Learning Stochastic Context-Free Grammars from Corpora Using a Genetic Algorithm

Bill Keller and Rüdi Lutz
School of Cognitive and Computing Sciences
The University of Sussex
email: {billk,rudil}@cogs.susx.ac.uk

Abstract

A genetic algorithm for inferring stochastic context-free grammars from finite language samples is described. Solutions to the inference problem are evolved by optimizing the parameters of a covering grammar for a given language sample. We describe a number of experiments in learning grammars for a range of formal languages. The results of these experiments are encouraging and compare very favourably with

$$
\begin{array}{ll}
\mathrm{S} \rightarrow \mathrm{A\ B} & (1.0) \\
\mathrm{A} \rightarrow a & (0.6) \\
\mathrm{A} \rightarrow \mathrm{C\ S} & (0.4) \\
\mathrm{B} \rightarrow b & (1.0) \\
\mathrm{C} \rightarrow a & (1.0)
\end{array}
$$

Figure 1: SCFG for the language $a^n b^n$ $(n \geq 1)$

speech recognition, part-of-speech tagging, optical character recognition and robust parsing. While Schwem and Ost recognize the importance of the stochastic inference problem, their approach is restricted to the inference of stochastic regular grammars. In contrast, the present work tackles the more general problem of inferring stochastic grammars for the class of context-free languages.

The following sections describe our approach to grammatical inference in more detail. Stochastic context-free grammars are briefly described in section 2. Section 3 discusses the problem of inferring stochastic grammars from corpora. Details of the genetic algorithm are then given in section 4 and in section 5 we present the results of several experiments in learning grammars for a range of formal languages.

2 Stochastic Context-Free Grammars

A *stochastic context-free grammar* (SCFG) is a variant of ordinary context-free grammar in which each grammar rule is associated with a probability, a real number in the range [0,1]. The set of production probabilities will be referred to as the *parameters* of the SCFG. For a SCFG to be *consistent*, the probabilities associated with all rules that expand the same non-terminal symbol must sum to one.

The language $L(\ )$ generated by a SCFG comprises the set of all strings of terminal symbols derivable from the start symbol of the grammar (typically, S). In addition, the parameters define a probability distribution over strings in $L(\ )$. For a string $\alpha \in L(\ )$, the probability of a parse tree for α is given by the product of the probabilities of all the grammar rules involved in its construction. The probability $P_G(\alpha)$ of the string α is the sum of the probabilities of all of its parses.

An example of a simple SCFG is shown in figure 1, with the probability associated with each production given in parentheses. The SCFG generates the language $\{a^n b^n | n \geq 1\}$, where $P_G(ab) = 0.6$, $P_G(aabb) = 0.24$, and so on.

A *corpus* C for a language L is a finite set of strings drawn from L, where each string $\alpha \in C$ is associated with an integer f_α representing its *frequency of occurrence*. The *size* N_C of the corpus is defined as the sum of the frequencies of the individual strings in C. That is:

$$N_C = \sum_{\alpha \in C} f_\alpha$$

The *relative frequency* p_α of a string $\alpha \in C$ is defined as $p_\alpha = f_\alpha / N_C$

ab	595
aabb	238
aaabbb	97
aaaabbbb	49
aaaaabbbbb	14
aaaaaabbbbbb	5

Figure 2: A Corpus for the Language $a^n b^n$

An example of a corpus for the language $\{a^n b^n | n \geq 1\}$ is shown in figure 2. The frequency of the string *ab* is 595, the frequency of *aabb* is 238, and so on. The total size of the corpus is 998. The relative frequencies of the

The population maintained by the genetic algorithm is organized as a two dimensional grid, with opposing sides of the grid identified (i.e. members of the population inhabit the surface of a torus). Thus, each member of the population has exactly eight neighbours. A member of the population encodes a set of parameters for the rules of the covering grammar, with each parameter encoded as a fixed-length bit string. If a block of n bits is used to encode each parameter, then for a covering grammar having M rules each member of the population has a genome of length Mn bits, where the jth parameter is encoded as the jth n-bit block.

Because the parameters of a SCFG are not independent of one another, we do not encode their values directly. Instead, each n-bit block is treated as an encoding of a numerical *weight*. To obtain the actual parameters of the SCFG, the weights are normalized as part of the decoding process. If w_j is the weight associated with rule r_j, then $p_j = w_j/W$ is the probability associated with this rule (where W is the sum of all those weights associated with rules expanding the same non-terminal as r_j). A weight of zero means that the corresponding rule does not belong to the rule-set of the SCFG.

An obvious scheme for encoding the weights is to treat each n-bit block as a binary representation of an integer value. However, this simple scheme has the drawback that it makes it relatively unlikely that a rule will be assigned a zero weight. In general, the covering grammar has many more rules than are required for the target SCFG, so it makes sense to use an encoding that is biased in favour of rules having zero weight rather than the other way around. To achieve this, the encoding scheme is modified by reserving a small number of initial bits in each block. If each of these initial bits is set to 1, then the remaining bits are decoded to obtain the rule weight. In all other cases the rule ¸weight is zero, and the remaining bits are ignored. The number of reserved bits effectively controls the amount of bias in favour of a rule being assigned a weight of zero, while the number of remaining bits controls the size of the rule weights and thus the precision of the rule probabilities. For the experiments described in the following section we have found that between one and three initial bits and 7 'weight' bits is sufficient[2].

The members of the initial population are generated randomly, after which the genetic algorithm repeatedly executes the following select-breed-replace cycle:

> **Select** a random member of the population for breeding, and choose the fittest of its eight neighbours as the second parent.
>
> **Breed** by applying crossover and mutation to produce two children.
>
> **Replace** the weakest parent by the fittest child.

Selection and replacement are carried out within a small, local population. This allows for rival solutions to emerge at different locations and discourages too rapid a spread of successful genetic information throughout the whole population. By replacing the weakest parent, rather than the weakest neighbour, relatively unfit individuals still get a chance at breeding while useful genetic material from the weakest parent may survive through the fittest child.

[2]The actual number of initial bits is determined automatically in proportion to the size of the covering grammar. The larger the grammar, the more weight bits are used.

A characteristic of our parameter encoding scheme is that the probability associated with any given rule does not depend solely on local properties of the genome (i.e. the state of the relevant n-bit block). In general, it will also depend

Language	Nonterminals	Parameters	Successful	Best	

S → C B	(0.51875)
S → D A	(0.48125)
A → *a*	(1.0)
B → S C	(0.24778)
B → *b*	(0.752212)
C → *b*	(1.0)
D → A S	(0.252066)
D → A C	(0.066116)
D → *a*	(0.681818)

Figure 4: Near-miss grammar for the language of two symbol palindromes

The results for the two palindrome languages initially appear less encouraging. For PAL1, only two runs attained the threshold fitness value, while for PAL2 only one of the three runs was terminated successfully. On the other hand, even on the worst runs in each case the algorithm found grammars with quite high fitness. Furthermore, it should be recalled that the threshold fitness value represents an arbitrary measure of success. In particular, failure to attain this threshold does not imply that the algorithm has failed to find a grammar with a correct (or nearly correct) set of rules. For example, it is possible that the grammar generates the target language exactly, but with a non-optimal probability distribution.

Inspection of the grammars produced for all runs of the PAL1 learning task showed that the algorithm had performed rather better than suggested by the 2/10 success rate given in the table. Figure 5 shows the grammar ranked fifth best (with a fitness of 0.897355) out of all those produced by the algorithm on this task. Aside from the presence of one spurious production D → A C, which has a low associated probability (0.066116), the grammar is otherwise correct. Indeed, it was found that the five fittest grammars produced by

each candidate solution, which requires parsing every string in the corpus in all possible ways. Although inference can

[Ri78] Risannen, J. (1978) Modelling by shortest data description, *Automatica*, 14, 465-471.

[SO95] Schwem, M. and A. Ost (1995) Inference of stochastic regular grammars by massively parallel genetic algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms*, 520–527, Morgan-Kaufmann, CA.

[SJ92] Sen, S. and J. Janakiraman (1992) Learning to construct pushdown automata for accepting deterministic context-free languages, in G.Biswas (ed.), *SPIE Vol. 1707: Applications of Artificial Intelligence X: Knowledge-Based Systems*, 207–213.

[So64] Solomonoff, R.J. (1964) A formal theory of inductive inference, *Information and Control*, 7, 1–22; 224–254.

[Wy91] Wyard, P. (1991) Context-free grammar induction using genetic algorithms, in R.Belew and L.B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA'92*, 514–418, Morgan Kaufmann. CA.

[ZG86] Zhou, H. and J.J. Grefenstette (1986) Induction of finite automata by genetic algorithms, *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics*, 170–174, Atlanta, GA.