Foundations of Artificial Intelligence

CS472/3

Lecture #38 Bart Selman

Slide CS472–1

- Neural Net Learning

Slide CS472–2

Perceptrons

Recap slides from previous slides.
Representational limit:
- linearly separable functions only.
  - intuition? xor example.
  - xor with hidden layer.
  - connectedness example (Minsky/Papert)

Perceptron Learning

A perceptron can learn any linearly separable function, given enough enough training examples.

What does this say about linearly separable functions?

Key idea: **adjust weights till all examples correct.**

update weights repeatedly (epochs) for each example.

weight update

Single output O; target output for example T.

Define error: $Err = T - O$

Now, just move weights in right direction!

If error is positive, then need to increase O.

Each input unit j contributes $W_j \, I_j$ to total input.

if I_j is positive, increasing W_j tends to increase O

if I_j is negative, decreasing W_j tends to increase O

So, use:

$W_j \leftarrow W_j + \alpha \times I_j \times Err$

Perceptron learning rule (Rosenblatt 1960). α is **learning rate**.

Rule is intuitively correct.

Gradient descent through weight space.

Surprise is **proof** of convergence.

Weight space has **no local minima.**

With enough examples, it will find the function.

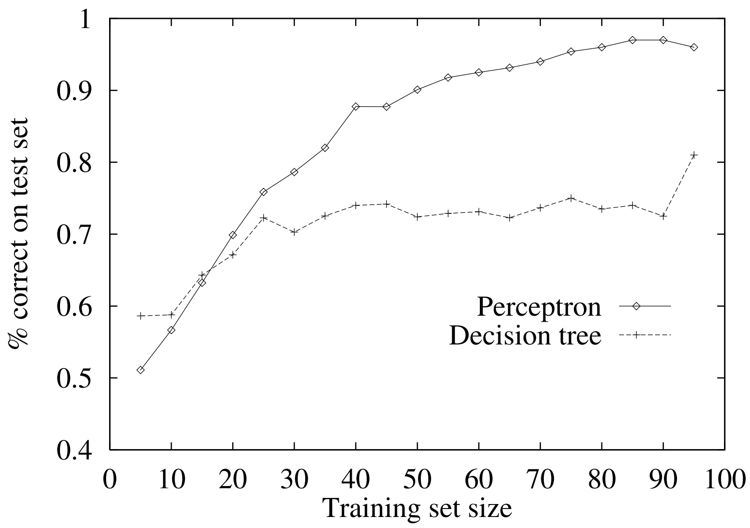
(provided α not too large)

Explains early popularity.

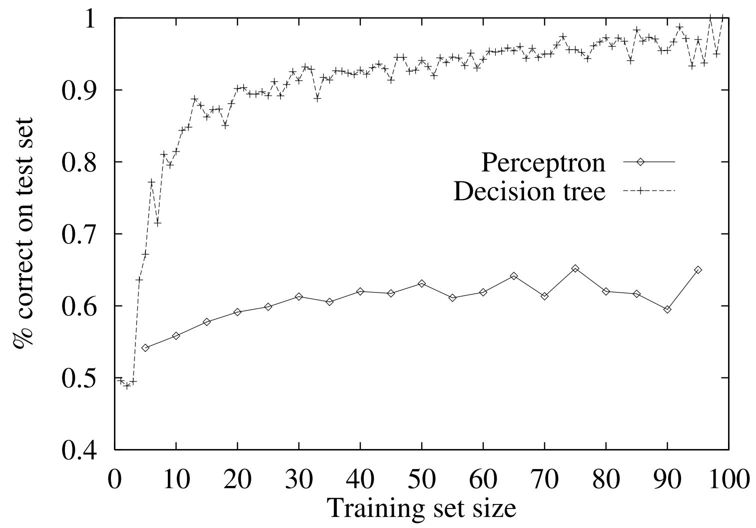
```
function NEURAL-NETWORK-LEARNING(examples) returns network

  network ← a network with randomly assigned weights
  repeat
     for each e in examples do
        O ← NEURAL-NETWORK-OUTPUT(network, e)
        T ← the observed output values from e
        update the weights in network based on e, O, and T
     end
  until all examples correctly predicted or stopping criterion is reached
  return network
```

Slide CS472–7



Slide CS472–8



Slide CS472–9

Perceptrons

From Patrick Winston (MIT) book

Basic learning strategy for perceptron:

(simplified from R&N; e.g., learning rate = 1)

Framework and notation:

0/1 signals

input signal: $\overrightarrow{X} =< x_1, x_2, \ldots, x_n, x_{n+1} >$

weight vetor: $\overrightarrow{W} =< w_1, w_2, \ldots, w_n, w_{n+1} >$

$x_{n+1} = 1$ with w_{n+1} simulates threshold.

O is output signal (0 or 1) (single output)

Slide CS472–10

Threshold function of perceptron:

Let $S = \sum_{k=1}^{k=n+1} w_k \times x_k$

If $S > 0$ then $O = 1$,
else $O = 0$.

Slide CS472–11

We want to train the perceptron with a set of examples.

Each example is given by an a pair $(\vec{X}, l)$, i.e., an input vector with a label l (0 or 1).

Slide CS472–12

Learning procedure for perceptron:

Called "the error correcting method"

Start with all zero weight vector.
Cycle (repeatedly) through examples and for
each example ($\vec{X}$, l) do:

If perceptron gives wrong answer,

if output perceptron is 0 while it should be 1,

add the input vector to the weight vector.

if output perceptron is 0 while it should be 1,

subtract the input vector to the weight vector.

otherwise do nothing.

Slide CS472–13

Note that procedure is intuitively correct.

(e.g., if output is 0 but should be 1 the weights
are increased.)

The remarkable thing is that the procedure can
be proved to converge (in poly steps) if
the function can be represented by perceptron
(i.e. is linearly separable)

Slide CS472–14

Let's do an example.

Consider learning the logical "or" function.

Our examples are:

```
sample       x_1    x_2    x_3      l
====================================
1              0      0      1      0
2              0      1      1      1
3              1      0      1      1
4              1      1      1      1
```

Slide CS472–15

We'll use a single perceptron with three inputs (x_1, x_2, x_3) and single output (l).

Note artificial input x_3 fixed at 1.

Slide CS472–16

We start with all weights at 0: $< 0, 0, 0 >$
Let's consider the 1st example.
Perceptron with current weights classifies
sample 1 correctly as 0, so no change in weights.

Consider sample 2
Perceptron says 0 (note all weights still 0; $S = 0$)
should be 1, so we add input vector to weight vector
New weights: $< 0, 1, 1 >$ $(=< 0, 0, 0 > + < 0, 1, 1 >)$
Note: standard addition and substraction. Weights can
take on arbitrary integer values.

Slide CS472–17

Consider sample 3
Classified correctly; do nothing. Consider sample 4
Classified correctly; do nothing. Consider sample 1
Perceptron says 1; should be 0.
Subtract input vector:
New weights: $< 0, 1, 0 >$ $(=< 0, 1, 1 > - < 0, 0, 1 >)$

The next slide shows the next couple of steps.
Please verify those for yourself.

Slide CS472–18

```
sample 2, correctly classified
sample 3, perceptron 0; should be 1
          new weights: <1, 1, 1>
sample 4, correctly classified
sample 1, perceptron 1; should be 0
          new weights: <1, 1, 0>
sample 2, correctly classified
sample 3, correctly classified
sample 4, correctly classified
sample 1, correctly classified
```

Slide CS472–19

So, the weight vector $< 1, 1, 0 >$ classifies
all examples correctly, and the perceptron
has learned the function.

Aside: in more realistic cases the
weight w_3 will not be 0.
(This was just a toy example!)
Also, in general many more inputs (100 to 1000).

Slide CS472–20