cis32-ai — lecture # 11 — wed-8-mar-2006

today's topics:

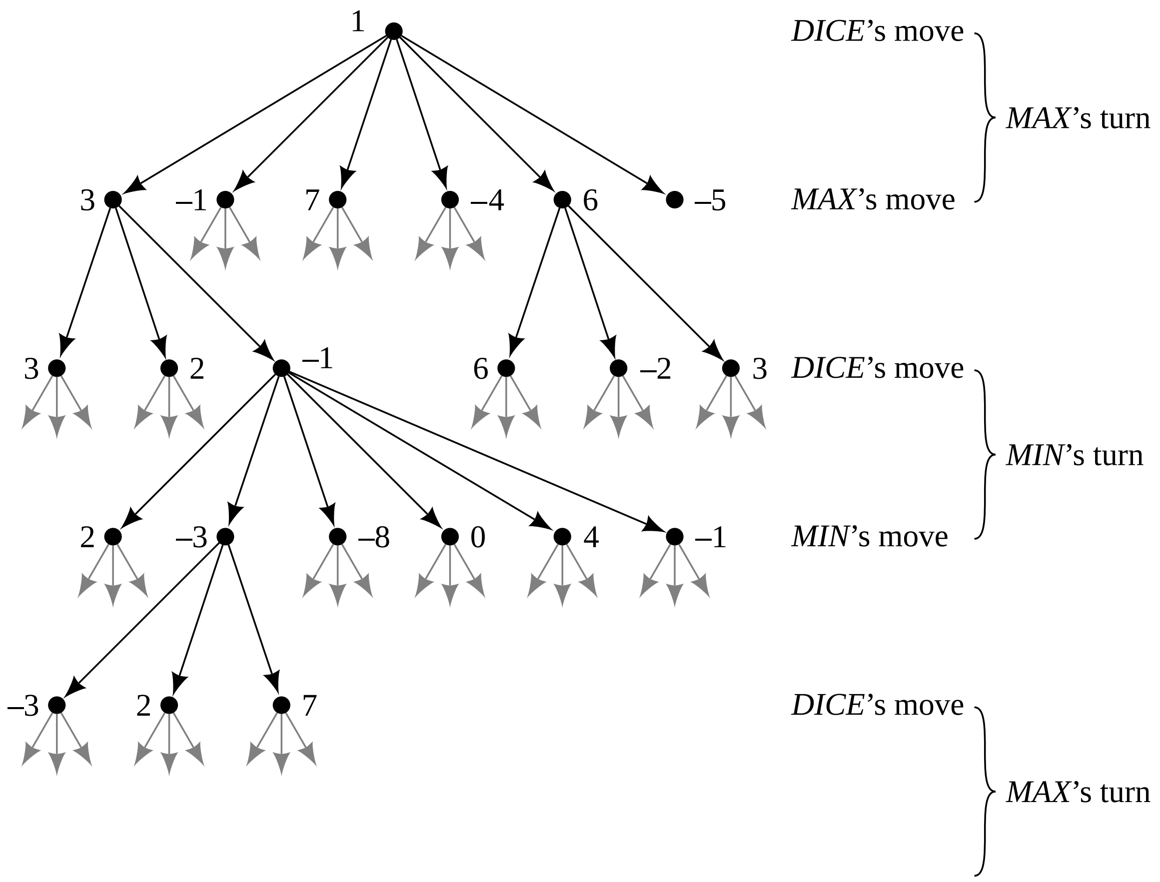
- finish up adversarial search
- neural networks

Horizon effects

- How do we know when to stop searching?
- What looks like a very good position for MAX might be a very bad position just over the horizon.
- Stop at *quiescent* nodes (value is the same as it would be of you looked ahead a couple of moves).
- Can be exploited by opponents; pushing moves back behind the horizon.
- A similar problem occurs because we assume that players always make their best move:
  - "Bad" moves can mislead a minimax-style player.

Games of chance

- How do we handle dice games?
- A neat trick is to model this as a another player DICE.
- We back up values in the usual way, maximising for MAX and minimising for MIN.
- For DICE moves, we back up the expected (weighted average) of the moves.
- For a single die, the weight is $1/6$.
- For more complex situations we use whatever probability distribution is indicated.


1
3
–1
7
–4
6
–5
3
2
–1
6
–2
3
2
–3
–8
0
4
–1
–3
2
7
DICE's move
MAX's turn
MAX's move
DICE's move
MIN's turn
MIN's move
DICE's move
MAX's turn

Summary: adversarial search

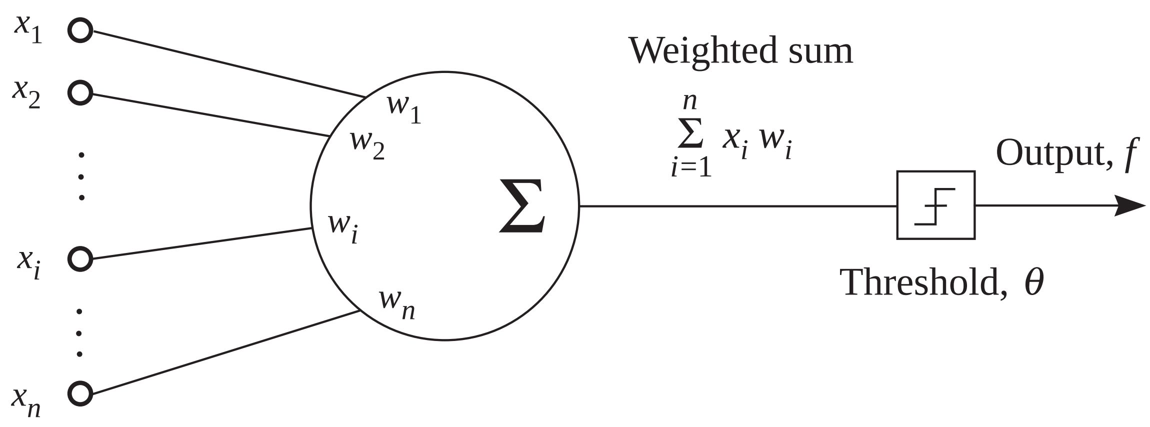
- We have looked at game playing as adversarial state-space search.
- Minimax search is the basic technique for finding the best move.
- Alpha/beta search gives greater efficiency.
- Games of chance can be handled by adding in the random player DICE.

Neural Networks: Introduction

- Now we will look at *neural networks*, so called because they mimic the structure of the brain.
- However, they don't have to be viewed in this way.
- We will start by thinking of them as an implementation of the kind of stimulus-response agents we looked at in the last lecture.
- They also provide us with our first taste of learning.
- The learning angle means we don't have to figure out the model parameters for ourselves.

Networks for Stimulus-Response

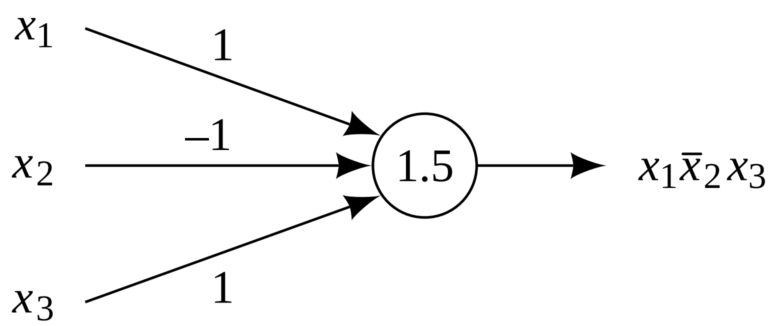
- Production systems can be easily implemented as computer programs.
- They may also be implemented directly as electronic circuits, as combinations of AND, OR, and NOT gates.
- (Or as simulations of electronic circuits.)
- One useful kind of circuit is built of elements whose output is a nonlinear function of a weighted combinations of its inputs.
- One kind of such unit is a *threshold logic unit* (TLU).
- This computes a weighted sum of its imputs, compares this to a threshold, and outputs 1 if the threshold is exceeded, 0 otherwise.



$$f = 1 \text{ if } \sum_{i=1}^{n} x_i w_i \geq \theta$$
$$= 0 \text{ otherwise}$$

- The Boolean functions that can be computed using a TLU are called *linearly seperable* functions.

- We can use TLUs to implement some Boolean functions, for instance a simple conjunction:



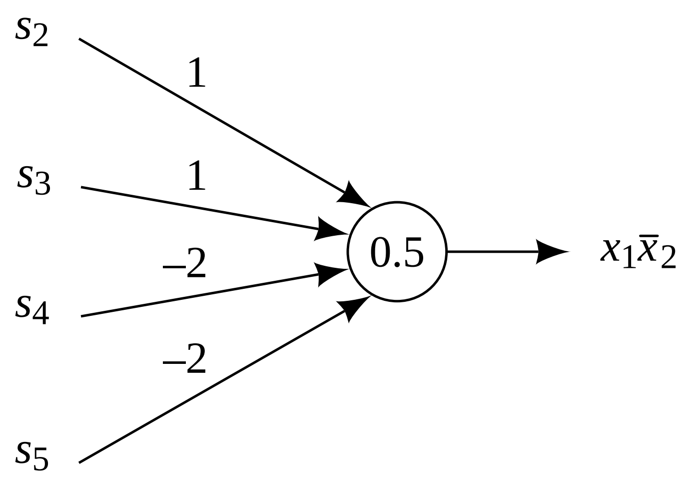
but we can't implement an exclusive-OR this way.

- We can implement the kind of function used for boundary following:

$$
\begin{aligned}
x_1\overline{x_2} &= (s_2 + s_3)\overline{(s_4 + s_5)} \\
&= (s_2 + s_3)\overline{s_4 s_5}
\end{aligned}
$$

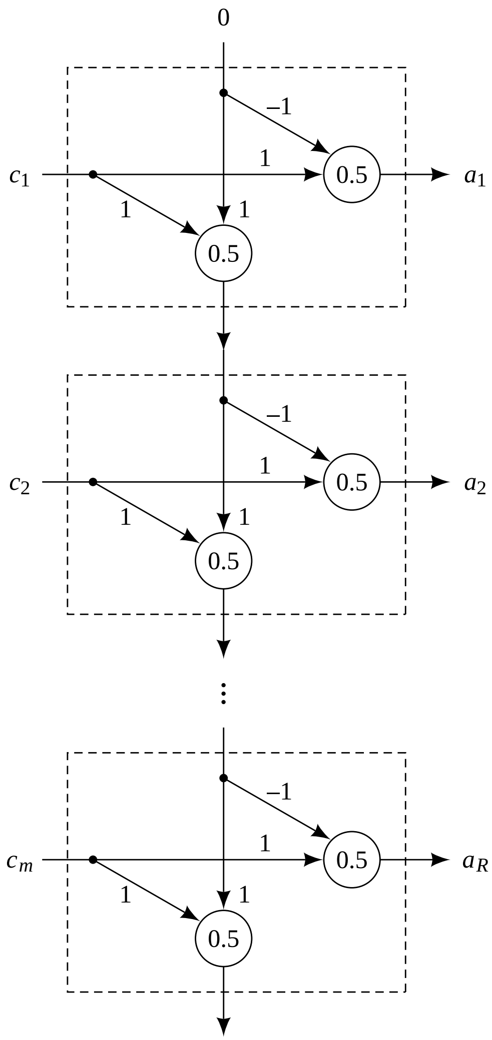
  as the figure overleaf

- If you don't see why, figure out what the weighted sum is for different combinations of sensor readings.

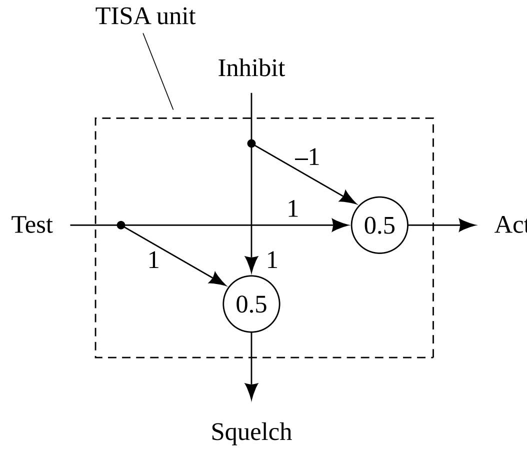


s_2
s_3
s_4
s_5
1
1
–2
–2
0.5
$x_1\bar{x}_2$

- When we have a simple problem, it is possible that a single TLU can compute the right action.
- For this to happen we need there to be only two possible actions.
- For more complex problems, we need a network of TLUs.
- These are often called *neural networks* because they have some similarity to the networks of neurons from which the brain is constructed.
- We can use such a network to implement a T-R program.



0
–1
1
c_1
0.5
a_1
1
1
0.5
–1
1
c_2
0.5
a_2
1
1
0.5
⋮
–1
1
c_m
0.5
a_R
1
1
0.5



TISA unit
Inhibit
–1
Test
1
0.5
Act
1
1
0.5
Squelch

- This network implements a set of production rules.
- The input to each unit on the left is the 1 or 0 of the condition.
- (This might be computed from the s_i by another circuit.)
- Each rule is a Test, Inhibit, Squelch, Act (TISA) circuit:
  - One TLU computes a conjunction.
  - The other computes a disjunction.

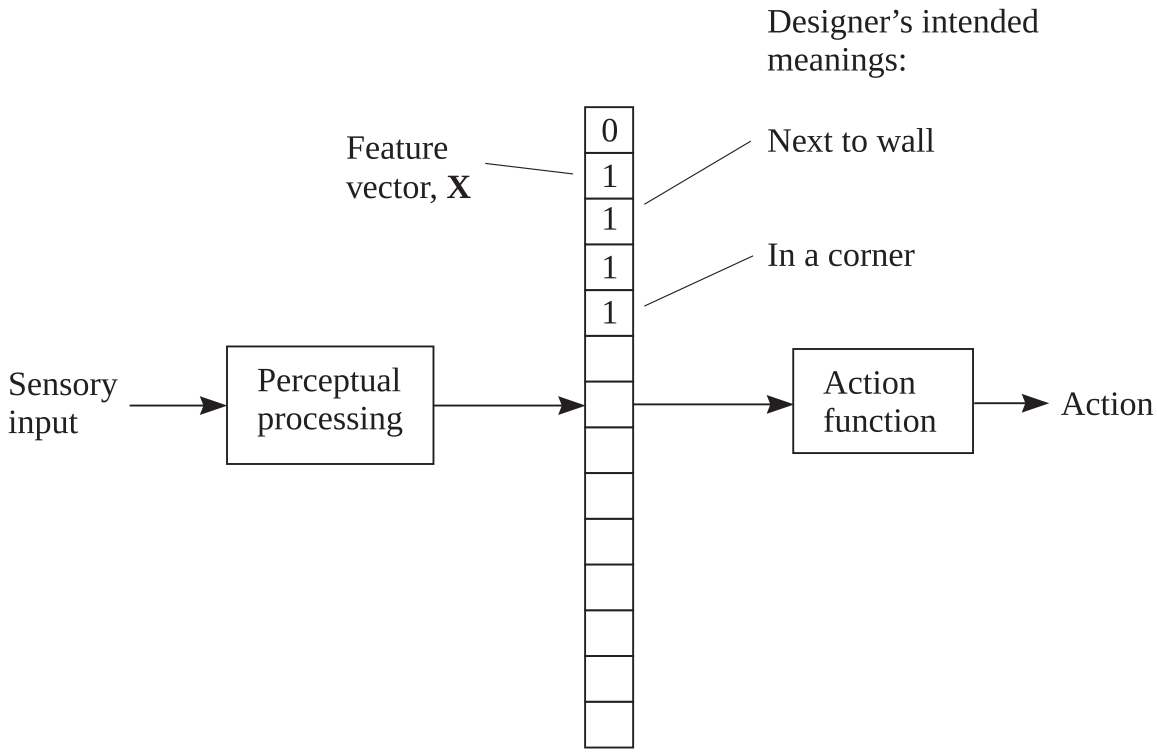
- Inhibit is 0 when no rules above have a true condition.
- Test is 1 if the condition is true.
- If Test is 1 and Inhibit is 0, Act is 1.
- If either Test is 1 or Inhibit is 1 then Squelch is 1.
- If Squelch is 1 then every TISA below is Inhibited.

Learning in neural networks

- So far we have assumed that the mapping between stimulus and response was programmed by the agent designer.
- That is not always convenient or possible.
- When it isn't, then it is possible to *learn* the right mapping.
- We will start to examine one way of doing that in this lecture.
- We will look at the case of learning the mapping for a single TLU.

- In brief, the learning procedure is as follows.
- We start with some set of weights:
  - random;
  - uniform
- We then run a set of inputs, and look at the outputs.
- If they don't match, we alter the weights.
- We keep learning until the weights are right.

- Remember the set up we had before.
- We have a feature vector X, which maps to a particular action a.

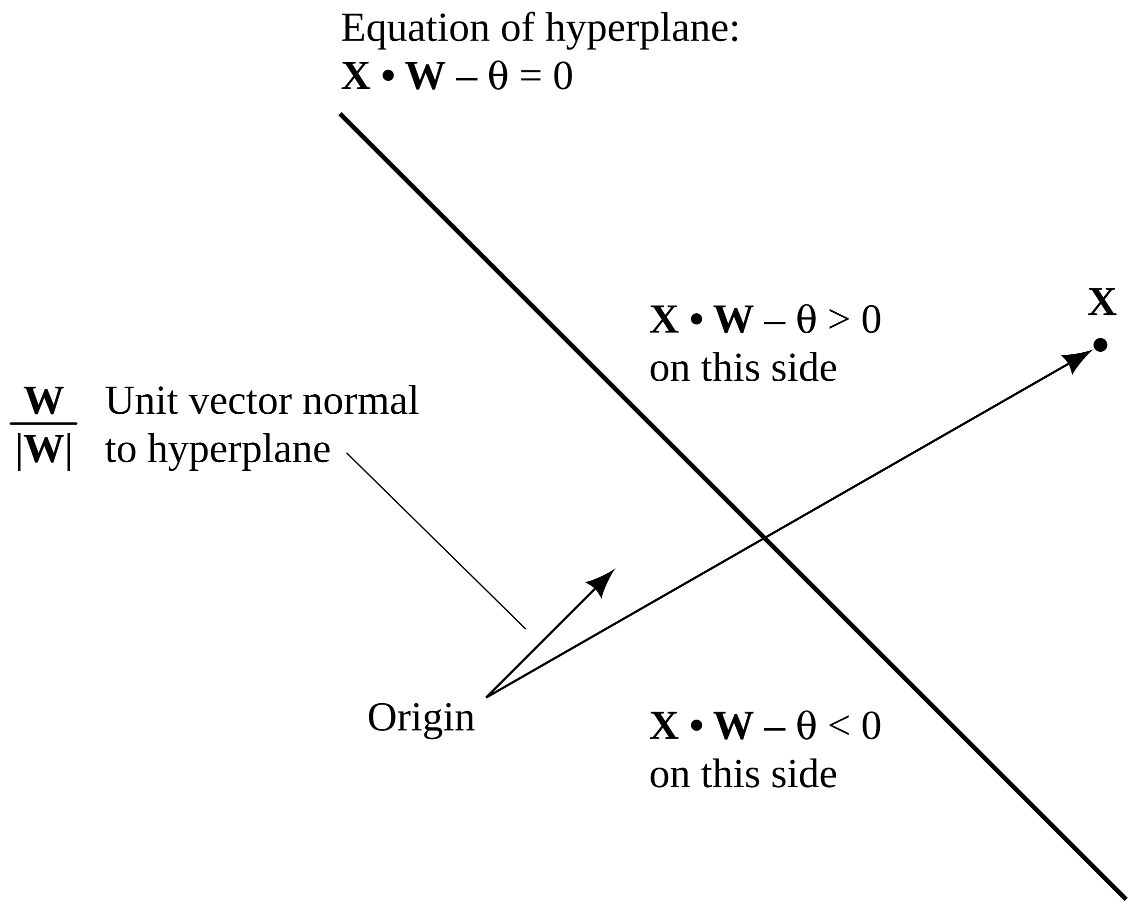


- Now consider that we have a set of these Θ.
- Every element of Θ is an X with a corresponding a.
- This is a *training set*, and the set A of all a are called the *classes* or *labels*.
- The learning problem here is to find a way of describing the mapping from each member of Θ to the appropriate member of A.
- We want to find a function $f(X)$ which is "acceptable".
- That is it produces an action which agrees with the examples for as many members of the training set as possible.
- Because we have a set of examples to learn from, we call this *supervised learning*.

Learning in a single TLU

- We train a TLU by adjusting the input weights.
- We assume that the vector X is numerical so that a weighted sum makes sense.
- The set of weights $w_1, w_2, \ldots, w_n$ is denoted by W.
- The threshold is written as θ, so:
  - Output is 1 if
$$s = X \cdot W > \theta$$
  - Output is 0 otherwise
- $X \cdot W$ is just a way of writing $x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$

- A TLU divides the space of feature vectors Θ:



- In two dimensions, the TLU defines a boundary between two parts of a plane (as in the picture).
- In three dimensions, the TLU defines a plane which separates two parts of the space.
- In higher-dimension spaces the boundary defined by the TLU is a hyperplane.
- Whatever it is, it separates:

$$X \cdot W - \theta > 0$$

  from

$$X \cdot W - \theta < 0$$

- Changing θ moves the boundary relative to the origin.
- Changing W alters the orientation of the boundary.
- Following the textbook we will assume that:

$$\theta = 0$$

- This simplifies the subsequent maths :-)
- Arbitrary thresholds can be obtained by adding in an extra weight $n + 1$ which is $-\theta$.
- The $n + 1$th element of the input vector is always 1.
- So, we don't restrict ourselves by making this assumption.

Summary: Neural Networks

- So, we introduced neural networks.
- We first considered them as an implementation of stimulus-response agents.
- In this incarnation we adjust the weights by hand.
- We also started thinking about how to learn the weights automatically.
- We will finish this line of work off next lecture.