Applied Intelligence 11, 59–77 (1999)
© 1999 Kluwer Academic Publishers. Manufactured in The Netherlands.

The Connectionist Inductive Learning and Logic Programming System

ARTUR S. AVILA GARCEZ
Department of Computing, Imperial College, 180 Queen's Gate, SW7-2BZ, London, UK
aag@doc.ic.ac.uk

GERSON ZAVERUCHA
COPPE Engenharia de Sistemas e Computação-UFRJ, Caixa Postal: 68511, CEP: 21945-970, Rio de Janeiro, Brazil
gerson@cos.ufrj.br

Abstract. This paper presents the Connectionist Inductive Learning and Logic Programming System (C-IL^{2}P). C-IL^{2}P is a new massively parallel computational model based on a feedforward Artificial Neural Network that integrates inductive learning from examples and background knowledge, with deductive learning from Logic Programming. Starting with the background knowledge represented by a propositional logic program, a translation algorithm is applied generating a neural network that can be trained with examples. The results obtained with this refined network can be explained by extracting a revised logic program from it. Moreover, the neural network computes the stable model of the logic program inserted in it as background knowledge, or learned with the examples, thus functioning as a parallel system for Logic Programming. We have successfully applied C-IL^{2}P to two real-world problems of computational biology, specifically DNA sequence analyses. Comparisons with the results obtained by some of the main neural, symbolic, and hybrid inductive learning systems, using the same domain knowledge, show the effectiveness of C-IL^{2}P.

Keywords: theory refinement, machine learning, artificial neural networks, logic programming, computational biology

1. Introduction

The aim of neural-symbolic integration is to explore and benefit from the advantages that each paradigm confers. Among the advantages of artificial neural networks are its massive parallelism, inductive learning and generalization capabilities. On the other hand, symbolic systems can explain their inference process (e.g., through automatic theorem proving), and use powerful declarative languages for knowledge representation.

"It is generally accepted that one of the main problems in building Expert Systems (which are responsible for the industrial success of Artificial Intelligence) lies in the process of knowledge acquisition, known as the *knowledge acquisition bottleneck*" [1]. An alternative is the automation of this process using Machine Learning techniques [2]. Symbolic machine learning methods are usually more effective if they can exploit a background knowledge (incomplete domain theory). In contrast, neural networks have been successfully applied as a learning method from examples only (data learning) [3]. As a result, the integration of theory and data learning in neural networks seems to be a natural step towards more powerful training mechanisms [59].

Learning strategies can be classified as: learning from instruction, learning by deduction, learning by analogy, learning from examples and learning by observation and discovery [4]; the latter two are forms of inductive learning. The inductive learning task

employed in symbolic machine learning is to find hypotheses that are consistent with a background knowledge to explain a given set of examples. In general, these hypotheses are definitions of concepts described in some logical language. The examples are descriptions of instances and non-instances of the concept to be learned, and the background knowledge provides additional information about the examples and the concepts' domain knowledge [1].

In contrast to symbolic learning systems, neural networks' learning implicitly encodes patterns and their generalizations in the networks' weights, so reflecting the statistical properties of the trained data [5]. It has been indicated that neural networks can outperform symbolic learning systems, especially when data are noisy [3]. This result, due also to the massively parallel architecture of neural networks, contributed decisively to the growing interest in combining, and possibly integrating, neural and symbolic learning systems (see [6] for a clarifying treatment on the suitability of neural networks for the representation of symbolic knowledge).

Pinkas [7, 8] and Holldobler [9] have made important contributions to the subject of neural-symbolic integration, showing the capabilities and limitations of neural networks for performing logical inference. Pinkas defined a bi-directional mapping between symmetric neural networks and mathematical logic [10]. He presented a theorem showing a weak equivalence between the problem of satisfiability of propositional logic and minimizing energy functions; in the sense that for every *well-formed formula (wff)* a quadratic energy function can efficiently be found, and for every energy function there exists a *wff* (inefficiently found), such that the global minima of the function are exactly equal to the satisfying models of the formula. Holldobler presented a parallel unification algorithm and an automated reasoning system for first order Horn clauses, implemented in a feedforward neural network, called the *Connectionist Horn Clause Logic* (CHCL) *System*.

In [11], Holldobler and Kalinke presented a method for inserting propositional general logic programs [12] into three-layer feedforward artificial neural networks. They have shown that for each program $\mathcal{P}$, there exists a three-layer feedforward neural network $\mathcal{N}$ with binary threshold units that computes $T_{\mathcal{P}}$, the program's fixed point operator. If $\mathcal{N}$ is transformed into a recurrent network by linking the units in the output layer to the corresponding units in the input layer, it always settles down in a unique stable state when $\mathcal{P}$ is an acceptable program[1] [13, 14]. This stable state is the least fixed point of $T_{\mathcal{P}}$, that is identical to the unique stable model of $\mathcal{P}$, so providing a declarative semantics for $\mathcal{P}$ (see [15] for the stable model semantics of general logic programs).

In [16], Towell and Shavlik presented KBANN (*Knowledge-based Artificial Neural Network*), a system for rules' insertion, refinement and extraction from neural networks. They have empirically shown that knowledge-based neural networks' training based on the backpropagation learning algorithm [17] is a very efficient way to learn from examples and background knowledge. They have done so by comparing KBANN's performance with other hybrid, neural and purely symbolic inductive learning systems' (see [1] and [18] for a comprehensive description of symbolic inductive learning systems, including Inductive Logic Programming).

The *Connectionist Inductive Learning and Logic Programming* (C-IL^{2}P) system is a massively parallel computational model based on a feedforward artificial neural network that integrates inductive learning from examples and background knowledge, with deductive learning from Logic Programming. Starting with the background knowledge represented by a propositional general logic program, a translation algorithm is applied (see Fig. 1 (1)) generating a neural network that can be trained with examples (2). Furthermore, that neural network computes the stable model of the program inserted in it or learned with the examples, so functioning as a massively parallel system for Logic Programming (3). The result of refining the background knowledge with the training examples can be explained by extracting a revised logic program from the network (4). Finally, the knowledge extracted can be more easily analyzed by an expert that decides if it should feed the system once more, closing the learning cycle (5). The extraction step of C-IL^{2}P (4) is beyond the scope of this paper, and the interested reader is referred to [19].

In Section 2, we present a new translation algorithm from general logic programs ($\mathcal{P}$) to artificial neural networks ($\mathcal{N}$) with bipolar semi-linear neurons. The algorithm is based on Holldobler and Kalinke's translation algorithm from general logic programs to neural networks with binary threshold neurons [11]. We also present a theorem showing that $\mathcal{N}$ computes the fixed-point operator ($T_{\mathcal{P}}$) of $\mathcal{P}$. The theorem ensures that the translation algorithm is sound. In Section 3, we show that the result obtained in [11] still holds, i.e.,

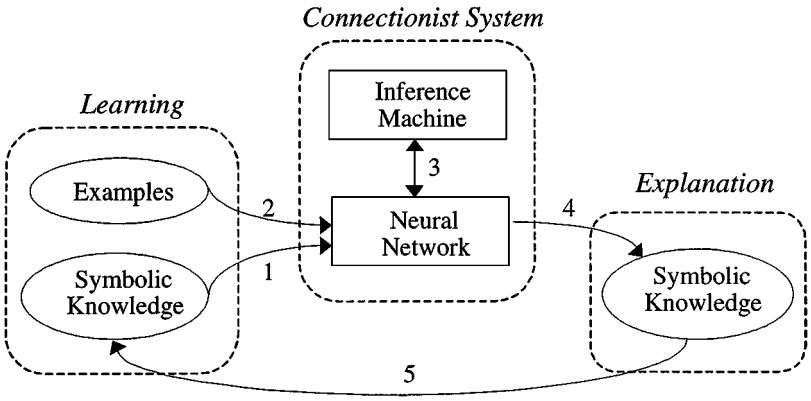

Figure 1. The connectionist inductive learning and logic programming system.

$\mathcal{N}$ is a massively parallel model for Logic Programming. However, $\mathcal{N}$ can also perform inductive learning from examples efficiently, assuming $\mathcal{P}$ as background knowledge and using the standard backpropagation learning algorithm as in [16]. We outline the steps for performing both deduction and induction in the neural network. In Section 4, we successfully apply the system to two real-world problems of DNA classification, which have become benchmark data sets for testing machine learning systems' accuracy. We compare the results with other neural, symbolic, and hybrid inductive learning systems. Briefly, C-IL^{2}P's test-set performance is at least as good as KBANN's and better than any other system investigated, while its training-set performance is considerably better than KBANN's. In Section 5, we conclude and discuss directions for future work.

2. From Logic Programs to Neural Networks

It has been suggested that the merging of theory (background knowledge) and data learning (learning from examples) in neural networks may provide a more effective learning system [16, 20]. In order to achieve this objective, one might first translate the background knowledge into a neural network initial architecture, and then train it with examples using some neural learning algorithm like backpropagation. To do so, the C-IL^{2}P system provides a translation algorithm from propositional (or grounded) general logic programs to feedforward neural networks with semi-linear neurons. A theorem then shows that the network obtained is equivalent to the original program, in the sense that what is computed by the program is computed by the network and vice-versa.

Definition 1. A general clause is a rule of the form $A \leftarrow L_1, \ldots, L_k$, where A is an atom and L_i $(1 \leq i \leq k)$ is a literal (an atom or the negation of an atom). A general logic program is a finite set of general clauses.

To insert the background knowledge, described by a general logic program ($\mathcal{P}$), in the neural network ($\mathcal{N}$), we use an approach similar to Holldobler and Kalinke's [11]. Each general clause (C_l) of $\mathcal{P}$ is mapped from the input layer to the output layer of $\mathcal{N}$ through one neuron (N_l) in the single hidden layer of $\mathcal{N}$. Intuitively, the translation algorithm from $\mathcal{P}$ to $\mathcal{N}$ has to implement the following conditions: (1) The input potential of a hidden neuron (N_l) can only exceed N_l's threshold (θ_l), activating N_l, when all the positive antecedents of C_l are assigned the truth-value "*true*" while all the negative antecedents of C_l are assigned "*false*"; and (2) The input potential of an output neuron (A) can only exceed A's threshold (θ_A), activating A, when at least one hidden neuron N_l that is connected to A is activated.

Example 2. Consider the logic program $\mathcal{P} = \{A \leftarrow B, C,\ not\ D;\ A \leftarrow E, F;\ B \leftarrow\}$. The translation algorithm should derive the network $\mathcal{N}$ of Fig. 2, setting weights ($W's$) and thresholds ($\theta's$) in such a way that conditions (1) and (2) above are satisfied. Note that, if $\mathcal{N}$ ought to be fully-connected, any other link (not shown in Fig. 2) should receive weight zero initially.

Note that, in Example 2, we have labelled each input and output neuron as an atom appearing, respectively, in the body and in the head of a clause of $\mathcal{P}$. This allows us to refer to neurons and propositional variables interchangeably and to regard each network's input vector $\mathbf{i} = (i_1, \ldots, i_m)$ $(i_{j(1 \leq j \leq m)} \in [-1, 1])$ as

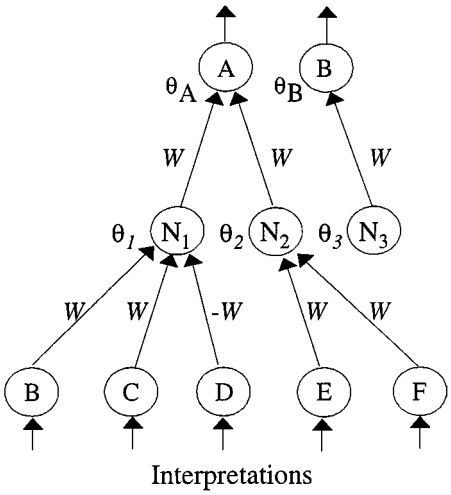

Figure 2. Sketch of a neural network for the above logic program $\mathcal{P}$.

an interpretation for $\mathcal{P}$.[2] If $i_j \in [A_{\min}, 1]$ then the propositional variable associated to the jth neuron in the network's input layer is assigned "*true*", while $i_j \in [-1, -A_{\min}]$ means that it is assigned "*false*", where $A_{\min} \in (0, 1)$ is a predefined value as shown in the notation below. Note also that each hidden neuron N_l corresponds to a clause C_l of $\mathcal{P}$.

The following notation will be used in our translation algorithm.

Notation: Given a general logic program $\mathcal{P}$, let q denote the number of clauses C_l $(1 \leq l \leq q)$ occurring in $\mathcal{P}$;
m, the number of literals occurring in $\mathcal{P}$;
$A_{\min}$, the minimum activation for a neuron to be considered "active" (or "*true*"), $0 < A_{\min} < 1$;
$A_{\max}$, the maximum activation for a neuron to be considered "not active" (or "*false*"), $-1 < A_{\max} < 0$;
$h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function, where β is the steepness parameter (that defines the slope of $h(x)$);
$g(x) = x$, the standard linear activation function;
W (resp. $-W$), the weight of connections associated with positive (resp. negative) literals;
θ_l, the threshold of hidden neuron N_l associated with clause C_l;
θ_A, the threshold of output neuron A, where A is the head of clause C_l;
k_l, the number of literals in the body of clause C_l;
p_l, the number of positive literals in the body of clause C_l;
n_l, the number of negative literals in the body of clause C_l;
μ_l, the number of clauses in $\mathcal{P}$ with the same atom in the head for each clause C_l;
$\max_{C_l}(k_l, \mu_l)$, the greater element among k_l and μ_l for clause C_l,
$\max_{\mathcal{P}}(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q)$, the greatest element among all k's and μ's of $\mathcal{P}$.

For instance, for the program $\mathcal{P}$ of Example 2, $q = 3, m = 6, k_1 = 3, k_2 = 2, k_3 = 0, p_1 = 2, p_2 = 2, p_3 = 0, n_1 = 1, n_2 = 0, n_3 = 0, \mu_1 = 2, \mu_2 = 2, \mu_3 = 1$, $\max_{C_1}(k_1, \mu_1) = 3$, $\max_{C_2}(k_2, \mu_2) = 2$, $\max_{C_3}(k_3, \mu_3) = 1$, and $\max_{\mathcal{P}}(k_1, k_2, k_3, \mu_1, \mu_2, \mu_3) = 3$.

In the translation algorithm below, we define $A_{\min} = \xi_1(k, \mu)$, $W = \xi_2(h(x), k, \mu, A_{\min})$, $\theta_l = \xi_3(k, A_{\min}, W)$, *and* $\theta_A = \xi_4(\mu, A_{\min}, W)$ such that conditions (1) and (2) are satisfied, as we will see later in the proof of Theorem 3.

Given a general logic program $\mathcal{P}$, consider that the literals of $\mathcal{P}$ are numbered from 1 to m such that the input and output layers of $\mathcal{N}$ are vectors of maximum length m, where the ith neuron represents the ith literal of $\mathcal{P}$. Assume, for mathematical convenience and without loss of generality, that $A_{\max} = -A_{\min}$.

1. Calculate $\max_{\mathcal{P}}(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q)$ of $\mathcal{P}$;
2. Calculate $A_{\min} > \frac{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)-1}{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)+1}$;
3. Calculate $W \geq \frac{2}{\beta} \cdot \frac{\ln(1+A_{\min})-\ln(1-A_{\min})}{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)(A_{\min}-1)+A_{\min}+1}$;
4. For each clause C_l of $\mathcal{P}$ of the form $A \leftarrow L_1, \ldots, L_k$ $(k \geq 0)$:
   (a) Add a neuron N_l to the hidden layer of $\mathcal{N}$;
   (b) Connect each neuron L_i $(1 \leq i \leq k)$ in the input layer to the neuron N_l in the hidden layer. If L_i is a positive literal then set the connection weight to W; otherwise, set the connection weight to $-W$;
   (c) Connect the neuron N_l in the hidden layer to the neuron A in the output layer and set the connection weight to W;
   (d) Define the threshold (θ_l) of the neuron N_l in the hidden layer as $\theta_l = \frac{(1+A_{\min})(k_l-1)}{2} W$
   (e) Define the threshold (θ_A) of the neuron A in the output layer as $\theta_A = \frac{(1+A_{\min})(1-\mu_l)}{2} W$.
5. Set $g(x)$ as the activation function of the neurons in the input layer of $\mathcal{N}$. In this way, the activation of the neurons in the input layer of $\mathcal{N}$, given by each input vector $\mathbf{i}$, will represent an interpretation for $\mathcal{P}$.

6. Set $h(x)$ as the activation function of the neurons in the hidden and output layers of $\mathcal{N}$. In this way, a gradient descent learning algorithm, such as *back-propagation*, can be applied on $\mathcal{N}$ efficiently.
7. If $\mathcal{N}$ ought to be fully-connected, set all other connections to zero.

Since $\mathcal{N}$ contains a bipolar semi-linear (differentiable) activation function $h(x)$, instead of a binary threshold non-linear activation function, the network's output neurons activations are real numbers in the range $[-1, 1]$. Therefore, we say that an output within the range $[A_{\min}, 1]$ represents the truth-value "*true*", while an output within $[-1, -A_{\min}]$ represents "*false*". We will see later in the proof of Theorem 3 that the above defined weights and thresholds do not allow the network to present activations in the range $(-A_{\min}, A_{\min})$.

Note that the translation of facts of $\mathcal{P}$ into $\mathcal{N}$, for instance $B \leftarrow$ in Example 2, is done by simply taking $k = 0$ in the above algorithm. Alternatively, each fact of the form $A \leftarrow$ may be converted to a rule of the form $A \leftarrow T$ that is inserted in $\mathcal{N}$ using $k = 1$, where T denotes "*true*", and is an extra neuron that is always active in the input layer of $\mathcal{N}$, i.e., T has input data fixed at "1". From the point of view of the computation of $\mathcal{P}$ by $\mathcal{N}$, there is absolutely no difference between the above two ways of inserting facts of $\mathcal{P}$ into $\mathcal{N}$. However, considering the subsequent process of inductive learning, regarding $\mathcal{P}$ as background knowledge, if $A \leftarrow$ is inserted in $\mathcal{N}$ then the set of examples to be learned afterwards can defeat that fact by changing weights and/or establishing new connections in $\mathcal{N}$. On the other hand, if $A \leftarrow T$ is inserted in $\mathcal{N}$ then A can not be defeated by the set of examples since the neuron T is clamped in $\mathcal{N}$. Defeasible and nondefeasible knowledge can therefore be respectively inserted in the network by defining variable and fixed weights.

The above translation algorithm is based upon the one presented in [11], where $\mathcal{N}$ is defined with binary threshold neurons. It is known that such networks have limited ability to learn. Here, in order to perform inductive learning efficiently, $\mathcal{N}$ is defined using the activation function $h(x)$. An immediate result is that $\mathcal{N}$ can also perform inductive learning from examples and background knowledge as in [16]. Moreover, the restriction imposed over W in [11], where it is shown that $\mathcal{N}$ computes $T_{\mathcal{P}}$ for $W = 1$, is weakened here, since the weights must be able to change during training.

Nevertheless, in [16], and more clearly in [21], the background knowledge must have a "*sufficiently small*" number of rules as well as a "*sufficiently small*" number of antecedents in each rule[3] in order to be accurately encoded in the neural network. Unfortunately, these restrictions become quite strong or even unfeasible if, for instance, $A_{\max} = \frac{1}{2}$ as in ([16], Section 5: Empirical Tests of KBANN). Consequently, an interpretation that does not satisfy a clause can wrongly activate a neuron in the output layer of $\mathcal{N}$. This results from the use of the standard (unipolar) semi-linear activation function, where each neuron's activation is in the range $[0, 1]$. Hence, in [16] both "*false*" and "*true*" are represented by *positive* numbers in the ranges $[0, A_{\max}]$ and $[A_{\min}, 1]$ respectively. For example, if $A_{\min} = 0.7$ and $k = 2$, an interpretation that assigns "*false*" to positive literals in the input layer of $\mathcal{N}$ can generate a *positive* input potential greater than the hidden neuron's threshold, wrongly activating the neuron in the output layer of $\mathcal{N}$.

In order to solve this problem we use bipolar activation functions, where each neuron's activation is in the range $[-1, 1]$. Now, an interpretation that does not satisfy a clause contributes *negatively* to the hidden neuron's input potential, since "*false*" is represented by a number in $[-1, -A_{\min}]$, while an interpretation that does satisfy a clause contributes *positively* to the input potential, because "*true*" is in $[A_{\min}, 1]$. Theorem 3 will show that the choice of a bipolar activation function is sufficient to solve the above problem. Furthermore, the choice of -1 instead of *zero* to represent "*false*" will lead to faster convergence in almost all cases. The reason is that the update of a weight connected to an input variable will be *zero* when the corresponding variable is *zero* in the training pattern [5, 22].

Thus making use of bipolar semi-linear activation function $h(x)$, let us see how we have obtained the values for the hidden and output neurons' thresholds θ_l and θ_A. To confer symmetric mathematical results, without loss of generality, assume that $A_{\max} = -A_{\min}$. From the input to the hidden layer of $\mathcal{N}$ ($L_1, \ldots, L_k \Rightarrow N_l$), if an interpretation satisfies $L_1, \ldots, L_k$ then the contribution of $L_1, \ldots, L_k$ to the input potential of N_l is greater than $I_+ = kA_{\min}W$. If, conversely, an interpretation does not satisfy $L_1, \ldots, L_k$ then the contribution of $L_1, \ldots, L_k$ to the input potential of N_l is smaller than $I_- = (p-1)W - A_{\min}W + nW$. Therefore, we define $\theta_l = \frac{I_+ + I_-}{2} = \frac{(1+A_{\min})(k-1)}{2}W$ (translation algorithm, step 4d). From the hidden to the output layer of $\mathcal{N}$ ($N_l \Rightarrow A$), if an interpretation satisfies N_l then the

contribution of N_l to the input potential of A is greater than $I_+ = A_{\min}W - (\mu - 1)W$. If, conversely, an interpretation does not satisfy N_l then the contribution of N_l to the input potential of A is smaller than $I_- = -\mu A_{\min}W$. Similarly, we define $\theta_A = \frac{I_+ + I_-}{2} = \frac{(1+A_{\min})(1-\mu)}{2}W$ (translation algorithm, step 4e). Obviously, $I_+ > I_-$ should be satisfied in both cases above. Therefore, $A_{\min} > \frac{k_l - 1}{k_l + 1}$ and $A_{\min} > \frac{\mu_l - 1}{\mu_l + 1}$ must be verified and, more generally, the condition imposed over $A_{\min}$ in the translation algorithm (step 2). Finally, given $A_{\min}$, the value of W (translation algorithm (step 3)) results from the proof of Theorem 3 below.

In what follows, we show that the theorem presented in [11], where $\mathcal{N}$ with binary threshold neurons computes the fixed point operator $T_\mathcal{P}$ of the program $\mathcal{P}$, still holds for $\mathcal{N}$ with semi-linear neurons. The following theorem ensures that our translation algorithm is sound. The function $T_\mathcal{P}$ mapping interpretations to interpretations is defined as follows. Let $\mathbf{i}$ be an interpretation and A an atom. $T_\mathcal{P}(\mathbf{i})(A) =$ "*true*" *iff there exists* $A \leftarrow L_1, \ldots, L_k$ *in* $\mathcal{P}$ *s.t.* $\bigwedge_{i=1}^{k} \mathbf{i}(L_i) =$ "*true*".

Theorem 3. *For each propositional general logic program* $\mathcal{P}$*, there exists a feedforward artificial neural network* $\mathcal{N}$ *with exactly one hidden layer and semi-linear neurons, obtained by the above "Translation Algorithm", such that* $\mathcal{N}$ *computes* $T_\mathcal{P}$*.*

Proof: We have to show that there exists $W > 0$ such that $\mathcal{N}$ computes $T_\mathcal{P}$. In order to do so we need to prove that given an input vector $\mathbf{i}$, each neuron A in the output layer of $\mathcal{N}$ is "active" if and only if there exists a clause of $\mathcal{P}$ of the form $A \leftarrow L_1, \ldots, L_k$ s.t. $L_1, \ldots, L_k$ are satisfied by interpretation $\mathbf{i}$. The proof takes advantage of the monotonically non-decreasing property of the bipolar semi-linear activation function $h(x)$, which allows the analysis to focus on the boundary cases. As before, we assume that $A_{\max} = -A_{\min}$ without loss of generality.

($\leftarrow$) $A \geq A_{\min}$ if $L_1, \ldots, L_k$ is satisfied by $\mathbf{i}$. Assume that the p positive literals in $L_1, \ldots, L_k$ are "true", while the n negative literals in $L_1, \ldots, L_k$ are "false". Consider the mapping from the input layer to the hidden layer of $\mathcal{N}$. The input potential (I_l) of N_l is minimum when all the neurons associated with a positive literal in $L_1, \ldots, L_k$ are at $A_{\min}$, while all the neurons associated with a negative literal in $L_1, \ldots, L_k$ are at $-A_{\min}$. Thus, $I_l \geq pA_{\min}W + nA_{\min}W - \theta_l$ and assuming $\theta_l = \frac{(1+A_{\min})(k-1)}{2}W$, $I_l \geq pA_{\min}W + nA_{\min}W - \frac{(1+A_{\min})(k-1)}{2}W$.

If $h(I_l) \geq A_{\min}$, i.e., $I_l \geq -\frac{1}{\beta}\ln(\frac{1-A_{\min}}{1+A_{\min}})$, then N_l is active. Therefore, Eq. (1) must be satisfied.

$$pA_{\min}W + nA_{\min}W - \frac{(1+A_{\min})(k-1)}{2}W \geq -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right) \quad (1)$$

Solving Eq. (1) for the connection weight (W) yields Eqs. (2) and (3), given that $W > 0$.

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1-A_{\min}) - \ln(1+A_{\min})}{k(A_{\min}-1) + A_{\min} + 1} \quad (2)$$

$$A_{\min} > \frac{k-1}{k+1} \quad (3)$$

Consider now the mapping from the hidden layer to the output layer of $\mathcal{N}$. By Eqs. (2) and (3) at least one neuron N_l that is connected to A is "active". The input potential (I_l) of A is minimum when N_l is at $A_{\min}$, while the other $\mu - 1$ neurons connected to A are at -1. Thus, $I_l \geq A_{\min}W - (\mu - 1)W - \theta_l$ and assuming $\theta_l = \frac{(1+A_{\min})(1-\mu)}{2}W$, $I_l \geq A_{\min}W - (\mu - 1)W - \frac{(1+A_{\min})(1-\mu)}{2}W$.

If $h(I_l) \geq A_{\min}$, i.e., $I_l \geq -\frac{1}{\beta}\ln(\frac{1-A_{\min}}{1+A_{\min}})$, then A is active. Therefore, Eq. (4) must be satisfied.

$$A_{\min}W - (\mu - 1)W - \frac{(1+A_{\min})(1-\mu)}{2}W \geq -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right) \quad (4)$$

Solving Eq. (4) for the connection weight W yields Eqs. (5) and (6), given that $W > 0$.

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1-A_{\min}) - \ln(1+A_{\min})}{\mu(A_{\min}-1) + A_{\min} + 1} \quad (5)$$

$$A_{\min} > \frac{\mu - 1}{\mu + 1} \quad (6)$$

($\rightarrow$) $A \leq -A_{\min}$ if $L_1, \ldots, L_k$ is not satisfied by $\mathbf{i}$. Assume that at least one of the p positive literals in $L_1, \ldots, L_k$ is "false" or one of the n negative literals in $L_1, \ldots, L_k$ is "true". Consider the mapping from the input layer to the hidden layer of $\mathcal{N}$. The input potential (I_l) of N_l is maximum when only one neuron associated to a positive literal in $L_1, \ldots, L_k$ is at $-A_{\min}$ or when only one neuron associated to a negative literal in $L_1, \ldots, L_k$ is at $A_{\min}$. Thus, $I_l \leq (p-1)W - A_{\min}W + nW - \theta_l$ or $I_l \leq (n-1)W - A_{\min}W + pW$

$-\theta_l$, respectively, and assuming $\theta_l = \frac{(1+A_{\min})(k-1)}{2}W$, $I_l \leq (k-1-A_{\min})W - \frac{(1+A_{\min})(k-1)}{2}W$.

If $-A_{\min} \geq h(I_l)$, i.e., $-A_{\min} \geq \frac{2}{1+e^{-\beta(I_l)}} - 1$, then $I_l \leq -\frac{1}{\beta}\ln(\frac{1+A_{\min}}{1-A_{\min}})$, and so N_l is not active. Therefore, Eq. (7) must be satisfied.

$$(k-1-A_{\min})W - \frac{(1+A_{\min})(k-1)}{2}W \leq -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right) \quad (7)$$

Solving Eq. (7) for the connection weight W yields Eqs. (8) and (9), given that $W > 0$.

$$W \geq \frac{2}{\beta}\cdot\frac{\ln(1+A_{\min}) - \ln(1-A_{\min})}{k(A_{\min}-1)+A_{\min}+1} \quad (8)$$

$$A_{\min} > \frac{k-1}{k+1} \quad (9)$$

Consider now the mapping from the hidden layer to the output layer of $\mathcal{N}$. By Eqs. (8) and (9), all neurons N_l that are connected to A are "not active". The input potential (I_l) of A is maximum when all the neurons connected to A are at $-A_{\min}$. Thus, $I_l \leq -\mu A_{\min}W - \theta_l$ and assuming $\theta_l = \frac{(1+A_{\min})(1-\mu)}{2}W$, $I_l \leq -\mu A_{\min}W - \frac{(1+A_{\min})(1-\mu)}{2}W$.

If $-A_{\min} \geq h(I_l)$, i.e., $-A_{\min} \geq \frac{2}{1+e^{-\beta(I_l)}} - 1$, then $I_l \leq -\frac{1}{\beta}\ln(\frac{1+A_{\min}}{1-A_{\min}})$, and so A is not active. Therefore, Eq. (10) must be satisfied.

$$-\mu A_{\min}W - \frac{(1+A_{\min})(1-\mu)}{2}W \leq -\frac{1}{\beta}\ln\left(\frac{1+A_{\min}}{1-A_{\min}}\right) \quad (10)$$

Solving Eq. (10) for the connection weight W yields Eqs. (11) and (12), given that $W > 0$.

$$W \geq \frac{2}{\beta}\cdot\frac{\ln(1+A_{\min}) - \ln(1-A_{\min})}{\mu(A_{\min}-1)+A_{\min}+1} \quad (11)$$

$$A_{\min} > \frac{\mu-1}{\mu+1} \quad (12)$$

Notice that Eqs. (2) and (5) are equivalent to Eqs. (8) and (11), respectively. Hence, the above theorem holds if for each clause C_l in $\mathcal{P}$ Eqs. (2) and (3) are satisfied by W and $A_{\min}$ from the input to the hidden layer of $\mathcal{N}$, while Eqs. (5) and (6) are satisfied by W and $A_{\min}$ from the hidden to the output layer of $\mathcal{N}$.

In order to unify the weights in $\mathcal{N}$ for each clause C_l of $\mathcal{P}$ given the definition of $\max_{C_l}(k_l, \mu_l)$, it is sufficient that Eqs. (13) and (14) below are satisfied by W and $A_{\min}$, respectively.

$$W \geq \frac{2}{\beta}\cdot\frac{\ln(1+A_{\min}) - \ln(1-A_{\min})}{\max_{C_l}(k_l,\mu_l)(A_{\min}-1)+A_{\min}+1} \quad (13)$$

$$A_{\min} > \frac{\max_{C_l}(k_l,\mu_l)-1}{\max_{C_l}(k_l,\mu_l)+1} \quad (14)$$

Finally, in order to unify all the weights in $\mathcal{N}$ for a program $\mathcal{P}$ given the definition of $\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)$, it is sufficient that Eqs. (15) and (16) are satisfied by W and $A_{\min}$, respectively.

$$W \geq \frac{2}{\beta}\cdot\frac{\ln(1+A_{\min}) - \ln(1-A_{\min})}{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)(A_{\min}-1)+A_{\min}+1} \quad (15)$$

$$A_{\min} > \frac{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)-1}{\max_{\mathcal{P}}(k_1,\ldots,k_q,\mu_1,\ldots,\mu_q)+1} \quad (16)$$

As a result, if Eqs. (15) and (16) are satisfied by W and $A_{\min}$, respectively, then $\mathcal{N}$ computes $T_{\mathcal{P}}$. □

Example 4. Consider the program $\mathcal{P} = \{A \leftarrow B, C, \textit{not } D;\ A \leftarrow E, F;\ B \leftarrow\}$. Converting fact $B \leftarrow$ to rule $B \leftarrow T$ and applying the *Translation Algorithm*, we obtain the neural network $\mathcal{N}$ of Fig. 3. Firstly, we

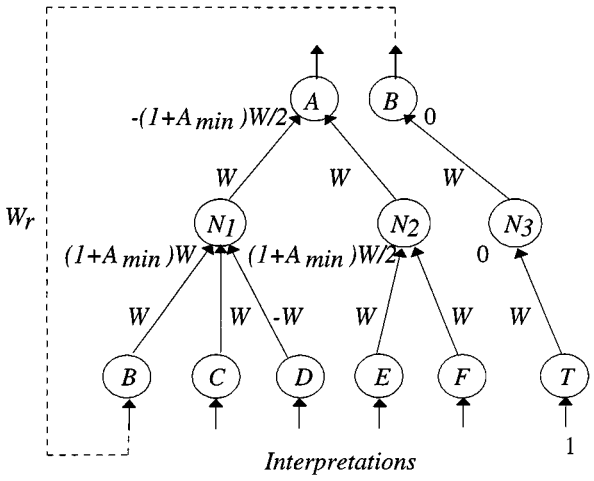

Figure 3. The neural network $\mathcal{N}$ obtained by the translation over $\mathcal{P}$. Connections with weight *zero* are not shown.

calculate $\max_{\mathcal{P}}(k_1, \ldots, k_n, \mu_1, \ldots, \mu_n) = 3$ (step 1), and $A_{\min} > 0.5$ (step 2). Then, suppose $A_{\min} = 0.6$, we obtain $W \geq 6.931/\beta$ (step 3). Alternatively, suppose $A_{\min} = 0.7$, then $W \geq 4.336/\beta$. Let us take $A_{\min} = 0.7$ and $h(x)$ as the standard bipolar semi-linear activation function ($\beta = 1$), then if $W = 4.5$, $\mathcal{N}$ computes the operator $T_{\mathcal{P}}$ of $\mathcal{P}$.[4]

In the above example, the neuron B appears at both the input and the output layers of $\mathcal{N}$. This indicates that there are at least two clauses of $\mathcal{P}$ that are linked through B (in the example: $A \leftarrow B, C,$ *not* D and $B \leftarrow$), defining a *dependency chain* [23]. We represent that chain in the network using the recurrent connection $W_r = 1$ to denote that the output of B must feed the input of B in the next learning or recall step. In this way, regardless of the length of the dependency chains in $\mathcal{P}$, $\mathcal{N}$ always contains a single hidden layer, thus obtaining a better learning performance.[5] We will explain in detail the use of recurrent connections in Section 3. In Section 4 we will compare the learning results of C-IL^{2}P with KBANN's, where the number of hidden layers is equal to the length of the greatest dependency chain in the background knowledge.

Remark 1. Analogously to [11], for any logic program $\mathcal{P}$, the time needed to compute $T_{\mathcal{P}}(\mathbf{i})$ in the network is constant; equal to two time steps (one to compute the activations from the input to the hidden neurons and another from the hidden to the output neurons). A parallel computational model requiring $p(n)$ processors and $t(n)$ time to solve a problem of size n is optimal if $p(n) \times t(n) = O(T(n))$, where $T(n)$ is the best sequential time to solve the problem [24]. The number of neurons and connections in the network that corresponds to a program $\mathcal{P}$ is given respectively by $O(q + r)$ and $O(q \cdot r)$, where q is the number of clauses and r is the number of propositional variables (atoms) occurring in $\mathcal{P}$. The sequential time to compute $T_{\mathcal{P}}(\mathbf{i})$ is bound to $O(q \cdot r)$, and so the above parallel computational model is optimal.

3. Massively Parallel Deduction and Inductive Learning

The neural network $\mathcal{N}$ can perform deduction and induction. In order to perform deduction, $\mathcal{N}$ is transformed into a partially recurrent network $\mathcal{N}_r$ by connecting each neuron in the output layer to its correspondent neuron in the input layer with weight $W_r = 1$,

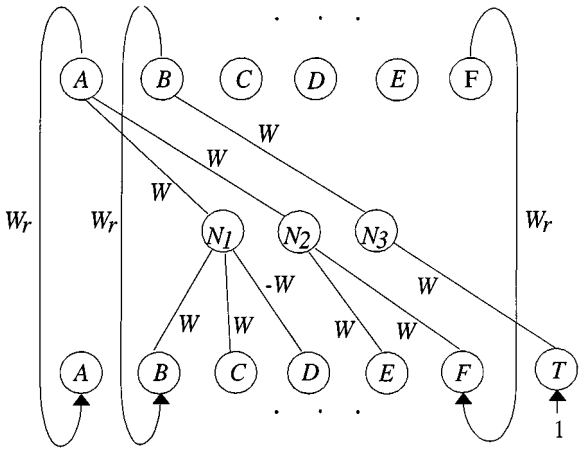

Figure 4. The recurrent neural network $\mathcal{N}_r$.

as shown in Fig. 4. In this way, $\mathcal{N}_r$ is used to iterate $T_{\mathcal{P}}$ in parallel, because its output vector becomes its input vector in the next computation of $T_{\mathcal{P}}$.

Let us now show that as in [11], if $\mathcal{P}$ is an acceptable program then $\mathcal{N}_r$ always settles down in a stable state that yields the unique fixed point of $T_{\mathcal{P}}$, since $\mathcal{N}_r$ computes the upward powers ($T^m_{\mathcal{P}}(\mathbf{i})$) of $T_{\mathcal{P}}$. A similar result could also be easily proved for the class of locally stratified programs (see [12]).

Definition 5 [13, 23]. Let $B_{\mathcal{P}}$ denote the *Herbrand base* of $\mathcal{P}$, i.e., the set of propositional variables (atoms) occurring in $\mathcal{P}$. A *level mapping* for a program $\mathcal{P}$ is a function $|\,|: B_{\mathcal{P}} \rightarrow \aleph$ of ground atoms to natural numbers. For $A \in B_{\mathcal{P}}$, $|A|$ is called the *level* of A and $|not\, A| = |A|$.

Definition 6 [13, 23]. Let $\mathcal{P}$ be a program, $|\,|$ a level mapping for $\mathcal{P}$, and $\mathbf{i}$ a model of $\mathcal{P}$. $\mathcal{P}$ is called *acceptable* w.r.t $|\,|$ and $\mathbf{i}$ if for every clause $A \leftarrow L_1, \ldots, L_k$ in $\mathcal{P}$ the following implication holds for $1 \leq i \leq k$: if $\mathbf{i} \models \bigwedge_{j=1}^{i-1} L_j$ then $|A| > |L_j|$.

Theorem 7 [14]. *For each acceptable general program $\mathcal{P}$, the function $T_{\mathcal{P}}$ has a unique fixed-point. The sequence of all $T^m_{\mathcal{P}}(\mathbf{i})$, $m \in \aleph$, converges to this fixed-point $T^{\omega}_{\mathcal{P}}(\mathbf{i})$ (which is identical to the stable model of $\mathcal{P}$ [15]), for each $\mathbf{i} \subseteq B_{\mathcal{P}}$.*

Recall that, since $\mathcal{N}_r$ has semi-linear neurons, for each real value o_i in the output vector ($\mathbf{o}$) of $\mathcal{N}_r$, if $o_i \geq A_{\min}$ then the corresponding ith atom in $\mathcal{P}$ is

assigned "*true*", while $o_i \leq A_{\max}$ means that it is assigned "*false*".

Corollary 8. *Let $\mathcal{P}$ be an acceptable general program. There exists a recurrent neural network $\mathcal{N}_r$ with semi-linear neurons such that, starting from an arbitrary initial input, $\mathcal{N}_r$ converges to a stable state and yields the unique fixed-point $(T_{\mathcal{P}}^{\omega}(\mathbf{i}))$ of $T_{\mathcal{P}}$, which is identical to the stable model of $\mathcal{P}$.*

Proof: Assume that $\mathcal{P}$ is an acceptable program. By Theorem 3, $\mathcal{N}_r$ computes $T_{\mathcal{P}}$. Recurrently connected, $\mathcal{N}_r$ computes the upwards powers $(T_{\mathcal{P}}^{m}(\mathbf{i}))$ of $T_{\mathcal{P}}$. By Theorem 7, $\mathcal{N}_r$ computes the unique stable model of $\mathcal{P}$ $(T_{\mathcal{P}}^{\omega}(\mathbf{i}))$. □

Hence, in order to use $\mathcal{N}$ as a massively parallel model for Logic Programming, we simply have to follow two steps: (i) add neurons to the input and output layers of $\mathcal{N}$, allowing it to be partially recurrently connected; (ii) add the correspondent recurrent links with fixed weight $W_r = 1$.

Example 9 (Example 4 continued). Given any initial activation in the input layer of $\mathcal{N}_r$ (Fig. 4), it always converges to the following stable state: $A =$ "false", $B =$ "true", $C =$ "false", $D =$ "false", $E =$ "false", and $F =$ "false", that represents the unique stable model of $\mathcal{P}$: $M(\mathcal{P}) = \{B\}$.

One of the main features of artificial neural networks is their learning capability. The program $\mathcal{P}$, viewed as background knowledge, may now be refined with examples in a neural training process on $\mathcal{N}_r$. Hornik et al. [25] have proved that standard feedforward neural networks with as few as a single hidden layer are capable of approximating any (Borel measurable) function from one finite dimensional space to another, to any desired degree of accuracy, provided sufficiently many hidden units are available. Hence, we can train single hidden layer neural networks to approximate the operator $T_{\mathcal{P}}$ associated with a logic program $\mathcal{P}$. Powerful neural learning algorithms have been established theoretically and applied extensively in practice. These algorithms may be used to learn the operator $T_{\mathcal{P}'}$ of a previously unknown program $\mathcal{P}'$, and therefore to learn the program $\mathcal{P}'$ itself. Moreover, DasGupta and Schinitger [26] have proved that neural networks with continuously differentiable activation functions are capable of computing a certain family of boolean functions with constant size (n), while networks composed of binary threshold functions require at least $O(\log(n))$ size. Hence, analog neural networks have more computational power than discrete neural networks, even when computing boolean functions.

The network's recurrent connections contain fixed weights $W_r = 1$, with the sole purpose of ensuring that the output feed the input in the next learning or recall process. As $\mathcal{N}_r$ does not learn in its recurrent connections,[6] the standard *backpropagation* learning algorithm can be applied directly [22] (see also [27]). Hence, in order to perform inductive learning with examples on $\mathcal{N}_r$, four simple steps should be followed: (i) add neurons to the input and output layers of $\mathcal{N}_r$, according to the training set (the training set may contain concepts not represented in the background knowledge and vice-versa); (ii) add neurons to the hidden layer of $\mathcal{N}_r$, if it is so required for the learning algorithm convergence; (iii) add connections with weight zero, in which $\mathcal{N}_r$ will learn new concepts; (iv) perturb the connections by adding small random numbers to its weights in order to avoid learning problems caused by symmetry.[7] The implementation of steps (i)–(iv) will become clearer in Section 4, where we describe some applications of the C-IL^{2}P system using *backpropagation*.

Remark 2. The final stage of the C-IL^{2}P system is the symbolic knowledge extraction from the trained network. It is generally accepted that "*rules' extraction*" algorithms can provide the so called explanation capability for trained neural networks. The lack of explanation for their reasoning mechanisms is one of neural networks' main drawbacks. Similarly, the lack of clarity of trained networks has been a main reason for serious criticisms. The extraction of symbolic knowledge from trained networks can considerably ameliorate these problems. It makes the knowledge learned accessible for an expert's analysis and allows for justification of the decision making process. The knowledge extracted can be directly added to the knowledge base or used in the solution of analogous domains problems.

Symbolic knowledge extraction from trained networks is an extensive topic on its own. Some of the main extraction proposals include [11, 20, 28, 29, 30, 31, 32, 33] (see [34] for a comprehensive survey). The main problem of the extraction task can be summarized as the quality × complexity trade-off, where the higher the quality of the extracted rule set, the higher the complexity of the extraction algorithm. In the context of the C-IL^{2}P system, the extraction task is defined as follows. Assume that after learning, $\mathcal{N}$ encodes a

knowledge $\mathcal{P}'$ that contains the background knowledge $\mathcal{P}$ expanded or revised by the knowledge learned with training examples. An *accurate extraction* procedure derives $\mathcal{P}'$ from $\mathcal{N}$ *iff* $\mathcal{N}$ computes $T_{\mathcal{P}'}$.

The extraction step of C-IL^{2}P is beyond the scope of this paper, and the interested reader is referred to [19]. However, we would like to point out that there is a major conceptual difference between our approach and other extraction methods. We are convinced that an extraction method must consider *default negation* in the final rule set, and not only "*if then else* " rules. Neural networks' behavior is commonly nonmonotonic [35], and therefore we can not expect to map it properly into a set of rules composed of Horn clauses only.

4. Experimental Results

We have applied the C-IL^{2}P system in two real-world problems in the domain of Molecular Biology; in particular the "*promoter recognition*" and "*splice-junction determination*" problems of DNA sequence analysis.[8] Molecular Biology is an area of increasing interest for computational learning systems analysis and application. Specifically, DNA sequence analysis problems have recently become a benchmark for learning systems' performance comparison. In this section we compare the experimental results obtained by C-IL^{2}P with a variety of learning systems.

In what follows we briefly introduce the problems in question from a computational application perspective (see [36] for a proper treatment on the subject). A DNA molecule contains two strands that are linear sequences of nucleotides. The DNA is composed from four different nucleotides—*adenine, guanine, thymine, and cytosine*—which are abbreviated by a, g, t, c, respectively. Some sequences of the DNA strand, called genes, serve as blueprint for the synthesis of proteins. Interspersed among the genes are segments, called noncoding regions, that do not encode proteins.

Following [16] we use a special notation to identify the location of nucleotides in a DNA sequence. Each nucleotide is numbered with respect to a fixed, biologically meaningful, reference point. Rules' antecedents of the form "*@3 atcg*" state the location relative to the reference point in the DNA, followed by the sequence of symbols that must occur. For example, "*@3 atcg*" means that an *a* must appear three nucleotides to the right of the reference point, followed by a *t* four nucleotides to the right of the reference point and so on. By convention, location zero is not used, while '$*$' means that any nucleotide will suffice in a particular location. In this way, a rule of the form *Minus*35 $\leftarrow$ @$-$36'ttg_*ca' is a short representation for *Minus*35 $\leftarrow$ @$-$36't', @$-$35't', @$-$34'g', @$-$32'c', @$-$31'a'. Each location is encoded in the network by four input neurons, representing nucleotides a, g, t and c, in this order. The rules are therefore inserted in the network as depicted in Fig. 5 for the hypothetical rule *Minus*5 $\leftarrow$ @$-$1'gc', @5't'.

In addition to the reference point notation, Table 1 specifies a standard notation for referring to all possible combinations of nucleotides using a single letter. This notation is compatible with the EMBL, GenBank, and

Table 1. Single-letter codes for expressing uncertain DNA sequence.

Code	Meaning	Code	Meaning	Code	Meaning
m	*a* or *c*	*r*	*a* or *g*	*W*	*a* or *t*
s	*c* or *g*	*y*	*c* or *t*	*K*	*g* or *t*
v	*a* or *c* or *g*	*h*	*a* or *c* or *t*	*D*	*a* or *g* or *t*
b	*c* or *g* or *t*	*x*	*a* or *g* or *c* or *t*		

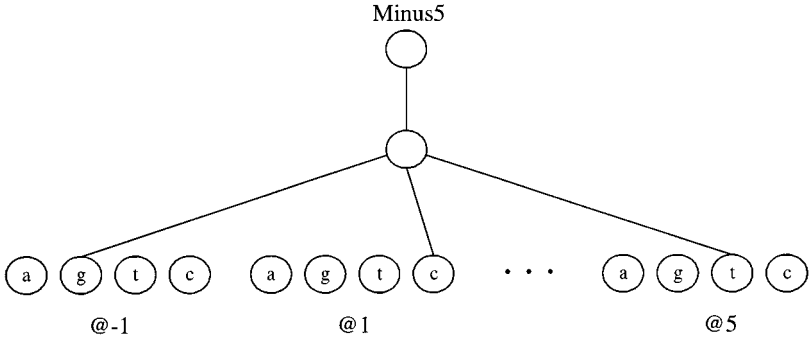

Figure 5. Inserting rule *Minus*5 $\leftarrow$ @$-$1'gc', @5't' into the neural network.

Table 2. Background knowledge for promoter recognition.

$Promoter \leftarrow contact, conformation$
$Contact \leftarrow Minus10, Minus35$
$Minus10 \leftarrow @-14\text{`}tataat\text{'}$ $\quad Minus35 \leftarrow @-37\text{`}cttgac\text{'}$
$Minus10 \leftarrow @-13\text{`}tataat\text{'}$ $\quad Minus35 \leftarrow @-36\text{`}ttgaca\text{'}$
$Minus10 \leftarrow @-13\text{`}ta_*a_*t\text{'}$ $\quad Minus35 \leftarrow @-36\text{`}ttgac\text{'}$
$Minus10 \leftarrow @-12\text{`}ta_{***}t\text{'}$ $\quad Minus35 \leftarrow @-36\text{`}ttg_*ca\text{'}$
$Conformation \leftarrow @-45\text{`}aa_{**}a\text{'}$
$Conformation \leftarrow @-45\text{`}a_{***}a\text{'}, @-28\text{`}t_{***}t_*aa_{**}t\text{'}, @-4\text{`}t\text{'}$
$Conformation \leftarrow @-49\text{`}a_{****}t\text{'}, @-27\text{`}t_{****}a_{**}t_*tg\text{'}, @-1\text{`}a\text{'}$
$Conformation \leftarrow @-47\text{`}caa_*tt_*ac\text{'}, @-22\text{`}g_{***}t_*c\text{'}, @-8\text{`}gcgcc_*cc\text{'}$

PIR data libraries—three major collections of data for molecular biology.

The first application in which we test C-IL^{2}P is the prokaryotic[9] promoter recognition. Promoters are short DNA sequences that precede the beginning of genes. The aim of "*promoter recognition*" is to identify the starting location of genes in long sequences of DNA. Table 2 contains the background knowledge for promoter recognition.[10]

The background knowledge of Table 2 is translated by C-IL^{2}P's translation algorithm to the neural network of Fig. 6. In addition, two hidden neurons are added in order to facilitate the learning of new concepts from examples. Note that the network is fully-connected, but low-weighted links are not shown in the figure. The network's input vector for this task contains 57 consecutive DNA nucleotides. The training examples consist of 53 promoter and 53 nonpromoter DNA sequences.

The second application that we use to test C-IL^{2}P is eukaryotic[11] splice-junction determination. Splice-junctions are points on a DNA sequence at which the non-coding regions are removed during the process of protein synthesis. The aim of "*splice-junction determination*" is to recognize the boundaries between the part of the DNA retained after splice—called exons—and the part that is spliced out—the introns. The task consists therefore of recognizing exon/intron (E/I) boundaries and intron/exon (I/E) boundaries. Table 3 contains the background knowledge for splice junction determination.[12]

The background knowledge of Table 3 is translated by C-IL^{2}P to the neural network of Fig. 7. In Table 3, "$\Leftarrow$" indicates nondefeasible rules, which can not be altered during training. Therefore, the weights set in the network by these rules are fixed. Rules of the form "*m of* (...)" are satisfied if at least m of the parenthesized concepts are true. Note that the translation of these rules to the network is done by simply defining $k_l = m$ in C-IL^{2}P's translation algorithm. Rules containing symbols other than the original (a, g, t, c) are split into a number of equivalent rules containing only the original symbols, according to Table 1. For example, since $y \equiv c \vee t$, the rule $IE \leftarrow pyramidine\_rich, @-3\text{`}yagg\text{'}, not\ IE\_Stop$ is

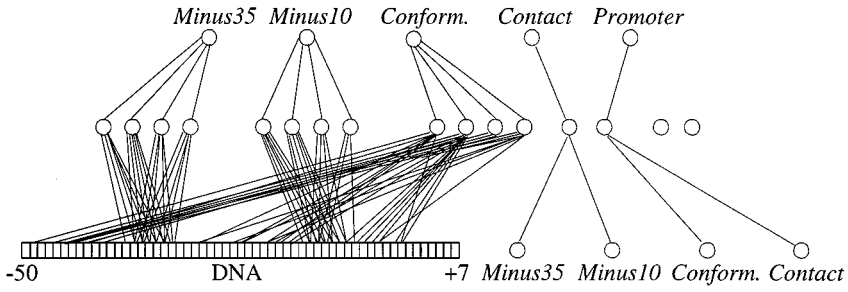

Figure 6. Initial neural network for promoter recognition. Each box at the input layer represents one sequence location that is encoded by four input neurons $\{a, g, t, c\}$.

Table 3. Background knowledge for splice-junction.

$EI \leftarrow @-3\text{`}maggtragt\text{'}, not\ EI\_Stop$		
$EI\_Stop \Leftarrow @-3\text{`}taa\text{'}$	$EI\_Stop \Leftarrow @-4\text{`}taa\text{'}$	$EI\_Stop \Leftarrow @-5\text{`}taa\text{'}$
$EI\_Stop \Leftarrow @-3\text{`}tag\text{'}$	$EI\_Stop \Leftarrow @-4\text{`}tag\text{'}$	$EI\_Stop \Leftarrow @-5\text{`}tag\text{'}$
$EI\_Stop \Leftarrow @-3\text{`}tga\text{'}$	$EI\_Stop \Leftarrow @-4\text{`}tga\text{'}$	$EI\_Stop \Leftarrow @-5\text{`}tga\text{'}$
$IE \leftarrow pyramidine\_rich, @-3\text{`}yagg\text{'}, not\ IE\_Stop$		
$pyramidine\_rich \leftarrow 6\ of\ (@-15\text{`}yyyyyyyyyy\text{'})$		
$IE\_Stop \Leftarrow @1\text{`}taa\text{'}$	$IE\_Stop \Leftarrow @2\text{`}taa\text{'}$	$IE\_Stop \Leftarrow @3\text{`}taa\text{'}$
$IE\_Stop \Leftarrow @1\text{`}tag\text{'}$	$IE\_Stop \Leftarrow @2\text{`}tag\text{'}$	$IE\_Stop \Leftarrow @3\text{`}tag\text{'}$
$IE\_Stop \Leftarrow @1\text{`}tga\text{'}$	$IE\_Stop \Leftarrow @2\text{`}tga\text{'}$	$IE\_Stop \Leftarrow @3\text{`}tga\text{'}$

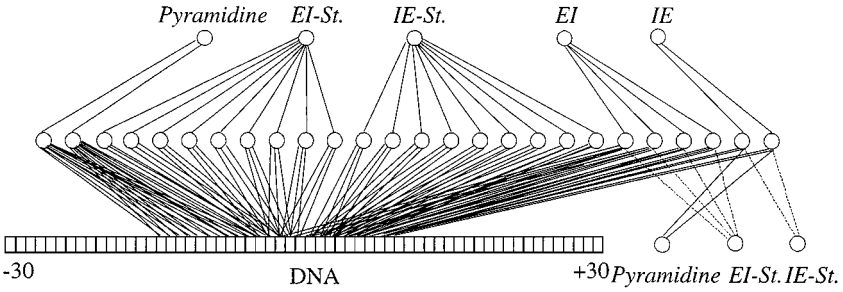

Figure 7. Initial neural network for splice-junction determination. Each box at the input layer of the network represents one sequence location which is encoded by four input neurons $\{a, g, t, c\}$.

encoded in the network as $IE \leftarrow pyramidine\_rich, @-3\text{`}cagg\text{'}, not\ IE\_Stop$ and $IE \leftarrow pyramidine\_rich, @-3\text{`}tagg\text{'}, not\ IE\_Stop$.

The training set for this task contains 3190 examples, in which approximately 25% are of I/E boundaries, 25% are of E/I boundaries and the remaining 50% are neither. The third category (neither E/I nor I/E) is considered true when neither I/E nor E/I output neurons are active. Each example is a DNA sequence with 60 nucleotides, where the center is the reference point. Remember that the network of Fig. 7 is fully-connected, but that low-weighted links are not shown. Dotted lines indicate links with negative weights.

In both applications, unless stated otherwise, the background knowledge is assumed defeasible, i.e., the weights are allowed to change during the learning process. Hence, some of the background knowledge may be revised by the training examples. Note however that the networks' recurrent connections are responsible for reinforcing the background knowledge during training. For instance, in the network of Fig. 7 the concepts *Pyramidine*, *EI-St.* and *IE-St.*, called intermediate concepts, have their input values calculated by the network in action, according to the background knowledge and to the DNA sequence input vector.

Let us now describe the experimental results obtained by C-IL^{2}P in the applications above. We compare it with other symbolic, neural and hybrid learning systems. Briefly, our tests show that C-IL^{2}P is a very effective system. Its test set performance is at least as good as KBANN's, and therefore better than any method analyzed in [16]. Moreover, C-IL^{2}P's training set performance is considerably superior to KBANN's, mainly because it always encodes the background knowledge in a single hidden layer network.

Firstly, let us consider C-IL^{2}P's test-set performance, i.e., its ability to generalize over examples not seen during training. We compare the results obtained by C-IL^{2}P in both applications with some of the main inductive learning systems from examples: Backpropagation [17], Perceptron [37] (neural systems), ID3 [38], and Cobweb [39] (symbolic systems). We also compare the results in the promoter recognition problem with a method suggested by biologists [40]. In addition, we compare C-IL^{2}P with systems that learn from both examples and background knowledge: Either [41],

Labyrinth-K [42], FOCL [43] (symbolic systems), and KBANN [16] (hybrid system).[13]

As in [16], we evaluate the systems using *cross-validation*, a testing methodology in which the set of examples is permuted and divided into n sets. One division is used for testing and the remaining $n - 1$ divisions are used for training. The testing division is never seen by the learning algorithm during the training process. The procedure is repeated n times so that every partition is used once for testing. For the 106-examples promoter data set, we use leaving-one-out cross-validation, in which each example is successively left out of the training set. Hence, it requires 106 training phases, in which the training set has 105 examples and the testing set has 1 example. Leaving-one-out becomes expensive as the number of available examples grows. Therefore, following [16], we use 10-fold cross-validation for the 1000-examples splice-junction determination data set.[14]

The learning systems that are based on neural networks have been trained until one of the following three stopping criteria was satisfied: (i) on 99% of the training examples, the activation of every output unit is within 0.25 of correctness; (ii) every training example is presented to the network 100 times, i.e., the network has been trained for 100 epochs; (iii) the network classifies at least 90% of the training examples correctly but has not improved its classification ability for 5 epochs. We have defined an epoch as one training pass through the whole training set. We used the standard backpropagation learning algorithm to train C-IL^{2}P networks.

C-IL^{2}P generalizes better than any empirical learning system (see Figs. 8 and 9) and better than any system that learns from examples and background knowledge (see Figs. 10 and 11) tested on both applications. In most cases differences are statistically significant. However, C-IL^{2}P is only marginally better than KBANN. This is because both systems are hybrid neural systems that perform inductive learning from examples and background knowledge.

Usually, theory and data learning systems require fewer training examples than systems that learn only

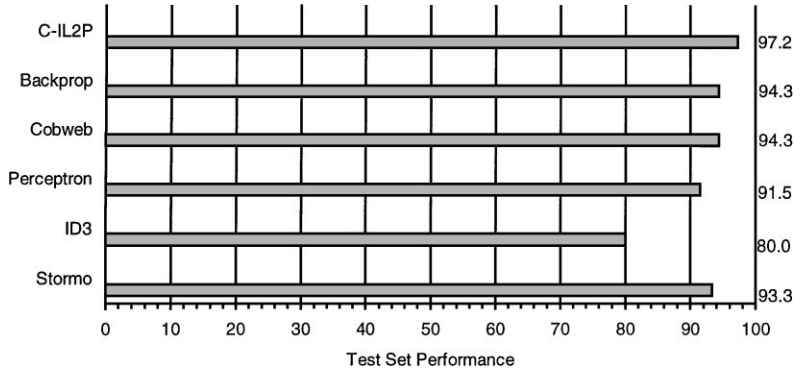

Figure 8. Test-set performance in the promoter problem (comparison with systems that learn strictly from examples).

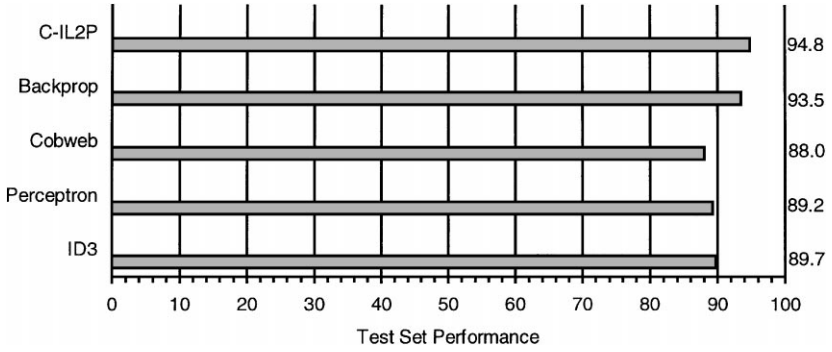

Figure 9. Test-set performance in the splice junction problem (comparison with systems that learn strictly from examples).

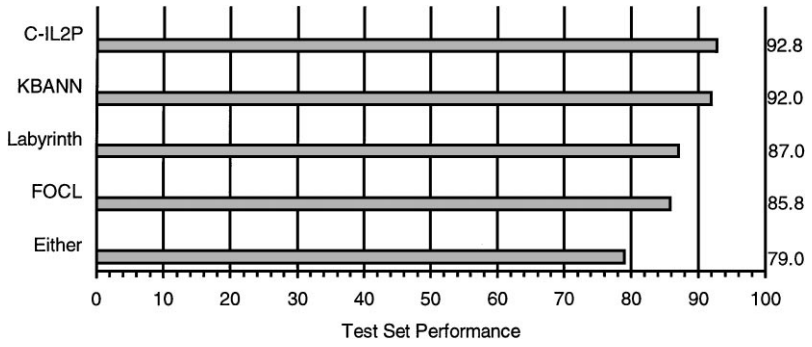

Figure 10. Test-set performance in the promoter problem (comparison with systems that learn both from examples and theory).

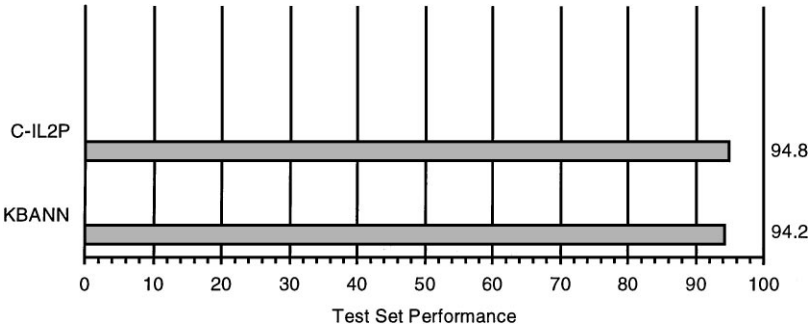

Figure 11. Test-set performance in the splice junction problem (comparison with systems that learn both from examples and theory).

from data. The background knowledge helps a learning system to extract useful generalizations from small sets of examples. This is quite important since, in general, it is not easy to obtain large and accurate training sets.

Thus, let us now analyze C-IL^{2}P's test-set performance given smaller sets of examples. The following tests will compare the performance of C-IL^{2}P with KBANN and *Backpropagation* only, because these systems have shown to be the most effective ones in the previous tests (Figs. 8–11). Following [16], the generalization ability over small sets of examples is analyzed by splitting the examples into two subsets: the testing set containing approximately 25% of the examples, and the training set containing the remaining examples. The training set is partitioned into sets of increasing sizes and the networks are trained using each partition at a time.

Figures 12 and 13 show that in both applications C-IL^{2}P generalizes over small sets of examples better than *backpropagation*. The results empirically show

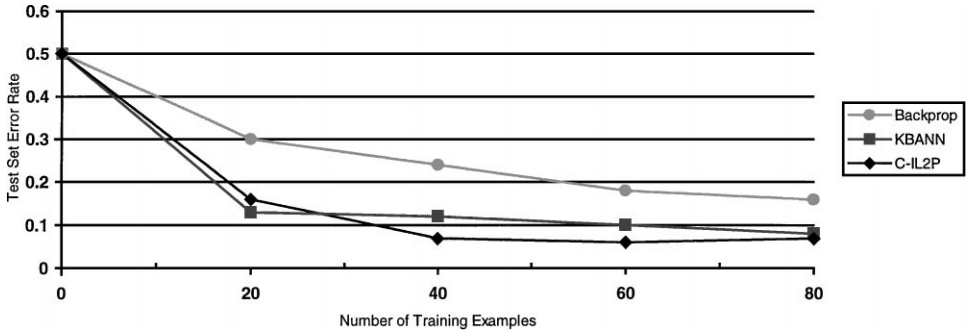

Figure 12. Test-set error rate in the promoter problem (26 examples reserved for testing).

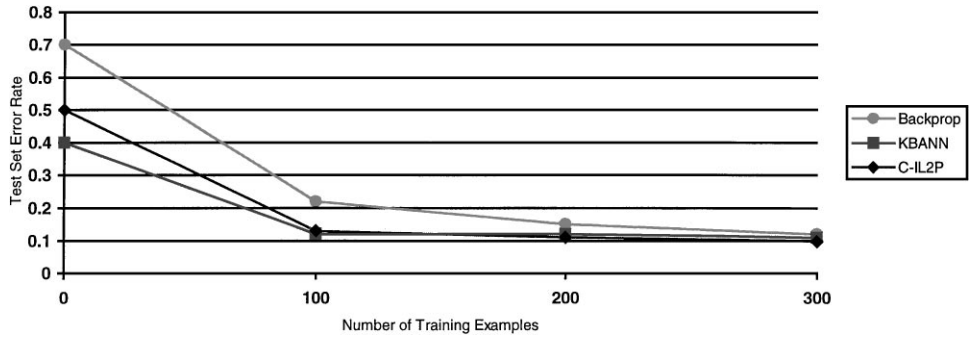

Figure 13. Test-set error rate in the splice junction problem (798 examples reserved for testing).

that the initial topology of the network, set by the background knowledge, gives it a better generalization capability. Note that the results obtained by C-IL^{2}P and KBANN are very similar, since both systems are based on the backpropagation learning algorithm and learn from examples and background knowledge.

Concluding the tests, we check the training-set performance of C-IL^{2}P in comparison again with KBANN and *backpropagation*. Figures 14 and 15 describe the training-set rms error rate decay obtained by each system during learning respectively in each application. The rms parameter indicates how fast a neural network learns a set of examples w.r.t training epochs. Neural networks' learning performance is a major concern, since it can become prohibitive in certain applications, usually as a result of the local minima problem.[15]

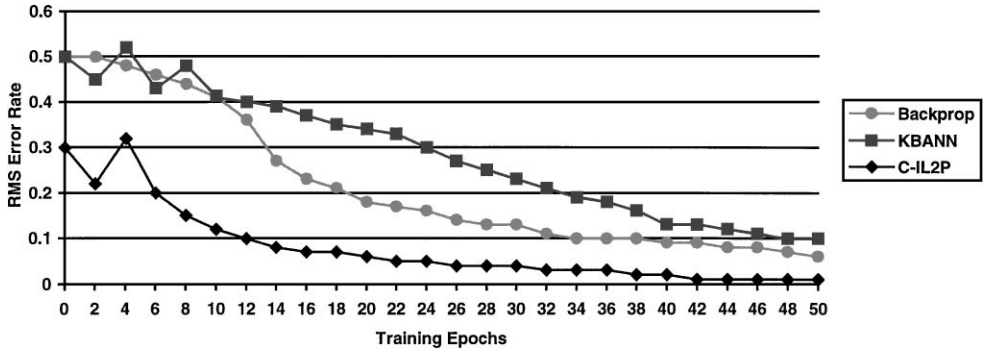

Figure 14. Training-set rms error decay during learning the promoter problem.

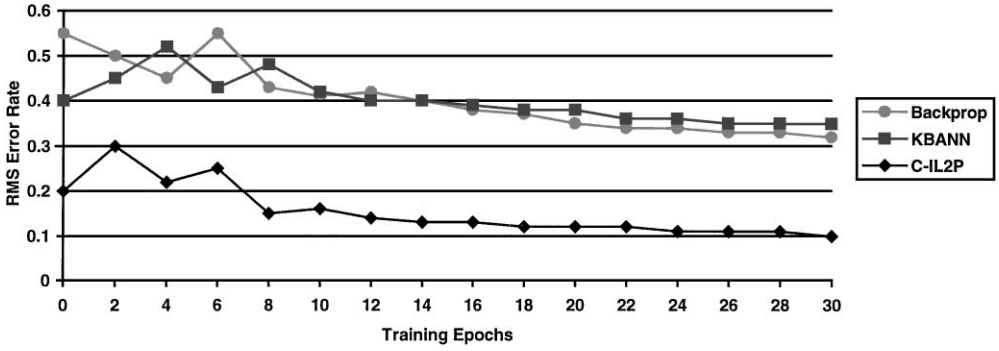

Figure 15. Training-set rms error decay during learning the splice junction problem.

Figures 14 and 15 show that C-IL^{2}P's learning performance is considerably better than KBANN's. The results suggest that our translation algorithm from symbolic knowledge to neural networks has advantages over the algorithm presented in [16]. The *Translation Algorithm* presented here always encodes the background knowledge into a single hidden-layer neural network. However, KBANN's translation algorithm generates a network with as many layers as there are dependency chains in the background knowledge. For example, if $B \leftarrow A, C \leftarrow B, D \leftarrow C$, and $E \leftarrow D$, KBANN generates a network with three hidden-layers in which concepts B, C, and D are represented. Obviously, this creates a respective degradation in learning performance. Towell and Shavlik have tried to overcome this problem with a symbolic pre-processor of rules for KBANN [44]. However, it introduces another preliminary phase to their translation process.[16] In our opinion the problem lies in KBANN's translation algorithm, and can be straightforwardly solved by an accurate translation mechanism.

Summarizing, the experiments described above suggest that C-IL^{2}P's effectiveness is a result of three of the system's features: C-IL^{2}P is based on *backpropagation*, it uses *background knowledge*, and it provides an *accurate and compact translation* from symbolic knowledge to neural networks.

5. Future Work and Conclusion

There are some interesting open questions relating to the *explanation capability* of neural networks, specifically to the trade-off between the *complexity* and *quality* of rules' extraction methods. One way to reduce this trade-off would be to investigate more efficient pruning methods for neural networks' input vectors search spaces [19].

Another interesting question relates to the class of extended programs [45], that is of interest in connection with the relation between Logic Programming and nonmonotonic formalisms. *Extended logic programs*, which add "classical" negation to the language of general programs, can be viewed as a fragment of Default theories [46]. Commonsense knowledge can be represented more easily when "classical" negation is available. We have extended C-IL^{2}P to deal with extended logic programs. The extended C-IL^{2}P system computes *answer sets* [45] instead of stable models. As a result it can be applied in a broader range of domains theories. The extended C-IL^{2}P has already been applied in power systems' fault diagnosis, obtaining promising preliminary results [47].

By changing the definition of T_P, variants of Default Logic and Logic Programming semantics can be obtained [48], defining a family of *nonmonotonic* (*paraconsistent*) *neural reasoning systems*. Another interesting direction to pursue would be the use of *labelled clauses* in the style of [49], whereby the proof of a literal is recorded in the label. The learning and generalization capabilities of the network must also be formally studied, so paying due regard to the logical foundations of the system. The system's extension to deal with *first order logic* is another complex and vast area for further research [50, 60].

As a massively parallel nonmonotonic learning system, C-IL^{2}P has interesting implications for the problem of *Belief Revision* [51] (see also [52]). In the splice-junction determination problem, part of the background knowledge was effectively encoded as defeasible, so that contradictory examples were able to specify a revision of the knowledge. Specifically, the knowledge regarding the Conformation of the genes has been changed. Hence, the background knowledge together with the set of examples can be inconsistent, and one needs to investigate ways to detect and treat *inconsistencies* in the system, viewing the learning process as a process of revision.

In this paper, we have presented the Connectionist Inductive Learning and Logic Programming System (C-IL^{2}P); a massively parallel computational model based on artificial neural networks that integrates inductive learning from examples and background knowledge, with deductive learning from Logic Programming. We have obtained successful experimental results when applying C-IL^{2}P to two real-world problems in the domain of molecular biology. Both kinds of Intelligent Computational Systems, Symbolic and Connectionist, have virtues and deficiencies. Research into the integration of the two has important implications [53], in that one is able to benefit from the advantages that each confers. We believe that our approach contributes to this area of research.

Acknowledgments

We are grateful to Dov Gabbay, Valmir Barbosa, Luis Alfredo Carvalho, Alberto Souza, Luis Lamb, Nelson Hallack, Romulo Menezes and Rodrigo Basilio for useful discussions. We would especially like to thank Krysia Broda and Sanjay Modgil for their

comments. This research was partially supported by CNPq and CAPES/Brazil. This work is part of the project ICOM/ProTem.

Notes

1. An acceptable program $\mathcal{P}$ has exactly one stable model.
2. An *interpretation* is a function from propositional variables to {"*true*", "*false*"}. A *model* for $\mathcal{P}$ is an interpretation that maps $\mathcal{P}$ to "*true*".
3. The "*sufficiently small*" restrictions are given by equations $\mu A_{\max} \leq \frac{1}{2}$ and $kA_{\max} \leq \frac{1}{2}$, respectively, where $A_{\max} > 0$ [21].
4. Note that a sound translation from $\mathcal{P}$ to $\mathcal{N}$ does not require all the weights in $\mathcal{N}$ to have the same absolute value. We unify the weights ($|W|$) for the sake of simplicity of the translation algorithm and to comply with previous work.
5. It is known that an increase in the number of hidden layers in a neural network results in a corresponding degradation in learning performance.
6. The recurrent connections represent an external process between output and input.
7. The perturbation should be small enough not to have any effects on the computation of the background knowledge.
8. These are the same problems that were investigated in [16] for the evaluation of KBANN. We have followed as much as possible the methodology used by Towell and Shavlik, and we have used the same background knowledge and set of examples as KBANN.
9. Prokaryotes are single-celled organisms that do not have a nucleus, e.g., E. Coli.
10. Rules obtained from [16], and derived from the biological literature [54] from Noordewier [55].
11. Unlike prokaryotic cells, eukaryotic cells contain a nucleus, and so are higher up the evolutionary scale.
12. Rules obtained from [16] and derived from the biological literature from Noordewier [56].
13. Towell and Shavlik compare KBANN with other hybrid systems [57] and [58], obtaining better results.
14. In accordance with [16], 1000 examples are randomly selected from the 3190 examples set for each training phase.
15. The network can get stuck in local minima during learning, instead of finding the global minimum of its error function.
16. KBANN already contains a preliminary phase of rules hierarchying, that rewrites the rules before translating them.

References

1. N. Lavrac and S. Dzeroski, "Inductive logic programming: Techniques and applications," *Ellis Horwood Series in Artificial Intelligence*, 1994.
2. T.M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
3. S.B. Thrun et al., "The MONK's problems: A performance comparison of different learning algorithms," Technical Report, Carnegie Mellon University, 1991.
4. R.S. Michalski, "Learning strategies and automated knowledge acquisition," *Computational Models of Learning, Symbolic Computation*, Springer-Verlag, 1987.
5. N.K. Bose and P. Liang, *Neural Networks Fundamentals with Graphs, Algorithms, and Applications*, McGraw-Hill, 1996.
6. F.J. Kurfess, "Neural networks and structured knowledge," in *Knowledge Representation in Neural Networks*, edited by Ch. Herrmann, F. Reine, and A. Strohmaier, Logos-Verlag: Berlin, pp. 5–22, 1997.
7. G. Pinkas, "Energy minimization and the satisfiability of propositional calculus," *Neural Computation*, vol. 3, no. 2, 1991.
8. G. Pinkas, "Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge," *Artificial Intelligence*, vol. 77, pp. 203–247, 1995.
9. S. Holldobler, "Automated inferencing and connectionist models," Post Ph. D. Thesis, Intellektik, Informatik, TH Darmstadt, 1993.
10. H.B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, 1972.
11. S. Holldobler and Y. Kalinke, "Toward a new massively parallel computational model for logic programming," in *Proc. Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*, 1994.
12. J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1987.
13. K.R. Apt and D. Pedreschi, "Reasoning about termination of pure prolog programs," *Information and Computation*, vol. 106, pp. 109–157, 1993.
14. M. Fitting, "Metric methods—three examples and a theorem," *Journal of Logic Programming*, vol. 21, pp. 113–127, 1994.
15. M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proc. Fifth International Symposium on Logic Programming*, MIT Press: Cambridge, pp. 1070–1080, 1988.
16. G.G. Towell and J.W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1, pp. 119–165, 1994.
17. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, edited by D.E. Rumelhart and J.L. McClelland, MIT Press, vol. 1, pp. 318–363,1986.
18. S. Muggleton and L. Raedt, "Inductive logic programming: Theory and methods," *Journal of Logic Programming*, vol. 19, pp. 629–679, 1994.
19. A.S. d'Avila Garcez, K. Broda, and D. Gabbay, "Symbolic knowledge extraction from trained neural networks: A new approach," Technical Report TR-98-014, Department of Computing, Imperial College, London, 1998.
20. L.M. Fu, *Neural Networks in Computer Intelligence*, McGraw Hill, 1994.
21. G.G. Towell, "Symbolic knowledge and neural networks: Insertion, refinement and extraction," Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, 1991.
22. J. Hertz, A. Krogh, and R.G. Palmer, "Introduction to the theory of neural computation," *Studies in the Science of Complexity*, Santa Fe Institute, Addison-Wesley Publishing Company, 1991.
23. K.R. Apt and N. Bol, "Logic programming and negation: A survey," *Journal of Logic Programming*, vol. 19, pp. 9–71, 1994.
24. R.M. Karp and V. Ramachandran, "Parallel algorithms for shared-memory machines," in *Handbook of Theoretical Computer Science*, edited by J. van Leeuwen, Elsevier Science, vol. 17, pp. 869–941, 1990.

25. K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
26. B. DasGupta and G. Schinitger, "Analog versus discrete neural networks," *Neural Computation*, vol. 8, pp. 805–818, 1996.
27. M.I. Jordan, "Attractor dynamics and parallelisms in a connectionist sequential machine," in *Proc. Eighth Annual Conference of the Cognitive Science Society*, pp. 531–546, 1986.
28. R. Andrews and S. Geva, "Inserting and extracting knowledge from constrained error backpropagation networks," in *Proc. Sixth Australian Conference on Neural Networks*, Sydney, 1995.
29. E. Pop, R. Hayward, and J. Diederich, *RULENEG: Extracting Rules from a Trained ANN by Stepwise Negation*, QUT NRC, 1994.
30. S.B. Thrun, "Extracting provably correct rules from artificial neural networks," Technical Report, Institut fur Informatik, Universitat Bonn, 1994.
31. M.W. Craven and J.W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *Proc. Eleventh International Conference on Machine Learning*, pp. 37–45, 1994.
32. G.G. Towell and J.W. Shavlik, "The extraction of refined rules from knowledge based neural networks," *Machine Learning*, vol. 13, no. 1, pp. 71–101, 1993.
33. R. Setiono, "Extracting rules from neural networks by pruning and hidden-unit splitting," *Neural Computation*, vol. 9, pp. 205–225, 1997.
34. R. Andrews, J. Diederich, and A.B. Tickle, "A survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-based Systems*, vol. 8, no. 6, pp. 1–37, 1995.
35. W. Marek and M. Truszczynski, *Nonmonotonic Logic: Context Dependent Reasoning*, Springer-Verlag, 1993.
36. J.D. Watson, N.H. Hopkins, J.W. Roberts, J.A. Steitz, and A.M. Weiner, *Molecular Biology of the Gene*, Benjamin Cummings: Menlo Park, vol. 1, 1987.
37. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books: New York, 1962.
38. J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.
39. D.H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, pp. 139–172, 1987.
40. G.D. Stormo, "Consensus patterns in DNA," *Methods in Enzymology*, Academic Press: Orlando, vol. 183, pp. 211–221, 1990.
41. D. Ourston and R.J. Mooney, "Theory refinement combining analytical and empirical methods," *Artificial Intelligence*, vol. 66, pp. 273–310, 1994.
42. K. Thompson, P. Langley, and W. Iba, "Using background knowledge in concept formation," in *Proc. Eighth International Machine Learning Workshop*, Evanston, pp. 554–558, 1991.
43. M. Pazzani and D. Kibler, "The utility of knowledge in inductive learning," *Machine Learning*, vol. 9, pp. 57–94, 1992.
44. G.G. Towell and J.W. Shavlik, "Using symbolic learning to improve knowledge-based neural networks," in *Proc. AAAI'94*, 1994.
45. M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, Springer-Verlag, vol. 9, pp. 365–385, 1991.
46. R. Reiter, "A logic for default reasoning," *Artificial Intelligence*, vol. 13, pp. 81–132, 1980.
47. A.S. d'Avila Garcez, G. Zaverucha, and V. da Silva, "Applying the connectionist inductive learning and logic programming system to power system diagnosis," in *Proc. IEEE International Joint Conference on Neural Networks IJCNN'97*, vol. 1, Houston, USA, pp. 121–126, 1997.
48. G. Zaverucha, "A prioritized contextual default logic: Curing anomalous extensions with a simple abnormality default theory," in *Proc. KI'94*, Springer-Verlag: Saarbrucken, Germany, LNAI 861, pp. 260–271, 1994.
49. D.M. Gabbay, *LDS—Labelled Deductive Systems—Volume 1 Foundations*, Oxford University Press, 1996.
50. N. Hallack, G. Zaverucha, and V. Barbosa, "Towards a hybrid model of first-order theory refinement," in *Neural Information Processing Systems, Workshop on Hybrid Neural Symbolic Integration*, Breckenridge, Colorado, USA, 1998.
51. P. Gardenfors (Ed.), "Belief revision," *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, 1992.
52. P. Gardenfors and H. Rott, "Belief revision," in *Handbook of Logic in Artificial Intelligence and Logic Programming*, edited by D. Gabbay, C. Hogger, and J. Robinson, Oxford University Press, vol. 4, pp. 35–132, 1994.
53. M. Hilario, "An overview of strategies for neurosymbolic integration," in *Proc. Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches, IJCAI 95*, 1995.
54. M.C. O'Neill, "Escherichia coli promoters: Consensus as it relates to spacing class, specificity, repeat substructure, and three dimensional organization," *Journal of Biological Chemistry*, vol. 264, pp. 5522–5530, 1989.
55. G.G. Towell, J.W. Shavlik, and M.O. Noordewier, "Refinement of approximately correct domain theories by knowledge-based neural networks," in *Proc. AAAI'90*, Boston, pp. 861–866, 1990.
56. M.O. Noordewier, G.G. Towell, and J.W. Shavlik, "Training knowledge-based neural networks to recognize genes in DNA sequences," *Advances in Neural Information Processing Systems*, Denver, vol. 3, pp. 530–536, 1991.
57. L.M. Fu, "Integration of neural heuristics into knowledge-based inference," *Connection Science*, vol. 1, pp. 325–340, 1989.
58. B.F. Katz, "EBL and SBL: A neural network synthesis," in *Proc. Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, pp. 683–689, 1989.
59. M. Minsky, "Logical versus analogical, symbolic versus connectionist, neat versus scruffy," *AI Magazine*, vol. 12, no. 2, 1991.
60. R. Basilio, G. Zaverucha, and A.S. J'Avila Garcez, "Inducing Relational Concepts with Neural Networks via the Linus System," in *Proc. International Conference on Neural Information Processing*, vol. 3, pp. 1507–1510, Japan, 1998.

Artur Garcez is a Ph.D. student in the Department of Computing at Imperial College, London, UK. He received his B.Eng. in Computing Engineering from the Pontifical Catholic University of Rio de Janeiro (PUC-RJ), Brazil, in 1993 and his M.Sc. in Computing and Systems' Engineering from the Federal University of Rio de Janeiro (COPPE/UFRJ), Brazil, in 1996. He also worked in the Latin-American Technology Institute of IBM-Brazil. His research interests include Symbolic-Connectionist Integration, Machine Learning, Neural Networks, Belief Revision and Commonsense (Practical) Reasoning.

Gerson Zaverucha is an Associate Professor of Computer Science at the Federal University of Rio de Janeiro (COPPE/UFRJ), Brazil. He received his B.Eng. in Electrical Engineering from the Federal University of Bahia, Brazil, in 1981, his M.Eng. in Electric Power Engineering from the Rensselaer Polytechnic Institute of Troy, New York, USA, in 1982 and his Ph.D. in Computing from Imperial College, London, UK, in 1990. His research interests include Connectionist First Order Theory Refinement, Machine Learning, Hybrid Systems, Inductive Logic Programming and Nonmonotonic Reasoning.