공학 연구논문집 제26권 제1호 pp. 291~311, 울산대학교
Journal of Engineering Research Vol. 26, No. 1, pp. 291~311 1995.
University of Ulsan.

뉴로 컴퓨팅 시스템 시뮬레이션 언어 개발 및 응용환경의 구축

이수동 · 홍규완 · 김정수
컴퓨터공학과

<요 약>

대부분의 신경망 시뮬레이션 프로그램은 기존의 언어(C,PASCAL,FORTRAN등)로 구현되었다. 기존의 언어들은 신경망구조와 신경망의 동적행위를 제어하는 데 적합한 자료구조와 문장들을 가지고 있지 않기때문에 시뮬레이션 프로그램을 작성하거나 모델의 변경에 어려움이 많다. 본 논문에서는 신경망 시뮬레이션 프로그램을 작성하는 데 적합한 언어를 설계하였다. 특히 제안된 언어는 신경망 구조를 쉽게 표현할 수 있고 신경망의 동적행위를 효율적으로 기술하기 위하여 층마크로 정의 개념을 도입하였다. 언어는 두개의 DIVISION으로 구성되었다. 하나는 망구조를 기술하기 위한 NETWORK STRUCTURE DIVISION이고 다른 하나는 신경망의 행위를 제어하는 PROCEDURE DIVISION이다.

Development of Simulation Language for Neuro-Computing system

Soo-Dong Lee · Gyu-Wan Hong · Jeong-Soo Kim
Dept. of Computer Engineering

<Abstract>

Most of neural network simulation programs have been implemented with the conventional computer languages. It is difficult to implement simulation programs and is various to build neural network, because the laguages have not data structures and statements suitable for expressing neural network structures and controlling neural behaviors. This paper describes a neural network simulation language, specifically designed for the development and modeling of neural network applications. It has a

layer macro definition concept to express the arbitrary network structures and dynamic neural behaviors easily. It consists of the two divisions. First division describes the network structures. Second division controls the network behaviors.

1. 서 론

최근 몇년간 신경망에 대한 연구가 급속하게 진전되면서 다양한 신경망 구조와 학습 알고리즘이 개발되었고 많은 응용분야에 적용되어 좋은 성과를 얻고 있다. 신경망은 신경망 구조와 학습 알고리즘의 결합이라고 할 수 있다. 구조적 관점에서 보면 계층적 측면과 기능적 측면으로 나눌 수 있다. 계층적 측면에서의 가장 하위 구조는 뉴론 구조이고 층 구조는 뉴론 구조의 상위구조로 뉴론 구조의 집합체로 이루어지며, 모델 구조는 다시 여러 개의 층 구조로 구성되고 가장 상위구조인 시스템 구조는 여러 개의 모델 구조로 설계된다. 기능적 측면에서의 신경망은 정적부분과 동적부분으로 나눌 수 있다. 정적부분은 신경망의 틀을 형성하는 부분이며, 동적부분은 신경망의 동작특성을 특징지어주는 부분이다[1].

이러한 신경망의 구조적 특성을 결합하여 다양한 응용분야에 적용하기 위하여 몇 개의 신경망 시뮬레이션 환경이 개발되었다. 다양한 분야의 전문가들은 특정한 문제를 해결하는 데 신경망을 이용하기를 희망하고 있으며, 연구와 실험을 보조할 수 있는 신경망 시뮬레이션 환경을 요구하고 있다. 또한 신경망의 개념을 빨리 이해하고, 원형모델을 손쉽게 세우는 데는 컴퓨터 시뮬레이션이 유연하고 값싼 방법이라고 할 수 있다.

시뮬레이션 환경은 전부 같은 특성을 가지고 있지는 않으며 구현방법에 따라서 [1,2,3,4,5]. 첫째는 일련의 전달함수, 학습방법 등에 대한 라이브러리와 사용자 인터페이스를 제공하고 사용자가 필요한 인자를 선택하여 임의의 모델을 만들도록 하는 것이다. 이와같은 방법은 사용하기는 편리하나 모델의 변경과 확장이 어렵다.

둘째는 신경망 기술언어와 사용자 그래픽 인터페이스를 제공하는 것이다.

이와 같은 접근방법은 표현의 일반성은 좋으나 신경망 자체보다는 시뮬레이션 언어를 익히는 데 많은 시간을 소비할 수가 있다. 신경망의 적용분야가 다양해지고 새로운 모델에 대한 연구가 급속하게 진행되면서 모델의 변경과 확장이 쉽게 이루어지는 것이 바람직하다. 따라서 위에서 언급한 후자의 관점에서 시뮬레이션 프로그램을 작성하기 위한 신경망 기술 언어의 개발이 절실히 요구되고 있다.

본 연구에서 제안하는 신경망 시뮬레이션 언어 NNL(Neural Network simulation Language)은 언어의 일반적 특성들을 만족시키기 위하여 앞에서 언급한 신경망의 두 가지 측면, 계층적 측면과 기능적 측면을 모두 고려하여 신경망의 본질적인 면을 흡수하도록 한다. 즉 기능적 측면을 고려하여 신경망의 정적부분과 동적부분을 자연스럽게 표현하기 위하여 언어를 두 개의 DIVISION으로 나누고 있다. NETWORK STRUCTURE DIVISION은 신경망의 정적인 틀을 기술하는 계층적 구조로 구성되며, 신경망 이름, 층의 수, 각 층의 뉴론의 수, 층과 층의 연결형태를 기술한다.

*본 연구는 1994년도 울산대학교 연구비 지원에 의해 수행되었음.

PROCEDURE DIVISION은 사용자가 신경망의 동적흐름을 제어하는 부분으로서 선언문, 제어문, 명령문 등으로 구성한다. 또한 신경망의 기능적 측면에서 동적 특성을 계층적 측면의 층구조 단위로 집합화하기 위하여 층마크로 정의 개념을 제안한다. 층마크로 정의에서는 동적부분의 특성들인 합성함수, 활성화 함수, 학습규칙 함수, 초기치 설정, 연결방향을 지정할 수 있다. 층마크로 정의에 대한 처리는 NETWORK STRUCTURE DIVISION에서 이루어지며, 관련된 함수에 대한 층마크로 이름표와 층마크로 정의표가 작성되고 주기억장치에 적재된다. 이것은 PROCEDURE DIVISION에서 수행되어지는 명령문과 더불어 관련된 함수가 호출될 때 확장된다. 즉 층마크로 정의는 신경망을 구성하고 있는 하나의 층과 관련한 여러 개의 함수들의 집합을 기술함으로써 하나의 함수를 호출하고 처리하는 것보다 매우 강력하게 모듈화를 가능하게 함으로써 프로그램의 구성을 단순화시킬 수 있다. 특히 층마크로 정의 개념은 대부분의 신경망 모델이 층단위로 구성되며, 하나의 모델속에서도 각 층의 동적특성이 다른 경우와 층과 층의 연결로서 모델의 확장 및 변경을 할 때 쉽게 이용될 수 있다.

2. 층마크로 정의

2.1 기본 개념

신경망을 사용하여 응용문제를 해결하기 위해서는 모델 구조 혹은 시스템 구조를 이용하고 있으며, 시스템 구조도 결국은 여러 모델의 결합이라 볼 수 있으므로 모델 구조로 귀착될 수 있다. 대부분 신경망 모델은 단층 혹은 다층 구조를 갖고 있으며, 각 층은 같은 기능을 가진 여러 개의 뉴론 구조로 구성된 층신경망 구조이다[1,6].

따라서 신경망 모델을 기술하는 데 있어서 언어의 단순성과 일반성을 고려할 때 어떤 수준의 구조를 기술단위로 둘 것인가를 결정해야 한다. 또한 신경망 모델의 기술은 구조적 측면의 기술단위를 가지고 신경망의 기능적 측면을 구성한다. 신경망 개념의 명료성을 갖기 위해서 앞장에서 설명한 신경망의 기능적 측면을 집합 개념을 사용하여 묘사해보면 표 1과 같다.

신경망(Neural Network)	: NN = { SP, DP }
정적부분(Static Part)	: SP = { T }
위상구조(Topology)	: T = { F, I }
프레임(Frame)	: F = { L1, ...,Lm }
층(Layer)	: L = { N1, ...,Nm }
뉴론(Neuron)	: N
상호연결구조	: I = { SL, TL }
시작층(Start Layer)	: SL
목적층(Target Layer)	: TL
동적부분(Dynamic Part)	: DP = { S(0),NF,LR }
초기상태(Initial State)	: S(0) = { w(0),t(0), a(0) }
초기가중치(weight)	: w(0)
초기임계치(threshold)	: t(0)
초기활성치(activation)	: a(0)
뉴론함수(Neuron Function)	: NF
학습규칙(Learning Rule)	: LR

표 1. 신경망의 집합적 묘사
Table 1 Description of Neural Network by set

층마크로 정의는 다음과 같고, 그림 1은 정의에 대한 층구조적 묘사를 나타내고 있다.

[정의1] 층마크로 정의(Layer Macro Definition)는 신경망의 동적특성의 집합을 층구조 단위로 기술하여 하나의 이름으로 정의하는 언어개념이다. 정의의 기술은 다음과 같다.

<층마크로 정의(Layer Macro Definition)>
LMD = { S(0), NF, LR, CT }
초기상태(Initial State): S(0) = { w(0),t(0)}
초기가중치(Initial Weight) : w(0)
초기임계치(Initial threshold) : t(0)
뉴론함수(Neuron Function) : NF = { sf, af }
합성함수(Summation Function) : sf
활성화함수(Activation Function) : af
학습규칙(Learning Rule) : LR
연결형태(Connection Type) : CT

그림 1. 층마크로 정의 특성
Fig.1 Characteristics of Layer Macro Definition

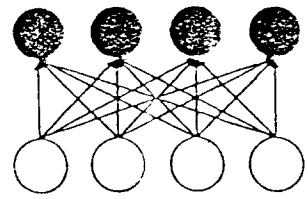
따라서, 층마크로 정의는 목적층과 더미층간의 일반적인 신경망의 동적동작을 기술한다. 목적층과 더미층사이를 연결하는 형태, 연결된 링크상에 존재하는 초기 가중치, 이 가중치를 변화시키기 위한 두 층간의 학습 방법, 그리고 목적층을 구성하는 뉴런에 적용되는 합성 방법, 활성화 방법, 초기 임계치 등이 정의된다.

2.2 층마크로 정의 문법

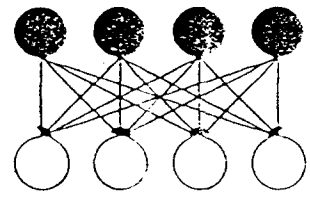
층마크로 정의에 대한 문법은 다음과 같다. 대문자로 된 단어는 언어의 키워드를 나타내고 있다.

<층마크로 정의 문법>
<Layer macro definition> ::=
LMACRO <macro name>
SUMMATION FUNCTION IS <function name>.
ACTIVATION FUNCTION IS <function name>.
LAYER CONNECTION IS <connection type>.
LAYER LEARNING RULE IS
<learning rule name>.
INITIAL WEIGHT VALUE IS <value type>.
INITIAL THRESHOLD VALUE IS <value type>.
ENDLMACRO
<macro name> ::= <identifier>
<function name> ::= <identifier> | NONE
<connection type> ::=
FEEDFORWARD | FEEDBACKWARD
| RECURRENT | COMPETITIVE
<learning rule name> ::= <identifier> | NONE
<value type> ::= RANDOM | ZERO | NONE

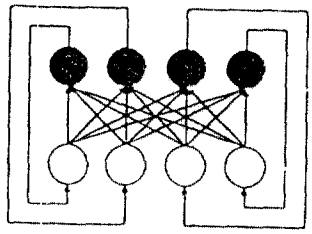
<macro name>은 층마크로 정의에 붙여지는 사용자 정의 식별자이다. SUMMATION FUNCTION IS와 ACTIVATION FUNCTION IS 다음에 오는 <function name>은 뉴론 함수명을 지정하는 것으로서 각각 지정된 층을 구성하는 뉴론의 입력신호를 합성하고, 활성화시키는 외부 함수에 대한 사용자 정의 식별자이다. LAYER CONNECTION IS 다음에 오는 <connection type>은 목적층과 더미층사이의 연결방향과 연결형태를 지정하는 것으로 4 가지 형태를 그림 2에 나타내었다. LAYER LEARNING RULE IS 다음에 오는 <learning rule name>은 지정된 층에 연결된 가중치에 대한 외부 학습 함수를 지정하는 사용자 정의 식별자이다. INITIAL 다음에 오는 <value type>은 각각 초기 가중치, 초기 임계치 형태를 지정하는 것으로써 난수를 지정하는 RANDOM 형태와 0을 지정하는 ZERO 형태가 있다. 또한 NONE은 요소를 지정할 필요가 없을 때 사용된다.



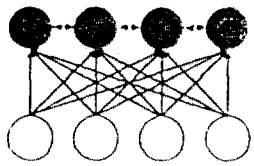
(a) feedforward 형태



(b) feedbackward 형태



(c) recurrent 형태



(d) competitive 형태

그림 2. 층의 연결형태
Fig. 2 Connection type of Layer

3. NNL 신경망 시뮬레이션 언어의 설계

본 연구에서 제안한 신경망 기술언어 NNL의 프로그램 구조는 크게 두개의 DIVISION으로 나누어진다. 첫번째는 신경망의 정적구조를 정의하는 영역으로서 층의 수, 각 층의 뉴런의 수, 각 층의 기능, 층과 층의 연결성, 층 마크로 정의을 기술하는 NETWORK STRUCTURE DIVISION이고, 두번째는 정의된 신경망 구조에서 뉴론과 시냅스의 동적동작을 제어하는 PROCEDURE DIVISION이다. 프로그램의 구조는 다음과 같다.

<program> ::= <network structure division part>
<procedure division part>

3.1 NETWORK STRUCTURE DIVISION

NETWORK STRUCTURE DIVISION은 신경망의 정적구조를 정의하는 영역이며 문법은 COBOL과 유사하다. 신경망 언어는 신경망의 정적구조를 쉽게 표현할 수 있도록 구성되어야 한다. NETWORK STRUCTURE DIVISION은 네 개의 부영역으로 구성된다. 네 개의 부영역에는 신경망 식별 부영역, 층정의 부영역, 망연결 부영역, 층마크로 정의 부영역이 있다. 이 DIVISION에 대한 EBNF 형태의 문법은 다음과 같다. 대문자는 키워드를 나타내며 소문자는 사용자 정의 변수이다.

<network structure division part> ::=
NETWORK STRUCTURE DIVISION.
<network identification subsection>
<layer definition subsection>
<layer connection subsection>
<layer macro definition subsection>

1) 신경망 식별 부영역
(Network identification subsection)

신경망 식별 부영역은 사용자가 정의한 신경망 프로그램의 이름을 인식하는 영역이며 문법은 다음과 같다.

<신경망 식별 부영역 문법>
<network identification subsection> ::=
NETWORK IDENTIFICATION.
NETWORK NAME : <network name>
<network name> ::= <identifier> . NET
<identifier> ::= <letter> { <letter or digit> }
<letter or digit> ::= <letter> | <digit>

키워드 NETWORK NAME 다음에 사용자-정의 신경망 이름을 정의할 수 있다.

2) 층정의 부영역
(Layer definition subsection)

층정의 부영역은 층의 수, 각 층의 뉴론의 수, 층의 기능을 정의하며 문법은 다음과 같다.

```
<층정의 부영역 문법>
<layer definition subsection> ::=
        LAYER DEFINITION.
      NUMBER OF LAYER : <number of layer>
NUMBER OF NEURON OF LAYER <layer number> :
                          <number of neuron>
LAYER <number> : <layer function>
<number of layer> ::= <number>
<number> ::= <digit>
<layer number> ::= <number>
<number of neuron> ::= <number>
<layer function> ::= <identifier>
```

키워드 NUMBER OF LAYER 다음에 신경망을 구성하고 있는 층의 수를 지정하며, NUMBER OF NEURON OF LAYER <layer number> 다음에는 각 층을 구성하고 있는 뉴론의 수를 지정하고 있으며, 이것은 층의 갯수 만큼 반복 지정된다. LAYER <number> 다음에는 각 층의 기능을 명시하고 있으며, 각 층의 기능은 사용자가 임의의 이름을 가지고 지정할 수 있다.

3) 층연결 부영역
(Layer connection subsection)

층연결 부영역은 층과 층의 연결성(connectivity)을 정의한다. 문법은 다음과 같다.

```
<층연결 부영역 문법>
<layer connection subsection> ::=
LAYER CONNECTION DEFINITION.
LAYER <number> & <number> : <connection type>
<connection type> ::= FULLY | PARTIALLY
```

LAYER 다음에 오는 <number> & <number>로 지정되는 층사이의 연결성을 명시하고 있으며, 연결성에는 완전연결과 부분연결이 있다.

4) 층마크로 정의 부영역
(Layer macro definition subsection)

층마크로 정의 부영역은 목적층에 대한 층마크로 정의를 기술한다. 문법은 다음과 같다.

```
<층마크로 정의 부영역 문법>
<layer macro definition subsection> ::=
  LAYER MACRO DEFINITION.
LAYER <dummy layer number>
        &<target layer number> :
                    <layer macro defintion name>
<dummy layer number> ::= <number>
<target layer number> ::= <number>
<layer macro defintion name> ::=
                          <identifier> . MCR
```

<target layer number> 다음에 오는 <layer macro definition name>에서는 층의 동적특성을 지정하는 층마크로 정의 함수의 외부 화일명이 명시되고 있다.

3.2 PROCEDURE DIVISION

PROCEDURE DIVISION은 NETWORK STRUCTURE DIVISION에서 정의된 망구조에서 뉴론과 시냅스의 동적동작을 제어하는 영역이며 문법은 PASCAL 언어와 유사하다. 특히 신경망 제어에 효율적인 여러가지 명령어인 LDO, LREAD, LCOPY, LPRINT 명령어가 있다. 이 영역은 세 개의 연속적인 부영역인 변수선언 영역, 몸체 영역, 부프로그램 영역으로 구성된다. 구조는 다음과 같다.

```
<procedure division part> ::=
                    PROCEDURE DIVISION.
                  <variable declaration part>
  BEGIN <main body part> END.
                          <subprogram part>
```

1) 변수선언 영역
(variable declaration part)

변수선언 영역은 프로그램 수행에 사용되는 데이타형과 변수를 선언하는 영역이다. 변수형태에는 단순변수, 구조화된 변

수, 미리정의된 변수가 있다. 문법은 다음과 같다.

```
<변수선언 영역 문법>
<variable declaration part> ::= <empty> |
        VAR <variable declaration>
                {; <variable declaration> };
<variable declaration> ::= <simple variable>
                        | <structured variable>
                        | <predefined variable>
```

(1) 단순변수(simple variable)

단순변수는 하나의 값을 갖기 위한 기억장소로서 식별자에 의하여 명시되며 정수형, 실수형, 문자형 변수가 있다. 문법은 다음과 같다.

```
<단순변수 문법>
<simple variable> ::= <s-datatype><s-variable>
<s-datatype> ::= INT | FLOAT | CHAR
<s-variable> ::= <identifier> {, <identifier> }
```

(2) 구조화된 변수(structured variable)

구조화된 변수는 벡터변수를 정의하며 정수형, 실수형 벡터변수가 있다. 키워드 IVECTOR 다음에는 정수형 벡터변수, DVECTOR 다음에는 실수형 벡터변수가 선언된다. 문법은 다음과 같다.

```
<구조화된 변수 문법>
<structured variable> ::=
                <v-datatype><v-variable>
<v-datatype> ::= IVECTOR | DVECTOR
<v-variable> ::= <identifier> '['<identifier>
                        or <digit>']'
```

(3) 미리정의된 변수(predefined variables)

실행시간에 인터프리터가 자동적으로 생성하는 변수는 10 가지가 있으며, 이를 미리정의된 변수라 정의하였고 각각의 의미는 표 2와 같다.

```
<predefined variable> ::= NNEURON | NLINK
                        | WEIGHT | LINKINPUT
                        | NETJ | OUT
                | THRESHOLD | CURRENTNEURON
                | CURRENTLINK | CURRENTLAYER
```

변 수	내 용
NNEURON	현재 실행되고 있는 층의 뉴론의 갯수를 나타낸다.
NLINK	현재 실행되고 있는 뉴론에 연결된 입력링크의 갯수를 나타낸다.
WEIGHT	현재 실행되고 있는 입력링크의 가중치를 나타낸다.
LINKINPUT	현재 실행되고 있는 입력링크에 연결된 뉴론의 출력치를 나타낸다.
NETJ	현재 실행되고 있는 뉴론의 입력치에 대한 가중합을 나타낸다.
OUT	현재 실행되고 있는 뉴론의 출력치를 나타낸다.
THRESHOLD	현재 실행되고 있는 뉴론의 임계치를 나타낸다.
CURRENTNEURON	현재 실행되고 있는 뉴론을 나타낸다.
CURRENTLINK	현재 실행되고 있는 입력링크를 나타낸다.
CURRENTLAYER	현재 실행되고 있는 층을 나타낸다.

표 2. 미리정의된 변수
Table 2 Predefined variables

2) 몸체 영역(main body part)

몸체 영역은 프로그램의 흐름을 제어하는 영역으로 여러 가지 형태의 제어문장들로 구성된다. 제공되는 문장은 크게 단순 문장, 조건 문장, 반복 문장, 입출력 문장, 호출 문장, 명령어 문장이 있다. 명령어 문장을 제외한 문장은 기존의 언어에서 제공하는 것과 유사하며, 명령어 문장은 신경망의 특징을 내포하는 문장이다. 문법은 다음과 같다.

```
<statement> ::= <simple statement>
              | <conditional statement>
              | <repeat statement>
              | <call statement>
              | <I/O statement>
              | <command statement>
```

(1) 단순 문장(simple statement)

단순 문장은 다른 문장을 포함하지 않는 문장으로 할당문을 의미하고 있다. 할당문은 수식으로 지정된 새로운 값을 변수의 현재 값으로 대치하는 역할을 한다. 문법은 다음과 같다.

```
<단순 문장 문법>
<simple statement> ::= <assignment statement>
<assignment statement> ::=
                    <variable> = <expression>
<expression> ::= <simple expression>
<simple expression> ::=
        <term> | <simple expression> + <term>
               | <simple expression> - <term>
<term> ::= <factor> | <term> * <factor>
                    | <term> / <factor>
<factor> ::= <digit> | <variable>
           | '(' <simple expression> ')'
```

(2) 조건 문장(conditional statement)

조건 문장은 여러 개의 구성 문장 중에서 하나를 선택하여 수행되도록 하는 문장으로 IF 문장, SWITCH 문장이 있다. 문법은 다음과 같다.

```
<conditional statement> ::=
        <if statement> | <switch statement>
```

(가) if 문장(if statement)

if 문장은 수식이 참이면 THEN 다음의 문장이 수행되고 그렇지 않으면 ELSE 다음의 문장이 수행된다. 문법은 다음과 같다.

```
<if 문장 문법>
<if statement> ::=
    IF '('<expression>')' THEN <statement>|
    IF '('<expression>')' THEN <statement>
    ELSE <statement>
```

(나) switch 문장(switch statement)

switch 문장은 수식과 구성요소의 리스트로 구성되며 각 구성요소는 수식의 형과 같은 명확한 상수의 집합에 의하여 제어된다. 라벨이 수식의 값과 같은 구성요소(element)가 수행되며, 어느 라벨도 일치되지 않으면 디폴트의 구성요소가 수행된다. 문법은 다음과 같다.

```
<switch 문장 문법>
<switch statement> ::=
    SWITCH <expression> OF <case list element>
                        {; <case list element>}
    ENDSWITCH
<case list element> ::=
        <case label> : <statement> | <empty>
<case label> ::= CASE <constant> | CASE DEFAULT
```

(3) 반복 문장(repeat statement)

반복 문장은 문장이 반복적으로 수행되는 것을 지정한다. 반복 문장에는 WHILE 문장, LOOP 문장, FOR 문장이 있다. 문법은 다음과 같다.

```
<repeat statement> ::= <while statement>
                     | <loop statement>
                     | <for statement>
```

(가) while 문장

WHILE 문장은 수식에서 주어진 값이 거짓일 때까지 계속 반복수행한다. 문법은 다음과 같다.

```
<while 문장 문법>
<while statement> ::= WHILE '('<expression>')'
                      <statement> {; <statement>};
                          ENDWHILE
```

(나) loop 문장

loop 문장은 constant에 지정된 횟수 만큼 반복수행한다. 문법은 다음과 같다.

```
<loop 문장 문법>
<loop statement> ::= LOOP <constant>
                     <statement> {; <statement>};
ENDLOOP
```

(다) for 문장

for 문장은 제어변수의 초기값에서 1씩 증가시키면서 제어변수가 최종치가 될 때까지 반복수행한다. 문법은 다음과 같다.

```
<for 문장 문법>
<for statement> ::=
        FOR <control variable> = <for list>
        DO  <statement> {; <statement> };
        ENDFOR
<for list> ::=
          <initial value> TO <final value>
<control variable> ::= <identifier>
<initial value> ::= <expression>
<final value> ::= <expression>
```

(4) 호출 문장(call statement)

호출 문장은 부프로그램 형태인 함수를 몸체부분에서 호출할 때 사용된다. 문법은 다음과 같다.

```
<call 문장 문법>
<call statement> ::= CALL <function name>
                     '('<actualparameterlist>')'';'
<function name> ::= <identifier>
<actualparameterlist> ::=
    <actualparameter> {,<actualparameter> }
<actualparameter> ::= <simple variable>
```

(5) 입출력 문장(I/O statement)

입출력 문장은 변수에 값을 입출력하기 위하여 사용된다. 입출력 문장에는 INPUT 문장, PRINT 문장, OPEN 문장, CLOSE 문장이 있다. 문법은 다음과 같다.

```
<i/o statement> ::=
      <input statement> | <output statement>
    | <open statement> | <close statement>
```

(가) INPUT 문장(Input statement)

INPUT 문장은 키보드 혹은 외부화일로부터 변수로 데이터를 받아 들이는 문장이다.

```
<Input 문장 문법>
<input statement> :: =
          INPUT <variable> FROM KEYBORD';'
                          | FILE <file name>
<variable> ::= <identifier>
<file name> ::= <identifier>
```

(나) PRINT 문장(Print statement)

PRINT 문장은 변수에 저장된 내용을 화면으로 출력하는 문장이다.

```
  <print 문장 문법>
<print statement> ::= PRINT "<string>" ';'
                    | <predefined variable>
                    | <simple variable>
```

(다) OPEN 문장(open statement)

OPEN 문장은 화일을 읽거나 쓰기 전에 개방한다.

```
  <open 문장 문법>
<open statement> ::=
        OPEN <file name> FOR READING';'
        | WRITING
```

(라) CLOSE 문장(close statement)

CLOSE 문장은 화일를 읽거나 쓰고난 후에 닫는다. 문법은 다음과 같다.

```
<close 문장 문법>
<close statement> ::= CLOSE <file name>';'
```

(6) 명령어 문장(command statement)

명령어 문장에는 하나의 층을 구성하고 있는 모든 뉴론에 대해 일괄적으로 수행되는 문장으로서 LDO 문장, LCOPY 문장, LREAD 문장, LPRINT 문장이 있다. 문법은 다음과 같다.

```
<command statement> ::=
        <ldo statement> | <lcopy statement>
      | <lread statement> | <lprint statement>
```

(가) ldo 문장(ldo statement)

ldo 문장은 지정된 층을 구성하는 뉴론으로 들어오는 입력신호의 합성, 각 뉴론의 활성화, 지정된 층에 연결된 가중치의 수정을 위한 학습을 실행할 때 사용된다. 사용되는 합성 함수, 활성화 함수, 학습규칙 함수는 층마크로 정의에서 명시된 것을 따르며, 층마크로 정의 프로세서에 의하여 관련된 함수가 확장된다. 문법은 다음과 같다.

```
  <ldo 문장 문법>
<ldo statement> ::=
      LDO LAYER <layer number> SUMMATION ';'
                    | ACTIVATION  | LEARNING
<layer number> ::= <digit>
```

키워드 LDO LAYER 다음에 오는 <layer number>는 지정된 층의 번호를 나타내며, 그 다음에 SUMMATION, ACTIVATION, LEARNING이 올 수 있다. SUMMATION은 입력신호의 합성, ACTIVATION은 뉴론의 활성화, LEARNING은 학습규칙의 수행을 뜻하고 있다.

(나) lcopy 문장(lcopy statement)

lcopy 문장은 층과 층사이의 벡터값을 복사하거나, 선언된 벡터 변수사이의 복사를 수행할 때 사용한다. 문법은 다음과 같다.

```
  <lcopy 문장 문법>
<lcopy statement> ::=
      LCOPY INPUT OF LAYER <layer number>
            | OUTPUT OF LAYER <layer number>
            | <variable>
      TO INPUT OF LAYER <layer number> ';'
            | OUTPUT OF LAYER <layer number>
            | <variable>
```

(다) lread 문장(lread statement)

lread 문장은 키보드 혹은 화일로부터 지정된 층의 입력, 층과 층사이의 가중치를 받아들일 때 사용된다. 문법은 다음과 같다.

```
  <lread 문장 문법>
<lread statement> ::=
        LREAD INPUT OF LAYER <layer number>
          | WEIGHT OF LAYER <layer number>
        AND LAYER <layer number>
        FROM KEYBORD ';'
          | FILE <file name>
<file name> ::= <identifier>
```

(라) lprint 문장(lprint statement)

lprint 문장은 지정된 층의 입력, 출력, 임계치, 층과 층 사이의 가중치를 화면 혹은 화일로 출력한다. 문법은 다음과 같다.

```
  <lprint 문장 문법>
<lprint statement> ::=
        LPRINT INPUT OF LAYER <layer number>
          | OUTPUT OF LAYER <layer number>
          | THRESHOLD OF LAYER <layer number>
          | WEIGHT OF LAYER <layer number>
          AND LAYER <layer number>
          TO SCREEN ';'
          | FILE <file name>
```

키워드 LPRINT 다음에는 INPUT, OUTPUT, THRESHOLD, WEIGHT가 올 수 있으며, 그 다음에 오는 키워드 TO 다음에는 SCREEN, FILE <file name>이 올 수 있다.

3) 부프로그램 영역(subprogram part)

함수는 부프로그램의 역할을 하고 있다. PROCEDURE DIVISION 다음에 별개로 기술되며, PROCEDURE DIVISION 안에서 CALL 문장에 의하여 호출된다. 문법은 다음과 같다.

```
  <부프로그램 영역 문법>
<subprogram part> ::=
                FUNCTION <function name> '('
        <formalparameterlist>')'{:<returntype>}
        <variable declaration part>
        <function body>
        RETURN { '('<returnvalue>')'}
<function name> := <identifier>
<formalparameterlist> ::=
<formalparameter>{,<formalparameter>}
        <formalparameter> ::= <simple variable>
<returntype> ::= <s-datatype>
<function body> ::= <statement>{;<statement>};
<returnvalue> ::= <simple expression>
```

4. 구현 및 실험결과

4.1 구현

본 논문에서 제안한 언어는 PC 상에서 C 언어로 구현하였다[7]. 번역기는 하나의 패스로 이루어진 인터프리터로 구현되었으며, 인터프리터를 작성할 때 필요한 신경망 구조에 관한 자료구조, 전체적인 프로그램의 흐름도와 관련된 동작함수를 중심으로 기술하고자 한다.

4.1.1 신경망 구조를 설계하기 위한 자료구조

신경망 구조를 설계하기 위한 구성요소에는 뉴론 구조, 시냅스 구조, 층 구조, 연결 구조, 망 구조가 있다. 개개의 구조에 대한 자료구조의 구현은 C 언어의 struct 데이타형을 사용하였으며, 구조의 집합은 연결 리스트로 구현하였다.

1) 뉴론 구조를 구성하기 위한 자료구조

뉴론 구조를 구성하고 있는 요소는 뉴론에 대한 입력값, 활성화된 출력값, 임계값, 뉴론에 연결된 시냅스 포인터, 다음

뉴론 포인터이다.

```
struct Neuron {
        float IValue;    /* input value  */
        float OValue;    /* output value  */
        float TValue;    /* threshold value  */
        struct Synapse *InputLinks;
                /* pointer of input synapse */
        struct Neuron *NextNeuron;
                /* pointer of next neuron */
};
```

2) 시냅스 구조를 구성하기 위한 자료구조

시냅스 구조를 구성하고 있는 요소는 가중치, 근원 뉴론 포인터, 다음 시냅스 포인터이다.

```
struct  Synapse {
        float WValue;        /* weight value */
        struct Neuron *FromNeuron;
                /* pointer of source neuron */
        struct Synapse *NextSynapse;
                /* pointer of next synapse */
};
```

3) 층 구조를 구성하기 위한 자료구조

층 구조를 구성하고 있는 요소는 뉴론의 수, 층의 기능, 뉴론 리스트 포인터, 다음 층 포인터이다.

```
struct Layer {
        int NNeuron;   /* number of neuron */
        int LayerType;  /* function of layer */
        struct Neuron *NeuronList;
                /* pointer of neuron list */
        struct Layer *NextLayer;
                /* pointer of next layer */
};
```

4) 연결 구조를 구성하기 위한 자료구조

연결 구조를 구성하고 있는 요소는 근원층, 목적층, 연결형태, 다음 연결 구조 포인터이다.

```
struct Connection {
        int FromLayer;      /* source layer */
        int ToLayer;        /* target layer */
        int ConType; /* connneection type */
       /* pointer of next connection */
       struct Connection *NextConnection;
};
```

5) 망 구조를 구성하기 위한 자료구조

망구조를 구성하고 있는 요소는 층의 수, 층리스트 포인터, 연결 리스트 포인터로 이루어진다.

```
struct Network {
        int NLayer;     /* number of layer */
        struct Layer *LayerList;
                /* pointer of layer list */
        struct Connection *ConnectList;
                /* pointer of connection list */
};
```

4.1.2 인터프리터의 흐름도와 함수들

1) 인터프리터의 주프로그램 흐름도

인터프리터의 주프로그램 구성은 크게 2개 영역으로 나누어진다. Network_S_D() 함수는 신경망의 정적구조를 정의하는 NETWORK STRUCTURE DIVISION 부분을 처리하는 프로그램이며, Procedure_D() 함수는 신경망의 동적흐름을 기술하는 PROCEDURE DIVISION 부분을 처리하는 프로그램이다. 인터프리터의 주프로그램에 대한 흐름도를 그림 3에 나타내었다.

2) NETWORK STRUCTURE DIVISION 처리 프로그램의 흐름도

NETWORK STRUCTURE DIVISION 처리 프로그램인 Network_S_D()는 원시

프로그램의 NETWORK STRUCTURE DIVISION 영역을 읽어들여 신경망의 정적 구조를 설정하는 프로그램으로서 신경망 식별 함수 Netname(), 층정의 함수 Layer_D(), 층연결 함수 Layer_C()로 구성된다. 이 영역에 대한 흐름도를 그림 4에 나타내었다.

Netname() 함수 : 신경망 이름을 처리하는 함수로서 "NETWORK NAME:" 다음에 정의된 사용자 정의 이름을 신경망 이름표에 넣는다.

Layer_D() 함수 : 층구조를 구성하는 함수로서 다음에 따라오는 토큰 스트림을 스캔하여 "NUMBER OF LAYER :" 다음에 정의된 층의 수와 "NUMBER OF NEURON OF LAYER"의 스트링열을 인식하여 각 층의 뉴론의 갯수를 얻고, "LAYER" 스트링열을 인식하여 각 층의 기능을 얻는다. 여기서 얻어진 정보를 이용하여 필요한 메모리에 동적으로 할당하고, 층구조는 연결 리스트로 구성한다.

Layer_C() 함수 : 층과 층의 연결성에 대한 정보를 얻기 위하여 스트링열 "layer" 다음에 오는 부분을 스캔하여 근원층과 목적층 그리고 연결형태에 대한 인자를 얻는다. 여기서 얻어진 정보를 이용하여 층과 층 사이의 연결을 구성한다.

3) PROCEDURE DIVISION 처리 프로그램의 흐름도

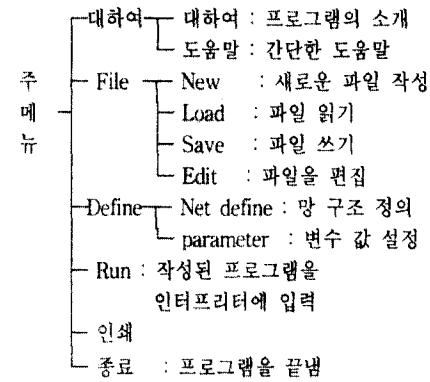
PROCEDURE DIVISION처리 프로그램인 Procedure_D()는 NETWORK STRUCTURE DIVISION의 처리가 끝난 다음 제어권을 받고, 원시 프로그램의 PROCEDURE DIVISION 영역을 읽어서 신경망의 동적흐름을 실행시키는 프로그램으로서 선언부분의 처리함수 VarDecl(), 몸체부분을 처리하는 함수 MainBlock(), 부프로그램을 처리하는 Sub_p()로 구성된다. 이 영역의 처리 프로그램에 대한 흐름도를 그림 5에 나타내었다.

MainBlock()함수는 PROCEDURE DIVISION을 구성하고 있는 모든 문장에 대한 처리함수로 구성된다. VarDecl()함수는 PROCEDURE DIVISION의 선언부분을 처리하는 함수로서 토큰 VAR 다음에 선언된 모든 데이타형을 처리한다. Sub_P() 함수는 함수 기술부분을 처리한다.

4.1.3 사용자 인터페이스의 구현

본 논문에서 제안한 언어의 번역기는 하나의 패스로 이루어진 인터프리터로 구성되어 있어서 사용자가 사용하기에는 컴파일러 보다 불편한 점이 많다. 이러한 점들을 보완하기 위해서 에디터를 포함한 사용자 인터페이스를 구현하였다. 인터페이스 프로그램은 IBM PC에서 Turbo-C로 작성되었으며, 한우진님의 한글 라이브러리를 사용하였다.

1) 인터페이스 프로그램의 메뉴 구성



2) 인터페이스 프로그램의 특징

프로그램을 작성하거나 수정하기 쉽도록 기본적인 에디터를 제공하였다. 그리고 작성된 프로그램은 Run 메뉴를 수행하므로써 직접 인터프리터의 입력으로 한행씩 제공된다. Define 메뉴는 망 구조를 정의하는 Net define 메뉴와 학습률(learning

rate), 관성항(momentum) 등의 변수값들을 정의하는 parameter 메뉴로 구성된다. Net define 메뉴를 실행하면 층(layer)의 수와 연결형태를 입력하여서 기본적인 망의 구조를 화면상의 그림으로 확인할 수 있고, 자동으로 NETWORK STRUCTURE DIVISION의 LAYER DEFINITION과 NET WORK CONNECTION DEFINITION을 자동으로 생성할수가 있다. parameter 메뉴는 기본적인 변수값들을 정의할수가 있다.

3) 사용자 인터페이스 프로그램의 실행

처음 실행시키면 로고를 보여준다(그림 6). 필요한 파일을 선택하여 Edit를 선택하면 수정을 할 수가 있다(그림7). 그림8은 Define 메뉴를 선택했을때의 모습이다. 입력된 층의 수와 연결형태에 의해서 그려진 망의 구조를 보여주는 화면과 각 변수를 설정하는 메뉴를 보여주고 있다.

4.2 실험결과

제안된 언어에 대한 성능을 평가하기 위하여 여러가지 신경망 모델을 PC 상에서 실험하였다. 부분결합 회귀신경망(Partially Connected Recurrent Neural Network) 모델과 오류역전파 (Error-Back Propagation) 다층퍼셉트론 모델을 사용하여 상태궤적 발생 문제의 실험을 하였다[1,6,8].

4.2.1 부분결합 회귀신경망과 오류역전파 다층퍼셉트론에 의한 실험

1) 상태궤적 발생 문제

상태궤적 발생은 신경망이 연속적으로 나타나는 상태궤적 좌표의 값을 학습한 다음, 모델이 학습된 궤적을 외부의 입력이 없이 발생시키는 문제이다. 학습에 사용된 데이타는 원형 궤적상의 12 개의 좌표점을 사용하였다.

2) 부분결합 회귀신경망 모델에 의한 실험결과

부분결합 회귀신경망은 순차처리의 특성을 갖고 있는 모델이다. 실험에 사용된 신경망 구조에 대한 기술언어의 묘사는 다음과 같다. 층의 수는 5 개이며, 첫번째 층은 입력층으로 2 개의 뉴론으로 구성되며 상태궤적 좌표(X,Y)를 받아들인다. 두번째 층은 은닉층으로 15 개의 뉴론으로 구성했으며, 세번째 층은 출력층으로서 2 개의 뉴론으로 구성됐다. 네번째 층은 내부상태층으로 15 개의 뉴론으로 구성되며 은닉층의 출력이 복사된다. 다섯번째 층은 결정층으로서 2 개의 뉴론으로 구성되며 출력층의 결과가 복사된다. 입력층과 은닉층, 은닉층과 출력층은 완전 연결되었으며, 내부상태층, 결정층과 은닉층은 부분연결되었다. 은닉층과 출력층에 대한 층마크로 정의의 기술에는 합성함수, 시그모이드 함수가 사용되었으며, 연결형태는 FEEDFORWARD이고 초기상태는 -1과 1 사이의 난수로 지정되었다. 학습함수는 오류역전파 알고리즘을 묘듈화한 2 개의 부분함수로 구성되었다. 그림 9는 실험에 사용된 신경망 구조가 층마크로 정의 단위로 구성됨을 보이고 있다. 표 3은 100 번 반복학습 후 출력층 뉴론의 활성화 정도를 나타내었다.

학습에 사용된 궤적 데이타		출력층 뉴론의 활성도	
X 좌표	Y 좌표	뉴론 1	뉴론 2
0.1	0.5	0.11991	0.48312
0.2	0.3	0.18852	0.29948
0.3	0.2	0.32001	0.18911
0.5	0.1	0.47862	0.12124
0.7	0.2	0.69045	0.23732
0.8	0.3	0.81920	0.28925
0.9	0.5	0.85922	0.54431
0.8	0.7	0.83546	0.67000
0.6	0.8	0.59015	0.78966
0.5	0.9	0.50418	0.89115
0.3	0.8	0.27999	0.80313
0.2	0.7	0.22608	0.72966

표 3. 부분결합 회귀신경망에 의한 실험결과
Table 3 Experiment result with PCRNN

학습에 사용된 궤적 데이타		출력층 뉴론의 활성도	
X 좌표	Y 좌표	뉴론 1	뉴론 2
0.1	0.5	0.10836	0.48132
0.2	0.3	0.21102	0.29559
0.3	0.2	0.29630	0.22532
0.5	0.1	0.49878	0.12059
0.7	0.2	0.79616	0.25878
0.8	0.3	0.80471	0.31026
0.9	0.5	0.89496	0.48456
0.8	0.7	0.79653	0.66031
0.6	0.8	0.58380	0.81834
0.5	0.9	0.49148	0.89203
0.3	0.8	0.27466	0.82320
0.2	0.7	0.23714	0.71434

표 4.오류역전파 모델에 의한 실험결과
Table 4 Experiment result with BackPropagation model

3) 오류역전파 다층퍼셉트론 모델에 의한 실험결과

오류역전파 다층퍼셉트론 모델에 대한 정적구조의 구성은 3 개 층으로 이루어졌으며 첫번째 층은 입력층으로 2 개의 뉴론으로 구성되며, 상태궤적 좌표(X,Y)를 입력으로 받는다. 두번째 층은 은닉층으로 15 개의 뉴론으로 구성된다. 세번째 층은 출력층으로 2 개의 뉴론을 갖고 있다. 입력층과 은닉층, 은닉층과 출력층은 완전연결되며, 동적특성을 기술하기 위하여 은닉층과 출력층에 대한 층마크로 정의가 설정되었다. 은닉층과 출력층에 대한 층마크로 정의 기술에는 합성 함수, 시그모이드 함수가 사용되었으며 연결형태는 FEED FORWARD이고 초기상태는 -1과 1사이의 난수로 지정되었다. 학습 함수는 오류역전파 알고리즘을 모듈화하여 2 개의 부분함수로 구성되었다. 그림 10은 층마크로 정의에 근거한 오류역전파 다층퍼셉트론 모델의 신경망 구조를 보이고 있다. 표 4는 100 번 반복학습 후 출력층 뉴론의 활성화 정도를 나타내었다.

4.3 NNL 언어의 평가

신경망 기술언어의 일반적 특성인 단순성, 일반성, 묘듈성, 재활용성을 평가기준으로 한다. 신경망은 신경망 구조와 학습 알고리즘의 결합이라 할 수 있으므로 언어는 이 두 부분을 효과적으로 기술하기 위하여 어떤 형태의 프로그램 구조, 문장과 언어 개념을 제공하고 있느냐 하는 관점에서 평가하겠다[1,7].

(1) 단순성

AXON 언어는 신경망의 구조를 설계하는 데 매개변수 정의블록, 처리요소와 층 정의블록, 망생성과 연결 정의블록을 사용하고 있다. 매개변수 정의블록에서는 record lt is 다음에 입력층, 은닉층, 출력층의 뉴론의 수를 지정하며 record rt is 다음에 실행시간 인자를 지정한다. 그림 11은 매개변수 정의블록의 예를 나타낸다.

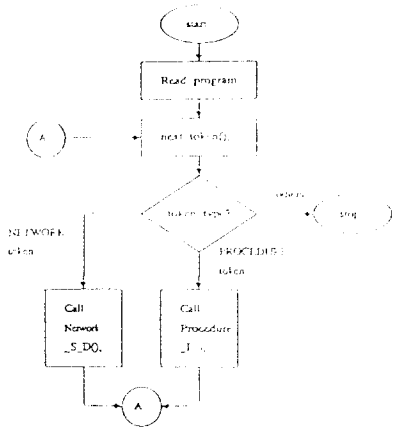


그림 3. 인터프리터의 주프로그램 흐름도
Fig.3 Flowchart for main program of interpreter

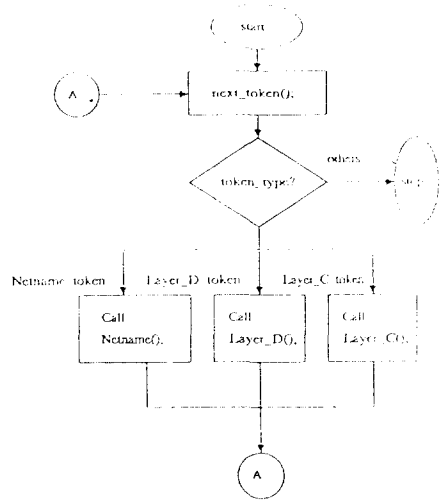


그림 4. NETWORK STRUCTURE DIVISION 처리 프로그램의 흐름도
Fig.4 Flowchart for processing program of NETWORK STRUCTURE DIVISION

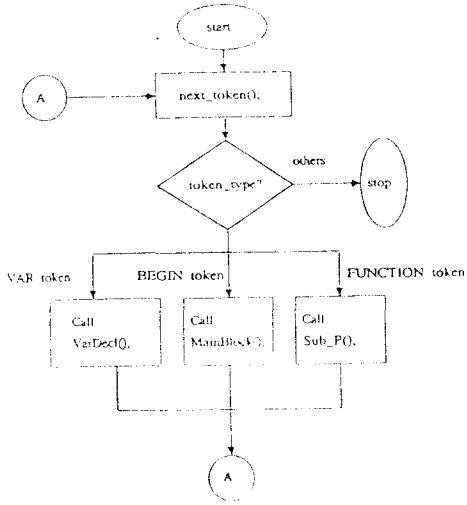


그림 5. PROCEDURE DIVISION 처리 프로그램의 흐름도
Fig.5 Flowchart for processing program of PROCEDURE DIVISION

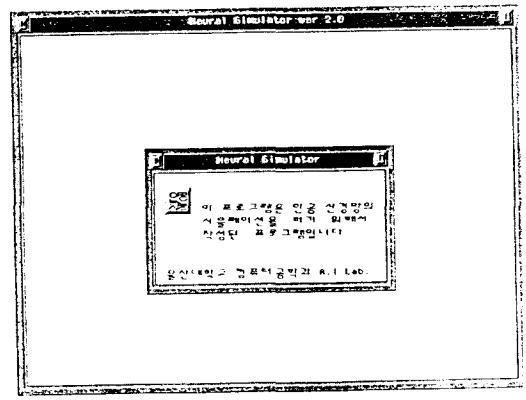


그림 6. 사용자 인터페이스의 로고화면

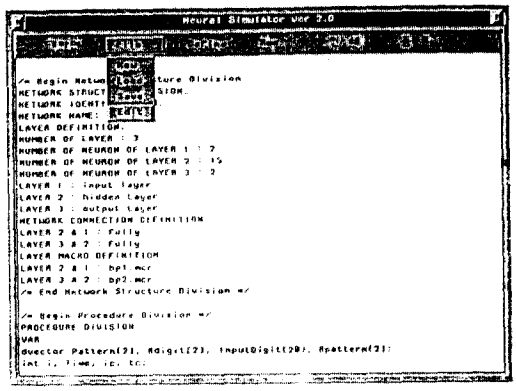


그림 7. 사용자 인터페이스의 실행화면

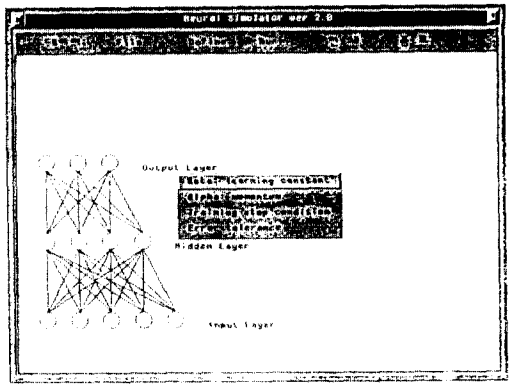


그림 8. 사용자 인터페이스의 실행화면

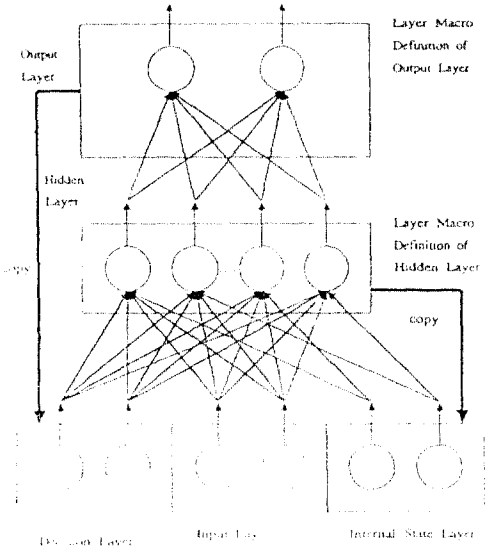


그림 9. 층별로 정의해 주어진 부분결합 회귀신경망 모델의 망구조

Fig. 9. Network structure of PCRNN based on layer macro definition

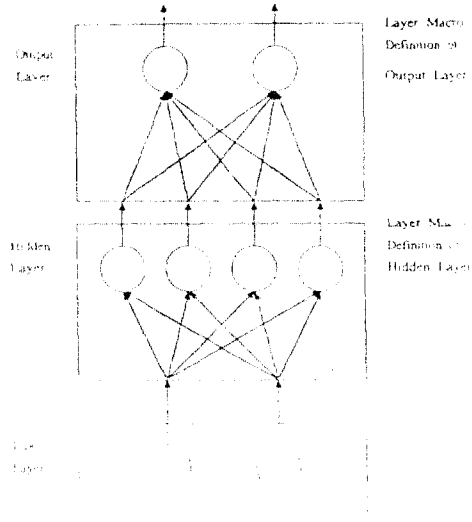


그림 10. 층별로 정의해 주어진 오류역전파 모델의 망구조

Fig. 10. Network structure of Backpropagation model based on layer macro definition

```
type record tCtsBpn is
    record lt is
        integer:32 iCnt;
        integer:32 hCnt;
        integer:32 oCnt;
        integer:32 RandomSeed;
        real:32    InitWeightMax;
        integer:32 ConnectInputs;
    end lt;
    record rt is
        real:32    HiddenAlpha;
        real:32    OutputAlpha;
        real:32    HiddenBeta;
        real:32    OutputBeta;
        integer:32 BatchSize;
        integer:32 LinearOutput;
        integer:32 LearnFlag;
    end rt;
end tCtsBpn;
```

그림 11. 매개변수 정의블록의 예
Fig.11 Example of Parameter Definition Block

처리요소와 층 정의블록에서는 뉴론과 층의 세부적인 특징을 기술하며 망생성과 연결 정의블록에서는 전체적인 신경망 구조가 생성되도록 한다. 그림 11은 처리요소에 대한 세부적인 기술과 연결 형태에 대한 예를 나타낸다. 즉, 세부적인 신경망 구조의 구성인자, 특징을 프로그래머가 기술하기 때문에 프로그램의 기술이 어렵다.

NNL 언어는 신경망의 구조를 구성하는데 NETWORK STRUCTURE DIVISION을 사용한다. 층 정의영역에서는 입력층, 은닉층, 출력층의 뉴론의 수와 각 층의 기능을 기술한다. 망연결 정의영역에서는 층 사이의 연결을 기술한다. 그림 12는 층 정의영역과 망연결 정의영역에 대한 예를 나타낸다.

```
/* Example of Layer definition */

LAYERDEFINITION.
NUMBER OF LAYER : 3
NUMBER OF NEURON OF LAYER 1 : 2
NUMBER OF NEURON OF LAYER 2 : 15
NUMBER OF NEURON OF LAYER 1 : 2
LAYER 1 : input layer
LAYER 2 : hidden layer
LAYER 3 : output layer

/* Example of network
        connection definition */

NETWORK CONNECTION DEFINITION.
LAYER 2 & 1 : fully
LAYER 3 & 2 : fully
```

그림 12. 층 정의영역과 망연결 정의영역의 예
Fig.12 Example of layer definition and network connection defintion

NNL 언어에서는 프로그래머가 정적구조의 인자들을 단순하게 지정하고, 각 뉴론의 동적특성을 층마크로 정의에 지정함으로써 전체적인 신경망 구조의 생성을 인터프리터가 처리하도록 한다. 따라서 NNL에서는 신경망 구조를 구성위하여 AXON 언어에서 사용된 세개의 블록을 하나의 블록으로 함축하고, 세부적인 처리를 인터프리터가 처리하도록 함으로써 신경망 구조의 설계를 쉽고 단순하게 한다.

(2) 일반성

언어는 다양한 신경망 구조와 동적특성들을 묘사할 수 있어야 한다. AXON 언어는 신경망 구조의 설계를 세개의 정의블록을 통하여 세분화하고 있다. 처리요소와 층 정의블록에서 각 층에 대한 처리요소의 구조를 프로그래머가 기술할 수 있다. 그림 13은 처리요소와 연결형태에 대한 기술의 예이다.

```
/* Example of processing element */
pe PeHidPlt is
    state is real : 32 xw;
    transfer function is
                HiddenPlanetXfer(tPass);
    input class list is F1, F2;
        signal of F1 is real:32 x;
        signal of F2 is real:32
                            HiddenError;
    local memory is
        weight of F1 is real:32 x;
end PeHidPlt;

/* Example of connection type */
for _pe in slabhidsun do
  hpdomain = slabhidplt[ipe:ipe
                + hidpltpersun -1:1];
  connect class F1 of HPdomain from
        OneAndInput using one_to_one();
  connect class F2 of HPdomain from
                        _pe using full();
  connect class F3 of _pe from HPdomain
                            using full();
  iPe += HidPltPerSun;
end;
```

그림 13. 처리요소와 연결형태 기술의 예
Fig.13 Example of processing element and connection type

그러나 지금까지 많은 신경망 구조에 사용된 뉴론 구조는 표준적 모형을 갖고 있다. 따라서 NNL 언어에서는 뉴론 구조의 모형을 표준화하여 인터프리터가 뉴론 구조의 세부적인 내용을 구성하도록 하고, 프로그래머는 뉴론의 동적특성을 나타내는 활성화 함수를 작성하여 층마크로 정의에 지정하도록 한다. 그림 9의 예처럼 NNL 언어에서는 신경망 구조를 층구조 단위로 구성하고 있다. 대부분 신경망 모델은 층구조로 구성되므로 NNL 언어는 일반성의 제한을 극소화할 수 있다.

(3) 모듈성

신경망의 동적특성을 기술하는 데는 뉴론의 입력치에 대한 합성 함수, 활성화 함수와 가중치를 수정하는 학습 함수 등이 필요하다. AXON 언어에서는 각 층의 동적 특성을 기술하기 위하여 하나의 함수로 정의하며 합성, 활성화, 학습 규칙 함수를 명확히 구분하여 지정하고 있지 않다. 따라서 각 층마다 합성함수, 활성화 함수가 같으나 학습함수가 다르므로 모든 층에 필요한 함수를 함수 정의블록에서

```
transfer function
HiddenPlanetXfer(tPass pass) is
   case pass of
        wien FORWARD;
             FOR Icn in this_pe.F1 do
                  this_pe.xw =
                            Icn.x * Icn.w;
             end;
        wien BACKWARD;
   for IcnF1 in this_pe.F1, IcnF2 in
                            this_pe.F2 do
                  IcnF1.w +=
                  Cts.rt.HiddenAlpha *
                  sIcnF2.HiddenError *
                  IcnF1.z;
             end;
   end case;
end HiddenPlanetXfer;
```

그림 14. 동적특성에 대한 함수의 예
Fig.14 Example of function for dynamic characteristics

기술해야한다. 즉, 같은 내용의 합성 함수, 활성화 함수를 각 층마다 중복하여 기술하는 결과를 초래하고 프로그램의 코드 크기가 커지며, 묘듈성을 악화시킨다. 그림 14는 은닉층의 동적특성을 기술하는 함수의 예를 나타낸다.

NNL 언어에서는 합성 함수, 활성화 함수, 학습 함수를 각각 별개의 라이브러리 함수로서 층마크로 정의에 지정할 수 있도록 함으로써 프로그램의 묘듈성을 높이고 프로그램의 크기를 축소하고 있다. 그

림 15는 층마크로 정의에서 합성함수, 활성화 함수, 학습규칙 함수를 지정하는 예를 나타낸다.

```
LMACRO bp1
        SUMMATION FUNCTION IS summationf.
        ACTIVATION FUNCTION IS sigmoidf.
        LAYER CONNECTION IS feedforward.
        LAYER LEARNING RULE IS deltarule1.
        INITIAL WEIGHT VALUE IS random.
        INITIAL THRESHOLD VALUE IS random.
ENDLMACRO
LMACRO bp2
        SUMMATION FUNCTION IS summationf.
        ACTIVATION FUNCTION IS sigmoidf.
        LAYER CONNECTION IS feedforward.
        LAYER LEARNING RULE IS deltarule2.
        INITIAL WEIGHT VALUE IS random.
        INITIAL THRESHOLD VALUE IS random.
ENDLMACRO
```

그림 15. 동적특성 함수의 모듈성의 예
Fig.15 Example of modularity for dynamic characteristic function

(4) 재활용성

사용자가 작성한 신경망 라이브러리를 쉽게 수정하여 이용하거나, 새로운 함수를 프로그램의 다른 부분에 영향을 미치지 않고 추가할 수 있는 명료한 언어 개념이 AXON 언어에는 없다. 즉, 오류역전파 다층퍼셉트론 모델에서 은닉층과 출력층을 구성하는 뉴론의 활성화 함수는 시그모이드 함수를 사용하고 있으나 AXON 언어에서는 각 층에 대하여 모든 동적특성의 함수를 하나의 함수로서 기술하므로 시그모이드 함수를 따로 기술할 수가 없다. 따라서 이미 작성된 함수를 여러 개의 층에 대하여 재사용하기가 어렵다. NNL 언어는 신경망의 동적특성을 나타내는 모든 함수를 별개의 라이브러리 함수로 작성하도록 하고, 같은 함수를 사용하는 모든 층에 대하여 층마크로 정의에 기술함으로써 재사용할 수 있도록 한다. 또한 모델이 다른 경우에도 같은 함수를 쉽게 재사용할 수가 있다. 그림 15는 두개의 층마크로 정의에서 같은 함수가 재사용됨을 나타낸다.

5. 결 론

본 논문에서는 신경망 기술언어의 일반적 특성을 만족시키기 위하여 신경망의 계층적 특성과 기능적 특성을 모두 고려하여 신경망의 본질적인 면을 흡수하도록 한 신경망 기술언어 NNL(Neural Network simulation Language)을 제안하였다. 첫째로 기능적 측면을 고려하여 신경망의 정적부분과 동적부분을 자연스럽게 표현하기 위하여 언어를 두 개의 DIVISION으로 나누었다. NETWORK STRUCTURE DIVISION은 신경망의 정적인 틀을 기술하는 부분으로서 계층적 구조로 구성하였다. PROCEDURE DIVISION은 사용자가 신경망의 동적흐름을 제어하도록 하는 부분이다.

둘째로 신경망의 기능적 측면에서 동적특성을 계층적 측면의 층구조 단위로 집합화하는 언어개념으로 층마크로 정의를 제안했다. 층마크로 정의에 의해 관련된 여러 개의 함수집합이 기술됨으로써 하나의 함수를 호출하고 처리하는 것보다 매우 강력하게 모듈화가 가능하게 되어 프로그램의 구성이 단순화되었다. 특히 층마크로 정의는 하나의 모델 속에서도 각 층의 동적 특성이 다른 경우 층과 층의 연결로써 모델의 확장및 변경을 할 경우 쉽게 이용될 수 있다.

그러나 언어의 번역기가 인터프리터로 구현되었기 때문에 컴파일러보다 프로그램을 실행하는 속도가 늦고, 층마크로 정의에서 인자전달 방법이 없는 단점이 있다. 또한, 신경망 특성중의 하나인 병렬성을 표현할 수 있도록 하기 위하여 언어의 특징을 확장할 필요가 있다.

그리고, 좀더 편리한 응용환경의 구축이 필요하다. 현재 풀다운 메뉴로 구성된 사

용자 인터페이스 프로그램을 좀더 실질적인 그래픽 사용자 인터페이스가 되도록 하기 위해서 메뉴의 구조도 개선되어져야 한다. 그리고 Definition 메뉴를 좀더 보강하여 CAD 프로그램과 같이 망 구조를 그리게 되면 신경망 언어가 생성되도록 구현하여야 한다. 또한, 언어의 인터페이스 뿐만 아니라 실제 응용환경에서 사용이 가능하도록 시뮬레이션 기능을 좀더 보강하여야 하겠다.

참고문헌

1. Robert Hecht-Nielsen, Neurocomputing, Addisin-Wesley,1990.
2. Andrew B. Smith, "A Parallel PDP Network Simulator," IEEE IJCNN, Vol.III, pp.377-384, 1988.
3. Nigel H.Goddard, Kenton J. Lynne, Toby Mintz, "Rochest Connectionist Simulator," Rochester University, 1988.
4. A.S.Bavan," NPS: A Neural Network Programming System," IEEE IJCNN, Vol. I, pp.35-38,1990.
5. Bruce P.Lester," A Neural Network Simulation Language Based on Multi-Pascal," IEEE IJCNN, Vol.III, pp.347- 354, 1988.
6. Alianna J. Maren, Craig T. Harston and Robert M. Pap, Handbook of Neural Computing Applications, Ac ademic Press, 1990.
7. Russel C. Eberhart, Roy W. Dobbins, Neural Network PC Tools, Academic press, Inc., 1990.
8. 김성석, "순차구조 처리를 위한 부분결합 회귀신경망," 공학박사 학위논문, 울산대학교, 6월 1990년.