Teuvo Kohonen

The Self-Organizing Map
Proceedings of the IEEE, vol.78, no.9, 1990

Presented by Joseph Chang
Neural Network Models

1. Feedforward networks

2. Feedback networks

3. Competitive / unsupervised / self-organizing
   - Nodes are different pattern detectors
   - Compete for input in training
• Localization of brain functions
• Related functions are together
• “Connected” nodes are similar
• Nodes as pattern detectors
Basic Competitive Learning

input : $x = x(t) \in \mathbb{R}^n$

model : $\{m_i(t) : m_i \in \mathbb{R}^n, i = 1...k\}$

distance : $d(x, m_i)$

$m_c = \text{argmin}_{m_i} d(x, m_i)$

$E = \int d(x, m_c)^r p(x) \, dx$
Competitive Learning

input: \( x = x(t) \in \mathbb{R}^n \)

model: \( \{ m_i(t) : m_i \in \mathbb{R}^n, i = 1...k \} \)

distance: \( d(x, m_i) \)

\( m_c = \arg\min_{m_i} d(x, m_i) \)

\( E = \int d(x, m_c)^r \ p(x) \ dx \)
Basic Competitive Learning

input: $x = x(t) \in \mathbb{R}^n$

model: $\{m_i(t): m_i \in \mathbb{R}^n, i = 1...k\}$

distance: $d(x, m_i)$

$m_c = \text{argmin}_{m_i} d(x, m_i)$

$E = \int d(x, m_c)^r p(x) \, dx$
Basic Competitive Learning

n=1, k =7

$p(x)$

input : $x = x(t) \in \mathbb{R}^n$

model : $\{m_i(t) : m_i \in \mathbb{R}^n, i = 1...k\}$

distance : $d(x, m_i)$

$m_c = \arg\min_{m_i} d(x, m_i)$

$E = \int d(x, m_c)^r p(x) \, dx$
Delta Rule Learning

- For each input, only update the best matching node $m_c$
- Monotonically decreasing learning rate

\[
m_c(t + 1) = m_c(t) + \alpha(t) [x(t) - m_c(t)]
\]

\[
m_i(t + 1) = m_i(t), \text{ for } i \neq c
\]

\[
0 < \alpha(t) < 1, \text{ monotonic decreasing}
\]
Competitive Learning

\[ m_c(t + 1) = m_c(t) + \alpha(t) \left[ x(t) - m_c(t) \right] \]

\[ m_i(t + 1) = m_i(t), \text{ for } i \neq c \]

\[ 0 < \alpha(t) < 1, \text{ monotonic decreasing} \]
K-Means

- Assign all input $x(0)\ldots x(t)$ to their best matching $m_i$
- Update $m_c$ to the mean of its inputs $x_j$

$$m_i(t + 1) = \frac{1}{n} \sum_{j=1}^{n} x_j$$
Self-Organizing Map

- Nodes are connected using pre-defined topology
- All nodes receive all input signals $x(t) \in \mathbb{R}^n$
- Also updates neighbors of $m_c: N_c(t)$

\[
m_i(t + 1) = \begin{cases} 
m_i(t) + \alpha(t) [x(t) - m_i(t)] & \text{for } i \in N_c(t) \\
m_i(t) & \text{for } i \notin N_c(t) \end{cases}
\]
Neighboring Function

• Based on time and the topology of nodes
• Advantageous to be wide initially, and shrink monotonically
• Special case: $N_c = \{c\}$
n=2, grid topology
\[ m_i(t + 1) = m_i(t) + \alpha(t) \left[ x(t) - m_i(t) \right] \text{ for } i \in N_c(t) \]

\[ m_i(t + 1) = m_i(t) \quad \text{for } i \notin N_c(t) \]
\[ m_i(t + 1) = m_i(t) + \alpha(t) \left[ x(t) - m_i(t) \right] \quad \text{for } i \in N_c(t) \]

\[ m_i(t + 1) = m_i(t) \quad \text{for } i \notin N_c(t) \]
\[ m_i(t + 1) = m_i(t) + \alpha(t) \left[ x(t) - m_i(t) \right] \quad \text{for } i \in N_c(t) \]

\[ m_i(t + 1) = m_i(t) \quad \text{for } i \notin N_c(t) \]
\[ m_i(t + 1) = m_i(t) + \alpha(t) \left[ x(t) - m_i(t) \right] \text{ for } i \in N_c(t) \]

\[ m_i(t + 1) = m_i(t) \quad \text{ for } i \notin N_c(t) \]
n=2, grid topology
n=2, linear topology
n=3, grid topology

Initial model

Input

Self-organized model
Basic Competitive Learning

$n=1, \ k=7$

\[ p(x) \]

- **input**: \( x = x(t) \in R^n \)
- **model**: \( \{m_i(t) : m_i \in R^n, i = 1...k\} \)
- **distance**: \( d(x, m_i) \)
- \( m_c = \text{argmin}_{m_i} d(x, m_i) \)
- \( E = \int d(x, m_c)^r \ p(x) \ dx \)
n=3 (rgb), grid topology
Some Practical Hints

- Initialized \( m_i \) randomly

- \( t \) should be larger than \( k \): \( t > 500 \times k \)

- Starts \( \alpha(t) \) with something close to 1.0: \( \alpha(t) = 0.9(1-t/1000) \)

- Initial neighbor should be large, up to more than half the network

- Final stage \( N_c = \{c\} \) for fine tuning

- If the neighborhood is too small to start with, the map will not be ordered globally. Instead various kinds of mosaic-like phenomena of the map are seen, between which the ordering direction changes discontinuously.
Learning Vector Quantization (LVQ)

- Using SOMs as classifiers
- An additional set of labeled input data $x(t)$
  1. Use voting to assign categories to nodes
  2. Update nodes $m_c$ according to $x(t)$
LVQ1

1. Use voting to assign categories to nodes

2. Update nodes $m_i$ according to $x(t)$

\[
\text{if } x \text{ is classified correctly} \\
\quad m_c(t + 1) = m_c(t) + \alpha(t) \left[ x(t) - m_c(t) \right]
\]

\[
\text{if } x \text{ is not classified correctly} \\
\quad m_c(t + 1) = m_c(t) - \alpha(t) \left[ x(t) - m_c(t) \right]
\]

\[
\text{if } i \neq c \\
\quad m_i(t + 1) = m_i(t)
\]

$\alpha(t) : 0.01 \text{ or } 0.02 \rightarrow 0.0$
LVQ2

- Optimal boundary
- Decision boundary

p(C_i)

m_i

p(C_j)

m_j

Window

x
LVQ2

- Input \( x(t) \) belongs to class \( C_j \)
- Only update when
  1. Input \( x(t) \) is incorrectly classified to \( m_i / C_i \)
  2. Node \( m_j / C_j \) is the “runner-up” node / class
  3. \( x(t) \) is near the current decision boundary
- Update both \( m_i \) and \( m_j \)
LVQ2

- Input $x(t)$ belongs to class $C_j$
- $x(t)$ is incorrectly classified to $m_i / C_i$
- $m_j / C_j$ is the "runner-up" class / node
- $x(t)$ is in the "window"

$$m_i(t + 1) = m_i(t) - \alpha(t) \ [x(t) - m_i(t)]$$

$$m_j(t + 1) = m_j(t) + \alpha(t) \ [x(t) - m_j(t)]$$

- else

$$m_k(t + 1) = m_k(t)$$
LVQ2

optimal boundary

decision boundary

\( p(C_i) \)

\( m_i \)

window

\( m_j \)

\( p(C_j) \)

\( x \)
Window size

- Trade off between accuracy and sample count
- With larger training set, use smaller window
- E.g., 10% ~ 20% of the distance between \( m_i \) and \( m_j \), for small training set
Problems with LVQ2

optimal boundary

decision boundary

window

$m_i$

$m_j$
LVQ3

optimal boundary

decision boundary

window

$m_i$

$m_j$
LVQ3

- Input $x(t)$ belongs to class $C_j$
- $m_i$ and $m_j$ are the two closest nodes to $x(t)$
- $x(t)$ in “window”
- if $m_j$ belongs to $C_j$ and $m_i$ does not belong to $C_j$
  
  $$m_i(t + 1) = m_i(t) - \alpha(t) \left[ x(t) - m_i(t) \right]$$
  $$m_j(t + 1) = m_i(t) + \alpha(t) \left[ x(t) - m_j(t) \right]$$

- if $m_i, m_j$ both belong to $C_j$
  
  $$m_i(t + 1) = m_i(t) + \epsilon \alpha(t) \left[ x(t) - m_i(t) \right]$$
  $$m_j(t + 1) = m_j(t) + \epsilon \alpha(t) \left[ x(t) - m_j(t) \right] \quad \epsilon : 0.1 \sim 0.5$$
LVQ3

optimal boundary  decision boundary

$m_i$  window  $m_j$

$x$
Application 1: Phoneme Recognition

- Task: Recognize phonemes from unlabeled recordings
- 9.83ms Hamming window
- Finnish phonemes: /u, o, a, ø, y, e, i, s, m, n, ð, l, r, j, v, h, d, k, p, t/
• Finnish phonemes: /u, o, a, æ, ø, y, e, i, s, m, n, η, l, r, j, v, h, d, k, p, t/
Application 1: Phoneme Recognition

- 1550 training vectors
- 1550 testing vectors
- 2 different speakers
- Error rate:

<table>
<thead>
<tr>
<th></th>
<th>Parametric Bayes</th>
<th>kNN</th>
<th>LVQ1</th>
<th>LVQ2</th>
<th>LVQ3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>12.1</td>
<td>12.0</td>
<td>10.2</td>
<td>9.8</td>
<td>9.6</td>
</tr>
<tr>
<td>Test 2</td>
<td>13.8</td>
<td>12.1</td>
<td>13.2</td>
<td>12.0</td>
<td>11.5</td>
</tr>
</tbody>
</table>
Application 2: Semantic Map

- Task: Organize a map of words based on its semantic meanings
- Input: Simple language generator
Language Generation

- 498 three word sentences concatenated as one string $S$

<table>
<thead>
<tr>
<th>Semantic Word Groups</th>
<th>Rules</th>
<th>Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob/Jim/Mary 1</td>
<td>1-5-12 1-9-2 2-5-14</td>
<td>Mary likes meat</td>
</tr>
<tr>
<td>horse/dog/cat 2</td>
<td>1-5-13 1-9-3 2-9-1</td>
<td>Jim speaks well</td>
</tr>
<tr>
<td>beer/water 3</td>
<td>1-5-14 1-9-4 2-9-2</td>
<td>Mary likes Jim</td>
</tr>
<tr>
<td>meat/bread 4</td>
<td>1-6-12 1-10-3 2-9-3</td>
<td>Jim eats often</td>
</tr>
<tr>
<td>runs/walks 5</td>
<td>1-6-13 1-11-4 2-9-4</td>
<td>Mary buys meat</td>
</tr>
<tr>
<td>works/speaks 6</td>
<td>1-6-14 1-10-12 2-10-3</td>
<td>dog drinks fast</td>
</tr>
<tr>
<td>visits/phones 7</td>
<td>1-6-15 1-10-13 2-10-12</td>
<td>horse hates meat</td>
</tr>
<tr>
<td>buys/sells 8</td>
<td>1-7-14 1-10-14 2-10-13</td>
<td>Jim eats seldom</td>
</tr>
<tr>
<td>likes/hates 9</td>
<td>1-8-12 1-11-12 2-10-14</td>
<td>Bob buys meat</td>
</tr>
<tr>
<td>drinks/eats 10/11</td>
<td>1-8-2 1-11-13 2-11-4</td>
<td>cat walks slowly</td>
</tr>
<tr>
<td>much/little 12</td>
<td>1-8-3 1-11-14 2-11-12</td>
<td>Jim eats bread</td>
</tr>
<tr>
<td>fast/slowly 13</td>
<td>1-8-4 2-5-12 2-11-13</td>
<td>cat hates Jim</td>
</tr>
<tr>
<td>often/seldom 14</td>
<td>1-9-1 2-5-13 2-11-14</td>
<td>Bob sells beer</td>
</tr>
<tr>
<td>well/poorly 15</td>
<td>(etc.)</td>
<td>(etc.)</td>
</tr>
</tbody>
</table>
Input

\[ x(t) = [x_s \ x_c]^T = [x_s \ 0]^T + [0 \ x_c]^T \]

- Words are represented by 7D unit length vectors
- Context is represented by 14 D unit length vectors
  - predecessor/successor word vectors concatenated
- Average context vector from \( S \) for every word
- Final input: thirty 7D+14D vectors, weighted by \( 0.2 : 1.0 \)
Results

\[ m_c \text{ for } x = [x_s \ 0]^T \]
Recap

• Basic competitive learning
  • Nodes compete for input. Update independently.

• Self-organizing Map
  • Nodes compete for input. Update according to node topology.

• Learning Vector Quantization
  • LVQ1: Update one node per input.
  • LVQ2: Update two nodes base on marginal misclassified inputs.
  • LVQ3: Update two nodes base on marginal and correct (ε) inputs.
References
