Digital representation of a letter

Optical character recognition: Predict meaning from features. E.g., given features $x$, what is the character $y$

$$f : x \in S_1^n \rightarrow y \in S_2^m$$
Examples given by lookup table

**Boolean AND function**

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

**Look-up table for a non-boolean example function**

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>5</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The population node as perceptron

**Update rule:** $r^{\text{out}} = g(wr^{\text{in}})$ (component-wise: $r^{\text{out}}_i = g(\sum_j w_{ij} r^{\text{in}}_j)$)

For example: $r^{\text{in}}_i = x_i, \tilde{y} = r^{\text{out}},$ linear grain function $g(x) = x$:

$$\tilde{y} = w_1 x_1 + w_2 x_2$$
How to find the right weight values?

**Objective (error) function**, for example: mean square error (MSE)

\[ E = \frac{1}{2} \sum_i (r_{i}^{\text{out}} - y_i)^2 \]

**Gradient descent** method: \( w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}} \)

\[ = w_{ij} - \epsilon (y_i - r_{i}^{\text{out}}) r_{j}^{\text{in}} \]

for MSE, linear gain

---

Initialize weights arbitrarily

Repeat until error is sufficiently small

Apply a sample pattern to the input nodes: \( r_{i}^{0} = r_{i}^{\text{in}} = \xi_{i}^{\text{in}} \)

Calculate rate of the output nodes: \( r_{i}^{\text{out}} = g(\sum_j w_{ij} r_{j}^{\text{in}}) \)

Compute the delta term for the output layer: \( \delta_i = g'(h_i^{\text{out}})(\xi_i^{\text{out}} - r_i^{\text{out}}) \)

Update the weight matrix by adding the term: \( \Delta w_{ij} = \epsilon \delta_i r_{j}^{\text{in}} \)
Example: OCR

A. Training pattern

```
>> displayLetter(1)
+++    
+++    
+++++  
++ ++   
++   ++  
+++   +++
... activation function
```

B. Learning curve

- Average number of wrong bits
- Training step
- Fraction of flipped bits

C. Generalization ability

- Threshold activation function
- Max activation function
- Average number of wrong letters

0 0.1 0.2 0.3 0.4 0.5
0 5 10 15 20 25
Example: Boolean function

A. Boolean OR function

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

$w_1 x_1 + w_2 x_2 = \Theta$

B. Boolean XOR function

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

$\overline{x}_1$

$\overline{x}_2$
%% Letter recognition with threshold perceptron

clear; clf;

nIn=12*13; nOut=26;
wOut=rand(nOut,nIn)-0.5;

% training vectors
load pattern1;
rIn=reshape(pattern1', nIn, 26);
rDes=diag(ones(1,26));

% Updating and training network
for training_step=1:20;
  % test all pattern
  rOut=(wOut*rIn)>0.5;
  distH=sum(sum((rDes-rOut).^2))/26;
  error(training_step)=distH;
  % training with delta rule
  wOut=wOut+0.1*(rDes-rOut)*rIn';
end

plot(0:19,error)
xlabel('Training step')
ylabel('Average Hamming distance')
The multilayer Perceptron (MLP)

Update rule: \( r^{out} = g^{out}(w^{out} g^{h}(w^{h} r^{in})) \)

Learning rule (error backpropagation): \( w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}} \)
The error-backpropagation algorithm

<table>
<thead>
<tr>
<th>Initialize weights arbitrarily</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat until error is sufficiently small</td>
</tr>
<tr>
<td>Apply a sample pattern to the input nodes: ( r_i^0 := r_i^{\text{in}} = \xi_i^{\text{in}} )</td>
</tr>
<tr>
<td>Propagate input through the network by calculating the rates of nodes in successive layers ( l ): ( r_i^l = g(h_i^l) = g(\sum_j w_{ij}^l r_{j}^{l-1}) )</td>
</tr>
<tr>
<td>Compute the delta term for the output layer: ( \delta_i^{\text{out}} = g'(h_i^{\text{out}})(\xi_i^{\text{out}} - r_i^{\text{out}}) )</td>
</tr>
<tr>
<td>Back-propagate delta terms through the network: ( \delta_{i}^{l-1} = g'(h_i^{l-1}) \sum_j w_{ji}^l \delta_j^l )</td>
</tr>
<tr>
<td>Update weight matrix by adding the term: ( \Delta w_{ij}^l = \epsilon \delta_i^l r_{j}^{l-1} )</td>
</tr>
</tbody>
</table>
% MLP with backpropagation learning on XOR problem
clear; clf;
N_i=2; N_h=2; N_o=1;
w_h=rand(N_h,N_i)-0.5; w_o=rand(N_o,N_h)-0.5;

% training vectors (XOR)
r_i=[0 1 0 1; 0 0 1 1];
r_d=[0 1 1 0];

% Updating and training network with sigmoid activation function
for sweep=1:10000;
    % training randomly on one pattern
    i=ceil(4*rand);
    r_h=1./(1+exp(-w_h*r_i(:,i)));
    r_o=1./(1+exp(-w_o*r_h));
    d_o=(r_o.*(1-r_o)).*(r_d(:,i)-r_o);
    d_h=(r_h.*(1-r_h)).*(w_o'*d_o);
    w_o=w_o+0.7*(r_h*d_o')';
    w_h=w_h+0.7*(r_i(:,i)*d_h')';
% test all pattern
    r_o_test=1./(1+exp(-w_o*(1./(1+exp(-w_h*r_i)))));
    d(sweep)=0.5*sum((r_o_test-r_d).^2);
end
plot(d)
MLP for XOR function

Learning curve for XOR problem
MLP approximating sine function
Overfitting and underfitting

Regularization, for example

\[ E = \frac{1}{2} \sum_i (r_i^{\text{out}} - y_i)^2 - \gamma r \frac{1}{2} \sum_i w_i^2 \]
Support Vector Machines

Linear large-margin classifier

\[ x_1 \]

\[ x_2 \]
SVM: Kernel trick

A. Linear not separable case

B. Linear separable case
Further Readings


Christopher M. Bishop (2006), **Pattern Recognition and Machine Learning**, Springer


Alex J. Smola and Bernhard Schölkopf (2004), **A tutorial on support vector regression** in *Statistics and computing* 14: 199-222.

