Perceptrons and
the LMS Algorithm

15-486/782: Artificial Neural Networks
David S. Touretzky

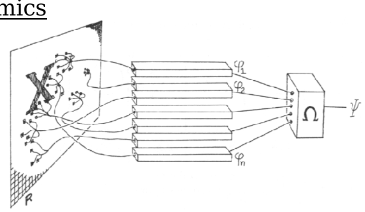
Fall 2006

1

Perceptrons

Perceptrons were the original “neural nets”.

Rosenblatt (1962)
<u>Principles of Neural Dynamics</u>



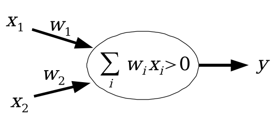
Lots of hype about “brain style computation.”

Minsky & Papert (1969)
<u>Perceptrons</u>

Lots of results of form “perceptrons can't compute x.”

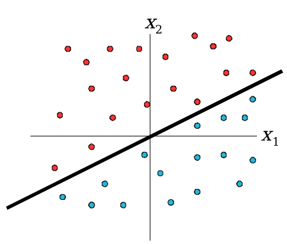
2

Peceptrons Are Linear Classifiers



$net = \sum_i w_i x_i$

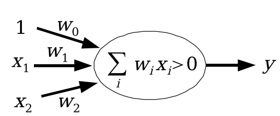
$y = \begin{cases} 1 & if\ net > 0 \\ 0 & otherwise \end{cases}$



$\vec{w} = [-1, 3]$

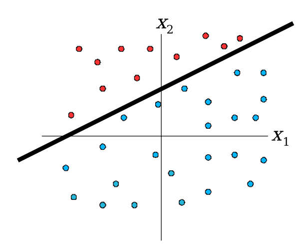
3

Decision Boundary Off the Origin?



Add a bias term w_0:

$w_0 + w_1 x_1 + w_2 x_2 > 0$



$\vec{w} = [-6, -1, 3]$

4

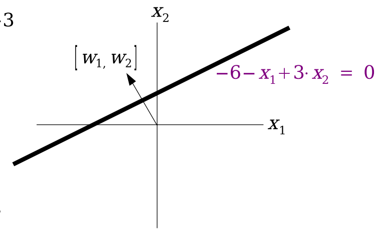
The Decision Boundary is Always Perpendicular to the Weight Vector

$\vec{w} = [-6,\ -1,\ 3]$

slope of weight vector $= w_2 / w_1 = -3$

slope of decision boundary $= 1/3$

If a line has slope m, the perpendicular has slope -1/m.



We're ignoring the bias term w_0, but the bias can only raise or lower the line, not change its slope.

Scaling the weight vector has no effect on the decision boundary!

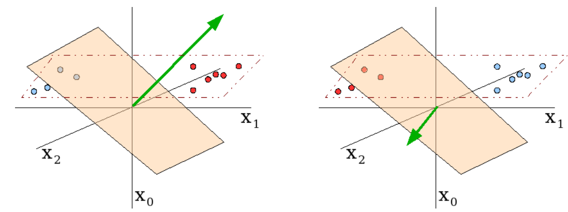
5

The Real Decision Boundary in 3D

The decision boundary passes through the origin.

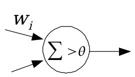
Data points lie in the plane at x_0=1.

Negating **w** leaves the boundary unchanged but flips the sign of the results.

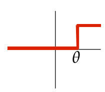


6

Thresholds vs. Biases

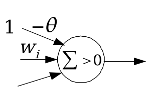


$$y = \begin{cases} 1 & \text{if} \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$



Learning rules adjust both $\vec{w}$ *and* θ

Simpler solution: $w_0 = -\theta$



A bias term of $-\theta$ *with threshold zero is equivalent to a threshold of* θ

7

Perceptron Learning Rule

Initialize $\vec{w} \leftarrow 0$

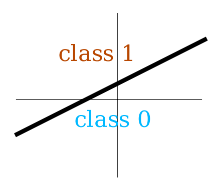
For each $\langle \vec{x}_i, d_i \rangle$ *in training set:*

$net = \vec{x}_i \cdot \vec{w}$

$$y = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{w} \leftarrow \begin{cases} \vec{w} & \text{if } y = d_i \\ \vec{w} + \vec{x}_i & \text{if } y < d_i \\ \vec{w} - \vec{x}_i & \text{if } y > d_i \end{cases}$$

Repeat until all $\vec{x}_i$ *classified correctly.*



8

How to Run the Matlab Demos

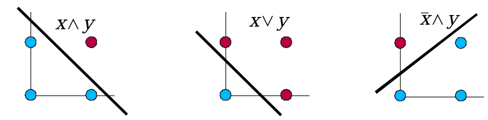
- cd /afs/cs/academic/class/15782-f06/matlab/perceptron

  ... or ...

  download the file perceptron.zip from the course web site, unzip it, and cd to the perceptron directory

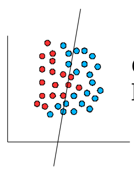
- matlab
- ls
- perceptron

Learning Boolean Functions

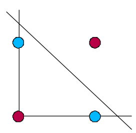


Are all Boolean functions learnable?

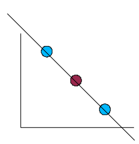
Some Problems Aren't Linearly Separable



Convex classes aren't linearly separable

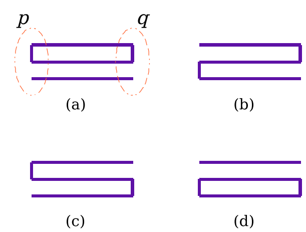


XOR

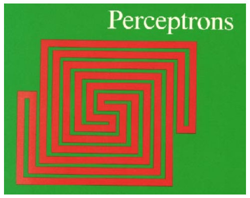


Not in "general position"

Perceptrons Can't Compute XOR



	p	q	$p\,XOR\,q$
a	0	0	0
b	1	0	1
c	0	1	1
d	1	1	0



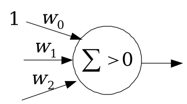
Minsky & Papert:
Perceptrons can't compute "connectedness".

Prove That Perceptrons Can't Compute XOR

	x_1	x_2	d
1.	0	0	0
2.	1	0	1
3.	0	1	1
4.	1	1	0

5. $w_0 \leqslant 0$ $(by\,1)$
6. $-w_1 < w_0$ $(by\,3)$
7. $-w_2 < w_0$ $(by\,2)$
8. $w_1 + w_2 < -w_0$ $(by\,4)$
9. $0 < w_0$ $(add\,6,7,8)$
10. $w_0 > 0$ $(by\,9)$

Lines 5 and 10 conflict



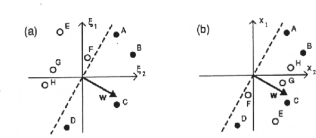
$w_0 + w_1 x_1 + w_2 x_2 \;>\; 0$

Let's Use +1/-1 Outputs

$$y \;=\; sgn(net) \;=\; \begin{cases} +1 & if\, net > 0 \\ -1 & otherwise \end{cases}$$

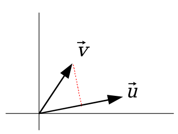
Let ϕ^n *= input pattern n*
Let t^n *= class of pattern n (-1 or +1)*

If a problem is linearly separable, all $\phi^n t^n$ *lie on the same side of the decision boundary.*



ϕ *space* $\qquad$ $\phi \cdot t$ *space*

Review of Dot Product



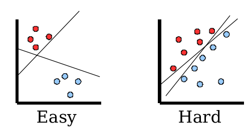
$$\vec{u} \cdot \vec{v} \;=\; \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos\theta$$

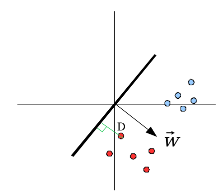
$$\vec{u} \;=\; [u_1 \;\; u_2 \;\; u_3]$$
$$\vec{v} \;=\; [v_1 \;\; v_2 \;\; v_3]$$

$$\vec{u} \cdot \vec{v} \;=\; u_1 v_1 + u_2 v_2 + u_3 v_3$$

If $\vec{u}$ *is a unit vector, then* $\vec{u} \cdot \vec{v}$ *is the length of the projection of* $\vec{v}$ *along* $\vec{u}$.

Easy vs. Hard Problems





$$D(\vec{w}) \;=\; \frac{1}{\|\vec{w}\|} \min_i (\vec{w} \cdot \phi^i t^i)$$

$$D_{max} \;=\; \max_{\vec{w}} D(\vec{w})$$

Large $D_{max} \;\rightarrow$ *easy problem.*
$D_{max} < 0 \;\rightarrow$ *not linearly separable.*

For AND, $D_{max} = \frac{1}{\sqrt{17}}$. $\quad$ *For XOR,* $D_{max} = \frac{-1}{\sqrt{3}}$

Perceptron ConvergenceTheorem
Rosenblatt (1962)

This theorem is very famous.

The version that follows is from Bishop (1995), based on Hertz, Krogh, and Palmer (1991).

Theorem:

If a problem is linearly separable, then
a perceptron will learn it
in a <u>finite number</u> of steps.

Proof of the Theorem (1)

Assume a vector $\hat{w}$ exists that correctly classifies all points.

Then $\hat{w}\cdot(\phi^n t^n) > 0$ *for all n.*

At each step of the algorithm:

$\bar{w}^{(\tau)} =$ *weights at step* τ

ϕ^n *is the misclassified vector at the current step*

$\bar{w}^{(\tau+1)} \leftarrow \bar{w}^{(\tau)} + \phi^n t^n$

Suppose ϕ^n *has been misclassified* τ^n *times so far.*

Total misclassifications $\tau = \sum_n \tau^n$

Therefore $\bar{w}^{(\tau)} = \sum_n \tau^n \phi^n t^n$

(assuming $\bar{w}^{(0)} = 0$*)*

Proof of the Theorem (2)

Find a lower bound on the growth rate of $\hat{w}\cdot\bar{w}^{(\tau)}$.

$$\hat{w}\cdot\bar{w}^{(\tau)} = \sum_n \tau^n \hat{w}\cdot\phi^n\cdot t^n$$
$$\geq \tau \min_n\left(\hat{w}\cdot\phi^n t^n\right)$$

So $\hat{w}\cdot\bar{w}^{(\tau)}$ *is bounded from below by a function that grows linearly in* τ.

If the algorithm runs forever, $\hat{w}\cdot\bar{w}^{(\tau)}$ *diverges.*

Proof of the Theorem (3)

Find an upper bound on the growth rate of $\bar{w}^{(\tau)}$.

$$\bar{w}^{(\tau+1)} = \bar{w}^{(\tau)} + \phi^n\cdot t^n$$

$$\|\bar{w}^{(\tau+1)}\|^2 = \|\bar{w}^{(\tau)}\|^2 + \|\phi^n\|^2 (t^n)^2 + 2\,\bar{w}^{(\tau)}\cdot\phi^n t^n$$
$$\leq \|\bar{w}^{(\tau)}\|^2 + \|\phi^n\|^2 (t^n)^2$$

because $\bar{w}^{(\tau)}\cdot\phi^n t^n < 0$ *since* ϕ^n *was misclassified.*

Note: $(t^n)^2 = 1$ *since* $t^n = \pm 1$

Let $\|\phi\|_{max} = \max_n \|\phi^n\|$

Then $\|\bar{w}^{((\tau)+1)}\|^2 - \|\bar{w}^{(\tau)}\|^2 \leq \|\phi\|^2_{max}$

Since $\|\bar{w}^{(0)}\| = 0$*, after* τ *weight updates we have:*

$$\|\bar{w}^{(\tau)}\|^2 \leq \tau \|\phi\|^2_{max}$$

Proof of the Theorem (4)

Show the bounds must cross.

$$\|\bar{w}^{(\tau)}\|^2 \leqslant \tau \|\phi\|^2_{max}$$

So $\|\bar{w}^{(\tau)}\|$ *grows no faster than* $\sqrt{\tau}$

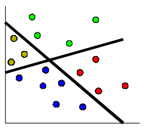
But $\hat{w} \cdot \bar{w}^{(\tau)}$ *has a lower bound that is linear in* τ *:*

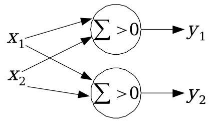
$$\hat{w} \cdot \bar{w}^{(\tau)} \geqslant \tau \min_n \left(\hat{w} \cdot \phi^n t^n \right)$$

The bounds would eventually cross if τ *got large enough.*
Hence, τ *must bounded, meaning we achieve correct classification of all points in a finite number of steps.*

QED.

More Than 2 Classes

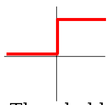




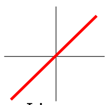
Classes:
- 0 0
- 0 1
- 1 0
- 1 1

The two neurons learn independently.

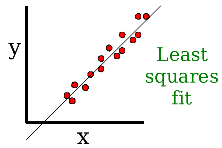
Linear Units: Function Approximators

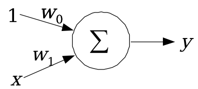


Threshold unit



Linear unit





$$y = \sum_i w_i x_i$$

The LMS (Least Mean Squares) Learning Algorithm

Define total sum-squared error over the training set:

$$E = \frac{1}{2} \sum_j \left(d_j - y_j \right)^2$$

Do gradient descent in the error E:

$$\frac{\partial E}{\partial y} = y - d \qquad \frac{\partial y}{\partial w_i} = \frac{\partial}{\partial w_i} \sum_i w_i x_i = x_i$$

Chain rule:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_i} = (y - d) x_i$$

Gradient descent in E:

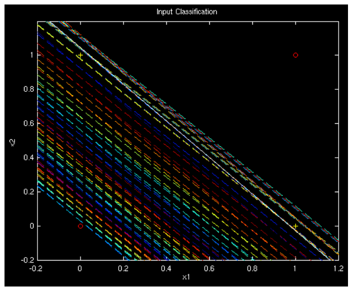
$$\Delta w_i = -\eta (y - d) x_i \qquad \eta \text{ is a learning rate constant}$$

LMS Convergence

If the learning rate η is small enough, LMS will always converge.

When $|E(t+1) - E(t)| < 0.001$, *stop.*

What about XOR?

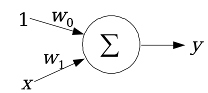


25

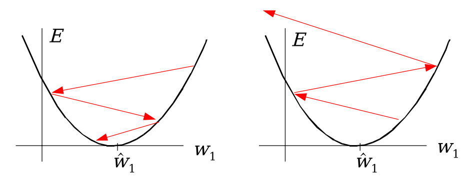
Why LMS Can Blow Up

Error is quadratic in $\vec{w}$.

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



So the error surface forms a bowl.
The one-dimensional projection is a parabola.

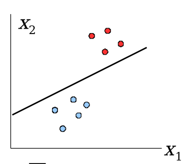


Small η *Large* η

See bowl and parabolas demos.

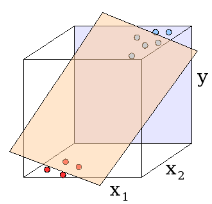
26

Classification vs. Mapping



$\sum w_i x_i > 0$

Train with perceptron algorithm.



$y = \sum w_i x_i$

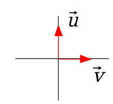
Train with LMS.

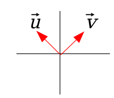
There are some pathological cases where LMS won't classify all points correctly, but the perceptron algorithm will.

27

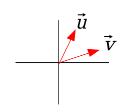
Orthogonality and Linear Independence

Orthogonal:

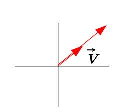


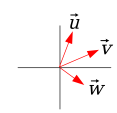


Linearly independent:



Not linearly independent:

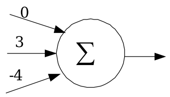




28

LMS Works Best with Orthogonal Input Patterns

$$Patterns = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Desired = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$



Even if not orthogonal, LMS will find a perfect solution as long as the patterns are linearly independent.

If not linearly independent, patterns interfere with each other and total sum-squared error cannot reach 0.

The Rescorla-Wagner Model of Animal Learning

UCS = shock

UCR = jumps; tries to escape

CS_1 = light

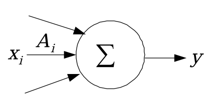
CS_2 = tone

CR = fear response: freezing, shivering, inhibition of drinking

Rescorla-Wagner is a Linear Model

A_i = *Associative strength between* CS_i *and UCS.*

x_i = *presence of* CS_i: [0,1]



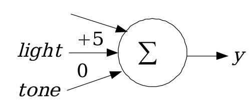
$$Conditioned\ Response = y = \sum_i A_i x_i$$

Conditioning Experiments

Simple conditioning:

Train: light --> UCS

Tests: light --> CR
tone --> no CR

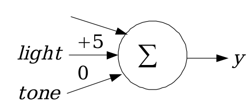


Blocking:

Train1: light --> UCS

Train 2: light + tone --> UCS

Tests: light --> CR
tone --> no CR

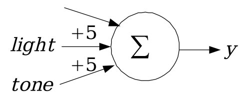


Conditioning Experiments

Summation:

Train: light --> UCS
tone --> UCS

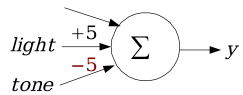
Tests: light --> CR
tone --> CR
light + tone --> big CR



Conditioned inhibition:

Train: light --> UCS
light + tone --> no UCS

Tests: light --> CR
light + tone --> no CR
"summation test"
"retardation test"



33

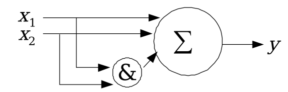
Rescorla-Wagner = LMS

The Rescorla-Wagner learning rule is the LMS rule, also called the delta rule or the Widrow-Hoff rule.

(Computer trivia: Ted Hoff later became Intel employee #12 and was a co-inventor of the microprocessor.)

Problem: Rescorla-Wagner can't learn XOR.
But rats can!

Solution: Use a conjunctive unit as a third input.
(Ad hoc? Cf. work on "configural learning".)



34

Limitations of Rescorla-Wagner

- The Rescorla-Wagner model neatly captures certain aspects of conditioning, but it is far from a complete model.
- It cannot handle:
  - Second-order conditioning
  - Latent inhibition
  - Rapid recovery from extinction
  - Stimulus timing effects (not a real-time model)
  - Effects of partial reinforcement on extinction
  - Response to novelty

35

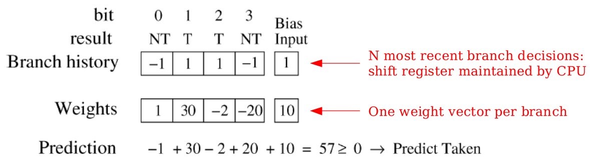
Application: Using a Perceptron for Branch Prediction

- "Branch prediction" is used in modern CPUs to support instruction-level parallelism through speculative execution.
- Processor "guesses" whether a branch will be taken or not, before the computation to determine the branch condition is completed.
- If guess confirmed, pipeline stall is avoided. Good!
- If guess was wrong, subsequent computation must be "undone". Expensive. So try to avoid this.

36

Jimenez & Lin (ACM TOCS, 2002) Neural Net Branch Predictor

- Processor keeps track of whether each of the last N branches was taken or not.



- Learning rule: update each weight w_i by +1 if current branch is taken, or -1 if not taken.

Jimenez & Lin Results

- Perceptron branch predictor is easy to implement in hardware: integer weights, no multiplication (because input is just +1 or -1), and summation can be sped up by using a ones-complement representation.
- Performs better than several other branch predictors (gshare; McFarling-style hybrid) measured on Alpha 21264 processor.
- Intel now using perceptron branch predictor in one of their Itanium simulators.