Learning and Reasoning about Norms using Neural-Symbolic Systems

Guido Boella[1], Silvano Colombo Tosatto[1,2], Artur D'Avila Garcez[3],
Valerio Genovese[1,2], Alan Perotti[1], and Leendert van der Torre[2]

[1]University of Turin, Italy. {*guido, genovese, perotti*}*@di.unito.it*
[2]CSC, University of Luxembourg. {*silvano.colombotosatto, leon.vandertorre*}*@uni.lu*
[3]City University London. *aag@soi.city.ac.uk*

ABSTRACT

In this paper we provide a neural-symbolic framework to model, reason about and learn norms in multi-agent systems. To this purpose, we define a fragment of Input/Output (I/O) logic that can be embedded into a neural network. We extend d'Avila Garcez et al. Connectionist Inductive Learning and Logic Programming System (CILP) to translate an I/O logic theory into a Neural Network (NN) that can be trained further with examples: we call this new system Normative-CILP (N-CILP). We then present a new algorithm to handle priorities between rules in order to cope with normative issues like Contrary to Duty (CTD), Priorities, Exceptions and Permissions. We illustrate the applicability of the framework on a case study based on RoboCup rules: within this working example, we compare the learning capacity of a network built with N-CILP with a non symbolic neural network, we explore how the initial knowledge impacts on the overall performance, and we test the NN capacity of learning norms, generalizing new Contrary to Duty rules from examples.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Experimentation, Theory, Legal Aspects

Keywords

Knowledge representation, Single agent reasoning, Computational architectures for learning, Single agent learning

1. INTRODUCTION

In artificial social systems, norms are mechanisms to effectively deal with coordination in normative multi-agent systems (MAS). An open problem in AI is how to equip agents to deal effectively with norms that change over time [3], either due to explicit changes made by legislators or due to different interpretations of the law by judges and referees.

In this paper we combine Input/Output (I/O) logic [11] with the neural-symbolic paradigm [7] in order to address the following research question:

- *How to define a formal framework for reasoning and learning about norms in a dynamic environment?*

Input/Output (I/O) logic [11] is a symbolic formalism used to represent and reason about norms. I/O logic provides some reasoning mechanisms to produce outputs from the inputs, and each of them bears a specific set of features.

The neural-symbolic paradigm of [7] embeds symbolic logic into neural networks. Neural-symbolic systems provide translation algorithms from symbolic logic to neural networks and vice-versa. The resulting network is used for robust learning and computation, while the logic provides (i) background knowledge to help learning (as the logic is translated into the NN) and (ii) high-level explanations for the network models (when the trained NN is translated into the logic).CILP is an advanced neural-symbolic systems and it has been shown an effective tool in exploiting symbolic background knowledge (i.e. on incomplete domain theory) with learning from examples.

We study how to represent I/O within the computational model of neural networks (NNs). We choose I/O logic because it presents a strong similarity with NNs: both have a separate specification of inputs and outputs. We exploit this analogy to encode symbolic knowledge (expressed in terms of I/O rules) into NNs, and then we use the NN to reason and learn new norms in a dynamic environment.
Hence two Research sub-Questions are:

- *How to represent I/O logic rules in neural networks?*
- *How to refine normative rules and learn new ones?*

Below, we define the language used to express norms and we present an extension of the "Connectionist Inductive Learning and Logic Programming" system (CILP) [7], called Normative-CILP (N-CILP).

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

With the exception of game-theoretic approaches [17, 5, 18], few machine learning techniques have been applied to tackle open problems like learning and/or revising new norms in open and dynamic environments.

We show how to use NNs to cope with some of the underpinnings of normative reasoning: *permissions*, *contrary to duties* (CTD) and *exceptions* by using the concept of priorities between the rules.

We also tested our tool on a case study based on the RoboCup competition, representing a significant set of the *rules of the game* from [13] in I/O logic and then studying the capability of the tool in learning new norms and performing reasoning. The results show that the I/O encoding improves the capacity of the NN of learning norms.

The contribution of this work is in studying and combining symbolic and sub-symbolic representations to provide a flexible and effective methodology for learning, normative reasoning and specification in MAS. In this process, we have also made a contribution to the area of neural-symbolic integration: by studying neural-symbolic systems from the point of view of normative reasoning we have been able to propose a new translation of priorities into object-level negation. From a theoretical perspective, we are interested in studying the similarities between I/O logic and neural networks. From a practical point of view, it is hoped that the network model will lead directly to an efficient hardware implementation. The normative CILP tool has been implemented in Java and is available for download (together with the dataset) at *http://www.di.unito.it/∼genovese/tools/NNSS.zip*. The experiments reported here indicate how promising is this line of research.

The paper is structured as follows: In section 2 we describe the relevant background about the neural-symbolic approach, I/O logic and normative agents. In Section 3 we introduce our approach and a motivating example. In Section 4 we show how to encode I/O logic into a neural-network using the Normative-CILP translation algorithm. In Section 5 we present and discuss the results obtained from the experiments. Section 6 concludes the paper and discusses directions for future work.

2. RELATED WORK

2.1 Neural-Symbolic approach

The main purpose of a *neural-symbolic approach* is to bring together connectionist and symbolic approaches [7]. In this way it is possible to exploit the strengths of both approaches and hopefully avoid their drawbacks. With such approach we are able to formally represent the norms governing the normative system in a neural network. In addition we are also capable of exploiting the instance learning capacities of neural networks and their massive parallel computation.

Algorithms like *KBANN*[19] and *CILP*[8] provide a translation of a symbolic representation of knowledge into a neural network. The advantage of CILP is that it uses a provably sound translation into single-hidden layer networks with sigmoid activation functions. This allows the efficient use of *backpropagation* for learning. In what follows, we use a variant of CILP since we are interested in the integration of reasoning and learning capabilities.

2.2 I/O Logic

To describe the norms regulating the system we use I/O Logic [11]. Rules used in I/O logic are defined as couples $R_1 = (A, B)$, where both A and B represent sets of literals that can be in disjunctive or conjunctive form. A is called the *antecedent* of the rule, while B is the *consequent*: A must hold for the rule to be activated, and B is consequently activated. I/O logic provides some reasoning mechanisms to produce outputs from the inputs, and each of them bears a specific set of features. The *simple-minded output* does not satisfy the principle of identity, but it allows the *strengthening input*, *conjoining output* and *weakening output* features. The *basic output* and *reusable output* mechanisms allow the additional features of *input disjunction* and *reusability*, while the *reusable basic output* approach satisfies both of the above. A detailed description of the I/O logic mechanisms and features can be found in [11], [12].

Boella et al. [1] described how a connectionist approach like neural networks can embed the different features of I/O logic: within this perspective, it is possible to use translation algorithms (like KBANN or CILP) to reproduce the mechanisms of I/O logic. In many examples of this paper, since we are dealing with normative reasoning, the consequents of the rules will be expressed using the O operator: for instance, $(getFine, O(payFine))$ represent the norm *If you are given a fine, you ought to pay it.*

2.3 Normative agent

In this paper we focus on modeling and reasoning about what a normative agent [2] is obliged or allowed to do in given states of the surrounding environment. Normative reasoning requires agents to deal with specific problems such as *dilemmas*, *exceptions* and *contrary to duties*.

Dilemmas: two obligations are said to be *contradictory* when they can not be accomplished together. A possible example of contradictory normas is the *dilemma*. This usually happens when an agent is subject to different normative codes (i.e. when an agent has to follow the *moral* and the *legal* code). Anyway it is outside the scope of this paper to discuss about how to overcome dilemmas, as we are focusing on how to use *priorities* to regulate *exceptions* and *contrary to duties*.

Priorities are used to give a partial ordering between norms. This is useful when, given two applicable norms, we always want one to preempt the other, for instance when dealing with *exceptions*.

We encode priorities among the norms by using *negation as failure* ($\sim$). Given two norms $R_1 = (A_1 \wedge A_3, \mathbf{O}(\beta_1))$ and $R_2 = (A_2 \wedge A_3, \mathbf{O}(\beta_2))$ and a priority relation $R_1 \succ R_2$ between them (such that the first norm has priority), we encode the priority relation by modifying the antecedent of the norm with lower priority. Specifically, we include in the antecedent of the norm with the lower priority the negation as failure of the literals in the antecedent of the higher prioritized norm that does not appear in the antecedent of the lower priority norm. We do so in order to ensure that, in a situation where both (unmodified) norms would be applicable, the newly inserted negation-as-failure atoms in the antecedent of the modified lower-prioritize rule evaluate to false and make the whole rule not applicable. Considering for example the two rules given above, we have to modify R_2. The only atom appearing in R_1's antecedent and not in R_2's antecedent is A_1, and therefore we introduce $\sim A_1$ as a conjunct in R_2's antecedent. After embedding the priority, the second rule becomes $R_2' = (A_2 \wedge \sim A_1 \wedge A_3, \mathbf{O}(\beta_2))$. Note that in a potentially conflicting situation when A_1, A_2 and A_3 hold, R_1 and R_2 are applicable, but R_2' is not, thus avoiding the conflict.

Exceptions occur when, due to particular circumstances, a norm should be followed instead of another one. Suppose that a norm $R_1 = (\alpha, \mathbf{O}(\beta))$ should be applied in all the situations containing α. For exceptional situations we consider an additional norm $R_2 = (\alpha \wedge \gamma, \mathbf{O}(\neg\beta))$. The latter norm should be applied in a subset of situations w.r.t. R_1: specifically all those when, in addition to α, also γ holds. We can call situations where both α and γ hold exceptional situations. In these exceptional situations both norms could be applied. This would produce two contrasting obligations: $\mathbf{O}(\beta)$ and $\mathbf{O}(\neg\beta)$. To avoid this we add the following priority relation: $R_2 \succ R_1$. Therefore we modify the antecedent of the norm with lower priority as described earlier. The result is a new norm $R_1' = (\alpha \wedge \sim \gamma, \mathbf{O}(\beta))$, that would not be applied in the exceptional situations, avoiding the problem of contrasting obligations.

Contrary to Duties: an important property of norms is that they are soft constraints. Accordingly to this feature they can be violated. Contrary to duties provide additional obligations to be fulfilled when a violation occurs.

For example, consider a norm $R_1 = (\alpha, \mathbf{O}(\beta))$ that should be applied in all situations containing α and producing the obligation $\mathbf{O}(\beta)$. As mentioned, norms can be violated, therefore we can also define a norm that produces alternative obligations to be followed in case of a violation. Let this new norm be $R_2 = (\alpha \wedge \neg\beta, \mathbf{O}(\gamma))$. The latter norm contains in its antecedent both the antecedent of R_1 and the negation of its consequent. In this way it describes which should be the alternative obligation to $\mathbf{O}(\beta)$ in the case that it can not be achieved, in this example $\mathbf{O}(\gamma)$.

We use a priority relation between the two norms in order to avoid the generation of the obligation $\mathbf{O}(\beta)$ in case it is already known that it is not satisfiable. We add then the following priority relation $R_2 \succ R_1$ that modifies the first norm as follows: $R_1' = (\alpha \wedge \sim \neg\beta, \mathbf{O}(\beta))$.

Permissions: an important distinction between *oughts* and *permissions* is that the latter will not be explicitly encoded in the neural network. In our approach we consider that something is permitted to the agent if not explicitly forbidden (note that we consider the ought of a negative literal as a prohibition). Due to this we consider that rules with a permission in their consequent implicitly have priority over the rules that forbid the same action. For example, consider two rules $R_1 = (A_1, \mathbf{P}(\beta_1))$, $R_2 = (A_2, \mathbf{O}(\neg\beta_1))$. The first rule permits β_1 and the second forbids it. In this case we assume the following priority relation $R_1 \succ R_2$ holds.

3. ARCHITECTURE AND CASE STUDY

Our goal is to allow the agent to learn from experience and take decisions which respect the norms she is subject to. Thus, the agent needs to know what is obligatory and forbidden according to norms (conditional rules) in any situation in real time. What is obligatory can eventually become an action of the agent, while what is forbidden inhibits such actions, like in agent architectures [6].

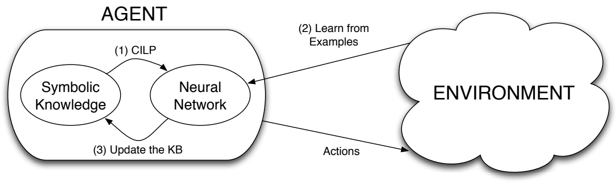

Figure 1: Normative agent architecture.

Rules may change: the normative environment changes over time so the agent should be flexible enough to adapt its behavior to the context using as information the instances of behaviors which have been considered illegal.

Figure 1 describes our approach. It starts from the symbolic knowledge-base (KB) of norms contained in the agent, transforming it into a neural network (NN) using an extension of the CILP algorithm (introduced below). The NN is structured as follows: input neurons of the network represent the state of the world (e.g., in the robocup domain, kickoff, have ball, etc.), while the output neurons represent the obligations of the agent, e.g., pass the ball (i.e. cooperate), minimize impact, etc., or the prohibitions, e.g., do not pass, do not score own goal, etc. The NN is used as part of the controller for the agent and, given its ability to learn, it is hoped to give the agent the required flexibility.

We then train the NN on instances of robocup match behaviors to adapt the agent to the current context. E.g., given a set of situations where the referee punishes an agent for kicking the ball backwards, we specify them as learning instances where there is the prohibition to kick the ball backwards. The NN can generalize the conditions under which this prohibition holds. To learn from behaviors which are regulated by norms, the NN must be able to cope with the peculiarities of normative reasoning.

In our tests we used a version of the RoboCup rules from the 2007 competition where, for simplicity, teams are composed of two players. To make things more interesting, in addition to those rules, we have added to the KB some norms representing the *coach*'s directions that regulate the behavior of the robots during the match.

Each rule is of the form IF α THEN β. The precondition α is a set of literals in conjunctive form while the postcondition β can be either an obligation or a permission concerning a single literal. Rules like IF $\top$ THEN O($\neg$*impact_opponent*) and IF *have_ball* $\wedge$ *opponent_approaching* THEN O(*pass*) contain obligations in their postconditions. Differently, a rule like IF *goalkeeper*$\wedge$*inside_own_area* THEN P(*use_hands*) contains a permission.

It is possible, however, that the environment requires the agent to adopt some sub-optimal behavior in circumstances when the optimal solution is not available. We use priorities to manage general and specific rules, creating a general-to-specific superiority relation and dealing with sub-optimal and exceptional situations. The two rules that compose an instance of contrary to duty are in the following configuration: the first one IF α THEN O(β) and IF $\neg\beta$ THEN O(γ); β represents the obligation to be fulfilled in an or-

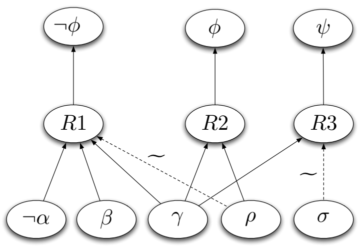

Figure 2: Example of I/O logic embedding in a NN

dinary situation α. If the agent is in a state of the world where β cannot be fulfilled, the second rule overcomes the first one through the use of priorities. For instance, IF $\top$ THEN O($\neg$*impact_opponent*) $\prec$ IF *impact_opponent* THEN O(*minimize_impact*). Intuitively, we use ($\prec$) such that (y) $\prec$ (x) means that, whenever the conclusion of rule (x) holds, the conclusion of (y) does not hold.

Figure 2 shows a neural network built from four rules: $R_1 = (\neg\alpha \wedge \beta \wedge \gamma, \mathbf{O}(\neg\phi))$, $R_2 = (\gamma \wedge \rho, \mathbf{O}(\phi))$, $R_3 = (\gamma, \mathbf{O}(\neg\psi))$ and the *permission rule* $R_4 = (\gamma \wedge \sigma, \mathbf{P}(\psi))$. In addition, a priority ordering $R_2 \succ R_1$ is expected to inhibit the activation of the first rule whenever the second rule applies. This priority is embedded within the rules as described earlier and, as a result, we obtain a new first rule: $R'_1 = (\neg\alpha \wedge \beta \wedge \gamma \wedge \sim \rho, \mathbf{O}(\neg\phi))$. Further, the implicit priority of R_4 over R_3 embeds in R_3 a negative literal obtaining a new rule, as follows: $R'_3 = (\gamma \wedge \sim \sigma, \mathbf{O}(\neg\psi))$. The neural network is built, then, from rules R'_1, R_2 and R'_3 (permission rules are not encoded in the network and are only used to define the priorities). Dotted lines in the figure indicate links with negative weighted which, in turn, implement the negation in the rules R'_1 and R'_3. Notice how input and output neurons in the network have a natural correspondence with inputs and outputs in I/O logic. Each hidden neuron represents a rule, e.g. R_1, and the network, sometimes called an AND/OR network, is supposed to compute conjunctions in its hidden layer and disjunctions in its output layer. In what follows, we detail the algorithm that achieves this translation and its proof of soundness w.r.t. an answer set semantics. Notice that, although the network is associated with a logic programming semantics, it has very naturally an input and output layer that make it appropriate, rather like I/O logic, for normative reasoning. This will be exemplified later.

4. NEURAL NETWORKS FOR NORMS

In this section we introduce a new approach for coding (a fragment of) I/O logic into a neural network. The main intuition behind this methodology is that, although logic programs do not capture the concepts of *inputs* and *outputs*, an extended logic program-based neural network does, on a purely structural level: inputs and outputs in I/O logics correspond to the input and output layers of the neural network.

Neural-symbolic algorithms (like CILP) provide a sound and complete translation of logic programs (LP) into a neural network (NN). Unfortunately, LP is not directly suitable for reasoning about normative systems (in particular about CTD and dilemmas). This is due to the fact that LP does not have an explicit representation of inputs.

A fact a in an LP could be mapped, at first sight, as the input of the NN, so to make rules like $a \rightarrow b$ fire to produce output b. At the same time, a should be also among the output of the network, due to identity property of the underlying logic: a follows from a. But this would require to implement identity property in the NN, making it more complicated.

CILP does not need to represent a fact as an initial input, thanks to transitivity property of logic, which is expressed by the fact that the NN is recurrent: every output neuron is connected to the corresponding input neuron.

If the fact a was directly represented as an output, it would not need to be represented as an explicit input, since the transitivity property allows to propagate output to input.

To minimize the structure of the network, CILP translates a fact a (representing the input to other rules) directly as an output a of the neural network and, given a rule like $a \rightarrow b$, to derive b as output, the output a becomes the "input" of the NN due to the fact that the NN is recurrent: every output becomes an input subsequently, rather than at the initial iteration. So in a sense the NN resulting from CILP given an LP returns always the same output after the network stabilizes, since it has no explicit input.

In normative reasoning, as captured by IO logic, the input does not become necessarily an output, since identity does not hold. The reason is that the output is interpreted as what is obligatory, thus, if a is in the input, it is not necessarily the case that a is obligatory as well. Differently from LP, what is in the input must be distinguished from the output: a fact a cannot be modeled as an output which becomes an input due to transitivity. As an example, the logic programs $P_1 = \{\emptyset\}$ and $P_2 = \{a \rightarrow b\}$ both have the empty set as model, this is because LP semantics do not reflect the meaning of the program rule. However, if we translate P_1 and P_2 with CILP we get two different networks, one with an empty set of input and output nodes and the other with a as the input note and b as output. The need to explicitly reason about inputs and outputs of rules in normative systems has been put forward by Makinson and van der Torre [11] in their Input-Output (I/O) Logic framework. In I/O logic, norms are represented as ordered pairs of formulas like (α, β), read as: if α is present in the current situation then β should be the case. These two formulae are also named correspondingly the *input* and the *output*, to make it clear that the input of the norm is the current situation and what is desirable for this situation is the output. A peculiarity of I/O logic (shared with conditional logics) is that it does not have (α, α) for any α (i.e. *identity* is not an axiom), while in LP we always have $\alpha \leftarrow \alpha$. This input/output perspective corresponds straightforwardly to the intuition behind a NN. However, to take advantage of the existing CILP algorithm and its proof of soundness we translate (a simplified) IO logic into LP to be processed by CILP without mapping the input into atoms translated as output. Rather the input is subsequently passed as input of the network producing an output representing what is obligatory, where some input appears in the output only if it is made obligatory by some rule.

In CILP output nodes are always connected to input nodes creating a recurrent network, to represent the transitivity property. In normative reasoning transitivity is not always accepted (since if you are obliged to do a and if a then you are obliged to b, does not imply that you are obliged to do b), thus the normative CILP thus extends CILP to account for the fact that certain outputs should not be connected to their corresponding inputs.

4.1 Mapping I/O Logic into Neural Networks

In this section, we first introduce a fragment of I/O logic, then we present an embedding of such fragment into extended logic programs and finally, we discuss how to represent priorities between rules within extended logic programs.

Definition 1. *An* extended logic program *is a finite set of clauses of the form $L_0 \leftarrow L_1, \ldots, \sim L_n, \sim L_{n+1}, \ldots, \sim L_m$, where L_i ($0 \leq i \leq n$) is a literal i.e., an atom or a classical negation of an atom denoted by $\neg$ and $\sim L_J$ ($n{+}1 \leq j \leq m$) is called* default literal *where $\sim$ represents negation as failure.*

Given an extended logic program P we identify its *answer sets* [9] as $EXT(P)$.

Definition 2 (I/O Normative Code). *A normative code $\mathbf{G} = \langle \mathbb{O}, \mathbb{P}, \succ \rangle$ is composed by two sets of rules $r : (\alpha, \beta)$ and a preference relation $\succ$ among those rules. Rules in $\mathbb{O}$ are called* obligations*, while rules in $\mathbb{P}$ are* permissions*. Rules in $\mathbb{O}$ are of the type (α, β) where*

- *$\alpha = \alpha_1 \vee \ldots \vee \alpha_n$ is a propositional formula in disjunctive normal form i.e., α_i (for $0 \leq i \leq n$) is a conjunction of literals $(\neg a_{\alpha_{i1}} \wedge \ldots \wedge \neg a_{\alpha_{im}} \wedge a_{\alpha_{i(m+1)}} \wedge \ldots \wedge a_{\alpha_{1(m+p)}})$. Without loss of generality we assume that the first m literals are negative while the others $(m+p)-1$ are positive.*
- *$\beta = \neg b_{\beta_1} \wedge \ldots \wedge \neg b_{\beta_m} \wedge b_{\beta_{m+1}} \wedge \ldots \wedge b_{\beta_{m+p}}$ is a finite conjunction of literals.*

While rules in $\mathbb{P}$ are of type (α, l) where α is the same as for obligations but l is a literal.

As put forward in [4] the role of permissions is to undercut obligations. Informally, suppose to have a normative code $\mathbf{G}$ composed of two rules:

1. b is obligatory (i.e., $(\top, b) \in \mathbb{O}$).
2. If a holds, then $\neg b$ is permitted (i.e., $(a, \neg b) \in \mathbb{P}$).

We say that the rule $(a, \neg b)$ *has priority* over $(\top, b)$, i.e., b is obligatory as long as a does not hold, otherwise $\neg b$ is permitted and, therefore b is not obligatory anymore.

The semantics of such fragment of I/O is defined by the rules in Fig 3. $I(G)$ is the set of literals in the antecedent of rules in $\mathbf{G}$. The rules are a syntactical restriction of the those presented in [11].

The fact that we consider only I/O rules as defined in Definition 2 permits us to define a natural embedding of I/O rules and extended logic programs.

Definition 3. *We define a function $\ulcorner \cdot \urcorner$ which embeds I/O logic rules into extended logic programs*

$$\lceil r : (\alpha_1 \vee \ldots \vee \alpha_n, \beta_1 \wedge \ldots \wedge \beta_m) \rceil =$$
$$\{ r_{11} : (\lceil \beta_1 \rceil_{out} \leftarrow \lceil \alpha_1 \rceil_{in}); \ldots ; r_{1m} : (\lceil \beta_m \rceil_{out} \leftarrow \lceil \alpha_1 \rceil_{in})$$
$$; \ldots ;$$
$$r_{n1} : (\lceil \beta_1 \rceil_{out} \leftarrow \lceil \alpha_n \rceil_{in}); \ldots ; r_{nm} : (\lceil \beta_m \rceil_{out} \leftarrow \lceil \alpha_n \rceil_{in}) \}$$

$$\lceil l_1 \wedge \ldots \wedge l_n \rceil_{in/out} = \lceil l_1 \rceil_{in/out}, \ldots, \lceil l_n \rceil_{in/out}$$
$$\lceil a \rceil_{in} = in\_a \qquad \lceil a \rceil_{out} = out\_a$$
$$\lceil \neg a \rceil_{in} = \neg in\_a \qquad \lceil \neg a \rceil_{out} = \neg out\_a$$

we call rules r_{ij} as instances of r and we informally write $r_{ij} \in Ints(r)$.

Notice that the program resulting from the application of $\lceil \cdot \rceil$ has a unique model because it is negation-as-failure-free (NAF). Given a set of obligations $\mathbb{O}$, its closure $\mathbb{O}'$ under the rules of Fig. 3 exists and is finite.

Lemma 1. *Given a set of obligations $\mathbb{O} = \{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$ and its closure $\mathbb{O}'$ under the rules defined in Fig. 3 we have*

$$\textit{If } (\alpha, \beta) \in \mathbb{O}' \textit{ then } \lceil \beta \rceil_{out} \in \mathcal{E} \in$$
$$EXT(\{\lceil (\alpha_1, \beta_1) \rceil; \ldots ; \lceil (\alpha_n, \beta_n) \rceil\} \cup \lceil \alpha \rceil_{in})$$

Proof. *First, we notice that $\mathcal{E}$ is unique (see Corollary 4.1). The* if *direction is trivial while the* only if *can be proved by showing that every application of the immediate consequence operator $\mathcal{T}$ (as defined in [9]) can be encoded into an application of the rules in Fig. 3.* □

We now show how to extend the preference relation $\succ$ w.r.t. rules generated with $\lceil \cdot \rceil$ as in Def. 3

Definition 4. *Given a normative code $\mathbf{G} = \langle \mathbb{O}, \mathbb{P}, \succ \rangle$ we define a transformation $Tr_o(\cdot)$ such that $Tr_o(\mathbf{G}) = \langle \lceil \mathbb{O} \rceil, \mathbb{P}, \succ' \rangle$ where $\succ'$ is defined as follows:*

- *$t_{ij} \succ' t'_{i'j'}$, for all $t_{ij} \in Inst(t)$ and $t'_{i'j'} \in Inst(t')$ for $t, t' \in \mathbb{O}$ such that $t \succ t'$.*

For this reason, for a given normative code $Tr_o(\mathbf{G})$, we define a further transformation $Tr_p(\cdot)$ defined as follows

Definition 5. *Given a normative code $\mathbf{G_o} = Tr_o(\mathbf{G}) = \langle \lceil \mathbb{O} \rceil, \mathbb{P}, \succ' \rangle$ we define $Tr_p(\mathbf{G_o}) = \langle \lceil \mathbb{O} \rceil, \mathbb{P}, \succ'' \rangle$, where $\succ''$ is defined as follows:*

- *For all $p : (\alpha, l) \in \mathbb{P}$, $p \succ'' t_{ij}$, for all $t_{ij} : (\alpha, \neg l) \in \lceil \mathbb{O} \rceil$*

We now discuss how to encode priorities between rules into extended logic programs [15].

Definition 6. *Given a preference relation between r_i and r such that $r_i \succ r$ for $1 \leq i \leq j$,*
Replace the clause $r : L_1, ..., L_p \rightarrow L_{q+1}$ by clause $L_1, ..., L_p, \sim L^1_{p+1}, ..., \sim L^1_q, ..., \sim L^j_{p+1}, ..., \sim L^j_q \rightarrow L_{q+1}$,
where $r_{i(1 \leq i \leq j)} : L^i_{p+1}, ..., L^i_q \rightarrow L^i_{q+1}$;

Example 1. *Suppose to have the following normative code $\mathbf{G} = \langle \{r : (a, \neg b \wedge c)\}, \{p : (d, b)\}, \{\} \rangle$, then $Tr_o(\mathbf{G}) = \{\langle r_{11} : (a, \neg b); r_{12} : (a, c)\}, \{p : (d, b)\}, \{\} \rangle$ and $Tr_p(Tr_o(\mathbf{G})) = \{\langle r_{11} : (a, \neg b); r_{12} : (a, c)\}, \{p : (d, b)\}, \{p \succ r_{11}\} \rangle$.*

Rules with permissions in the consequent, which are of the form $p_i : L_{i_1}; \ldots ; L_{i_n}; L_{i_{n+1}}; \ldots ; L_{i_m} \rightarrow L_{i_{m+1}}$ such that, for any other rule $r : L_{i_1}; \ldots ; L_{i_n} \rightarrow \neg L_{i_{m+1}}$ (resulting from the application of $\lceil \mathbf{G} \rceil$) we impose $p_i \succ r$. The role of permission rules is to undercut (obligations rules) in $\lceil \mathbf{G} \rceil$ and will not be encoded into the symbolic neural network (every output encoded in the NN counts as an obligation, permission are not represented in the network but something is permitted if the contrary is not obligatory, see Section 4.2).

Lemma 2. *Let $P_\succ = \{r_1, r_2, ..., r_n\}$ be an extended program with an explicit superiority relation $\succ$. Let P denote the translation of $P_\succ$ into a program without $\succ$. We have that $EXT(P_\succ) = EXT(P)$.*

$$\frac{(\alpha, \alpha_{o_1} \wedge \alpha_{o_2} \wedge \ldots \wedge \alpha_{o_n})}{(\alpha, \alpha_{o_2} \wedge \ldots \wedge \alpha_{o_n})}\,(WO) \quad \frac{(\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n, \beta)}{(\alpha_2 \vee \ldots \vee \alpha_n, \beta)}\,(WI) \quad \frac{(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}, \alpha_{o_1}) \qquad (\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}, \alpha_{o_2})}{(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}, \alpha_{o_1} \wedge \alpha_{o_2})}\,(CO)$$

$$\frac{(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}, \gamma_{o_1}) \qquad (\beta_{i_1} \wedge \ldots \wedge \beta_{i_n}, \gamma_{o_1})}{((\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}) \vee (\beta_{i_1} \ldots \beta_{i_n}), \gamma_{o_1})}\,(DI) \qquad \frac{(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n}, \alpha_{o_1})}{(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_n} \wedge \beta_{i_1}, \alpha_{o_1})}\,(SI)_{\text{with } \beta_{i_1} \in I(G)}$$

Figure 3: Semantics for I/O Logic

We are interested in the translations above between $P_{\succ}$ and P because it is well-known that CILP networks will always settle down in the unique answer set of P provided P is well-behaved (i.e. locally stratified or acyclic or acceptable, see [7]). This result will be explored further in what follows.

4.2 The N-CILP algorithm

In this section we introduce the translation algorithm we have implemented in order to encode a normative code into a feed-forward NN (with semi-linear neurons), namely the Normative-CILP (N-CILP) algorithm. The proposed algorithm differs from standard CILP [7] in how priorities are encoded into the resulting neural network and does not connect input and output neurons that represent the same atom.

N-CILP

Given a normative code $\mathbf{G}$

1. $\mathbf{G}' = Tr_o(\mathbf{G}); G'' = Tr_p(\mathbf{G}')$
2. Apply the encoding of priorities as described in Definition 6 to $\mathbf{G}''$.
3. For each rule $R_k = \beta_{o_1} \leftarrow \alpha_{i_1}; \ldots; \alpha_{i_n}; \sim \alpha_{i_n+1}; \ldots; \sim \alpha_{i_m} \notin \mathbb{P}$.
   (a) For each literal α_{i_j} $(1 \leq j \leq m)$ in the input of the rule. If there is no input neuron labeled α_{i_j} in the input level, then add a neuron labeled α_{i_j} in the input layer.
   (b) Add a neuron labeled N_k in the hidden layer.
   (c) If there is no neuron labeled β_{o_1} in the output level, then add a neuron labeled β_{o_1} in the output layer.
   (d) For each literal α_{i_j} $(1 \leq j \leq n)$; connect the respective input neuron with the neuron labeled N_k in the hidden layer with a positive weighted arc.
   (e) layer with a negative weighted arc (the connections between these input neurons and the hidden neuron of the rule represents the priorities translated with the NAF).
   (f) Connect the neuron labeled N_i with the neuron in the output level labeled β_{o_1} with a positive weighted arc (each output in the rules is considered as a positive atom during the translation, this means that if we have a rule with a negative output $\neg\beta$, in the network we translate an output neuron labeled β' that has the same meaning of $\neg\beta$ but for the translation purpose can be treated as a positive output).

PROPOSITION 1. *For any normative code in the form of an extended logic program there exists a neural network obtained from the N-CILP translation algorithm such that the network computes the answer set semantics of the code.*

PROOF. *Def. 3.3 translates a normative code into an extended logic program having a single extension (or answer set). From Lemma 3.11, the program extended with a priority relation also has a single extension. In [7] it is shown that any extended logic program can be encoded into a neural network. N-CILP performs one such encoding using network weights as defined in [7]. Hence, N-CILP is sound. Since the program has a single extensions, the iterative recursive application of input-output patterns to the network will converge to this extension, which is identical to the unique answer set of the program, for any initial input.* □

5. EXPERIMENTAL RESULTS

The N-CILP algorithm was implemented as part of a simulator which is available online. In the simulator, the KB contains the rules that an agent knows. We assume that the priorities are embedded in the rules following the description used in the previous section. The KB is then read as input for the *N-CILP* translation which produces a standard NN for training. The network can be then trained within the simulator by backpropagation.

In this section, we describe the results of experiments carried out using the N-CILP simulator for network translation and training.

To evaluate the performance of the network, we use two distinct measures: *tot* and *part*.

$$tot = \frac{\sum_{i=1}^{n} I(\bigwedge_{j=1}^{k} (c_{ij} == o_{ij}))}{n}$$

$$part = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k} I(c_{ij} == o_{ij})}{n * k}$$

where n refers to the cardinality of the test set, k is the number of output neurons in the network, o_{ij} is the value of the j-th output of the NN for the i-th test instance, c_{ij} is the true value (desired value) of the j-th literal for the i-th test instance, $I(\cdot)$ is the indicator, a function returning 1 if the argument is true and zero otherwise. The *tot* measure evaluates how many instances were processed entirely correctly, whle *part* considers the number of single output neurons correctly activated.

In our experiments we train the network using a *10fold cross validation*. We divide the initial data set of instances in ten distinct subsets. Each subset is then used as test set while the others are used together as training set. In this way the instances seen during training are left out of the testing phase, ten networks are trained and the results are averaged. The test-set performance provides us with an estimate of the network's generalization capability, i.e. its ability to predict the results (network output) for new instances (inputs), not seen during training. In all the experiments, we set the training parameters for the networks as follows: *learning rate*: 0.8, *momentum*: 0.3 and *training cycles*: 100. The reader is referred to [10] for the details of the backpropagation learning algorithm with momentum.

Non-symbolic approach comparison: we compare the learning capacity of a network built with N-CILP with a non-symbolic neural network. One of the well known issues in neural-network training is how to decide the number of neurons in the hidden layer. In the case of N-CILP, this number is given by the number of symbolic rules. We adopt the same number of hidden neurons for both networks, in order to avoid the risk of an unfair comparison with a randomly assessed topology for the non-symbolic network. The difference between the networks involved in this test lies in their connection weights. The neural network built with N-CILP sets its weights according to the rules in the KB. Instead, the non-symbolic network has its weights randomly initialized. One advantage of a network built with N-CILP is that even without any training, it is capable of correctly processing certain instances by applying the rules contained in the KB (if the rules are correct).

The network built with N-CILP has the head-start of a KB containing 20 rules. During the training phase, the network tries to learn 9 additional rules provided in the form of training instances (examples of input/output patterns). The non-symbolic network is provided with the same instances, including the instances for the initial 20 rules, but has to learn all the 29 rules using backpropagation.

The results from this little experiment show that the non-symbolic neural network is not able to achieve the same level of accuracy as the N-CILP network. For the non-symbolic network $tot = 5.13\%$ and $part = 45.25\%$. For the N-CILP network $tot = 5.38\%$ and $part = 49.19\%$. We can see that with the same knowledge provided as rules or instances, the networks achieve different results with the N-CILP network showing an improved performance.

Enhancing the knowledge base: the second experiment measures how the neural network performs by increasing the number of rules in the knowledge base. This test is important because the goal of a *Neural-Symbolic System*, is not only to construct a neural network capable to compute the same semantics as rule into the knowledge base. Another important objective is to exploit the learning capabilities of the neural networks, allowing the agent to increase the number of rules in its knowledge base from what it learned[7].

The test is done incrementally. From the full set of 29 rules, the experiment first step starts with a knowledge base containing 20 rules and tries to learn the remaining 9. Successively 2 rules are incrementally added into the initial knowledge base during each step. In this way the unknown rules that the network has to learn decreases by 2 each step. In example at the second step of the experiment the starting knowledge base contains 22 rules and the network tries to learn 7 rules during the training phase.

During each step the neural network is tested over instances where the full set of rules is applied. In this way the network continues to process using the rules already known, reducing the risk to forget them and in the meantime it tries to learn of the unknown rules.

The results of this experiment are shown in Figure 4. We can see that for the first two steps of the experiment the accuracies measured quite low. instead for the last two steps the performance of the neural network increases, reaching an accuracy peak of 98,01% for the *part* measure and 91,18% for the *tot*.

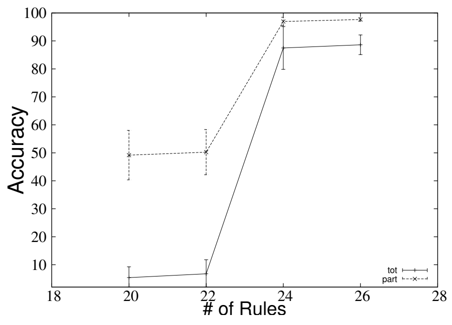

Figure 4: Accuracy of *tot* and *part* measures increasing the number of rules

From the experiment proposed we observed a direct correlation between the number of the rules in the starting knowledge base and the performance of the neural network. Another thing that can be noticed is that the smaller becomes the number of rules that the network does not know, w.r.t. the number of rules in the initial knowledge base can impact the performances of the network, also due to the fact that a network built from a larger knowledge base possesses more connections.

Learning Contrary to Duties: in this test we measure the capacity of a neural network built with N-CILP to learn new contrary to duties. In this case we use a starting knowledge base where the priority-based orderings regulating the contrary to duties were missing.

We tested the network on learning three different contrary to duties. The first refers to a situation where a robot player should never impact on an opponent. But if a collision route is inevitable, then the robot should make its best to minimize the impact. The second manages the situation where the robot is in physical contact with an opponent, which is forbidden by the RoboCup rulings. The robot should then try to terminate the contact. The third handles the situation where the robot is touching the ball with his hands, but he is not supposed to.

By removing the priority based orderings what is obtained is an incomplete system that produces, in similar situations, both the unfulfillable obligation and the relative obligation to handle the suboptimal situation that is being analyzed. What we expect from this test is that our approach is capable to learn from the examples, the priority based orderings that regulates the contrary to duties.

The neural network is trained with a set of instances that contain both normal situations and situations in which the contrary to duty is applied. The resulting network is tested with a test set containing sub-optimal situations, where an application of the contrary to duty is necessary. From the results of this test we verify that regarding the first contrary to duty, in the test set 95% of the instances were processed correctly and generating only the output obligation for the suboptimal situation that is what is desired on those situations. For the two other contrary to duties, we obtain an accuracy equal to 93% and 87% with their respective test sets.

Our approach is capable to learn contrary to duties not included in the construction of the neural network. This is a strength of the neural-symbolic architecture, that allows to avoid a total description of the investigated domain that could be, in some cases, very expensive and infeasible.

6. CONCLUSION

To the best of our knowledge this paper is the first to combine normative reasoning and learning with connectionist systems. Concerning the learning of normative systems in general, it is possible – as this paper also shows through the proposed translation of I/O logic into extended logic programs – to use a purely symbolic set-up. In the experiments proposed we see that a neural-symbolic approach has some advantages w.r.t. a pure connectionist one. This approach solves problems like the decision a priori of the NN size. From the results obtained in the tests, we empirically show that embedding previous knowledge in the NN increases its learning and processing performances. Notably, it should be possible to learn the kind of extended programs that we are considering here through the use of Inductive Logic Programming (ILP) [14] (or some adaptation of it to accommodate the use of negation, for example [16]). ILP has been used successfully in bioinformatics, but we are not aware of its application in normative systems. It would be interesting to compare and contrast the performance of the symbolic and connectionist approaches in the context of a real normative-systems application. Measurable criteria for comparison would include: accuracy, learning performance and noise tolerance indexes.

In this paper we chose to focus on the translation in one direction, as this can be used for tasks such as creating adaptable controllers. As future work, we'll be working on extensions of the tool to include an extraction module so it can be used when explicit explanations are required.

The system described thus far considers only one type of norms: the so called regulative norms, i.e., the norms prescribing the behavior of agents, in terms of what is obligatory, forbidden or permitted. Future work is also introducing constitutive rules besides regulative ones prescribing what is obligatory, forbidden or permitted. Constitutive rules provide a classification of reality in terms of the so called institutional facts, like marriages, licences, authorizations, institutions, etc. In the antecedents of regulative rules refer to the situation in which the norm should apply not only in terms of the brute facts (i.e., to the physical world) but also to institutional facts. Institutional facts are also inputs of constitutive rules meaning that differently than regulative rules, constitutive rules respect cumulative transitivity. Constitutive rules can be seen as a component whose output is fed as input to the component of regulative rules. The challenge is to study the interaction between the learning of the two components.

7. REFERENCES

[1] G. Boella, S. Colombo Tosatto, A. S. d'Avila Garcez, and V. Genovese. On the relationship between i-o logic and connectionism. In *13th International Workshop on Non-Monotonic Reasoning*, 2010.

[2] G. Boella, S. Colombo Tosatto, A. S. d'Avila Garcez, D. Ienco, V. Genovese, and L. van der Torre. Neural symbolic systems for normative agents. In *10th International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[3] G. Boella, G. Pigozzi, and L. van der Torre. Normative framework for normative system change. In *8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems AAMAS 2009*, pages 169–176. IFAAMAS, 2009.

[4] G. Boella and L. van der Torre. Permission and authorization in normative multiagent systems. In *Procs. of Int. Conf. on Artificial Intelligence and Law ICAIL*, pages 236–237, 2005.

[5] G. Boella and L. van der Torre. A game theoretic approach to contracts in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 36(1):68–79, 2006.

[6] J. Broersen, M. Dastani, J. Hulstijn, and L. van der Torre. Goal generation in the BOID architecture. cognitive science quarterly. In *Cognitive Science Quarterly*, volume 2(3-4), pages 428–447, 2002.

[7] A. d'Avila Garcez, K. Broda, and D. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing. Springer, 2002.

[8] A. S. d'Avila Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11:59–77, July 1999.

[9] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[10] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[11] D. Makinson, L., and V. D. Torre. Input-output logics. *J. of Philosophical Logic*, 29:2000, 2000.

[12] D. Makinson and L. van der Torre. Constraints for input/output logics. *Journal of Philosophical Logic*, 30:155–185, 2001.

[13] E. Menegatti. Robocup soccer humanoid league rules and setup, 2007.

[14] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

[15] D. Nute. Defeasible logic. In D. Gabbay and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 353–396. Oxford University Press, 1994.

[16] O. Ray. Automated abduction in scientific discovery. In *Model-Based Reasoning in Science, Technology, and Medicine*, volume 64 of *Studies in Computational Intelligence*, pages 103–116. Springer, 2007.

[17] S. Sen and S. Airiau. Emergence of norms through social learning. In *Procs. of the 20th International Joint Conference on Artificial Intelligence - IJCAI*, pages 1507–1512, 2007.

[18] Y. Shoham and M. Tennenholtz. On the emergence of social conventions: Modeling, analysis, and simulations. *Artif. Intell.*, 94(1-2):139–166, 1997.

[19] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artif. Intell.*, 70:119–165, October 1994.