Grammar Inference, Automata Induction, and Language Acquisition

Rajesh Parekh
Allstate Research and Planning Center
321 Middlefield Road, Menlo Park CA 94025
e-mail: *rpare@allstate.com*

Vasant Honavar*
Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames IA 50011-1040. U.S.A.
e-mail: *honavar@cs.iastate.edu*

Abstract

The natural language learning problem has attracted the attention of researchers for several decades. Computational and formal models of language acquisition have provided some preliminary, yet promising insights of how children learn the language of their community. Further, these formal models also provide an operational framework for the numerous practical applications of language learning. We will survey some of the key results in formal language learning. In particular, we will discuss the prominent computational approaches for learning different classes of formal languages and discuss how these fit in the broad context of natural language learning.

1 Introduction

The problem of how children acquire and use the language of the community is a fundamental problem that has attracted the attention of researchers for several decades. A plausible theory of language acquisition would have to account for a wide range of empirical observations. For example, it would have to explain how children are able to acquire

*This research was partially supported by grants from the National Science Foundation (IRI-9409580 and IRI-9643299) and the John Deere Foundation to Vasant Honavar.

a language on the basis of a finite number sentences that they encounter during their formative years. Consequently, natural language acquisition has been, and continues to be, a major focus of research [48, 75]. The past decade has seen significant theoretical as well as experimental advances in the study of natural language acquisition. Examples of some current developments include *bootstrapping hypotheses* and *constraint-based theories* [9], *optimality theory* [80, 89], and *neural theory of language* [27, 28]. Empirical evidence from these results argues in favor of language learnability to address some of the key problems encountered in child language acquisition.

Besides obtaining a better understanding of natural language acquisition, interest in studying formal models of language learning stems also from the numerous practical applications of language learning by machines. Research in *instructible robots* [19] and *intelligent software agents and conversational interfaces* [6] is geared towards the design of agents that can understand and execute verbal instructions given in a natural language (such as English) or some restricted subset of a natural language. For example, one might describe the steps needed for cooking dinner in terms of a sequence of instructions: *"Open the microwave door"* - *"Insert the packaged food"* - *"Set the controls"* - *"Push the cook button"* and so on. Further, the human may want to summarize this entire sequence of actions by the phrase *"Cook dinner"*. The robot should learn the above sequence of actions and identify it with the phrase *"Cook dinner"* [19]. A satisfactory solution to this problem will necessarily have to build on recent advances in a number of areas in artificial intelligence including natural language processing, machine learning, machine perception, robotics, planning, knowledge representation, and reasoning.

Formal language models are extensively used in syntactic or linguistic pattern classification systems [29, 30]. The structural inter-relationships among the linguistic pattern attributes are easily captured by representing the patterns as strings (a collection of syntactic symbols). Learning a set of rules for generating the strings that belong to the same class or category (language) provides for a simple yet adequate syntactic or linguistic classification system. Linguistic pattern recognition has been put to several practical uses including *speech recognition*, *discovery of patterns in biosequences*, *image segmentation*, *interpretation of ECG*, *handwriting recognition*, *recognition of seismic signals* and the like [8, 30, 57]. The issues and practical difficulties associated with formal language learning models can provide useful insights for the development of language understanding systems. Several key questions in natural language learning such as the role of prior knowledge, the types of input available to the learner, and the impact of semantic information on learning the syntax of a language can possibly be answered by studying formal models of language acquisition.

Research in language acquisition has benefited from advances in several disciplines including *cognitive psychology* [9], *linguistics* [15, 16, 75], *theoretical computer science* [42, 55], *computational learning theory* [45, 63], *artificial intelligence* [84], *machine learning* [48, 61], and *pattern recognition* [30, 57]. Psychological studies of language acquisition have explored the development of the children's cognitive faculties, the nature of stimuli available to them, the types of word combinations they form, and related questions.

Studies in linguistics attempt (among other things), to study similarities and differences among the different languages in order to characterize language properties that are universal (i.e., common to all languages). Theoretical computer science has taken a formal view of languages and constructed a language hierarchy based on representational power of the different formal languages. Research in computational learning theory has explored language learnability and has provided insights in to questions such as: whether it is possible, at least in principle, to learn a language from a set of sentences of the language; whether there are classes of languages which can be efficiently learned by computers, etc. Artificial intelligence has been concerned with the design of languages for communication among intelligent agents, deployment of instructible robots, and development of natural language understanding systems. Studies in machine learning and pattern recognition have contributed to the development and application of algorithms for language learning.

The language learning problem concerns the acquisition of the syntax or the grammar (i.e., rules for generating and recognizing valid sentences in the language) and the semantics (i.e., the underlying meaning conveyed by each sentence) of a target language. Thus far, most of the work on computational approaches to language acquisition has focused on learning the syntax. This component of language acquisition is generally referred to as *grammatical inference*. In order to make the problem of syntax acquisition well-defined, it is necessary to choose an appropriate class of grammars (or equivalently languages) which is guaranteed to contain the unknown grammar. The classes of grammars such as *regular* or *context free* grammars that belong to the Chomsky hierarchy of formal grammars [13, 42] are often used to model the target grammar. The methods for grammar inference typically identify an unknown grammar (or an approximation of it) from a set of candidate hypotheses (e.g., the set of regular grammars defined over some alphabet). These algorithms are based on the availability of different types of information such as positive examples, negative examples, the presence of a knowledgeable teacher who answers queries and provides hints, and the availability of additional domain specific prior knowledge.

The remainder of this chapter is organized as follows: The necessary terminology and definitions concerned with language learning are introduced in section 2. A variety of approaches for learning regular grammars are studied in section 3. Algorithms for learning stochastic regular grammars and hidden markov models are described in section 4. These are motivated by the need for robust methods for dealing with noisy learning scenarios (e.g., when examples are occasionally mislabeled) and the unavailability (in several practical application domains) of suitably labeled negative examples. The limited representational power of regular grammars has prompted researchers to explore methods for inference of more general classes of grammars such as context free grammars. Several approaches for learning context free grammars are reviewed in section 5. Section 6 concludes with a brief discussion of several interesting issues in natural language learning that have been raised by recent research in cognitive psychology, linguistics, and related areas.

2 Languages, Grammars, and Automata

Syntax acquisition is an important component of systems that learn to understand natural language. The syntax of a language is typically represented by grammar rules using which one may generate legal (grammatically correct) sentences of the language, or equivalently, decide whether a given sentence belongs to the language or not. *Grammar Inference* provides a tool for automatic acquisition of syntax. It is defined as the process of learning a target grammar from a set of labeled sentences (i.e., from examples of grammatically correct and, if available, examples of grammatically incorrect sentences). Formal languages were initially used to model natural languages in order to better understand the process of natural language learning [13]. Formal languages are associated with recognizing (or equivalently, generating) devices called *automata*. The task of grammar inference is often modeled as the task of automata induction wherein an appropriate automaton for the target language is learned from a set of labeled examples.

A *formal grammar* is a 4-tuple $G = (V_N, V_T, P, S)$ where V_N is the set of non-terminals, V_T is the set of terminals, P is the set of production rules for generating valid sentences of the language, and $S \in V_N$ is a special symbol called the start symbol. The production rules are of the form $\alpha \longrightarrow \beta$ where α, β are sentences over the alphabet $(V_N \cup V_T)$ and α contains at least one non-terminal. Valid sentences of the language are sequences of terminal symbols V_T and are obtained by repeatedly applying the production rules as shown below.

Example

Consider a simple grammar for declarative English sentences in Fig. 1[1]. These rules can be used to generate the sentence *The boy saw a ferocious tiger* as shown in Fig. 2. Note that at each step the left most non-terminal is matched with the left hand sides of the rules and is replaced by the right hand side of one of the matching rules.

Fig 1 comes here (Grammar for Declarative English Sentences)

Fig 2 comes here (Generation of the sentence)

The *language* of a grammar is the set of all valid sentences that can be generated using rules of the grammar. Chomsky proposed a hierarchy of formal language grammars based on the types of restrictions placed on the production rules [13, 42]. The simplest class of grammars in this hierarchy is the class of *regular grammars* which are recognized by *finite state automata* (see section 3.1). Regular grammars are limited in their representational power in that they cannot be used to describe languages such as *palindromes* whose sentences would require the grammar to memorize a certain sequence within the sentence

[1]Note that a rule of the form $\alpha \longrightarrow \beta|\gamma$ is a concise way of writing two separate rules $\alpha \longrightarrow \beta$ and $\alpha \longrightarrow \gamma$.

in order to determine the validity of the sentence. This limitation is overcome in the next class of grammars in the hierarchy, the *context free* grammars. Context free grammars are recognized by *pushdown automata* which are simply *finite state automata* augmented with a *pushdown stack*. Context free grammars are adequate for several practical natural language modeling tasks [19, 73]. Context sensitive grammars represent the next level of grammars in the hierarchy and unrestricted grammars (which place no restriction on the form of the production rules) complete the formal language hierarchy.

3 Regular Grammar Inference

The regular grammar inference task is defined as follows: Given a finite set of positive examples (sentences belonging to the language of the target grammar) and a finite (possibly empty) set of negative examples (sentences that do not belong to the language of the target grammar), identify a regular grammar G^* that is equivalent[2] to the target grammar G. An important design choice concerns the representation of the target. Regular grammars can be equivalently represented by a set of production rules, a regular expression, a deterministic finite state automaton (DFA), or a non-deterministic finite state automaton (NFA). Most work in regular grammar inference has chosen DFA as the representation of the target regular grammar. This is attributed to the following characteristics of DFA: they are simple and easy to understand; there exists a unique minimum state DFA corresponding to any regular grammar; there exist efficient (polynomial time) algorithms for several operations (such as minimization of a DFA, determining the equivalence of two DFA, determining whether the language of one DFA is a superset of the language of another DFA and the like) that are frequently used in regular grammar inference methods [42].

3.1 Finite State Automata

A *deterministic* finite state automaton (DFA) is a quintuple $A = (Q, \delta, \Sigma, q_0, F)$ where, Q is a finite set of states, Σ is the finite alphabet, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and δ is the transition function: $Q \times \Sigma \longrightarrow Q$. $\delta(q, a)$ denotes the state reached when the DFA in state q reads the input letter a. A state $d_0 \in Q$ such that $\forall a \in \Sigma,\ \delta(d_0, a) = d_0$ is called a *dead* state. The extension of δ to handle input strings (i.e., concatenations of symbols in Σ) is standard and is denoted by δ^*. $\delta^*(q, \alpha)$ denotes the state reached from q upon reading the string α. A string α is said to be accepted by a DFA if $\delta^*(q_0, \alpha) \in F$. Strings accepted by the DFA are said to be *positive examples* (or valid sentences) of the language of the DFA. The set of all strings accepted by a DFA A is its language, $L(A)$. Correspondingly, strings not accepted by the DFA are called *negative examples* (or invalid sentences) of the language of the DFA. The language of a DFA is called a *regular language*. DFA can be conveniently represented using state

[2]Two grammars G_1 and G_2 are equivalent if their languages are exactly the same.

transition diagrams. Fig. 3 shows the state transition diagram for a sample DFA. The start state q_0 is indicated by the symbol $>$ attached to it. Accepting states are denoted using concentric circles. The state transition $\delta(q_i, a) = q_j$ for any letter $a \in \Sigma$ is depicted by an arrow labeled by the letter a from the state q_i to the state q_j.

Figure 1 comes here (DFA)

A *non-deterministic* finite automaton (NFA) is defined just like the DFA except that the transition function δ defines a mapping from $Q \times \Sigma \longrightarrow 2^Q$ i.e., there can potentially be more than one transition out of a state on the same letter of the alphabet. In general, a finite state automaton (FSA) refers to either a DFA or a NFA. Given any FSA (A'), there exists a minimum state DFA also called the *canonical DFA* (A) such that $L(A) = L(A')$. Without loss of generality we will assume that the target DFA being learned is a canonical DFA. A labeled example $(\alpha,\ c(\alpha))$ for A is such that $\alpha \in \Sigma^*$ and $c(\alpha) = +$ if $\alpha \in L(A)$ (i.e., α is a positive example) or $c(\alpha) = -$ if $\alpha \notin L(A)$ (i.e., α is a negative example). Thus, $(a, -)$, $(b, +)$, $(aa, +)$, $(aaab, -)$, and $(aaaa, +)$ are labeled examples for the DFA of Fig. 3. Given a set of positive examples denoted by S^+ and a set of negative examples denoted by S^-, we say that A is consistent with the *sample* $S = S^+ \cup S^-$ if it accepts all positive examples and rejects all negative examples.

3.2 Results in Regular Grammar Inference

Regular grammar inference is a hard problem in that regular grammars cannot be correctly identified from positive examples alone [36]. Further, it has been shown that there exists no efficient learning algorithm for identifying the minimum state DFA that is consistent with an arbitrary set of positive and negative examples [37]. Efficient algorithms for identification of DFA assume that additional information is provided to the learner. This information is typically in the form of a set of examples S that satisfies certain properties or a knowledgeable teacher's responses to queries posed by the learner. Trakhtenbrot and Barzdin have proposed an algorithm to learn the smallest DFA consistent with *complete labeled sample* i.e., a sample that includes all sentences up to a particular length with the corresponding label that indicates whether the sentence is a positive example or a negative example [92]. Oncina and Garcia have defined a *characteristic set* of examples which is a representative set that includes information about the states and transitions of the target DFA. They showed that it is possible to exactly identify the target DFA from a characteristic sample [65]. Angluin has described the use of a *minimally adequate teacher* to guide the learner in the identification of the target DFA. A minimally adequate teacher is capable of answering *membership queries* of the form "Does this sentence belong to the target language?" and *equivalence queries* of the form "Is this DFA equivalent to the target?" Using labeled examples together with membership and equivalence queries it is possible to correctly identify the target DFA [3].

3.3 Search Space

Regular grammar inference can be formulated as a search problem in the space of all finite state automata (FSA). Clearly, the space of all FSA is infinite. One way to restrict the search space is to map a set of positive examples of the target DFA to a *lattice* of FSA which is constructed as follows [67]. Initially, the set of positive examples (S^+) is used to define a *prefix tree automaton* (PTA). The PTA is a DFA with separate paths (modulo common prefixes) from the start state leading to an accepting state for each string in S^+. The PTA accepts only the sentences in S^+ i.e., the language of the PTA is the set S^+ itself. For example, the PTA corresponding to a set $S^+ = \{b, aa, aaaa\}$ of positive examples is shown in Fig. 4.

Figure 4 comes here (Prefix Tree Automaton)

The lattice is now defined set of all partitions of the set of states of the PTA together with a relation that establishes a *partial order* on the elements. Each element of the lattice (i.e., an individual partition of the set of states of the PTA) is called a *quotient automaton* and is constructed from the PTA by merging together the states that belong to the same block of the partition. For example, the quotient automaton obtained by merging the states q_0 and q_1 of the PTA is shown in Fig. 5. The corresponding partition of the set of states of the PTA is $\{\{0, 1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. The quotient automaton is *more general than* its parent automaton in that, its language is a superset of the language of the parent.

Figure 5 comes here (Quotient Automaton)

The individual partitions in the lattice are partially ordered by the *grammar covers* relationship. One partition is said to *cover* another if the former is obtained by merging together two or more blocks of the latter. For example, the partition $\{\{0, 1, 2\}, \{3\}, \{4, 5\}\}$ covers the partition $\{\{0, 1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. The important consequence of the grammar covers property is that if a partition covers another then the language of the FSA representing the former is a superset of the language of the FSA representing the latter. The PTA is the *most specific* element of the lattice. The universal DFA that is obtained by merging all the states of the PTA into a single state is the *most general* element of the lattice. Given the PTA of Fig. 4, the partition corresponding to the *universal DFA* is $\{\{0, 1, 2, 3, 4, 5\}\}$. The modified search space, though finite, is still exponentially large in terms of the number of states of the PTA. Explicit enumeration of the entire search space is thus not practical. Typical search procedures start with either the PTA or the universal DFA and use search operators such as *state merging* and *state splitting* to generate new elements within the search space.

By mapping the set of positive examples to a lattice of FSA we demonstrated how we could restrict the search space to be finite. How can we guarantee that the target DFA lies within this restricted search space? It can be shown that if the set of positive examples provided to the learner is a *structurally complete* set then the lattice constructed above is guaranteed to contain the minimum state DFA equivalent to the target [24, 67, 69]. A structurally complete set for the target DFA is a set of positive examples (S^+) such that for each transition of the target DFA there is at least one string in S^+ that covers the transition and for each accepting state of the target DFA there is at least one string in S^+ that visits the accepting state. For example, the set $S^+ = \{b, aa, aaaa\}$ is structurally complete with respect to the DFA in Fig. 3.

3.4 Search Strategy

Even though the lattice is guaranteed to contain the target DFA, the size of the lattice is prohibitively large to be efficiently searched using an uninformed strategy like *breadth first* search. Several methods that avail of some additional information provided to the learner have been designed to search the lattice for the target DFA. In the following sections we summarize three different search methods.

3.4.1 Bi-directional Search using Membership Queries

Parekh and Honavar have proposed a bi-directional search strategy based on the *version space* approach [69, 70]. Their method assumes that a structurally complete set of positive examples (S^+) is provided to the learner at the start and a knowledgeable teacher is available to answer the membership queries posed by the learner. The lattice of finite state automata defined above is implicitly represented using a *version space* [60]. The version space contains all the elements of the lattice that are consistent with the labeled examples and the membership queries. It is represented using two sets of finite state automata: a set $\mathcal{S}$ of *most specific* FSA which is initialized to the PTA and a set $\mathcal{G}$ of *most general* FSA which is initialized to the universal DFA obtained by merging all the states of the PTA. Thus the initial version space implicitly includes all the elements of the lattice. It is gradually refined by eliminating those elements of the lattice that are found to be inconsistent with the observed data. The version space search proceeds as follows: At each step two automata (one each from the sets $\mathcal{S}$ and $\mathcal{G}$ respectively) are selected and compared for equivalence. If they are not equivalent, then the shortest string accepted by one automaton but not the other is posed as a membership query to the teacher. The refinement of the sets $\mathcal{S}$ and $\mathcal{G}$ is guided by the teacher's response to the membership query. For example, if the query string is labeled as a negative example by the teacher then FSA in $\mathcal{S}$ that accept the negative example are eliminated from $\mathcal{S}$. Further, all FSA of $\mathcal{G}$ that accept the negative example are progressively specialized by splitting their states until the resulting FSA do not accept the negative example. Thus, effectively all automata accepting the negative example are eliminated from the version space. Membership queries serve to progressively *generalize* the elements of $\mathcal{S}$ and *specialize* the elements of $\mathcal{G}$ thereby moving the two frontiers of the lattice represented by $\mathcal{S}$ and $\mathcal{G}$ closer to each other. At all times the two frontiers implicitly encompass the elements of the lattice that are consistent with the data observed by the learner. Convergence to the target is guaranteed and is attained when $\mathcal{S} = \mathcal{G}$. The essence of this bi-directional search of the lattice is captured in Fig. 6. Although this algorithm is guaranteed to converge to the target grammar, the speed of convergence often depends on the actual queries posed. Some query strings have the potential for eliminating huge chunks of the search space. The worst-case time complexity of this method is exponential in the size of the PTA.

Figure 6 comes here (Bi-directional search)

3.4.2 Randomized Search

Genetic algorithms offer an attractive framework for randomized search in large hypotheses spaces [40]. A typical genetic search involves evolving a randomly generated set of individuals (from the hypothesis space) based on the *survival of the fittest* principle of *Darwinian evolution*. A population of randomly selected elements of the hypothesis space is evolved over a period of time. A suitable representation scheme is chosen to encode the elements of the hypothesis space as *chromosomes*. A *fitness function* is defined to evaluate the quality of the solution represented by the individual chromosomes. The search progresses over several generations. In each generation a group of fittest (as determined by the fitness function) individuals is selected for genetic reproduction which is represented by suitably defined operators (such as *mutation* and *crossover*). The genetic operators serve to identify new and potentially interesting areas of the hypothesis space. The offspring are added to the population and the evolution continues. Over a period of time the individuals tend to converge towards high fitness values (i.e., reasonably good solutions).

Dupont has proposed a framework for regular grammar inference using genetic algorithms [24]. The learner is given a set $S = S^+ \cup S^-$ of labeled examples. A PTA is constructed from the set S^+ as explained in section 3.3. The initial population comprises of a random selection of elements from the set of partitions of the set of states of the PTA. Each element of the initial population is thus a quotient automaton belonging to the lattice of FSA constructed from the PTA. The fitness assigned to each quotient FSA is a function of two variables: the number of states of the FSA and the number sentences in S^- that are mis-classified by the FSA. Individuals with fewer states that make fewer errors are assigned higher fitness. After each generation a sub-population of individuals is randomly selected for reproduction based on their fitness values. Two genetic operators *structural mutation* and *structural crossover* (designed specifically for the problem on hand) are applied to a sub-population to produce offspring. Structural mutation involves randomly selecting an element from one of the existing blocks of the partition and randomly assigning it to one of the other blocks or creating a new block. Thus, $\{\{0, 1, \mathbf{3}, 5\}, \{2\}, \{4\}\} \longrightarrow \{\{0, 1, 5\}, \{2, \mathbf{3}\}, \{4\}\}$ is an example of structural mutation. In structural crossover one block is chosen at random from each parent and both offspring inherit the block obtained by merging the two selected blocks. The remaining blocks for the offspring are obtained by taking the difference[3] of the blocks of the partitions corresponding to the parents that were not chosen above with the common block inherited by both offspring. For example, if the parents are represented by the following partitions $\{\{0\}, \{\mathbf{1}, \mathbf{4}\}, \{2, 3, 5\}\}$ and $\{\{0\}, \{\mathbf{1}, \mathbf{3}\}, \{2\}, \{4\}, \{5\}\}$ then a single application of the structural crossover might result in the following offspring $\{\{0\}, \{\mathbf{1}, \mathbf{3}, \mathbf{4}\}, \{2, 5\}\}$ and $\{\{0\}, \{\mathbf{1}, \mathbf{3}, \mathbf{4}\}, \{2\}, \{5\}\}$. Here the blocks $\{1, 4\}$ and $\{1, 3\}$ are first merged to obtain the common block inherited by both offspring. The difference of the block $\{0\}$ with $\{1, 3, 4\}$ is $\{0\}$, the difference of the block $\{2, 3, 5\}$ with $\{1, 3, 4\}$ is $\{2, 5\}$ and so on. The offspring produced by the genetic reproduction are valid partitions belonging to the lattice. These are added to the original population. A *fitness-proportionate* selection scheme that assigns high probability to individuals with higher fitness is used to randomly select individuals for the next generation. After a pre-specified number of generations the fittest individual from the population is selected and returned by the algorithm as the inferred FSA. Although, this approach is not guaranteed to converge to the target DFA, empirical results have shown that this evolutionary search does result in identifying sufficiently small DFA that make reasonably fewer errors on the set S^-.

3.4.3 Ordered Depth-First Search with Backtracking

The *regular positive and negative inference* (RPNI) algorithm performs an ordered *depth-first* search of the lattice guided by the set of negative examples S^- and in polynomial time identifies a DFA consistent with a given sample $S = S^+ \cup S^-$ [65]. Further, if S happens to be a superset of a *characteristic set* for the target DFA then the algorithm is guaranteed to return a canonical representation of the target DFA. A characteristic set of examples $S = S^+ \cup S^-$ is such that S^+ is *structurally complete* with respect to the target and S^- prevents any two states of the PTA of S^+ that are not equivalent to each other from being merged together (see [65] for a precise definition).

The algorithm first constructs a PTA for S^+. The states of the PTA are numbered in standard order as follows: The set of strings that lead from the start state to each individual state of the PTA is determined. The strings are sorted in lexicographic order (i.e., λ, a, b, aa, ab, $\ldots$). Each state is numbered based on the position of the corresponding string in the sorted list. The states of the PTA shown in Fig. 4 are numbered in standard order. The algorithm initializes the PTA to be the current solution and systematically merges the states of the PTA to identify a more general solution that is consistent with S.

The states of the PTA are systematically merged using a quadratic loop i.e., at each step i, the algorithm attempts to merge the state q_i with the states q_0, q_1, $\ldots$, q_{i-1} in order. If the quotient automaton obtained by merging any two states does not accept any example belonging to S^- then the quotient automaton is treated as the current target

[3]The standard set difference operator is used here.

and the search for a more general solution is continued with the state q_{i+1}. If $||S^+||$ and $||S^-||$ denote the sums of lengths of each example in S^+ and S^- respectively then it can be shown that the time complexity of the RPNI algorithm is $\mathcal{O}((||S^+|| + ||S^-||) \cdot ||S^+||^2)$. The interested reader is referred to [23, 65] for a detailed description of the algorithm.

Example

We demonstrate a few steps in the execution of the RPNI algorithm on the task of learning the DFA of Fig. 3. Assume that we are given the *characteristic* sample $S = S^+ \cup S^-$ where $S^+ = \{b, aa, aaaa\}$ and $S^- = \{\lambda, a, aaa, baa\}$. The PTA is depicted in Fig. 4 where the states are numbered in the standard order. The initial partition is $\{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. The blocks 1 and 0 of the partition are merged. The quotient shown in Fig. 7 accepts the negative example a and thus the merge is rejected. Next the blocks 2 and 0 of the partition are merged. The quotient automaton shown in Fig. 8 accepts the negative example λ and thus even this merge is rejected. Continuing the merging in this manner it can be shown that the algorithm returns the partition $\{\{0\}, \{1, 4\}, \{2\}, \{3, 5\}\}$ which is exactly the target DFA we are trying to learn (Fig. 3).

Figure 7 comes here (RPNI eg 1)

Figure 8 comes here (RPNI eg 2)

3.5 The L^* Algorithm

The L^* algorithm infers a minimum state DFA corresponding to the target with the help of a *minimally adequate teacher* [3]. Unlike the approaches described so far, the L^* algorithm does not search a lattice of FSA. Instead, it constructs a hypothesis DFA by posing membership queries to the teacher. The learner maintains a table where the rows correspond to the states of the hypothesis DFA and the columns correspond to suffix strings that distinguish between pairs of distinct states of the hypothesis. Individual states are labeled by the strings that lead from the start state to the states themselves. The start state is labeled by the string λ. The states reached from the start state by reading one letter of the alphabet are labeled by a and b respectively, and so on. Thus, for the DFA in Fig. 3, Q_0 is labeled by the empty string λ, Q_1 is labeled by a, Q_2 is labeled by b, and Q_3 is labeled by aa. For each state the column labels indicate the state that would be reached from that state after reading the string corresponding to the column label. Membership queries are posed on the string obtained by concatenating the row and column labels. The first column is labeled as λ. The queries $\lambda \cdot \lambda$, $a \cdot \lambda$, etc. ask whether the state Q_0 is an accepting state, Q_1 is an accepting state, and so on. The cell representing the intersection of a row r and a column c is marked either 1 or 0 depending on whether the string $r \cdot c$ is accepted or not. Two states are presumed equivalent if all the entries for their corresponding rows are identical. If two states (say q_i and q_j) are presumed equivalent but the states reached after reading a single letter of the alphabet from these states are not equivalent (i.e., the rows representing these adjacent states are not identical) then the states q_i and q_j cannot be equivalent. A column (representing a distinguishing suffix) is then augmented to the table to distinguish the two states. When pairs of states in the hypothesis DFA cannot be distinguished any further the learner poses an equivalence query asking whether the current hypothesis is equivalent to the target. If the teacher answers *yes* the algorithm terminates and returns the current hypothesis. However, if the teacher answers *no* it also provides a counterexample that is accepted by the target DFA but not by the current hypothesis or vice-versa. The counterexample together with all its prefixes is incorporated into the table as new rows. Intuitively, this modification of the table represents adding new states to the current hypothesis. The learner modifies the hypothesis using the counterexample and additional membership queries and poses another equivalence query. This interaction between the learner and the teacher continues until the teacher's answer to an equivalence query is *yes*. The algorithm runs in time polynomial in the size of canonical representation of the target DFA and the length of the longest counterexample provided by the teacher and is guaranteed to converge to the unknown target.

The L^* algorithm assumes that the learner has the capacity to *reset* the DFA to the start state before posing each membership query. In other words, the teacher's response to the membership query indicates whether or not the DFA starting in state q_0 accepts the query string. This assumption might not be realistic in certain situations. For example, consider a robot trying to explore its environment. This environment may be modeled as a finite state automaton with the different states corresponding to the different situations the robot might find itself in and the transitions corresponding to the different actions taken by the robot in each situation. Once the robot has made a sequence of moves it may find itself in a particular state (say facing a barrier). However, the robot has no way of knowing from where it started or of retracing its steps to the start state. The robot has to treat its current state as the start state and explore the environment further. Rivest and Schapire have proposed a method based on *homing sequences* to learn the target DFA in these situations [83]. If it is assumed that each state of the DFA has an output (the output could simply be 1 for an accepting state and 0 for a non-accepting state) then a homing sequence is defined as a sentence whose output always uniquely identifies the final state the DFA is in even if the state the DFA started from before processing the sentence is unknown. Rivest and Schapire's algorithm runs N copies of the L^* algorithm (one for each of the N states of the target DFA) in parallel and using homing sequences over comes the limitation that the start state of the target DFA is not known.

3.6 Connectionist Methods

Artificial neural networks (ANN) or connectionist networks are biologically inspired models of computation. ANN are networks of elementary processing units that are interconnected by trainable connections. Each processing unit (or *neuron*) computes an elementary function of its inputs. The connections are responsible for transmitting signals between the neurons. Each connection has an associated *strength* or *weight* which prescribes the magnitude by which the signal it carries is amplified or suppressed. ANN are typically represented by weighted directed graphs. The nodes of the graph correspond to neurons, the edges refer to the connections, and the weights on the edges indicate the strength of the corresponding connections. Each neuron performs its computation locally on the inputs it receives from the neurons to which it is connected. These computations are independent of each other. Thus, ANN have potential for massive parallelism. Typical neural network architectures organize neurons in layers. The *input* layer neurons allow for external signals to be input to the network. The network's output is read from the neurons at the *output* layer. Additionally, a network might have one or more *hidden* layers of neurons to facilitate learning.

ANN have received considerable attention for their ability to successfully model language learning tasks. The advantages of connectionist methods over the symbolic techniques described above include robust behavior in the presence of limited amount of noise, ability to learn from relatively smaller training sets, and scalability to larger problem sizes [51, 52]. Most connectionist language learning approaches specify the learning task in classical terms (for example using an automaton to parse grammatically correct sentences of the language) and then implement these using a suitable neural network architecture such as *recurrent neural networks* (RNN) [62]. A variety of RNN architectures have been investigated for learning grammars from a set of positive and negative examples of the target language (see for example [17, 18, 25, 26, 31, 34, 32, 58, 78, 85, 97]). In the following section we describe the second order recurrent neural network architecture due to Giles *et al* [31].

3.6.1 Second Order RNN for Regular Grammar Inference

Recurrent neural networks have feedback connections from the output neurons back to the input that give them the ability to process temporal sequences of arbitrary length. The second order RNN is depicted in Fig. 9. The network has N recurrent state neurons $S_0, S_1, \ldots, S_{N-1}$, L non-recurrent input neurons $I_0, I_1, \ldots, I_{L-1}$ and $N^2 \times L$ recurrent real valued connection weights W_{ijk}. The vector of output values of the state neurons is called a *state* vector $\mathbf{S}$. $\mathbf{S}$ is fed back to the network via synchronous time delay lines i.e., at time $t+1$ the network's output at time t is fed to the network along with the input. The state neuron S_0 is arbitrarily designated as the output neuron and is periodically sampled to read the network's output. The weight W_{ijk} modifies a product of the state neuron S_j and the input neuron I_k. For example, W_{211} would correspond to the weight connecting the state neuron S_2 with the unit computing the product of the state neuron S_1 with the input neuron I_1. This quadratic form directly represents the state transition function $\delta : Q \times \Sigma \longrightarrow Q$ and is responsible for the name *second order* RNN.

Figure 9 comes here (RNN diagram)

Typically, the number of input neurons L is chosen to be $|\Sigma|+1$ with one neuron per input symbol and the extra neuron used to denote the end of an input string. The input is fed to the network one character at a time. Thus, the string 011 would be presented over 4 time steps with the activations of the input neurons $I_0, I_1,$ and I_2 respectively being $< 100 >$, $< 010 >$, $< 010 >$, and $< 001 >$ at $t = 0, 1, 2,$ and 3 respectively. The choice of the number of state neurons N is a design parameter. Typically N is chosen between 3 and 5. Recently Siegelmann and Giles have derived a bound on the size of a continuous-valued recurrent network that is needed to correctly classify a given regular set [86]. This bound is easily computable and is tighter than the size of the canonical DFA that is commonly used as the upper bound.

The net input activation $y_{i+1}^{(t+1)}$ of the state neuron S_i is defined by the following expression.

$$y_i^{(t+1)} = \sum_{j=1}^{N}\sum_{k=1}^{L} W_{ijk} S_j^{(t)} I_k^{(t)} + \theta_i$$

where θ_i is the threshold of unit i. Note that the *mult* units in Fig. 9 compute the product $S_j^{(t)} I_k^{(t)}$. The state neurons typically implement the *sigmoid activation function* of the net input.

$$O_i^{(t+1)} = \frac{1}{1+e^{-y_i^{(t+1)}}}$$

The output of the state neuron S_0 is observed after the complete input string is presented. For a suitably chosen threshold ϵ (where $0 < \epsilon < 1/2$), if the output is greater than $1-\epsilon$ then the string is declared a positive example and if the output is less than ϵ then the string is declared a negative example.

The RNN is trained using an iterative *gradient descent* strategy to minimize a selected error function. The squared error is a typical choice for the error function. The error E_p for the sentence p is defined as follows:

$$E_p = \frac{1}{2}(T_p - S_0^{(f)})^2$$

where T_p is the desired label of sentence p (i.e., 1 or 0 depending on whether p is accepted by the target or not) and $S_0^{(f)}$ is the output of S_0 after the entire sentence is presented. During training an entire input pattern is presented to the network and the final state of the network is determined. In case of a misclassification the network's weights are modified by an amount proportional to the *negative gradient* of the error function. The weight update equation is derived as follows:

$$\begin{aligned}\Delta W_{ijk} &= -\eta \frac{\partial E_p}{\partial W_{ijk}} \\ &= -\eta (T_p - S_0^{(f)}) \frac{\partial S_0^{(f)}}{\partial W_{ijk}}\end{aligned}$$

The results of experiments on the Tomita benchmark for regular grammars [91] showed that second order neural networks converged fairly quickly to compact representations of the target finite state automata and generalized reasonably well on strings not belonging to the target language [34].

3.6.2 Extracting Finite State Automata from Trained RNN

One significant disadvantage of connectionist methods is that the learned models are not as transparent as is the case with symbolic approaches. Several researchers have studied methods for extracting finite state automata from a trained RNN [20, 33, 64, 97]. It is observed that RNN develop an internal state representation in the form of clusters in the activation space of the recurrent state neurons. Therefore, a symbolic description of the learned finite state automaton can be extracted from the trained network using clustering techniques. We describe an approach based on partitioning the space of the state neurons (due to [34]) and a hybrid method combining symbolic and connectionist approaches (due to [1]) for extracting state information from trained RNN.

Giles *et al*'s method [34] divides the output range $[0, 1]$ of each recurrent state neuron into q (typically $q = 2$) equal sized partitions thereby mapping the outputs of the N state neurons to a set of q^N states of the FSA. The partition P^λ corresponding to the RNN's response to the empty string (i.e., the end of string symbol) is treated as the start state of the FSA. Next the input symbol 0 is fed to the RNN and the output partition P^0 is treated as the state obtained by reading the symbol 0 in the start state P^λ. The partitions in response to the various input strings 1, 00, 01, ... are recorded and the learned FSA is reconstructed. Clearly, the number of states of the learned FSA can be no more than q^N. The extracted DFA is *minimized* (see [42] for a description of the state minimization procedure). Experimental results show that the extracted DFA approximates the target DFA fairly closely.

Alquezar and Sanfeliu proposed a method based on *hierarchical clustering* to extract a FSA from the trained RNN [1]. Their algorithm constructs a *prefix tree automaton* (PTA) from the set of positive training examples (see section 3.3). Simultaneously a set of single-point clusters is initialized to the set of activation vectors of the network's state neurons in response to each symbol of each training string. Note that there is a one-to-one correspondence between the states of the PTA and the initial single-point clusters. The PTA is treated as the initial hypothesis FSA. Hierarchical clustering is used to repeatedly merge the two closest clusters. The states of current hypothesis FSA that correspond to the two clusters are also merged together. If the resulting automaton is inconsistent with the training set (i.e., it accepts a negative example) then the merge is disallowed. Otherwise the resulting automaton is treated as the current hypothesis FSA. The next two closest clusters are then considered for merging and so on. This procedure terminates when all further merging results in the hypothesis FSA being inconsistent with some negative example of the training set.

3.6.3 Using Evolutionary Programming to learn RNN

Most traditional approaches to training RNN for regular grammar inference use an *a-priori* fixed network architecture where the number of state neurons is often selected in an *ad-hoc* manner. An inappropriate choice of the network architecture can potentially prevent the network from efficiently learning the target grammar. Angeline *et al* proposed an evolutionary network induction algorithm for simultaneously acquiring both the network topology and the weight values [2]. Their algorithm called GNARL (*GeNeralized Acquisition of Recurrent Links*) uses *evolutionary programming* techniques which reply on *mutation* as the sole genetic reproduction operator (as against genetic algorithms which tend to rely primarily on the *crossover* operator).

GNARL has a fixed number M of input nodes $I_1, I_2, \ldots, I_M$, a fixed number N of output nodes $O_1, O_2, \ldots, O_N$, and a variable number of hidden nodes (up to a maximum of h_{max}). Implementational constraints such as no links to an input node, no links from an output node, and at most one connection between any two nodes are enforced to ensure that only plausible network architectures are considered. The algorithm initializes a population of randomly generated networks. The number of hidden neurons, number of network connections, the incident neurons for each network connection, and the values of the connection weights are selected uniformly at random for each network in the initial population. In each generation a subset of the networks in the population is selected for genetic mutation based on their scores on a user provided *fitness* function. The *sum squared error* metric is a commonly used fitness measure. GNARL considers two types of mutation operators: *parametric* mutation which alters the values of the connection weights and *structural* mutation which alters the structure of the network (by adding or deleting nodes or links). The mutation operators are designed to retain the created offspring within a specific locus of the parent i.e., care is taken to ensure that the network created as a result of mutation is not too different from its parent in terms of both structure and fitness. Experiments on the Tomita benchmark for regular grammars showed that the GNARL networks consistently exhibited better average test accuracy as compared to the second order recurrent networks described in [95].

3.7 Incremental Approaches

The problem of regular grammar inference is made more tractable by assuming that the set of examples provided to the learner is *structurally complete* (see section 3.4.1) or is a *characteristic* sample (see section 3.4.3). In many practical learning scenarios, a sample that satisfies such properties may not be available to the learner at the outset. Instead, a sequence of labeled examples is provided intermittently and the learner is required to construct an approximation of the target DFA based on the information available to it at that point in time. In such scenarios, an online or incremental model of learning that is guaranteed to eventually converge to the target DFA in the limit is of interest. Particularly, in the case of intelligent autonomous agents, incremental learning offers an attractive framework for characterizing the behavior of the agents [11].

Parekh *et al* have proposed an efficient incremental algorithm for learning regular grammars using membership queries [72]. Their method extends Angluin's *ID* algorithm to an incremental framework. The learning algorithm is intermittently provided with labeled examples and has access to a knowledgeable teacher capable of answering membership queries. Based on the observed examples and the teacher's responses to membership queries, the learner constructs a DFA. As in the case of the L^* algorithm, input strings are used to label the individual states. Membership queries are posed on the strings representing the states and the strings representing the adjacent states (obtained from the current state by reading one letter of the alphabet). A set of suffix strings is constructed to distinguish among non-equivalent states. The current hypothesis is guaranteed to be consistent with all the examples observed thus far. If an additional example provided is inconsistent with the learner's current representation of the target DFA then the learner modifies the current representation suitably to make it consistent with the new example. The algorithm is guaranteed to converge to a minimum state DFA corresponding to the target when the information obtained via labeled examples and membership queries is sufficient to enable the learner to identify the different states and their associated transitions. Further, the time and space complexities of this approach are polynomial in the sum of the lengths of the counterexamples seen by the learner.

Two other incremental algorithms for learning DFA are based only on labeled samples and do not require membership queries. Porat and Feldman's incremental algorithm uses a *complete ordered sample* (i.e., all examples up to a particular length provided in strict lexicographic order) [79]. The algorithm maintains a current hypothesis which is updated when a counterexample is encountered and is guaranteed to converge in the limit. It needs only a finite working storage. However, it is based on an ordered presentation of examples and requires a consistency check with all the previous examples when the hypothesis is modified. An incremental version of the *RPNI* algorithm for regular grammar inference was proposed by Dupont [23]. It is guaranteed to converge to the target DFA when the set of examples seen by the learner includes a *characteristic set* for the target automaton as a subset. The algorithm runs in time that is polynomial in the sum of lengths of the observed examples. However, it requires storage of all the examples seen by the learner to ensure that each time the representation of the target is modified, it stays consistent with all examples seen previously.

3.8 Issues in Efficient Methods for Regular Grammar Inference

The above sections bring out an interesting contrast between the different methods for regular grammar inference. Exact learning of the target DFA requires that the learner be provided with additional information in the form of a characteristic sample or have access to a knowledgeable teacher who answer queries. Incremental approaches guarantee convergence in the limit i.e., when the set of examples seen by the learner satisfy certain properties. Other approaches that attempt to learn the target DFA only from the labeled examples often rely on heuristics and are not guaranteed to converge to the target [29].

An interesting question would be to see whether DFA are *approximately* learnable.

Valiant's distribution-independent model of learning also called the *probably approximately correct model* (PAC model) [93] is a widely used model for approximate learning. When adapted to the problem of learning DFA, the goal of a PAC learning algorithm is to obtain from a randomly drawn set of labeled examples, in *polynomial time*, with high probability, a DFA that is a *good approximation* of the target DFA [76]. A good approximation of the target is defined as one for which the probability of error on an unseen example is less than a pre-specified *error parameter* ϵ. Since on a given run, the randomly drawn examples of the target DFA might not be fully representative of the target, a concession is made to the PAC learning algorithm to produce a good approximation of the target with high probability (greater than $1-\delta$ where δ is a pre-specified *confidence parameter*). In the case of learning DFA the accepted notion of polynomial time is that the algorithm for approximate learning of DFA must run in time that is polynomial in N (the number of states of the target DFA), $1/\epsilon$ (the error parameter), $1/\delta$ (the confidence parameter), $|\Sigma|$ (the alphabet size), and m (the length of the longest example seen by the learner) [76].

Angluin's L^* learning algorithm can be adapted to the PAC learning framework to show that DFA can be efficiently PAC learned given that the learner is allowed to pose a polynomial number of membership queries [3]. However, PAC learning of DFA is proven to be a hard problem in that there exists no polynomial time algorithm that efficiently PAC learns DFA from labeled examples alone [44, 77]. The PAC model's requirement of learnability under all conceivable probability distributions is often considered too stringent in practice. Pitt identified the following open research problem: *Are DFA's PAC-identifiable if examples are drawn from the uniform distribution, or some other known simple distribution?* [76].

It is not unreasonable to assume that in a practical learning scenario a learner is provided with *simple representative* examples of the target concept. Intuitively, when we teach a child the rules of *multiplication* we are more likely to first give *simple* examples like 3×4 instead of examples like 1377×428. A *representative set* of examples would enable the learner to identify the target concept exactly. For example, a characteristic set for a DFA would constitute a suitable representative set for the DFA learning problem. The question then is whether we can formalize what simple examples mean. *Kolmogorov complexity* provides a machine independent notion of *simplicity* of objects. Intuitively, the Kolmogorov complexity of an object (represented by a binary string x) is the length of the shortest binary program that computes x. Objects that have regularity in their structure (i.e., objects that can be easily compressed) have low Kolmogorov complexity. Consider for example, the string $s_i = 010101...01 = (01)^{500}$. A program to compute this string would be *Print 01 500 times*. On the other hand consider a totally random string $s_j = 1100111010000\ldots01$ of length 500. This string cannot be compressed (as s_i was) which means that a program to compute s_j would be *Print 1100111010000 ... 01* (i.e., the entire string s_j would have to be specified). The length of the program that computes s_i is shorter than that of the program that computes s_j thus $K(s_i) \leq K(s_j)$.

If there exists a *Turing Machine* M such that it reads a string y as input and produces x as output then y is said to be a *program* for x. The Kolmogorov complexity of x is thus less than or equal to $|y| + \eta$ (i.e., the length of y plus some constant). Clearly, the Kolmogorov complexity of any string is bounded by its length because there exists a Turing Machine that computes the identity function $x \longrightarrow x$. Further, there is a *universal* Turing machine U that is capable of simulating all Turing machines. It can be shown that the Kolmogorov complexity of an object with respect to U differs from the Kolmogorov complexity of the object with respect to any other machine by at most an additive constant. Thus, the Kolmogorov complexity of an object with respect to the Universal Turing machine is treated as the Kolmogorov complexity of the object. The interested reader is referred to [54] for a complete treatment of Kolmogorov complexity and related topics.

The *Solomonoff-Levin* universal distribution $\mathbf{m}$ assigns high probability to objects that are *simple* i.e., $\mathbf{m}(x) \approx 2^{-K(x)}$. If we assume that Kolmogorov complexity is a reasonable notion of simplicity of an object then universal distribution gives us a way of sampling simple objects. Li and Vitányi proposed the framework of PAC learning under the universal distribution (called the *simple PAC* model) and demonstrated that several concept classes such as *simple* k-reversible DFA and $\log n$-term DNF (i.e., boolean formulas in *disjunctive normal form*) are PAC learnable under this model while their learnability under the standard PAC model is unknown [53]. Recently, Denis *et al* proposed a variant of the *simple* PAC learning model where a teacher might intelligently select simple examples based on his/her knowledge of the target. Under this model (called the PACS model) they showed that the entire class of k-reversible DFA and the class of *poly*-term DNF are efficiently PAC learnable. Parekh and Honavar have answered Pitt's open research question in the affirmative by demonstrating that the class of *simple*-DFA (DFA whose canonical representations have low Kolmogorov complexity) can be efficiently PAC learned under the *simple* PAC learning model and the entire class of DFA can be efficiently PAC learned under the PACS model [68, 71]. Their approach is based on the fact that for any DFA there exists a characteristic set of *simple* examples. Further, it can be demonstrated that with very high probability this set will be drawn when examples are sampled according to the universal distribution. Thus, by using the RPNI algorithm in conjunction with a randomly drawn set of simple examples, it is showed that DFA are efficiently PAC learnable. This work naturally extends the teachability results due to Goldman and Mathias [38] and the equivalent model of learning grammars from characteristic samples due to Gold [37] to a probabilistic framework (see [68] for details). This idea of learning from simple examples holds tremendous potential in difficult learning scenarios. It is of interest to explore the applicability of learning from simple examples (for an appropriately defined notion of *simplicity*) in natural language learning.

4 Stochastic Regular Grammar Inference

Often in practical applications of grammar inference, negative examples are not directly available to the learner. Gold proved that regular grammars cannot be learned from positive examples alone [37]. In such cases where only positive examples are available, learning is made more tractable by modeling the target as a stochastic grammar which has probabilities associated with the production rules.

4.1 Stochastic Finite State Automata

A stochastic finite state automaton (SFA) is defined as $A = (Q, \Sigma, q_0, \pi)$ where Q is the finite set of N states (numbered q_0, q_1, q_2, $\ldots, q_{N-1}$), Σ is the finite alphabet, q_0 is the start state, and π is the set of probability matrices. $p_{ij}(a)$ is the probability of transition from state q_i to q_j on observing the symbol a of the alphabet. A vector of N elements π_f represents the probability that each state is an accepting state. The following normalizing constraint must hold for each state q_i:

$$\sum_{q_j \in Q} \sum_{a \in \Sigma} p_{ij}(a) + \pi_f(i) \quad = \quad 1$$

It states that the total probability of each transition out of the state q_i together with the probability that q_i is an accepting state must be 1. The probability $p(\alpha)$ that a string α is generated by the SFA is computed as the sum over all states j of the probability that the SFA ends up in state j upon reading α times the probability that j is an accepting state.

$$\begin{aligned} p(\alpha) &= \sum_{q_j \in Q} p_{0j}(\alpha)\pi_f(j) \\ p_{ij}(\alpha) &= \sum_{q_k \in Q} \sum_{a \in \Sigma} p_{ik}(\beta)p_{kj}(a) \quad \text{where } \beta a = \alpha \end{aligned}$$

The language generated by a SFA A is called a stochastic regular language and is defined as $L(A) = \{w \in \Sigma^* | p(w) \neq 0\}$ (i.e., the set of sentences that have a non-zero probability of being accepted by the SFA). Fig. 10 shows the state transition diagram and associated probabilities of a stochastic finite state automaton.

Figure 10 comes here (deterministic SFA)

4.1.1 The Alergia Algorithm for Learning SFA

Carrasco and Oncina have developed an algorithm for the inference of deterministic stochastic finite state automata (DSFA) [12]. A DSFA is a SFA where for each state $q_i \in Q$ and symbol $a \in \Sigma$ there exists at most one state q_j such that $p_{ij}(a) \neq 0$. This algorithm, called *Alergia*, is based on a state merging approach and is quite similar to the *RPNI* algorithm for inference of DFA. A *prefix tree automaton* (PTA) is constructed from the given set of positive examples S^+ and its states are numbered in standard order (as described in section 3.4.3). The initial probabilities π and π_f are computed based on the relative frequencies with which each state and transition of the PTA is visited by the examples in S^+ (see example below).

A quadratic loop merges the states of the PTA in order i.e., at each step q_i the algorithm attempts to merge state q_i with the states q_0, q_1, $\ldots$ q_{i-1} in order. However, unlike the RPNI algorithm where the state merges were controlled by a set of negative examples S^-, *Alergia* merges states that are considered to be similar in terms of their *transition* behavior (described by the states reached from the current state) and their *acceptance* behavior (described by the number of positive examples that terminate in the current state). The determination of similarity is statistical in nature and is controlled by a parameter α which ranges between 0 and 1. The probabilities π and π_f are re-computed after each state merge. The algorithm is guaranteed to converge to the target stochastic finite state automaton in the limit when a complete sample is provided. The worst case complexity of *Alergia* is cubic in the sum of the lengths of examples in S^+.

Example

Consider the set $S^+ = \{\lambda, a, \lambda, abb, \lambda, bb, a, aa, abab, bb, \lambda, \lambda, a, a, bb\}$. The corresponding PTA is shown in Fig. 11. In the figure, the pair $[x, y]$ besides each state denotes the number of times the state is visited (x) and the number of strings that terminate in the state (y). Similarly, the number $[z]$ by each transition indicates the number of times the arc was used by the strings in S^+. Given this, we compute $\pi_f = \{\frac{5}{15}, \frac{4}{7}, 0, 1, 0, 1, 0, 1, 1\}$. Similarly, $p_{01}(a) = \frac{7}{15}$, $p_{01}(b) = 0$, $p_{02}(a) = 0$, $p_{02}(b) = \frac{3}{15}$, and so on. Intuitively, 5 strings out of 15 that visit state q_0 terminate in q_0. Hence the probability that q_0 is an accepting state is $\frac{5}{15}$ i.e., $\pi_f(0) = \frac{5}{15}$. Similarly, 7 of the 15 strings that visit q_0 take the transition $\delta(q_0, a) = q_1$. Hence the transition probability, $p_{01}(a) = \frac{7}{15}$ and so on.

Figure 11 comes here (Stochastic regular grammars PTA)

The state merging procedure considers q_0 and q_1 for merging. Assuming that q_0 and q_1 are found to be similar they would be merged together. This introduces non-determinism since now the new state obtained by merging q_0 and q_1 has two transitions on the letter b (*viz.* to states q_1 and q_2 respectively). The resulting SFA is determinized by recursively merging the states that cause non-determinism. This deterministic SFA (obtained by recursive state merging) is shown in Fig. 12. The probabilities π and π_f are re-computed and the state merging is continued. The procedure terminates when no further state merging is considered statistically significant and returns the most current DSFA.

Figure 12 comes here (Det SFA after merging q0 and q1)

4.2 Hidden Markov Models

Hidden Markov Models (HMM) are the widely used generalizations of stochastic finite state automata where both the state transitions and the output symbols are governed by probability distributions. HMMs have been applied successfully in speech recognition and cryptography [81].

Formally, a HMM comprises of the following elements:

- A finite set of states Q (labeled q_1, q_2, ..., q_N)
- A finite alphabet Σ (of symbols labeled σ_1, σ_2, $\ldots \sigma_M$)
- The state transition probability matrix A $(N \times N)$ where a_{ij} represents the probability of reaching state q_j from the state q_i.
- The observation symbol probability matrix B $(N \times M)$ where b_{ij} is the probability of observing the symbol σ_j in the state q_i.
- The initial state probability matrix π $(N \times 1)$ where π_i is the probability of model starting in state q_i.

Note that in the case of HMM, the symbols are associated with individual states instead of the individual transitions as is the case with SFA. These models satisfy the Markovian property which states that the probability of the model being in a particular state at any instant of time depends only on the state it was in at the previous instant. Further, since each symbol is associated with possibly several different states, given a particular symbol it is not possible to directly determine the state that generated it. It is for this reason that these Markov models are referred to as *hidden*.

The probabilistic nature of HMM makes them suitable for use in processing temporal sequences. For example, in the case of speech recognition, a separate HMM is created to model each word of the vocabulary. Given an observed sequence of sounds, one then determines the most likely path of the sequence through each model and selects the word associated with the most likely HMM.

Example

Fig. 13 shows a HMM with $N = 8$ states and an alphabet $\Sigma = \{a, b, c\}$. The state transition probabilities and the observation symbol probabilities are shown beside the respective transitions and symbols in the figure. If the initial state probability matrix $\pi = \{1, 0, 0, 0, 0, 0, 0, 0\}$ then q_1 is the only initial state. There are two distinct ways of generating the string *bacac* with non-zero probability using this HMM *viz.* using the state sequences $q_1 q_2 q_3 q_4 q_5$ and $q_1 q_2 q_6 q_7 q_8$ respectively.

Figure 13 comes here (HMM)

4.2.1 Learning Hidden Markov Models

Given a N-state HMM with the model parameters $\lambda = (A, B, \pi)$ and an observation sequence $O = O_1 O_2 \ldots O_t$, the probability of this observation sequence $\Pr(O|\lambda)$ can be computed using the *forward-backward* estimation procedure [81]. Further, it is possible to determine an optimal state sequence $i_i i_2 \ldots i_t$ (where $i_k \in Q,\ 1 \leq k \leq t$) that most likely produced the observation sequence O using the *Viterbi* algorithm. Perhaps the most critical problem with learning HMMs is the adjustment of the model parameters that would maximize the probability of a single observation sequence. In order to determine the model parameters an iterative procedure called the *Baum-Welch* algorithm can be used [7].

Stolcke and Omohundro present a more general approach to the HMM learning problem [88]. Their approach is a *Bayesian model merging* strategy which facilitates learning the HMM structure as well as the model parameters from a given set of positive examples. The first step constructs an initial model comprising of a unique path from the start state q_1 to a final state q_f for each string in the set of positive examples S^+. This is similar to the PTA constructed by the *Alergia* algorithm for learning SFA. The initial probabilities are assigned as follows: The probability of entering the path corresponding to each string from the start state is uniformly distributed (i.e., if there are k paths corresponding to k strings in S^+ then the probability of entering each path is $1/k$). Within each path the probability of observing the particular symbol at each state and the probability of taking the outgoing arc from the state are both set to 1. Next a *state merging* procedure is used to obtain a generalized model of the HMM. At each step the set of all possible state merges of the current model M_i are considered and two states are chosen for merging such that the resulting model M_{i+1} maximizes the *posteriori* probability $\Pr(M_{i+1}|S^+)$ of the data. By Bayes' rule, $\Pr(M_{i+1}|S^+) = \Pr(M_{i+1}) \Pr(S^+|M_{i+1})$. $\Pr(M_{i+1})$ represents the *model prior probability*. Simpler models (for example, models that have a fewer number of states) have a higher prior probability. The data likelihood $\Pr(S^+|M_{i+1})$ is determined using the *Viterbi path* for each string in S^+ (i.e., the optimal state sequence for each string). The state merging procedure is stopped when no further state merge results in an increase in the posteriori probability.

A brief comparison of *Alergia* with the *Bayesian model merging* approach is in order. *Alergia* uses a depth-first order of merging states and runs in time that is polynomial in the size of S^+. Further, is guaranteed to converge to the target deterministic stochastic finite automaton in the limit. On the other hand the Bayesian model merging procedure for HMM considers all possible state mergings at each step before merging two states and is thus computationally more expensive. Additionally, there is a need for appropriately chosen priors in the Bayesian model merging procedure. Experimental results have shown that relatively uninformed priors perform reasonably well. Although there is no guaranty for convergence to the target HMM, the Bayesian model merging procedure has been successfully used in various practical applications. Further, the Bayesian model merging procedure is also used to learn class based *n-grams* and *stochastic context free grammars* which are more general in terms of their descriptive capabilities than the deterministic stochastic finite automata inferred by Alergia.

5 Inference of Context Free Grammars

Context Free Grammars (CGF) represent the next level of abstraction (above regular grammars) in the Chomsky hierarchy of formal grammars. A CFG is a formal language grammar $G = (V_N, V_T, P, S)$ where the production rules are of the form: $A \longrightarrow \alpha$ where $A \in V_N$ and $\alpha \in (V_T \cup V_N)^*$. Most work in inference of context free grammars has focussed on learning the standardized *Chomsky Normal Form* (CNF) of CFG in which the rules are of the form $A \longrightarrow BC$ or $A \longrightarrow a$ where $A, B, C \in V_N$ and $a \in V_T$. The example grammar for simple declarative English language sentences (Fig. 1) is in CNF. Given a CFG G and a string $\alpha \in V_T^*$ we are interested in determining whether or not α is generated by the rules of G and if so, how is it derived. A *parse tree* graphically depicts how a particular string is derived from the rules of the grammar. For example, a parse tree for the sentence: *The boy saw a ferocious tiger* is shown in Fig. 14. The non-leaf nodes of a parse tree represent the non-terminal symbols. The root of each sub-tree together with its children (in order from left to right) represents a single production rule. Parse trees provide useful information about the structure of the string or the way it is interpreted. Often, in learning context free grammars, parse trees obtained from example sentences are provided to the learner. Parse trees simplify the induction task since they provide the learner with the necessary components of the structure of the target grammar.

Figure 14 comes here (Parse Tree)

5.1 Inducing Context Free Grammars

The theoretical limitations that plague regular grammar inference carry over to the context free case as well because the set of context free languages properly contains the set of regular languages. The problems are further compounded by the fact that several decision problems related to CFG are undecidable. Given two CFG G_1 and G_2 there exists no algorithm that can determine whether G_1 is *more general than* G_2 (i.e., $L(G_1) \supseteq L(G_2)$). Similarly, there exists no algorithm that can answer the question "*Is* $L(G_1) \cap L(G_2) = \phi$?" [42]. Thus, exact learning of the target grammar is seldom attempted in the case of CFG. However, several practically successful approaches have been devised for heuristically learning approximations to the target grammar.

5.1.1 Learning Recursive Transitions Networks

Recursive Transition Networks are an equivalent representation of CFG. They are essentially simple finite state diagrams where the tests on the arcs refer to either terminal symbols or to entire subnetworks [48]. For example, the recursive transition diagram corresponding to the context free grammar of Fig. 1 is shown in Fig. 15. The first network indicates the start symbol S and invokes the subnetworks NP and VP by specifying them as tests on the arcs. The arcs of the subnetworks to the right of the figure specify individual words from the set of terminals.

Figure 15 comes here (simple RTN)

One approach to learning recursive transition networks based on Langley's GRIDS algorithm [47] is similar to the state merging techniques used for learning DFA. Given a set of positive examples a degenerate one level transition network is constructed that accepts exactly the sentences that belong to the training set. This initial network is analogous to the prefix tree automaton in the DFA learning problem. For example, if we are given the sentences "The boy saw a sly fox", "A boy followed a ferocious tiger", "The tiger saw a sly fox", etc. then a sample one level transition network can be constructed as shown in Fig. 16.

Figure 16 comes here (Initial RTN)

The algorithm identifies patterns that co-occur in the initial network and represents these as new subnetworks that correspond to phrases in the language. Measures such as simplicity of the subnetwork (in terms of nodes and links) are used to select a subnetwork among several competing alternatives. The creation of subnetworks for the above example proceeds as shown in Fig. 17. The subnetwork creation is repeated until further splitting results in a degradation of the performance. Similar subnetworks are then repeatedly merged to give more general transition networks. Sample merging steps are shown in Fig. 18. Negative examples (if available) or heuristic knowledge can be used to guide the merging process to prevent the creation of overly general networks.

Figure 17 comes here (Creating Subnetworks)

Figure 18 comes here (Merging Subnetworks)

5.1.2 Learning CFG using Version Spaces

VanLehn and Ball proposed a version space based approach for learning CFG [94]. Given a set of labeled examples, a trivial grammar that accepts exactly the set of positive examples can be constructed. Again, this is similar to the PTA constructed by several regular grammar inference algorithms. A version space is defined as the set of all possible generalizations of the trivial grammar that are consistent with the examples. Since the set of grammars consistent with a given presentation is infinite it becomes necessary to restrict the grammars included in the version space. The *reducedness* bias restricts the version space to contain only *reduced* grammars i.e., grammars that are consistent with a given sentences but no proper subset of their rules is consistent with the given sentences. A further problem in the case of CFG is that the *more general than* relationship which determines if two grammars G_1 and G_2 are such that $L(G_1) \supseteq L(G_2)$ is undecidable. This problem is circumvented by defining a *derivational version space* i.e., a version space based on unlabeled parse trees constructed from positive sentences. The idea is to induce partitions on the unlabeled non-terminal nodes of the parse trees. For example, consider a presentation of two sentences *cat* and *black cat*. Two unlabeled parse trees corresponding to these sentences are depicted in Fig. 19. The set of three unlabeled nodes can be partitioned in 5 different ways (two of which are shown in Fig. 19). Each partition corresponds to a grammar that is consistent with the sentences. The derivational version space is a union of all the grammars obtained from the unlabeled parse trees.

Figure 17 comes here (Derivational Version Space - Vanlehn and Ball)

An operation *FastCovers* based on the partitions of the sets of non-terminals determines the partial order among elements of the derivational version space. The algorithm for updating the version space then proceeds as follows: Each new positive example causes the version space to expand by considering the new parse trees corresponding to the example. Each negative example causes the grammars that accept the negative example to be eliminated from the version space. The *FastCovers* relation (which is a variant of the *grammar covers* property described in section 3.3) is used to prune those grammars in the version space that cover the grammar accepting the negative example. This approach solves small induction problems completely and provides an opportunity for incorporation of additional biases to make the learning tractable in the case of larger problems [94].

Giordano studied the version space approach to learning context free grammars using a notion of *structural containment* to define a partial order on the search space of CFG [35]. This approach provides another mechanism to circumvent the problem that arises because the operation *more general than* is not decidable in CFG. A *structure* generated by a grammar is its derivation tree with all non-terminal labels removed. A grammar G_i is *structurally contained in* (or *structurally equivalent to*) a grammar G_j if the set of structures generated by G_i is contained in (or is equal to) the set of structures generated by G_j. Structural containment implies containment in the sense of languages (i.e., if G_i is structurally contained in G_j then the language of G_i is a subset of the language of G_j). Further, structural containment is polynomial time computable for *uniquely invertible* grammars (i.e., grammars in which the right hand sides of no two production rules are the same). It can be shown that the set of uniquely invertible grammars is equivalent to the whole set of context free grammars hence applying the version space strategy for learning uniquely invertible grammars does not restrict the capacity of the inference system in any way.

5.1.3 Learning NPDA using Genetic Search

Lankhorst presented a scheme for learning non-deterministic pushdown automata (NPDA) from labeled examples using genetic search [49]. *Push down automata* (PDA) are recognizing devices for the class of context free grammars (just as FSA are recognizing devices for regular grammars). A PDA comprises of a FSA and a stack. The extra storage provided by the stack enables the PDA to recognize languages such as *palindromes* which are beyond the capability of FSA. Lankhorst's method encodes a fixed number of transitions of the PDA on a single chromosome. Each chromosome is represented as a bit string of a fixed length. Standard mutation and crossover operators are used in the genetic algorithm. The fitness of each chromosome (which represents a NPDA) is a function of three parameters viz. *training accuracy*, *correct prefix identification*, and *residual stack size*. The *training accuracy* measures the fraction of the examples in the training set that are correctly classified. PDA that are able to parse at least a part of the input string correctly are rewarded by the *correct prefix identification* measure. A string is said to be accepted by a PDA if after reading the entire string either the FSA is in an accepting state or the stack is empty. The *residual stack size* measure assigns higher fitness to PDA that leave as few residual symbols on the stack as possible after reading the entire string.

5.1.4 Learning Deterministic CFG using Connectionist Networks

Several researchers have explored connectionist approaches for learning context free grammars. Das *et al* proposed an approach for learning deterministic context free grammars using recurrent neural network push down automata (NNPDA) [21, 22]. This model uses a recurrent neural network similar to the one described in section 3.6.1 in conjunction with an external stack to learn a proper subset of deterministic context free languages. Moisl adopted deterministic push down automata (DPDA) as adequate formal models for general natural language processing (NLP) [62]. He showed how a simple recurrent neural network can be trained to implement a finite state automaton which simulates the DPDA. Using a computer simulation of a parser for a small fragment of the English language Moisl demonstrated that the recurrent neural network implementation results in a NLP device that is broadly consistent with the requirements of a typical NLP system and has desirable emergent properties. We briefly describe the NNPDA model proposed by Das *et al* [21].

The NNPDA consists of a recurrent neural network integrated with an external stack through a hybrid error function. It can be trained to simultaneously learn the state transition function of the underlying push down automaton and the actions that are required to control the stack operation. The network architecture is similar to the one described in section 3.6.1. In addition to the *input* and *state* neurons the network maintains a group of *read* neurons to read the top of the external stack and a single non-recurrent *action* neuron whose output identifies the stack action to be taken (i.e., *push*, *pop*, or *no-op*). The complexity of this PDA model is reduced by making the following simplifying assumptions: The input and stack alphabets are the same, the *push* operation places the current input symbol on the top of the stack, and ϵ-transitions are disallowed. These restrictions limit the class of languages learnable under this model to a finite subset of the class of deterministic context free languages. *Third* order recurrent neural networks are used in this model where the weights modify a product of the current state, the current input, and the current top of stack to produce the new state and modify the stack. The activation of the state neurons at time $t+1$ is determined as follows.

$$s_i(t+1) \quad = \quad g(\sum_j \sum_k \sum_l w_{ijkl} s_j(t) i_k(t) r_l(t))$$

where w_{ijkl} is the third order weight that modifies the product of the activation of the state neuron S_j, the input I_k, and the activation of the read neuron R_l at time t. g is the standard sigmoid function $g(x) = \frac{1}{1+e^{-x}}$. During training, input sequences are presented one at a time and the activations are propagated until the end of the sequence is reached. At this time the activation of the designated output neuron is compared with the class label assigned to the particular input sequence (1 or 0). If the sequence is misclassified then the network's weights are modified according to the weight update rule. The weight update rule is designed to minimize a suitably chosen error function E. E is a function of the activation of the designated output neuron and the length of the stack at the end of the presentation of the entire sequence. For positive sequences it is desired that the output activation be close to 1.0 and the stack be empty and for negative sequences it is desired that the output activation be close to 0 and the stack be non-empty. This NNPDA model is capable of learning simple context free languages such as $a^n b^n$ and *parenthesis*. However, the learning task is computationally intensive. The algorithm does not converge for more nontrivial context free languages such as $a^n b^n c b^m a^m$.

Das *et al* studied the incorporation of prior knowledge in NNPDA to make learning more tractable [22]. Two different types of knowledge could be made available to the model: knowledge that depends on the training data alone and partial knowledge about the automaton being inferred. As explained in section 3.8, presenting simpler examples of the target concept can considerably simplify the learning task. Specifically, in the case of context free grammars the learning process benefits from seeing shorter examples early on. Simulation results demonstrated that incremental learning (where the training examples are presented in increasing order by length) achieved a substantial reduction in the training time of the NNPDA algorithm. Additionally, a knowledgeable teacher might be asked to indicate for each negative string the character which forces the PDA to a reject state. This information (if available) could be used to stop training the NNPDA on negative examples beyond the rejection point identified by the teacher. Experiments showed that such selective presentation of strings enables the NNPDA algorithm to converge on nontrivial languages such as $a^n b^n c b^m a^m$. The knowledge of the task being solved can be exploited to further assist the NNPDA learning algorithm. Two such methods including pre-determining the initial values of some of the network's weights and presenting structured training examples where the order of generation of each word of the sentence is indicated by parentheses were shown to improve learning speed [22].

5.2 Stochastic CFG

The success of HMM in a variety of speech recognition tasks leads one to ask if the more powerful stochastic context free grammars (SCFG) could be employed for speech recognition. The advantages of SCFG lie in their ability to capture the embedded structure within the speech data and their superior predictive power in comparison with regular grammars as measured by prediction entropy [50].

The *Inside-Outside* algorithm can be used to estimate the free parameters of a SCFG. Given a set of positive training sentences and a SCFG whose parameters are randomly initialized, the inside-outside algorithm first computes the most probable parse tree for each training sentence. The derivations are then used to re-estimate the probabilities associated with each rule and the procedure is repeated until no significant changes occur to the probability values.

Prior to invoking the inside-outside algorithm one must decide on some appropriate structure for the SCFG. Stolcke and Omohundro's Bayesian Model merging approach can be used for learning both the structure and the associated parameters of the SCFG from a set of training sentences [88]. This approach is analogous to the one for HMM induction described earlier. New sentences are incorporated by adding a top level production from the start symbol S. Model merging involves *merging* of non-terminals in the SCFG to produce a more general grammar with fewer non-terminals and *chunking* of non-terminals where a sequence of non-terminals is abbreviated using a new non-terminal. A powerful beam search (which explores several relatively similar grammars in parallel) is used to search for the grammar with the highest a-posteriori model probability.

6 Discussion

Research in computational models of formal language acquisition has received significant attention over the past three decades. Their study is motivated partly by the desire to better understand the process of natural language acquisition and partly by the numerous practical applications of grammar inference. The preceding sections presented several key algorithms for learning regular, stochastic regular, context free, and stochastic context free grammars.

Regular grammars represent the simplest class of grammars in the Chomsky hierarchy. The study of regular grammar inference methods is of significant practical interest for a number of reasons: every *finite* language is regular; a context-free language can often be closely approximated by a regular grammar [30]. Unfortunately, the regular grammar inference problem is known to be hard (i.e., there exists no efficient algorithm that can learn the target grammar from an arbitrary set of labeled examples). It is shown that the regular grammar inference problem can be solved efficiently with the help of *representative samples*, or by drawing examples according to a *helpful* distribution (e.g., *simple* distributions), or with the help of a knowledgeable teacher capable of answering queries. Further, as demonstrated by the results of the recent *Abbadingo One DFA Learning Competition* several efficient heuristic approaches provide satisfactory solutions to the regular grammar inference problem [39, 43, 46].

Context-free and context-sensitive grammars generate languages that are clearly more expressive than those generated by regular grammars. This has motivated a large body of research on algorithms for learning such grammars. However, existing algorithms for induction of these grammars are generally heuristic in nature. Although some of them have been demonstrated to perform well on small to medium sized problems, they do not appear to scale well to larger problem sizes. While the context-free and context-sensitive grammars (and their corresponding recognition devices) constitute useful conceptual abstractions for studying fundamental theoretical questions in computer science, and the study of algorithms for learning such languages (or interesting subclasses of such languages) is clearly of significant theoretical interest, their utility in modeling real-world language learning scenarios is open to question. This is because any language that can be generated (and recognized) by a computational device (e.g., a general purpose computer) with a finite amount of memory can be modeled by a regular language. Sentences that are encountered in practice (e.g., in typical use of natural languages) seldom require infinite memory for generation or recognition. Note that no computer in use today has infinite memory and hence is necessarily less powerful than a Turing Machine with its infinite tape. But that has never prevented the widespread use of computers for all sorts of practical applications. Similarly, one might argue that algorithms for learning regular grammars are, for the most part adequate for practical applications of grammar induction.

While languages generated and recognized by finite state, finite memory computational devices (e.g., a push-down automaton with a bounded stack) can in principle be modeled by regular languages, one might still want to use context free or context sensitive languages for reasons of elegance or compactness. Hence, it might be of interest to explore efficient algorithms for learning such languages under the assumption of finite state, bounded memory recognition devices, or a strong *inductive bias* in favor of *simpler* grammars, or under *helpful* distributions.

The different approaches to language acquisition can be grouped into three broad categories: those that make use of purely *symbolic* representations (e.g., L*); those that make use of purely numeric representations (e.g., most connectionist networks); and hybrid approaches that combine both symbolic as well as numeric representations. Algorithms that employ purely symbolic representations often lend themselves to rigorous analysis and guarantee of convergence to the unknown target grammar given a non-contradictory, noise-free, representative set of examples. On the other hand, such algorithms can be quite brittle in the presence of noise. Algorithms that employ purely numeric representations are robust in the presence of limited amounts of noise, are able to learn from relatively smaller training sets, and are known to scale well to larger problem sizes [51, 52]. However, their convergence properties are generally poorly understood and the learned models are not as transparent. While it is possible to extract a symbolic description of a learned grammar from a connectionist network, the procedures used for doing so are largely heuristic in nature. Against this background, it is of interest to investigate hybrid algorithms that exploit the strengths of both numeric as well as symbolic representations [41].

Research in cognitive psychology, linguistics, and related areas has raised several important issues that must be (at least partially) addressed by any practical system for language acquisition. An in-depth discussion of recent developments in these areas and their implications for models of language learning is beyond the scope of this chapter. However, in what follows, we attempt to briefly highlight some of the key issues.

Mechanisms of language acquisition have been the subject of intense debate for several decades. As in the case of most cognitive phenomena, the opinions that have been voiced span a wide spectrum. At one end of the spectrum is the *behaviorist* view advocated by Skinner which suggests a *tabula rasa* approach to learning [87]. In this view, language acquisition is driven collectively by *stimuli* experienced by an individual, the *response* evoked by the stimuli, and the *reinforcing stimuli* that follow the response. For example, when a child feels thirsty – the stimulus of *milk-deprivation*, it might respond by saying "*milk* " (essentially by a process of trial and error) and the parents might provide a reinforcing stimulus in the form of a bottle of milk. Thus, it was argued that language originates from physical need and is a means to a physical end. This behaviorist view equates learning with essentially the formation of (typically but not necessarily direct) stimulus-response associations. Note that *semantics* plays a central role in this model of language learning.

At the other end of the spectrum is the view advocated by Chomsky that language is *not* a cultural artifact that can be learned [14]. This claim is based on what Chomsky has called the *argument from the poverty of stimulus* which states that the language stimuli received by the child are insufficient for acquiring abstract grammatical rules solely via inductive learning [15]. Language is thus innate in the biological make-up of the brain. In other words, language is a human *instinct* [75]. Thus, children must be born with a *Universal Grammar* or a system of principles, conditions, and rules that are elements or properties of all human languages [15]. Chomsky's subsequent writings [16] argue that the innate *universal grammar* is organized into modular subsystems of principles concerning *government*, *binding*, and *thematic roles*. Principles or, alternatively, particular lexical items may have parameters associated with them whose values are set according to language experience [56].

It is perhaps worth noting that Chomsky's arguments of innateness of language appear to be restricted to language *syntax*, while the behaviorist view of language acquisition is primarily concerned with language *semantics*. If we restrict our attention to syntax acquisition, results in machine learning and computational learning theory appear to lend credence to Chomsky's view. Almost all learning algorithms either explicitly or implicitly use appropriate *representational* and *inductive* biases [59, 61]. The representational bias of the algorithm determines the space of candidate hypotheses (or sets of grammars in the case of language acquisition) that are considered by the learner. In order for learning to succeed, the hypothesis space must be *expressive enough* to contain the target grammar. On the other hand, since learning typically involves searching the hypothesis space in order to identify an unknown target on the basis of data or other information available in the learner's the environment, if it is *overly expressive*, it can increase the complexity of the learning task. Inductive bias of the algorithm tends to focus or bias the search for the unknown target hypothesis. For instance, an algorithm for regular language acquisition has a representational bias that allows it to consider only regular grammars as candidate solutions. In addition, such an algorithm might have a strong inductive bias that directs its search in favor of, say, *simpler* rules.

Most efficient algorithms that have been devised for language learning rely on the availability of negative examples of the unknown target grammar. While this may be a reasonable assumption to make in several practical applications of language acquisition or grammatical inference (e.g., syntactic approaches to pattern recognition), it has been argued that the apparent lack of *negative examples* presents a language *learnability dilemma*. Wexler and Manzini point out that children seldom see negative examples of a grammar and are rarely corrected when they utter grammatically incorrect sentences [96] and consequently they often over-generalize from the limited number of sentences they hear. The lack of negative examples prevents any direct recovery from over-generalizations of the target grammar. Against this background, several mechanisms (such as *constraint sampling* and use of the *uniqueness principle*) have been suggested for recovering from such over-generalizations [74].

It is tempting to think of Chomsky's Universal Grammar (if in fact it exists) as determining the representational bias of a natural language learner. The inductive bias of the learner might be enforced by the innately determined range of values that the various parameters are allowed to take. It might also be a result of the process of brain development in children. In this context, it is worth noting that Prince and Smolensky [80] have recently developed a grammatical formalism called *optimality theory* which brings together certain connectionist computational principles into the essentially symbolic theory of universal grammar. In this formalism, possible human languages share a common set of universal constraints on well-formedness. These constraints are highly general, and hence are conflicting. Consequently, some of them must be violated in *optimal* (i.e., grammatical) structures. The differences in the world's languages emerge via different priority rankings of the universal constraints: each ranking is a language-particular grammar, a means of resolving the inherent conflicts among the universal constraints.

In this chapter, we have studied language learning primarily from the point of view of acquiring syntax. The learning of *semantics* or the meanings of words and sentences is central natural language acquisition. An excellent overview of empirical methods for natural language processing (including techniques for speech recognition, syntactic parsing, semantic processing, information extraction, and machine translation) appears in [10]. Thompson *et al* have designed a system for automatically acquiring a semantic lexicon from a corpus of sentences paired with representations of their meanings [90]. This system learns to parse novel natural language sentences into an suitable semantic representation (such as a logical database query) and is shown to successfully learn useful lexicons for a database interface in four different natural languages. Current understanding of semantic development in children as summarized by Pan and Gleason [66] is surprisingly consistent with the essential tenets of the behaviorist view advocated by Skinner. It is believed that children typically learn meanings of words and phrases by identifying suitable mappings between them and perceptual stimuli. In summary, it seems plausible that something akin to a universal grammar and its interaction with the rest of the brain is what enables children to become fluent in any language during their childhood. Thus language acquisition has to necessarily rely on not only mechanisms for learning language-specific grammars (which might involve tuning the parameters of the universal grammar) but also processes for inducing the meaning of the words and sentences in the language. This argues for a model of language learning that accounts for both syntactic as well as semantic components of the language. Recent research results offer some intriguing preliminary insights on the development of formal models of how children learn language [27, 28]. Regier proposed a computational model of how some lexical items describing spatial relations might develop in different languages [82]. His system includes a simple model of the visual system which is common to all human beings and thus must be the source from which all visual concepts arise. Using conventional backpropagation techniques his system was able to learn a spatial terms from labeled example movies for a wide range of languages. Stolcke's Bayesian model merging approach [88] was able to learn semantics on the same domain (of labeled example movies) in the form of a simple stochastic attributed grammar. Bailey proposed a computational model to address the issue of how a child *makes use* of the concepts that he/she has learned [5, 4]. His system learns to produce verb labels for actions and also carries out the actions specified by the verbs it has learned.

References

[1] R. Alquezar and A. Sanfeliu. A hybrid connectionist-symbolic approach to regular grammar inference based on neural learning and hierarchical clustering. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second ICGI-94, Lecture Notes in Artificial Intelligence 862*, pages 203–211. Springer-Verlag, 1994.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[4] D. Bailey. *When Push Comes to Shove: A Computational Model of the Role of Motor Control in the Acquisition of Action Verbs.* PhD thesis, University of California at Berkeley, Berkeley, CA, 1997.

[5] D. Bailey, Feldman J., S. Narayanan, and G. Lakoff. Modeling embodied lexical development. In *Proceedings of the* 19^{th} *Cognitive Science Society Conference*, pages ??–?? Stanford University Press, 1997.

[6] G. Ball, D. Ling, D. Kurlander, J. Miller, D. Pugh, T. Skelly, A. Stankosky, D. Thiel, M. Van Dantzich, and T. Wax. Lifelike computer characters: The persona project at microsoft research. In J. Bradshaw, editor, *Software Agents.* MIT Press, Cambridge, MA, 1997.

[7] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions in markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[8] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–303, 1998.

[9] M. R. Brent. Advances in the computational study of language acquisition. *Cognition*, 61:1–38, 1996.

[10] E. Brill and R. Mooney. An overview of empirical natural language processing. *AI Magazine*, 18(4, Winter 1997):13–24, 1997.

[11] D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proceedings of the AAAI-96 (vol. 1)*, pages 62–67. AAAI Press/MIT Press, 1996.

[12] R. C. Carrasco and J. Oncina. Learning stochastic regular grammar by means of a state merging method. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second ICGI-94, Lecture Notes in Artificial Intelligence 862*, pages 139–152. Springer-Verlag, 1994.

[13] N. Chomsky. Three models for the description of language. *PGIT*, 2(3):113–124, 1956.

[14] N. Chomsky. Review of b. f. skinner verbal behavior. *Language*, 35:26–58, 1959.

[15] N. Chomsky. *Reflections on Language*. Temple Smith, London, 1976.

[16] N. Chomsky. *Some Concepts and Consequences of the Theory of Government and Binding*. MIT Press, Cambridge, MA, 1982.

[17] A. Cleeremans, D. Servan-Schreiber, and J. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1:372–381, 1989.

[18] D. Clouse, C. Giles, B. Horne, and G. Cottrell. Time-delay neural networks: Representation and induction of finite state machines. *IEEE Transactions on Neural Networks*, 8(5):1065–1070, 1997.

[19] C. Crangle and P. Suppes. *Language and Learning for Robots*. CSLI Lecture Notes: No. 41. CSLI Publications, Stanford, CA., 1994.

[20] S. Das and R. Das. Induction of discrete-state machine by stabilizing a simple recurrent network using clustering. *Computer Science and Informatics*, 21(2):35–40, 1991.

[21] S. Das, C. Giles, and G. Z. Sun. Learning context free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 791–795, 1992.

[22] S. Das, C. Giles, and G. Z. Sun. Using prior knowledge in a nnpda to learn context free languages. In C. Giles, S. Hanson, and J. Cowan, editors, *Advances in Neural Information Processing Systems, 5*, pages 65–72, 1993.

[23] P. Dupont. Incremental regular inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147*, pages 222–237, Montpellier, France, 1996. Springer.

[24] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In *Proceedings of the Second International Colloquium on Grammatical Inference (ICGI'94)*, pages 25–37, Alicante, Spain, 1994.

[25] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[26] J. Elman. Distributed representation, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[27] J. Feldman, G. Lakoff, D. Bailey, S. Narayanan, T. Regier, and A. Stolcke. Lzero: The first five years. *Artificial Intelligence Review*, 10:103–129, 1996.

[28] J. A. Feldman. Real language learning. In *Proceedings of the Fourth International Colloquium on Grammatical Inference (ICGI'98)*, Ames, IA, 1998. (To appear).

[29] K. Fu. *Syntactic Pattern Recognition and Applications.* Prentice Hall, N.J., 1982.

[30] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey (part 1). *IEEE Transactions on Systems, Man and Cybernetics*, 5:85–111, 1975.

[31] C. Giles, D. Chen, H. Miller, and G. Sun. Second-order recurrent neural networks for grammatical inference. *Proceedings of the International Joint Conference on Neural Networks*, 2:273–281, 1991.

[32] C. Giles, B. Horne, and T. Lin. Learning a class of large finite state machines with a recurrent neural network. *Neural Networks*, 8(5):1359–1365, 1995.

[33] C. Giles and C. Omlin. Extraction, insertion and refinement of symbolic rules in dynamically-driven recurrent neural networks. *Connection Science – special issue on Architectures for Integrating Symbolic and Neural Processes*, 5(3,4):307–337, 1993.

[34] C. Giles, C. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[35] J. Giordano. Inference of context-free grammars by enumeration: Structural containment as an ordering bias. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second ICGI-94, Lecture Notes in Artificial Intelligence 862*, pages 212–221. Springer-Verlag, 1994.

[36] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[37] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[38] S. Goldman and H. Mathias. Teaching a smarter learner. In *Proceedings of the Workshop on Computational Learning Theory (COLT'93)*, pages 67–76. A. C. M. Press, 1993.

[39] Colin de la Higuera, J. Oncina, and E. Vidal. Identification of dfa: Data-dependent vs data-independent algorithms. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147*, pages 313–326, Montpellier, France, 1996. Springer.

[40] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[41] V. Honavar. Toward learning systems that integrate multiple strategies and representations. In V. Honavar and L. Uhr, editors, *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, pages 615–644. Academic Press: New York., 1994.

[42] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[43] H. Juillé and J Pollack. Sage: A sampling-based heuristic for tree search. *Machine Learning*, 1998. (submitted).

[44] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the* 21^{st} *Annual ACM Symposium on Theory of Computing*, pages 433–444, New York, 1989. ACM.

[45] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[46] K. Lang, B. Pearlmutter, and R. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of the Fourth International Colloquium on Grammatical Inference (ICGI'98)*, Ames, IA, 1998. (To appear).

[47] P. Langley. Simplicity and representation change in grammar induction. Robotics Laboratory, Stanford Laboratory. (Unpublished manuscript), 1994.

[48] P. Langley. *Elements of Machine Learning.* Morgan Kauffman, Palo Alto, CA, 1995.

[49] M. Lankhorst. A genetic algorithm for induction of nondeterministic pushdown automata. Technical Report CS-R 9502, University of Groningen, The Netherlands, 1995.

[50] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[51] S. Lawrence, S. Fong, and C. Giles. Natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and symbolic Approaches*

to Learning for Natural Language Processing, Lecture Notes in Artificial Intelligence 1040, pages 33–47. Springer-Verlag, 1996.

[52] S. Lawrence, C. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. In *IEEE Transactions on knowledge and Data Engineering*. IEEE Press, 1998. (accepted).

[53] M. Li and P. Vitányi. Learning simple concepts under simple distributions. *SIAM Journal of Computing*, 20:911–935, 1991.

[54] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications, 2^{nd} edition*. Springer Verlag, New York, 1997.

[55] J. C. Martin. *Introduction to Languages and The Theory of Computation*. McGraw-Hill Inc., New York, 1991.

[56] J. M. Meisel. Parameters in acquisition. In P. Fletcher and B. MacWhinney, editors, *A Handbook of Child Language Acquisition*, pages 10–35. Blackwell, 1985.

[57] L. Miclet and J. Quinqueton. Learning from examples in sequences and grammatical inference. In G. *et al* Ferrate, editor, *Syntactic and Structural Pattern Recognition*, pages 153–171. NATO ASI Series Vol. F45, 1986.

[58] C. Miller and C. Giles. Experimental comparison of the efect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):849–872, 1993.

[59] T. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, New Bruskwick, NJ, 1980.

[60] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[61] T. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

[62] H. Moisl. Connectionist finite state natural language processing. *Connection Science*, 4(2):67–91, 1992.

[63] B. K. Natarajan. *Machine Learning: A Theoretical Approach*. Morgan Kauffman, 1991.

[64] C. Omlin and C Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[65] J. Oncina and P. García. Inferring regular languages in polynomial update time. In N. *et al* Pérez, editor, *Pattern Recognition and Image Analysis*, pages 49–61. World Scientific, 1992.

[66] B. Pan and J. Gleason. Semantic development: Learning the meaning of words. In J. Gleason, editor, *The Development of Language*, pages 122–158. Allyn and Bacon, Boston, MA, fourth edition, 1997.

[67] T. Pao and J. Carr. A solution of the syntactic induction-inference problem for regular languages. *Computer Languages*, 3:53–64, 1978.

[68] R. Parekh. *Constructive Learning: Inducing Grammars and Neural Networks*. PhD thesis, Iowa State University, Ames, IA, 1998.

[69] R. G. Parekh and V. G. Honavar. Efficient learning of regular languages using teacher supplied positive examples and learner generated queries. In *Proceedings of the Fifth UNB Conference on AI*, pages 195–203, Fredricton, Canada, 1993.

[70] R. G. Parekh and V. G. Honavar. An incremental interactive algorithm for regular grammar inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147*, pages 238–250, Montpellier, France, 1996. Springer.

[71] R. G. Parekh and V. G. Honavar. Learning dfa from simple examples. In *Proceedings of the Eighth International Workshop on Algorithmic Learning Theory (ALT'97), Lecture Notes in Artificial Intelligence 1316*, pages 116–131, Sendai, Japan, 1997. Springer. Also presented at the *Workshop on Grammar Inference, Automata Induction, and Language Acquisition* (ICML'97), Nashville, TN. July 12, 1997.

[72] R. G. Parekh, C. Nichitiu, and V. G. Honavar. A polynomial time incremental algorithm for regular grammar inference. In *Proceedings of the Fourth International Colloquium on Grammatical Inference (ICGI'98)*, Ames, IA, 1998. Springer. (To appear).

[73] S. Pinker. Formal models of language learning. *Cognition*, 7:217–283, 1979.

[74] S. Pinker. Productivity and conservatism in language acquisition. In W. Demopoulos and A. Marras, editors, *Language Acquisition and Concept Learning*. Ablex Publishing Corporation, Norwood, N. J., 1986.

[75] S. Pinker. *The Language Instinct*. William Morrow and Company, 1994.

[76] L. Pitt. Inductive inference, dfas and computational complexity. In *Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence 397*, pages 18–44. Springer-Verlag, 1989.

[77] L. Pitt and M. K. Warmuth. Reductions among prediction problems: on the difficulty of predicting automata. In *Proceedings of the* 3^{rd} *I.E.E.E. Conference on Structure in Complexity Theory*, pages 60–69, 1988.

[78] J. B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:123–148, 1991.

[79] S. Porat and J. Feldman. Learning automata from ordered examples. *Machine Learning*, 7:109–138, 1991.

[80] A. Prince and P. Smolensky. Optimality: From neural networks to universal grammar. *Science*, 275:104–1610, 1997.

[81] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE Acoustics, Speech, and Signal Processing magazine*, 3(1):4–16, 1986.

[82] T. Regier. *The Human Semantic Potential.* MIT Press, Cambridge, MA, 1996.

[83] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[84] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[85] D. Servan-Schreiber, A. Cleeremans, and J. McClelland. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–193, 1991.

[86] H. Siegelmann and C. Giles. The complexity of language recognition by neural networks. *Neurocomputing*, 15:327–345, 1997.

[87] B. F. Skinner. *Verbal Behavior.* Appleton-Century-Crofts, New York, 1957.

[88] A. Stolcke and S. Omohundro. Inducing probabilistic grammars by bayesian model merging. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second ICGI-94, Lecture Notes in Artificial Intelligence 862*, pages 106–118. Springer-Verlag, 1994.

[89] B. Tesar and P Smolensky. Learnability in optimality theory. Technical Report JHU-CogSci-96-2, Johns Hopkins University, Cognitive Science Department, 1996.

[90] C. Thompson and R. Mooney. Semantic lexicon acquisition for learning natural language interfaces. Technical Report TR AI98-273, Artificial Intelligence Lab, University of Texas, Austin, TX, 1998.

[91] M. Tomita. Dynamic construction of finite-automata from examples using hill-climbing. In *Proceedings of the* 4^{th} *Annual Cognitive Science Conference*, pages 105–108, 1982.

[92] B. Trakhtenbrot and Ya. Barzdin. *Finite Automata: Behavior and Synthesis.* North Holland Publishing Company, Amsterdam, 1973.

[93] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[94] K. Vanlehn and W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2:39–74, 1987.

[95] R. L. Watrous and G. M. Kuhn. Induction of finite-state automata using second-order recurrent networks. In *Advances in Neural Information Processing, 4*. Morgan Kaufmann, 1992.

[96] K. Wexler and R. Manzini. Parameters and learnability in binding theory. In T. Poeper and E. Williams, editors, *Parameter Setting*, pages 41–76. Dordrecht: D. Reidel, 1987.

[97] Z. Zeng, R. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5:976–990, 1993.

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & AR\ \ N\ \ |\ \ AR\ \ AP \\
AP & \rightarrow & AD\ \ N \\
VP & \rightarrow & V\ \ NP \\
N & \rightarrow & boy \mid tiger \mid fox \\
V & \rightarrow & saw \\
AR & \rightarrow & a \mid the \\
AD & \rightarrow & ferocious \mid sly
\end{array}
$$

Figure 1: Grammar for Declarative English Sentences.

$$
\begin{aligned}
S &\rightarrow NP\ \ VP\\
&\rightarrow AR\ \ N\, VP\\
&\rightarrow The\ N\ \ VP\\
&\rightarrow The\ boy\ VP\\
&\rightarrow The\ boy\ V\ \ NP\\
&\rightarrow The\ boy\ saw\ NP\\
&\rightarrow The\ boy\ saw\ AR\ \ AP\\
&\rightarrow The\ boy\ saw\ a\ AP\\
&\rightarrow The\ boy\ saw\ a\ AD\ \ N\\
&\rightarrow The\ boy\ saw\ a\ ferocious\ N\\
&\rightarrow The\ boy\ saw\ a\ ferocious\ tiger
\end{aligned}
$$

Figure 2: Generation of the Sentence – *The boy saw a ferocious tiger*

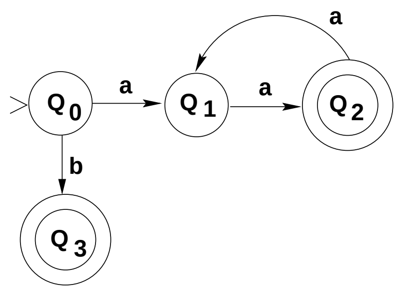

Figure 3: Deterministic Finite State Automaton.

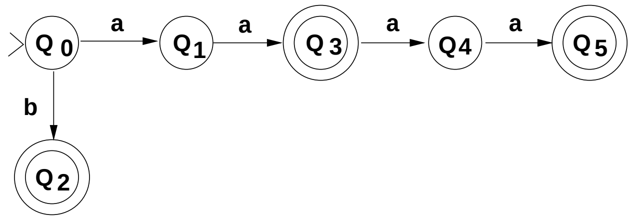

Figure 4: Prefix Tree Automaton.

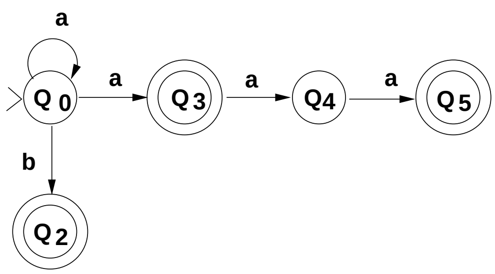

Figure 5: Quotient Automaton.

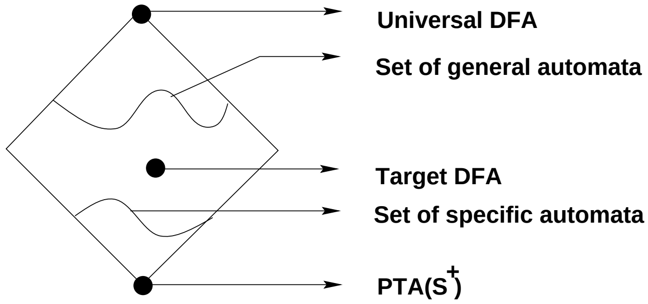

Figure 6: Bi-directional Search of the Lattice.

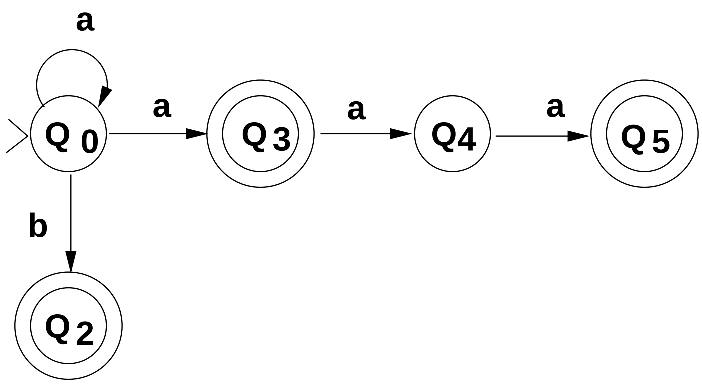

Figure 7: Quotient Automaton Obtained by Fusing Blocks 1 and 0 of the Current Hypothesis.

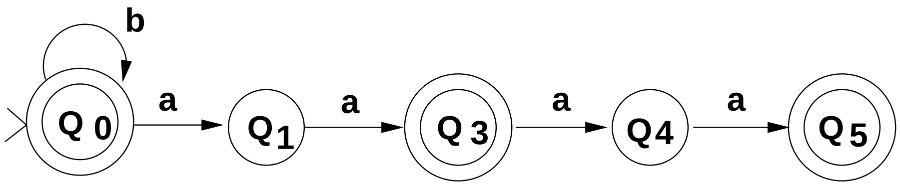

Figure 8: Quotient Automaton Obtained by Fusing Blocks 2 and 0 of the Current Hypothesis.

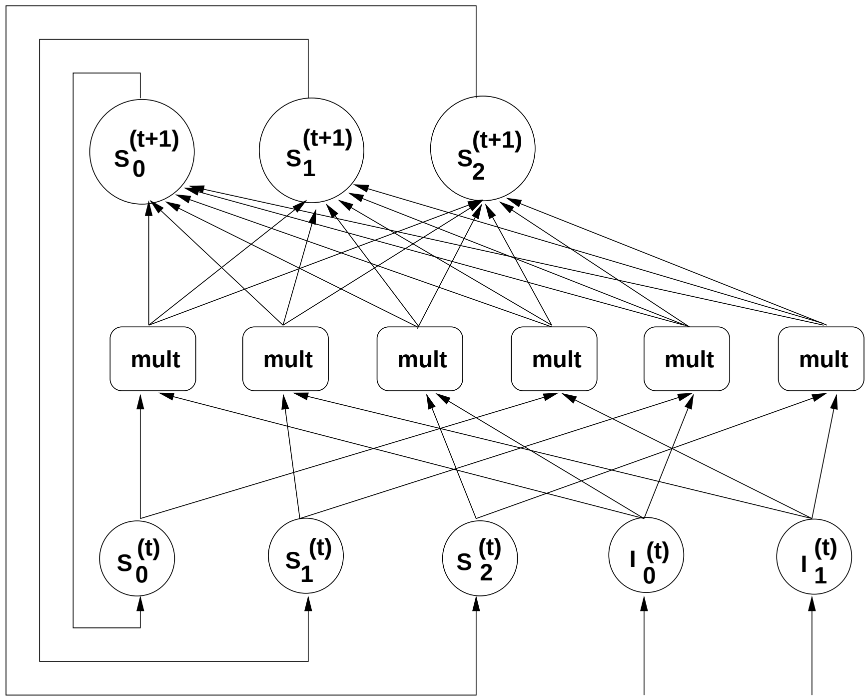

Figure 9: Recurrent Neural Network Architecture.

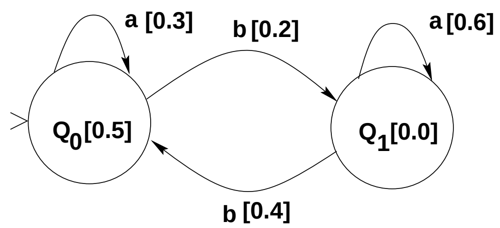

Figure 10: Deterministic Stochastic FSA.

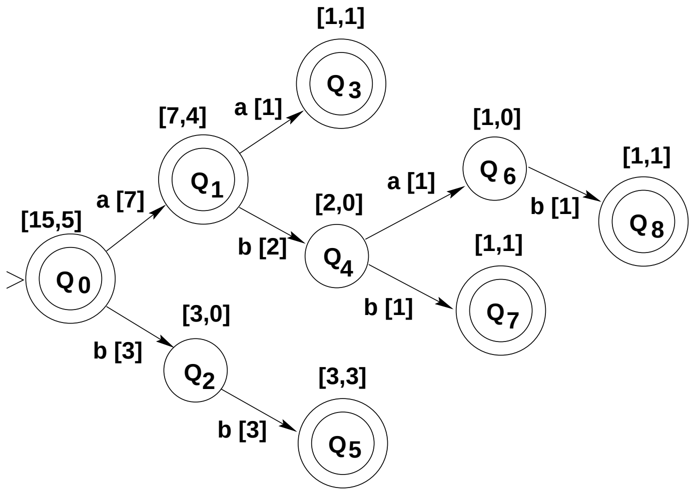

Figure 11: Prefix Tree Automaton.

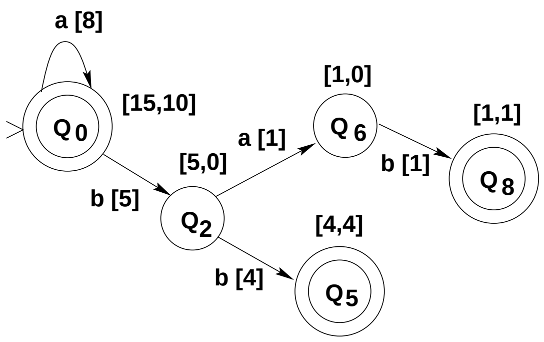

Figure 12: Deterministic Stochastic FSA after merging q_0 and q_1.

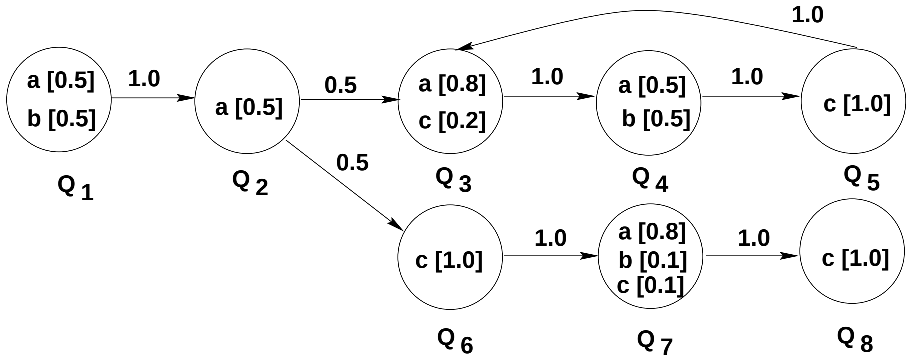

Figure 13: Hidden Markov Model.

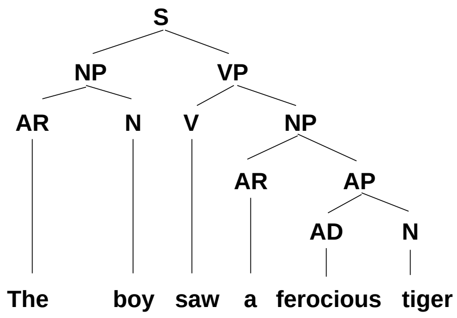

Figure 14: Parse Tree.

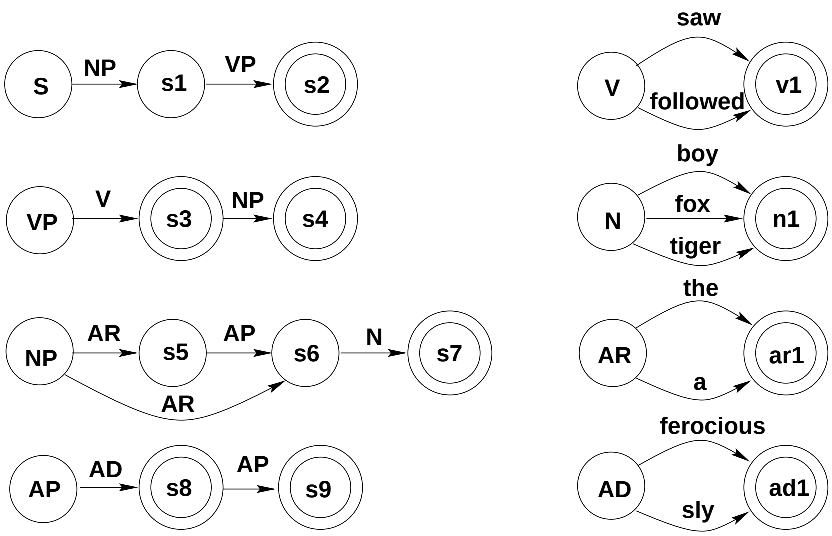

Figure 15: Recursive Transition Network.

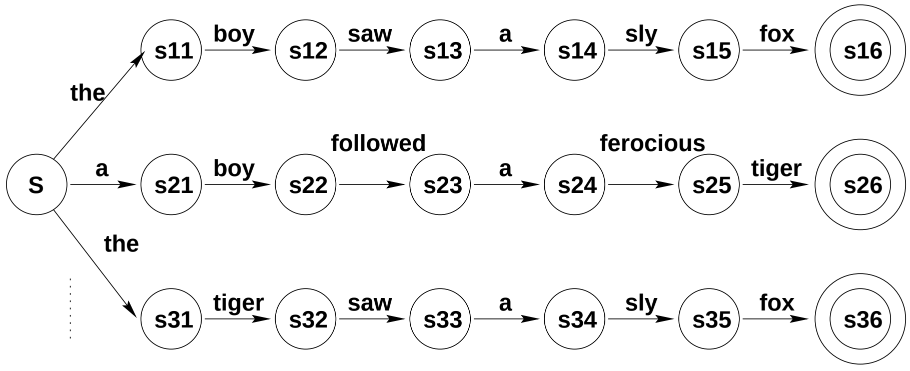

Figure 16: Initial Recursive Transition Network from Positive Sentences.

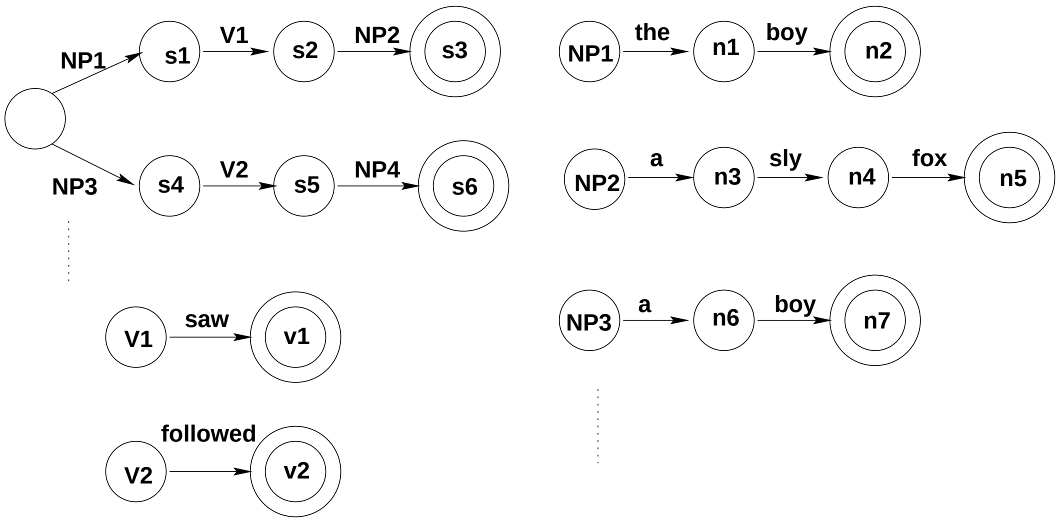

Figure 17: Creating Subnetworks.

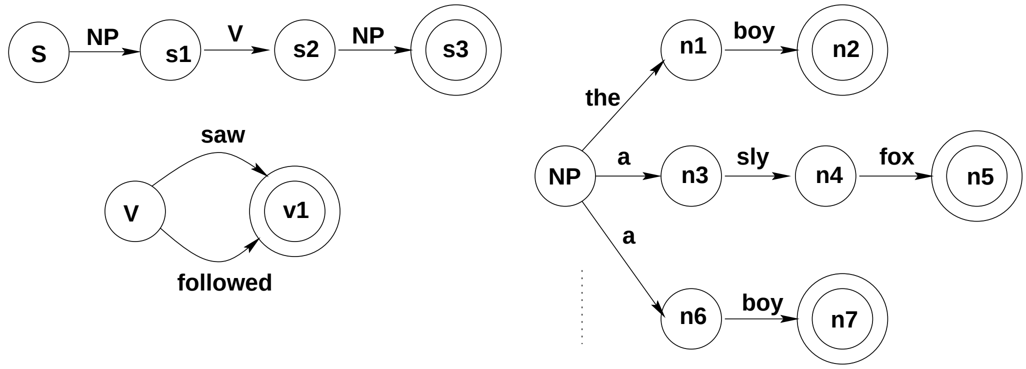

Figure 18: Merging Subnetworks.

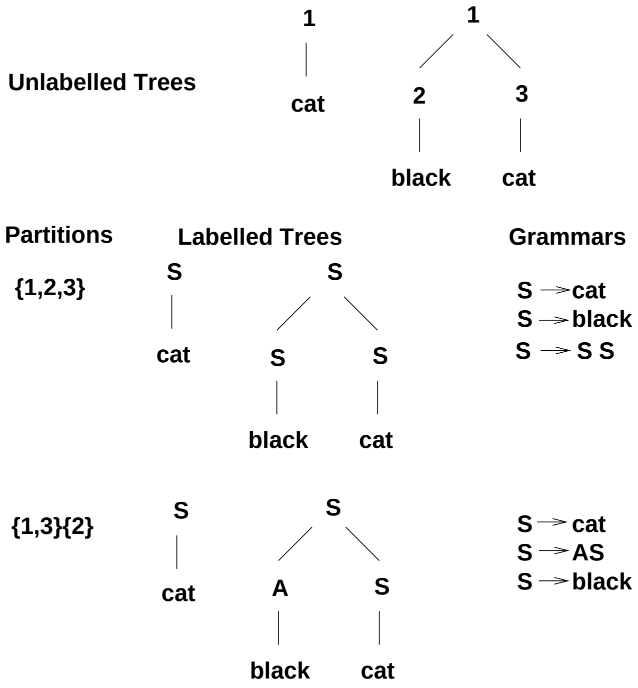

Figure 19: Derivational Version Space of CFG.