Today's Lecture

- Tangential questions (warm up)
- Administrative Details
- Learning
  - Decision trees: cleanup & details
  - Sub-symbolic learning
    - Neural networks

- Why do I use an Apple Macintosh?
  - Who cares
  - It's elegant
  - I respect Apple's innovation.

- On learning...

Administrativia

- Please signup AGAIN using the web page.
  - There was a bug in the CGI/HTML connection.
- Note that the normal late policy will not apply to the project.
  - You **__must__** submit the electronic (executable) on time, or it may not be evaluated!
  - It must run on LINUX. Be certain to compile and test it on one of the linux machines in the lab well before it's due.
    - If you are developing on another platform, regularly test on linux during development.

ID3

Considered information implicit in a query about a set of examples.

- This provides the **total** amount of information implicit in a decision tree.
- Each question along the tree provides some fraction of this total information.
  - How much ??

- Consider **information gain** per attribute *A*.
  - Gain(Q:X) = E(Q:X) - E(A:X)
  - Info needed to complete tree is weighted sum of the subtrees

$$E(P) = \sum_{i=1}^{v} |E_i|/|X| \; I(E_i)$$

Has been used with success for diverse problems including chess endgames & loans [Quinlan 1983, 1986: Michalski *et al.* book, Machine Learning; Machine Learning journal]

Is there a fixed circuit network topology that can be used to represent
a family of functions?

Yes! Neural-like networks (a.k.a. artificial neural networks) allow us this flexibility and more; we can represent arbitrary families of continuous functions using fixed topology networks.

Neural Networks?

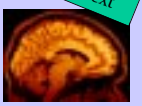
Artificial Neural Nets
a.k.a.
Connectionist Nets
(connectionist learning)
a.k.a.
Sub-symbolic learning
a.k.a.
Perceptron learning (a special case)

The idealized neuron

- Artificial neural networks come in several "flavors".
  - Most of based on a simplified model of a neuron.
- A set of (many) inputs.
- One output.
- Output is a function of the sum on the inputs.
  - Typical functions:
    - Weighted sum
    - Threshold
    - Gaussian

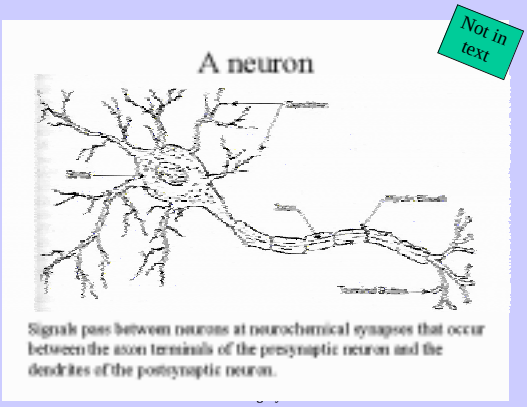
Why neural nets?

Not in text



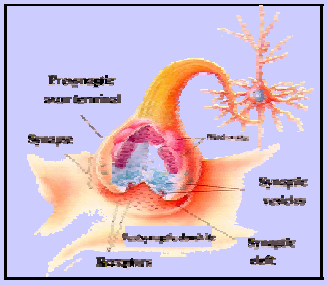
- Motives:
  - We wish to create systems with abilities akin to those of the human mind.
    - The mind is usually assumed to be be a direct consequence of the structure of the brain.
      - Let's mimic the structure of the brain!
  - By using simple computing elements, we obtain a system that might scale up easily to parallel hardware.
  - Avoids (or solves?) the **key unresolved problem** of how to get from "signal domain" to symbolic representations.
  - Fault tolerance

Not in text



Real and fake neurons

- Signals in neurons are coded by "spike rate".
- In ANN's, inputs can be either:
  - 0 or 1 (binary)
  - [0,1]
  - [-1,1]
  - R (real)
- Each input I_i has an associated real-valued weight w_i
- Learning by changing weights at synapses.



CS-424 Gregory Dudek

Not in text

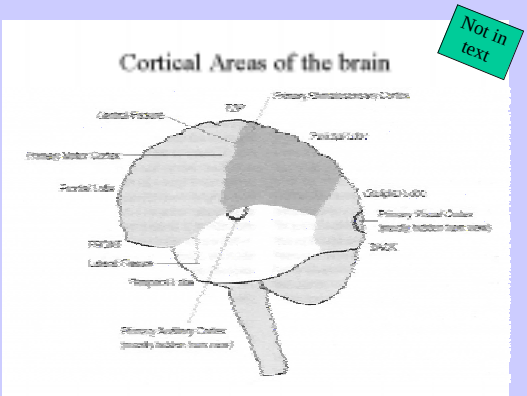
Structure and function of the physical brain

- Over 100 billion neurons, constantly active
- Ramon y Cajal (1852-1934)
  - first to show the neuron as the smallest unit of the nervous system
- Estimated 60 trillion synapses in human cortex (Shepard & Koch, 1990)
- Multiple, overlapping neural networks, each capable of learning

CS-424 Gregory Dudek

Not in text

Cortical Areas of the brain



Inductive bias?

Not in text

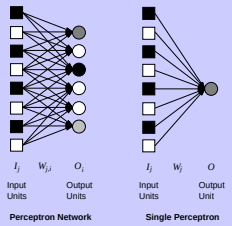
- Where's the inductive bias?
  - In the topology and architecture of the network.
  - In the learning rules.
  - In the input and output representation.
  - In the initial weights.

Simple neural models

Not in text

- Oldest ANN model is McCulloch-Pitts neuron [1943] .
  - Inputs are +1 or -1 with real-valued weights.
  - If sum of weighted inputs is > 0, then the neuron "fires" and gives +1 as an output.
  - Showed you can comput logical functions.
  - Relation to learning proposed (later!) by Donald Hebb [1949].
- Perceptron model [Rosenblatt, 1958].
  - Single-layer network with same kind of neuron.
    - Firing when input is about a threshold: $\sum x_i w_i > t$.
  - Added a **learning rule** to allow weight selection.

Perceptron nets



Perceptron learning

- Perceptron learning:
  - Have a set of training examples (TS) encoded as input values (I.e. in the form of binary vectors)
  - Have a set of desired output values associated with these inputs.
    - This is **supervised learning**.
  - Problem: how to adjust the **weights** to make the actual outputs match the training examples.
    - NOTE: we to not allow the topology to change! [You should be thinking of a question here.]
- Intuition: when a perceptron makes a mistake, it's weights are wrong.
  - Modify them so make the output bigger or smaller, as desired.

Learning algorithm

- Desired T_i Actual output O_i
- Weight update formula (weight from unit j to i):

$$W_{j,i} = W_{j,I} + k^* x_j * (T_i - O_i)$$

Where k is the **learning rate.**

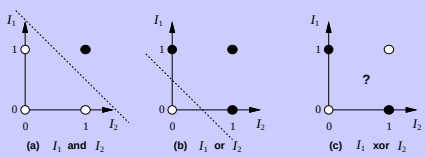
- **If the examples can be learned** (encoded), then the **perceptron learning rule** will find the weights.
  - How?

Gradient descent.

Key thing to prove is the absence of local minima.

Perceptrons: what can they learn?

- Only **linearly separable** functions [Minsky & Papert 1969].



- N dimensions: N-dimensional hyperplane.

More general networks

- Generalize in 3 ways:
  - Allow continuous output values [0,1]
  - Allow multiple layers.
    - This is **key** to learning a larger class of functions.
  - Allow a more complicated function than thresholded summation
    [why??]

Generalize the learning rule to accommodate this: let's see how it works.

The threshold

- The key variant:
  - Change threshold into a **differentiable function**
  - **Sigmoid**, known as a "soft non-linearity" (silly).

$$M = \sum x_i w_i$$

$$O = 1 / (1 + e^{-k M})$$