Lecture 6: Artificial Neural Networks

- Overview
- Perceptron learning

The human brain

- Contains ~ 10^{11} neurons, each of which may have up to ~ 10^{4-5} input/output connections
- Each neuron is fairly slow, with a switching time of ~ 1 millisecond
- Yet the brain is very fast and reliable at computationally intensive tasks (e.g. vision, speech recognition, knowledge retrieval)
- Although computers are at least 1 million times faster in raw switching speed!
- The brain is also more fault-tolerant, and exhibits graceful degradation with damage
- Maybe this is due to its architecture, which ensures massive parallel computation!

Connectionist models

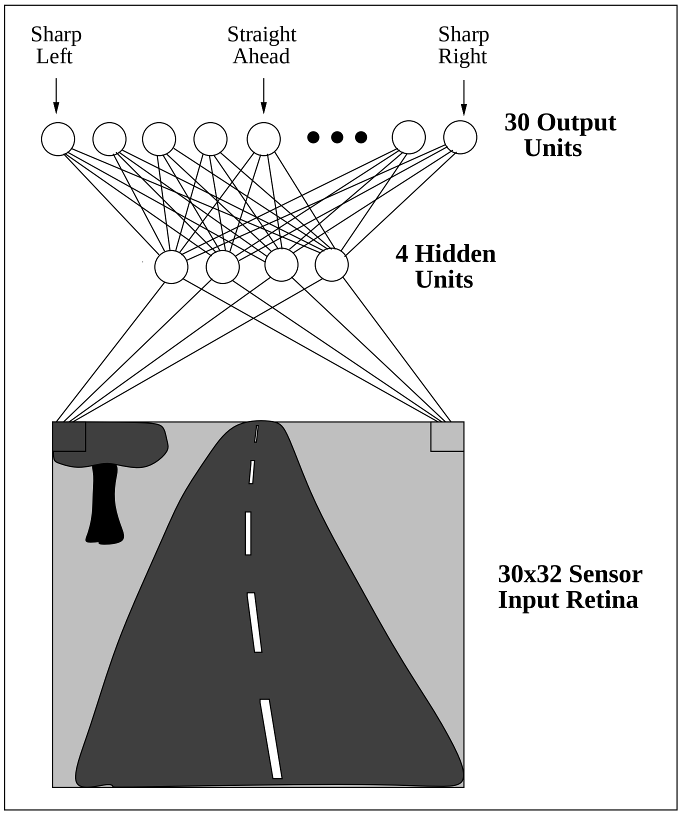
Based on the assumption that a computational architecture similar to the brain would duplicate (at least some of) its wonderful abilities.

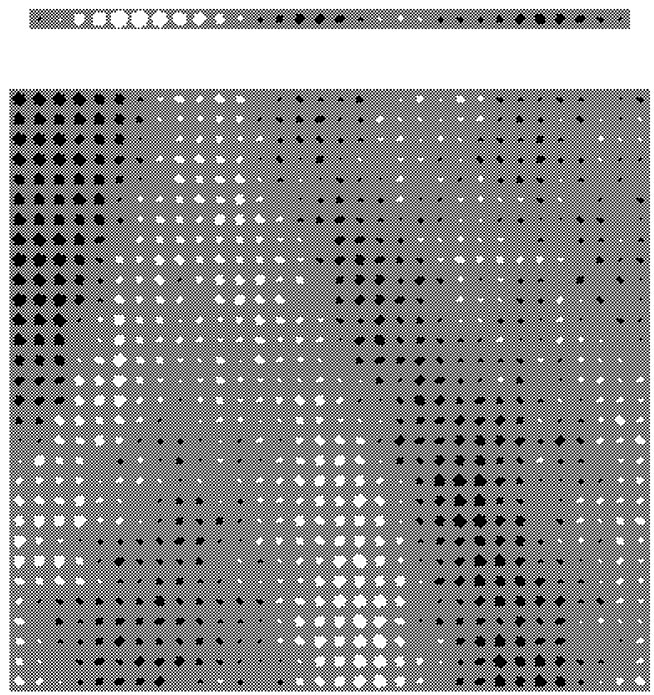
Properties of artificial neural nets (ANNs):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

MANY different kinds of architectures, motivated both by biology and mathematics/efficiency of computation

Example: ALVINN (Pomerleau, 1993)





What is a neural network?

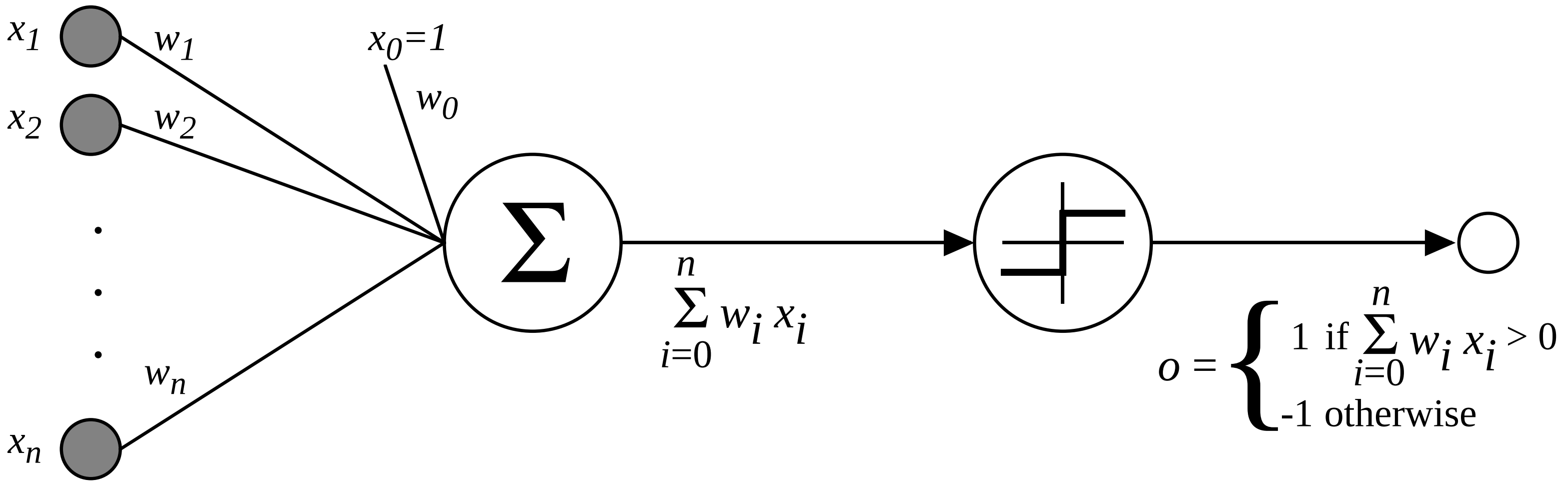
A graph of simple individual units (“neurons”)

- The edges of the graph are links on which the neurons can send data to each other

  The edges have **weights**, which multiply the data that is sent
- *Learning = choosing weight values for all edges in the graph*

  Sometimes learning means adding/deleting nodes
- In the vast majority of applications, the graph is acyclic and directed.

Perceptron



Sometimes we will add a fixed component $x_0 = 1$ to all the instances and use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron learning algorithm

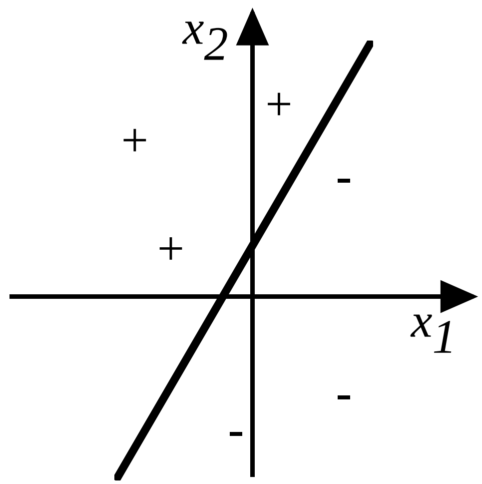
Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value.

1. Initialize all weights w_i to small random values.
2. Let $\langle \vec{x}, t \rangle$ be a training instance.
3. Compute the output $o = sgn(\vec{w} \cdot \vec{x})$.
4. If $o \neq t$, **adapt the weights**:

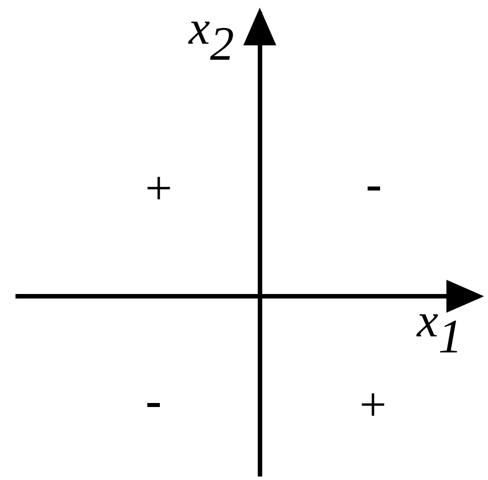
   $$w_i \leftarrow w_i + \alpha(t - o)x_i, \forall i,$$

   where $0 < \alpha < 1$ is the **learning rate**.
5. Repeat from step 2, until no errors are made.

Decision surface of a perceptron



(a)



(b)

Represents some useful functions.

Example: what weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not linearly separable! E.g. XOR.

Therefore, we will want networks of perceptron-like elements.

Convergence of the perceptron algorithm

- Converges if training data is linearly separable and α sufficiently small (usually decreased over time)
- Oscillates if the data is not linearly separable.

We would like to have an algorithm that converges when the training examples are not separable too

Ideally, it would converge to a “best fit” or “minimum error” on the training data