Perceptrons and
the LMS Algorithm

15-486/782: Artificial Neural Networks
David S. Touretzky
Fall 2006

Perceptrons

Perceptrons were the original "neural nets".

Rosenblatt (1962)
Principles of Neural Dynamics

Lots of hype about "brain style computation."

Minsky & Papert (1969)
Perceptrons

Lots of results of form "perceptrons can't compute x."

Peceptrons Are Linear Classifiers

Decision Boundary Off the Origin?

Add a bias term \( w_0 \):

\[ w_0 + w_1x_1 + w_2x_2 > 0 \]
The Decision Boundary is Always Perpendicular to the Weight Vector

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

slope of weight vector: \( w_1/w_2 = -3 \)
slope of decision boundary: \( 1/3 \)

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

We're ignoring the bias term \( w_3 \) 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!

Thresholds vs. Biases

\[ y = \begin{cases} 
1 & \text{if } \sum w_ix_i > 0 \\
0 & \text{otherwise} 
\end{cases} \]

Learning rules adjust both \( \mathbf{w} \) and \( \theta \)

Simpler solution: \( w_0 = -\theta \)

A bias term of \(-\theta\) with threshold zero is equivalent to a threshold of \( \theta \)

The Real Decision Boundary in 3D

The decision boundary passes through the origin.
Data points lie in the plane at \( x_3 = 1 \).

Negating \( \mathbf{w} \) leaves the boundary unchanged but flips the sign of the results.

Perceptron Learning Rule

Initialize \( \mathbf{w} = 0 \)

For each \( (\mathbf{x}_i, d_i) \) in training set:

- net = \( x_i \cdot \mathbf{w} \)
- \( y = \begin{cases} 
1 & \text{if net} > 0 \\
0 & \text{otherwise} 
\end{cases} \)

\[ \mathbf{w} \leftarrow \begin{cases} 
\mathbf{w} & \text{if } y = d_i \\
\mathbf{w} + x_i & \text{if } y < d_i \\
\mathbf{w} - x_i & \text{if } y > d_i 
\end{cases} \]

Repeat until all \( \mathbf{x}_i \) classified correctly.
How to Run the Matlab Demos

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

... or ...

download the file perceptron.zip from the course website, 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

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

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p XOR q</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Prove That Perceptrons Can't Compute XOR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3.</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5.</td>
<td>$w_5&lt;0$ (by 1)</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>$w_6&lt;0$ (by 3)</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>$w_7&lt;0$ (by 2)</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>$w_8+w_9&lt;0$ (by 4)</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>$0&lt;w_9$ (add 6, 7, 8)</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>$w_9&gt;0$ (by 9)</td>
<td></td>
</tr>
</tbody>
</table>

Lines 5 and 10 conflict

Let's Use $+1/-1$ Outputs

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

Let $\phi^e =$ input pattern $n$
Let $t^e =$ class of pattern $n$ (-1 or +1)

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

Review of Dot Product

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

$\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

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

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

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

For AND, $D_{\max} = \frac{1}{\sqrt{3}}$
For XOR, $D_{\max} = \frac{-1}{\sqrt{3}}$
Perceptron Convergence Theorem
Rosenblatt (1962)

This theorem is very famous.

Theorem:
If a problem is linearly separable, then a perceptron will learn it in a finite number of steps.

Proof of the Theorem (1)

Assume a vector \( \tilde{w} \) exists that correctly classifies all points. Then \( \tilde{w}(t^*) > 0 \) for all \( n \).

At each step of the algorithm:
\[
\tilde{w}^{(t)} = \text{weights at step } t
\]
\( \phi^k \) is the misclassified vector at the current step
\[
\tilde{w}^{(t+1)} = \tilde{w}^{(t)} + \phi^k t
\]

Suppose \( \phi^k \) has been misclassified \( t^* \) times so far.
Total misclassifications \( \tau = \sum_n t^* \)

Therefore \( \tilde{w}^{(T)} = \sum_n t^* \phi^k t^* \)
(assuming \( \tilde{w}^{(0)} = 0 \))

Proof of the Theorem (2)

Find a lower bound on the growth rate of \( \tilde{w} \cdot \tilde{w}^{(t)} \).
\[
\tilde{w} \cdot \tilde{w}^{(t)} = \sum_n t^* \tilde{w}_n \phi^k t^*
\]
\[
> \tau \min_n \left| \tilde{w}_n \phi^k t^* \right|
\]

So \( \tilde{w} \cdot \tilde{w}^{(t)} \) is bounded from below by a function that grows linearly in \( \tau \).

If the algorithm runs forever, \( \tilde{w} \cdot \tilde{w}^{(t)} \) diverges.

Proof of the Theorem (3)

Find an upper bound on the growth rate of \( \tilde{w}^{(t)} \).
\[
\tilde{w}^{(T+1)} = \tilde{w}^{(T)} + \phi^k t^k
\]
\[
\| \tilde{w}^{(T+1)} \|^2 = \| \tilde{w}^{(T)} \|^2 + \| \phi^k t^k \|^2 + 2 \tilde{w}^{(T)} \cdot \phi^k t^k
\]
\[
< \| \tilde{w}^{(T)} \|^2 + \| \phi^k t^k \|^2
\]
because \( \tilde{w}^{(T)} \cdot \phi^k t^k < 0 \) since \( \phi^k \) was misclassified.
Note: \( t^k_2 = 1 \) since \( t^k = \pm 1 \)

Let \( \| \phi \|^2 \max = \max_n \| \phi^k \|^2 \)

Then \( \| \tilde{w}^{(T+1)} \|^2 - \| \tilde{w}^{(T)} \|^2 < \| \phi \|^2 \max \)

Since \( \| \tilde{w}^{(0)} \|^2 = 0 \), after \( \tau \) weight updates we have:
\[
\| \tilde{w}^{(T)} \|^2 < \tau \| \phi \|^2 \max
\]
Proof of the Theorem (4)

Show the bounds must cross.

\[ \|w\|^2 \leq \tau \|\phi\|^2_{\text{max}} \]

So \( \|w\|^2 \) grows no faster than \( \sqrt{\tau} \). But \( \hat{w} \cdot w^{(t)} \) has a lower bound that is linear in \( \tau \):

\[ \hat{w} \cdot w^{(t)} \geq \tau \min_n (\hat{w} \cdot \phi^n \cdot \epsilon) \]

The bounds would eventually cross if \( \tau \) got large enough. Hence, \( \tau \) must be bounded, meaning we achieve correct classification of all points in a finite number of steps.

QED.

More Than 2 Classes

The two neurons learn independently.

Linear Units: Function Approximators

- Threshold unit
- Linear unit

The LMS (Least Mean Squares) Learning Algorithm

Define total sum-squared error over the training set:

\[ E = \frac{1}{2} \sum_j (d_j - y_j)^2 \]

Do gradient descent in the error \( E \):

\[ \frac{\partial E}{\partial y} = y - d \quad \frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_i w_i x_i = x_i \]

Chain rule:

\[ \frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_j} = (y - d) x_i \]

Gradient descent in \( E \):

\[ \Delta w_j = -\eta(y - d)x_i \quad \eta \text{ is a learning rate constant} \]

15-486/782: Artificial Neural Networks  David S. Touretzky  Fall 2006
LMS Convergence

If the learning rate $\eta$ is small enough, LMS will always converge.
When $|E(t+1) - E(t)| < 0.001$, stop.

What about XOR?

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.

Orthogonality and Linear Independence

Orthogonal:

Linearly independent:

Not linearly independent:

Why LMS Can Blow Up

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

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

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

See bowl and parabolas demos.
LMS Works Best with Orthogonal Input Patterns

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix} \quad \text{Desired} = [3, -4]^T
\]

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.

Rescorla-Wagner is a Linear Model

\[ A_i = \text{Associative strength between CS}_i \text{ and UCS.} \]
\[ x_i = \text{presence of CS}_i: [0, 1] \]

\[ \text{Conditioned Response} = y = \sum A_i x_i \]

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

Conditioning Experiments

Simple conditioning:
Train: light \(\rightarrow\) UCS
Tests: light \(\rightarrow\) CR

Blocking:
Train1: light \(\rightarrow\) UCS
Train 2: light + tone \(\rightarrow\) UCS
Tests: light \(\rightarrow\) CR
tone \(\rightarrow\) no CR
Conditioning Experiments

**Summation:**

Train: light $\rightarrow$ UCS
     tone $\rightarrow$ UCS

Tests: light $\rightarrow$ CR
      tone $\rightarrow$ CR
      tone + light $\rightarrow$ big CR

**Conditioned inhibition:**

Train: light $\rightarrow$ UCS
     tone $\rightarrow$ no UCS

Tests: light $\rightarrow$ CR
      light + tone $\rightarrow$ no CR
      "summation test"
      "retardation test"

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").

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.

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

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

<table>
<thead>
<tr>
<th>bit result</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Bias Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch history</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Weights: 1 30 -2 -20 10

One weight vector per branch

Prediction: \(-1 + 30 - 2 + 20 + 10 = 57 \geq 0\) → Predict Taken

- Learning rule: update each weight \(w\) 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.