Heterogeneous Knowledge Representation: integrating connectionist and symbolic computations

Danilo Montesi*

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
d.montesi@doc.ic.ac.uk

Technical Report DOC 95/19

Abstract

Heterogeneous knowledge representation allows to combine several knowledge represent techniques. For instance connectionist and symbolic systems are two different computational paradigms and knowledge representation tools. Unfortunately, the integration of different paradigms and knowledge representations is not easy and very often is informal. In this paper, we propose a formal approach to integrate these two paradigms where as symbolic system we consider a (logic) rule based system. The integration is operated at language level between neural networks and rule languages. The formal model that allow the integration is based on constraint logic programming and provide an integrated framework to represent and process heterogeneous knowledge. In order to achieve this we define a new language that allow to express and model in a natural and intuitive way the above issues together with the operational semantics.

*The work of this author has been partially supported by the EU *Human Capital and Mobility* grant N. ERBCHBGCT930365 "Compulog-Group".

1 Introduction

Knowledge bases are designed to solve problems on a specific domain acquiring, representing and processing knowledge expressed in symbolic form. In general, computer science is confronted with two types of problems. They can be classified as *low entropy* and *high entropy* problems. By low entropy problems we mean problems which are clearly and completely defined through a programming language. This class deals with structured problems such as sorting, searching, data processing and deduction systems. Indeed, many knowledge bases use rule languages to infer new knowledge from previous one. For instance, from the fact that socrates is a human (`human(socrates)`) and the knowledge that any human is mortal (`mortal(X)` $\leftarrow$ `human(X)`) we can derive that socrates is mortal (`mortal(socrates)`). High entropy problems are those that do not allow a complete description of the problem or if such description is available it would be very complex and not appropriate to encode uncertainty. For instance, object and speech recognition are high entropy problems for which a complete description of the problem is not realistic [9]. Symbolic systems have been widely used to express and solve low entropy problems [2]. The main advantages of these systems are related to fact that rely on a logical framework and thus are declarative and with a formal semantics [12]. In addition, program optimization based on partial evaluation and semantics preserving transformation have solved many inefficiency of these systems [18]. Finally, they allow to construct a explanation of the result in term of premises/consequences. In short, symbolic systems rely on a well established knowledge representation and processing framework.

However, symbolic systems are not appropriate to handle high entropy problems where the description of the problem (or its solution) through a programming language is very difficult, time consuming and thus expensive. For instance, object recognition is hard to express with any programming languages (i.e., imperative, logic or functional). Imagine how difficult it would be to write a logic program that would recognize a chair among other objects in a room. The resulting program may not cover all the possible configurations due to a lack of the full description of the object to recognize or not being able to recognize a folding chair as the target object. Even if such program is available it would be extremely complex and not appropriate to recognize objects that are not encoded in the program. From here the need to have a computational paradigm which can learn and adapt to the source of objects it needs to recognize, and - most important - does not need a complete description of the problem through a programming language.

The above drawback of symbolic systems have lead to a new paradigm to solve high entropy problems. Such model has been called (Artificial) Neural

Networks (NN) since it present analogy with the principles of biological entities [7, 14]. Basically, a neural network consists of many processing neurons linked together through connections. Each neuron receives input signal via weighted incoming connections, and responds by sending a signal to all of the neurons it has outgoing connections to. Neural networks are superior with respect to rule based systems at problems involving a massive amount of uncertain and noisy data where it is important to extract the relevant items quickly. The learning capability with the distributed representation (over the neurons) make them candidates for such problems. Learning capability is not an exclusive feature of neural networks. Knowledge bases too have learning capability [4, 16, 11]. However, such capability is not enough in the contest of uncertain and noisy data such as object and speech recognition. Thus inductive logic programming is not appropriate for those problems. Another force of neural networks is their ability to generalize from examples. They will always give reasonable answers by interpolating from the examples (provided during training) in some way. The important features of NN approach are the powerful learning algorithms. Training instead that of programming will help to reduce the cost and time of software development in many cases where high entropy problems are involved. The distributed representation of data is very robust, insensitive to inaccuracies, noise incompleteness and even physical damage.

At this point we should clarify the relationship among neural network and rule languages. Both the approaches have the same expressive power since they are Turing complete. Thus any problem that can be solved with one approach can be solved with the other too. Obviously, the difference is not on the expressive power since they compute the same class of functions but on using the most appropriate paradigm to represent and solve a problem. Since the above two paradigms seem to complement each other they should be combined to solve complex problems [17]. Indeed, very often we need to handle both high and low entropy problems and this results into integrated paradigms to acquire, represent and process heterogeneous knowledge. Consider for instance the classic AI problem where a monkey, a chair and a banana are in a room in different positions. In order to take the banana that is hanged from the ceiling the monkey should recognize the banana, the chair, infer that it has to walk to the chair, take it, push it below the banana and jump on the chair to take the fruit. This implies a interleaving between high (connectionist) computation such as object recognition (i.e., banana and chair) and low entropy (symbolic) one such as deductive reasoning (i.e., if it is a banana, walk to the chair, push the chair below the banana, jump on the chair and then take the banana).

The contribution of this paper is to provide a formal model to integrate neural networks and rule languages following [15]. This integration allow to express the cooperation among connectionist and symbolic paradigms in a clear and sound way while preserving all the features of both these paradigms. Thus the rule language relies on the logical framework that need to be extended to express neural networks. The derivation of new knowledge through rules is expressed extending the logical framework to cope with neural networks while preserving the nice features of the logical systems in order to take full advantage of the body of results already developed in this area (see [21] for a survey of the state of the art). On the neural networks side we do not make any restriction on its topology, computational or learning properties. We do not aim to transform one paradigm into the other or to claim that one of them can be avoided [22]. Indeed, there are several approaches in literature that follow these approach. For instance some of them transform the neural networks into propositional logic [1] and show that are equivalent. Others transform the propositional logic into a neural network [20].

The basic idea of our approach is to use constraint logic programming (CLP(X)) to model the integration of rule languages (expressed as Horn clause language) with neural networks (expressed as constraints). The importance of a clear and sound integration between the two paradigms is obvious. The integration of these paradigms can succeed in solving complex problems that require heterogeneous knowledge bases [19]. There are two important points in this plan. The former is to use the well known paradigm of constraint logic programming and the already developed results [8]. The latter is that neural networks can be trained and tested as a stand alone component and then "plugged" into a rule system and cooperate with it. Thus the neural networks does not need any change to cooperate with the rule based system. The formal approach to integration rely on the fact that neural networks can be expressed as a set of equations and thus as constraints over an appropriate domain. The original purpose of CLP(X) was to integrate, in a clear and sound way, constraint and logic programming. In the general scheme X stands for a constraint domain. The integration of neural networks with rule languages is inherited from the CLP(X) schema where the constraint domain X will be specialized to express the neural network and the logic programming side expresses the rule language.

The rest of the paper is organized as follow. Section 2 introduces the basic idea as well as the general architecture for the resulting heterogeneous knowledge base. Sections 3 and 4 recall the relevant concepts on neural networks and constraint logic programming respectively. Section 5 provides the formal model to integrate the above two paradigms defining the resulting

language and its operational semantics. Finally, Section 6 concludes the paper and sketches some future applications.

2 An overview of the approach

There are several ways to combine two systems. For instance serial or parallel composition. Unfortunately, however, these form of composition do not allow to integrate the reasoning model of different paradigms. Ideally, an integrated model should allow to transform an input (K_1) into an output (K_t) through a derivation relation that can model both the logic (or symbolic) and connectionist reasoning. According to this vision we can define such computation as a finite derivation of the input into the output ($K_1 \vdash \ldots \vdash K_t$) according to some derivation relation $\vdash$. Since our system allow to combine heterogeneous knowledge, all the elements of the above computation have two components: the logic part is denoted with G and the connectionist one is denoted with N. Thus the above computation can be rewritten as $N^1 \mid G^1 \vdash \ldots \vdash N^n \mid G^m$. The symbol '|' has not meaning and is introduced just to separate the two components and will be used only in the examples. The derivations relation $\vdash$ can be denote more precisely with $\vdash_{L,C}$ to highlight the heterogeneous reasoning model. The above derivation shows the logic and connectionist reasoning expressed through the derivation relation. Under this approach any element of the derivation is derived from the previous one. For instance $N^k \mid G^r$ is derived from $N^i \mid G^j$ by means of a simple logic step (denoted $\vdash_L$) if $k = i$ and $r = j + 1$ or a simple connectionist step (denoted $\vdash_C$) if $k = i + 1$ and $r = j$ or a combined logic and connectionist step (denoted $\vdash_{L,C}$) if $k = i + 1$ and $r = j + 1$. The case of $\vdash_{L,C}$ is the most general and the others ($\vdash_L$ and $\vdash_C$) can be easily derived from this one. In the following $\vdash$ stands for any derivation relation (logic, connectionist or both). The general architecture of the resulting system is depicted in Figure 1. 1.

The above vision leads to several questions. How can we integrate the logic and the connectionist paradigms? What is a formal model for the resulting model? How can we preserve the nice features of both the paradigms? These questions will be answered in the rest of the paper after recalling the concepts of neural networks and constraint logic programming. For the time being, we note that constraint logic programming was developed to integrate constraint and logic programming in a clear and sound framework. Therefore it relies on a logical framework while preserving the nice features of logic programming such as declarativeness and formal semantics [8]. Since neural networks are expressed through equations we can express a whole network

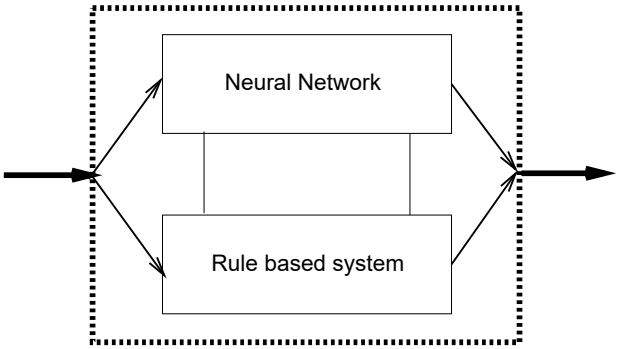

Figure 1: Coupling neural networks with rule based systems

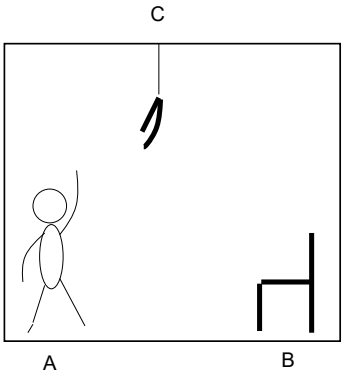

Figure 2: Monkey and banana problem

as a constraint program. Informally, this correspond to model the neural network as a set of equations. These equations will relate the inputs of a network with the outputs. The network should be trained to perform its task as a stand alone component and then plugged into the knowledge base. This is exemplified next leaving out some details.

Example 2.1 *Consider the monkey-banana problem described in Section 1 and depicted in Figure 2. The monkey, the chair and the banana have different positions that denoted with A, B and C respectively.*

Consider the following functions describing the state transformation (on the left) and the relations describing the problem (on the right)

$$\begin{array}{ll} \texttt{walk} : \texttt{position} \times \texttt{position} \times \texttt{state} \rightarrow \texttt{state} & \texttt{monkey}(\texttt{position}, \texttt{state}) \\ \texttt{push} : \texttt{position} \times \texttt{position} \times \texttt{state} \rightarrow \texttt{state} & \texttt{chair}(\texttt{position}, \texttt{state}) \\ \texttt{jump} : \texttt{state} \rightarrow \texttt{state} & \texttt{onthechair}(\texttt{state}) \end{array}$$

The objects to recognize are the banana and the chair. In addition we need to locate them assigning a position together with the monkey. This is a

suitable task for neural networks. Assume that we already have a network that would recognize bananas, chairs, and monkeys and locate them providing their position [10]. Such a network takes as input an image, and provide for the recognized objects their positions. For instance assume that this can be expressed with $\texttt{NN}(\texttt{Image}, \texttt{monkey}, \texttt{A})$ *in the case of the monkey. Then the following knowledge base allow to solve the monkey-banana problem through cooperation of neural networks and rule based system.*

f_1 $\texttt{monkey}(\texttt{A}, \texttt{s}_0) \leftarrow \texttt{NN}(\texttt{Image}, \texttt{monkey}, \texttt{A})$
f_2 $\texttt{chair}(\texttt{B}, \texttt{s}_0) \leftarrow \texttt{NN}(\texttt{Image}, \texttt{chair}, \texttt{B})$
r_1 $\texttt{taken}(\texttt{S}) \leftarrow \texttt{NN}(\texttt{Image}, \texttt{chair}, \texttt{C}) \mid \texttt{onthechair}(\texttt{C}, \texttt{S})$
r_2 $\texttt{onthechair}(\texttt{X}, \texttt{jump}(\texttt{S})) \leftarrow \texttt{monkey}(\texttt{X}, \texttt{S}), \texttt{chair}(\texttt{S}, \texttt{X})$
r_3 $\texttt{monkey}(\texttt{Y}, \texttt{walk}(\texttt{X}, \texttt{Y}, \texttt{S})) \leftarrow \texttt{monkey}(\texttt{X}, \texttt{S})$
r_4 $\texttt{monkey}(\texttt{Y}, \texttt{push}(\texttt{X}, \texttt{Y}, \texttt{S})) \leftarrow \texttt{monkey}(\texttt{X}, \texttt{S}), \texttt{chair}(\texttt{X}, \texttt{S})$
r_5 $\texttt{chair}(\texttt{Y}, \texttt{walk}(\texttt{X}, \texttt{Y}, \texttt{S})) \leftarrow \texttt{chair}(\texttt{X}, \texttt{S})$
r_6 $\texttt{chair}(\texttt{Y}, \texttt{push}(\texttt{X}, \texttt{Y}, \texttt{S})) \leftarrow \texttt{monkey}(\texttt{X}, \texttt{S}), \texttt{chair}(\texttt{X}, \texttt{S})$

Note that fact f_1 *asserts that monkey has position* A *and state* s_0*. The position is computed by the network that takes as input the image of Figure 2 and computes as output the recognized object and its position (using* $\texttt{NN}(\texttt{Image}, \texttt{monkey}, \texttt{A})$*). Similarly, in fact* f_2 *the chair and its position is computed. Rule* r_1 *states that* $\texttt{taken}(\texttt{S})$ *is true if the network recognize the banana, provides its position and this make* $\texttt{onthechair}(\texttt{C}, \texttt{S})$ *true. Rule* r_2 *says that the monkey can jump on the chair if it has the same position of the chair. The rest of the rules just allow the monkey and the chair to move under the banana.*

In the next two sections we introduce the necessary concepts of neural networks and constraint logic programming.

3 Neural Networks

An artificial neural network consists of a network of neurons that can be expressed in a weighted directed graph $G = (V, A, W)$, where V is the set of vertices, A a set of directed edges and W a set of edge weights [7]. Each vertex in the graph represents an artificial neuron. An important topological feature of a network is its layer structure. In layered networks V can be partitioned into a number of pairwise disjoint subsets $V = V^0 \cup V^1 \cup \ldots \cup V^L$ in such a way that the neurons in layer l only have edges to neurons in layer $l+1$ and $l-1$. We enumerate neuron number i within layer l by $v_i^{(l)}$ and the corresponding state by $x_i^{(l)}$. Since multi layered networks are the most general

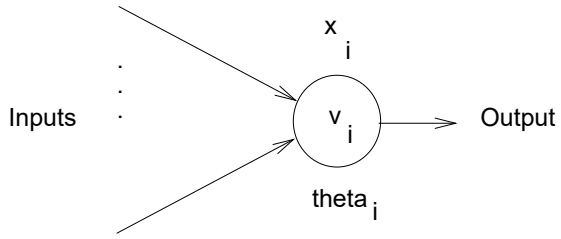

Figure 3: An artificial neuron

in the rest of the paper we will consider layered networks. Each neuron v_i, as shown in Figure 3, has two attributes: the *state* x_i and the *threshold* θ_i. These attributes can be discrete or continuous valued. The state vector defined by all the neuron is denote x. The corresponding state space is denoted Ω_S. The threshold is a measure of the input strength from neighbour neurons needed to activate neuron i. The edges of the graph are referred to as connections, as they connect the neurons. With each connection in the graph there is associated an attribute called the weight, representing the strength of the connection. Connection weights are usually real valued. A connection with a positive weight is called excitatory. A connection with a negative weight is called inhibitory. We denotes the connection from neuron j layer $l-1$ to neuron i in layer l by w_{ij}^l. The matrix W, where w_{ij} is the in row i, column j, is called the connectivity matrix. The corresponding weight space is denoted Ω_W. Layered networks have a separate matrix for each layer. The network's interface to the outside world is defined by partitioning the neurons into input neurons, output neurons and hidden neurons: $V = V_I \cup V_O \cup V_H$. This is reflected in the state vector where x_I, x_O and x_H denotes the corresponding states of the layers. The union of input and output neurons is called the visible units denoted $V_{I/O}$. The input and output neurons (V_I and V_O) are accessible to the outside world. The rest of the neurons, V_H are hidden from the outside world as shown in Figure 4.

Neural networks use the linear sum of states of neighbour neurons and connection weight in order to determine the next state as defined by

$$u_i^{(l)}(t) \quad = \quad \sum_{j=1}^{n} w_{ij}^{(l)} x_j^{(l-1)}(t) - \theta_i^{(l)}(t) \tag{1}$$

where l denotes the layer, $x_j^{(l-1)}(t)$ the state of the neuron j in layer $l-1$ at time t. The weight between neuron j and i (in layer l) is denoted with $w_{ij}^{(l)}$. The threshold of neuron i is $\theta_i^{(l)}(t)$. Neural networks can be described as dynamic systems and have two different operational modalities: learning and computation. As dynamical systems both learning and computation are described as processes of moving in space. Learning in neural networks

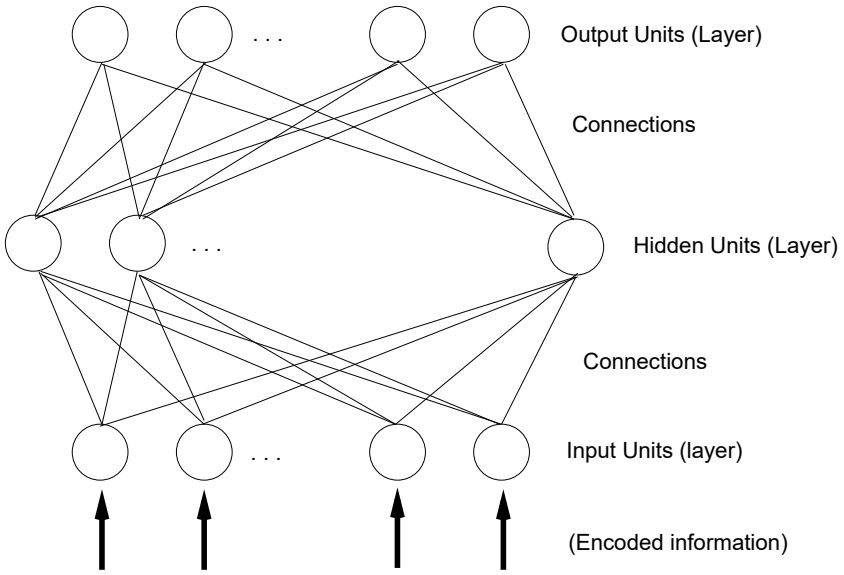

Figure 4: Multi layer Network

entails training the network by presenting training patterns to its visible neurons. The aim of training is to set the connection weights and neurons thresholds to some proper values, so that the desired network computation is obtained. The learning process is therefore a search in weight space Ω_W . A discussion on learning methods and algorithms is behind the scope of this paper. Once the weights are computed the network is ready to provide its expected behavior. Indeed, in the computation mode, the network computes a function (a mapping) from input space to output space. Given an input x_I, the network computes an output vector x_O, as determined by its connections W. The expected computation (or dynamic behavior) of the network is that every neuron applies some activation rule to enter a new state [1]. The activation rule is a function of the states of the neurons that have an outgoing connection to the neuron in question and the connection weights. A neuron applies its activation rule g_i to the formula (1) to determine the next internal state as defined by

$$x_i^{(l-1)}(t+1) = g_i(u_i^{(l)}(t)). \tag{2}$$

Let us now turn out attention to constraint logic programming.

[1]This can be done synchronous or asynchronous. There are also differences related to the type of network: feed-forward or recurrent but these issues are not relevant for our objective.

4 Constraint logic programming

Constraint logic programming (CLP) is a merge of two declarative paradigms: constraint solving and logic programming [8]. Viewing constraint logic programming rather broadly it involves the incorporation of constraints and constraint solving methods in a logic based language. This characterization suggests the possibility of many interesting languages, based on different constraints and different logics. However, work on CLP has almost exclusively been devoted to languages based on Horn clauses. We briefly introduce the general CLP(X) schema, where X suggest that many different constraints can be accommodated over a generic domain X. Specifically for any signature Σ, X is a pair: the domain of computation $\mathcal{D}$ and the constraints $\mathcal{L}$. A constraint logic program is a collection of rules of the form

$$h \leftarrow c, b_1, \ldots, b_n. \tag{3}$$

where c is the conjunction of constraints and $h, b_1, \ldots, b_n$ are ordinary atoms as in logic programming [12]. Sometimes we represent the rule by $h \leftarrow B$ where B is the collection of atoms and constraints in the body. A fact is a rule $h \leftarrow c$. A goal (or query) is a rule $\leftarrow c, b_1, \ldots, b_n$. The informal meaning of a rule is: "if $b_1, \ldots, b_n$ is true and c is solvable, then h is true". The notion of solvability is related to the domain X. For instance in CLP($\mathcal{R}$) [8] has arithmetic constraints and computes over the real numbers where the signature Σ has the function symbols $+$ and $*$, and the binary predicate symbols $=$, $<$ and $\leq$. If D is the set of real numbers then $\mathcal{D}$ interpret the symbols of Σ as usual (i.e., $+$ is interpreted as addition, etc). Then $\mathcal{L}$ are the constraints and $\mathcal{R} = (\mathcal{D}, \mathcal{L})$ is the constraint domain of arithmetic over the real numbers. If we omit from Σ the symbol $*$, then the corresponding constraint domain $\mathcal{R}_{Lin}$ is the constraint domain of linear arithmetic over real numbers. Similarly if we restrict Σ to $+, =$, we obtain the constraint domain $\mathcal{R}_{LinEqn}$, where the only constraints are linear equations. Thus the constraint domain is built selecting the relevant elements for Σ and providing an appropriate interpretation. Finally we will identify conjunction and multiset union.

5 The Logic(NN) language

To integrate neural networks with rule languages we note that equations (1) and (2) allow to express a single layer in a neural network. Thus a set of the above equations will allow to express a whole neural network. To this purpose we abstract from the details about topology network, learning algorithms and type of activation rule (continuous/discrete). This will allow

us to integrate any neural network regardless to the specific features into a rule language. According to the above view a neural network is expressed as a set of equations (or constraints) over real or boolean domain. Thus the set of constraints can be seen as a constraint program where the input variables are x_l and the output variable are x_0. Thus the constraint program of a network can be expressed as:

Program NEURAL NETWORK
Input: x^l
Output: x^0
begin
$u_i^l(t) = \sum_{j=1}^{n} w_{ij}^l x_j^{l-1}(t) - \theta_i^l(t)$
$\vdots$
$u_i^1(t) = \sum_{j=1}^{n} w_{ij}^1 x_j^0(t) - \theta_i^1(t)$
$x_i^{l-1}(t+1) = g_i(u_i^l(t))$
$\vdots$
$x_i^0(t+1) = g_i(u_i^l(t))$
end.

Once that the network is interpreted as constraint program it is simple to model its integration with a rule language. We need just to recall that the constraint logic programming allow to define constraints in rule bodies Thus the resulting rules have the form

$$h(x) \leftarrow NN(x^l, x^0), b_1(x), \ldots, b_n(x).$$

where $b_1(x), \ldots, b_n(x)$ and $h(x)$ are atoms and $NN(x^l, x^0)$ is a conjunction of constraints (the constraint program). The shared variables between the constraint part and the logic part of the above rule represents the *communication channels* between the connectionist and symbolic systems. In the monkey and banana example, the inputs of the network is the $Image$ that does not come from the logic part (in this specific case) but from the environment. The output variables of the network are: monkey, chair and banana and their positions: `A`, `B` and `C`. In this example the object recognition and its position is used in the logic part to deduce new information such as `take(S)`. Thus the shared variable among the constraint and the logic part (or the connectionist and the symbolic systems) `(A,B,C)` express the communication channels. Note that a fact is a rule with empty body and a query has empty head.

In the NEURAL NETWORK program we have not defined the activation function g_i. This is due to the fact that the activation function can be non linear. For instance in the Boltzmann machine, the state of a neuron can

only be specified as a probability. The probability that a neuron is active is then

$$P(x_i = 1) = \frac{1}{1 + exp(-2u_i/T)} \tag{4}$$

where T is the temperature. T may be viewed as a measure of the noise in the system and affect the probability distribution of states. Other examples of activation are: McCulloch-Pitts' rule, sigmoid type rule, simple linear rule and stepwise linear rule. Since we want to be parametric with respect to activation we will consider the activation function built in within the constraint domain.

Thus the resulting constraint domain is defined as a triple $(\mathcal{D}, g_i, \mathcal{L}_{\mathcal{NN}})$. where $\mathcal{R}_{LinEqn} = (\mathcal{D}, \mathcal{L}_{\mathcal{NN}})$ and g_i is an activation function. Finally, note that a test for the satisfability of constraints is required. The constraint c is said to be satisfiable iff $\mathcal{D} \models \exists c$ (where $\exists$ denotes the existential closure of formula). The constraints c is a conjunction of well formed formula built over the language $\mathcal{L}_{\mathcal{NN}}$ and represent a neural network. Thus the resulting language that expresses the integration of neural networks and rule languages is called $Logic(NN)$ and is a set of rules of the form (4) where the constraint domain is $\mathcal{R}_{LinEqn}$ extended with a uninterpreted function g_i. Therefore a heterogeneous knowledge base (HKB) is a set of rules expressed in $Logic(NN)$. In order to understand the computational model of the HKB we now turn our attention to the operational semantics of $Logic(NN)$.

5.1 Operational semantics

We present the operational semantics as a transition system on states. Consider the tuple $\langle A, C\rangle$ where A is a multiset of atoms and constraints and C is multiset of constraints. There is one other state, denoted by *fail*. We assume as give a computation rule which select a transition type and an appropriate element of A (if necessary) for each state. The transition system is also parameterized by a predicate *consistent*, which we will discuss later. An initial goal G for execution is represented as a state by $\langle G, \emptyset\rangle$. The transition in the transition systems are:

T_1 $\langle A \cup a, C\rangle \vdash_L \langle A \cup B, C \cup (a = h)\rangle$.

If a is selected by the computation rule, a is an atom, $h \leftarrow B$ is a rule of HKB, renamed to new variables, and h and a have the same predicate. The expression $a = h$ is an abbreviation for the conjunction of equations between corresponding arguments of a and h.

T_2 We say a is *rewritten* in this transition

$\langle A \cup a, C \rangle \vdash_L fail$

if a is selected by the computation rule, a is an atom and for every rule $h \leftarrow B$ of HKB, h and a have different predicates.

T_3 $\langle A \cup c, C \rangle \vdash_{coop} \langle A, C \cup c \rangle$

if c is selected by the computation rule and c is a constraint.

T_4 $\langle A, C \rangle \vdash_C \langle A, C \rangle$

if $consistent(C)$.

T_5 $\langle A, C \rangle \vdash_C fail$

if $\neg consistent(C)$.

The transitions T_1 and T_2 arise from resolution denoting the logic steps. T_3 introduces constraints into the constraint solver (which correspond to invoke the neural network) denoting the cooperation of the two systems. T_4 and T_5 test whether the constraints are consistent, that is, if the neural network computes a suitable outputs for the given inputs denoting connectionist steps. We write $\vdash$ to refer to a transition of arbitrary type. The predicate $consistent(C)$ expresses a test for consistency of C, that is, the outputs of the network are computed for some inputs. Usually it is defined by: $consistent(C)$ iff $\mathcal{D} \models \exists C$, that is, a consistency test over the extended $\mathcal{R}_{LinEqn}$ domain that model the neural computation.

6 Conclusion

We have presented a formal technique that allow to express heterogeneous knowledge bases integrating neural networks and rule languages. This allow the smooth integration of knowledge representation through logic and connectionist models. The resulting model is defined in the well known setting of constraint logic programming and has a formal model both in term of language and operational semantics. The later also provide a simple top-down execution model.

We believe that this approach is very promising for further investigations. From a practical point of view we are studying how to use the combined logic and connectionist paradigms for image indexing and retrieval in multimedia database systems [5, 6]. From a theoretical point of view we intend to extend it to allow modular rule bases and several neural networks to express more

complex problem using structured modular systems both in the connectionist and logic paradigms [3, 13].

Ackowledgement I would like to thank Ursula Iturraran for friutful discussions on artificial neural networks.

References

[1] A. Aiello, E. Burattini, and Guglielmo Tamburrini. Purely Neural, Rule-Based Diagnostic Systems. Technical Report 100/93/IC, Instituto di Cibernetica CNR, 1993.

[2] A. Barr and E. A. Feigenbaum. *The Handbook of Artificial Intelligence.* Addison-Wesley, 1982.

[3] M. Bugliesi, E. Lamma, and P. Mello. Modular Logic Programming. *Journal of Logic Programming*, 19/20:443–502, 1994.

[4] J. G. Carbonell. *Machine Learning: Paradigms and Methods.* The MIT Press, 1990.

[5] P. Costantinopoulos, J. Drakopoulos, and Y. Yeorgaroudakis. Retrieval of Multimedia Documents by Pictorial Content: A Prototype system. In *Proc. Int'l Conf. Multimedia Information Systems*, pages 35–48. MacGraw-Hill, 1991.

[6] A. E. Günhan. Neural Network based retrieval of information from Database Systems State of the Art and Future Trend. Technical Report TR 25/94, Department of Information Science, University of Bergen, 1994.

[7] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[8] J. Jaffar and M. J. Maher. Constraint Logic Programming: a Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[9] D. Jones and S. P. Franklin. Choosing a Network: matching the architecture to the application. In A. Maren et al., editor, *Handbook of Neural Computing Applications*, pages 219–232. Academic Press, 1990. ISBN 0125460902.

[10] J. Koh, M. Suk, and S. M.Bhandarkar. A multilayer self-organizing feature map for range image segmentation. *Journal of Neural Networks*, 8(1):67–86, 1995.

[11] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[12] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1987. Second edition.

[13] A. Maren. Hybrid and Complex Networks. In A. Maren et al., editor, *Handbook of Neural Computing Applications*, pages 203–218. Academic Press, 1990. ISBN 0125460902.

[14] E. Masson and Y. J. Wang. Introduction to computation and learning in artificial neural networks. *European Journal of Operational Research*, 47:1–28, 1990.

[15] D. Montesi. A Formal Model to Integrate Rule Languages and Artificial Neural Networks. In *Int. Symp. on Knowledge Acquisition, Representation and Processing*, 1995.

[16] S. Muggleton. *Inductive Aquisition of Expert Knowledge*. Addison-Wesley, 1990.

[17] S. Y. Nof. *Information and Collaboration Models of Integration*. Kluwer Academic Publishers, 1994. NATO ARW Series.

[18] A. Pettorossi and M. Proietti. Transformation of Logic Programs: a Foundation and Techniques. *Journal of Logic Programming*, 19/20:261–320, 1994.

[19] P. Smolensky, G. Legendre, and Y. Miyata. Integrating Connectionist and Symbolic Computation for the Theory of Language. In V. Honavar and L. Uhr, editors, *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, pages 509–530. Academic Press, 1994.

[20] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165, 1994.

[21] Special Issue: Ten years of Logic Programming. *Journal of Logic Programming*, volume 19/20. 1994.

[22] Y.Pao and D. J. Sobajic. Neural Networks and Knowledge Engineering. *IEEE Tran. on Knowledge and Data Eng.*, 3(2):185–192, 1991.