3. Neural networks - Connectionism

- a totally different approach to machine learning
- applicable in many concept learning contexts, particularly when applications deal with large amount of real numbers (e.g. signal processing)
- motivated by neurology

A neuron:

- A neuron consists of a cell body, one axon, and many dendrites.
- Dendrites receive inputs from axons of other neurons via excitation or inhibition synapses. Real neurons may have many more dendrites.

**dendrites, axons, synapses**

**all-or-nothing device**
Artificial neuron

A simulated neuron. Inputs from other neurons are multiplied by weights, and then are added together. The sum is then compared with a threshold $t$. If the sum is above the threshold, the output is 1; otherwise, the output is 0.

Feed-forward Artificial Neural Networks

A neural net that recognizes siblings and acquaintances. All but the two indicated weights are 1.0. Thresholds are indicated inside the nodes.

Recognizing family relationships from examples
- Top 3 are siblings, as are bottom 3.
- Any 2 that are not siblings are acquaintances.
- Examples (pairs) are specified by setting their inputs to 1. All other inputs at that time are 0.
- Weights are 1 if not specified.

Analyze 3 effects of $H1$, $H2$ firing on Acquaintances, Siblings.
- Weight setting is crucial for correct learning.
• weight setting by hill climbing: given a training example (known output value), make small change to each weight. Measure effect of change for the difference between the output value obtained and the known, expected output value.

• try ea. possible step, choosing the one that decreases the difference by most. Impractical.

• back propagation – backprop – is an efficient procedure to compute how much performance of the net improves in what direction. It propagates the change back through the net.

• no need to search weights, thresholds separately:

• better have a smooth threshold than a step function:
gradient ascent

• idea: in order to change the value of a function $y$ of several variables $x_i$ as fast as possible, change ea. initial $x_i$ in proportion to the partial derivative of $y$ wrt that $x_i$

• suppose $y$ is a function of several vars $x_i$, and ea. $x_i$ is a function of one var. $z$. Then

$$\frac{dy}{dz} = \sum_i \frac{\partial y}{\partial x_i} \frac{dx_i}{dz} = \sum_i \frac{dx_i}{dz} \frac{\partial y}{\partial x_i}$$

(the chain rule)
let

- P is the measured performance
- s - index ranging over all sample inputs
- z - index over all output nodes
- $d_{sz}$ - desired output on input sample s in node z
- $o_{sz}$ - observed output on input sample s in node z

A common way of measuring P is

$$P = -\sum_s \sum_z (d_{sz} - o_{sz})^2$$

An input sample can be considered separately; index s can be dropped

gradient ascent: calculate the partial derivative of P wrt ea.
weight; change weights in proportion to corresp. partial derivative
consider \( \frac{\partial P}{\partial w_{i \rightarrow j}} \) \((^*)\), \\
where \( o_j \) is the output in j-th node.

we know how a node functions: \( o_j = f(\sum o_{Wi \rightarrow j}) \) \\
where f is the threshold function.

let \( \sigma = \sum_{i} o_{Wi \rightarrow j} \) be an intermediate variable.

Then we get for part of \((^*)\) and back into \((^*)\) we get \\
\[
\frac{\partial P}{\partial o_j} = \frac{\partial P}{\partial o_k} \frac{\partial o_k}{\partial o_j} = \sum_k \frac{\partial P}{\partial o_k} \frac{\partial o_k}{\partial o_j}
\]

Now a key fact: partial deriv. of P wrt \( o_j \) can be represented in terms of the partial deriv. of P wrt next level to the right:

\[
(**) \quad \frac{\partial P}{\partial o_j} = \sum_k \frac{\partial P}{\partial o_k} \frac{\partial o_k}{\partial o_j}
\]

but we know that \( o_k = f(\sum_o o_{Wi \rightarrow k}) \) and again \\
denoting the sum as \( \sigma_k \), we get \\
\[
\frac{\partial o_k}{\partial o_j} = \frac{df(\sigma_k)}{d\sigma_k} \frac{\partial o_k}{\partial o_j} = \frac{df(\sigma_k)}{d\sigma_k} W_{ij} \rightarrow k = W_j \rightarrow k \frac{df(\sigma_k)}{d\sigma_k}
\]

plugging this back into \((***)\) we get \\
\[
\frac{\partial P}{\partial o_j} = \sum_k W_{ij} \rightarrow k \frac{df(\sigma_k)}{d\sigma_k} \frac{\partial P}{\partial o_k}
\]
• Develop \( \frac{df(\sigma)}{d\sigma} \)
• Select \( f(\sigma) \) so that it is smooth
• We choose \( f(\sigma) = \frac{1}{1-e^{\sigma}} \)
• Now, \( \frac{df(\sigma)}{d\sigma} = f'(\sigma)(1-f(\sigma)) \)

which shows that for performance \( P \)
• partial deriv. wrt a weight depends on the partial deriv. of the following output
• partial deriv. wrt an output depends on the partial derivs. of performance wrt outputs in the next layer.

To train a network, choose a change rate \( r \), and then
– until performance satisfactory
– for ea sample input, compute the corresp. output
– compute \( \beta_i = d_i - a_i \)
– compute for all other nodes \( j \) \( \beta_j = \alpha_j - \alpha_k \)
– compute weight changes for all weights:
– change the weights \( \Delta w_i \rightarrow j = r \alpha_i \alpha_j (1 - \alpha_j) \beta_j \)
practically, 0 and 1 on output are never achieved, as the net is trained proportionally to output errors. In practice, outputs > .9 and < .1 are acceptable.

rate has to be chosen large enough so that quick learning is possible, but not as large as to overshoot local maxima.

for the acquaintances/siblings example, 225 weight changes are necessary for the net to perform satisfactorily. But as there are 15 inputs, 15*225 training runs are necessary!

<table>
<thead>
<tr>
<th>Weight</th>
<th>Initial value</th>
<th>End of first task</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>0.1</td>
<td>1.99</td>
</tr>
<tr>
<td>WRobert→H1</td>
<td>0.2</td>
<td>4.65</td>
</tr>
<tr>
<td>WAcquaint→H1</td>
<td>0.3</td>
<td>4.65</td>
</tr>
<tr>
<td>WJen→H1</td>
<td>0.4</td>
<td>4.65</td>
</tr>
<tr>
<td>W2</td>
<td>0.5</td>
<td>2.28</td>
</tr>
<tr>
<td>WJohn→H2</td>
<td>0.6</td>
<td>5.28</td>
</tr>
<tr>
<td>WJames→H2</td>
<td>0.7</td>
<td>5.28</td>
</tr>
<tr>
<td>WJuliet→H2</td>
<td>0.8</td>
<td>5.28</td>
</tr>
<tr>
<td>Acquaintances</td>
<td>0.9</td>
<td>9.07</td>
</tr>
<tr>
<td>W1→Acquaintances</td>
<td>1.0</td>
<td>6.27</td>
</tr>
<tr>
<td>W2→Acquaintances</td>
<td>1.1</td>
<td>6.12</td>
</tr>
</tbody>
</table>
Perceptron
a simple, one node net with no hidden units

\[ P = \begin{cases} 1 & \text{if } \sum w_i x_i > T \\ 0 & \text{otherwise} \end{cases} \]

perceptrons can only recognize linearly separable concepts

perceptron training can be solved in poly. time, but...

Theoretical results for NNs

- the loading (finding weights for a fixed network consistent with a training set) problem is NP complete
- backprop is actually a greedy heuristic
- practically, for NNs use \(10^*\text{VC} \) training instances

\[ 2N - \frac{N_i}{2} \leq \text{VC} \leq 2N \log(eN) \]
• but the main problem with NNs is their lack of expressive power (the learned network is often difficult to interpret in domain terms). They cannot be used as readable, user-verifiable concept definitions.
  • large training sets are needed
  • training times are lengthy: 100 to 1000 times longer to train than C4.5 on the same task
  • sensitive to the init. weight assignment
  • no criteria to choose network topology
  • …

NN - application

• ALVINN: Driving a car
• image of the road is converted into a steering direction - a classification problem
• a NN is used to train the classifier
FIGURE 4.1
FIGURE 4.1
Neural network learning to steer an autonomous vehicle. The ALVINN system uses backpropagation to learn to steer an autonomous vehicle (photo at top) driving at speeds up to 70 miles per hour. The diagram on the left shows how the image of a forward-mounted camera is mapped to 900 neural network inputs, which are fed forward to 4 hidden units, connected to 30 output units. Network outputs encode the commanded steering direction. The figure on the right shows weight values for one of the hidden units in this network. The $30 \times 32$ weights into the hidden unit are displayed in the large matrix, with white blocks indicating positive and black indicating negative weights. The weights from this hidden unit to the 30 output units are depicted by the smaller rectangular block directly above the large block. As can be seen from these output weights, activation of this particular hidden unit encourages a turn toward the left.