Perceptron

2013-03-26
Eun-Sol Kim
**NEURON.** A neuron fires by transmitting electrical signals along its axon. When signals reach the end of the axon, they trigger the release of neurotransmitters that are stored in pouches called vesicles. Neurotransmitters bind to receptor molecules that are present on the surfaces of adjacent neurons. The point of virtual contact is known as the synapse.
Basic operation

• Intercell communication via the synapse
• Can be excitatory (making a receiving neuron more likely to fire) or inhibitory (making it less likely to fire)
• Typically communicate via neurotransmitters
  • Released on the axon side and trigger electrical changes on the dendrite side
• Neurologists believed that the basic unit of information is the rate of firing of a neuron
  • This is usually discussed in terms of a neuron’s activation level
McCulloch/Pitts neurons

- McCulloch/Pitts neurons can then be used to compute any (finite) logical function

\[ \theta = 1.5 \quad \text{AND} \]

\[ \theta = 0.5 \quad \text{OR} \]

- BUT, McCulloch/Pitts networks can’t learn.
Perceptron (Rosenblatt, 1958, 1962)

- Rosenblatt explored the properties of networks of McCulloch-Pitts neurons (linear-threshold) with connections that could be modified by learning
Perceptron: architecture

- Most commonly discussed architecture
  - Only connections between feature units and output unit was modifiable (the $w_i$’s). The input feature unit values ($x_i$) were set by hand.
Perceptrons

\[ x_0 \quad 1 \]
\[ x_1 \quad 0 \]
\[ x_2 \quad 0 \]

\[ \sum \quad -0.06 \]

\[ x_1 \quad x_2 \quad 0 \]
\[ 0 \quad 0 \quad 0 \]
\[ 0 \quad 1 \quad 0 \]
\[ 1 \quad 0 \quad 0 \]
\[ 1 \quad 1 \quad 1 \]

RIGHT
Perceptrons

\[
\begin{align*}
X_1 & \quad X_2 & \quad 0 \\
0 & \quad 0 & \quad 0 \\
0 & \quad 1 & \quad 0 \\
1 & \quad 0 & \quad 0 \\
1 & \quad 1 & \quad 1 \\
\end{align*}
\]

RIGHT
Perceptrons

\[
\sum \cdot .06
\]

\[
\sum \cdot .1
\]

\[
\sum \cdot .05
\]

\[
\sum \cdot .16
\]

\[
\begin{array}{ccc}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

RIGHT
Perceptrons

\[
\sum = x_1 \times 0.06 + x_2 \times -0.1 + 1 \times 0.05 - 0.11
\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

WRONG
Perceptrons

Fails to fire, so add proportion, \( \eta \), to weights.

\[
\begin{array}{ccc}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Perceptrons

\[ \sum = \eta \cdot 0.01 \]

\[ \sum = -0.06 + 0.01x_1 \]

\[ \sum = -0.1 + 0.01x_1 \]

\[ \sum = 0.05 + 0.01x_1 \]

\[ \begin{array}{c|c|c}
  x_1 & x_2 & 0 \\
  \hline
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
\end{array} \]
Perceptrons

\[
\begin{array}{c|c|c}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Perceptrons

Table:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Decrease!
Perceptrons

\[ \eta = 0.01 \]

-0.05 - 0.01x1

-0.09

0.06 - 0.01x1

\[
\begin{array}{ccc}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Perceptrons

\[ \Sigma \]

\[ \eta = .01 \]

\[ \begin{array}{ccc}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array} \]

\[-.08\]
Perceptrons

\[ \eta = 0.01 \]

\[
\begin{array}{c}
\sum \\
.06+.01x1 \\
-.09+.01x1 \\
.06+.01x1
\end{array}
\]

\[
\begin{array}{ccc}
X_1 & X_2 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1
\end{array}
\]
Perceptrons: How to learn

- Start with random connections
- Present an input pattern
- Propagate activation through network to the output.
  - If output is correct, don’t change anything.
  - If incorrect, change weights only of active feature units
    - Rule: (Perceptron Learning Rule)
      - \( Y=\text{off}, x_i = \text{on} \rightarrow w_i \text{ decrease} \)
      - \( Y=\text{on}, x_i = \text{on} \rightarrow w_i \text{ increase} \)
- Perceptron was very powerful method for learning various relationships.
Exercise


• Observe the behavior of the Perceptron in the following cases
  – Case 1: $y = [0 \ 0 \ 0 \ 0]$
  – Case 2: $y = [1 \ 0 \ 0 \ 0]$
  – Case 3: $y = [0 \ 1 \ 1 \ 0]$
  – Case 4: $y = [0 \ 0 \ 0 \ 1]$
  – Case 5: $y = [1 \ 0 \ 0 \ 1]$
  – Case 6: the effect of learning rate (change the learning rate at your will and observe the result)
• Problem 1
  - Can the Perceptron solve the following question and **prove your answer** (hint: consider contents in slide 8):

\[
Patterns \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 2 & 6 \end{bmatrix} \quad Outputs = [0 \quad 1 \quad 1]
\]

• Problem 2
  - Explain the effect of learning rate using the following applet.
  - [http://freeisms.com/PerceptronAppletItself.html](http://freeisms.com/PerceptronAppletItself.html)
  - Experiment with learning rates of **0.06, 0.30, 0.6, 1.0** and discuss the effect of learning rate using the “error view.”
Minsky & Papert (1969)

- Presented a formal analysis of the properties of perceptrons and revealed several fundamental limitations.
- Limitations
  - Can’t learn nonlinearly separable problems like the XOR
  - More…
Linearly separable
Nonlinearly separable
Minsky & Papert cont.

• Limitations
  • So….can’t learn nonlinearly separable problems like the XOR
  • Although including “hidden layers” allows one to hand-design a
    network that can represent XOR and related problems, they
    showed that the perceptron learning rule can’t learn the required
    weights.
  • They also showed that even those functions that can be learned by
    perceptron rule learning may require huge amounts of learning
    time
Multi-layer Perceptron

- Can solve ‘XOR’ pattern learning
- No connection within a layer
- No direct connection input and output layers
- Fully connected between layers
For each pattern in the training set:
- Compute the error at the output nodes
- Compute $\Delta w$ for each $w_t$ in 2$^{\text{nd}}$ layer
- Compute $\delta$ (generalized error expression) for hidden units
- Compute $\Delta w$ for each $w_t$ in 1$^{\text{st}}$ layer

After amassing $\Delta w$ for all weights and all patterns, change each $w_t$ a little bit, as determined by the learning rate.
Backpropagation

• BP learns the weights for a multilayer network, given a network with a fixed set of units and interconnections.

• BP employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.

• Two stage learning
  • forward stage: calculate outputs given input pattern $x$.
  • backward stage: update weights by calculating delta.
Differentiable Threshold Unit

\[ \begin{align*}
x_0 &= 1 \\
\text{net} &= \sum_{i=0}^{n} w_i x_i \\
o &= \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \\
\end{align*} \]

\[ \sigma(x) \text{ is the sigmoid function} \]

\[ \frac{1}{1 + e^{-x}} \]

Nice property: \( \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \)

- Sigmoid function: nonlinear, differentiable
Pattern Separation and NN architecture

<table>
<thead>
<tr>
<th>Structure</th>
<th>Type of Decision Regions</th>
<th>Exclusive-Or Problem</th>
<th>Classes with Meshed Regions</th>
<th>Most General Region Shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-layer</td>
<td>Half plane bounded by hyperplane</td>
<td><img src="#" alt="A" /> <img src="#" alt="B" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
</tr>
<tr>
<td>Two-layer</td>
<td>Convex open or closed regions</td>
<td><img src="#" alt="A" /> <img src="#" alt="B" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
</tr>
<tr>
<td>Three-layer</td>
<td>Arbitrary (Complexity limited by number of nodes)</td>
<td><img src="#" alt="A" /> <img src="#" alt="B" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
<td><img src="#" alt="B" /> <img src="#" alt="A" /></td>
</tr>
</tbody>
</table>
960 x 3 x 4 network is trained on gray-level images of faces to predict whether a person is looking to their left, right, ahead, or up.