The Development of Logic Programming On Deduction Neural Networks

Wai-Ip Kam

Bachelor of Science in Computer Science
The University of Bath
April 2009
This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:
Logic Programming On Neural Networks

Submitted by: Wai-Ip Kam

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).
This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:
Abstract

This dissertation is mainly focus on a practical development of logic programming on neural networks, which it is claimed as SLD neural networks. Most works are done by based on the thesis - Learning and deduction in neural networks and logic wrote by Ekaterina Komendantskaya. This project is searching a practical way to develop such system, in addition we are developing an API for constructing normal and SLD neural networks which support big numbers, moreover this project is motivated by developing an interpreter of logic program by using SLD neural networks instead of SLD-resolution and other related works which are supported by neural networks or first order logic.
# Contents

1 Introduction 1

1.1 Project description 1

1.2 Project aims 2

2 Literature Survey 3

2.1 Logic programming 3

2.1.1 First order logic 3

2.1.2 Unification and SLD resolution 4

2.2 Neural networks 5

2.2.1 Neural networks basic 5

2.2.2 Network structure 7

2.2.3 Learning 8

2.3 Logic programming on neural networks 10

2.3.1 Tp networks 10

2.3.2 Unification in neural networks 11

2.3.3 SLD neural networks 13

3 Requirements 14

3.1 Requirements Scope 14

3.2 Requirements specification 15

3.2.1 Functional requirements 15

3.2.2 Godel enumeration 16

3.2.3 SLD neural networks constructor 16
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 Non-functional requirements</td>
<td>17</td>
</tr>
<tr>
<td>3.3.1 Compatibility</td>
<td>17</td>
</tr>
<tr>
<td>4 Design</td>
<td>18</td>
</tr>
<tr>
<td>4.1 High level design</td>
<td>18</td>
</tr>
<tr>
<td>4.2 API structure</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Modify Godel Encoding</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Design of node in neural network constructor</td>
<td>24</td>
</tr>
<tr>
<td>4.5 Design of neural network in neural network constructor</td>
<td>27</td>
</tr>
<tr>
<td>4.6 SLD neural network constructor</td>
<td>30</td>
</tr>
<tr>
<td>5 Implementation and Testing</td>
<td>32</td>
</tr>
<tr>
<td>5.1 Language choice</td>
<td>32</td>
</tr>
<tr>
<td>5.2 Examples of the construction of neural networks by using the API</td>
<td>32</td>
</tr>
<tr>
<td>5.2.1 A simple neural network</td>
<td>32</td>
</tr>
<tr>
<td>5.2.2 Encoding example</td>
<td>35</td>
</tr>
<tr>
<td>5.2.3 Unification in neural network</td>
<td>37</td>
</tr>
<tr>
<td>5.2.4 First example of SLD neural network</td>
<td>39</td>
</tr>
<tr>
<td>5.2.5 Auto construction of SLD neural network</td>
<td>43</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>45</td>
</tr>
<tr>
<td>6.1 Critical analysis of the project</td>
<td>45</td>
</tr>
<tr>
<td>6.2 Project evaluation</td>
<td>46</td>
</tr>
<tr>
<td>A Code</td>
<td>48</td>
</tr>
<tr>
<td>A.1 File: encoding.h</td>
<td>49</td>
</tr>
<tr>
<td>A.2 File: encoding.c</td>
<td>49</td>
</tr>
<tr>
<td>A.3 File: n_network.h</td>
<td>56</td>
</tr>
<tr>
<td>A.4 File: n_network.c</td>
<td>56</td>
</tr>
<tr>
<td>A.5 File: node.h</td>
<td>59</td>
</tr>
<tr>
<td>A.6 File: sld_nn.h</td>
<td>60</td>
</tr>
<tr>
<td>A.7 File: sld_nn.c</td>
<td>61</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Neural network construction .................................................. 6
2.2 Neural network construction .................................................. 8

4.1 Neural network construction .................................................. 19
4.2 Interpreter construction ....................................................... 19
4.3 API structure .................................................................. 20
4.4 First design of node structure ............................................... 24
4.5 Second design of node structure ............................................ 25
4.6 Neural network example ....................................................... 26
4.7 Third design of node structure .............................................. 26
4.8 Final design of node structure .............................................. 27
4.9 Design of neural network and layer structure ....................... 28
4.10 Brief view of the construction neural networks .................... 28
4.11 Neural network with layers ................................................. 29

5.1 Neural network of the example code ..................................... 35
5.2 SLD neural network of the example code ............................... 40
List of Tables

2.1 The change from original to modify Godel encoding . . . . . . . . . . . . . 11

4.1 The change from original to modify Godel encoding . . . . . . . . . . . . . 23
4.2 Types and encoding of types . . . . . . . . . . . . . . . . . . . . . . . . . . 23
4.3 Terms, types and IDs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
4.4 Construction of Atoms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
4.5 Construction of goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
Acknowledgements

I would like to thank my supervisor Dr John power for providing much time and giving relevant advice. I would also like to thank Shiwei Zheng for helping and supporting. Finally, I would also like to thank my friend for encourage me.
Chapter 1

Introduction

1.1 Project description

Neural network has been known by people since the late 19th century, by that time neural networks were one of the study in biology research area, people knew it as human brain's structure and try to understand how it may works. Sometimes later around 1940s, research on neural network has been improved, one of the great idea is to implement neural networks with electrical circuits, by Warren McCulloch and Walter Pitts, also the idea on how neural networks would learn when they are operating, by Donald Hebb. These two works gave neural networks a brilliant future, because people no longer know neural networks as the studies of biology, but also other areas such as mathematics and computing etc.

Human brain are so powerful because it has the ability of learning, thinking and creating, what makes it so powerful is its own structure: the neural networks. Human brain’s neural network are constructed by lot of tiny cells, call neurons, each neuron contain input and output units that interconnect with other neurons, in addition each neuron has a single soma, which use for calculating inputs and push it to output. The idea is simple, but the structure is complex. In theory the more complex the network is, the more power of learning and pace the network will be. It may not be easy to prove that, but we may try to consider, human brain is 10 billion times fast than modern computer but each neuron in human brain is about 5 to 6 orders of magnitude slower than silicon logic gates. Such that, we can imagine the structure on neutral network give human brain such incredible result, we can also imagine if we can construct a neural network with silicon logic gates which has enough complexity to be as powerful as human brain, most our modern problems can be solved faster with its great learning ability.

Learning abilities are not the only advantage of neural networks, whereas it can solves logic problem, especially logic problem. Similarly, neural networks can solve logic problems. In the real world, we face many logic problems everyday, for example, when we ask “Can I have lunch with Tony?”, with some conditions “I can only have lunch with my friends.” and “Tony is one of my many friends.”. Although, this logic problem can be solved easily by
our brain, it is not easy to be solved by computer system. Before consider how to solve such problems in neural, it is important to understand how these logic problems can be represented in computer. There are numbers of representations of logic problems, however in this project, we focus on first order logic. The reason of it is, first order logic has replaced propositional logic due to the latter is not adequate for formalising valid arguments that rely on the internal structure of the propositions involved. Therefore, we only deal with first order logic in this project.

In the paper (Komendantskaya, 2007), it described how neural networks can perform SLD resolution. Which SLD-resolution is the deductive process to solving logic program. However, the method of SLD neural networks is introduced in this paper, which solves logic programs in deductive process by using neural networks, such that this project is focusing on the development of SLD-neural networks. However, this is not the only task in this project, whereas we are going to construct an application programming interface to construct both general and SLD neural networks.

1.2 Project aims

In this project we will discuss the design and implementation of the application programming interface, which allows user to construct to construct neural networks in a relatively simple and fast environment, in addition with the ability to construct SLD neural networks, whereas to prove SLD neural networks are able to be implemented in any practical computer system.
Chapter 2

Literature Survey

2.1 Logic programming

In this review, it assume the readers understand the basic logic such as propositional logic, in the following sections, we are going to look at the more advance logic system, call formal logic and logic program, which allows people to solve more difficult logic problems in computer system.

2.1.1 First order logic

Formal system

Formal logic system intends to put logic in a more formal and readable way (Richardson, 1998).

Definition 1.1 A formal system is \((D, R)\), where \(D\) is a set of data structure, and \(R\) is a set of rules which transitions between objects in \(D\) are allowed.

Definition 1.2 If \((D, R)\) is a formal system, let \(X \rightarrow_R Y\) mean that \(X\) and \(Y\) are in \(D\) and that transition from \(X\) to \(Y\) is allowed by the rules \(R\).

Definition 1.3 \(X \rightarrow^*_R Y\) mean that, there exists a finite sequence \(X_1 \ldots X_n\) of items in \(D\) so that \(X = X_1 \rightarrow_R X_2 \rightarrow_R \ldots X_n = Y\).

Each \(X\) in \(D\) is a term is formal system, they can be a variable, function or predicate.

Clause form

Logic programming language such as Prolog are often use clause form. First order clause form as the following

\[ \forall x_1 \ldots \forall x_n (A_1 \lor A_2 \lor \ldots \lor A_m \lor \neg B_1 \lor \neg B_2 \lor \neg B_k) \], where \(A_1 \ldots A_m\) and \(B_1 \ldots B_k\) are
atoms, and \( x_1 \ldots x_n \) are variables within those atoms.

We can rewrite in the clause form as
\[
A_1 \ldots A_m \leftarrow B_1 \ldots B_k
\]

There are three type of horn form

A definite program clause, which contain a head \( A \) and a body of sequence of atoms \( B \).
\[
A \leftarrow B_1 \ldots B_l
\]

An unit clause form, which only has a head \( A \).
\[
A \leftarrow
\]

And the definite goal, which only has a body without head.
\[
\leftarrow B_1 \ldots B_l
\]

2.1.2 Unification and SLD resolution

Unification is the heart of deductive logic programming, as shows in formal system \( X \rightarrow_R Y \), the purpose of unification is to find a series of substitution of two atoms. SLD resolution is an approach to use unification to find the final result of the programme. In the following sections, we will first discuss substitution of atom, which is the first approach to unification, later we will look at unification and SLD resolution.

Substitution

Variables bound in atoms can be substituted by other terms, which could lead into a new atom.

Definition 1.4 If \( N \) is a term, and \( x_1, \ldots, x_n \) are variables, and \( M_1, \ldots, M_n \) are terms, then \( N[x_1 := M_1, \ldots, x_n := M_n] \) means the result is simultaneously replaced all occurrences of \( x_1, \ldots, x_n \) by terms \( M_1 \ldots M_n \) in term \( N \).

For example, let \( A = Q(x_1, x_2) \) and \( \sigma = [x_1 := Q(x_1, x_2), x_2 := x_1] \) , the result of \( A \circ \sigma \) is equal to \( Q(Q(x_1, x_2), x_1) \), because it only simultaneously substitute one time, such that the \( x_2 \), will be replaced by \( x_1 \) and should not be replaced by \( Q(x_1, x_2) \) at this substitution.

Unification

Unification is very similar substitution, the different is, unification use substitution to get two atoms equal to each other. For example let \( A \circ \sigma \rightarrow B \), \( C \circ \sigma \rightarrow B \), whereas \( B = D \). However, not every two atoms can be unified with each other. That is, for example if an atoms begin with a predicate not equal to the predicate in another atom, they are not unifiable.

There are many algorithms of unification, in this review, we will focus on the algorithm based on (Richardson, 1998)
CHAPTER 2. LITERATURE SURVEY

Simple case:

1. If two atoms contain different numbers of term, they are not unifiable
2. If two atoms are the same, no unifier is necessary.
3. If two atoms are not equal, and none of them contain a variable, then they are not unifiable

Recursive case:

1. Let \( n = m \), and they are greater than 1. We try to \( \text{unify}((S_1, \ldots, S_n), (T_1, \ldots, T_m)) \), we first find \( \text{unify}(S_1, T_1) \). If it fail, then the whole unification fail. If we succeed and get \( \alpha \), then we apply it to both two atoms and continue unify two atoms as \( \text{unify}((S_1, \ldots, S_n)\alpha, (T_1, \ldots, T_m)\alpha) \), if it fail, then the whole whole unification fail, otherwise if we find the most general unifier \( \beta \), then \( \text{unify}((S_1, \ldots, S_n), (T_1, \ldots, T_m)) = \alpha \odot \beta \)
2. If none of a previous situation is applied. Let \( S_1 = f(A_1, \ldots, A_j) \) and \( T_1 = g(B_1, \ldots, B_k) \), if \( f \) is not equal to \( g \), then it is not unifiable, otherwise \( \text{unify}((A_1, \ldots, A_j), (B_1, \ldots, B_k)) \)

SLD-resolution

Finally, we come to look at how to find the answer of a logic programme, we use an algorithm call SLD resolution. The algorithm we are going to look at is in (Richardson, 1998). Let \( G \) be a goal clause, and \( P \) be a set of k Horn clause forms. Shows as followings \( G \leftarrow A_1, \ldots, A_n \) and \( P = \{\{A_1 \leftarrow B_{11}, \ldots, B_{1m}\} \ldots\} \)

1. Select a subgoal \( A_i \), and clause \( C \) from \( G \) where \( 1 \leq i \leq n \), find \( A_i \) in the set \( P \), if it is false, then stop and the theory has been satisfied.
2. Use unification to find \( \theta \), the mgu of \( A_i \) and \( C \), if there is not exist a mgu, then stop, the theory is not satisfied, and refutation has been found.
3. Derive \( G' \leftarrow (A_1, \ldots, A_{i-1}, B_{11}, \ldots, B_{1m}, A_{i+1}, \ldots, A_n)\theta \)
4. Repeat Step 1.

The refutation is found, when then failure of finding mgu in the processes SLD-resolution.
2.2 Neural networks

2.2.1 Neural networks basic

What is neural network?

Human brain is constructed by a very complex neural network, inside our brain, there are huge numbers of cell, and each of them connects to each others. Beside, even the cells from our eyes to toes also connect to the brain either directly or via spine. They are interact with each others by sending and receiving signal to other cells, as the result, we have the ability of thinking, perception and reacting.

Each of these cells we call them neurons, as the large numbers of neuron connect to each other, they created a neural network. In biology study, each neuron contains a cell body, numbers of synaptic terminals and dendrites show in the following figure. Since in this project, we are tring to simulate a neural networks in computer system, it is not necessary to understand how it works in biological term. Moreover, it will give us a much nice overview on how it works if we refer them into computer science term.

![Figure 2.1: Neural network constrution](image)

We can imagine each neuron is a computer in the worldwide web. In a computer, there are lot of types of inputs, such as keyboard, joystick and also information from the worldwide web via a Lan cable. All of them can refer as dendrites of a neuron, the dendrites are the input of a neuron, they receive signal from other neurons. Come back to our computer world, inside the worldwide web, if a computer receiving information from other computers, it means some computers is sending information to it via a Lan cables, the output in computer is the same as synaptic terminals of neuron. As you can imagine the cell body is the computer itself, which computes all the signals from input and throws them out to the output.

An idea of a single neuron is simple as shows above, however the neural networks are very complicated, which even more complex than the worldwide web. We will come back to it later.
Neurons working together

It is not make sense a single neuron only works with its own, they are useful only when they interact with others, and it is worth to understand how they could work with together.

Every single neuron sends signals to the output and receives signals from input. Moreover, each neuron can receive input from more than one neurons, likewise they can send to more than one neurons. Signals send between neurons are named value. Assume there are two neurons \( j \) and \( k \), signals send from \( j \) to \( k \) is call value of \( j \) denote as \( V_j \). A connection between two units also has a value call weight, it can be changed by time, which plays a very important role in learning ability of neural networks. A signal sends from neuron \( j \) to \( k \) has a weight denote as \( W_{kj} \). Neuron \( k \) now receives a signal is now become \( V_j W_{kj} \). However it is still not the end of the story, each neuron has a value call threshold inside itself, which able to alter the final signal, denote as \( \theta \), the final signal is now become \( V_j W_{kj} - \theta \).

What happen if a neuron has more than one input? It is as simple as just one input. We only need to get the sum of all input and is subtracted by the threshold. For example, we have a list of neural \( \{j_1, j_2, \ldots, j_n\} \) and a neuron \( k \), which the sum of all signals is equal to \( \sum_{i=1}^{n} V_j W_{kj} \), after we get the sum, then we deduct it by the threshold and we have the final signal for this neuron. In (Haykin, 1998), it gives us another way to compute the final input signal without deduct the threshold at the end, which by creating an extra unit \( j_0 \), the output value of it is equal to -1 and the weight of the connection between it and the target neuron is equal to absolute value of threshold, moreover, the equation strat from 0 rather than 1 as \( \sum_{i=0}^{n} V_j W_{kj} \). This method will give us a same value and makes it more readable.

Neuron activation formula

In the real world, when a computer receives some information, it normally computes these data and throws it to an output, which is also the case in neural networks, and works very similar as a computer in the worldwide web. Every single neuron has its own activation formula, as the CPU unit in a computer. In a neuron, when they receive a signal, they compute these signals and send the result as an output value to the next neurons. There are many different types of formula, however in this dissertation, we are not going discuss them in depth.

2.2.2 Network structure

There are no magic in neural networks, they are powerful due to the complexity of their structure. Some of them can be simulate by computer, by some of them could not. The primary classifications of associative neural networks are feedforward and recurrent networks.
Feedforward networks

Most of these types of networks contain more than 2 layers, which must have an input and an output layer. Although this type of have 2 layers, it is named singlelayer feedforward networks. The reason of its name is because we do not count the input layer as a layer, since they are only external source which does not do any computation at all, beside it does not contain neurons, they are only nodes. The purpose of this layer is to push external signal to next layer.

A more common feedforward network usually has 3 layers, which has a hidden layer on the top of the 2 layers networks. Input layer usually get signals from external sources, and push it into the hidden layers. The hidden layers will collect these signals and compute them then push an output signal to the output layers. At the end, neurons in the output layer receive signals from the hidden layer and compute them with their activation functions, and then push a final output as the results. We call this type of feedforward neural network as multilayer feedforward networks, which they have more than one layer perform computation.

Recurrent networks

This type of networks different from the feeddeforward networks in the way which it must contain at least one feedback loops in the network. Feedback loops are everywhere in creature’s neural networks, they are common, but not difficult to simulate. A feedback loop can happen in a single neuron, which the input receive signals from the output the neuron itself.

2.2.3 Learning

Learning models is the heart of neural networks, although it is very important and flexible, the mechanism of them are still very simple. There are numbers of types of learning models,
this is done by changing the weight in connection between neurons. In this section, we will look at some types of learning models which can benefit in our topic.

**Hebbian learning**

This is the type of unsupervised learning, which means we leave the network to calculate the correct result by itself. In more general word, we assume there have two neurons $j$ and $k$, neuron $j$ send $V_j(t)$ to neuron $k$, and we have an output $V_k(t)$ from neuron $k$, then we can generate a function which can alter the weight of this connection, that able to alter the final signal receive from $k$. Shows as follows.

$$
\Delta W_{kj}(t) = F(V_j(t), V_k(t))
$$

$$
\Delta W_{kj}(t) = \eta(V_j(t), V_k(t))
$$

Where $\eta$ is a constant which determines the rate of learning. When $\eta$ is positive, which means Hebbian learning, otherwise it is an anti-Hebbian learning.

**Error-correction learning**

This is the type of supervised learning, the main difference between supervised and unsupervised learning is, we will supply an expect output for the networks, as the result, it will adjust the weight which able to give us the expect output. We discuss the processes based on (Komendantskaya, 2007).

Let $d_k(t)$ be a desired response for unit $k$ at time $t$, and $V_k(t)$ is the actual output from $k$ at time $t$. We define an error signal between this two value as $e_k(t) = d_k(t) - V_k(t)$. At this stage we supply an input $V_j(t)$, the weight will be adjust as follows.

$$
\Delta W_{kj}(t) = \eta e_k(t), V_j(t)
$$

Where $\eta$ is a constant which determines the rate of learning. Finally, we can get the next weight at time $t+1$ as

$$
W_{kj}(t+1) = W_{kj}(t) + \Delta W_{kj}(t)
$$

**Filter learning and Grossberg’s Law**

This is a type of unsupervised learning, this type of learning is very useful when a layer receive multiple inputs, which able to filter out signals that not useful at this time. The following processes are based on (Komendantskaya, 2007).

Let $V_1, V_2, \ldots, V_n$ be multiple input signal, where $V_i, i \neq n$ are “conditioned stimuli” and $V_n$ is an “unconditioned stimulus”, Grossberg assume that $V_i$ are equal to 0 most of the time, and took large positive value when it became active.

Choose some unit $c$ with income signal $V_1, V_2, \ldots, V_n$, we get

$$
W_{ci}^{\text{new}} = W_{ci}^{\text{old}} + a[V_i V_n - W_{ci}^{\text{old}}]U(V_i), \{1 \leq i \leq n\}
$$

where $0 \leq a \leq 1$, and $U(V_i) = 1$ if $V_i \geq 0$, otherwise $U(V_i) = 0$
Competitive learning and Kohonen’s layer

This type of learning often consider is an alternative of Hebbian learning, it defined as followings.

The layer consist N units, each units receives input value \( \{V_1, \ldots, V_n\} \) denote as \( V \), the input value \( V_j \) to layer unit \( i \) with weight \( W_{ij} \) denote as \( W_i \) of a vector of all weights between unit \( \{W_1, \ldots, W_m\} \).

It then calculate its input intensity \( I_i \), with formula \( I_i = D(W_i, V) \), which \( D \) is a distance measure function. After we get all \( I_i \), the value with the lowest intensity is set to 1, all other value in \( V \) will be set to 0.

2.3 Logic programming on neural networks

Deductive logic program can be done in many ways, in logic programming language such as Prolog, it uses a way call SLD-Resolution which has been mentioned in the previous section in this review. In (Onyeyiri, 2008), it presents a way to use to power of neural network to perform such operation.

The way to perform it in (Onyeyiri, 2008) is to generate a modified SLD neural networks which is based on the SLD neural networks in (Komendantskaya, 2007). This kind of network performs a very similar way of deduction as SLD-resolution, in fact the network simulates the procedures of SLD resolution, in this way, whereas result in same unification as SLD-resolution.

We first discuss the neural networks call \( T_P \) network, it is a first approach of SLD neural network. Later we will look at SLD neural networks.

2.3.1 \( T_P \) networks

Use \( T_P \) network can be easily constructed if the programme is small. Unlike SLD-resolution, it does not need substitution and unification, by making this happen, the networks need to generate all possible terms, such that, the size of this networks could become infinite. But since it works very straightforward, and can be constructed easily with only few simple rules, it is worth to understand its structure and the way it works.

To construct a \( T_P \) network, we do the following. For each definite propositional logic program P, there exists a 3-layers recurrent neural network built of binary threshold units that computes \( T_P \) network.

1. The input and output layers are binary threshold units of length m, which m is the number of all propositional variable in the programme, and all units in the input and output layers have threshold equal to 0.5.
2. Connect the output of each term to its input.

3. For each clause \( A \leftarrow B_1, \ldots, B_n \), we do the following.
   
   (a) Add a binary threshold unit \( c \) in hidden layer.
   
   (b) Connect \( c \) to the output of \( A \) with weight equal to 1.
   
   (c) For each \( B_i \), where \( 1 \leq i \leq n \), connect the input of \( B_i \) to \( c \) with weight equal to 1.
   
   (d) Set the threshold of \( c \) equal to \( n - 0.5 \).

### 2.3.2 Unification in neural networks

\( T_P \) works very straightforward, but a problem rise if the programme is not small, which the size of the network could become infinite as we mentioned in the previous section. To solve this problem, we need another approach to simulate SLD-resolution, since it has the mechanism of unification, then it is not necessary to generate all atoms.

Unlike \( T_P \) neural networks, this type of networks do not compute binary values, in fact, it can proceed any goal of a logic program. The way which dealt with goal gives us much more flexibility. Neural network can only dealt with numbers, but logic programs are strings, therefore we will need to covert string into numbers, this is the first thing we need to sort out, later we look at the important step of SLD neural network: unification of neural networks.

#### Godel encoding

Godel encoding is one of the approaches to covert a term into numbers, it is not the only possible encoding method, but since many paper use this type of encoding, so if worth to look at this this approach.

It is not difficult to covert an atom into numerical values with Godel encoding, there are several ways to do it, but we will consist with the papers we based on. For each atom, we generate a numeric values as follows.

#### Predicate threshold

Before we start to unify two atoms, we have to make sure they are using the same predicate, if they are not using the same predicate, it means they are not unifiable. Predicate threshold is an approach to determine if two atoms are unifiable.

The way to do it is simple, assume we have an atom \( g_1 \) is read to be unified with another atom \( g_2 \). We start to read the first Godel number of \( g_1 \), if it start with the number 4 follows by numeric value 1, we count the length of the 1 in \( g_1 \) and \( g_2 \), if they have the same length, it means it may be unifiable and will be send into the networks, and otherwise will
be blocked. In fact it will not be blocked, what we do is to set different value of weight between the two layers. The way to “block” the value will be specified in the next sections.

**Unification in neural network**

Unification is the heart of first order logic programming, it substitutes a goal to an atom, and we have seen how it can be done in mathematical ways in the previous section. Since it is one of the most important parts of SLD-resolution and SLD neural network, we are going to focus on how we can simulate these processes in neural network.

Unification in neural networks are based on error-correction learning model, we have clarified the procedures of unification of neural networks in (Komendantskaya, 2007) into 5 steps

1. Assume we have two layers $k$ and $h$ in the networks, a source input layer $j$ and an atom $gA$, the weight of connection between the node in layer $j$ and the neuron in layer $k$ is set by the predicate threshold. We assume there is an atom call $d_k(t)$, which is the desired value in neuron at time $t$, and the predicate threshold will $gA$ with $d_k(t)$, if the predicate of them are the same, weight between $j$ and $k$ will be set to the $gA$, otherwise 0.

2. Set $\theta_k = \theta_h = 0$, initial $W_{hk} = 0$. We can get $p_k = V_j(t)W_{kj}(t) - \theta_k$. Set $V_k = p_k(t)$, if $p_k \geq 0$, otherwise $V_k = 0$.

3. Set the errorcorrection value at time t as $e_k(t) = \text{disagreement}(d_k(t), V_j(t))$, which the function $s$ is substitute of the disagreement between this two term. Set $e_k(t) = 0$, if the disagreement between the two atoms is an empty set. Set $e_k(t) = -W_{kj}(t)$, if
the disagreement set between them is not a empty set and the substitution of it is an empty set, it means these two terms are not unifiable.

4. At this step we need to update our weight for the next recursion step. Set $\Delta W_{kj}(t) = e_k(t)$. We can get $W_{kj}(t + 1) = \text{substitution}(W_{kj}(t), \Delta W_{kj}(t))$ and $d_k(t + 1) = \text{substitution}(d_k(t), \Delta W_{kj}(t))$.

5. At the end, we are going to update the weight between layers $k$ and $h$, $W_{hk}(t + 1) = \text{concatenate}(W_{hk}(t), \Delta W_{hk}(t))$, if $\Delta W_{kj}(t) \geq 0$, otherwise $W_{hk}(t + 1) = 0$.

At this stage, the only thing we left is to read the result of the unification, which is the output value of the neuron in layer $o$. We determine $V_h$ by a value $p_h$. We get $p_h(t + \Delta t) = \text{substitute}(V_k(t + \Delta t), W_{hk}(t + \Delta t))$. Now we can determine $V_h(t + \Delta t) = W_{hk}(t + \Delta t)$, if $p_h(t + \Delta t) \geq 0$, and $V_h(t + \Delta t) = 0$ otherwise.

2.3.3 SLD neural networks

Finally, we are going to look at how to generate a SLD neural network, which is based on the processes in (Komendantskaya, 2007), we only describe it briefly in this review.

We start with a neural network with layers $k$, $h$ and $o$, there is a input layer $i$, which only contain nodes. Moreover, similar to the unification in neural networks, neurons in layer contain heads of clauses $d_k$ in program $P$. Each neuron in layer $k$ connect to one neuron in layer $h$. Next, each neuron in $h$ connect to numbers of neurons that represent the body of the clause. Finally, each neuron in layer $o$ connect to all neurons in layer $k$. The following steps show how to generate weight and value between layers.

1. Layer $k$ filters excessive signals from layers $i$, only one goal is allowed at once, this is done by applying Kohonen’s layer and Grossberg’s laws. Moreover the predicate threshold will ensure only one weight is being activated.

2. At this step, we apply unification to the network as the previous section in layers $i$, $k$ and $h$. Each neuron in layer $k$ should have one connection to the output of unification result.

3. If there is more subgoal or further unification, neurons from layer $h$ will pass a value 1 with the weight of the unification result to each connected units in layer $o$. Finally repeat step 1.

This is a very brief instruction, we will discuss more of it with implementation in the actual dissertation.
Chapter 3

Requirements

3.1 Requirements Scope

At the beginning of formal requirements, a general view of the requirements will be needed. This project intents to produce an API based on the thesis of (Komendantskaya, 2007) for logic programming interpreter or other related works. Therefore supporting functionality for the construction of SLD neural networks are vital. However, according to the (Komendantskaya, 2007), however there are some thesis may not be as important as the others, such as TP-networks and BAP, those concepts will not be discussed in this project, as they are not necessary for constructing SLD neural networks.

When discussing what functionality of the application programming interface should provide, we should consider which functionality for further development of logic programming interpreter will be required.

Base on (Komendantskaya, 2007), it concludes the following feature.

1. Neural networks constructor
2. Godel enumeration
3. SLD neural networks constructor

It is undoutable a neural networks constructor will be required for this project, however, there are some issues that may cause difficult of impleation of it, these problems should be discussed in the following sections. Since neural networks only take numbers as signals, a transition between atoms and numbers will be required, therefore Godel enumeration is a good way to do it. As soon as, connective neural networks can be constructed, the next step is to find the logic unification of the logic program, which is the a very important step in SLD neural networks. At the end, by using all functionality above, then we can construct the SLD neural netowks.
Despite we have discussed some of the features of the API, but we have to discuss the requirements of this project in a more formal way.

### 3.2 Requirements specification

The requirements specification has been divided into two sections, functional and non-functional requirements (Sommerville, 2004). Whereas the functional requirements is specified as “statements of service the system should provide, how the system should react to particular inputs and how the system should behave in particular situations”, moreover, the non-functional requirements are specified as “constraints on the services or functions offered by the system”.

Some of the requirements maybe difficult to be achieve, such as Godel enumeration and neural networks constructor, therefore, we may briefly describe some of the possible solutions in the requirements.

#### 3.2.1 Functional requirements

**Neural networks constructor**

It is no doubt a neural networks constructor is required in this API, as this project is mainly focus on solving first order logic in neural networks. There are numbers of neural networks APIs are provided freely on the internet, but none of them are fit into this project, that they are not support big numbers format, the reason for the use of big number will be discussed in the next section. Although most of them can be rewritten to support big numbers system, the change of support data type for a neural can result into huge work and difficult to manage, such that, we will focus on implement our own neural networks constructor.

This project is neither focus on pure neural networks nor logic programming, but the combination of two, as the result create a complete general use neural networks is not necessary at this project. However, the API still require the ability for constructing neural networks with the ability of learning.

Each neural network contains numbers of neurons, layers and connection between every neurons. Due to the structure of neural network has been described in literature review, therefore we will not discuss such structure in this section again. However, the action of neuron in usual neural networks has few difference compare to SLD neural networks, therefore it is necessary to construct neurons with different types.

As the result, we can conclude the following requirements for a general neural networks.

1. The constructor must be able to create neurons with normal or SLD types.
   
   (a) Each neuron must be to hold big number data.
CHAPTER 3. REQUIREMENTS

(b) Each neuron must be to receive input values.
(c) Each neuron must be to send output values to other neurons.
(d) Each neuron must be to compute output value with its activation function.
(e) Each neuron must be able to form connection with other nodes.
(f) Each neuron must be able to hold weights between connections.
(g) Each neuron must be able to perform different types of learning models.

2. The neural networks must be able to contain layers.
   (a) Neural networks must be able to proceed layers.
   (b) Neural networks must be able to proceed neurons.
   (c) Neural networks must be able to proceed their result.

3. Each layer must be able to hold both normal and SLD types neurons.
   (a) Layers must be able to perform different types of learning models.

3.2.2 Godel enumeration

In the previous chapter, we have discussed the usage of Godel enumeration, however this
enumeration is not a practical system, the reason of it will be discussed in the design chapter.
However we will briefly discuss its structure. The biggest difference between the original
Godel enumeration and modify Godel enumeration is instead of use the numbers of 1 after
the type value, the modify type will use a two digit number. Therefore we conclud the
following requirements.

1. A struture of modified Godel number.
   (a) The structure must be divided into numbers of 3 digits terms.
   (b) Each term must contains a term type.
   (c) Each term must contains an index.
   (d) The structure must be able to link with the next structure.
   (e) The structure must be to transit into complete numerical value.

3.2.3 SLD neural networks constructor

This constructor intends to general SLD neural networks base on logic program in a simple
way. Such procedures have been specified in the literature review, moreover we will discuss
more deeply in the design chapter. However, we will specify the requirements as the
followings.

1. The API must be able to store logic terms.
2. The API must be able to store atoms.
3. The API must be able to build goals based on atoms.
4. The API must be able to construct a partial SLD-neural networks based on each goal.
5. The API must be able to construct a complete SLD-neural by using all partial SLD neural networks.

3.3 Non-functional requirements

3.3.1 Compatibility

In this project, we intend to create a cross-platform API. Therefore, we concluded the following requirements. Moreover, since we are going to create an application programming interface, such that it must be reusable.

1. The API must be able to work on multiple platforms.
2. The API must be able to reuse by other users.
3. The API may be able to proceed fast enough to make sure it is valuable.
Chapter 4

Design

4.1 High level design

As we discuss in previous chapters, the API is design for numbers of different purpose, especially on neural network base AI and first order logic interpreters. However, it supports numbers of different features, but most further works which based on this API can be created with a very similar approaches. These approaches are shows as the following graphs

The only different in this two figures is, in the first graph, we use SLD neural network to construct a back end of the first order logic interpreter, whereas the second figure illustrate manually construction of a neural network by only using the neural network constructor. However, they looks different, but they are practically the same, because it is not necessary to construct SLD-neural network by using the SLD network constructor, in contrast we can manully create such network as the second figure manually, which has been described in literature review.

Consider the first figure. In this figure, users intends to manually build neural networks, these networks can be either general neural networks or SLD neural networks. Before using the API, users should have the design of the neural networks already, after that user can create neurons, layers, learning models and activation functions by using the support features of the API, all these features are wrapped into the neural networks constructor. Laterly, user can proceed the neural networks for any purpose.

Now look at the first figure again, which the user intends to build a first order logic programming interpreter by using the SLD neural network constructor. Before the users try to use the API for this purpose, they should have a front end of the interpreter ready and can be adopt into the API. However it is not necessary to change to front end to adopt to the API directly, because users are encourage to construct an intermedia language by following the compiler techniques (Appel, 1998), we will not look at front end and any type of intermedia language in this project, because they are compiler technique, that is not relate
CHAPTER 4. DESIGN

Figure 4.1: Neural network construction

Figure 4.2: Interpreter construction
in our topic. When users have an intermedia language that adopt to our API, the SLD neural network constructor will automatically generate a correspond SLD neural network base on the logic program. After this step, the interpreter is basically done and ready to use, other users are now ready to ask questions in this program, and the interpreter should come up with correct answers.

4.2 API structure

At the beginning, we first look at the entire structure of the API, then we will look at the designs of each part of the API. The API contains three main structure for constructing general and SLD neural networks, and numbers of features supporting it. Shows as the following graph

![API structure diagram](image)

Figure 4.3: API structure

Main structures are represented as square, and ellipse are support features. It goes through from the top to the bottom, where dash lines are optional usages for the construction of neural networks, and real line represent necessary usages. We discuss each feature step by step from the top to the bottom of the previous figure.

At the very top level the graph, the modify Godel encoding, as mentioned in previous chapter, this project is will be used modified Godel encoding instead of Godel encoding.
In this project we will use this type of structure to represent logic terms, atoms and goals. However, it is not necessary to use this structure to create general neural networks other than SLD-neural networks, such that, we use a dash line instead to real line to form a link between it and to next next structure. As it is the one of the main part of the SLD neural network, we will discuss this topic into deeper detail in the following section.

Although modified Godel encoding fixed the big usage of memory problem of the original Godel encoding, it still require suffient memory to hold the new type of Godel encofing. A 32-bits or even a 64-bits integer is still far not enough for it, on the other hand we require another struture that can hold lager number up to 128-bits or even bigger, the reason of it will be discuss in the following sections. There are number of reusable package for handling larger number algebric on the internet, we chose decNum which available on (DecNumber, n.d.) for our project, because it offer a very good reusable structure, fast algebraic computability and comparitily easy to use. Moreover, as big number is supported not only in SLD neural networks, any type of neural networks that require big numbers is also supported, therefore this API offer more flexibility than most of the neural network construction application on the internet.

Neural networks are complex system, they contain differnt layers, neurons, functions and so on, for this reason this project break all these parts into small pieces and allows users to create any type of neural networks more easily. On the top of it, we create a neural network constructor framework, this framwork including four features, which are two main structures and two support functionalities, which are represented as square and ellipse. We start our discussion at the top of the framework, which is the node structure. Node structure is a variant of neuron in a neural system, this structure has extend its ability to be more flexible than origianl neuron. This sturcture allows not only neuron type, in addition the SLD neuron type is also to be proceed in any type of neural system, in addition of the flexiable trainning and learning models, all of these features allow neural system to be easier to created and more flexiable controlability. As this structure is a vital part of our project, it will be discussed in more detail in the following sections.

At the next stage, we are going to look at the neural networks and layers structure. This strucrer intends to allow user to construct the entire networks protocol in a very simple way. Users do not need to understand actual structure of the implementation of the networks, whereas users can easily generate a protocol will his/her neural networks design in addition with the understanding of its tranning function. For more detail of this structure, it will be discuss in the follwing section.

Now, we coming to the two support features in the framework. Usually activation and tranning functions are defined in mathematical approaches, but this is not the case in our API, whereas we ask users to define these functions in C language implementation. The reason of this is, although most of all of these functions and models can be defined in a mathematical structure, but any two of them can be entirely different with each others, even two different error-correction learning functions can be very different with each others. However, this statment can be argued by theory of learning of each learning models, which every learning models have their basic mathematical defined structures, recall at section
2.3, every learning model have their own mathematical defined structures, but this not the case in SLD neural network, recall at section 3.2.3 and 3.2.4, \( e_k(t) = d_k(t) \odot V_j(t) \), which use a disagreement instead of subtraction between desired value and incoming signal, that is different with original definition of error-correction model. For this reason, we do not want to fix the structure of all these learning models and allow user to implement all these model by their own wish with maximum flexibility, moreover this reason is also apply to activation functions.

Now we are ready to discuss the SLD neural networks constructor. The SLD neural network constructor is a auto SLD neural network generator, it allow use construct a SLD neural network in a very simple way. Users do not need even need to understand the actual implementation of the SLD neural networks, the only thing they need is to insert every logic terms, atoms and goals into the API, after that, the system will return a corresponding SLD neural network. As this is the main feature of the project, we will discuss into more depth in the following sections.

Finally, we brief look at the utility package of the API. Utility API contain numbers of functions that help users to create, it simplified the direct function call from decnumber package by creating a simplier function. Although function in this package is not necessary for creating neural system, but it is recommend and keep the further implementation tidy.

### 4.3 Modify Godel Encoding

Modify Godel encoding is an implementation base on the original Godel encoding, we denote modify Godel encoding as MGE and the original on as GE. Original Godel number was introduced by Kurt Godel for the prove of incomplete theorem, it was designed for mathematic use, by that time, electronic computer has not been invented. In this case, Godel number is not a pratical system for computer science. When discussing it is not designed for a pratical computer system, consider a logic program that has up to hundred variables, it can takes more than 324 bits for a single variables. The consequence of implementing it into a practical system may result in run out of memory, and complexities between these Godel numbers can easily get very high, as each computation has to count the numbers of numbers from the highest decimal place to the lowest decimal place. Although the computation of it is only linear, but the consumtion of memory is still high, moreover these problems can be solved by creating a similar enumeration system to get better result.

This solution is call modify Godel number. The soulution is by replacing the numbers of 1s afther the type number by a two digit index. This approach can be illustrated by the following table

<table>
<thead>
<tr>
<th>Original Encoding</th>
<th>Modified Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101100</td>
<td>21101100</td>
</tr>
</tbody>
</table>

In the previous table, we can see all 1s after the type value has been replaced by a two digit index number. However, it is necessary to evaluate the usability of this approach before we decide to invoke it in our project. When look at the first two examples in the table, at the first example, the original encoding only use two digits for encoding, however it takes
Table 4.1: The change from original to modify Gödel encoding

<table>
<thead>
<tr>
<th>Term and atom</th>
<th>Gödel encoding</th>
<th>Modify Gödel encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>21</td>
<td>201</td>
</tr>
<tr>
<td>c2</td>
<td>211</td>
<td>203</td>
</tr>
<tr>
<td>c5</td>
<td>211111</td>
<td>205</td>
</tr>
<tr>
<td>c7</td>
<td>21111111</td>
<td>207</td>
</tr>
<tr>
<td>p3</td>
<td>4111</td>
<td>403</td>
</tr>
<tr>
<td>p3(c5, c7)</td>
<td>4111521111172111111116</td>
<td>4035002057002076000</td>
</tr>
<tr>
<td>p4(c4, c4)</td>
<td>411115211117211111116</td>
<td>4045002047002046000</td>
</tr>
</tbody>
</table>

three digits in the modify encoding. It indicates this memory usage in the modify encoding is higher than the original encoding, but we have to consider in more depth. The second example shows a more pleased result, that both encoding take three digits. The examples following by the second example are more progressive. All these four examples takes much less digits in the modify encoding then the original encoding, whereas it take 3 digits, 5 digits, 1 digit and 5 digits in examples 3, 4, 5, 6. These indicate modify Gödel encoding is valuable in some cases.

At this stage, we have to evaluate when this enumeration is valuable, and when it is not. We notice that, every single logic term takes at least 3 digits in the MGE, such that if any term takes 3 or more digit in the GE will be valuable, which means when any type of term has more than one term, MGE will be valuable. When encode atom, the memory usage is depended on terms, such that when MGE is valuable when encoding terms, it is also valuable when encoding atoms. Most logic programs have more than one predicate, constant and variables, for this reason, we can state that MGE is valuable in this system. We list all the types in the following table

Table 4.2: Types and encoding of types

<table>
<thead>
<tr>
<th>Type in logic programming</th>
<th>Type in encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>1</td>
</tr>
<tr>
<td>Constant</td>
<td>2</td>
</tr>
<tr>
<td>Function</td>
<td>3</td>
</tr>
<tr>
<td>Predicate</td>
<td>4</td>
</tr>
<tr>
<td>)</td>
<td>5</td>
</tr>
<tr>
<td>,</td>
<td>6</td>
</tr>
<tr>
<td>⊕</td>
<td>7</td>
</tr>
<tr>
<td>⊖</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

In the GE, only variable, constant, function and predicate have 1s follow by their types,
other types such as open bracket, close bracket and so on do not have 1s follow by their type values. However as we mentioned before, every terms have an index follow by their types including open bracket and so on. The reason of this is because, it is an good approach to be precise for each type. However it is not necessary, in addition it will cause more memory, this can be reevaluate in the future. Although it may cause more memory sometimes, when look at the final example in table 4.1, every terms in the atom consume 1 digit less in MGE, but both MGE and GE take 18 digits, which means MGE is still valuable at most of the time.

In our previous case, the largest example takes 18 digits, which takes about 60 bits, this means a long type of C language is able to hold this value. But consider a predicate which takes 3 arities, which is very common in logic programming, this encoding will takes about 80 bits, therefore long type is far not enough at this time. For this reason, we assume a 128 bits integer should be just enough for encoding, therefore we use a package which able to handle 128 bits integers, therefore we use decNumber package which available on the internet for MGE. Now we conclude the design of MGE.

4.4 Design of node in neural network constructor

Nodes are variant of neuron of neural networks, however we have to consist nodes precise as neuron, recall at functional requirements of neuron, the following graphs shows the design protocol

It contains numbers of members, we discuss each members from the top to bottom. The identification member is the identity of each neuron, every nodes have only one identity, in addition it must not be duplicated with other node’s identities. The next member is potential, which has been defined at literature review, it collects incoming signals from other nodes, which the potential is equal to the sum of all signals and subtract the threshold of the fifth member of the node. Finally we discuss the members left, which are output_value,
weight and activation function, the output_value depends on its activation_function, in addition the signal sends to the next node is equal to output_value multiply weight, this has been defined at literature review.

However, the protocol design is not enough for a more complex network, consider a node has multiple input nodes or multiple output nodes, in the protocol design, it only has one input value from input node, and one weight value to output node, this simply not enough. For this reason, we extend our protocol to the following design

![Figure 4.5: Second design of node structure](image)

In the newer design, we replace some members by two new structures, the input_table and output_table. Input_table contain a direct link to input nodes, the weights between current node and each input nodes and the input value from input nodes. The brackets of each members represent array of the members, which this allows multiple input connections, therefore it fix the problem of only one input node is allowed in the protocol design. Output_table is very similar input_table, it allow nodes to form a multiple connections to other nodes, the structure contain the direct to output nodes and the weights of each connections. As the result, potential can be collect from multiple input nodes from the input_table, similarly, multiple signals can be send through output_table.

Consider the following graph

The network process from node A through D, at the beginning, node A sends data to nodes B and D, then nodes B compute the output data by its activation function with the argument of the input signal from A. At the second stage, node B sends its output to node C, but as node C has not receive from node D, therefore, the input value of node C is not complete, in addition node C can not compute a correct output of node C. This can be fixed by extend the previous node structure shows as the following graph

This new structure will solve the previous problem, which every node has a working_state, when working_state is at waiting state, which means, it still need to compute its output data and sent it to its output nodes, however, this still depends on whethere if has recevied all data from input nodes, this is the reason we insert a new member data_receive_state
Figure 4.6: Neural network example

Figure 4.7: Third design of node structure
in input_table structure. If a node receive all data and still in waiting state, then this node is allow to to comput output data and send to output nodes, otherwise it skip the process and run the next node then start a new loop. As the result, all data receive is preserved before compute output data.

Finally we extend node structure that allows learning functions to compute the weights between connections as the following structure

![Figure 4.8: Final design of node structure](image)

Error_correction function and function are design for process error correction learning model, and hebbian_data and function are design for process hebbian learning model, moreover extra_data is design for general usage, for example it can be use as desire value and other purpose, because it is not necessary to use any type of learning model, therefore, we has two members that state if learning models are active. Now we finally conclue the design of node struct.

### 4.5 Design of neural network in neural network constructor

Now, we need to consider insert nodes into layers and form a complete neural network. Most networks have more than one layers, in addition, each layer contains numbers of neurons, however in out case is nodes, it should not change the nature of neural network. Moreover, layers should allow learning models such as Grossberg’s law and Kohonen’s layer, therefore we create a following structure

As shows in the structure, every layer has a name for identification, it allow duplication, but it is not recomend. In addition, in contains direct links to each nodes in the layer, which allows layers to read and write data on nodes, more importantly allows layers to process nodes. The last four members are Grossberg’s law and Kohonen’s layer as the same as learning models in nodes’ structure. Now coming to the neural network structure, the first members indicates it allows multiple layers in neural network structure, consider
the last two members, they are created for the purpose of networks processing. Recall in the previous chapter, we process node A through D, latterly we start a new loop at neuron D, this is the use of last two members. We discuss networks process into more depth later in this section.

Firstly, we discuss the high level structure between nodes, layers and neural networks. This can be illustrated at the following graph

In this high level structure, the neural network contains m layers, moreover each layer contains numbers of nodes. N nodes form j connections, which each connection does not
affect the layer level, which means a node in layer i can form a connection to a node in any layer k.

Return to network processing, recall at the previous section, we process each node to complete the interval of the network. However, we change the process procedures at this time, now we process layers before process nodes. This is a better approach compare to the previous processing, because in most networks, values of nodes send from one layer to the next layer, which make sure most nodes in a layer have computed before going to the next layer. Moreover, this new processing also allows learning models such as Kohonen’s layer can be processed. Now we update the network processing as the following graph.

Now we go through the new processing approach, at the beginning, the network contains 4 nodes and the process_count is set to 0. Layer i is being processed, if any learning model in the layer is actived, the model should change the weight between node A with B and D, then layer i will go through each nodes in this layer, which only node A is in this layer, whereas node A will compute its output value, afterward, it sends signal to nodes B and D, the actual signals are depend on the weights between AB and AD, moreover the process_count is set to 1. At this stage, the layer h is being processed, in contrast in the previous example, only node B is being process, now both node B and D are being processed at this time, then the process_count is set to 3. Finally, layer o is being processed, also node C is being processed either. Since node C has received all values that required to compute its output value, therefore we can set the process_count 4, and process_count is equal to the numbers of nodes in the network, therefore we complete the entire process of one interval without any loops. It is easy to see that, this approach require less computation than the previous one and effectively aviod uncessary loops. Now we have concluded the design of neural network and layer structure, and we are ready to discuss the SLD neural network constructor.
4.6 SLD neural network constructor

As mentioned in previous chapters, this project is not going to develop an interpreter, however it must prove sufficient to support future interpreter development, as the result, we provide this functionality for convert atoms to neurons and construct a partial SLD-neural networks. In order to achieve that, user must define the type for every terms, such as functions, predicates, variables and so on, the API will return an identification of the term, finally, these identification will be used for create atoms and goal, whereas the system will construct a corresponding neuron and partial SLD-neural networks. It will be more clear with an example as follow

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>term ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>father</td>
<td>predicate</td>
<td>401</td>
</tr>
<tr>
<td>grandfather</td>
<td>predicate</td>
<td>402</td>
</tr>
<tr>
<td>test</td>
<td>predicate</td>
<td>403</td>
</tr>
<tr>
<td>tom</td>
<td>constant</td>
<td>301</td>
</tr>
<tr>
<td>dave</td>
<td>constant</td>
<td>302</td>
</tr>
<tr>
<td>peter</td>
<td>constant</td>
<td>303</td>
</tr>
<tr>
<td>X</td>
<td>variable</td>
<td>101</td>
</tr>
<tr>
<td>Y</td>
<td>variable</td>
<td>102</td>
</tr>
<tr>
<td>Z</td>
<td>variable</td>
<td>103</td>
</tr>
</tbody>
</table>

Where the IDs are automatically generate by the API, user cannot manually change them. At this point, user may create atoms by using these IDs, in addition with a predicate and function construction function provided by the API, similarly, it will return a identification of every atoms and generate a corresponding neuron, shows as follow

<table>
<thead>
<tr>
<th>Atom</th>
<th>predicate ID</th>
<th>arity IDs</th>
<th>numbers of arities</th>
<th>atom ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>father(tom, dave)</td>
<td>401</td>
<td>301, 302</td>
<td>2</td>
<td>a1</td>
</tr>
<tr>
<td>father(dave, peter)</td>
<td>401</td>
<td>302, 303</td>
<td>2</td>
<td>a2</td>
</tr>
<tr>
<td>father(X, Y)</td>
<td>401</td>
<td>101, 102</td>
<td>2</td>
<td>a3</td>
</tr>
<tr>
<td>father(Y, Z)</td>
<td>401</td>
<td>102, 103</td>
<td>2</td>
<td>a4</td>
</tr>
<tr>
<td>grandfather(X, Z)</td>
<td>402</td>
<td>101, 103</td>
<td>2</td>
<td>a5</td>
</tr>
<tr>
<td>test(tome, dave, peter)</td>
<td>403</td>
<td>301, 302, 303</td>
<td>3</td>
<td>a6</td>
</tr>
</tbody>
</table>

At the stage we construct each goal use a similar method as the previous example, and similarly, this approach will generate some part of the complete SLD-neural networks, and return a corresponding ID, show as follows
### Table 4.5: Construction of goals

<table>
<thead>
<tr>
<th>Goal</th>
<th>Head ID</th>
<th>Definite goals IDs</th>
<th>numbers of definite goals</th>
<th>goal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>father(tom, dave)</td>
<td>a1</td>
<td></td>
<td>0</td>
<td>g1</td>
</tr>
<tr>
<td>father(dave, peter)</td>
<td>a2</td>
<td></td>
<td>0</td>
<td>g2</td>
</tr>
<tr>
<td>grandfather(X, Z) ←</td>
<td>a5</td>
<td>a3, a4</td>
<td>2</td>
<td>g3</td>
</tr>
<tr>
<td>father(X, Y), father(Y, Z)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finally, by combine all thses partial SLD-neural networks, we finally get a complete SLD-neural networks.
Chapter 5

Implementation and Testing

5.1 Language choice

In this project, we are going to implement our API in C language. However, neural networks system is relatively simple and suitable to be implemented in object oriented languages such as Java and C++. It is because, each object of the language can be referred into neurons in neural networks, moreover object oriented languages are intend to be implemented to be reusable. For this reason, object oriented languages are very suitable for this project.

However the reason for implement this project in C languages is because it provides the fastest runtime process. In most neural networks system, they usually contains large number of computation, therefore a fast procecess of neural networks system is required, moreover C language also provides a very good flexibility for implementation, this is also very important for such complex system. When consider the reusability of the language, which is more important that relies on the programmer rather then the language, that means a good implementation of the program can always maintain reusable without the help of object oriented theories. For all this reason, we chose C for the implementation language of this project.

5.2 Examples of the construction of neural networks by using the API

5.2.1 A simple neural network

This example demonstrate the procedure of manually construction of a simple neural network by using the API, we illustrate the example with the following code.

```c
init_table();
```

Listing 5.1: A simple neural network
```c
init_stack();
init_global_set();
n_network *neta = (n_network *)malloc(sizeof(n_network));
init_network(neta);
create_layer("I", neta, NULL, NULL);
create_layer("H", neta, NULL, NULL);
create_layer("O", neta, NULL, NULL);
node *a = NULL, *b = NULL, *c = NULL, *d = NULL;
a = init_node(nchange_pass, 0, NODE_TYPE, NULL, NULL);
b = init_node(nchange_pass, 0, NODE_TYPE, NULL, NULL);
c = init_node(nchange_pass, 0, NODE_TYPE, NULL, NULL);
d = init_node(nchange_pass, 0, NODE_TYPE, NULL, NULL);
form_link(a, b, create_decQuad_int(1));
form_link(a, d, create_decQuad_int(1));
form_link(b, c, create_decQuad_int(1));
form_link(d, c, create_decQuad_int(1));
a->out = create_decQuad_int(1);
a->work_state = READY_SEND_WSTATE;
insert_node(a->id, "I", neta);
insert_node(b->id, "H", neta);
insert_node(c->id, "O", neta);
insert_node(d->id, "H", neta);
perform_network(neta);
```

This example demonstrates the procedure of manually constructing a neural network. First, we should always initialise the environment from line 1 to 3, it is necessary whereas these steps make sure all pointers are pointing to a correct address, which can effectively avoid some of the runtime errors and memory leaks. Then we create and initialise a neural network protocol at line 4 and 5, in addition, create layers I, H, and O at line 6, 7, and 8. At this stage, an empty neural network with 3 layers has been constructed. Now we need to generate some nodes, from line 10 to 14, we create nodes A, B, C, and D, the first argument of the `init_node` function represents the activation function, which has already been implemented in the `i_fn_set` header, the user can define their own activation functions if required, the `nchange_pass` is the function that sends the same output value as the input signal to their output nodes. The next argument represents its threshold, then the type of node is the next argument, there are only two types of nodes in our project; they are node type and std node type, the difference between them is how they send signals to the next. For node type, the signals send to the
next node is equal to $W_k \ast V_k$, on the other hand the signals send from SLD node is equal to $W_k \odot V_k$. Finally, the last two arguments are error-correction and Hebbian functions, due to we are not going to use any learning model in this example, theredfore we set both functions to null.

Next, we can form connections between nodes, which is done by using the form_link function. From line 16 to 19, we form a network as figure 4.6. The first argument of the form_link function is the node users expect to be the input node, whereas the second argument is the target node, and the last argument is the weight between the connection. Afterward we give node A a output value and set node A’s work state to READY_SEND_WSTATE, this means node A will send its output value immediately at the start of the network process. Finally we put each node into its corresponding layers as figure 4.11. In deafult all learning models are set inactive state in every nodes and layers, therefore no learning models are going to run in the network.

Then we are ready to proceed the network, this is done by apply the network to perform_network function. Finally we can get output from each node, and we get the following result

```
Listing 5.2: Result of the simple neural network
output: a:1  d:1  b:1  c:2
```

We get a correct result which is illustrated as the the following graph
As we can see, the answer is exactly match the graph in the first iteration, this means this result is correct. Now we are going to look at the next example.

### 5.2.2 Encoding example

In this example, we are going to discuss at how to create logic terms and atoms manully with MGE, which start with the following code

```
Listing 5.3: Encoding example
1 init_table();
2 init_stack();
3 init_global_set();
4 m_godel *p1 = create_m_godel(PREDICATE_ENTYPE, 1);
5 m_godel *p2 = create_m_godel(PREDICATE_ENTYPE, 2);
6 m_godel *f = create_m_godel(FUNCTION_ENTYPE, 1);
7 m_godel *c1 = create_m_godel(CONSTANT_ENTYPE, 1);
8 m_godel *c2 = create_m_godel(CONSTANT_ENTYPE, 2);
9 m_godel *v1 = create_m_godel(VARIABLE_ENTYPE, 1);
10 m_godel *v2 = create_m_godel(VARIABLE_ENTYPE, 2);

//atoms
13 //g1:  p1(c1,c2)
14 //g2:  p1(c1,c2)
15 //g3:  p1(v1,c2)
16 //g4:  p1(c1,v1)
17 //g5:  p1(v1,c1)
18 //g6:  p2(v1,c2)
19 //g7:  p1(v1,v2)
20 //g8:  p2(v1,c1)
21 //g9:  f(v1,c2)
```
At the first 3 lines of the code, we always initialise the environment, afterward, we can create logic terms with the `create_m_godel` function. This function takes two arguments, the first one represents the type of MGE, such as predicate, constant and so on, the second argument represents the index of the logic term. From line 4 to 9, we create one predicate, one function, two constants and two variables. As we have all the logic terms we require, then we can use them to create atoms, we can use the `create_atom` function to satisfy this need. The `create_atom` function takes multiple arguments, because atoms always begin with predicate or function type, such that the first argument of the function must be any logic terms with predicate or function types. The second argument takes an integer, which is the numbers of arities. Finally we put the arities of the atom into the function arguments, however this function can take unlimited arguments, such that the atom is able to take multiple arities.
Now, we can transfer each atom into integer, which is illustrated from line 33 to 42, finally we can get view the result as integer, the following are the encodings of each atoms

Listing 5.4: Result of the encoding example

g1: 401500201700202600
g2: 401500201700202600
g3: 401500101700202600
g4: 401500201700101600
g5: 401500101700201600
g6: 402500101700202600
g7: 401500101700102600
g8: 402500101700201600
g9: 301500101700202600
g10: 101

The result is completely correct. Now we can start to look the next example.

5.2.3 Unification in neural network

The following example demonstrate the unification in neural network, which followed the example base on (Komendantskaya, 2007). Firstly, we start this demonstration with the following code

Listing 5.5: Unification in neural network

```
init_table();
init_stack();
init_global_set();
n_network *unification = (n_network
    *)malloc(sizeof(n_network));
init_network(unification);
create_layer("S", unification, NULL, NULL);
create_layer("K", unification, NULL, NULL);
create_layer("H", unification, NULL, NULL);

m_godel *p1 = create_m_godel(PREDICATE_ENTYPE, 1);
m_godel *c1 = create_m_godel(CONSTANT_ENTYPE, 1);
m_godel *c2 = create_m_godel(CONSTANT_ENTYPE, 2);
m_godel *v1 = create_m_godel(VARIABLE_ENTYPE, 1);
m_godel *v2 = create_m_godel(VARIABLE_ENTYPE, 2);
m_godel *g1 = create_atom(p1, 2, c1, c2);
m_godel *g7 = create_atom(p1, 2, v1, v2);

data dg1 = get_value(g1);
data dg7 = get_value(g7);

node *i = NULL, *k = NULL, *h = NULL;
i = init_node(always_send_one, 0, NODE_TYPE,
    error_correction, i, NULL);
```
23 \textit{k} = \texttt{init\_node(\texttt{always\_send\_one}, 0, \texttt{NODE\_TYPE}}  \\
\quad \textit{.error\_correction\_k, NULL);}  \\
24 \textit{h} = \texttt{init\_node(\texttt{unchange\_pass}, 0, \texttt{SLD\_NODE\_TYPE}, \texttt{NULL, NULL);}  \\
25 \texttt{add\_extra\_data(k, copy(dg7));}  \\
26 \texttt{k->error\_correction = ACTIVE;}  \\
27 \texttt{k->error\_correction = ACTIVE;}  \\
28 \texttt{form\_link(i,k,copy(dg1));}  \\
29 \texttt{form\_link(k,h,create\_decQuad\_int(0));}  \\
30 \texttt{insert\_node(i->id, "S", unification);}  \\
31 \texttt{insert\_node(k->id, "K", unification);}  \\
32 \texttt{insert\_node(h->id, "H", unification);}  \\
33  \\
34 \textbf{while}(1)  \\
35 \quad \{  \\
36 \quad \texttt{perform\_network(unification);}  \\
37 \quad \texttt{i->work\_state = WAITING\_WSTATE;}  \\
38 \quad \texttt{k->work\_state = WAITING\_WSTATE;}  \\
39 \quad \texttt{h->work\_state = WAITING\_WSTATE;}  \\
40 \quad \texttt{unification->process\_count = 0;}  \\
41 \quad \}  \\
42 \quad \texttt{printf("result:\n", decQuad\_to\_string(h->out));}  \\
43  \\
This example looks like the combination previous two example, therefore we will not 
duplicate the explanation. Despite this example looks much similar to the previous 
example, however due to the different structure of the networks, the results are also com-
plete different. However, we will discuss the difference between the current example and 
the previous example. Consider the code from line 20 to 26, we use error-correction model 
in node i and k, moreover we has set the learning models of these two nodes as active, 
therefore weights in these two nodes can be altered by the learning models. Because we 
need to use error-correction model, such that we set the desire value of node k to the MGE 
of $p_1(v_1, v_2)$. After that, we set the weight between node i and k to the MGE of $p_1(c_1, c_2)$, 
and now we are ready to proceed the network.

Firstly, at the first iteration node i send value 1 with weight $p_1(v_1, v_2)$ to node k, then 
according to the desire value of the node k, error-correction data of node i is set to the 
disagreement of $p_1(v_1, v_2)$ and $p_1(c_1, c_2)$, which is equal to the MGE of $v_1/c_1$. Moreover 
the weight between node i and k is now set to the substitution of $p_1(v_1, v_2)$ and $v_1 / c_1$, and 
now is equal to $p_1(c_1, v_2)$. Then we proceed node k, the weight between node k and h is 
also set to the MGE of $v_1 / c_1$. Afterward it sends the value 1 with its weight node h, and 
now we complete the first iteration.

Between line 38 to 41, we rest the work states of every nodes to waiting state and reset 
to process\_count to 0, these few lines allow the network proceed the next iteration. At the 
second iteration, the new error-correction data is now set to the MGE of $v_1 / c_1 \oplus v_2 / c_1$, in
addition the weight between node i and k is not set to $p_1(c_1, c_2)$. Similar to the previous iteration weight between node k and h is again set to the MGE of $v_1/c_1 \oplus v_2/c_1$. Now node h receive the value 1 with the new connection weight between node k and h. Finally, we have the correct result, which is also equal the MGE of weight between node k and h. The following is the result of the network

Listing 5.6: Result of unification in neural network
result: 800101900201800102900202

Since we have to correct answer, no more iteration is require, and we have completed this example.

5.2.4 First example of SLD neural network

In this example, we demonstrate the procedures of manually construction of SLD neural networks, firstly we state the following logic program

Listing 5.7: Logic program
\[
\begin{align*}
\text{father} & (\text{tom}, \text{dave}) . \\
\text{father} & (\text{dave}, \text{peter}) . \\
\text{grandfather} (X, Z) : & \neg \text{father} (X, Y), \text{father} (Y, Z) . \\
\text{question} : & \text{grandfather} (X, \text{peter}) . \\
\end{align*}
\]

\[
\begin{align*}
a_1 & = \text{father}(\text{tome}, \text{dave}) \\
a_2 & = \text{father}(\text{dave}, \text{peter}) \\
a_3 & = \text{father}(X, Y) \\
a_4 & = \text{father}(Y, Z) \\
a_5 & = \text{grandfather}(X, Z) \\
a_6 & = \text{question}
\end{align*}
\]

Base on this logic program, we can construct the following SLD neural network as the following graph
This graph is designed based on the thesis (Komendantskaya, 2007), however in this chapter, we will not discuss the design of the network, because these procedures have been discussed in literature review. At this stage we will start to discuss the manually implementation of the SLD neural network. We start to review the following code:

```
Listing 5.8: Manually construction of SLD neural network

1(init_table());
2(init_global_set());
3(init_stack());
4n_network *pro1 = (n_network *)malloc(sizeof(n_network));
5init_network(pro1);
6create_layer("S", pro1, NULL, NULL);  
7create_layer("K", pro1, kohonen_layer_fn_k, gross_law_fn_k);
8create_layer("H", pro1, NULL, NULL);
9create_layer("O", pro1, NULL, NULL);
10get_layer(pro1, "K")->kohonen_layer = ACTIVE;
11get_layer(pro1, "K")->gross_law = ACTIVE;
12m_godel_father = create_m_godel(PREDICATE_ETYPE, 1);
```
m_godel *grandfather = create_m_godel(PREDICATE, 2);
m_godel *tom = create_m_godel(CONSTANT, 1);
m_godel *dave = create_m_godel(CONSTANT, 2);
m_godel *peter = create_m_godel(CONSTANT, 3);
m_godel *X = create_m_godel(VARIABLE, 1);
m_godel *Y = create_m_godel(VARIABLE, 2);
m_godel *Z= create_m_godel(VARIABLE, 3);
m_godel *M = create_m_godel(VARIABLE, 4);

m_godel *a1 = create_atom(father, 2, tom, dave);
m_godel *a2 = create_atom(father, 2, dave, peter);
m_godel *a3 = create_atom(father, 2, X, Y);
m_godel *a4 = create_atom(father, 2, Y, Z);
m_godel *a5 = create_atom(grandfather, 2, X, Z);
m_godel *question = create_atom(grandfather, 2, M, peter);

data deca1 = get_value(a1);
data deca2 = get_value(a2);
data deca3 = get_value(a3);
data deca4 = get_value(a4);
data deca5 = get_value(a5);
data decq = get_value(question);

node *i = NULL;
node *k1 = NULL, *k2 = NULL, *k3 = NULL;
node *h1 = NULL, *h2 = NULL, *h3 = NULL;
node *o1 = NULL, *o2 = NULL;
i = init_node(always_send_one, 0, NODE_TYPE ,error_correction_i_second, NULL);
k1 = init_node(always_send_one, 0, NODE_TYPE ,error_correction_k_second, NULL);
k2 = init_node(always_send_one, 0, NODE_TYPE ,error_correction_k_second, NULL);
k3 = init_node(always_send_one, 0, NODE_TYPE ,error_correction_k_second, NULL);
h1 = init_node(nchange_pass, 0, SLD_NODE_TYPE, NULL, NULL);
h2 = init_node(nchange_pass, 0, SLD_NODE_TYPE, NULL, NULL);
h3 = init_node(nchange_pass, 0, SLD_NODE_TYPE, NULL, NULL);
o1 = init_node(always_send_one, 0, NODE_TYPE, NULL, hebbian_o);
o2 = init_node(always_send_one, 0, NODE_TYPE, NULL, hebbian_o);
add_extra_data(k1, copy(deca5));
add_extra_data(k2, copy(deca1));
add_extra_data(k3, copy(deca2));

i->error_correction = ACTIVE;
k1->error_correction = ACTIVE;
k2->error_correction = ACTIVE;
CHAPTER 5. IMPLEMENTATION AND TESTING

k3->error_correction = ACTIVE;
o1->hebbian = ACTIVE;
o2->hebbian = ACTIVE;

form_link(i, k1, copy(decq));
form_link(i, k2, copy(decq));
form_link(i, k3, copy(decq));
form_link(k1, h1, create_decQuad_int(0));
form_link(k2, h2, create_decQuad_int(0));
form_link(k3, h3, create_decQuad_int(0));
form_link(h1, o1, copy(deca3));
form_link(h1, o2, copy(deca4));
form_link(o1, k1, create_decQuad_int(0));
form_link(o1, k2, create_decQuad_int(0));
form_link(o1, k3, create_decQuad_int(0));
form_link(o2, k1, create_decQuad_int(0));
form_link(o2, k2, create_decQuad_int(0));
form_link(o2, k3, create_decQuad_int(0));
insert_node(i->id, "S", pro1);
insert_node(k1->id, "K", pro1);
insert_node(k2->id, "K", pro1);
insert_node(k3->id, "K", pro1);
insert_node(h1->id, "H", pro1);
insert_node(h2->id, "H", pro1);
insert_node(h3->id, "H", pro1);
insert_node(o1->id, "O", pro1);
insert_node(o2->id, "O", pro1);

while(1)
{
    perform_network(pro1);
    i->work_state = WAITING_WSTATE;
    k1->work_state = WAITING_WSTATE;
    k2->work_state = WAITING_WSTATE;
    k3->work_state = WAITING_WSTATE;
    h1->work_state = WAITING_WSTATE;
    h2->work_state = WAITING_WSTATE;
    h3->work_state = WAITING_WSTATE;
    o1->work_state = WAITING_WSTATE;
    o2->work_state = WAITING_WSTATE;
    pro1->process_count = 0;

    printf("result |h1: %s | h2: %s | h3: %s \n",
            decQuad_to_string(h1->out),
            decQuad_to_string(h2->out),
            decQuad_to_string(h3->out));
As we have discussed how to construct normal neural network in the previous example, however SLD neural network is not different with normal neural network, therefore, we are not going to discuss the detail of this example. With this code, we successfully construct the previous SLD neural network. However it is not necessary to discuss the process of network, because it is complete follows the procedure based on the (Komendantskaya, 2007).

The following is the result output from the example

Listing 5.9: Result of the SLD neural network
```
result h1: 800104900101 h2: 800101900201 h3: 0
```

Now we are ready to discuss the SLD neural network for auto construction.

### 5.2.5 Auto construction of SLD neural network

This auto-constructor is intends to allow user to construct SLD neural network in a simple way, the following code demonstrate this example

Listing 5.10: Auto construction of SLD neural network
```
init_table();
init_global_set();
init_stack();

int father = auto_create_term(PREDICATE,ENTYPE);
int grandfather = auto_create_term(PREDICATE,ENTYPE);
int tom = auto_create_term(CONSTANT,ENTYPE);
int dave = auto_create_term(CONSTANT,ENTYPE);
int peter = auto_create_term(CONSTANT,ENTYPE);
int X = auto_create_term(VARIABLE,ENTYPE);
int Y = auto_create_term(VARIABLE,ENTYPE);
int Z = auto_create_term(VARIABLE,ENTYPE);

int a1 = auto_create_atom(father, 2, tom, dave);
int a2 = auto_create_atom(father, 2, dave, peter);
int a3 = auto_create_atom(father, 2, X, Y);
int a4 = auto_create_atom(father, 2, Y, Z);
int a5 = auto_create_atom(grandfather, 2, X, Y);

n_network *n1 = auto_create_goal(a1, 0);
n_network *n2 = auto_create_goal(a2, 0);
n_network *n3 = auto_create_goal(a5, 2, a3, a4);

n_network *pro1 = auto_create_sld(3, n1, n2, n3);
perform_sld(pro1);
```
This example is very easy to be understood, in addition it is very easy to be constructed. It produce the exactly the same result as the previous example, which means there is no difference between this SLD neural network and the previous one.
Chapter 6

Conclusions

6.1 Critical analysis of the project

In this section, we will discuss the critical analysis of this project, however we sucessfully implement the API for creating both general and SLD neural networks, there still exist some problems that need to be aware. There are two most important problem, the first problem is, it is impossible to determine the numbers of iteration process of the SLD neural network, that is before we get the final answer, we can not even determine if there exit an answer. The second problem is, every SLD neural can only proform once, that the SLD neural network can not maintain its structure after it proceed once.

For the first problem, consider a large logic program with a unsolvable question, if we can not determine the numbers of iteration of the process, therefore we may wait forever without know whether an answer is exist or not. As the result, we may assume the SLD neural networks is unvaluable, however this is not true, from the experience of testing, we usually get the result less than 6 iterations, therefore we assume a small logic program should takes around or less than 6 iterations, and a bigger programe should take about 6 to 10 iterations. Therefore it is still possible to determine whether there is an answer.

For the second problem, it has been described in (Onyeyiri, 2008), every time when we proceed the network, the weights between connections are altered, moreover, data such as input value of each neuron and desire value of some neurons has been changed, therefore, the networks can not be reuse. The solution has also been claimed in (Onyeyiri, 2008), which by using matrix, however this approach will not be discussed in this project, whereas it is not directly relate to this project.
6.2 Project evaluation

Despite the aim of the project has been met, but most of the testing base on black-box testing, due to the length and the complexity of the project. The testing of the API may not be sufficient, moreover runtime errors usually cannot be detected with only black-box testing. Other problems also encountered during implementation, despite we implement all training models are required for constructing SLD neural networks, but they are not exactly as the same as specified on (Komendantskaya, 2007), it is because our implementation is much simpler, for this reason, we can not be sure every SLD neural networks will be run correctly, but this can be corrected by reconstruct these models. Finally due to the duration of the project and which is completed without much testing, therefore a further development and fixes of the API should be made in the future, at this stage the project has been complete with adequate features, however it also means this dissertation may not have enough time to cover all the detail of the API.
Bibliography


Komendantskaya, E. (2007), Learning and deduction in neural networks and logic, PhD thesis, Department of Mathematics College of Science, Engineering and Food Science National University of Ireland.

Onyeyiri, P. (2008), Maintaining Logic Program Structure in Neural Networks, PhD thesis, Department of Computer Science, University of Bath.

Richardson, D. (1998), Formal system, logic and semantics, Department of Computer Science, University of Bath.

Appendix A

Code
A.1 File: encoding.h

/* File: encoding.h */
/* Author: galaxy8691 */
/* Created on 13 February 2009, 16:30 */

#ifndef _ENCODING_H
#include "decNum/decQuad.h"
#include "decNum/decContext.h"
define _ENCODING_H
#define EN_TYPE 10
#define INT_TYPE 11
#define VARIABLE_TYPE 1
#define CONSTANT_TYPE 2
#define FUNCTION_TYPE 3
#define PREDICATE_TYPE 4
#define OBRA_TYPE 5
#define CBRA_TYPE 6
#define COMMA_TYPE 7
#define CONCAT_TYPE 8
#define DISSAGRN_TYPE 9
#define UNKNOWN_TYPE -1

struct m_godel {
    int type;
    int en_type; // -1 if integer;
    int index;
    struct m_godel *next;
};
typedef struct m_godel m_godel;

m_godel *create_m_godel(int type, int index);
m_godel *concatenate(const m_godel *first, const m_godel *next);
m_godel *convert_single_m_godel(int num);
m_godel *convert_whole_m_godel(const decQuad *num);
m_godel *m_godel_copy(const m_godel *src);
void m_godel_free(m_godel *tar);
void m_godel_all_free(m_godel *tar);
m_godel *disagreement(const m_godel *first, const m_godel *second);
m_godel *f_p_block(const m_godel *f_p); // function or predicate block
m_godel *disagreement_theta_fp(const m_godel *var, const m_godel *fp); // theta for function or predicate
m_godel *disagreement_theta_cv(const m_godel *var, const m_godel *cv); // constant or variable
m_godel *create_atom(const m_godel *fp, int n arity, ...)
m_godel *substitute(const m_godel *target, const m_godel *theta);
m_godel *substitute_cv(m_godel *target, const m_godel *cv);
m_godel *substitute_fp(m_godel *target, const m_godel *fp);
m_godel *create_atom_ary(const m_godel *fp, int n arity, m_godel *[]);
#endif /*_ENCODING_H*/

A.2 File: encoding.c

/* gvgg */
/* File: encoding.c */
/* Author: galaxy8691 */
/* Created on 13 February 2009, 17:00 */

#include "encoding.h"
#include <stdlib.h>
#include "decNum/decQuad.h"
#include "decNum/decContext.h"
#include "stack.h"
#include <string.h>
#include "util.h"
#include "stdio.h"
#include <stdarg.h>

decQuad *get_value(const m_godel *v) {
    decQuad *num = create_decQuad_int(0);
    decQuad tempnum1, tempnum2;
    const m_godel *temp = v;
    decQuad inc100, inc1000;
    decQuadFromInt32(&inc100, 100);
    decQuadFromInt32(&inc1000, 1000);
    while (temp != NULL) {
        decQuadFromInt32(&tempnum1, temp->type);
        decQuadMultiply(&tempnum1, &tempnum1, &inc100, &global_set);
        decQuadFromInt32(&tempnum2, temp->index);
        decQuadAdd(&tempnum1, &tempnum1, &tempnum2, &global_set);
        decQuadAdd(num, num, &inc1000, &global_set);
        decQuadAdd(num, num, &tempnum1, &global_set);
        temp = temp->next;
    }
    return num;
}

m_godel *create_m_godel(int en_type, int index) {
    m_godel *temp = (m_godel *) malloc(sizeof(m_godel));
    temp->en_type = en_type;
    temp->index = index;
    temp->type = EN_TYPE;
    temp->next = NULL;
    return temp;
}

m_godel *concatenate(const m_godel *first, const m_godel *next) {
    if (next == NULL)
    }

m_godel *convert_single_m_godel(int num) {
    m_godel *temp = create_m_godel(UNKNOWN_ENTYPE, 0);
    int v = num % 100;
    int t = num / 100;
    temp->en_type = t;
    temp->index = v;
    temp->type = EN_TYPE;
    temp->next = NULL;
    return temp;
}

m_godel *convert_whole_m_godel(const decQuad *num) {
    decQuad *temp_num = (decQuad *) malloc(sizeof(decQuad));
    decQuadCopy(temp_num, num);
    m_godel *first, *last;
    char s[500];
    decQuadToString(temp_num, s);
    if (strcmp(s, "0") == 0) {
        first = convert_single_m_godel(0);
        free(temp_num);
        return first;
    }
    printf("%s\n", s);
    while (strcmp(s, "0") != 0) {
    }
}
decQuad n1000;
decQuad *temp = (decQuad *)
    malloc(sizeof(decQuad)));
decQuadFromInt32(&n1000, 1000);
decQuadRemainder(temp, temp_num, &n1000, &global_set);
//
decQuadToString(temp, s);
decQuadToString(temp_num, s);
decQuadSubtract(temp_num, temp_num, temp_num, &global_set);
decQuadToString(temp_num, s);
//
first = convert_single_m_godel(pop());
last = first;
while(!is_empty())
{
    m_godel *temp = convert_single_m_godel(pop());
last->next = temp;
last = last->next;
}
last->next = NULL;
free(temp_num);
return first;

m_godel *m_godel_copy(const m_godel *src)
{
    if(src == NULL)
    {
        return NULL;
    }
    m_godel *temp = create_m_godel(UNKNOWNETYPE, 0);
temp->next = m_godel_copy(src->next);
    return temp;
}

void m_godel_all_free(m_godel *tar)
{
    if(tar == NULL)
    {
        return;
    }
    if(tar->next != NULL)
    {
        free(tar->next);
tar->next = NULL;
    }
    else
    {
        free(tar);
    }
    return;
}

m_godel *disagreement(const m_godel *first, const m_godel *second)
{
    m_godel *temp = create_m_godel(UNKNOWNETYPE, 0);
const m_godel *temp_first = first;
const m_godel *temp_second = second;
int done = 0;
while(!done)
{
    if(temp_first->en_type == OBRAETYPE ||
        temp_first->en_type == CBRAETYPE ||
        temp_first->en_type == COMMAETYPE)
    {
        if(temp_first->next == NULL ||
temp_second->next == NULL)
        {
            m_godel_all_free(temp);
            return NULL;
        }
    }
    done = 1;
}
else
{
    temp_first =
    temp_first->next;
    temp_second =
    temp_second->next;
}

else if((temp_first->en_type ==
    temp_second->en_type) && (temp_first->index ==
    temp_second->index))
{
    if(temp_first->next == NULL ||
        temp_second->next == NULL)
    {
        m_godel_all_free(temp);
        return NULL;
    }
    else
    {
        temp_first =
        temp_first->next;
        temp_second =
        temp_second->next;
    }
}

else if((temp_first->en_type !="|
    temp_second->en_type !="|
    temp_first->index !=
    temp_second->index) && (temp_first->en_type ==
    VARIABLE_ENTYPE
    || temp_second->en_type ==
    VARIABLE_ENTYPE))
{
    if(temp_first->en_type ==
        PREDICATE_ENTYPE ||
        temp_second->en_type ==
        PREDICATE_ENTYPE)
    {
        temp_first->en_type
        = FUNCTION_ENTYPE
        ||
        temp_second->en_type
        = PREDICATE_ENTYPE
        ||
        temp_second->en_type
        = FUNCTION_ENTYPE)
    {
        if(temp_first->en_type ==
        VARIABLE_ENTYPE)
        {
            m_godel *fp_block
            = f_p_block(temp_second);
            temp =
            disagreement_theta_fp(temp_first, fp_block);
            m_godel_all_free(fp_block);
            return temp;
        }
        else
        {
            m_godel *fp_block
            = f_p_block(temp_first);
            temp =
            disagreement_theta_fp(temp_second, fp_block);
            m_godel_all_free(fp_block);
            return temp;
        }
    }
    else
    {
        if(temp_first->en_type ==
        VARIABLE_ENTYPE)
        {
            temp =
            disagreement_theta_cv(temp_first, temp_second);
        }
    }
}
m_godel *disagreement_theta_fp(const m_godel *var, const m_godel *fp) {
    temp = disagreement_theta_cv(temp_second, temp_first);
    m_godel *temp = create_m_godel(UNKNOWN_ENTYPE, 0);
    m_godel *p_temp = temp;
    const m_godel *p_fp = fp;
    temp->en_type = var->en_type;
    temp->index = var->index;
    p_temp->next = create_m_godel(DISSAGRN_ENTYPE, 0);
    p_temp = p_temp->next;
    while (p_fp != NULL)
    {
        p_temp->next = create_m_godel(p_fp->en_type, p_fp->index);
        p_temp = p_temp->next;
        p_fp = p_fp->next;
    }
    return temp;
}

m_godel *disagreement_theta_cv(const m_godel *var, const m_godel *cv) {
    m_godel *temp = create_m_godel(UNKNOWN_ENTYPE, 0);
    temp->en_type = var->en_type;
    temp->index = var->index;
    temp->next = create_m_godel(DISSAGRN_ENTYPE, 0);
    cv->index = temp;
    return temp;
}

m_godel *create_atom(const m_godel *fp, int n_arity, ...) {
    va_list list;
    va_start(list, n_arity);
    int i;
    m_godel *temp = m_godel_copy(fp);
m_godel *p_temp = temp;
p_temp->next = create_m_godel(OBRAETYPE, 0);
p_temp = p_temp->next;
for (i = 0; i < n arity; i++)
{
p_temp->next = m_godel_copy(va_arg(list, m_godel *));
if (i + 1 < n arity)
{
p_temp->next->next =
create_m_godel(COMMAETYPE, 0);
p_temp = p_temp->next->next;
}
else
{
p_temp = p_temp->next;
}
}
p_temp->next = create_m_godel(CBRAETYPE, 0);
va_end(list);
return temp;

m_godel *substitute(const m_godel *target, const m_godel *theta)
{
  m_godel *temp = m_godel_copy(target);
m_godel *p_temp = temp;
const m_godel *p_theta = theta;
while (1)
{
  if (p_theta == NULL)
  {
    return NULL;
  }
  else if (p_theta->en_type ==
  CONCATIENTYPE)
  {
p_theta = p_theta->next;
  }
  else if (p_theta->en_type ==
  VARIABLEIENTYPE &&
  p_theta->next->en_type ==
  DISSAGRNETYPE)
  {
    while (p_temp != NULL)
    {
      if (p_temp->en_type ==
        p_theta->en_type &&
        p_temp->index
        ==
        p_theta->index)
      {
        *p_p_theta=
        p_theta->next->next;
        if (p_p_theta->en_type
        ==
        FUNCTIONETYPE
        ||
        p_p_theta->en_type
        ==
PREDICATEENTYPE)
        {
          substitute_fp(p_temp,
            p_p_theta);
        }
      }
      else
      {
        substitute_cv(p_temp,
            p_p_theta);
      }
    }
    p_p_theta;
    int done = 0;
    while (!done)
    {
      done = 1;
      p_temp = p_temp->next;
    }
    p_temp = temp;
  }
}
APPENDIX A. CODE

```c
if(p_theta == NULL)
{
    return temp;
}
else if(p_theta->en_type != CONCAT_ENTYPE)
{
    p_theta = p_theta->next;
}
else
{
    done = 1;
}
}
}
else
{
    int done = 0;
    while(!done)
    {
        if(p_theta == NULL)
        {
            return temp;
        }
        else if(p_theta->en_type != CONCAT_ENTYPE)
        {
            p_theta = p_theta->next;
        }
        else
        {
            done = 1;
        }
    }
    return temp;
}
}
```

```c
m_godel *substitute_fp(m_godel *target, const m_godel *fp)
{
    m_godel *head = target;
    m_godel *next = target->next;
    m_godel *fp_block = f_block(fp);
    m_godel *fp->next = fp_block;
    while(p_fp_block->next != NULL)
    {
        p_fp_block = p_fp_block->next;
    }
    if(p_fp_block!= NULL)
    {
        free(fp_block);
    }
    return target;
}
```

```c
m_godel *create_array(const m_godel *fp, int n arity, m_godel *g[])
{
    int i;
    m_godel *temp = m_godel_copy(fp);
    m_godel *p_temp = temp;
    p_temp->next = create_m_godel(OBRA_ENTYPE, 0);
    p_temp = p_temp->next;
    for(i = 0; i < n arity; i++)
    {
        p_temp->next = m_godel_copy(g[i]);
        if(i + 1 < n arity)
        {
            p_temp->next->next = create_m_godel(COMMA_ENTYPE, 0);
            p_temp = p_temp->next->next;
        }
    }
    else
    {
        p_temp = p_temp->next;
    }
    return temp;
}
```
Appendix A. Code

A.3 File: n_network.h

/* File: n_network.h */
/* Author: galaxy8691 */
/* Created on 02 February 2009, 16:16 */

#ifndef _N_NETWORK_H
#define _N_NETWORK_H
#include "global.h"
#endif /* _N_NETWORK_H */

#define TRUE 1
#define FALSE 0

struct layer
{
    char *name;
    int *n_id;
    int n_nodes;

    int gross_layer;
    void (*gross_layer_fn)(struct layer *, current_layer);

    int kohonen_layer;
    void (*kohonen_layer_fn)(struct layer *, current_layer);
};
typedef struct layer layer;

struct n_network
{
    layer **layers;
    int n_layers;
    int n_hold;
    int process_count;
};
typedef struct n_network n_network;

void init_network(n_network *network);
void insert_node(int id, char *layer_name, n_network *network);
void perform_node(int id, n_network *network);
int has_layer(char *name, n_network *network);
void create_layer(char *name, n_network *network, void (*kl_fn)(layer *), void (*gl_fn)(layer *));
void perform_layer(layer *l, n_network *network);
layer *get_layer(n_network *network, char *name);
#endif /* _N_NETWORK_H */

A.4 File: n_network.c

#include "n_network.h"
#include <stdlib.h>
#include "node.h"
#include "string.h"
#include "util.h"

void init_network(n_network *network)
{
    network->n_hold = 0;
    network->process_count = 0;
    network->n_layers = 0;
}

void insert_node(int id, char *layer_name, n_network *network)
{
    int j;
    if(!has_layer(layer_name, network))
    {
        create_layer(layer_name, network, NULL, NULL);
    }
}

p_temp->next = create_m_godel(CBRAJETYPE, 0);
return temp;
APPENDIX A. CODE

```c
for (j = 0; j < network->n_layers; j++)
    {
        if (strcmp(layer_name, network->layers[j]->name) == 0)
            {
                if (network->layers[j]->n_nodes == 0)
                    {
                        network->layers[j]->n_id = (int *)malloc(sizeof(int));
                        network->layers[j]->n_id[0] = id;
                        network->layers[j]->n_nodes = 1;
                        network->n_hold = 1;
                        look_for_node(id)->layer = network->layers[j];
                        return;
                    }
                else
                    {
                        int i = 0;
                        void perform_node(int id, n_network *network)
                        {
                            int *temp_holder = (int *)malloc(sizeof(int) * network->layers[j]->n_nodes);
                            look_for_node(id)->work_state = INIT_STATE;
                            for (i = 0; i < network->layers[j]->n_nodes; i++)
                                temp_holder[i] = network->layers[j]->n_id[i];
                            realloc(network->layers[j]->n_id, sizeof(int) * (network->layers[j]->n_nodes + 1));
                            if (can_data_collect(look_for_node(id)) == INIT_STATE)
                                {
                                    cal_out_data(look_for_node(id));
                                }
                            else if (look_for_node(id)->work_state == READY_SEND_WSTATE)
                                {
                                    send_data_all(look_for_node(id));
                                    network->process_count++;
                                }
                            else if (look_for_node(id)->work_state == WORKDONE_WSTATE)
                                {
                                    network->process_count++;
                                }
                        }
                        return;
                    }
            }
```
```c
for (i = 0; i < network->layers[j]->n_nodes; i++)
    {
        network->layers[j]->n_id[i] = temp_holder[i];
        network->layers[j]->n_id[i] = id;
        network->layers[j]->n_nodes++;
        network->n_hold++;
        look_for_node(id)->layer = network->layers[j];
        free(temp_holder);
        return;
    }
```
```c
}```
```c
send_data_all(look_for_node(id));
network->process_count++;
}
// printf("%i: %s\n", id, decQuad_to_string(look_for_node(id)->out));

void perform_network(n_network *network)
{
    int i;
    while(network->process_count < network->n_hold)
    {
        for(i = 0; i < network->n_layers; i++)
        {
            perform_layer(network->layers[i], network);
        }
    }
}

void perform_layer(layer *l, n_network *network)
{
    if(l->kohonen_layer == ACTIVE)
    {
        l->kohonen_layer_fn(l);
    }
    if(l->gross_law == ACTIVE)
    {
        l->gross_law_fn(l);
    }
    int i;
    for(i = 0; i < l->n_nodes; i++)
    {
        perform_node(l->n_id[i], network);
    }
}

int has_layer(char *name, n_network *network)
{
    if(network->n_layers == 0)
    {
        return FALSE;
    }
    else
    {
        int i;
        for(i = 0; i < network->n_layers; i++)
        {
            if(strcmp(network->layers[i]->name, name) == 0)
            {
                return TRUE;
            }
        }
        return FALSE;
    }
}

void create_layer(char *name, n_network *network, void (*kl_fn)(layer*), void (*gl_fn)(layer*))
{
    if(network->n_layers == 0)
    {
        network->layers = (struct layer **)malloc(sizeof(struct layer *) * ++network->n_layers);
        struct layer *l = (struct layer *)malloc(sizeof(struct layer));
        l->n_nodes = 0;
        l->name = name;
        l->gross_law = INACTIVE;
        l->kohonen_layer = INACTIVE;
        l->kohonen_layer_fn = gl_fn;
        l->kohonen_layer_fn = kl_fn;
        network->layers[network->n_layers - 1] = l;
    }
    else
    {
        int i;
        struct layer **temp_layer = (struct layer **)malloc(sizeof(struct layer *) * network->n_layers);
        for(i = 0; i < network->n_layers; i++)
        {
            temp_layer[i] = network->layers[i];
        }
    }
```
APPENDIX A. CODE

```c
network->layers = (struct layer **)&realloc(network->layers,
sizeof(struct layer *) *
++network->n_layers);
for(i = 0; i < network->n_layers - 1;
i++)
{
    network->layers[i] =
    temp_layer[i];
}
struct layer *l = (struct layer *)malloc(
    sizeof(struct layer));
l->n_nodes = 0;
l->name = name;
l->gross_law = INACTIVE;
l->kohonen_layer = INACTIVE;
l->gross_law_fn = gl_fn;
l->kohonen_layer_fn = kl_fn;
network->layers[network->n_layers - 1] =
l;
free(temp_layer);
}

layer *get_layer(n_network *network, char *name)
{
    int i;
    for(i = 0; i < network->n_layers; i++)
    {
        if(strcmp(network->layers[i]->name, name) == 0)
        {
            return network->layers[i];
        }
    }
    return NULL;
}

A.5 File: node.h

#ifndef NODE_H
#define NODE_H
#include "decNum/decQuad.h"
#include "n_network.h"
#include "global.h"
#define INIT_ISTATE 1
#define DATA_RECEIVED_ISTATE 2
#define NODE_TYPE 3
#define SLD_NODE_TYPE 4
#define WAITING_WSTATE 5
#define WORKDONE_WSTATE 6
#define READY_SEND_WSTATE 7
#define SEND_AT_START_WSTATE 8
#define TABLE_SIZE 50000

struct node
{
    int id;
    int type;
    layer *layer;

    struct in_table
    {
        struct node **in_link;
        data *weight; // have to point to the same
                       // address as in_link node's weight
        int no_in;
        int *state;
        data *in_pool;
    } in_t;
    data in;

    struct out_table
    {
        struct node **out_link;
        int no_out;
        data *weight;
    } out_t;
};
```

A.5 File: node.h
typedef struct node node;

data *n_table [TABLE_SIZE];

void cal_out_data (node *);
void send_data (node *src, node *des);
node *init_node (data (*fn) (node *), int *error_correction_data);
int error_correction;
void form_in_link (node *src, node *des, data *fn);
void form_out_link (node *src, node *des, data *fn);
void collect_inpool (node *n);
void send_data_all (node *from);
int can_data_collect (node *);
void clear_in_state (node *);
void replace_e_data (node *n, data *d);
void replace_h_data (node *n, data *d);
void replace_extra_data (node *n, data *d, int i);

void init_table (void);
void insert_to_table (node *n);
node *look_for_node (int id);

A.6 File: sld_nn.h

#ifndef SLD_NN_H
#define SLD_NN_H
#define MAX_HOLD 50000

#include "n_network.h"
#include "encoding.h"

typedef struct node node;
node *n_table [TABLE_SIZE];

void cal_out_data (node *);
void send_data (node *src, node *des);
node *init_node (data (*fn) (node *), int *error_correction_data);
int error_correction;
void form_in_link (node *src, node *des, data *fn);
void form_out_link (node *src, node *des, data *fn);
void collect_inpool (node *n);
void send_data_all (node *from);
int can_data_collect (node *);
void clear_in_state (node *);
void replace_e_data (node *n, data *d);
void replace_h_data (node *n, data *d);
void replace_extra_data (node *n, data *d, int i);

void init_table (void);
void insert_to_table (node *n);
node *look_for_node (int id);

#endif
A.7 File: sld_nn.c

```c
/*
 * sld_nn.c
 * Created on: 27-Apr-2009
 * Author: wik20
 */

#include <stdio.h>
#include <stdlib.h>
#include "node.h"
#include "ifnset.h"
#include "network.h"
#include "encoding.h"
#include "decNum/decQuad.h"
#include "util.h"
#include "train_fnset.h"
#include "stack.h"
#include "sld_nn.h"
#include "global.h"

void init_sld_nn(void)
{
    int i = 0;
    for(i = 0; i < MAXHOLD; i++)
    {
        m_godel_holder[i] = NULL;
    }
    m_godel_head = 0;
    pre_count = 1;
    fun_count = 1;
    con_count = 1;
    var_count = 1;
}

int auto_create_term(int en_type)
{
    if(en_type == PREDICATE_ENTYPE)
    {
        m_godel_holder[m_godel_head] =
        create_m_godel(en_type, pre_count);
        pre_count++;
    } else if(en_type == FUNCTION_ENTYPE)
    {
        m_godel_holder[m_godel_head] =
        create_m_godel(en_type, fun_count);
        fun_count++;
    } else if(en_type == CONSTANT_ENTYPE)
    {
        m_godel_holder[m_godel_head] =
        create_m_godel(en_type, con_count);
        con_count++;
    } else if(en_type == VARIABLE_ENTYPE)
    {
        m_godel_holder[m_godel_head] =
        create_m_godel(en_type, var_count);
        var_count++;
    }
    return m_godel_head++;
}

int auto_create_atom(int pf, int n_ary, ...)
{
    va_list list;
    va_start(list, n_ary);
    int i = 0;
    m_godel_holder[n_ary] =
    for(i = 0; i < n_ary; i++)
    {
        m_godel_holder[i] = m_godel_holder[va_arg(list, int)];
    }
    m_godel_holder[m_godel_head] =
    create_atom_ary(m_godel_holder[pf], n_ary, mh);
    va_end(list);
    return m_godel_head++;
}

n_network *auto_create_goal(int head, int n_dg, ...)
{
    int index;
    va_list list;
```
va_start(list, n_dg);

n_network *pro1 = (n_network *) malloc(sizeof(n_network));
init_network(pro1);

create_layer("K", pro1, kohonen_layer_fn_k, gross_layer_fn_k);
create_layer("H", pro1, NULL, NULL);
create_layer("O", pro1, NULL, NULL);
get_layer(pro1, "K")->kohonen_layer = ACTIVE;
get_layer(pro1, "K")->gross_layer = ACTIVE;

node *k1 = init_node(always_send_one, 0, NODE_TYPE, error_correction);
node *h1 = init_node(nchange_pass, 0, SLD_NODE_TYPE, NULL, NULL);
form_link(k1, h1, create_decQuad_int(0));
node *o[n_dg];
decQuad *w[n_dg];

for(index = 0; index < n_dg; index++)
{
    w[index] = get_value(m_godel_holder[va_arg(list, int)]);
    o[index] = init_node(always_send_one, 0, NODE_TYPE, NULL, hebbian_o);
    o[index]->hebbian = ACTIVE;
    form_link(h1, o[index], w[index]);
}
insert_node(k1->id, "K", pro1);
insert_node(h1->id, "H", pro1);

for(index = 0; index < n_dg; index++)
{
    insert_node(o[index]->id, "O", pro1);
}
va_end(list);
return pro1;

n_network *auto_create_sld(int n_nn, ...)
{
    int i;

    va_list list;
    va_start(list, n_nn);
    int n_node_o = 0;
    int l_k = 0, l_h = 0, l_o = 0;

    n_network *pro1 = (n_network *) malloc(sizeof(n_network));
    init_network(pro1);

create_layer("S", pro1, NULL, NULL);
create_layer("K", pro1, kohonen_layer_fn_k, gross_layer_fn_k);
create_layer("H", pro1, NULL, NULL);
create_layer("O", pro1, NULL, NULL);
get_layer(pro1, "K")->kohonen_layer = ACTIVE;
get_layer(pro1, "K")->gross_layer = ACTIVE;

node *k[n_nn];
node *h[n_nn];

for(i = 0; i < n_nn; i++)
{
    temp[i] = va_arg(list, n_network*);
    n_node_o += get_layer(temp[i], "O")->n_nodes;
}
node *o[n_node_o];

for(i = 0; i < n_nn; i++)
{
    int j;
    for(j = 0; j < get_layer(temp[i], "K")->n_nodes; j++)
    {
        k[l_k] = look_for_node(get_layer(temp[i], "K")->n_id[j]);
        l_k++;
    }
    for(j = 0; j < get_layer(temp[i], "H")->n_nodes; j++)
    {
        h[l_h] = look_for_node(get_layer(temp[i], "H")->n_id[j]);
        l_h++;
    }
}
APPENDIX A. CODE

"H"->n_id[j];
1_h++;
}
for(j = 0; j < get_layer(temp[i],
"O")->n_nodes; j++)
{
o[l_o] =
look_for_node(get_layer(temp[i],
"O")->n_id[j]);
1_o++;
}
}
for(i = 0; i < l_k; i++)
{
insert_node(k[i]->id, "K", pro1);
}
for(i = 0; i < l_h; i++)
{
insert_node(h[i]->id, "H", pro1);
}
for(i = 0; i < l_o; i++)
{
int j;
for(j = 0; j < n_nn; j++)
{
form_link(o[i], k[j],
create_decQuad_int(0));
}
insert_node(o[i]->id, "O", pro1);
}
va_end(list);
return pro1;

void perform_sld(n_network *sld)
{
while(1)
{
  int i, j;
  perform_network(sld);
  for(i = 0; i < sld->n_layers; i++)
  {
    for(j = 0; j < sld->layers[i]->n_nodes; j++)
    {
      look_for_node(sld->layers[i]->n_id[j])->work_state = WAITING_WSTATE;
    }
  }
  sld->process_count = 0;
}
}

void insert_question(n_network *sld, int q)
{
  node *i = init_node(always_send_one, 0, NODE_TYPE
, error_correction, 0, error_correction, NULL);
i->error_correction = ACTIVE;
int index = 0;
decQuad *w = get_value(m_godel_holder[q]);
for(index = 0; index < get_layer(sld,
"K")->n_nodes; index++)
{
  form_link(i, look_for_node(get_layer(sld,
"K")->n_id[index]), copy(w));
}
free(w);
}