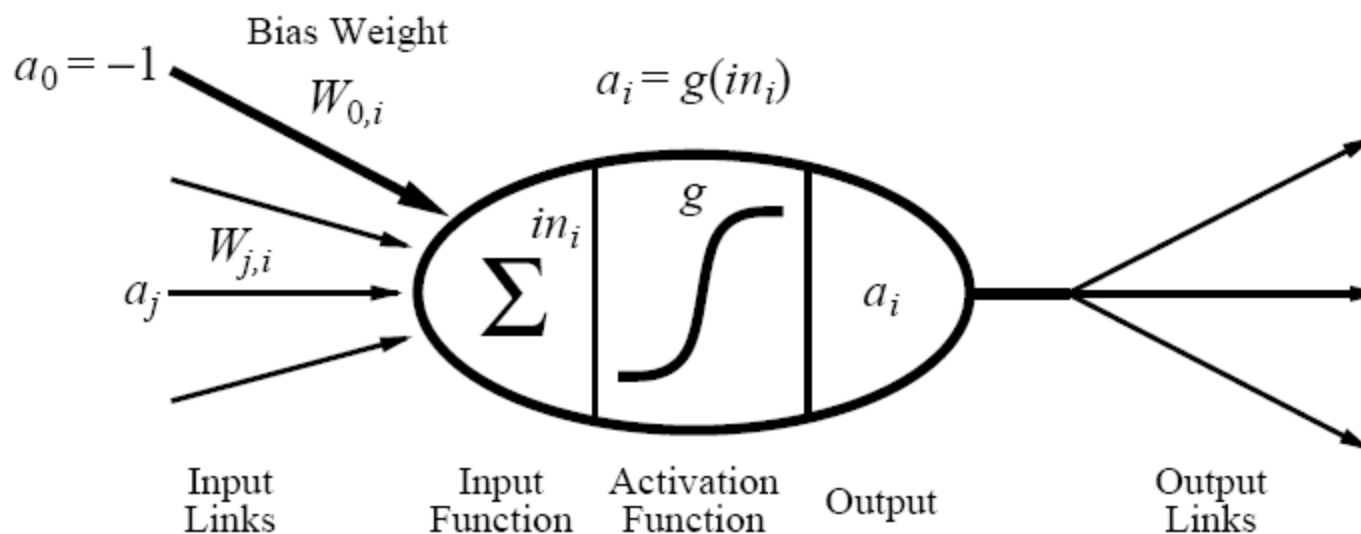
- 10^{11} neurons of > 20 types, 10^{14} synapses, 1ms-10ms cycle time
- Signals are noisy “spike trains” of electrical potential



McCulloch-Pitts "unit"

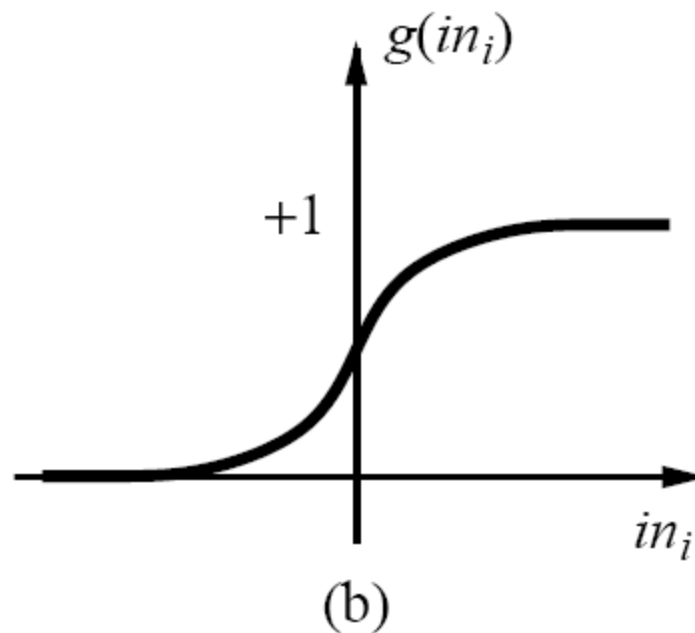
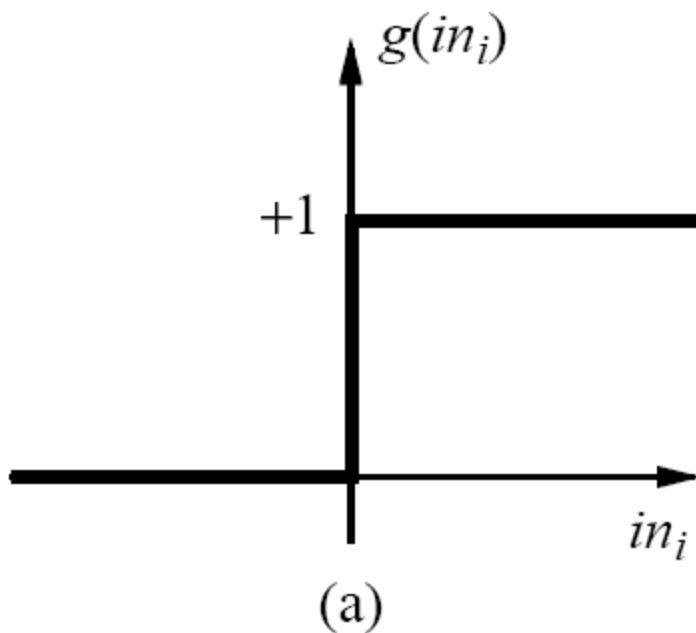
Output is a "squashed" linear function of the inputs:

$$a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_j)$$



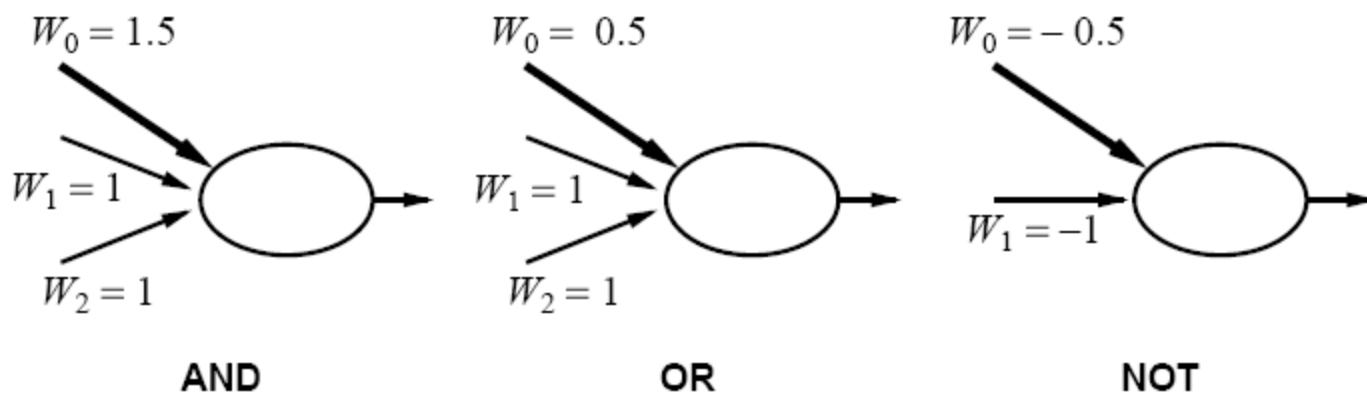
A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions



- (a) is a **step function** or **threshold function**
- (b) is a **sigmoid function** $1/(1 + e^{-x})$
- Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

Network structures

Feed-forward networks:

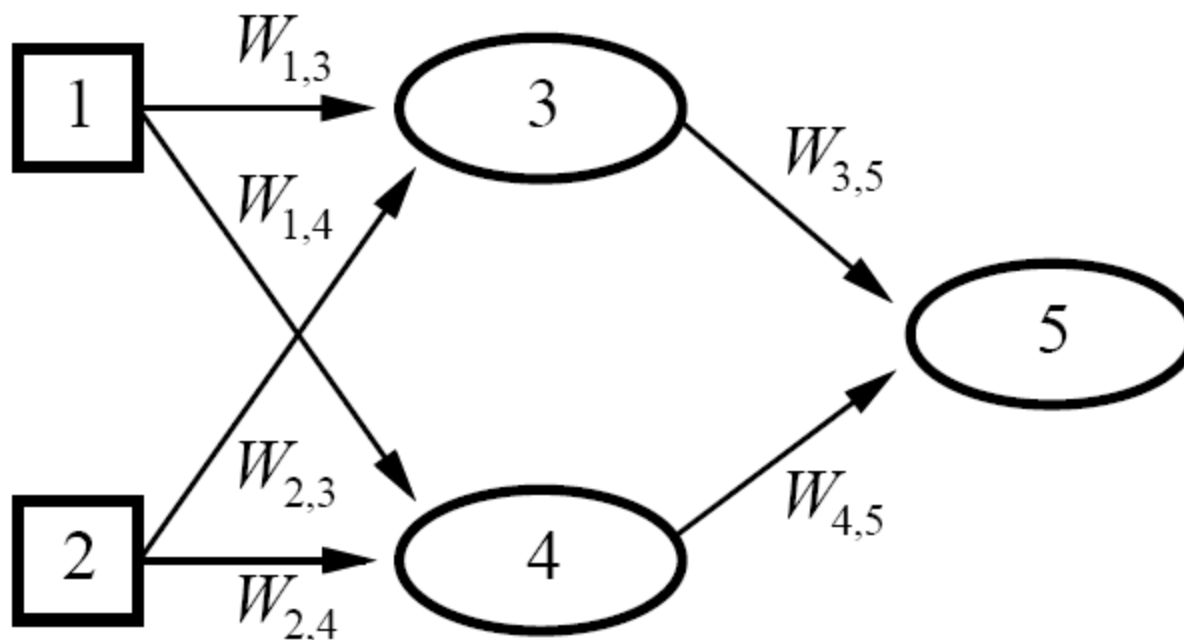
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
 $g(x) = \text{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
 \approx MCMC in Bayes nets
- recurrent neural nets have directed cycles with delays
 \Rightarrow have internal state (like flip-flops), can oscillate etc.

Feed-forward example

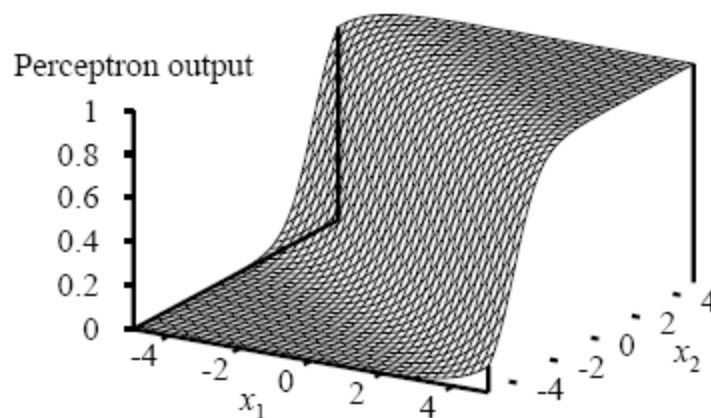
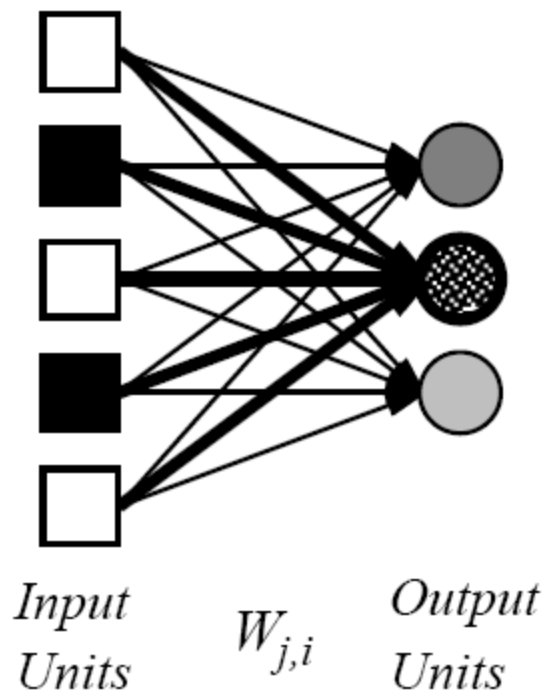


Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Single-layer perceptrons



Output units all operate separately—no shared weights
Adjusting weights moves the location, orientation, and steepness of cli

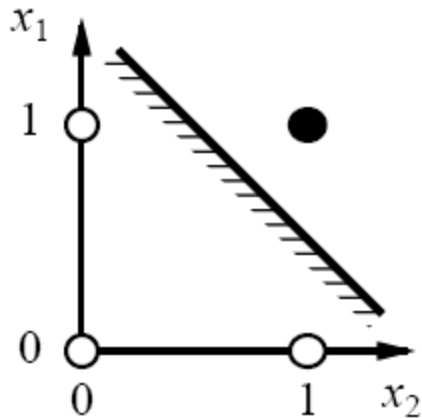
Expressiveness of perceptrons

Consider a perceptron with $g = \text{step function}$ (Rosenblatt, 1957, 1960)

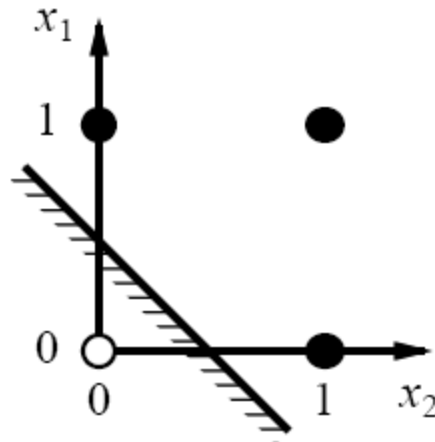
Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a **linear separator** in input space:

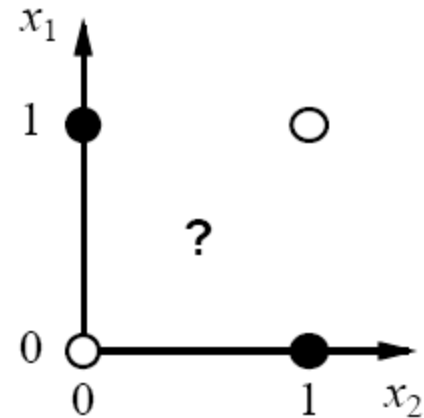
$$\sum_j W_j x_j > 0 \text{ or } \mathbf{W} \cdot \mathbf{x} > 0$$



(a) x_1 and x_2



(b) x_1 or x_2



(c) x_1 xor x_2

Minsky & Papert (1969) pricked the neural network balloon

Perceptron learning

Learn by adjusting weights to reduce **error** on training set
The **squared error** for an example with input \mathbf{x} and true output y is

$$E = \frac{1}{2} \text{Err}^2 \equiv \frac{1}{2} (y - h\mathbf{W}(x))^2$$

Perform optimization search by gradient descent:

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

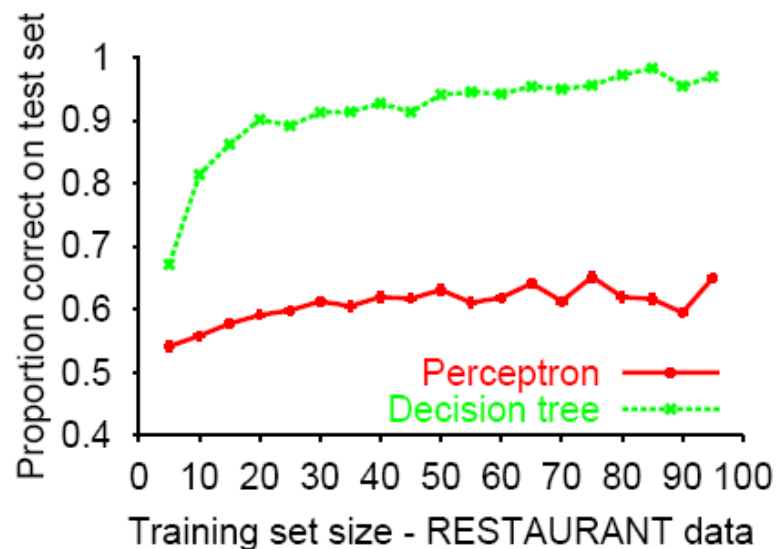
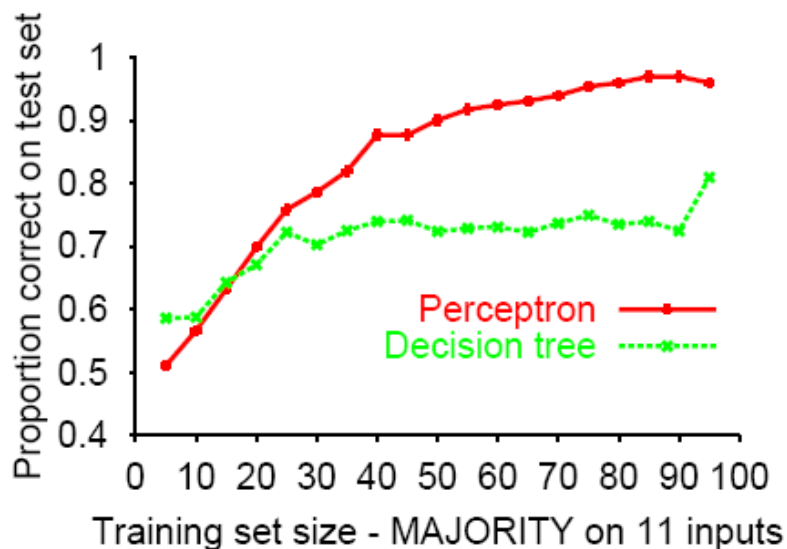
Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times \text{Err} \times g'(in) \times x_j$$

E.g., +ve error \Rightarrow increase network output
 \Rightarrow increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning cont'd

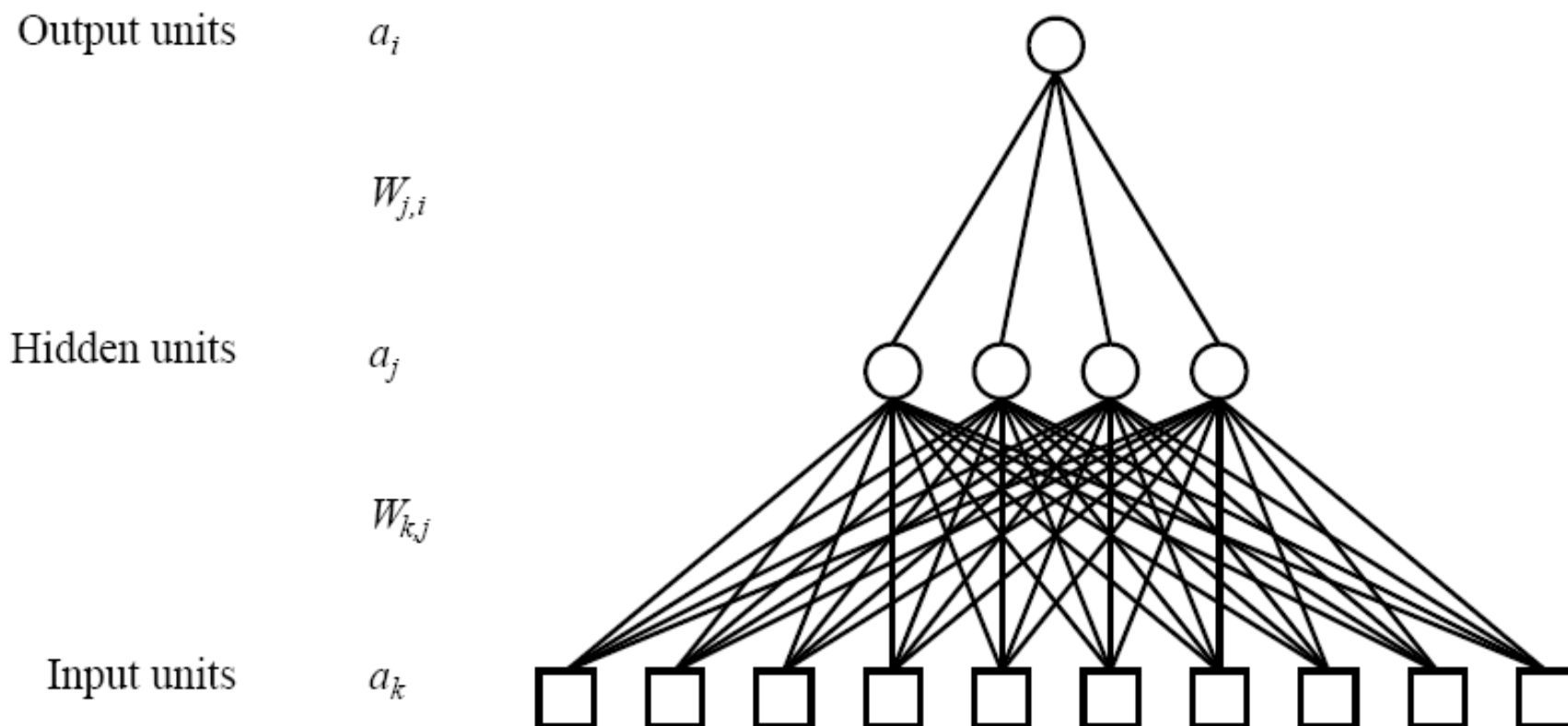
- Perceptron learning rule converges to a consistent function for any linearly separable data set



- Perceptron learns majority function easily, DTL is hopeless
- DTL learns restaurant function easily, perceptron cannot represent it

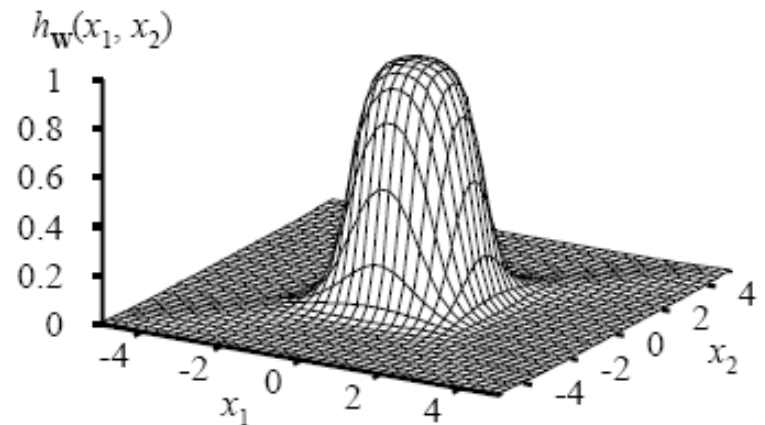
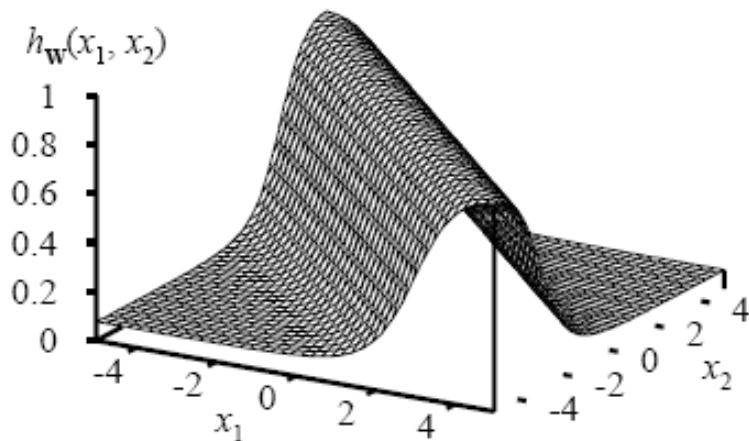
Multi-layer perceptrons

- Layers are usually fully connected;
- numbers of **hidden units** typically chosen by hand



Expressiveness of MLPs

- All continuous functions w/ 2 layers, all functions w/ 3 layers



- Combine two opposite-facing threshold functions to make a ridge
- Combine two perpendicular ridges to make a bump
- Add bumps of various sizes and locations to any surface
- Proof requires exponentially many hidden units (cf DTL proof)

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

Where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

Update rule for weights i in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

where the sum is over the nodes in the output layer.

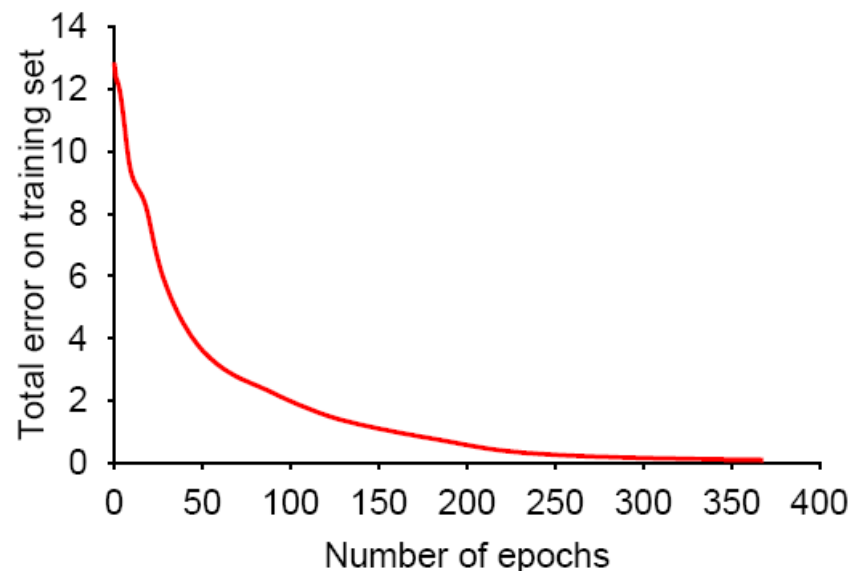
$$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left(\sum_i W_{j,i} a_j \right) \\ &= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i \end{aligned}$$

Back-propagation derivation cont'd

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left(\sum_j W_{j,i} a_j \right) \\ &= -\sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right) \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j\end{aligned}$$

Back-propagation learning cont'd

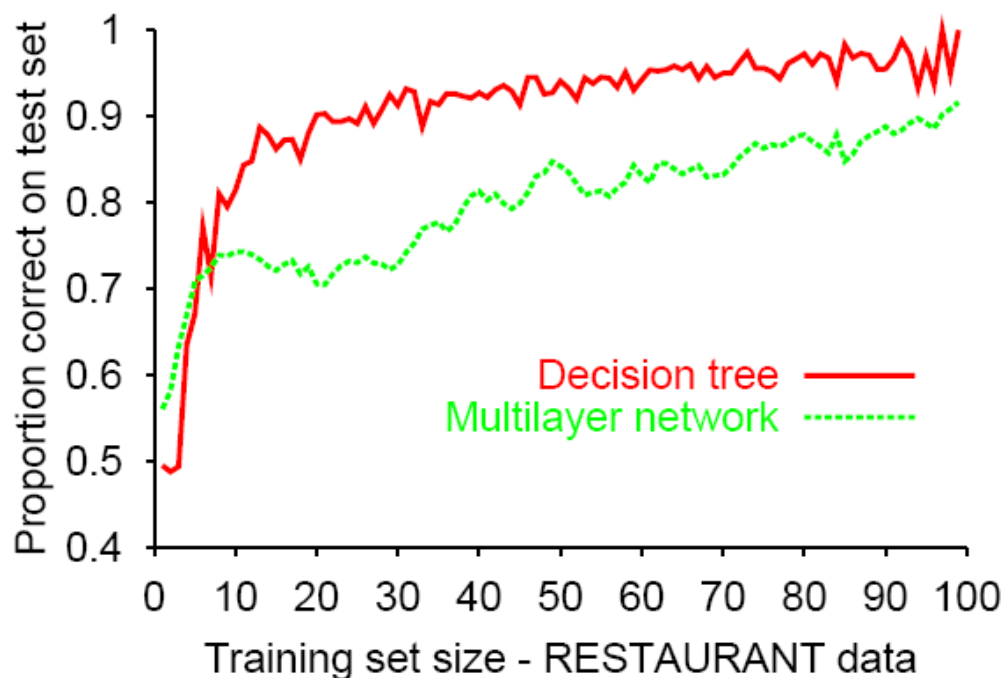
- At each **epoch**, sum gradient updates for all examples and apply
- **Training curve** for 100 restaurant examples: finds exact fit



- Typical problems: slow convergence, local minima

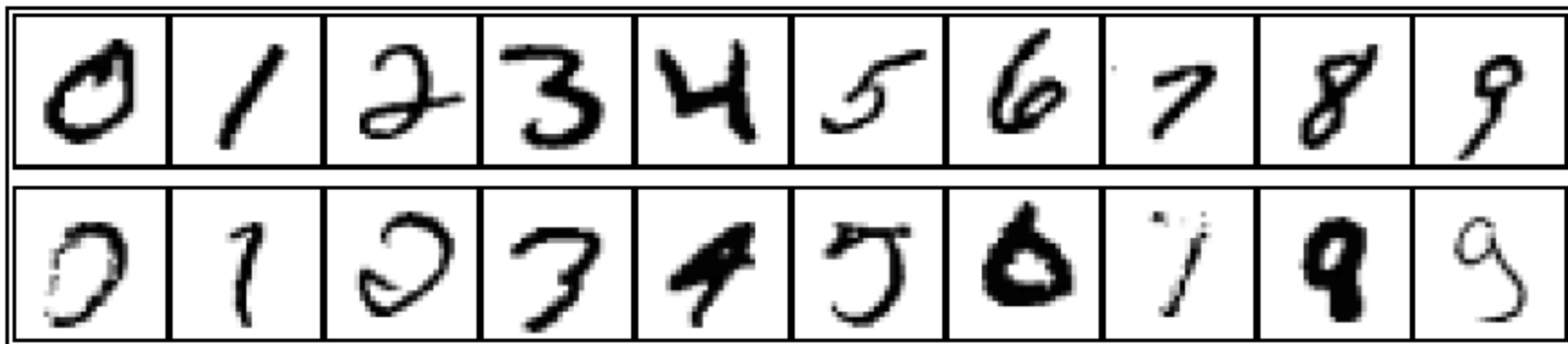
Back-propagation learning cont'd

- Learning curve for MLP with 4 hidden units:



- MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily

Handwritten digit recognition



- 3-nearest-neighbor = 2.4% error
- 400-300-10 unit MLP = 1.6% error
- LeNet: 768-192-30-10 unit MLP = 0.9% error
- Current best (kernel machines, vision algorithms) \approx 0.6% error

Summary

- Most brains have lots of neurons; each neuron linear-threshold unit (?)
- Perceptrons (one-layer networks) insufficiently expressive
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged