Relational Learning for NLP using Linear Threshold Elements

Roni Khardon*
Division of Informatics
University of Edinburgh
Edinburgh, EH9 3JZ
Scotland
roni@dcs.ed.ac.uk

Dan Roth*
Computer Science
University of Illinois, Urbana
Urbana, IL 61801
USA
danr@cs.uiuc.edu

Leslie G. Valiant*
Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
USA
valiant@deas.harvard.edu

Abstract

We describe a coherent view of learning and reasoning with relational representations in the context of natural language processing. In particular, we discuss the Neuroidal Architecture, Inductive Logic Programming and the SNoW system explaining the relationships among these, and thereby offer an explanation of the theoretical basis for the SNoW system. We suggest that extensions of this system along the lines suggested by the theory may provide new levels of scalability and functionality.

1 Introduction

The paper explores some aspects of relational knowledge representation and their learnability. While the discussion is to a large extent general it is made in the context of low-level natural language processing (NLP) tasks. Recent efforts in NLP emphasize empirical approaches, that attempt to learn how to perform various natural language tasks by being trained using an annotated corpus. These approaches have been used for a wide variety of fairly low level tasks such as part-of-speech tagging, prepositional-phrase attachment, context-sensitive text correction, and word selection in speech recognition and translation.

In this paper we study the general form of such problems in a relational setting. Thus, our interest here is in *learning to perform* such tasks given a set of previously processed sentences, each represented using a relational representation. We use the task of *Part of speech tagging (POS)* - one of the basic tasks studied in this line of research and often viewed as a prerequisite to performing many of the other tasks - as our running example. POS is the task of assigning each word in a given sentence the part of speech it assumes in that sentence. For example, assign N or V to talk in the following pair of sentences: Have you listened to his (him) talk?

The SNoW system [Roth, 1998] has been used successfully for several NLP tasks and, in particular, for learning to perform part of speech tagging [Roth and Zelenko, 1998]. The system uses perceptron-like representations and propositional learning algorithms. On the other hand, relational representations are very natural for NLP tasks and logic programs have been widely used in this context [Hobbs *et* al., 1993; Pereira and Shieber, 1987]. A natural approach would thus be to apply learning in first order logic or Inductive Logic Programming (ILP) for learning to perform the relevant tasks, as done e.g in [Cussens, 1997; Dzeroski and Erjavec, 1997].

In this paper we discuss the theoretical basis underlying the SNoW system in terms of the Neuroidal Architecture, particularly following specific suggestions [Valiant, 1998a; 1999] for representing relational information using linear threshold elements as discussed earlier in [Valiant, 1994]. Using this analysis we show that SNoW can be viewed as a relational learning system, and this enables us to go on to compare it more directly with ILP.

The main contribution of the paper is therefore in analyzing the SNoW system in terms of the Neuroidal Architecture and consequently as a system learning relational information. This analysis provides insight into some of the theoretical and practical considerations that have been made in the search for scalable systems in this context. Secondly, since the theory is somewhat more general than current implementations, this suggests that extending the SNoW system in a manner consistent with the theory may provide new levels of functionality. Finally, by providing a single framework for discussing both SNoW and ILP we enable future comparative work of these seemingly disparate approaches.

The presentation is organized as follows. We first describe the input (a sentence) and program representation (Horn rules and variants) for the relational tasks under consideration. We then discuss various computational aspects of the representations and their learnability. This leads to a description of the SNoW system and a discussion of some of the pragmatics of systems that learn to perform these tasks.

* Partly supported by EPSRC grant GR/M21409.

+Research supported by NSF grants IIS-9801638 and SBR-9873450.

* Research supported by grants NSF-CCR-95-04436 and ONR-N00014-96-1-0550.

2 Representing a Sentence

We use first order logic statements with a finite set of predicates and allow only constants (i.e. no other function symbols) in expressions. Since the specific predicates used do not affect the conceptual issues, we shall give a simple running example that will serve to illustrate the computational issues.[1] The representation is as follows:

- Each word has a constant associated with it.
- The linear structure of the sentence is captured by the predicate $bef(w1, w2)$ which means that word $w1$ is immediately before *w2* in the sentence.
- A variety of relations conveying linguistic or semantic information may be added. For example we may add $ppos(w,p)$ **if** p is a possible part of speech for word w. Another example is adding $isa(w1, w2)$ where $w1$ is a word describing an object in a class of objects described by the word $w2$.

For simplicity we assume in the examples below that 6e/() and *pposQ* are the only relations used in the representation. Thus the sentence this can will rust is represented as the collection of all the atoms that hold (are active) in it:

S = {be f (this, can),bef (can, will),bef (will, rust),
ppos(this, art),ppos(can, noun), ppos (can, verb),
ppos(will, noun),ppos(will, verb),ppos(rust, noun),
ppos(rust,verb)} (1)

It is implicit in the above that atoms not listed (such as *be f (this,rust))* are false for this sentence.

The above representation may be too restricted if words appear more than once in a sentence. This can be easily dealt with formally by replacing constants representing words with unary predicates and using "token constants" to represent positions in the sentence [Valiant, 1998a; 1999]. In order to keep the notation simple we ignore this issue in the rest of the paper.

3 Representing Programs

3.1 Horn Representations

A program should take a sentence as an input and produce part of speech tags for the words in this sentence. One possible representation for such a program is a set of Horn rules. For example, the rules[2]

$$R_1 = [\forall x, (\exists y, z, bef(x,y) \wedge bef(y,z) \wedge ppos(z, verb)) \rightarrow f_1(x)]$$
$$R_2 = [\forall x, bef(x, will) \rightarrow f_2(x)]$$
$$R_3 = [\forall x, f_1(x) \wedge f_2(x) \rightarrow pos(x, noun)]$$

represent a simple program for predicting whether a word is a noun or not. These rules exemplify several aspects of the representation. First, we may want to construct "complex features" such as $f_1()$ and f2() out of the base predicates. Secondly, such features may include existentially quantified variables as in the first rule or constants as in the first or second rule. Note that the same program can be represented by the single rule

$$R_4 = [\forall x, (\exists y, z, bef(x,y) \wedge bef(y,z) \wedge ppos(z, verb)) \wedge bef(x, will) \rightarrow pos(x, noun)]$$

3.2 Threshold Elements with Quantified Propositions

This section reviews a previous proposal for implementing relational predicates by means of linear threshold elements [Valiant, 1998a; 1999]. Consider a relational Horn rule

$$R_5 = [\forall x, (\exists y, c_1(x,y) \wedge c_2(x,y)) \rightarrow new(x)]$$

where y appears only in the antecedent. By an appropriate choice for w_1, w_2, θ, this rule can be represented using a threshold element

$$R_6 = [\forall x, (\exists y, [w_1 \cdot c_1(x,y) + w_2 \cdot c_2(x,y) > \theta]) \rightarrow new(x)]$$

We impose the restriction that the *quantified antecedent variables appear in each condition separately.* That is, the rule *R6* is not legal since *y* is shared between c_1 and C2. An example for a legal rule (not equivalent to the one above) is:

$$R_7 = [\forall x, ((\exists y_1 c_1(x, y_1)) \wedge (\exists y_2 c_2(x, y_2))) \rightarrow new(x)]$$

In fact, our restriction also allows for universally quantified variables within a single condition (which differs from quantification outside the rule). Under this restriction, for any sentence and given a binding for x, the conditions of the form $(Qy, c(x,y))$ (where Q is a quantifier) are essentially Boolean variables. That is, they are either true or false but have no parameters. We call these *quantified propositions* to emphasize this point.

As before, rules that use quantified propositions can be described using a linear threshold element:

$$R_8 = [\forall x, [w_1 \cdot (\exists y_1 c_1(x, y_1)) + w_2 \cdot (\exists y_2 c_2(x, y_2)) > \theta] \rightarrow new(x)]$$

Thus a special case of rule representations is used but it is generalized through the use of linear threshold elements (that can represent $\wedge$ as well as other functions), and more general quantifiers.

Note that the restriction can be overcome by changing the set of relations. For example, instead of R_5 above we could have

$$R_9 = [\forall x, y, (c_1(x,y) \wedge c_2(x,y)) \rightarrow f_3(x,y)]$$
$$R_{10} = [\forall x, (\exists y, f_3(x,y)) \rightarrow new(x)]$$

While the resulting program is the same, the representation will need to include f3() that explicitly mentions the variable *y.*

4 Learning

In this section we discuss the learnability of programs of the form presented in Sec. 3 in a *supervised learning* paradigm. The input to the learning algorithm is given in terms of an input sentence representation 5 as in

*The examples are meant only to illustrate the language used and may not be the most accurate - indeed, one reason learning is crucial in this domain is that it is hard to come up with concise rules that perform well.

[2] Note that the first rule is logically equivalent to $[\forall x, y, z, (bef(x,y) \wedge bef(y,z) \wedge ppos(z, verb) \rightarrow f_1(x))]$, but the existential presentation is more intuitive from an operational perspective.

Eq. (1) along with an atom p, such as pos (rust, verb), that holds in this sentence. We call the pair < S,p > a *labeled example.* A single sentence may thus give rise to several examples, one for each word.

In the following discussion it is sometimes useful to assume that there is a program in the class of programs discussed, that can produce correct labels for any sentence. Such a program is called the *target program,*

4.1 Relational Methods

An example < S,p > can be represented as (5 → *p).* An example of this form is a ground Horn clause that is implied by the target program (a set of Horn rules). This is exactly the scheme of *learning from entailment* studied in inductive logic programming. (Notice that this representation assumes that negative atoms are not relevant for the label i.e. we cannot use $\neg bef(x, can)$ in the condition of a rule). Several techniques from ILP use examples in this form, and can be applied to this task [Muggleton and De Raedt, 1994; Cohen, 1995a].[3] One of the main features that distinguishes ILP from other formalizations of learning is the use of background knowledge. A similar effect can be achieved in our formalization by enriching the input representation of sentences with the relevant relational information. Thus, given rules, say, that compute *isa()* relations between words, (e.g., using WordNet) we can augment our initial representation *S* with the *isa()* atoms implied by these rules (for words in the sentence).

Studies in ILP suggest that unless the rule representation is restricted the learning problem is intractable. We briefly discuss some of the common restrictions studied and their relation to the use of quantified propositions. A Horn clause is *constrained* if all the variables in the consequent also appear in the antecedent [Page and Prisch, 1992]. This is a special case of *determinacy* defined as follows. Assume one imposes an order (left to right) on the predicates in a rule's antecedent. A predicate is *determinate* if in any example and given a binding for the consequent and the first *i* — 1 predicates, there is at most one binding that makes the i'th predicate true. Clearly, given a binding for the consequent, every predicate in the condition is a Boolean variable, similar to our restriction above, a fact which has been used both in theoretical results and practical applications [Dzeroski *et* a/., 1992; Lavrac and Dzeroski, 1994]. In our case, while there may be more than one binding for each quantified proposition $(\exists y \ldots)$ the condition is still Boolean.

Another related restriction is the *depth* of terms in a clause. We assign a depth to terms using the order on conditions in the antecedent from left to right. A term in the consequent is of depth 0. If $p(u, v, \ldots)$ is the next atom in the condition, such that *u* has been assigned depth *i* (and no other term has been assigned a lower depth) and *v* has not yet been assigned a depth, then *v* is of depth *i* + 1. Depth is only assigned to "linked" clauses where at least one term in any atom is already assigned a depth by the previous atoms. Clearly our restriction means that variables in clauses are of depth at most 1 but constants may create chains of longer depth. Positive results in ILP [Dzeroski *et* a/., 1992; Cohen, 1995a] show that a single non-recursive (or with limited recursion) determinate clause of constant depth is learnable in polynomial time. However, relaxing any of the restrictions, to 2 clauses, non-constant depth, recursive clauses, or non determinate clauses makes the problem computationally hard [Kietz and Dzeroski, 1994; Cohen, 1995b; Cohen and Page, 1995].

On the other hand, there are heuristics for learning programs with more than one rule and these may be used here as well [Muggleton and De Raedt, 1994; Cussens, 1997]. Another way to make the problem tractable is to use queries. In particular, for *entailment membership queries,* the learner presents a new example (S_n -> p_n) and asks whether it is implied by the target. If a user can answer such questions or if one can otherwise simulate these then techniques from [Khardon, 1998; Reddy and Tadepalli, 1998] can be used to learn programs with more than one rule.

To summarize, the restriction imposed by using quantified propositions creates a situation that is similar to determinacy and constant depth in that it reduces the complexity of the learning problem but is incomparable to these. One important difference is the fact that determinacy is a property of the data that may not hold for every domain whereas our restriction is purely syntactic.

4.2 Using Quantified Propositions

In the rest of this section *we concentrate on the problem of learning a single rule that uses quantified propositions and can be expressed given the predicates in the representation of the sentence.* To make this concrete, in terms of the examples above, we might be trying to learn the rule *R3* from sentence representations that include 6e/(),ppos(),f1(),/2(). In the next section we discuss combining rules learned separately in a single system.

Projection

A possible reduction in complexity comes from splitting the learning task into a few smaller ones. This is natural for part of speech tagging. It seems unlikely that the same condition can be used to predict the part of speech of, say, both nouns and verbs. Therefore, it may be better to avoid trying to learn a general rule of the form *condition →pos(x, y)* and instead learn a set of rules, one (or more) for each part of speech, e.g. *(conditioni → pos(x,noun))> (condition2 → pos(x,verb))* etc. If general patterns do not occur (and thus will not be useful) then we are effectively reducing the complexity of the learning problem.

One can think of this as *projecting* the relation *pos(x,y)* over its second argument (the pos tag), where *posQ* is implemented as a logical disjunction of the various sub-relations. This clearly works if the instances

[3] Previous ILP work for similar problems [Cussens, 1997; Dzeroski and Erjavec, 1997] use a different formalization and are therefore not directly comparable here.

describe disjoint sets, which they do. The projection we use here (for *pos())* depends on the label of the example. This does not cause a problem during the learning process but one may need to resolve between competing learned rules when the programs are used. Other projections, that depend only on the input sentence, can be used in other NLP tasks. In fact, this has been done implicitly in NLP studies, where the task is defined as disambiguating between a small set of candidates.[4]

Our examples of projections simply partition the learned predicate into several disjoint parts, where the parts are identifiable in advance, and all parts are needed (e.g. we need to learn *pos(x, y)* for each possible value of *y).* Projection learning [Valiant, 1998b] provides some more general conditions under which learning projections on the input space is guaranteed to work. This holds even if one has a set of projections that is not disjoint and not all projections are needed. In addition, as we discuss below, this can be done in an attribute efficient manner.

Using Propositional Learning Algorithms

As presented in Sec. 3.2 the Neuroidal Architecture [Valiant, 1998a] restricts the use of rules so that quantified variables in different parts of the condition can be evaluated separately from the others. The main advantage of this restriction is that it allows one to encode relational learning examples into a propositional setting. Consider learning the rule

$R7 = [\forall x, ((\exists y_1 c_1(x, y_1)) \wedge (\exists y_2 c_2(x, y_2))) \rightarrow new(x)]$

where C1() and C2O are the features used in the representation. For a sentence representation 5 and a binding of the variable *x* (which determines the example provided to the learning algorithm) each predicate in a rule (e.g. $\exists y_1 c_1(x, y_1))$ is assigned a single binary value. Thus an example can be described by assigning a binary value to each of the given features. In our example sentence (learning *R3* in terms *of bef(),ppos(), f1*(), f2O) for *x* =this we have *f1(x)* = 1, f2*{x)* = 0, bef(x,*can)* = 1, and for *x* =can we have *f1{x)* = 1, f2(x) = 1, and bef(x, *can)* = 0. Since an example always specifies the binding for the atom in the consequent this can always be done.[5] As a result, under this restriction any propositional learning algorithm can be used for learning, even though the rule itself is relational! The rule learned is an approximation to an equivalence rather than an implication [Valiant, 1999].

Dealing with Many Attributes

Recall that in the ILP setting an example is a clause of the form (5 —> p), implicitly assuming that the atoms that do not hold in the sentence are irrelevant for it. This "closed world assumption" is especially important if the number of atoms that hold in a sentence (positive atoms) is much smaller than the number of those that do not hold in it (negative atoms). This is likely to be the case in the current context especially if we use features that are partly ground, such as bef(x, *can), bef(x, will), bef(x, table)* etc., repeated for every word in our lexicon. In this setting, the number of potential features may be in the order of 10^5 and explicitly mentioning all of them may be computationally costly.

Therefore, when using a propositional learning algorithm in this setting one would like to make sure that the algorithm handles only the features active in the example and, preferably, does not even represent negative features since even their representation and manipulation will be time consuming. In turn, this efficiency requirement may imply that the program learned cannot depend on negative features. The infinite attribute model suggested by Blum [1992] provides a theoretical framework in which this can be studied. In this model, an example is represented as a list of the attributes that are active in it, and a learning algorithm is required to be efficient (polynomial) in terms of the number of the features active in the examples rather than the total number of attributes. To a certain extent, algorithms in the infinite attribute model can deal with negative features [Blum, 1992].

Attribute Efficient Algorithms

The sample complexity - the number of examples required in order to achieve learnability - is another important aspect that requires attention. Consider the case when the target program depends only on n_r (relevant) attributes, which is small relative to n_c, the number of attributes active in any particular example, and very small relative to n_a the number of potential attributes. As indicated above, this is realistic in many NLP problems, in which the number of potential attributes depends on the size of the lexicon. In this case, one would like the number of examples required for learning the program to depend only weakly on n_e or na. Littlestone's Winnow algorithm [Littlestone, 1988] (adapted to the infinite attribute model) can achieve this and requires $0(n_r \log n_c)$ examples. That is, the number of examples needed is independent of n_a and depends only logarithmically on n_e.

An important aspect of projection learning, as mentioned previously, is that it can be achieved attribute efficiently [Valiant, 1998b]. This requires no special attention if the projection forms a partition, as in the POS example, but even in a more general case when we have a large number of projections of which only a small number is relevant, there is a logarithmic dependence on the number of irrelevant projections.

[4] When learning to perform context sensitive spelling, the task is to decide what is the correct word to use for a given confusion set e.g. piece or peace. In this case, projection on the confusion set (i.e., learning for piece, peace and for weather, whether separately) makes sense since it is unlikely that the same condition will disambiguate different sets.

[5] This should be contrasted with the expression *R5* = [Va;,*(3y,Ci(x,y)* Ac2(x,y) -» *new(x))]* where for each binding for *y* we still get a binary value for each attribute, but these binary values are dependent upon each other, so that the example cannot be described simply using a set of binary values.

Target-Word Centered Representation

Our initial description used a single representation for a sentence, as in Eq. (1). In this formulation, a single sentence gives rise to several examples, each corresponding to the part of speech of a word in the sentence, but all have the same relational representation and differ only in their label. It is thus worth noting that when using the propositional translation this does not hold. In particular, the Boolean value of the features depends on the binding for x, the word for which we are trying to predict the part of speech. In this way, a sentence produces several examples, each corresponding to a word and its part of speech, but the propositional representation of each of these examples is different.

5 The SNoW System

The SNoW system[6] provides an architecture within which several learning algorithms - all making predictions using linear functions over the feature space - can be implemented. The architecture is a network of linear threshold gates. Nodes in the input layer of the network represent simple relations on the input sentence and are being used as the input features. Target nodes represent features for which programs are sought. Target nodes are linked via weighted edges to input features; if F_t is the set of features linked to the target node *t, Wi* is the weight associated with the ith feature, and 0_t is a threshold associated with t, we can say that *t* is represented by the generalized rule

$$\text{if } (\sum_{i \in \mathcal{F}_t} \omega_i x_i > \theta_t) \text{ then } t \text{ is active.}$$

Each SNoW *unit* may include a collection of subnetworks, one for each of the relations for which a program is learned. A given example is treated autonomously by each target subnetwork; an example labeled *t* is treated as positive example by the subnetwork for *t* and as a negative example by the rest of the target nodes (modulo projection considerations). The learning policy is online and mistake-driven, and the most successful update rule used within SNoW is the Sparse Winnow Algorithm. This is a variant of Littlestone's [1988] multiplicative update rule, tailored to the situation in which the set of input features is not known a priori, as in the infinite attribute model [Blum, 1992]. Once target subnetworks have been learned and the network is being evaluated, a decision support mechanism is employed to select the dominant active target node in the network or to output an otherwise coherent output [Munoz *et al,* 1999].

Originally, the SNoW system was described as a propositional system. In the rest of this section we shall detail a new interpretation of it as a relational system. We briefly discuss how-some of the aspects presented earlier are reflected in the design of the system, and then present a few more pragmatic aspects that are somewhat harder to discuss theoretically at this point. Most of these aspects have been evaluated experimentally and found to perform successfully on a variety of NLP tasks. In particular, single SNoW units have been evaluated on context-sensitive spelling correction [Golding and Roth, 1999]; chaining of SNoW units (as well as other aspects) has been studied in [Roth and Zelenko, 1998; Munoz *et* al., 1999] several aspects of projections have been studied in [Rosen, 1999], and a preliminary study of incorporating background knowledge by augmenting the input features is in [Krymolovsky and Roth, 1998].

5.1 Sentence representation

While the description above suggests that SNoW uses a propositional representation, a relational representation of sentences as in Eq. (1) is used. This is done by utilizing quantified propositions and using a propositional representation of these as features. Accordingly, examples are target-word centered and a single relational sentence representation may give rise to several propositional examples, corresponding to the target word.

Quantified propositions include features with constants (as in f2() in Sec. 3) and existential features (as in /i() in Sec. 3). The system generates the features and the word-centered representation automatically using a simple language for relational feature definitions. Specifically, features are relations of the form $\exists y, c(x, y)$ where the predicate c() is one of three types: (1) relations that can be readily read from the sentence, like *bef {this, can).* (2) Relations that become active as a result of evaluating other, previously learned programs. (3) Relations that are read from an outside source. These may include predicates like *isa(),* and *synonymQ.* In this way background knowledge is incorporated in a transparent way and plays the same role in the representation as do simpler predicates.

The predicate c() can also be a small conjunction of predicates of the above types. The pattern for existential features is restricted to bind words within a fixed distance of the target word. Thus, /i() above can be rephrased as *"ppos(y, verb)* holds for the word *y* which is two positions to the right of x". Notice that due to the semantics of bef(), these features are determinate in any rule; more general features like *"ppos{y, verb)* holds for some word *y* which is at most 5 positions away from *x"* are also used, but they too have bounded determinacy.

The above syntactic description generates a large number of possible features. Following the positive sentence representation principle discussed above, features are allocated only if they are active in the example sentence. The architecture supports a limited use of negative features by learning several programs at the same time and using positive features for one program as potential negative features for others.

[6] Other NLP learning systems like Brill's system [1995] also incorporate some of the aspects we present; in particular, they use similar features and decompose the tasks similarly. However, SNoW corresponds more directly to the Neuroidal Architecture in its use of threshold elements.

5.2 Learning

- SNoW is a propositional learning system that uses linear threshold functions over the set of quantified propositions as its knowledge representation.
- Two complementary aspects of projection are handled in SNoW. *Precondition* projection is similar to the one described above. This is a condition applied to the input representation (either just the sentence or the label as well) that is employed to learn several target programs rather than a single, more abstract, program as described above for POS and the spelling examples. In this way, different examples can be used to train different programs. *Task* projection is a form of projection that is employed at prediction time, when several learned programs are being evaluated, and is used to restrict the programs that are being evaluated on a given input example, thereby simplifying the prediction process. For example, when learning programs for each of the possible POS, a (learned) task projection may prevent the evaluation of the *pos(x, adj)* program given that the target word is can. In particular, this form of projection addresses one of the issues discussed above, when the projection is with respect to the *label.* An experimental evaluation of the forms of projections within SNoW, exhibiting both significant efficiency and performance gains is presented in [Rosen, 1999].
- SNoW implements several *on line* learning algorithms for linear functions, including variations of Winnow, Perceptron, naive Bayes and a memory based algorithms (the last two use only positive examples for learning). In all cases, the variation implemented is tuned to (be on-line and) deal with many attributes along the lines described above. This is imposed by the *sparse architecture* of the system, which assigns a positive weight to a feature only if it is active together with the target predicate in a single example.

5.3 Other Pragmatic Issues

- SNoW can learn programs that are represented in terms of predicates which are themselves consequents of other programs (that were learned previously). Thus SNoW supports chaining of learned programs assuming that each of the learning stages is supervised. One difficulty in this process is that learned rules may not be completely accurate, resulting in potentially noisy input to the other learning stages. This imposes another practical constraint on the algorithms used in this setting, namely robustness to attribute noise.
- SNoW allows for some use of recursive rules in the following way. When learning from labeled data, the *pos()* labels are available for all words and may be included in the input representation. As a result, the learned rule may be recursive in that it includes *pos()* predicates in the antecedent. However, when evaluating the learned program for *posQ* on new sentences that are not annotated, one needs to initialize the *posQ* predicates in the antecedent in some way (e.g., use the most frequent part of speech for each word) and then, the same program may be applied to the sentence representation (even several times) as to improve the approximate input representation before the output *posQ* is decided upon [Roth and Zelenko, 1998; Munoz *et al.*, 1999].
- Finally, whenever a collection of programs is learned, given an input sentence, there is a need to make a decision that may involve several learned programs. In particular, as in POS, the programs compute competing pos tags and there is a need for a procedure that determines which of them to select. (Note that in SNoW this results from the use of projection.) SNoW employs a few mechanisms for decision making including a winner-take-all mechanism and a *learned* decision making mechanism.

6 Discussion

Based on an analysis of SNoW in terms of the Neuroidal Architecture, we presented a framework describing SNoW as a relational learning system and explained its relation to ILP. This, in particular, facilitates future comparative studies involving SNoW and ILP.

Since the theory presented is more general than current implementations, extensions of SNoW along the lines suggested by the theory may provide new levels of scalability and functionality. This is, however, not straightforward and requires careful consideration. Additionally, our analysis may provide guidelines for further analysis of some issues which are not well understood yet.

One of the issues that can be addressed in future work is the following tradeoff: adding more complex existential features to the representation increases computational complexity, but at the same time results in more expressive programs and more significant gain from the use of attribute efficient algorithms. A second issue is the use of more general forms of projections. Finally, one advantage of relational methods is the ability to choose the right features dynamically and in fact change them during the learning process. For example, if a sentence includes the atom *bef (can, will),* this feature may be needed as part of a learned rule. Naturally, we do not know in advance whether the feature should be ground or whether a more general form, one of *bef(x,will),* 6e/(can,y), or *bef(x,y)* should be used. In the current propositional setting, each of these is used separately as a feature and the learning algorithm may use any of them. On the other hand, computing the least general generalization [Plotkin, 1970] over several example sentences we may be able to find the right general form of the feature. It is not clear whether a technique that changes the features in this way, during the on-line process of learning, can be incorporated in the propositional setting.

References

[Blum, 1992] A. Blum. Learning Boolean functions in an infinite attribute space. *Machine Learning,* 9(4):373-386, 1992.

[Brill, 1995] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics,* 21(4):543-565, 1995.

[Cohen and Page, 1995] W. Cohen and D. Page. Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing,* pages 369-409, 1995.

[Cohen, 1995a] W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research,* 2:501-539, 1995.

[Cohen, 1995b] W. Cohen. PAC-learning recursive logic programs: Negative result. *Journal of Artificial Intelligence Research,* 2:541-573, 1995.

[Cussens, 1997] J. Cussens. Part-of-speech tagging using progol. In *International Workshop on Inductive Logic Programming,* pages 93-108, Prague, Czech Republic, 1997. Springer. LNAI 1297.

[Dzeroski and Erjavec, 1997] S. Dzeroski and T. Erjavec. Induction of Slovene nominal paradigms. In *International Workshop on Inductive Logic Programming,* pages 141-148, Prague, Czech Republic, 1997. Springer. LNAI 1297.

[Dzeroski *et al,* 1992] S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Conference on Computational Learning Theory,* pages 128-135, Pittsburgh, PA, 1992. ACM Press.

[Golding and Roth, 1999] A. R. Golding and D. Roth. A winnow-based approach to context-sensitive spelling correction. *Machine Learning,* 1999. Special issue on Machine Learning and Natural Language; to appear.

[Hobbs *et al,* 1993] R. Hobbs, J, M. Stickel, P. Martin, and D. Edwards. Interpretation as abduction. *Artificial Intelligence,* 63:69-142, 1993.

[Khardon, 1998] R. Khardon. Learning first order universal Horn expressions. In *Proceedings of the Conference on Computational Learning Theory,* pages 154-165, Madison, WI, 1998. ACM Press.

[Kietz and Dzeroski, 1994] J. Kietz and S. Dzeroski. Inductive logic programming and learnability. *SIGART Bulletin,* 5(l):22-32, 1994.

[Krymolovsky and Roth, 1998] Y. Krymolovsky and D. Roth. Incorporating knowledge in natural language learning: A case study. In *COLING-ACL 98 workshop on the Usage of WordNet in Natural Language Processing Systems,* Aug 1998.

[Lavrac and Dzeroski, 1994] N. Lavrac and D. Dzeroski. *Inductive Logic Programming: Techniques and Applications,* Ellis Horwood, London, 1994.

{Littlestone, 1988] N. Eittlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning,* 2:285-318, 1988.

[Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming,* 20:629-679, 1994.

[Munoz *et al,,* 1999] M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. A learning approach to shallow parsing. Technical Report UIUCDCS-R-99-2087, UIUC Computer Science Department, April 1999.

[Page and Prisch, 1992] D. Page and A. Frisch. Generalization and learnability: A study of constrianed atoms. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, 1992.

[Pereira and Shieber, 1987] F. Pereira and S. Shieber. *Prolog and natural-language analysis.* Stanford : Center for the Study of Language and Information, 1987.

[Plotkin, 1970] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5,* pages 153-163. American Elsevier, 1970.

[Reddy and Tadepalli, 1998] C. Reddy and P. Tadepalli. Learning first order acyclic Horn programs from entailment. In *International Conference on Inductive Logic Programming,* pages 23-37, Madison, WI, 1998. Springer. LNAI 1446.

[Rosen, 1999] J. Rosen. Scaling up context sensitive text correction. Master's thesis, UIUC, Department of Computer Science, May 1999.

[Roth and Zelenko, 1998] D. Roth and D. Zelenko. Part of speech tagging using a network of linear separators. In *COLING-ACL 98, The 17th International Conference on Computational Linguistics,* pages 1136-1142, 1998.

[Roth, 1998] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the National Conference on Artificial Intelligence,* pages 806-813, 1998.

[Valiant, 1994] L. G. Valiant. *Circuits of the Mind.* Oxford University Press, November 1994.

[Valiant, 1998a] L. G. Valiant. Neuroidal architecture for cognitive computation. In *ICALP'98, The International Colloquium on Automata, Languages, and Programming,* pages 642-669. Springer-verlag, 1998. Lecture notes in Computer Science, vol. 1443.

[Valiant, 1998b] L. G. Valiant. Projection learning. In *Proceedings of the Conference on Computational Learning Theory,* pages 287-293, 1998.

[Valiant, 1999] L. G. Valiant. Robust logics. In *Proceedings of the Annual ACM Symp. on the Theory of Computing,* 1999. To Appear.