Recent advances of grammatical inference

Yasubumi Sakakibara*

Department of Information Sciences, Tokyo Denki University, Hatoyama, Hiki-gun,
Saitama 350-03, Japan

Abstract

In this paper, we provide a survey of recent advances in the field "Grammatical Inference" with a particular emphasis on the results concerning the learnability of target classes represented by deterministic finite automata, context-free grammars, hidden Markov models, stochastic context-free grammars, simple recurrent neural networks, and case-based representations.

1. Introduction

Loosely speaking, Grammatical Inference is an inductive inference problem where the target domain is a formal language and the representation class is a family of grammars. The learning task is to identify a "correct" grammar for the (unknown) target language, given a finite number of examples of the language. Grammatical Inference is a well-established research field in Artificial Intelligence as it dates back to the 60s. Gold [27] originated this study and introduced the notion of identification in the limit. His motivation for studying the problem is to construct a formal model of human language acquisition. Since his seminal work, there has been a remarkable amount of work to establish a theory of Grammatical Inference, to find effective and efficient methods for inferring grammars, and to apply those methods to practical problems. For example, Grammatical Inference is applied to natural language processing [19, 44] and computational biology [35, 53].

Grammatical Inference has been investigated, more or less independently, within many research fields, including machine learning, computational learning theory, pattern recognition, computational linguistics, neural networks, formal language theory, information theory, and many others. Recently, the international conference on Grammatical Inference has been established with an aim to bring together researchers from diverse fields and to bring about a stimulating interdisciplinary interaction between them. The first colloquium on Grammatical Inference was held in UK in April 1993, the second one in Spain in September 1994 [19], and the third one in France in September 1996 [41].

* E-mail: yasu@j.dendai.ac.jp.

0304-3975/97/$17.00 © 1997 — Elsevier Science B.V. All rights reserved

PII S0304-3975(97)00014-5
There are several excellent survey articles on the field of grammatical inference. An early survey on inductive inference is Angluin and Smith's article [14]. An early good introduction to Grammatical Inference is Miclet's article [40]. A recent extensive survey of the inference of deterministic finite automata is Pitt's paper [45].

Much of the recent research activities on Grammatical Inference have been stimulated by the new learning models proposed recently within computational learning theory framework: the query learning model of Angluin [10] and the PAC (probably approximately correct) learning model of Valiant [63]. These new models put much more emphasis on the computational efficiency of the inference algorithm. A good introduction to computational learning theory is Laird's paper [36], and a very recent survey of computational learning theory is Angluin's paper [12]. Thus Grammatical Inference is an old and new paradigm in artificial intelligence.

This paper, rather than being a thorough survey on the topic, is intended mainly as a review of the research carried out by the author and his colleagues (at the Machine Learning group at Fujitsu Laboratories Ltd.) and related work done at other institutions. The interested reader can consult the above cited survey papers. Also, consult the papers [14, 69] for the extensive surveys of inductive inference of indexable classes of formal languages.

We will begin with the problem of identifying deterministic finite automata (DFAs) from examples. DFAs are the bottom class of formal grammars in the Chomsky hierarchy, and the problem of identifying DFAs from examples has been studied quite extensively [14, 45]. We will pick up several interesting results on identifying DFAs: polynomial-time identification of DFAs from queries, identification of subclasses of DFAs from positive data, computationally hardness results, and identification from erroneous examples. In Section 4, we will consider the problem of identifying context-free grammars (CFGs) because the questions of whether there are analogous results held for context-free grammars would be more interesting and important. The results contain identification of CFGs from examples in the form of structured strings, polynomial-time reduction to identification of finite automata, and efficient identifications of several subclasses of CFGs. In Section 5, since stochastic modeling is very important for practical applications, we will consider the problem of identifying stochastic grammars. A stochastic grammar is obtained by specifying a probability for each production in a grammar. We will review some fundamental methods for training probabilistic parameters in the grammar based on expectation maximization (EM), and their applications to biological sequence analyses. In Section 6, we will see two special topics which use non-grammatical representations for Grammatical Inference or language learning. One is simple recurrent neural networks and the other is case-based representations.

2. The learning models

Within computational learning theory, there are three major established formal models for learning from examples or inductive inference: Gold's model of identification
in the limit [27], the query learning model by Angluin [10], and the PAC learning model by Valiant [63]. Each model provides a learning protocol and a criterion for the success of learning. Identification in the limit views learning as an infinite process and provides a learning model where an infinite sequence of examples of the unknown grammar $G$ is presented to the inference algorithm $M$ and the eventual or limiting behavior of the algorithm is used as the criterion of its success. A complete presentation of the unknown grammar $G$ is an infinite sequence of ordered pairs $(w, l)$ from $\Sigma^* \times \{0, 1\}$ such that $l = 1$ if and only if $w$ is generated by $G$, and such that every string $w$ of $\Sigma^*$ appears at least once as the first component of some pair in the sequence, where $\Sigma$ is the terminal alphabet. An inference algorithm $M$ takes as input initial segments of a complete presentation of $G$, and outputs a next conjecture. If for every complete presentation of the unknown grammar $G$, $M$ guesses a correct grammar which is equivalent to $G$ at some point, and never changes its guess after this, then $M$ is said to identify $G$ in the limit from complete presentations.

Angluin [10] has considered a learning situation in which a teacher is available to answer specific kind of queries on the unknown grammar $G$ and devised an elegant formulation of such a teacher and learner paradigm. In the query learning model, a teacher is a fixed set of oracles that can answer specific kinds of queries made by the inference algorithm on the unknown grammar $G$. For example, the following two types of queries are typical:

(i) **Membership.** The input is a string $w \in \Sigma^*$ and the output is "yes" if $w$ is generated by $G$ and "no" otherwise.

(ii) **Equivalence.** The input is a grammar $G'$ and the output is "yes" if $G'$ is equivalent to $G$ (i.e., $G'$ generates the same language as $G$) and "no" otherwise. If the answer is "no", a string $w$ in the symmetric difference of the language $L(G)$ generated by $G$ and the language $L(G')$ generated by $G'$ is returned.

For the equivalence query, the returned string $w$ is called a counterexample. In this setup, an inference algorithm $M$ runs with oracles for queries for the unknown grammar $G$, and eventually halts and outputs a correct grammar in a certain finite time. This is no longer a limiting criterion of learning. A membership query returns one bit of information. Nevertheless, it often plays an important role in efficient exact identification. For example, the class of DFAs can be identified in polynomial time using equivalence queries and membership queries while it cannot efficiently be identified from equivalence queries only [9, 11].

Valiant [63] has introduced the distribution-independent probabilistic model of learning from random examples, which is called probably approximately correct learning (PAC learning, in short). In the PAC learning model, we assume that random samples are drawn independently from the domain $\Sigma^*$ whose probability distribution $D$ may be arbitrary and unknown. The inference algorithm takes a sample as input and produces a grammar as output. The success of identification is measured by two parameters: the accuracy parameter $\varepsilon$ and the confidence parameter $\delta$, which are given as inputs to the inference algorithm. The error of a grammar $G'$ with respect to the unknown grammar $G$ is defined to be the sum of probabilities $D(w)$ of the strings $w$ in the symmetric
difference of $L(G')$ and $L(G)$ with respect to $D$. A successful inference algorithm is one that with high probability (at least $1 - \delta$) finds a grammar whose error is small (less than $\varepsilon$).

We measure the efficiency of the inference algorithm with respect to relevant parameters: the size of examples and the size of the unknown grammar. The Size of an example in the form of string is the length of the string. The Size of the unknown grammar is usually the number of states, in the case of finite automata, and the number of production rules, in the case of context-free grammars.

3. Learning finite automata

The study of the identifiability of deterministic finite automata is an excellent mean for studying a number of general aspects of inductive inference and grammatical inference [45]. In this section, we will review several important results and useful techniques related to computationally efficient identifications of deterministic finite automata. In order to get a complete picture concerning previous works for identification of deterministic finite automata, good references are the early work by Trakhtenbrot and Barzdin [62], the work of Wiehagen concerning the learnability from “good” examples [65], and an excellent survey by Pitt [45].

A deterministic finite (state) automaton (DFA) is defined by a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is an alphabet of input symbols, $\delta$ is the state-transition function $\delta : Q \times \Sigma \rightarrow Q$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. The language accepted by a DFA $A$ is denoted by $L(A)$.

3.1. Learning from representative samples

When trying to identify an unknown DFA $A = (Q, \Sigma, \delta, q_0, F)$ from examples, a useful information about $A$ is a representative sample $S$ of $A$, that is, a finite subset of $L(A)$ that exercises every live transition in $A$. Taking the set $R(S)$ of all prefixes of strings in $S$, for every live state $q$ of $A$, there must exist a string $u$ in $R(S)$ such that $\delta(q_0, u) = q$. Further, for every state $q$ and every transition $\delta(q, a)$ from $q$ where $a \in \Sigma$, there exists a string $va$ in $R(S)$ such that $\delta(q_0, v) = q$ and $\delta(q, a) = \delta(q_0, va) = q'$. Thus, every state and transition are represented by strings in $R(S)$. It remains to distinguish two states $q_u$ and $q_v$ represented by two strings $u$ and $v$ in $R(S)$, i.e., $q_u = \delta(q_0, u)$ and $q_v = \delta(q_0, v)$, if $q_u$ and $q_v$ are different states in $A$. Angluin [7] has given an efficient procedure to solve this problem using membership queries.

**Theorem 1** (Angluin [7]). The class of deterministic finite automata can be identified in polynomial time from a representative sample and using membership queries.

3.2. Learning with teachers

Angluin [9] has considered a learning protocol which is based on what is called “minimally adequate teacher”. This teacher can answer two types of queries about the
unknown DFA $A$ made by an inference algorithm: membership query and equivalence query. Angluin [9] has shown that equivalence queries compensate for the lack of representative samples, and presented an efficient inference algorithm for identifying DFAs using equivalence and membership queries.

**Theorem 2** (Angluin [9]). The class of deterministic finite automata can be identified in polynomial time using equivalence queries and membership queries.

The important data structure used in Angluin’s algorithm is called an observation table. An observation table is a two-dimensional matrix with rows and columns labelled by strings. The entry is 0 or 1, and the intended interpretation is that the entry for row $s$ and column $e$ is equal to 1 if and only if the string $s \cdot e$ is accepted by the unknown automaton. The rows consist of two parts, the ones labelled by a nonempty prefix-closed set $S$ of strings and the others labelled by the set $S \cdot \Sigma$. Rows labelled by $S$ are the candidates for states of the automaton being constructed and rows labelled by $S \cdot \Sigma$ are used to construct the state-transition function. The columns labelled by a nonempty suffix-closed set $E$ of strings play a role of witnesses to distinguish the candidates for representing states. The observation table has $S = E = \{\varepsilon\}$ (the set of only the empty string) at the beginning of learning, and is augmented as the algorithm runs. Two specific observation tables are defined, which are called closed and consistent. When we have a closed, consistent observation table, we can construct the minimum DFA consistent with the data contained in the table in time polynomial in the size of the table. The algorithm is going to find a closed, consistent observation table by asking membership queries to fill the entries. It has been shown in [9] that the algorithm asks at most $O(mn^2)$ membership queries and $n - 1$ equivalence queries, and eventually terminates and outputs the minimum DFA which is equivalent to the unknown DFA, where $m$ is the maximum length of any counterexample returned by the teacher during the running of the algorithm and $n$ is the number of states in the minimum DFA equivalent to the unknown DFA. The idea of the observation table is also related to the state characterization matrix by Gold [28].

Yokomori [67] has studied efficient identification of non-deterministic finite automata from equivalence and membership queries.

3.3. Learning from positive data

One interesting and important topic on Gold’s framework of identification in the limit for language learning is identification from positive data. A positive presentation of the unknown DFA $A$ is any infinite sequence of examples such that the sequence contains all and only the strings in the language $L(A)$. Gold [27] has shown that there is a fundamental, important difference in what could be learned from positive versus complete presentations, and shown a negative result that no “superfinite” class of languages can be identified in the limit from positive presentation. A class of languages is called superfinite if it contains all the finite languages and at least one infinite language. Since the class of regular languages is superfinite, we need to
restrict DFAs somehow to subclasses to establish identifiability results from positive presentation.

The problem is to avoid "overgeneralization", which means guessing a language that is a strict superset of the unknown language. Angluin [8] has introduced a series of subclasses of DFAs, called \( k \)-reversible automata for \( k = 0, 1, 2, \ldots \), and shown that the existence of characteristic samples is sufficient for identification from positive presentation (to avoid overgeneralization) for \( k \)-reversible automata and there exist such characteristic samples for the class of \( k \)-reversible automata. A characteristic sample of a \( k \)-reversible automaton \( A \) is a finite sample \( S \subseteq L(A) \) such that \( L(A) \) is the "smallest" \( k \)-reversible language that contains \( S \) with respect to set inclusion. It turns out that any characteristic sample is a representative sample for \( k \)-reversible automata.

As we have seen in Section 3.1, a representative sample provides enough information for reconstructions of states and state transitions. By utilizing the structural properties specific to \( k \)-reversible automata, we could accomplish the main task of state distinctions in identifying \( k \)-reversible automata without the use of membership queries. For example, a zero-reversible automaton is a DFA such that it has at most one final state and no two edges entering any state are labeled with the same symbol. Given a representative sample \( S \) for the unknown zero-reversible automaton, we construct the prefix tree automaton \( A' \) that precisely accepts the set \( S \), and then merge states in \( A' \) to satisfy the conditions for zero-reversible automata.

**Theorem 3** (Angluin [8]). The class of \( k \)-reversible automata, for \( k = 0, 1, 2, \ldots \), can be identified in the limit from positive presentation.

Furthermore, the inference algorithm updates a conjecture in time polynomial in the size of the inputs.

Another interesting class of DFAs which can be identified in the limit from positive presentation is the class of strictly deterministic automata investigated by Yokomori [68]. A strictly deterministic automaton is a DFA such that the set of labels \( W \) for state-transition edges is extended to be a finite subset of strings over \( \Sigma \), each edge has the unique label (no same label is attached to different edges), and for each symbol \( a \in \Sigma \) there is at most one label in \( W \) starting with \( a \).

**Theorem 4** (Yokomori [68]). The class of strictly deterministic automata can be identified in the limit from positive presentation.

An inference algorithm can be constructed so that it not only runs in time polynomial in \( m \) to update the conjecture, where \( m \) is the maximum length of all positive examples provided so far, but also makes at most a polynomial number of implicit errors of prediction in \( m \) and \( n \), where \( n \) is the size of the unknown strictly deterministic automaton. Pitt [45] has proposed the definition of implicit errors of prediction that after seeing \( i \)th example in the presentation, the inference algorithm \( M \) is said to make an implicit error of prediction at step \( i \) if the conjecture output by \( M \) is not consistent with the \((i + 1)\)th example.
Other interesting topics and results on identification from positive presentation which may not be directly related to DFAs are Angluin’s characterization of identifiability from positive presentation [6], Angluin’s pattern languages [5], Koshiba’s extension to typed pattern languages [34], Shinohara’s general result for identifiability from positive presentation [58], and Oncina et al.’s subsequential transducers [43].

3.4. Hardness results

There are many computationally hardness results related to identifying DFAs. Gold [28] has shown that the problem of finding a DFA with a minimum number of states consistent with a given finite sample of positive and negative examples is NP-hard. This result is generally interpreted as indicating that even a very simple case of grammatical inference, identifying DFAs from positive and negative examples, is computationally intractable. Further, Pitt and Warmuth [46] have proven a stronger result, namely that it is NP-hard to find a DFA of at most \( n(1-\varepsilon)\log \log n \) states consistent with a given finite sample of positive and negative examples for any constant \( \varepsilon > 0 \), where \( n \) is the number of states of a minimum DFA consistent with the given sample.

Angluin [11] has shown negative results for efficient identifications of various classes of grammars from equivalence queries only. She has developed the useful technique of “approximate fingerprints” to obtain negative results for identification from equivalence queries only. As applications of the technique, she has shown that there is no polynomial-time algorithm using only equivalence queries that identifies the class of DFAs, nondeterministic finite automata, context-free grammars, or disjunctive or conjunctive normal form boolean formulas.

3.5. Learning from erroneous examples

In practice, it is natural to assume that the examples may contain some noise. There are fewer works to study the effect of noise on learning from queries in the Valiant’s probabilistic framework of PAC-learnability.

Sakakibara [51] has defined a benign model for errors in the responses to membership queries where answers to queries are subject to random independent noise (i.e., for each query there is some independent probability to receive an incorrect answer and these errors are not persistent), and shown that these errors can be effectively removed by repeating the query until the confidence in the correct answer is high enough.

Ron and Rubinfeld [49] have considered a model of persistent noise in membership queries in which a fixed but randomly chosen fraction of membership queries are answered incorrectly but any additional query on the same string is replied consistently with the same incorrect answer when queried again. They have shown by modifying Angluin’s algorithm (Theorem 2) for identifying DFAs using equivalence and membership queries that DFAs can be learned in polynomial time from membership queries with persistent noise under the uniform distribution on inputs.

Sakakibara and Siromoney [56] have studied a noise model which is specific to language learning where the examples are corrupted by purely random errors affecting only
the strings (and not the labels). They have considered three types of errors on strings, called \textit{EDIT operation errors}. EDIT operations consist of "insertion", "deletion", and "change" of a symbol in a string. They have shown efficient identification from random examples with EDIT noise for a small subclass of regular languages defined by containment decision lists, a variant of \textit{decision list} \cite{48} to represent languages.

4. Learning context-free grammars

As we have seen in the previous sections, there has been extensive research into the problem of identifying DFAs from examples. The question of whether there are analogous results for context-free grammars is interesting and important simply because context-free grammars are more expressive.

A \textit{context-free grammar} (CFG) is defined by a quadruple $G = (N, \Sigma, P, S)$, where $N$ is an alphabet of \textit{nonterminal symbols}, $\Sigma$ is an alphabet of \textit{terminal symbols} such that $N \cap \Sigma = \emptyset$, $P$ is a finite set of production rules of the form $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in (N \cup \Sigma)^*$, and $S$ is a special nonterminal called the \textit{start symbol}. The \textit{language generated} by a CFG $G$ is denoted $L(G)$.

Angluin \cite{11} has shown that the whole class of CFGs cannot be identified in polynomial time using equivalence queries only. Furthermore, Angluin and Kharitonov \cite{13} have shown that the problem of identifying the class of CFGs from membership and equivalence queries is computationally as hard as the cryptographic problems for which there is currently no known polynomial-time algorithm (e.g., inverting RSA encryption, or factoring Blum integers). Despite these negative results, we will present in the following sections several positive results for identifying the whole class of CFGs with additional information or identifying subclasses of CFGs efficiently.

4.1. Learning from structural information

We consider an identification problem for CFGs where, besides given examples, some additional information is available for the inference algorithm. A useful (and maybe reasonable) information would be information on the grammatical structure of the unknown CFG. We assume example presentations in the form of strings with grammatical structure. Levy and Joshi \cite{38} have already suggested the possibility of efficient grammatical inferences in terms of strings with grammatical structure.

A string with grammatical structure, called a \textit{structured string} or a \textit{structural description} (of string), is a string with some parentheses inserted to indicate the shape of the derivation tree of a CFG, or equivalently an unlabeled derivation tree of the CFG, that is, a derivation tree whose internal nodes have no labels (see Fig. 1). It is known that the set of derivation trees of a CFG constitutes a rational set of trees, where a \textit{rational set} of trees is a set of trees which can be recognized by some tree automaton. Further, the set of unlabeled derivation trees of a CFG also constitutes a rational set of trees. Based on these observations, the problem of identifying CFGs from structured strings is reduced to the problem of identifying tree automata.
Sakakibara [50] has shown by extending Angluin’s inference algorithm (Theorem 2) for DFAs to tree automata that the class of CFGs can be identified in polynomial time using structural membership queries and structural equivalence queries.

Theorem 5 (Sakakibara [50]). The class of context-free grammars can be identified in polynomial time using structural equivalence queries and structural membership queries.

Let $D(G)$ denote the set of derivation trees of a CFG $G$ and $s(D(G))$ denote the set of unlabeled derivation trees (structured strings) of $G$. A structural membership query is a membership query for a structured string to ask whether it is generated by the unknown CFG $G$, and a structural equivalence query returns “yes” if a queried CFG $G'$ is structurally equivalent to the unknown CFG $G$ and returns “no” with a counterexample otherwise, that is, a structured string in the symmetric difference of $s(D(G))$ and $s(D(G'))$.

As we have seen in the previous section, Angluin’s algorithm for identifying DFAs uses the observation table to organize the information about a finite collection of strings with the indication whether they are strings accepted by the unknown DFA. We extend the observation table to the one for tree automata. The extended observation table has rows labelled by structured strings and columns labelled by structured strings with a special symbol. The intended interpretation is that the entry for row $s$ and column $e$ is equal to 1 if and only if the structured string of the concatenation of $s$ and $e$ is a structured string generated by the unknown grammar $G$.

Since the class of CFGs is superfinite, Gold’s negative result [27] on identifiability from positive presentation implies that the class of CFGs cannot be identified in the limit from positive presentation. Sakakibara [52] has demonstrated information on the grammatical structure of the unknown CFG could help the inference. He has shown that there exists a class of CFGs, called reversible context-free grammars, which can be identified in the limit from positive presentations of structured strings, that is, all and only unlabeled derivation trees of the unknown CFG, and shown that the reversible context-free grammar is a normal form for CFGs, that is, reversible context-free grammars can generate all the context-free languages.
A reversible context-free grammars is a CFG $G = (N, \Sigma, P, S)$ such that (1) $A \rightarrow \alpha$ and $B \rightarrow \beta$ in $P$ implies that $A = B$ and (2) $A \rightarrow \alpha \beta$ and $A \rightarrow \alpha C \beta$ in $P$ implies that $B = C$, where $A$, $B$, and $C$ are nonterminals, and $\alpha, \beta \in (N \cup \Sigma)^*$. 

**Theorem 6** (Sakakibara [52]). The class of reversible context-free grammars can be identified in the limit from positive presentation of structured strings provided that the structured strings are generated with respect to a reversible context-free grammar for the unknown context-free language.

Since the inference algorithm for reversible context-free grammars is an extension of Angluin's inference algorithm which identifies zero-reversible automata (Theorem 3), the algorithm updates a conjecture in time polynomial in the size of the inputs. Note that the above result does not imply that the whole class of CFGs can be identified from positive presentation of structured strings.

Here we give an informal example of learning process to illustrate the inference algorithm developed for learning reversible context-free grammars [52]. The learning algorithm takes as input a finite set $S_a$ of structured strings. First the algorithm constructs a context-free grammar $G_0$ that precisely generates the set $S_a$, that is, $s(D(G_0)) = S_a$. Next the algorithm merges nonterminals and generalizes it to get a reversible context-free grammar $G$ such that

$$s(D(G)) = \min\{s(D(G')) \mid S_a \subseteq s(D(G'))\},$$

$G'$ is a reversible context-free grammar.

Suppose that the following set $S_a$ of structured strings is given to the learning algorithm.

$$S_a = \{ \langle a b \rangle \langle c \rangle, \langle a a b \rangle \langle c \rangle, \langle a b \rangle \langle c c \rangle \}.\$$

The algorithm constructs the following grammar $G_0$ such that $s(D(G_0)) = S_a$:

- $S \rightarrow A B$
- $A \rightarrow a b$
- $B \rightarrow c$
- $S \rightarrow C D$
- $C \rightarrow a C' b$
- $C' \rightarrow a b$
- $D \rightarrow c D'$
- $D' \rightarrow c$
- $S \rightarrow E F$
- $E \rightarrow a b$
- $F \rightarrow c F'$
- $F' \rightarrow c$. 


Then nonterminals $A$, $C'$, and $E$ in $G_0$ are merged and nonterminals $B$, $D'$, and $F'$ in $G_0$ are merged to satisfy the condition (1) in the definition of reversible context-free grammar:

\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow ab \\
B & \rightarrow c \\
S & \rightarrow CD \\
C & \rightarrow aA b \\
D & \rightarrow cB \\
S & \rightarrow AF \\
F & \rightarrow cB.
\end{align*}
\]

Again to satisfy the condition (1), nonterminals $D$ and $F$ are merged:

\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow ab \\
B & \rightarrow c \\
S & \rightarrow CD \\
C & \rightarrow aA b \\
D & \rightarrow cB \\
S & \rightarrow AD.
\end{align*}
\]

To satisfy the condition (2), nonterminals $B$ and $D$ are merged and nonterminals $A$ and $C$ are merged. Finally, the algorithm outputs the following reversible context-free grammar:

\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow a b \\
A & \rightarrow aA b \\
B & \rightarrow c \\
B & \rightarrow cB,
\end{align*}
\]

which generates the language $\{a^m b^m c^n \mid m, n \geq 1\}$.

A related early work to identifying CFGs from positive presentation of structured strings is Crespi-Reghizzi’s [20]. He has described a constructive method for identifying a subclass of CFGs, which is a different class from reversible CFGs, from positive samples of structured strings. His class of CFGs defines only a subclass of context-free languages, called noncounting context-free languages. Mäkinen [39] has refined Sakakibara’s inference algorithm for reversible CFGs to gain more efficiency, and also investigated a subclass of reversible CFGs, called type invertible grammars, that can
be identified from positive presentation of structured strings in time linear in the size of the inputs.

4.2. Reductions to finite-automata learning problems

A well-known technique often used to establish identifiability results is a reduction technique that reduces an inference problem to some other inference problem whose result is known. Takada [60] has shown that the inference problem for even linear grammars can be solved by reducing it to the one for DFAs, and presented a polynomial-time algorithm for the reduction. For example, we can identify the class of even linear grammars using equivalence and membership queries in polynomial time by employing Angluin's efficient algorithm for DFAs (Theorem 2) via reduction.

An even linear grammar is a CFG that has productions only of the form $A \rightarrow uBv$ or $A \rightarrow w$ such that $u$ and $v$ have the same length, where $A$ and $B$ are nonterminals and $u$, $v$, and $w$ are strings over $\Sigma$. Let $G = (N, \Sigma, P, S)$ be an even linear grammar. We write $x \xrightarrow{\gamma} y$ to mean that $y$ is derived from $x$ applying the production $\pi$ in $P$, where $x, y \in (N \cup \Sigma)^*$. We denote a derivation from $x_0$ to $x_k$ obtained by applying a sequence $\gamma = \pi_1\pi_2 \cdots \pi_k$ of productions by $x_0 \xrightarrow{\gamma} x_k$. $\gamma$ is called an associate word and a set of associate words is called a control set on $G$. The language generated by $G$ with a control set $C$ is defined by $L(G, C) = \{w \in \Sigma^* \mid S \xrightarrow{\gamma} w$ and $\gamma \in C\}$. It can be shown that there is a universal even linear grammar $G_U$ such that for any even linear grammar $G$, $L(G) = L(G_U, C)$ for some regular control set $C$.

**Theorem 7** (Takada [60]). The problem of identifying the class of even linear grammars is reduced to the problem of identifying the class of finite automata.

Note that the class of even linear languages properly contains the class of regular languages and is a proper subclass of context-free languages. By iteratively applying the above reduction technique, Takada [61] has further developed an infinite hierarchy of families of languages whose identification problems are reduced to the identification problem of DFAs.

4.3. Learning subclasses of context-free grammars

Because the whole class of CFGs seems to be hard to be identified efficiently without any additional information, there have been some attempts to design polynomial-time algorithms for identifying subclasses of CFGs from examples.

Ishizaka [32] has investigated a subclass of CFGs, called simple deterministic grammars, and gave a polynomial-time algorithm for exactly identifying it using equivalence and membership queries in terms of general CFGs. This inference algorithm may sometimes ask an equivalence query for a CFG which is not simple deterministic.

A CFG $G = (N, \Sigma, P, S)$ in 2-standard form is called simple deterministic if $A \rightarrow ax$ and $A \rightarrow a\beta$ in $P$ implies that $x = \beta$, where $A$ and $B$ are nonterminals, $a$ is a terminal, and $x, \beta \in (N \cup \Sigma)^*$. 
Theorem 8 (Ishizaka [32]). The class of simple deterministic grammars can be identified in polynomial time using equivalence queries and membership queries in terms of general context-free grammars.

Note that given any regular language \( L \), the language \( L# \) is simple deterministic, where \( # \) is a special symbol not in \( \Sigma \). In this sense, the class of simple deterministic languages properly contains the class of regular languages.

Yokomori [66] has considered a smaller class of simple deterministic grammars with the goal of finding a polynomial-time algorithm to identify it in the limit from positive presentation. A CFG \( G = (N, \Sigma, P, S) \) in Greibach normal form is called very simple if for each terminal symbol \( a \) in \( \Sigma \), there exists exactly one production rule starting with \( a \) (i.e., exactly one production rule of the form \( A \to ax \), where \( x \in (N \cup \Sigma)^* \)). He has shown that the class of very simple grammars can efficiently be identified in the limit from positive presentation, and this result has provided the first instance of language class containing non-regular languages that can be identified in the limit in polynomial time in a criterion proposed by Pitt [45], that is, the time for updating a conjecture is bounded by a polynomial in the size \( n \) of the unknown grammar and the sum of lengths of examples provided, and the number of implicit errors of prediction made by the inference algorithm is bounded by a polynomial in \( n \).

Theorem 9 (Yokomori [66]). The class of very simple grammars can be identified in the limit from positive presentation in polynomial time.

From this result, it immediately follows that the class of very simple grammars can be identified in polynomial time using only equivalence queries.

Related to identification of very simple grammars, Burago [18] has investigated the structurally reversible context-free grammars, and shown that the class of structurally reversible CFGs can be identified in polynomial time using equivalence queries and membership queries. A CFG is called structurally reversible if among all nonterminal strings that might derive a given terminal string, no one is an extension of the other. The class of structurally reversible CFGs is a subclass of CFGs and the class of structurally reversible context-free languages properly contains the class of very simple languages.

Other representation forms for languages which are not in the form of grammars sometimes help designing efficient inference algorithms. Fahmy and Biermann [23] have investigated identification of real time acceptors. The class of languages accepted by real time acceptors is a subclass of context-sensitive languages and incomparable with the class of context-free languages.

5. Learning stochastic grammars

Another major research topic in grammatical inference is stochastic modeling and training of stochastic grammars. Stochastic modeling has become increasingly important
for applications such as speech recognition, natural language processing, and biological sequence analysis. A stochastic grammar is obtained by specifying a probability for each production in a grammar. A stochastic grammar assigns a probability to each string which it derives and hence defines a probability distribution on the set of strings. Stochastic (probabilistic) automata are the probabilistic counterpart of finite automata that are known as hidden Markov models (HMMs) and very extensively used in speech recognition. Stochastic context-free grammars (SCFGs) is a superclass of and goes one step beyond hidden Markov models in the Chomsky hierarchy.

The problem of identifying stochastic grammars from examples has two aspects: determining the discrete structure (topology) of the grammar and estimating probabilistic parameters in the grammar. Based on the maximum likelihood criterion, efficient estimation algorithms for probabilistic parameters have been proposed. forward-backward algorithm for HMMs [47] and inside-outside algorithm for SCFGs [16,37]. The relative success of stochastic grammars in real tasks is due to the existence of these techniques for automatic estimation of probabilities and distributions. Both algorithms are iterative algorithms which are based on the expectation-maximization (EM) technique that increases the likelihood of the training sample in each step until a local maximum is reached. Therefore, the initialization in the iterative process is a crucial point since it affects the speed of convergence and the goodness of the results. On the other hand, finding an appropriate discrete structure of the grammar is a harder problem. In certain cases, it might be possible to consider the inference of the discrete structure as a result of the probability estimation process. For example, in the case of HMM, we start with a fully connected HMM, get a locally maximum estimation of probabilities, and obtain a structure of HMM by pruning out zero or low probability transitions. However, this method does not seem to be efficient. In fact, Abe and Warmuth [2] have shown a computationally hardness result for the inference of probabilistic automata.

In the remaining of this section, we will focus on probability estimation procedures for HMM and SCFG.

5.1. Hidden Markov models

A hidden Markov model (HMM) is defined by a 5-tuple \( A = (Q, \Sigma, T, \pi, O) \), where \( Q \) is a finite set of states, \( \Sigma \) is an alphabet of output symbols, \( T \) is a state transition probability distribution, \( O \) is an output symbol probability distribution, and \( \pi \) is an initial state distribution. Let \( Q = \{q_1, \ldots, q_n\} \). \( T \) is the set \( \{t_{ij} | 1 \leq i, j \leq n\} \) of state transition probabilities where \( t_{ij} \) is a state transition probability from state \( q_i \) to state \( q_j \) such that \( \sum_{j=1}^{n} t_{ij} = 1 \). \( O \) is the set \( \{o_j(a) | 1 \leq j \leq n, a \in \Sigma\} \) of output symbol probabilities where \( o_j(a) \) is a probability to output \( a \) at state \( q_j \) such that \( \sum_{a \in \Sigma} o_j(a) = 1 \), and \( \pi \) is the set \( \{\pi_i | 1 \leq i \leq n\} \) of initial state probabilities where \( \pi_i \) is the probability to start at state \( q_i \) such that \( \sum_{i=1}^{n} \pi_i = 1 \).

Given a HMM \( A \), there are three basic problems for dealing with \( A \): given a string \( w = a_1 \cdots a_m \),
(1) calculate $Pr(w|A)$, the probability of the string $w$ generated by $A$.

(2) find the most probable path $s = q_{i_1} \cdots q_{i_l}$ of states to maximize $Pr(s|w,A)$.

(3) estimate the parameters in $A$ to maximize $Pr(w|A)$.

These problems can be solved efficiently using dynamic programming techniques [47]. A polynomial-time algorithm for solving the second problem is known as Viterbi algorithm, and a polynomial-time algorithm for the third problem is known as Forward-Backward (Baum-Welch) algorithm.

To solve the problem (1), the probability of a path $q_{i_1} \cdots q_{i_m}$ of states generating a string $w = a_1 \cdots a_m$ can be calculated as the product of the probabilities of the initial state, the output symbols, and the state transitions to generate the string along the path. The probability of a string $w$ is the sum of probabilities over all possible paths that $A$ could use to generate $w$, written as follows:

$$Pr(w|A) = \sum_{\text{all paths } q_{i_1} \cdots q_{i_m}} Pr(w, q_{i_1} \cdots q_{i_m} | A)$$

$$= \sum_{\text{all paths } q_{i_1} \cdots q_{i_m}} \pi_{i_1} \cdot o_{i_1}(a_1) \cdot t_{i_1,i_2} \cdots o_{i_{m-1}}(a_{m-1}) \cdot t_{i_{m-1},i_m} \cdot o_{i_m}(a_m)$$

Efficiently computing $Pr(w|A)$ presents a problem because the number of possible paths for $w$ is exponential in the length of the string. To solve the problem, we consider the forward variable $\alpha_k(q_i)$ defined as $\alpha_k(q_i) = Pr(a_1 \cdots a_k, q_i | A)$, i.e., the probability of the initial segment $a_1 \cdots a_k$ of the string $w$ and state $q_i$ at time $k$. The probability $\alpha_k(q_i)$ can be calculated inductively as follows:

(i) Initialization:

$$\alpha_1(q_i) = \pi_i o_i(a_1)$$

(ii) Induction:

$$\alpha_{k+1}(q_j) = \left( \sum_{i=1}^{n} \alpha_k(q_i) t_{ij} \right) o_j(a_{k+1})$$

(iii) Termination:

$$Pr(w|A) = \sum_{i=1}^{N} \alpha_m(q_i)$$

To solve the problem (2), we can compute the most probable path efficiently using a variant of the above procedure for calculating $Pr(w|A)$. To obtain the most probable path for the string $w$, we calculate

$$\max_{\text{paths of states } s} Pr(s, w|A).$$

To solve problem (3), the forward–backward algorithm is an EM (expectation maximization) algorithm which finds parameters in the HMM $A$, i.e., state transition probabilities $t_{ij}$, output symbol probabilities $o_j(a)$, and initial state probabilities $\pi$, to maximize $Pr(w|A)$. It proceeds as follows:

(i) Let $A_{old}$ be an initial guess for the parameters.
(ii) Based on $A_{old}$ and the given string $w$,

(a) For each pair $q_i, q_j$ of states, estimate the fraction of times a transition is made from $q_i$ to $q_j$ among all transitions out of $q_i$. In $A_{new}$, set $t_{ij}$ to this value.

(b) For each state $q_j$ and output symbol $a$, estimate the fraction of times that $a$ is output in state $q_j$. In $A_{new}$, set $o_j(a)$ to this value.

(iii) Set $A_{old} = A_{new}$ and iterate starting at step 2 until there are no significant changes in $A_{old}$.

The forward–backward algorithm for HMMs is very efficient because of the use of dynamic programming techniques, including the forward procedure and the symmetric "backward" procedure. Each iteration in the algorithm increases $\Pr(w|A)$, but the algorithm can still get caught in local maxima. The algorithm is easily extended to handle a set of strings, but the algorithm suffers from the usual problems with maximum likelihood estimates: when it observes something 0 times, it sets the probability to 0.

5.2. Stochastic context-free grammars

A stochastic context-free grammar (SCFG) $G$ consists of a set of nonterminal symbols $N$, a terminal alphabet $\Sigma$, a set $P$ of production rules with associated probabilities, and the start symbol $S$. The associated probability for every production $A \rightarrow \alpha$ in $P$ is denoted $\Pr(A \rightarrow \alpha)$, and a probability distribution exists over the set of productions which have the same nonterminal on the left-hand sides.

The three basic problems to deal with SCFGs which are the same as in HMMs can be solved efficiently. The first two problems, calculating the probability $\Pr(w|G)$ of a given string $w$ assigned by a SCFG $G$ and finding the most likely derivation tree of $w$ by $G$, can be solved using dynamic programming methods analogous to the Cocke–Kasami–Young or Early parsing methods [4]. There is a standard method for estimating the parameters of an SCFG (i.e., the probabilities of the productions) from a set of training strings. This procedure is known as the inside-outside algorithm [37]. Just like the forward–backward algorithm for HMMs, this procedure is an expectation-maximization (EM) method for obtaining maximum likelihood of the grammar's parameters. However, it requires the grammar to be in Chomsky normal form, which is inconvenient to handle in many practical problems (and requires more nonterminals). Further, it takes time at least proportional to $n^3$, whereas the forward–backward procedure for HMMs takes time proportional to $n^2$, where $n$ is the length of the string $w$. There are also many local maxima in which the method can get caught.

To avoid such problems, Sakakibara et al. [53] have developed a new method for training SCFGs that is a generalization of the forward–backward algorithm to tree grammars and which is more efficient than the inside–outside algorithm. The new algorithm, called Tree-Grammar EM, requires structured strings as training examples. This algorithm uses a similar idea to identification of CFGs from structured strings shown in Section 4.1. Since information on the grammatical structure is given
explicitly in training strings, Tree-Grammar EM does not have to (implicitly) consider all possible derivations of the training strings when reestimating the grammar's parameters, as the inside–outside algorithm must do. This reduces the time complexity to a time proportional to \( n \) per training string of length \( n \), and hence may be practical on longer strings. Tree-Grammar EM also tends to converge faster because each training structured string is much more informative.

We describe this new algorithm. A tree is a rooted, directed, connected acyclic finite graph in which the direct successors of any node are linearly ordered from left to right. The predecessor of a node is called the parent; the successor, a child; and a child of the parent, a sibling. We assume all internal nodes in \( t \) are numbered from 1 to \( N \) (the number of internal nodes) in some order. For an internal node \( n \) (\( 1 \leq n \leq N \)), let \( t/n \) denote the subtree of \( t \) with root \( n \) and let \( t\setminus n \) denote the tree obtained by removing a subtree \( t/n \) from \( t \). The probability of any structured string \( t \) given by a SCFG \( G = (N, \Sigma, P, S) \) is efficiently calculated using a dynamic programming technique, as is done with the forward algorithm in HMMs.

Let the quantity \( \text{in}_n(X) \) define the probability of the subtree \( t/n \) given that the nonterminal \( X \) is assigned to node \( n \) and given grammar \( G \), for all nonterminals \( X \) and all nodes \( n \) such that \( 1 \leq n \leq N \). We can calculate \( \text{in}_n(X) \) inductively as follows:

(i) **Initialization:** \( \text{in}_n(a) = 1 \), for all leaf nodes \( n \) and all terminals \( a \in \Sigma \).

(ii) **Induction:**

\[
\text{in}_m(X) = \sum_{Y_1, \ldots, Y_k \in (N \cup \Sigma)} \text{in}_{n_1}(Y_1) \cdots \text{in}_{n_k}(Y_k) \cdot \Pr(X \to Y_1 \cdots Y_k).
\]

for all nonterminals \( X \), all internal nodes \( m \) and all \( m \)'s children nodes \( n_1, \ldots, n_k \).

(iii) **Termination:** For the root node \( n \) and the start symbol \( S \),

\[
\Pr(t|G) = \text{in}_n(X).
\]

This calculation enables us to estimate the new parameters of a SCFG in time proportional to the square of the number of nonterminals in the grammar multiplied by the size of the given structured string. We need one more quantity, \( \text{out}_n(X) \), which defines the probability of \( t\setminus n \) given that the nonterminal \( X \) is assigned to node \( n \) and given grammar \( G \), which we can obtain similarly. Here we present a version of the EM method to estimate the parameters of a SCFG from a training structured string. The inner loop of our Tree-Grammar EM algorithm proceeds as follows:

(i) An initial grammar is created by assigning values to the production probability \( \Pr(X \to Y_1 \cdots Y_k) \) for all productions \( X \to Y_1 \cdots Y_k \) in \( P \). The current grammar is set to this initial grammar.

(ii) Using the current grammar, the values \( \text{in}_n(X) \) and \( \text{out}_n(X) \) for each nonterminal \( X \) and each node \( n \) of the structured string are calculated in order to get a new
estimate of each production probability,

$$Pr'(X \rightarrow Y_1 \cdots Y_k) = \sum_{m} \frac{\text{nodes} \cdot \text{out}_m(X) \cdot Pr(X \rightarrow Y_1 \cdots Y_k) \cdot \text{in}_{m_1}(Y_1) \cdots \text{in}_{m_k}(Y_k) / Pr(t|G)}{\text{norm}}$$

where $G$ is the old grammar and “norm” is the appropriate normalizing constant such that $\sum_{Y_1, \cdots, Y_k} Pr'(X \rightarrow Y_1 \cdots Y_k) = 1$.

(iii) A new current grammar is created by replacing $Pr(X \rightarrow Y_1 \cdots Y_k)$ with the reestimated probability $Pr'(X \rightarrow Y_1 \cdots Y_k)$.

(iv) Steps 2 and 3 are repeated until the parameters of the current grammar change only insignificantly.

Sakakibara et al. [53] have also modified the algorithm to train SCFGs even from (unstructured) strings. If only unstructured training strings are available, we iteratively estimate the structure of the training strings as follows:

(i) Start with an initial grammar and parse the training strings to obtain a set of partially structured strings.

(ii) Estimate a new SCFG using the partially structured strings and the estimation algorithm Tree-Grammar EM.

(iii) Use the trained grammar to obtain more accurately structured training strings.

(iv) Repeat steps 2 and 3 until finding the structures stabilizes.

In natural language processing, Pereira and Schabes [44] have developed a similar method to Tree-Grammar EM for training SCFGs from bracketed sentences to incorporate linguistic information. Their method utilizes phrase bracketing information during the estimation process of the inside-outside algorithm to get a linguistically-motivated maximum.

Stolcke and Omohundro [59] have considered identification of a discrete structure of the stochastic grammar. They have proposed efficient heuristic methods for finding the topology of HMM and for finding an appropriate set of production rules of SCFG based on Bayesian criterion, and shown some experimental results.

5.3. Applications to molecular sequence analyses

Both computer science and molecular biology are evolving rapidly as disciplines, and predicting the structure of macromolecules by theoretical or experimental means remains a challenging problem. The increasing numbers of DNA, RNA and protein sequences yielded in recent years highlight a growing need for developing new approaches in computational biology such as hidden Markov models [35] and other approaches [31]. Determining common or consensus patterns among a family of sequences, producing a multiple sequence alignment, discriminating members of the family from non-members and discovering new members of the family will continue to be some of the most important and fundamental tasks in mathematical analysis and comparison of macromolecular sequences. In this section, we show an application of stochastic context-free grammars (SCFGs) to the problems of statistical modeling,
Fig. 2. This set of productions $P$ generates RNA sequences with a certain restricted structure. $S_0, S_1, \ldots, S_{13}$ are nonterminals; $A$, $U$, $G$ and $C$ are terminals representing the four nucleotides.

multiple alignment, discrimination and prediction of the secondary structure of RNA families [53].

In RNA, the nucleotides adenine ($A$), cytosine ($C$), guanine ($G$) and uracil ($U$) interact in specific ways to form characteristic secondary-structure motifs such as helices, loops and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular $A$-$U$ and $G$-$C$ Watson–Crick pairs. Such base pairs constitute the so-called biological palindromes in the genome. As in the elegant work of Searls [57], we view the strings of characters representing pieces of DNA, RNA and protein as sentences derived from a formal grammar. Searls has shown base pairing in RNA can be described by a context-free grammar. In particular, the productions of the forms $S \rightarrow A S U$, $S \rightarrow U S A$, $S \rightarrow G S C$ and $S \rightarrow C S G$ describe the structure in RNA due to Watson–Crick base pairing. Using productions of this type, a CFG can specify the language of biological palindromes. For example, application of productions in the grammar shown in Fig. 2 could generate the RNA sequence CAUCAGGGAAGAUCUCUUG by the following derivation:

$$S_0 \Rightarrow S_1 \Rightarrow C S_2 G \Rightarrow C A S_3 U G \Rightarrow C A S_4 S_5 U G$$
$$\Rightarrow C A U S_5 A S_6 U G \Rightarrow C A U C S_6 G A S_7 U G$$
$$\Rightarrow C A U C A G G A S_8 S_9 U G \Rightarrow C A U C A G G A A S_{10} U U G$$
$$\Rightarrow C A U C A G G A A G A S_{11} C U U G$$
$$\Rightarrow C A U C A G G A A G A S_{12} U C U U G$$
$$\Rightarrow C A U C A G G A A G A U S_{13} U C U U G$$
$$\Rightarrow C A U C A G G A A G A U S_{14} U C U U G.$$

A derivation can be arranged in a tree structure called a derivation tree (Fig. 3, left). A derivation tree represents the syntactic structure of a sequence produced by a grammar. For an RNA sequence, this syntactic structure corresponds to the physical secondary structure (Fig. 3, right).
Fig. 3. A derivation tree (left) generated by a simple CFG for RNA molecules and the physical secondary structure (right) of the RNA sequence which is a reflection of the derivation tree.

Searls' original work [57] argues the benefits of using CFGs as models for RNA folding, but does not discuss stochastic grammars or methods for creating the grammar from training sequences. Sakakibara et al. [53] provide an effective method for building a stochastic context-free grammar (SCFG) to model a family of RNA sequences.

We assess the trained grammar's ability to perform three tasks: to discriminate transfer RNA (tRNA) sequences from non-tRNA sequences, to produce multiple alignments and to ascertain the secondary structure of new sequences. The results show that after having been trained on tRNA sequences, the grammar can discern general tRNA from similar-length RNA sequences of other kinds, can find secondary structure of new tRNA sequences, and can produce multiple alignments of large sets of tRNA sequences.

Comparative analyses of two or more protein or nucleic-acid sequences have been used widely in detection and evaluation of biological similarities and evolutionary relationships. Several methods for producing these multiple sequence alignments have been developed, most based on dynamic programming techniques (for example, see works by Waterman [64]). However, when RNA sequences are to be aligned, both the primary and secondary structure need to be considered since generation of a multiple sequence alignment and analysis of folding are mutually dependent exercises. As shown in Fig. 4, our learned grammar has successfully produced a very accurate multiple alignments for some family (gene) of molecular sequences, called tRNA.

Related to the above work, Krogh et al. [35] have applied HMMs to the problems of statistical modeling, database searching and multiple sequence alignment of protein families and protein domains. These methods are demonstrated on the globin family, the protein kinase catalytic domain, and the EF-hand calcium binding motif. In each case, the parameters of an HMM are estimated from a training set of unaligned sequences. The HMM produces multiple alignments of good quality that agree closely with the alignments produced by programs that incorporate three-dimensional structural information. When employed in discrimination tests, the HMM is able to distinguish members of these families from non-members with a high degree of accuracy.
Fig. 4. Comparison of the alignment of several representative tRNAs produced by trained (learned) grammar (bottom) with that from the biologically trusted database (top). Parentheses indicate base-paired positions; === the anticodon; and "[" and "]" the 5' and 3' sides of the acceptor helix, respectively.

Recently, Abe and Mamitsuka [1] have studied a more powerful class of grammars, called stochastic ranked node rewriting grammars, than SCFGs and applied it to the problem of secondary structure prediction of proteins.

6. Learning with non-grammatical representations

In grammatical inference, formal grammars or finite automata are usually used to represent the unknown languages. There are many other forms of representations which define languages. A typical example of representations which are not in the form of grammars is regular expressions for regular languages. Brzozowa and Ćerná [17] have studied efficient identification of regular expressions from good examples. Arikawa et al. [15] have considered elementary formal systems, a variant of logic programs, for identification of context sensitive languages.

In this section, we study two non-grammatical representation classes which are very hot and interesting topics in machine learning: one is simple recurrent neural networks and the other is case-based representations.

6.1. Connectionist approach

In neural network studies, recurrent neural networks have been shown to have the potential to encode temporal properties of a sequence of inputs. There have been proposed many recurrent neural network models [30] for dealing with temporal sequences, and here we consider variations of the simple recurrent neural network introduced by Elman [22]. In addition to the input and hidden units, the architecture
of simple recurrent networks has an extra hidden layer of context units which acts as the memory or the internal state of the network (Fig. 5). Thus, the simple recurrent network is a two-layer feedforward network augmented by the context units and the feedback connections to context units. There has been a great deal of interest in training simple recurrent networks to recognize grammars and simulate finite automata [24].

Sakakibara and Golea [54] have proposed the simple recurrent network of Fig. 5. The random variables represented by the input units take real values in \( \mathbb{R} \), and the hidden variables represented by the hidden units take values in \( \{0, 1\} \). The hidden variables represented by context units also take values in \( \{0, 1\} \). The context units simply hold a copy of the activations (state) of the hidden units from the previous time step. Thus, the next state of the hidden units is determined by the inputs and the state of the context units, the latter is equal to the previous state of the hidden units. From the finite-automata point of view, this dynamic structure is a finite-state machine and the simple recurrent network represents a state-transition function. Hence, simple recurrent networks should be able to perform the same type of computations as finite automata and solve grammatical inference problems.

Sakakibara and Golea [54] have proposed these simple recurrent neural networks as probabilistic models for representing and predicting time-sequences, and shown that the model can be viewed as a generalized hidden Markov model with distributed representations. First, the state transition and the output probability functions are nonlinear. Second, the model can deal with high-dimensional, real valued vectors as output symbols. Third, it has an efficient learning algorithm using dynamic programming based
on gradient descent (the algorithm can be seen as an extension of back-propagation). Moreover, compared to the previous attempts to link neural nets and HMM, the present model is more appealing because it does not require a specifically tailored architecture, e.g. second order connections where the multiplication operation is used between connection weights [24].

We denote by \( x^{(t)} = (x_1^{(t)}, \ldots, x_d^{(t)}) \in \mathbb{R}^d \) the state of the \( d \) input units at time \( t \), by \( h^{(t)} = (h_1^{(t)}, \ldots, h_m^{(t)}) \in \{0, 1\}^m \) the state of the \( m \) hidden units at time \( t \), and by \( h^{(t-1)} \in \{0, 1\}^m \) the state of the context units at time \( t \). The latter is a copy of the state of the hidden units at time \( t - 1 \). The network has no output and defines a probability distribution on the input space at time \( t \) via a suitable chosen energy function. Let \( w_j = (w_{j1}, \ldots, w_{jd}) \) represent the connection weights between the input units and the \( j \)th hidden unit, and \( \theta_j \) represent the bias of the \( j \)th hidden unit. Let \( u_j = (u_{j1}, \ldots, u_{jm}) \) represent the connection weights between the context units and the \( j \)th hidden unit. The set of parameters that defines both the network and the resulting distribution model is denoted by \( \Phi = \{(w_1, u_1, \theta_1), \ldots, (w_m, u_m, \theta_m)\} \). For a given model \( \Phi \), the state of the network at time \( t \) is defined by \((x^{(t)}, h^{(t)})\).

The model of Sakakibara and Golea [54] provides a new probabilistic formulation of learning in simple recurrent networks. We define the energy of a state configuration \((x^{(t)}, h^{(t)})\) at time \( t \), given the parameters of the model \( \Phi \) and the state of context units \( h^{(t-1)} \), by

\[
E_t(x^{(t)}, h^{(t)} | \Phi, h^{(t-1)}) = -\sum_{j=1}^m (w_j \cdot x^{(t)} + u_j \cdot h^{(t-1)} + \theta_j) h_j^{(t)} + \frac{1}{2} \|x^{(t)}\|^2,
\]

where \( w_j \cdot x^{(t)} = \sum_{i=1}^d w_{ji} x_i^{(t)} \), \( u_j \cdot h^{(t-1)} = \sum_{k=1}^m u_{jk} h_k^{(t-1)} \), and the factor \( \frac{1}{2} \|x^{(t)}\|^2 \) is included to make it possible to normalize probabilities. The probability of a state \((x^{(t)}, h^{(t)})\) at time \( t \) is defined to be

\[
Pr(x^{(t)}, h^{(t)} | \Phi, h^{(t-1)}) = \frac{1}{Z_t} e^{-E_t(x^{(t)}, h^{(t)} | \Phi, h^{(t-1)})},
\]

where \( Z_t \) is the appropriate normalization factor.

Given a sequence of inputs \( w = x^{(1)}, \ldots, x^{(N)} \), learning is formulated as the problem of choosing the set \( \Phi \) of parameters of the recurrent network that maximize the likelihood \( Pr(x^{(1)}, \ldots, x^{(N)} | \Phi) \). For gradient-based minimization procedures, we need the derivatives of the likelihood with respect to the parameters \( \Phi \) (here we use the negative log-likelihood for minimization instead of just the probability):

\[
\frac{\partial}{\partial \Phi_i} \log Pr(x^{(1)}, \ldots, x^{(N)} | \Phi) = \sum_{h^{(0)}, \ldots, h^{(N)}} Pr(h^{(0)}, \ldots, h^{(N)} | x^{(1)}, \ldots, x^{(N)}, \Phi) \times \frac{\partial}{\partial \Phi_i} \log Pr(x^{(1)}, \ldots, x^{(N)}, h^{(0)}, \ldots, h^{(N)} | \Phi)
\]
The first term in the above formula is calculated efficiently using the forward and backward probabilities (just like in the HMM). Evaluating the gradient in the second term results in the following formulas. For the weight vector \( w_j \) connecting the \( j \)th hidden unit to the input units, we get

\[
\frac{\partial}{\partial \theta_j} \log \Pr(x^{(t)}, h^{(t)} | x^{(t-1)}, h^{(t-1)}, \Phi) = \sum_{h^{(t)} \in \{0,1\}^n} \Pr(h^{(t)} | h^{(t-1)}, \Phi) h^{(t)} w_j h^{(t)}.
\]

The derivatives with respect to the weight vectors \( u_j \) and the bias parameters \( \theta_j \) \((j = 1, \ldots, m)\) can be obtained in a similar way. The gradient-based minimization using these derivatives can be seen as an extension of back-propagation.

They have presented some very preliminary simulation results to demonstrate the potential capabilities of the model. The very simple test uses a simple recurrent network with one input, two hidden units, and two context units to learn the periodic sequence 

\[-2, 0, 2, -2, 0, 2, -2, 0, 2, \ldots\]

The learned recurrent network is shown in Figs. 6 and 7. It is easy to see that the network represents a probabilistic finite automaton (HMM) and that each binary vector in the hidden layer corresponds to a state in the automaton. The output probability distribution is a time-varying (state-dependent) mixture of four basic Gaussians with variance \( \sigma = 1 \) and means 0, \( w_1 \), \( w_2 \), and \( w_1 + w_2 \). It is interesting to see how the learned recurrent network encodes the state transition function and the output function of the finite automaton in a distributed manner. For example, the learned network starts with two initial states, represented by the vectors \((0,0)\) and \((0,1)\), which have significant initial probabilities, and then repeats a sequence of state transitions: 

\[(0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (0,1)\]

Golea et al. [29] have also presented another experimental results that use simple recurrent networks for time series prediction, and shown that the learned network is robust for outliers in noisy time sequences.

Giles et al. [24] has enhanced simple recurrent networks by connecting to an external analog stack memory. It is called a neural net pushdown automaton, and manipulates the operations “push” and “pop” of the external stack and reads the top of the stack. They have tested its ability to learn some simple context-free grammars.

6.2. Case-based representation and learning

Case-based reasoning is deemed an important technology to alleviate the bottleneck of knowledge acquisition in Artificial Intelligence. In case-based reasoning, knowledge
Fig. 6. The learned RNN and its equivalent probabilistic FA.

Fig. 7. Output probability distributions for state transitions: left-upper for (0,0) \rightarrow (1,0), left-lower for (0,1) \rightarrow (1,0), right-upper for (1,0) \rightarrow (1,1), and right-lower for (1,1) \rightarrow (0,1).
Case-based representation

\[
G = \begin{cases}
  S \rightarrow AB, & B \rightarrow cd,
  \\
  A \rightarrow aAb, & B \rightarrow cd,
  \\
  A \rightarrow ab
\end{cases}
\]

\[
\begin{aligned}
  \{ (abcd, 1) \} &+ \sigma(w, v) = \\
  \{ (aabcddd, 1) \} &+ \sigma(w, v) = \\
  \{ (bbb, 0) \} &+ \sigma(w, v) = \\
\end{aligned}
\]

\[
\{ 1, w = abcd \text{ or } aabcddd \text{ and } v \in L, \\
1, w = bbb \text{ and } v \notin L, \\
0, \text{ otherwise.} \}
\]

**Fig. 8.** An example of grammatical representation and case-based representation for the language \( L = \{ a^m b^n c^n d^n \mid m, n > 0 \} \).

is represented in the form of particular cases with an appropriate similarity measure rather than any form of rules. Those cases are collected during knowledge processing. For solving particular new problems, cases representing former experience are retrieved. The most similar cases are chosen as a basis for generating new solutions including techniques of case adaptation. Within case-based reasoning, case-based learning as investigated in [3] is a natural way of designing learning procedures. The main task of case-based learning is to collect good cases which will be stored in the case base for describing knowledge and classifying unknown examples. Thus, case-based learning algorithms do not construct explicit generalizations from examples which most other supervised learning algorithms derive. Their hypotheses consist of case bases together with similarity concepts. Both constituents may be subject to learning. In this section, we see some of results on the power and the limitations of case-based representation and learning.

Representing a formal language by means of a finite set of labeled strings, also called cases, and a similarity function results in a finite description of an acceptor that is different from those usually used in formal language theory.

A similarity measure \( \sigma \) on \( \Sigma^* \) which defines a similarity between two strings is a computable function from \( \Sigma^* \times \Sigma^* \) to real interval \([0, 1]\). A case base \( CB \) is a finite subset of \( \Sigma^* \times \{0, 1\} \). We call a case \((w, 1)\) in \( CB \) a positive case and \((w, 0)\) a negative case. The language \( L(CB, \sigma) \) represented by a similarity measure \( \sigma \) and a finite case base \( CB \) is defined as follows:

\[
L(CB, \sigma) = \{ w \in \Sigma^* \mid \exists (u, 1) \in CB [\sigma(u, w) > 0 \land \forall (v, 0) \in CB [\sigma(u, w) > \sigma(v, w)] \}
\]

We restrict all positive cases to be taken from the unknown language and all negative cases to be taken from the complement of the language.

Globig and Lange [26] have shown that any indexable class of recursive languages is case-based representable. An indexable class is a set of formal languages that has an effective enumeration which admits a uniform procedure for deciding membership.

**Theorem 10** (Globig and Lange [26]). Let \( \mathcal{L} \) be any indexed class of recursive languages. There is a universal similarity measure \( \sigma \) such that every language \( L \) in \( \mathcal{L} \) can be represented by \( \sigma \) and a finite case base \( CB \) of positive and negative cases, i.e., \( L = L(CB, \sigma) \).
To prove the above theorem, we use an encoding technique that chooses any effective enumeration of words to represent languages and to relate words and languages in a somehow artificial but effective way.

We set \( \hat{L} = \mathcal{L} \cup \{ \Sigma^* \} \), and choose any effective enumeration \( L_0, L_1, L_2, \ldots \) of \( \hat{L} \) satisfying \( L_{2j+1} \in \mathcal{L} \) and \( L_{2j} = \Sigma^* \) for all \( j \geq 1 \). Let \( w_0, w_1, w_2, \ldots \) be any effective repetition-free enumeration of all strings in \( \Sigma^* \). First, we will define a unary total recursive function \( r \) that assigns to each string \( w_k \) a particular language \( L_r(k) \). Next, we will define a similarity measure \( \sigma \) in a way such that \( L(\{(w_k, 1)\}, \sigma) = L_r(k) \).

Initially, set \( r(0) = 0 \). We proceed inductively. Let \( k \in \mathbb{N} \). We set \( r(k + 1) = j \), if \( j \) is the least index \( j' < 2k \) satisfying \( w_{k+1} \in L_j \) and \( r(n) \neq j' \) for all \( n < k \). Next, we define the desired similarity measure \( \sigma \). Let \( k, n \in \mathbb{N} \). We set

\[
\sigma(w_k, w_n) = \begin{cases} 
1 & \text{if } w_k = w_n, \\
1 - \frac{1}{k+2} & \text{if } w_n \in L_r(k) \setminus \{w_k\}, \\
0 & \text{otherwise.}
\end{cases}
\]

By construction, \( L(\{(w_k, 1)\}, \sigma) = L_r(k) \) for all \( k \in \mathbb{N} \), and therefore \( w_k \) serves as a representative case for \( L_r(k) \).

They [26,25] have further shown that case bases consisting of exactly two cases are still sufficient to represent each language of any given indexable class. This upper bound is tight.

A formal framework for case-based learning has recently been developed by Jantke and Lange [33] in an inductive inference manner. Globig and Lange [26] and Sakakibara et al. [55] have investigated the power and the limitations of such case-based learning algorithms for formal languages in this framework.

For a complete presentation \( s \) of the form \((w_0, l_0), (w_1, l_1), (w_2, l_2), \ldots \) and a natural number \( n \), let \( s_{\leq n} \) denote the initial segment of \( s \) of length \( n \), i.e., \( s_{\leq n} = (w_0, l_0), (w_1, l_1), \ldots, (w_{n-1}, l_{n-1}) \). Let \( \mathbb{N} = \{0, 1, 2, \ldots\} \) be the set of all natural numbers. A class of languages \( \mathcal{L} \) is case-based learnable (in the limit) from complete presentation if and only if there are an algorithm \( M \) and a similarity measure \( \sigma \) such that for all \( L \in \mathcal{L} \) and for all complete presentation \( s \) of \( L \), there exists some case base \( CB \):

(i) \( \forall n \in \mathbb{N} : M(s_{\leq n}) = CB_n \) is defined,
(ii) \( \forall n \in \mathbb{N} : \emptyset \subseteq CB_0 \subseteq \{(w_0, l_0)\} \) and \( CB_n \subseteq CB_{n+1} \subseteq CB_n \cup \{(w_{n+1}, l_{n+1})\} \),
(iii) \( \lim_{n \to \infty} M(s_{\leq n}) = CB \),
(iv) \( L = L(CB, \sigma) \).

Sakakibara et al. [55] have shown that many classes of languages including the class of all regular languages are not case-based learnable with a fixed universal similarity measure, even if both positive and negative examples are presented.

**Theorem 11** (Sakakibara et al. [55]). Let \( \mathcal{L} \) be the class of all finite and all co-finite languages. Then \( \mathcal{L} \) is not case-based learnable from complete presentation.

Next Globig et al. [25] have considered a framework of case-based learning where the learning algorithm is allowed to learn similarity measures, too. An interesting and
important method for learning similarity measures is given by adopting weighting scheme for cases like the weighted nearest neighbor algorithm. This scheme is based on the idea that some cases stored within the case base are more reliable than others. This can be accomplished with the weights in similarity measures: reliable strings are given larger weights making them more similar to strings in the target domain. Then by allowing only to learn parameters of the weights in the similarity measures, they have shown that any indexable class of recursive languages is case-based learnable. This implies, in particular, that all context-free languages are case-based learnable by collecting cases and learning parameters of the similarity measure.

7. Conclusions

We have reviewed many recent advances in the grammatical inference research. Grammatical inference is considered a main subject of inductive inference, and grammars are important representations to be investigated in machine learning from both theoretical and practical points of view. In particular, recent research activities appeal more to the practical aspects such as computational linguistics and molecular sequence processing.

Since stochastic modeling would strongly be required for practical applications, an important future problem is to find efficient algorithms which solve both problems of determining the structure of the grammar and estimating the probabilistic parameters on identifying stochastic grammars. Those algorithms should be guaranteed theoretically for their correctnesses of identifiabilities and efficiencies.

As we have stated, it is a hard problem to find an appropriate discrete structure of the grammar for learning stochastic grammars. An interesting approach to the problem may be the use of Genetic Search for identification of the grammars, i.e., a search using genetic algorithm techniques for finding both the structure and the probabilistic parameters of the stochastic grammar. Some works (e.g., [21]) have been done to see the effectiveness of genetic search for grammatical inference problems.

On the other hand, there still remain many interesting open problems in identifying non-stochastic grammars. We have seen that some kinds of additional informations like answers by membership queries or structural information help the efficiency for identifying several classes of grammars. Hence, we would ask what kind of queries or what kind of additional information will contribute to establish polynomial-time identifications of which classes of grammars. These new types of queries or additional information should be reasonable in the sense of inductive inference.

The formal language domain (in particular, DFAs) has also been studied quite well in the PAC learning model (e.g., [42]) while we have reviewed only a few works for PAC learnabilities of formal languages. Especially, it is a very important open problem to solve the PAC learnability of DFAs.

Finally, we have seen many negative results for identifying grammars in the general settings, that is, identifying from every presentation of examples. It is very natural
and important to investigate efficient identifications of some classes of grammars from “good” examples [65], not from every kind of presentations.

Acknowledgements

We would like to thank Mostefa Golea, Takashi Yokomori, Takeshi Koshiba and Mitsuhiko Toda for their reading the draft and providing many valuable comments. Especially, Mostefa has helped very much for improving the paper. We also greatly thank our colleagues at the Machine Learning group at Fujitsu Labs., Hiroki Ishizaka (currently, at Kyushu Institute of Technology), Takeshi Koshiba, Masahiro Matsuoka, Yuji Takada, and Takashi Yokomori (currently, at University of Electro-Communications) for their excellent contributions to grammatical inference and enjoyable discussions. Finally, we gratefully acknowledge the reports of anonymous referees. Their significant comments and constructive suggestions helped very much to improve the paper. This work has been done while the author was at ISIS, Fujitsu Laboratories Ltd.

References


