Learning Dependency Languages from a Teacher

JÉRÔME BESOMBES AND JEAN-YVES MARION

We investigate learning dependency grammar from partial data and membership queries as a model of natural language acquisition. We define a learning paradigm based on a dialogue between the learner and a referent who knows the target language. This dialogue consists in a presentation of structured partial sentences and queries about the membership of original sentences constructed by the learner. We define an efficient algorithm corresponding to this paradigm and illustrate it on examples.

2.1 Introduction

For the definition of our model we consider several hypotheses which are largely inspired by the works of the linguist Chomsky Chomsky (1986) the psycholinguist Pinker Pinker (1994). First, the child learns the language of his parents or more specifically, the language he hears. Correct sentences are so presented to the learner, possibly partially understood and constitute the input data of the model. These data are not only linear sentences but pre-calculated structures. Indeed, semantic information or prosody are information included in the signal.

The structures are commonly considered as tree in which nodes are labelled by the words of the sentence. Since the linear order of the words is conserved during the structuration process, we will consider dependency trees as a relevant model of the input data (Figure 1).
Pinker underlines that the learner and the referent take part in a communication process; this communication is a key point of the learning process since it turns out that the language acquisition is not possible with no physical presence of the referent. If we suppose that a new sentence produced by a child and not understood by the referent can provide the conclusion that this sentence is not correct (doesn’t belong to the language he learns), we will take into account membership queries: the algorithm submits sentences to an Oracle who replies yes or no whether they belong to the target language or not.

The algorithm $A$ learns a class of language if and only if, for any language $L$ in the class, there is a finite set of partial data $RS$ (representative sample) such that $A$ determines $L$ from $RS$ with help of membership queries. Properties of $A$ are summarized in Figure 2.
Related works
Angluin first introduced the paradigm of learning with queries in Angluin (1987) for the case of regular languages and in Angluin (1988) is studied a paradigm of learning from positive examples, membership queries and equivalence queries (the possibility to ask an Oracle whether a guess language corresponds to the target language or not). Obviously, for our motivation of modelling, this kind of queries are not relevant. Angluin’s works have been extended in particular by Sakakibara (1987b,a, 1990) for the inference of context-free grammars from structured data. The learnability of dependency languages has been studied in Besombes and Marion (2002); in this work, an algorithm for a sub-class of lexical dependency languages has been defined. As far as we know, the idea of learning from partial data and membership queries is original.

2.2 Lexical dependency grammar
Following Dikovsky and Modina (2000), we present a class of projective dependency grammars which was introduced by Hays (1961) and Gaifman (1965).

A lexical dependency grammar (LDG) $\Gamma$ is a quadruplet $(\Sigma, N, P, S)$, where:

- $\Sigma$ is the set of terminal symbols,
- $N$ is the set of non-terminal symbols,
- $S \in N$ is the start symbol,
- $P$ is the set of productions.

Each production is of the form

$$X \rightarrow X_1 \ldots X_p a X_{p+1} \ldots X_q$$

or of the form

$$X \rightarrow a$$

where $X$ and each $X_i$ are in $N$ and $a$ in $\Sigma$. The terminal symbol $a$ is called the head of the production. In other words, the head is the root of the flat tree formed by the production right hand side. Actually, if we forget dependencies, we just deal with context free grammars.

Given a grammar $G$, partial dependency trees $t$ generated by a non-terminal $X$ of $G$ are recursively defined as follows.

- $X$ is a partial dependency tree.

\[1\text{This form is corresponding to the previous with } p = q = 0.\]
If \( X' \ldots b \ldots \) is a partial dependency tree generated by \( X \), and if \( X' \rightarrow X_1 \ldots X_p \ a \ X_{p+1} \ldots X_q \) is a production of \( G \), then \( \ldots X_1 \ldots X_p \ a \ X_{p+1} \ldots X_p \b \ldots \) is a partial dependency tree generated by \( X \).

We note \( X \rightarrow^* t \) to express that \( t \) is generated by \( X \).

A dependency tree generated by a non-terminal \( X \) is a partial dependency tree generated by \( X \) in which all nodes are terminal symbols. A dependency tree is a dependency sub-tree generated by \( S \). The language \( DL(G) \) is the set of all dependency trees \( (DL(G) = \{ d \ : \ text{dependency tree and } \ S \rightarrow^* d \}) \).

(1) Example. Consider the grammar \( G \) defined by:

\[
G = (\Sigma, N, P, S)
\]

where
- \( \Sigma = \{a, b, c\} \),
- \( N = \{S, X_1, X_2, X_3, X_4\} \),
- \( P \) is the following set of productions.

\[
S \rightarrow X_2 \ a \ X_3
\]

\[X_2 \rightarrow X_1 \ b \quad X_3 \rightarrow c \ X_4\]

\[X_1 \rightarrow X_2 \ b \quad X_3 \rightarrow c\]

The language \( DL(G) \) is the set of dependency trees

\[
\{ \text{dependencies can be drawn either over or under the word line for a reason of clarity.} \}
\]
A subtree of a dependency tree is inductively defined as follows:

- of \( d = a \) for a terminal symbol \( a \), then \( d \) is the only subtree of \( d \),
- if \( d = d_1 \ldots d_p \) is a dependency tree then:
  - \( d \) is a subtree of \( d \),
  - any subtree of \( d_i \) is a subtree of \( d \).

If \( d \) is a dependency tree, \( S(d) \) is the set of subtrees of \( d \) and if \( D \) is a set of dependency trees, \( S(D) \) is the set of all subtrees of the elements of \( D \). A context \( c[x] \) of a dependency tree \( d \) is obtained by replacing exactly one occurrence of a subtree of \( d \) by a special symbol \( \# \). In particular \( \# \) is a context of all dependency trees. If \( d \) is a dependency tree, \( C(d) \) is the set of contexts of \( d \) and if \( D \) is a set of dependency trees, \( C(D) \) is the set of all contexts of the elements of \( D \).

We will also use the notation \( d = c[d'] \) to express that \( d' \) is a subtree of \( d \).

A grammar homomorphism \( \phi \) between two grammars \( G = \langle \Sigma, N, P, S \rangle \) and \( G' = \langle \Sigma, N', P', S' \rangle \) is defined from a surjective mapping from \( N \) to \( N' \) which satisfies the following properties:

- \( \phi(S) = S' \)
- \( P' \) is the set of productions \( \phi(X) \rightarrow \phi(X_1) \ldots \phi(X_p) a \phi(X_{p+1}) \ldots \phi(X_q) \)
  for every production \( X \rightarrow X_1 \ldots X_p a X_{p+1} \ldots X_q \) of \( P \).

We note \( G' = \phi(G) \) and in this case we have \( DL(G) \subseteq DL(G') \).

### 2.3 Observation table

Following Angluin (1988), information obtained from the membership queries is stored in a table. Let \( DL \) be a dependency language, \( D \) a finite set of subtrees and \( C \) a finite set of contexts. The observation table \( T = T_{DL}(S(D), C) \) is the table defined by:

- rows are labelled by the subtrees of \( D \),
- columns are labelled by elements of \( C \),
- cells \( T_{DL}(d, c[x]) \), where \( d \in S(D) \) and \( c[x] \in C \), are labelled with 1 and 0 in such a way that:
  \[
  T_{DL}(d, c[x]) = \begin{cases} 
  1 & \text{if } c[d] \in DL \\
  0 & \text{otherwise}
  \end{cases}
  \]
For any $d \in S(D)$, we denote by $\text{row}_T(d)$ the binary word corresponding to the reading from left to right of the row labelled by $d$ in $T$.

(2) Example. Let be $\mathcal{DL} = \mathcal{DL}(G)$ the dependency language defined in Example 1, $D$ the singleton $\{b b a c c c\}$ and $C$ the set of contexts $\{\#, \# b a c c c, \# a c c c, b b a c c \#, b b a c \#, b b a \#\}$.

The corresponding observation table $T = T_{\mathcal{DL}}(S(D), C)$ is the table of figure 2.

An observation table $T = T_{\mathcal{DL}}(S(D), C)$ is coherent if and only if for any pair $(d, d')$ of trees in $D \times D$, $\text{row}_T(d) = \text{row}_T(d')$. A coherent observation table $T = T_{\mathcal{DL}}(S(D), C)$ defines a grammar $G_T$:

$$G_T = \langle \Sigma, N, P, S \rangle$$

where:

- $\Sigma$ is set of symbols occuring in $D$,
- $N = \{\text{row}_T(d) : d \in S(D)\}$
- $S = \text{row}_T(d)$ for any dependency tree $d \in D$
- $P$ is the set of productions of the form $\text{row}_T(d_1 \ldots d_p a d_{p+1} \ldots d_q) 
\rightarrow \text{row}_T(d_1) \ldots \text{row}_T(d_p) a \text{row}_T(d_{p+1}) \ldots \text{row}_T(d_q)$

for all $d_1 \ldots d_p a d_{p+1} \ldots d_q$ in $S(D)$.

(3) Example. The table of Example 2 is coherent and the corresponding grammar is $\phi(G)$, where $G$ is the grammar given in Example 1 and $\phi$ the homomorphism defined by $\phi(S) = 100000$, $\phi(X_1) = 010000$, $\phi(X_2) = 001000$, $\phi(X_3) = 000101$, $\phi(X_1) = 000010$.

A coherent table $T = T_{\mathcal{DL}}(S(D), C)$ is consistent if and only if for every dependency trees $d = d_1 \ldots d_p a d_{p+1} \ldots d_q$ and $d' = d'_1 \ldots d'_p a d'_{p+1} \ldots d'_q$. 


<table>
<thead>
<tr>
<th>$z$</th>
<th>$\overline{z}$</th>
<th>$\overline{z} \overline{b} a c c$</th>
<th>$\overline{z} a c c$</th>
<th>$b b a c c$</th>
<th>$b b a c$</th>
<th>$b b a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b b a c c$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$b b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c c c$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c c$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 3** An observation table
in $S(D)$, for all $i$, $\text{row}_T(d_i) = \text{row}_T(d'_i)$ implies that $\text{row}_T(d) = \text{row}_T(d')$.

2.4 Representative sample

We now define the property, for a finite set of subtrees of a language, to contain the minimum information necessary to explicitly identify this language. This constitutes a minimal hypothesis to conclude in the learnability of the language. Let $DL$ be a dependency language generated by a grammar $G$. Any finite subset $RS$ of $S(DL)$ is said to be representative for $DL$ if and only if for any transition $X \rightarrow X_1 \ldots X_p \leftarrow X_{p+1} \ldots X_q$ of $G$, there is an element $d = (d_1 \ldots d_p \leftarrow d_{p+1} \ldots d_q)$ in $S(RS)$ such that for all $i$, $X_i \rightarrow d_i$. Informally, a finite set $RS$ is a representative sample for $G$ if and only if each production of $G$ has been used at least once to produce the elements of $RS$.

Lemma 1 Let $G$ be a dependency grammar, $RS$ a representative sample for $DCL(G)$ and $C$ a finite set of contexts containing $C(RS)$, if $T_{DCL(G)}(S(RS), C)$ is consistent then $DCL(G_T) = DCL(G)$.

Theorem 2 The algorithm defined in Figure 4 learns the class of dependency languages from representative samples and membership queries.

The algorithm works as follows: it take a finite set of dependency trees as input and this set is decomposed in a finite set of subtrees and a finite set of contexts. With help of membership queries, a first observation table is constructed and the consistence is checked. If the table is not consistent, new contexts are calculated and added in the table which is then completed. The process stops as the table is consistent and a grammar is then output.

2.5 Examples

(4) Example. The singleton \{b b a c c c\} is a representative sample

for the dependency tree language defined in Example 1. The observation table of Figure 3 is constructed from this input with help of membership queries; this table is consistent that implies
INPUT: a finite set of dependency trees $D$
INITIALIZATION: $C = C(D)$; construct $T = T_{DL(G)}(S(D), C)$ with
help of queries
WHILE $T$ not consistent DO

find two dependency trees $d_1 \ldots d_p \ a \ d_{p+1} \ldots d_q$ and $d'_1 \ldots d'_p \ a \ d'_{p+1} \ldots d'_q$
in $S(D)$ such that for all $i$, $row_T(d_i) = row_T(d'_i)$ and $row_T(d) \neq row_T(d')$
add every contexts $d_1 \ldots d_p \ a \ d_{p+1} \ldots d_q$ in $C$ and $d'_1 \ldots d'_p \ a \ d'_{p+1} \ldots d'_q$

complete $T = T_{DL(G)}(S(D), C)$ with help of queries
ENDWHILE
RETURN $G_T$

FIGURE 4 The learning algorithm

that the corresponding dependency grammar given in Example 3 is computed by the algorithm and the language is learnt imme-
diately (the loop is not processed).

The following example illustrates the iterative behavior of the algo-
rithm.

(5) Example. Let $G$ be the following grammar:

$$S \rightarrow aX_1, aX_2, bX_2$$

$$X_1 \rightarrow dX_3, c \quad X_3 \rightarrow c$$

$$X_2 \rightarrow dX_4 \quad X_4 \rightarrow f$$

We have: $DL = \{b, c, a, c, a, d, c, b, d, c, a, d, f\}$.
Let now consider the following representative sample:

\[ RS = \{ b \ c, a \ d, e \} \]

From it, the algorithm constructs a first table that is not consistent (Figure 5). Indeed we have:

\[ \text{row}_T(e) = \text{row}_T(f) \]

but

\[ \text{row}_T(d \ e) \neq \text{row}_T(d \ f) \]

The new context \( b \ d \ y \) is computed and added to the table that is completed with queries. The table obtained is then consistent and the process stops with the construction of grammar \( \phi(G) \), where \( \phi \) is defined by \( \phi(S) = 10000, \phi(X_1) = 01010, \phi(X_2) = 00010, \phi(X_3) = 00101, \phi(X_4) = 00100 \)

References


FIGURE 5 The learning algorithm processing