A Neural Network Simulating System

In this paper, an experimental neural network description language is proposed and with this language a neural network simulation system is also developed. The purpose of this work is to try to find a unifying way of describing neural network models and thus facilitate the design of neural networks.

Received June 1989, revised August 1990

1. Motivation: problems with developing neural nets

Neural networks have been studied for a long time; hundreds of models have been proposed and a great deal of work has been implemented. However, the most popular way to describe a model is through mathematical equations, with some auxiliary statements to make it clear. At some time, we may have to define artificially some action that our neural net should include in natural language. For example, in the Kohonen map model it is necessary to define the time-varying neighborhoods. Furthermore, it is easier to show the relationship of neighbourhood in pictures than in words. Neural network simulation program writers or neurocomputer experimenters have to convert these statements and equations accurately into code and circuit layouts. Some energy may be wasted on repetitive work because some networks share the same features. Besides, this puts a heavy burden on the programmer, and as far as software and hardware development are concerned it may not be economic, because a long time has to be spent in translating a rigid model into machine-level codes (or basic circuit blocks), while this work could be done automatically if the model is carefully presented in some language (of course, in order that the target language can be processed by contemporary 'stupid' computers, the description language should be less ambiguous than natural language but more flexible than traditional programming languages, so that we can conveniently present any equation or definition). With the neural network description language the model-to-product process could be automated, hence the model-design and testing cycle could be shortened and the neural network developer could concentrate more on model designing rather than on other less important details like alias effects in programming languages. It seems that there are only a few languages proposed for neural nets description, perhaps even a standard way, general enough to present various models while regular enough to be processed by current computers.

2. The neural net simulation system description language

For the purpose of facilitating neural networks development and shortening the developing cycle of design-test-revision, we are developing a neural network description language and a neural network simulation system; we shall examine the simulation system first. Because the system is still at its testing stage, and for convenience in system development, we decided to use an interpreter-like internal structure and a quasi-mnemonic language notation. The program receives the user specification from an indicated file or keyboard and translates this specification into an array of function calls. Later on, the system uses this internal representation to simulate the neural network. During the execution process the information needed by the neural network (such as its training patterns, desired outputs, and the result of configuration after one train if the user wishes to see it) will be prompted on the screen to ask the user to key in the appropriate data. This can be a tedious task, and we are now improving this by developing a file-oriented information-retrieving mechanism. The reason for an interpreter-like internal representation of the net is that we can easily modify our simulation system without a great change on the original version. Furthermore, the neural network primitives (the lowest-level function needed) are much more than can be presented with just a few assembly lines. They should be implemented with function calls, and from the developing experience of LISP (a language that consists of a few primitivies, most of which should be implemented through function calls although some of the compilation work may be wasted), we think that it is worth using this simpler and clearer internal structure during the early system development stage even at the cost of efficiency. The obvious defect of the system is that its simulation work may be very inefficient. However, after the system is proved to be successful, we can transform the system into an equivalent compiling system that can translate its input of neural net specification into a destination machine code (or circuit layout suitable for some kinds of integrated circuit) with moderate effort.

In this system we adopt a notation that is not easy to memorise. For example '!' denotes the 'one' function: $! (x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases}$

The reason why we take this notation is that it will save our time by eliminating most of the parsing and synthesis work needed. On the other hand, our main purpose in designing the simulation system and the language is to explore the nature of the specification of the neural nets. We want to know the most appropriate and optimal information in describing a neural net and the lowest-level primitive needed while offering a moderate abstraction. Take the example of building models in blocks (cylces, cylinders, pyramids and so on). We can build many models with these basic blocks. However, when we are constructing a neural net model, what do the basic blocks look like? We may automate the translation from model specification to implementation (in hardware or software) so long as we know the basic building blocks of the various neural net models. Then developing and experimenting with different models should be more convenient.

3. Language and specification

Part 1. The Neural-description language

Specification of function scanner for activation and output

<table>
<thead>
<tr>
<th>Nomenclature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>current processed weight</td>
</tr>
<tr>
<td>$S$</td>
<td>source output for processed connection</td>
</tr>
<tr>
<td>$= \text{current activation}$</td>
<td></td>
</tr>
<tr>
<td>$\text{Note}: n$</td>
<td>$\text{the n slot time before weight data (0 \leq n \leq 4)}$</td>
</tr>
<tr>
<td>$\text{teacher information} (e.g. \text{get nth teacher's corresponding info})$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$\text{source's extra information 1}$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\text{source's extra information 2}$</td>
</tr>
<tr>
<td>$g$</td>
<td>$\text{change indicated group number to become current processing group}$</td>
</tr>
<tr>
<td>$\text{declaration of a control variable}$</td>
<td>$\text{prefix of a control variable in declaration, put its value on result stack otherwise}$</td>
</tr>
<tr>
<td>$\text{beginning of function declaration}$</td>
<td>$\text{end of function declaration}$</td>
</tr>
</tbody>
</table>

The following functions put their result on stack with group source (weight will be automatically multiplied):

- \text{pairwise operation with summed result put on result stack}:
  \begin{align*}
  (1) & \quad \text{e.g. } \sum_i \left[ A[i] \cdot [B] + C[i] \right] \rightarrow ^* \text{B} \cdot [S] \\
  (2) & \quad \text{between the delimiters you can put any meaningful operations you want}$

- \text{integration (e.g. } A \cdot B \text{) put whole group multiplication (P function) with result on stack}$

- \text{summation on a group of neurons}$

The following functions do arithmetic operations on stack (the convention comes from Forth):

- one function
- $\# \phi$ phi function (\text{#}(x) = (\frac{1}{1 + \exp (-(a - \text{theta}))}$)
- \text{sigmoid function} (stack top $\rightarrow$ theta; stack second $\rightarrow$ a; $1/(1+\exp (-(a - \text{theta}))$)
- \text{hard limiter function} (\text{#}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$)
- \text{random number} ($-1.0 < x < 1.0$)
- \text{square root}$
- \text{addition}$
- \text{sub (former-rear)}$
- \text{multiplication}$
- \text{div (former/rear)}$

92 THE COMPUTER JOURNAL, VOL. 34, NO. 1, 1991
Part 2. The simulation system

Name. Neural Network Simulating System.

Designer. S. T. Cheng, Student NTUCSIE.

Language. C on IBM PC/XT with 2-5 acceleration card compiled with large mode.

Purpose. For general-purpose simulation in neural network. Of course, true ‘general purpose’ is not easy, thus, here we only provide a tool for neural network simulation program development. Besides, some of the features of current neural nets have not yet been included.

Internal Structure. The simulation system keeps an internal stack of the user-specified size. When it’s updating neuron connections, calculating neuron states or neuron outputs, it utilizes this stack, using the result on the stack as the calculation result.

4. An example of neural network model description

Generally speaking, a description of a model can be separated into two parts, the functions it uses (updating rules for cell states, weights, and cell outputs) and the cell connection configuration. According to this structure, the neural description language can be divided into three parts. At first, we declare all the variables the model needs (thresholds, special constants) and then the rules and formulas used. Finally, the connections between cells are given. Let’s consider the Hopfield Net as an illustrative example. Keep the stack in mind when you are examining these functions.

Example. Hopfield model (assume that there are five cells in the Hopfield layer X).

Model outline

A single cell looks like this:

\[
\begin{align*}
    x_0 & \rightarrow \begin{pmatrix} \cdot \end{pmatrix} \\
    x_1 & \rightarrow \begin{pmatrix} \cdot \end{pmatrix} \\
    \vdots & \rightarrow \begin{pmatrix} \cdot \end{pmatrix} \\
    x_n & \rightarrow \begin{pmatrix} \cdot \end{pmatrix}
\end{align*}
\]

where \( y = f(\sum w_i x_i) \) and \( f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \)

The connection is as shown in the learning rule (for updating connection strength):

\[
\Delta w_{ij} = (2x_i - 1)(2x_j - 1), \Delta x_{ij} = \Delta x_{ij}
\]

where \( x_i \) and \( x_j \) have values 0 or 1, \( x_i \) is the output of the current processing element, \( x_j \) is an input to the processing element, \( w_{ij} \) is the connection between the \( j \)-th processing element and the \( i \)-th one.

(1) variable declaration

?=two
?=one

(2) functions

\[
\begin{align*}
    &H[\cdot] \text{ function } H \text{ for neuron state} \\
    &O[\cdot] \text{ function } O \text{ for neuron output} \\
    &W \text{ for weight updating} \\
    &\text{connection function}
\end{align*}
\]

(3) connection

\[
\begin{align*}
    \text{cell name} & \rightarrow \text{state maximum} \\
    &\rightarrow \text{group name} \\
    &\rightarrow \text{state minimum}
\end{align*}
\]

\[
\begin{align*}
    x_0 & \rightarrow 0.3 \\
    x_1 & \rightarrow 0.001 \\
    \vdots & \rightarrow \text{or}
\end{align*}
\]

\[
\begin{align*}
    &\rightarrow \text{weight update function} \\
    &\rightarrow \text{output function} \\
    &\rightarrow \text{state update function}
\end{align*}
\]

# of cells in 1st input group

\[
\begin{align*}
    x_0 & \rightarrow 0.01 \\
    x_1 & \rightarrow 0.05 \\
    \vdots & \rightarrow 0.004
\end{align*}
\]

# of kind of sources

\[
\begin{align*}
    &\rightarrow \text{connect to connection weight}
\end{align*}
\]

\[
\begin{align*}
    &x_0 \rightarrow 0.2 \text{ or}
\end{align*}
\]

\[
\begin{align*}
    &x_1 \rightarrow 0.001 \text{ or}
\end{align*}
\]

\[
\begin{align*}
    &\rightarrow \text{connect to connection weight}
\end{align*}
\]

\[
\begin{align*}
    &x_0 \rightarrow 0.001 \\
    x_1 & \rightarrow 0.001 \\
    \vdots & \rightarrow 0.001
\end{align*}
\]

It can be easily shown that the other model takes a similar structure to that of Hopfield.

\[
\begin{align*}
    \text{Hebb:} & \rightarrow \text{H} \text{ for neuron state} \\
    \text{Hebb/anti-Hebb:} & \rightarrow \text{connect to connection weight}
\end{align*}
\]

Similarly, Perceptron with back-propagation learning and the HASP (Human Associative Processor) Net have been worked out successfully.

For example, if we want to apply Hebb/anti-Hebb or Hebb updating rule to the above net, we only have to rewrite the \( W \) function.

5. Conclusion

A lot of neural net models can be specified by the proposed neural-description language and simulated by the simulation system. However, the system is just for experimental purposes, hence the user interface is not very friendly and the input syntax checking is not very rigid. Besides, there are still many models which cannot be defined successfully by the experimental language (e.g. the Kohonon map) because some of their properties are too artificial to be defined. Another defect of the language is that it does not support random initialisation of connective configuration and weights, but it can easily be improved. We hope that future theoretical and application development can help in trying to find a unifying way of describing neural network models and thus facilitate the design of neural networks.

J.-L. WU and S.-T. CHENG

Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10764, Taiwan, Republic of China

References