6. Feed-forward mapping networks


Lecture Notes on Brain and Computation

Byoung-Tak Zhang

Biointelligence Laboratory
School of Computer Science and Engineering
Graduate Programs in Cognitive Science, Brain Science and Bioinformatics
Brain-Mind-Behavior Concentration Program
Seoul National University

E-mail: btzhang@bi.snu.ac.kr
This material is available online at http://bi.snu.ac.kr/
Outline

6.1 Perception, function representation, and look-up tables
6.2 The sigma node as perceptron
6.3 Multilayer mapping networks
6.4 Learning, generalization, ad biological interpretations
6.5 Self-organizing network architectures and genetic algorithms
6.6 Mapping networks with context units
6.7 Probabilistic mapping networks
6.1 Perception, function representation, and look-up tables

6.1.1 Optical character recognition (OCR)

- To illustrate the abilities of the networks
- Optical character recognition
  - Letter
  - Spell-checking
  - Scan a handwritten page into the computer
    - Difficult task
- Two major components in the perception of the letter
  - The ‘seeing’
  - Attaching a meaning to such an image
6.1.2 Scanning with a simple model retina

- Recognizing the letter ‘A’
- A simplified digitizing model retina of only 10 x 10 = 100 photoreceptors
- A crude approximation of a human eye
- Simply intended to illustrate a general scheme

6.1. (Left) A printed version of the capital letter A and (right) a binary version of the same letter using a 10 x 10 grid.
6.1.3 Sensory feature vectors

- Sensory feature vectors
  - We give each model neuron an individual number and write the value of this neuron into a large column at a position corresponding to this number of node.

Fig 6.2 Generation of a sensory feature vector. Each field of the model retina, which corresponds to the receptive field of a model neuron, is sequentially numbered. The firing value of each retinal node, either 0 or 1 depending on the image, represents the value of the component in the feature value corresponding to the number of the retinal node.
6.1.4 Mapping function

- A sensory feature vector is the necessary input to any object recognition system
- Mapping to *internal representation*
  - Ex) ASCII code
- Recognize a letter
  - Internal object vector with a single variable (1-D vector)
- The recognition process to a vector function
  - Mapping $f : \mathbf{x} \in S_1^n \rightarrow \mathbf{y} \in S_2^m$ (6.1)
  - A vector function $f$ from a vector $\mathbf{x}$ to another vector $\mathbf{y}$ as where $n$ is the dimensionality of the sensory feature space. and $m$ is dimensionality of the internal object representation space
  - $S_1$ and $S_2$ are the set of possible values for each individual component of the vector
6.1.5 Look-up tables

- How can we realize a mapping function?
- Look-up table
  - Lists for all possible sensory input vectors the corresponding internal representations

<table>
<thead>
<tr>
<th>Table 6.1</th>
<th>Look-up table for a specific binary mapping function (Boolean AND function) in a two-dimensional feature space. (B) Partial look-up table for a sample function in a two-dimensional feature space with discrete but not binary feature values.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Boolean AND function</strong></td>
<td><strong>B. Non-Boolean function</strong></td>
</tr>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
6.1.6 Prototypes

- Another possibility for realizing a mapping function
  - Prototypes
    - A vector that encapsulates, on average, the features for each individual object
- How to generate the prototype vectors
  - To present a set of letters to the system and to use the average as a prototype for each individual letter
  - Learning system
- Disadvantage of the prototype scheme
  - The time for recognition might exceed reasonable times in problems with a large set of possible objects
6.2 The sigma node as perceptron

- A simple neuron (sigma node)
  - Represent certain types of vector functions
- Setting the firing rate of the related input channels to $r_{i}^{in} = x_{i}$ (6.2)
- The firing rate of the output defines a function $\tilde{y} = r^{out}$ (6.3)
- The output of such a linear perceptron is calculated from the formula $\tilde{y} = w_{1}x_{1} + w_{2}x_{2}$ (6.4)

Fig 6.3 Simple sigma node with two input channels as a model perceptron for a 2-D feature space
6.2.1 An example of mapping function

- The function listed partially in the look-up table in Table 6.1B
- \( w_1 = 1, \ w_2 = -1, \)

\[
\tilde{y}^1 = \tilde{y}(x^1) = \tilde{y}(x_1 = 1, x_2 = 2) = 1 \cdot 1 - 1 \cdot 2 = -1 = y^1 \quad (6.5)
\]

\[
\tilde{y}^2 = 1 \cdot 2 - 1 \cdot 1 = 1 = y^2 \quad (6.6)
\]

\[
\tilde{y}^3 = 1 \cdot 3 - 1 \cdot (-2) = 5 = y^3 \quad (6.7)
\]

Fig 6.4 Output manifold of sigma node with two input channels that is able to partially represent the mapping function listed in the look-up table 6.1B.
6.2.2 Boolean functions

- **Binary functions or Boolean functions**

Fig 6.5 (A) Look-up table, graphical representation, and single threshold sigma node for the Boolean OR function. (B) Look-up table and graphical representation of the Boolean XOR function, which cannot be represented by a single threshold sigma node because this function is not linear separable. A node that can rotate the input space and has a non-monotonic activation function can, however, represent this Boolean function.
6.2.3 Single-layer mapping networks

- The functionality of a single output node generalizes directly to networks with several output nodes to represent vector functions.

Weight matrix

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n_{out}} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n_{out}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n_{in}}^{1} & w_{n_{in}}^{2} & w_{n_{in}}^{3} & \cdots & w_{n_{in}}^{n_{out}} \end{pmatrix}$$  (6.8)

Single layer mapping network (simple perceptron)

$$\mathbf{r}_{out}^{\text{out}} = g(\mathbf{w}\mathbf{r}_{in}^{\text{in}})$$  (6.9)

- $g$, activation function

$$r_{i_{out}} = g\left(\sum_{j} w_{ij}r_{j}^{\text{in}}\right)$$  (6.10)
6.3 Multilayer mapping networks

- Multilayer mapping network
  - Hidden layer
  - The back-propagation algorithms
- The number of weight values, \(n^w\)
- The number of \(n^h\) in the hidden layer
  \[
  n^w = n^{in} n^h + n^h n^{out} \tag{6.11}
  \]
- \(n^{in}\) is the number of input nodes
- \(n^{out}\) is the number of output nodes

Fig 6.6 The standard architecture of a feed-forward multilayer network with one hidden layer, in which input values are distributed to all hidden nodes with weighting factors summarized in the weight matrix \(w^h\). The output values of the nodes of the hidden layer are passed to the output layer, again scaled by the values of the connection strength as specified by the elements in the weight matrix \(w^{out}\). The parameters shown at the top, \(n^{in}, n^h,\) and \(n^{out}\), specify the number of nodes in each layer, respectively.
6.3.1 The update rule for multilayer mapping networks

- $w^h$, the weight to the hidden layer
- A matrix-vector multiplication $h^h = w^h r^in$ (6.12)
- $h^h = \sum_j w^h_{ij} r^in_j$ (6.13)
- $h^h$, activation vector of the hidden nodes
- The firing rate of the hidden layer $r^h = g^h (h^in)$ (6.14)
- The final output vector $r^out = g^out (w^out r^h)$ (6.15)
- All the steps of the multilayer feed-forward network
  $r^out = g^out (w^out g^h (w^h r^in))$ (6.16)
- Ex) 4-layer network with 3 hidden layers and 1 output layer
  $r^out = g^out (w^out g^h3 (w^h3 g^h2 (w^h2 g^h1 (w^h1 r^in))))$ (6.17)
- Linear activation function ($g(x)=x$)
  $r^out = g^out (w^out w^h3 w^h2 w^h1 r^in) = g^out (\tilde{w} r^in)$ (6.18)
6.3.2 Universal function approximation

- A multilayer feed-forward network is a universal function approximator.
  - They are not limited to linear separable functions
  - The number of free parameters is not restricted in principle
- How may hidden nodes we need?
- Activation function

![Diagram of XOR function](image1)

![Diagram of sine function approximation](image2)

Fig 6.7 One possible representation of the XOR function by a multilayer network with two hidden nodes. The numbers in the nodes specify the firing threshold of each node.

Fig 6.8 Approximation (dashed line) of a sine function (solid line) by the sum of three sigmoid functions shown as dotted lines.
6.4 Learning, generalization, and biological interpretations

6.4.1 Adaptation and learning

- Multilayer networks representing arbitrarily close approximations of any function by properly choosing values for the weights between nodes.
  - How can we choose proper values?
- Adaptation, the process of changing the weight values to represent the examples
- Learning or training algorithms, the adaptation algorithms
- Adjust the weight values in abstract neural networks
  - Weight values = the synaptic efficiency
- Represent developmental organizations of the nervous system
6.4.2 Information geometry

- Information geometry
- Network-manifold
- Solution subspace
- Solution manifold
- Statistical learning rules
- Random search
- Gradient descent method
  - Error function
  - Supervised learning algorithms

Fig. 6.9 Schematic illustration of the weight space and learning in multilayer mapping networks.
6.4.3 The biological plausibility of learning algorithms

- Learning algorithms
  - Designed to find solution to the training problem in the solution space
  - Different algorithms might find different solutions
  - Depends on the algorithm employed

- Difficult to relate the weights in a trained network to biological systems

- The training of a multilayer network corresponds to nonlinear regression of data in the statistical sense to specific high-dimensional model
6.4.4 Generalization

- **Generalization ability**
  - The performance of the network on data that were not part of the training set

- **To prevent over-fitting**
  - Regularization or use only small number of hidden nodes

Fig 6.10 Example of over-fitting of noisy data. Training points were generated from the ‘true’ function \( f(x) = 1 - \tanh(x-1) \), and noise was added to these training points. A small network can represent the ‘true’ function correctly. The function represented by a large network that fits all the training points is plotted with a dashed line.
6.4.5 Caution in applying mapping networks as brain models

- The network approximation of particular problem depends strongly on the number of hidden nodes and the number of layers
- Limit the number of hidden nodes to enable better generalization performance
- But, the number of hidden nodes in biological systems is often very large.
  - The biological plausibility of learning algorithms
- We have to be very careful in interpreting the simplified neural networks as brain models
6.5 Self-organizing network architectures and genetic algorithms

6.5.1 Design algorithms

- The architecture is often crucial for the abilities of such networks

- *Design algorithms*, several algorithms have been proposed to help with the design of networks.
  - A node creation algorithm
  - Pruning algorithms

  - weight decay

$$w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij}(t) - \epsilon^{\text{decay}} w_{ij}(t)$$  (6.19)

---

Fig. 6.11 Example of the performance of a mapping network with a hidden layer that was trained on the task of adding two 3-digit binary numbers. After a fixed amount of training a new hidden node was created.
6.5.2 Genetic algorithms

- A design algorithm that is very relevant in biological systems.
  - Genetic algorithms
    - Evolutionary computing
- Genomes, a large vector with component that are 1 or 0
- Population
- Objective function
- The generation of a new population
  - Genetic operators
    - Survival operator
    - Cross-over
    - Random mutation
- Genomes are optimized to perform a certain task
6.6 Mapping networks with context units
6.6.1 Contextual processing

- Feed-forward mapping networks are powerful function approximators
  - They can map an input to any desired output
- Feed-forward mapping networks have become a standard tool in cognitive science
- The study of cognitive abilities of humans reveals that our behavior, for example, the execution of particular motor actions, often depends on the context in which we encounter a certain situation
- The context of a sensory input can be important
6.6.2 Recurrent mapping networks

- *Simple recurrent network or Elman-net*

Fig. 6.12 Recurrent mapping networks as proposed by Jeffrey Elman consisting of a standard feedforward mapping network with 4 input node, 3 hidden nodes, and 4 output nodes. However, the network also receives internal input from context nodes that are efferent copies of the hidden node activities. The efferent copy is achieved through fixed one-to-one projections from the hidden nodes to the context nodes that can be implemented by fixed weights with some time delay.

- *recurrences*
- *short-term memory*
6.6.3 The effect of recurrences

- The context units
  - to contain the activity (firing rate) of the hidden nodes at the previous time step
  - some delay in the projections

- The network functions
  - The external input from the input nodes and from the context nodes (which memorized the previous firing rate of the hidden nodes)

- To take into account the context during training
  - Train the network on whole sequences of inputs
6.7 Probabilistic mapping networks

- The output of a mapping network can also be interpreted as probabilities
- Probabilistic feed-forward networks
  - The activity of each output node can be interpreted as the probability of membership of the object to the class represented by the node
- Normalize the sum of all output
  \[ \sum_i r_{i}^{out} = 1 \] (6.20)
- Winner-take-all
  - The strong activity of one node inhibits the firing of other nodes

Fig. 6.13 (A) Mapping network with collateral connections in the output layer that can implement competition in the output nodes. (B) Normalization of the output nodes with the softmax activation function can be implemented with an additional layer.
6.7.1 Soft competition

- Class-membership probability
- Soft competition in the output layer
- Softmax function
  \[
  \tilde{r}_i^{out} = \frac{e^{r_i^{out}}}{\sum_j e^{r_j^{out}}}
  \]  
  (6.21)

6.7.2 Cross-entropy as an objective function

- Learning algorithms for the probabilistic interpretation
  - Use cross-entropy
    \[
    E = -\sum_\mu \sum_i t_i^{\mu} \ln (r_i^{in}, W; \mu)
    \]  
    (6.22)
- Training algorithms can be designed to minimize such objective functions
Conclusion

- Networks of simple sigma nodes
  - Feed-forward manner
  - Mapping networks
- Multilayer network
  - A universal approximator
- Learning and adaptation
- Design Algorithm
  - Network architecture
- Probabilistic mapping networks