Computer Science Journal of Moldova, vol.**12**, no.1(34), 2004

Symbolic Deductive Reasoning using Connectionist Models

Mihai Horia Zaharia Florin Leon Dan Gâlea

Abstract

In this paper, we try to combine the possibility of symbolic deductive reasoning with the learning capability of the connectionist models. We introduce several algorithms for learning relations between concepts and finding paths in a transitive manner between learned concepts. An application that implements the proposed model is described and a number of case studies are presented.

Keywords: artificial intelligence, cognitive agents, schemata, knowledge representation, deductive reasoning, neural networks

1 Introduction

Until recently, Artificial Intelligence methods were based on two main paradigms: symbolic and connectionist. Symbolic methods extended classic logic and were mainly used for problem solving and reasoning, whereas connectionist, sub-symbolic methods were mainly used for pattern recognition and functional approximation. The former had the advantage of explicit knowledge representation but learning was difficult, and the latter had the advantage of learning, but were called "black-box" models because the representation of knowledge was not explicit at any time.

Cognitive psychology defines a cognitive system as a physical system with two characteristics: representation and processing. For our model, these two components will be presented in the subsequent paragraphs. Our model can represent a base for developing the cognitive

system of a cognitive agent, i.e. an intelligent agent capable of learning, planning and problem solving.

2 The Model of a Learning Context

Information is not received and encoded in isolation. Every piece of information is related to other information, close to it spatially, temporally or even semantically. This is the reason why small fragments of the learning circumstances can trigger the retrieval of the whole situation [3].

Schemata are hypothetical mental structures for storing generic concepts, pieces of appropriately categorized information, into memory. They can be regarded as a sort of framework structure for representing knowledge. Schemata are similar in structure to type definition. A concept type may have at most one definition, but arbitrarily many schemata. Schemata are thought to store generic, abstract, or prototypical knowledge, i.e. they encapsulate perceived regularities in experiences of which there have been many particular instances. Once formed, they are used to guide further encoding, organization and retrieval of information. [2]

We define a learning context as a set of semantically related schemata that are directly or indirectly linked to each other in the representation:

$$LC = \{s_1 \in S \mid \forall s_j \in S, i \neq j, link(s_i, s_j) = 1\} \tag{1}$$

where $link : s^2 \to \{0, 1\}$ is defined as follows:

$$link(s_i, s_j) = \begin{cases} 1 & \text{if } \exists\ directlink(s_i, s_j) \vee \\ & \exists\ s_k \in S, link(s_i, s_k) \cdot link(s_k, s_j) = 1); \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The *knowledge base* of the agent contains a set of learning contexts:

$$KB = \{LC_i \mid i = \overline{1, n}\} \tag{3}$$

where n is the number of learning contexts in the knowledge base.

We define a *concept* as any element of the natural language that can have a meaning by itself. From a functional perspective, a concept can represent an object, an action or an attribute.

The knowledge base is represented as a localized neural network, in which nodes correspond to concepts. The same concept can appear in different learning contexts that can give it different interpretations. Therefore, we differentiate between two types of nodes: simple processing nodes and dictionary nodes. A dictionary node is a node that uniquely corresponds to a natural language concept. A processing node can be linked both to one dictionary node and to one or more processing node. Processing nodes are used for actual information processing, while dictionary nodes take over the internal computation results and act as an interface for external communication.

This distinction is also based on the model of the biological brain, where special areas exist for language processing [4, 5]. Two very important areas are the Broca's area, a region on the inferior posterior side of the frontal lobe, and the Wernike's area, situated inside and around the superior posterior side of the temporal lobe. The Broca's area contributes to sentences formulation, and the Wernike's area is responsible for language understanding.

Just like in a classic neural network, the nodes are linked by connections with corresponding connection weights. Connections weights can be either positive (excitatory connections) or negative (inhibitory connections). The nodes have bipolar sigmoid (hyperbolic tangent) activation functions, with no bias:

$$f(s) = \frac{2}{1 + e^{-s}} - 1 = \frac{1 - e^{-s}}{1 + e^{-s}} \tag{4}$$

3 The Dynamic of Learning

As the agent receives new information, it builds his cognitive structure by adding or updating the connections between the nodes that correspond to the concepts it has to process. Learning can occur in four

situations:

- *Concept learning:* When the agent discovers a new concept, it creates a new dictionary node to hold the concept, and a new processing node in the current learning context, which is linked to the former;
- *Concept definition:* When the agent encounters a definition of a concept, i.e. the object-concept is described by a series of attribute-concepts, it finds the processing nodes corresponding to the concepts in the definition and then creates or updates the connections between the processing node of the object-concept and the processing nodes of the attribute-concepts;
- *Problem solving:* When the agent must solve a deductive problem, i.e. finding a path in its knowledge base structure that links an object-concept to an attribute-concept, it updates the connections weights on the path (if a path can be found), and creates a new direct connection between the processing node of the object-concept and the processing nodes of the attribute-concept. If a path is found, this new connection is given a positive weight. Otherwise, it is given a negative weight;
- *Forgetting:* Although learning and forgetting seem antonymic in common language, we define learning in a more general sense that refers to the changes in the weights of the memory connections. Thus, forgetting means decreasing the absolute values of the connection weights before a new context is learned.

3.1 Concept Learning

Suppose C is the new concept that has been encountered by the agent. In this case, two nodes are automatically created: a dictionary node $N_d(C)$) and a processing node N_p. They are then linked by two unidirectional connections, whose weights are initialized as follows:

$$weight(N_d(C), N_p) = weight(N_p, N_d(C)) = \eta_1, \tag{5}$$

where η_1 is a learning rate.

The two connections are needed because, similar to the case of a human mind, a mental image can be activated by understanding the corresponding word or, conversely, a word can be found to describes a mental image.

An acceptable value we have chosen for η_1 is 0.2 (see paragraph 3.5).

3.2 Concept Definition

In this case, the agent discovers a relation between an object-concept C_{obj} and a set of attribute-concepts $\{C^i_{attr}\}$. The relation can be positive or negative, e.g. "George *is* tall" or "The flower *is not* red".

The learning algorithm is:

```
find the dictionary node N_d(C_obj) associated with C_obj
find the processing node N_po linked to N_d(C_obj)
for each concept C_attr in {C^i_attr}
begin
   find the dictionary node N_d(C_attr) associated with C_attr
   find the processing node N_pa linked to N_d(C_attr)
   if the relation is excitatory then
      weight(N_po, N_pa) = weight(N_po, N_pa) + η2
   else // the relation is inhibitory
      weight(N_po, N_pa) = weight(N_po, N_pa) - η2
   end if
end for
```

It must be noted that unidirectional connections are made only form the object-concept to each attribute-concept. The reason for this is obvious: "A dog is an animal" is true, but "An animal is a dog" is false, because not every animal is a dog.

We use here another learning rate, η_2, which can be initialized with 0.1. If a connection doesn't exist, it is created and its weight is set to η_2 or $-\eta_2$, according to the polarity of the relation.

3.3 Problem Solving

In the above-presented framework, we define a deductive problem as the attempt to find a path in the concept space of the internal knowledge representation between an object-concept and an attribute-concept.

Our solution was intended to reflect an analogy with the human memory model. Traditionally, psychology distinguishes between two types of memory: long term and short term memory [3, 6]. The long-term memory (LTM) especially stores past information of great importance and usefulness for the life of the individual. The short-term memory (STM) allows the processing of recent, immediate information, which has only temporary significance. While short-term memory gives consistence and meaning to the moment, long-term memory ensures the continuity of the entire course of life.

Various experiments showed that those two types of memory differ not only by their content, but they have different characteristics:

- *Capacity:* STM has a capacity of 7 ± 2 chunks of information; LTM has an almost unlimited capacity;
- *Type of information encoding:* in STM there is mainly a verbal (phonologic) or imagistic encoding (it stores patterns of sounds or images); in LTM there is a semantic encoding;
- *Information maintenance:* in STM maintenance is achieved by continuous repetition; in LTM it is done by elaborative repetition that forms associations between the existing items;
- *Information retrieval:* in STM, the retrieval is supposed to be serial (e.g. remembering an item from a previously presented list is done by comparing the item to all the other items in the list); in LTM it is supposed to be parallel (e.g. the use of language).

Recent research in cognitive psychology proposes a unified model, where STM and LTM are not separate memory structures. The difference has a functional, not structural nature and mainly results from

the different levels of activation. STM is caused by the temporary activation of some parts of LTM. In our model, we also have a short-term (or working) memory:

$$STM = \{node_i \in LTM \mid activation(node_i) > 0\}. \tag{6}$$

When the agent must solve a problem, it successively activates different memory nodes, thus forming an activation path form the object-concept to the attribute-concept.

The path-finding and learning algorithm is:

```
find the dictionary node N_d(C_obj) associated with C_obj
find the dictionary node N_d(C_attr) associated with C_attr

// activate the dictionary node N_d(C_obj) associated with
C_obj activation(N_d(C_obj) = 1)
// activate the dictionary node N_d(C_attr) associated with
C_attr activation(N_d(C_attr) = 1
N = N_d(C_obj)
solution = not found

while solution is not found and
   the set of non-activated nodes is not void
begin
   if there exists a direct connection from N to N_d(C_attr)
   then solution = found
      break while
   end if

   // the absolute values of the weights are considered
   find the maximum weighted direct connection
      from N to a non-activated node N_next

   // activate the node N_next by applying the activation
function to the netinput
activation(N_next)=tanh(activation(N)*weight(N, N_next))
```

```
   // avoid inhibitory connections
   if activation(N_next) > 0 then
      N = N_next
   end if

   // an inhibitory connection suggest the possibility,
not the certainty, that a path won't be found
   if activation(N_next) < 0 and
      the dictionary node associated with N_next is N_d(C_attr)
   then
      display warning ('a path may not be found') or
      break while
   end if

end while

find the processing node N_po linked to N_d(C_obj)
find the processing node N_pa linked to N_d(C_attr)

if solution is found then
   for each connection CN on the path from N_d(C_obj)
      to N_d(C_attr)
      weight(CN) = weight(CN) + η3
   end for
   weight(N_po, N_pa) = weight(N_po, N_pa) + η2
else
   weight(N_po, N_pa) = weight(N_po, N_pa) - η2
end if
```

The algorithm shows that the connections on the path from the object-concept to the attribute-concept are increased every time the problem is solved. Therefore, the higher a connection weight, the higher the probability for that connection to be useful for solving the current problem, too. By analogy with the human mind, when people try to solve problems, they tend to first apply the methods they used more often and proved to be successful.

After a conclusion is drawn, a direct connection is created or updated from the object-concept to the attribute-concept. Once we solved a problem, we can directly apply the results for future tasks, without having to solve the same problem over and over again. However, this may not be easy at first. Until we get used to the result, it may be easier to use our previous methods, although they take a longer time to accomplish the task.

To simulate this behavior, we chose a third learning rate, $\eta_3 = 0.05$. This can be viewed as a generalization of the Hebbian learning rule [1] that stated that the change of a connection weight is proportional to the product of the input and the output of that connection. We simplified the idea to the extent that a connection weight is increased or decreased depending on the sign of the product between the activations of the input and output nodes. Therefore, if we find a path with positive activations that solves the problem, then all connections on that path will have their weights increased by the learning rate. Decreasing the other connections is not necessary, because they are not necessarily useless. If they didn't help finding a certain solution, it doesn't mean they will not help finding another in the future.

The use of activated nodes forbids infinite loops for circular definitions. If, in the reasoning process, the agent reaches a previously activated node (which is not the solution), it simply ignores it and goes to the next non-activated node.

3.4 Forgetting

After learning a context, or whenever needed, the absolute values of the connection weights are decreased by a proportional factor, $\eta_f < 1$:

```
for each connection CN
   weight(CN) = weight(CN) * η_f
end for
```

There are two reasons for this operation. The first is to avoid the saturation of the sigmoid function. If the values of the connection

weights grew unlimitedly, the activation function would produce results closer and closer to 1 or -1. By keeping the weight in an acceptable interval, the effects of learning are more precisely differentiated. The second reason is that by decreasing the old values, new experiences become more important than past ones. The forgetting factor we used was 0.8.

3.5 Choosing Learning Rates

Throughout the learning process of our cognitive model, three learning rates were used:

- η_1: the learning rate used to initialize the connection weights between the dictionary node of a newly encountered concept and its corresponding processing node;
- η_2: the learning rate used to initialize and update the connection weights between any two nodes in the memory in other situations than the previous one;
- η_3: the learning rate used to increase the connection weights on the path from an object-concept to an attribute-concept when a deductive problem is solved.

These learning rates must verify the following relation:

$$\eta_1 \geq \eta_2 > \eta_3. \tag{7}$$

η_1 dictates the strength by which the symbol (e.g. word) of a concept is linked to its internal representation. We assumed that this strength must be greater or at least equal to the strength by which two internal representations are linked (η_2), in order to stress the importance of the linguistic interface with the exterior, i.e. communication with the user or with other agents. Also, η_2 must be greater than η_3 to ensure that a new direct connection between the two terms of a deductive problem will eventually gain more strength than the path used to find the solution, so that after a few times the agent is presented the problem, it should choose the shorter path.

4 Deductive Reasoning Application

"Deductive Reasoning" application (figure 1) implements the cognitive model described above. In order to test different cases, we developed a script language that permits the input of new knowledge and the statement of deductive problems.

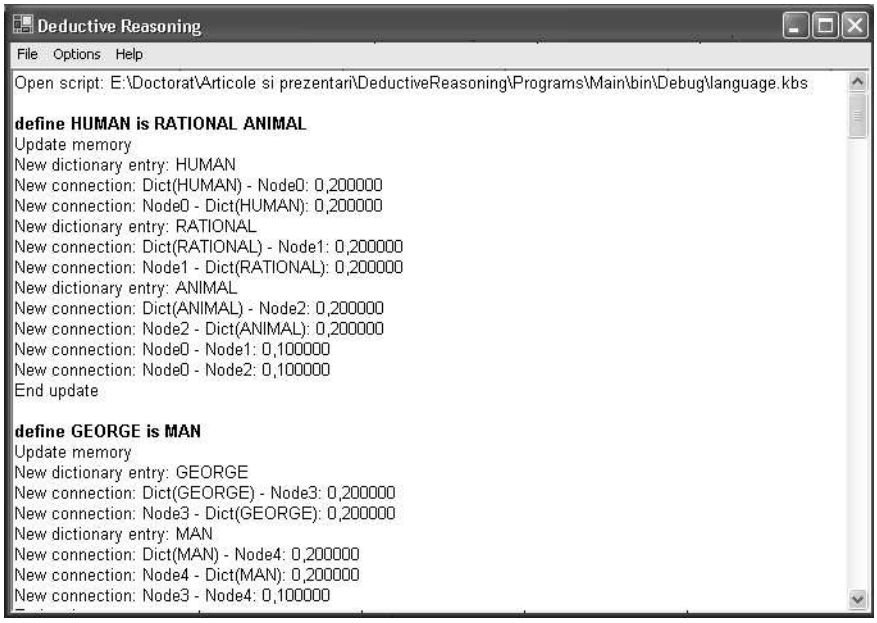

Figure 1. Deductive Reasoning application.

The allowed syntax of the language is presented below:

<knowledge>:= define <concept> is [not] {< *concept* >}
<problem>:= find <concept> <concept>
<forgetting>:= forget
<list_all>:= list
<list_processing>:= listproc

Examples of pieces of knowledge are:

```
define ANIMAL is BEING MOVE SENSE
define ANIMAL is not PLANT
```

Examples of deductive problems are:

```
find GEORGE SENSE
find ANIMAL PLANT
```

Let us consider the following script:

```
define A is B C
define B is not D
define D is E F
define C is A F
find A F
```

The six pieces of knowledge generate the following graph:

It can be noted that this example contains circular reasoning: A is defined as C and vice versa. The reasoning procedure is as follows:

```
Start reasoning
Activate Dict(A): 1
Activate Dict(F): 1
Activate Node0:  0.148885
Update connection:  Dict(A) - Node0:  0.350000
Dict(A) already activated:  1.000000
Activate Node1:  0.007444
Update connection:  Node0 - Node1:  0.150000
Activate Dict(B): 0.001117
Update connection:  Node1 - Dict(B): 0.350000
Node1 already activated:  0.007444
Update connection:  Node1 - Dict(B): 0.300000
```

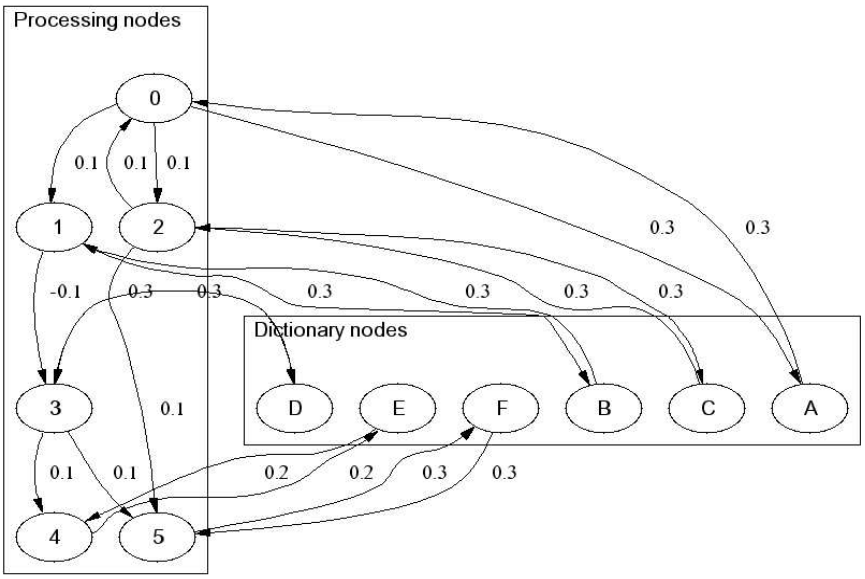

Figure 2. Memory model of circular reasoning.

```
Update connection:  Node0 - Node1:  0.100000
Activate Node2:  0.007444
Update connection:  Node0 - Node2:  0.150000
Activate Dict(C): 0.001117
Update connection:  Node2 - Dict(C): 0.350000
Node2 already activated:  0.007444
Update connection:  Node2 - Dict(C): 0.300000
Node0 already activated:  0.148885
Activate Node5:  0.000372
Update connection:  Node2 - Node5:  0.150000
Dict(F) already activated:  1.000000
Path found
New connection:  Node0 - Node5:  0.100000
Deactivate working memory nodes
```

```
End reasoning
```

The agent does not enter a reasoning loop because of its working memory that contains the activated nodes of its long-term memory. When an already visited node is encountered, the agent doesn't activate it again, but only recognizes the situation: "Node X already activated". The goal node is activated from the start. If an activated node is the solution, it means that a path has been found. Also, one can see that the connection weights on that path increase.

The following script shows a situation when a path cannot be found:

```
define A is B
define B is not C
find A C
find A C
```

The corresponding reasoning procedure is:

```
Start reasoning
Activate Dict(A): 1
Activate Dict(D): 1
Activate Node0:  0.099668
Update connection:  Dict(A) - Node0:  0.250000
Dict(A) already activated:  1.000000
Activate Node1:  0.004983
Update connection:  Node0 - Node1:  0.150000
Activate Dict(B): 0.000748
Update connection:  Node1 - Dict(B): 0.350000
Node1 already activated:  0.004983
Update connection:  Node1 - Dict(B): 0.300000
Connection:  Node1 - Node2:  -0.100000
A path may not be found
Update connection:  Node0 - Node1:  0.100000
Update connection:  Dict(A) - Node0:  0.200000
Path not found
```

```
New connection:  Node0 - Node2:  -0.100000
Deactivate working memory nodes
End reasoning
```

When the agent finds an inhibitory connection to a processing node linked to the goal node, it presents a warning: "A path may not be found". However, it continues the search because other paths may still exist. If the query is not very important, or if there is a time limit for the search, the agent may stop at this point and draw a negative conclusion. After the search, a direct inhibitory connection is created between A and C. The memory of the agent after the second search is presented in figure 3.

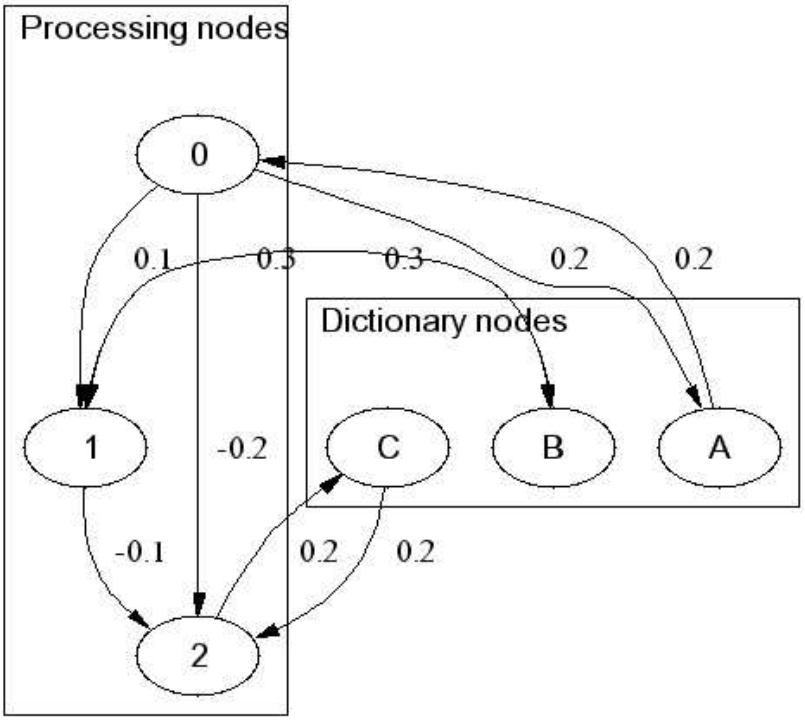

Figure 3. Memory model of negative statements.

The following script demonstrates the ability to learn of our cognitive model:

```
define A is B C
define B is D
define D is G H
define C is E F
find A F
forget
find A E
forget
find A H
forget
find A H
forget
define H is I
find A I
list
```

A simple, more intuitive structure of this knowledge base is presented in figure 4.

At first, when the agent must find a path from A to F, it visits the processing nodes corresponding to the following dictionary nodes: $A-B-D-G-H-C-E-F$. Because a path can be found ($A-C-F$), the connections weights between these nodes are increased. Thus, the connection weight between the processing nodes corresponding to A and B becomes smaller than the one corresponding to A and C. When the agent is requested to find a path between A and E, the order in which the nodes are visited changes: $A - C - F - E$.

The connection weight between the processing nodes corresponding to A and C is once again increased. When the agent must find the path between A and H, it first processes the link from A to C. It doesn't find a solution that way, so it later returns to B and finds the path to H. The nodes are activated in this order: $A-C-E-F-B-D-G-H$.

The second time, the A to B link is still smaller than the link

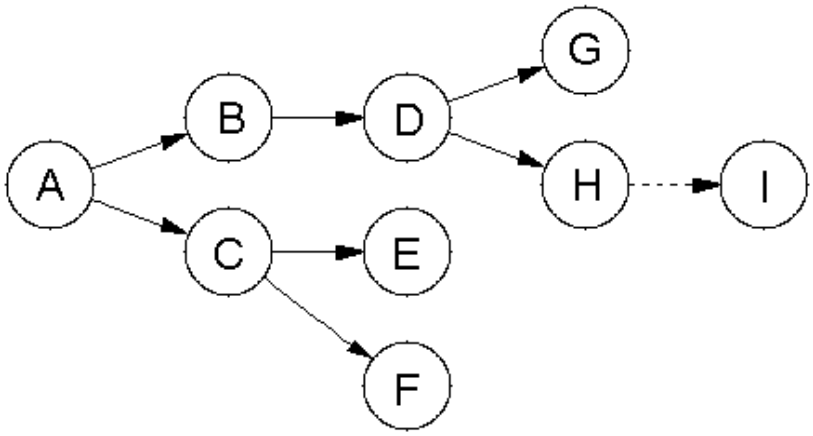

Figure 4. Simplified structure of the knowledge base.

from A to C. So is the newly created link between A and H. The only difference in the agent response is that G is no longer activated, because the link between D and H is now greater than the link between D and G. The activation order becomes: $A - C - E - F - B - D - H$.

A new concept I is introduced and linked to H. A path from A to I must go through H. But a direct link between A and H has been created the first time and reinforced the second time, so the activated nodes are only: $A - H - I$. This example clearly shows the learning capability of our cognitive model.

Of course, the knowledge base may contain natural language concepts, too. The following script demonstrates how these concepts are integrated into the memory of the agent.

```
define HUMAN is RATIONAL ANIMAL
define GEORGE is MAN
define MAN is MALE HUMAN
define GEORGE is TALL
define ANIMAL is BEING MOVE SENSE
```

```
define ANIMAL is not PLANT
find GEORGE RATIONAL
find GEORGE PLANT
find GEORGE SENSE
```

The agent can prove that George is rational, that he can sense, but cannot prove he is a plant. Figure 5 illustrates that even a simple natural language knowledge base requires a large number of memory nodes.

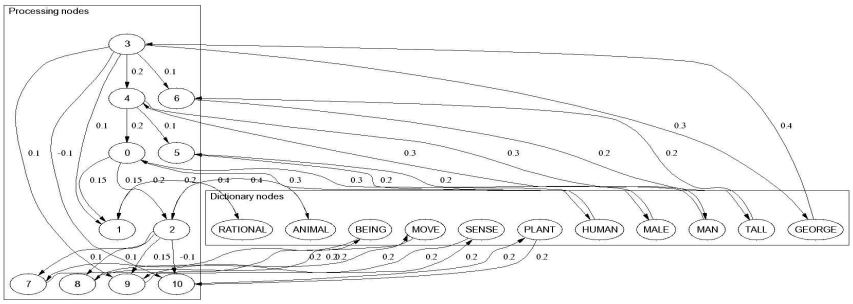

Figure 5. Memory model of natural language statements.

5 Conclusions

The model presented above proved successful in using a connectionist infrastructure for symbolic deductive reasoning, along with acquisition and integration of new facts into the existing knowledge base. This model can be used by an intelligent agent for gathering information about its environment and other agents in order to draw conclusions related to its tasks. The present model will be completed with the possibility of inductive reasoning. In this way, this general model can be the processing engine of a cognitive agent, with learning and planning capabilities.

References

[1] D.O.Hebb, The Organization of Behavior, Wiley & Sons, New York, 1949

[2] F.Leon F., D.Gâlea, M.Zbancioc, Knowledge Representation Through Interactive Networks, in Proceedings of the European Conference on Intelligent Systems, Iaşi, July 2002

[3] M.Miclea, Psihologie cognitivă, Polirom, Iaşi, 1999

[4] R.Penrose, Mintea noastră... cea de toate zilele - Despre gândire, fizică şi calculatoare, Ed. Tehnică, Bucureşti, 1996

[5] R.W.Sikes, Neuroanatomy, Northeastern University, Department of Physical Therapy, Boston, http://physicaltherapy.neu.edu/pth1366b/LectureSlides/ CerebralCortexIntro.ppt

[6] M.Zlate, Psihologia mecanismelor cognitive, Polirom, Iaşi, 1999

[7] Deductive Reasoning Application, http://eureka.cs.tuiasi.ro/~fleon/fldoctorat_e.htm

Mihai Horia Zaharia, Florin Leon, Dan Gâlea,

Department of Automatic Control
and Computer Engineering
Technical University "Gh. Asachi"
Iaşi

Received February 1, 2004