Linguistic Abduction Machines

Working Paper No. 1

by

Ulf Grenander
Division of Applied Mathematics
Brown University
Providence, Rhode Island

Date: Jan. 1976

11

12 P.

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

DDC
MAR 21 1977

RECEIVED

A
Let the syntactic variables be 1,2,3...v and words will be denoted x,y,z, etc. Introduce the matrix

\[ P(x) = \{p_{ij}(x)\} \]

where \( p_{ij}(x) \) is the probability of rewriting \( 1 \rightarrow jx \), and the vector

\[ r(x) = \{r_i(x)\} \]

where \( r_i(x) \) is the probability of rewriting \( 1 \rightarrow x \) (see Grenander's paper in Neyman Festschrift for further details and equation (3) used below).

When we search for the syntactic variables it may be best to organize the search from below following a suggestion by Henry Kucera. This means that we first try to group words into classes \( C_1, C_2, \ldots, C_r \), then group classes into higher level classes and so on. It seems as if this would reduce the search effort drastically since the number of words \( n_w \) is much larger than the number of syntactic classes \( n_v \).

When we do this we have to proceed by testing for linguistic equivalence similarly to method in paper on abduction machine. Two words \( x \) and \( y \) are said to be equivalent, written as \( x \equiv y \), if \( uxv \) and \( uyv \) are either both grammatical or both non-grammatical, \( u \) and \( v \) arbitrary lexical strings.

The search will depend crucially on how difficult it is to separate \( x \) from \( y \) by equivalence when they are not equivalent. The trouble is that when we test with \( u \) and \( v \), a negative answer
is enough to establish \( x \neq y \), but a positive answer is not. In principle we would have to go through all \( u \) and \( v \).

**Lemma 1.** The statement \( x \equiv y \) is the same as to say that \( P(uxv) \) and \( P(uyv) \) are both zero or not zero, all \( u \) and \( v \) strings.

**Proof:** For a given lexical string \( S = x_1, x_2, \ldots, x_n \) we get the probability

\[
(3) \quad P(S) = dP(x_1)P(x_2)\ldots P(x_{n-1})v(x_n)
\]

where \( d \) is the vector \((1, 0, 0, \ldots, 0)\). We know from our earlier work however, that \( S \) being grammatical is the same as \( P(S) \) being positive, hence our statement correct, and we shall see that (3) can be used to clear up the situation more.

Introduce the function of two words \( x \) and \( y \)

\[
(4) \quad d(x, y) = \max \left\{ \sum |p_{ij}(x) - p_{ij}(y)| \right\} + \max \{|r_1(x) - r_1(y)|\}.
\]

**Lemma 2.** The function \( d \) is a pseudo distance

(a) \( d \geq 0, d(x, x) = 0 \)
(b) \( d(x, y) = d(y, x) \)
(c) \( d(x, z) \leq d(x, y) + d(y, z) \).

**Proof:** (a) and (b) are obvious. We have

\[
(5) \quad d(x, z) = \max \left\{ \sum |p_{ij}(x) - p_{ij}(z)| \right\} + \max \{|r_1(x) - r_1(z)|\}
\]

\[
\leq \max \left\{ \sum |p_{ij}(x) - p_{ij}(y)| \right\} + \max \left\{ \sum |p_{ij}(y) - p_{ij}(z)| \right\} + \max \{|r_1(x) - r_1(y)| + \max \{|r_1(y) - r_1(z)|\} = d(x, y) + d(y, z).
\]
Note however that \( d(x,y) = 0 \) does not imply \( x = y \), it only means that \( P(x) = P(y) \) and \( r(x) = r(y) \). But using Lemma 1 this means that \( x \equiv y \) so that the pseudo distance separates the words in the dictionary into equivalence classes. Also \( x \equiv y \) does not imply \( d(x,y) = 0 \). It is also clear that \( d \leq 2 \) since \( \sum_{x} \sum_{j} [p_{ij}(x) + r_1(x)] = 1 \).

We can now get a bound on how difficult it is to separate \( x \) from \( y \) by the testing procedure mentioned above. We have using (3) for \( S = x_1 x_2^n x_{r-1} x x_{r+1} \ldots x_n \) and \( S' = x_1 x_2^n x_{r-1} x x_{r+1} \ldots x_n \)

\[
\begin{align*}
P(s) - P(s') &= d[P(x_1) \ldots P(x_{r-1})P(x)P(x_{r+1}) \ldots r(x_n)] - dP(x_1) \ldots P(x_{r-1})P(y)P(x_{r+1}) \ldots r(x_n)) = dA[P(x) - P(y)]Br(x_n). \end{align*}
\]

The matrix \( P(x) \) has now bounded by 1 since

\[
\|P(x)\| \leq \max_{i,j} \sum p_{ij}(x) \leq 1
\]

Hence \( \|A\| \) and \( \|B\| \leq 1 \) so that

\[
|P(s) - P(s')| \leq \|P(x) - P(y)\| \leq d(x, y).
\]

This was when \( v \) is not empty. If \( v \) is empty, so that the sentences end with \( x \) and \( y \) respectively, we get instead with a similar argument

\[
|P(s) - P(s')| \leq \|r(x) - r(y)\| \leq d(x, y).
\]
Hence we have

**Theorem 1.** The difference in test probabilities for two words \( x \) and \( y \) is bounded by

\[
|P(S) - P(S')| \leq d(x, y). 
\]

It is known at present how sharp this inequality is.

We now start the abduction from below and consider the dictionary \( D = \{1, 2, \ldots, n\} \), to begin with consider as a single class called \( 1 \).

**Partition Algorithm:** After \( t \) sentences have been heard \( D \) has been partitioned into classes \( c_1, c_2, \ldots, c_n \), mutually disjoint and exhaustive. When sentence No. \( t+1 \) is heard, \( S = uxy \) one word \( x \) appearing in it is picked (systematically or at random?) and replaced by another word \( y \) in the same class \( c(x) \). The following action is taken.

(a) if \( S' = uyv \) is grammatical nothing is done and the algorithm loops to the next sentence.

(b) if \( S' = uyv \) is not grammatical \( y \) is removed from class \( c(x) \) and we move to (c).

(c) start a new loop going through all the other classes \( c_1, c_1 \neq c(x) \). In each pick a word \( z \) (systematically or at random) and test for \( x \neq z \) as before. The first time the answer is positive move \( x \) to this class. Otherwise move to (d).

(d) create a new class \( c_{n+1} \) consisting of just \( x \). Then move back to start.
A realization of this scheme may look like Figure 1. Note that the storage requirement for this scheme is very modest: a vector of length $n_w$ whose entries are integers.

Of course step (c) can fail to establish a negative answer with positive probability. I guess, but this may be wrong, that the performance would improve by the following modification.

(c') same as (c) except that $R$ words are picked (without replacement) from each class. If all lead to positive answers put $x$ in this class, otherwise go on to the next class, etc. $R$ could be called the replication number.

<table>
<thead>
<tr>
<th>word</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 1

Before implementing the partition algorithm for abduction, let us carry out an experiment as follows to gain some insight in its functionality and (lack of?) computational efficiency.
Define a probability measure \( p_x \) over the dictionary. Say that the true partition is

\[
\begin{align*}
\mathcal{C}_1 &= \{1, 2, \ldots, n_1\} \\
\mathcal{C}_2 &= \{n_1+1, n_1+2, \ldots, n_1+n_2\}
\end{align*}
\]

and say that if \( x \in \mathcal{C}_i, y \in \mathcal{C}_j \) then the test will give negative answer with a probability \( d_{ij} \), depending only upon the class index. Let us choose

\[
\begin{cases}
  d_{ii} = 0 \quad \text{(this is not necessary, perhaps)} \\
  0 \leq d_{ij} \leq 1
\end{cases}
\]

and let us also make \( d_{ij} \) into a distance. A simple choice would be

\[
d_{ij} = \frac{|i-j|}{\delta}
\]

Simulate the procedure and count number of tests until true partition has been reached. The mean number of tests is an appropriate number of computational work required by the algorithm.

Finally a remark about a distance measure between two partitions described by the incidence matrices \( M \) and \( M' \). Pick two elements \( x \) and \( y \) at random and look for the probability that \( (x \notin y) \neq (x \notin y) \).

**Lemma 3.** This probability is a distance.

**Proof:** We have
\[ \delta(M, M') = \sum_{x, y} p_{xy} \{(x \in M) \neq (x \in M') \} = \sum_{x, y} p_{xy} |m_{xy} - m'_{xy}| \]

This is a \( l_1 \)-metric so that the statement in the lemma is true.

It may be more natural to replace \((h)\) by

\[ d(x, y) = \max \left\{ \sum_{i, j=1}^{n_{v+1}} |p_{ij}(x) - p_{ij}(y)| \right\} \]

where we have defined \( p_{in_{v+1}}(x) = r_i(x) \).