Unsupervised Incremental Structure Learning of Stochastic And-Or Grammars with Monte Carlo Tree Search

Luyao Yuan[1], Jingyue Shen[1], Zipeng Fu[1], Song-Chun Zhu[21]
[1]Department of Computer Science, University of California, Los Angeles
[2]Department of Statistics, University of California, Los Angeles
{luyao@g.,brianshen@,fu-zipeng@engineering.,sczhu@stat.}ucla.edu

Abstract

Stochastic And-Or grammars form a compact representation of probabilistic context-free grammars. They explicitly model compositionality and reconfigurability in a hierarchical manner and can be utilized to understand the underlying structures of different kinds of data (e.g., language, image, or video). In this paper, we proposed an unsupervised And-Or grammar learning approach that iteratively searches for better grammar structure and parameters to optimize the grammar compactness and data likelihood. To handle the complexity of grammar learning, we developed an algorithm based on the Monte Carlo Tree Search to effectively explore the search space. Also, our method enables incremental grammar learning. Experimental results show that our approach significantly outperforms previous greedy-search-based approaches, and our incremental learning results are comparable to previous batch learning results.

1 Introduction

Stochastic grammar has been a fundamental model in language comprehension and production tasks for decades (Charniak, 1997; Chater and Manning, 2006). In recent years, it has also been successfully applied to other fields such as image processing, object detection (Si and Zhu, 2013), human activity understanding (Ivanov and Bobick, 2000; Pei et al., 2011), and robot control (Liu et al., 2018). Given the great success of using stochastic grammar to explicitly analyze the underlying structure/process of various types of data, however, automatic learning of stochastic grammar remains a challenging problem.

Grammar learning, particularly learning its structure (i.e., production rules), is an NP-hard problem due to the innate ambiguity of grammar and the tremendous cardinality of the grammar space (Gold, 1978; Brabrand et al., 2007). To tackle the complexity, previous approaches relied on greedy/heuristic search (Stolcke and Omohundro, 1994; Solan et al., 2005; Tu et al., 2013). Inspired by this paradigm, we formulated learning stochastic grammar as a search problem and employed a Bayesian probabilistic function to integrate both structure and parameter learning into a unified framework. Furthermore, we utilized Monte Carlo Tree Search (MCTS) to effectively explore the search space and avoid unpromising local minima. Experimental results show that our MCTS based method significantly outperforms previous greedy methods.

Besides the improvement in batch learning, we further enhanced our approach to enable incremental learning. Contrast to batch learning, incremental learning does not require the entire training dataset to be loaded but only a small portion of data is used in each round of updating the learned model. Although such incremental setting has been well-adopted in Deep Learning (Le et al., 2011), it is less studied in grammar learning and we believe it is important for leveraging the memory-efficiency and model-transferability of a grammar learning approach. Empirical results show that our incremental learning approach can achieve comparable performance to previous batch learning approaches.

The main contributions of this paper are: (i) develops an MCTS based grammar learning approach that learns both the structure and parameters of a stochastic grammar unsupervisedly; (ii) extends this approach to support incremental learning and achieves comparable performance.

2 Related Work

Grammar learning is an NP-hard problem (Gold, 1978) that can trace back to finite-state automata

learning (Hill III, 1979). Stolcke and Omohundro (1994) first introduced the idea of using heuristic search to tackle grammar learning and proposed a probability-based objective function to guide the search. Afterward, different heuristic functions were proposed and used with greedy/beam search for unsupervised grammar learning on natural language and other types of data (Solan et al., 2005; Tu et al., 2013). The major limitation of these previous approaches is the restricted search scope in the grammar space.

There were also research efforts in incremental grammar learning, but they either required access to both positive and negative examples (Nakamura and Matsumoto, 2005; Nakamura and Imada, 2010), or only positive examples but restricted to a simple-to-complex ordering (Javed et al., 2008). Incremental parameter learning with fixed grammar structure was also tackled by using online EM algorithms (Liang and Klein, 2009; Latombe et al., 2007). In contrast, our incremental grammar learning approach learns both the grammar structure and parameters in an unsupervised unified manner without any extra requirements of data.

3 Method

3.1 Stochastic And-Or Grammars

Stochastic And-Or grammars originate from CFGs. A CFG is defined as a 4-tuple $\langle \Sigma, N, S, R \rangle$. Σ is a set of terminal nodes representing atomic patterns that are not decomposable, while N is a set of non-terminal nodes representing decomposable patterns. $S \in N$ is a special non-terminal node named start symbol, representing the origin of the entire grammar. R is a set of grammar rules in format $A \rightarrow BCD$, where $A \in N, \{B, C, D\} \subseteq \Sigma \cup N / \{S\}$. An And-rule represents the decomposition of a pattern into a configuration of non-overlapping sub-patterns, while an Or-rule represents an alternative configuration of a composite pattern. Intuitively, if a non-terminal node is the source of multiple rules, we call this node an Or-node and these rules Or-rules.

Assigning probability distributions to Or-rules of a CFG engenders stochastic CFGs, a generalized version of CFGs. It is defined as a 5-tuple, $\langle \Sigma, N, S, R, P \rangle$, whose component P attaches probabilities to all Or-rules. Stochastic And-Or grammars are generative models characterized by P. Given a grammar, one can generate valid data by recursively sampling from the start symbol according to P.

3.2 Grammar Learning with MCTS

In both batch and incremental learning settings, we search for the best grammar by iteratively optimizing an objective function, the posterior probability of the grammar given the training data:

$$P(G|X) \propto P(G)P(X|G) \quad (1)$$

$$= \frac{1}{Z} e^{-\alpha ||G||} \prod_{x_i \in X} P(x_i|G) \quad (2)$$

where G is the grammar, $X = \{x_i\}$ is the set of i.i.d. training samples, Z is the normalization factor of the prior, α is a constant, and $||G||$ is the size of the grammar. The prior prefers compact grammars with smaller size, which we define as the sum of sizes of all rules in a grammar. In the structure learning process, we use Viterbi likelihood (the probability of the best parse of the data sample x_i) to approximate $P(x_i|G)$ (Tu et al., 2013; Tu and Honavar, 2012; Spitkovsky et al., 2010). In batch learning, the full dataset X is available to the algorithm all through, while in incremental learning, we fix the size of the algorithm's memory and replace old data with newer one after the memory is full. Thus, $P(G|\widehat{X})$ is utilized to approximate $P(G|X)$, with $||\widehat{X}|| \ll ||X||$.

Searching starts with a naive grammar G_0 that can generate X. The grammar tree of G_0 has the start symbol S as the root. S is an Or-node whose children correspond to the data sequences in X. We assign an And-node A_i for each $x_i \in X$, and A_i produces terminal symbols consisting x_i. The probability S that engenders A_i is uniformly initialized. G_0 with a single Or-node overfits to X. Our algorithm increases its prior by introducing new nonterminal nodes into the grammar and substitute redundant structures in a bottom-up manner.

At each search iteration, we substitute in a grammar fragment called And-Or Fragment (AOF) (Tu et al., 2013), which is rooted at a new nonterminal node and contains a set of grammar rules that specify how the new nonterminal node generates one or more configurations of existing terminal or non-terminal nodes. An AOF is a three-level grammar snippet rooted at an And-node, with Or-nodes in the middle and previous terminal/nonterminal nodes as leaves. We call the combinations of the leaves as the configuration of an AOF. Given a grammar G and an AOF A, we

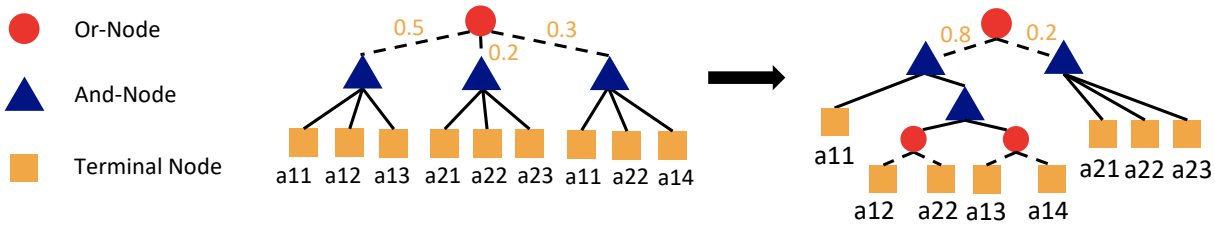


Figure 1: AOF Fragment and Its Substitution. The AOF has 4 configurations $a_{12}a_{13}$, $a_{12}a_{14}$, $a_{22}a_{13}$, and $a_{22}a_{14}$.

can get G' by substituting A into G and replacing all appearance of A's configurations in the third-level of G with the root of A. See figure 1 for an example.

Substituting an AOF into a grammar changes both the likelihood and the prior. First, a decrease in likelihood is caused by the introduction of new Or-nodes in the AOF. For a substitution, a configuration is replaced by an AOF root. Instead of generating the configuration directly without uncertainty, the new grammar needs to sample this configuration from the AOF stochastically. Hence, the likelihood is reduced by the product of the corresponding Or-nodes' probabilities. Meanwhile, after substituting an AOF in, originally distinctive data may now become identical, thus merging with each other, which increases the weights of some Or-branches coming out of the root. The likelihoods for all affected data sequences are increased by the ratio of the new weight to the old weight. Besides likelihood, substitution of new AOF also changes the grammar size. We can compute the change of the objective function as the posterior gain, which is the product of the likelihood and prior gain. Along the search process, we keep looking for the AOF with highest posterior gain and substitute it into the current grammar. Details for the likelihood gain can be found in Appendix A.

We used MCTS to improve the greedy search methods in (Tu et al., 2013). The root of the MCTS search tree is the initial grammar, and to transit from one grammar to another, we apply an AOF substitution. For a given grammar, we find a pool of AOFs that can raise its posterior and sample one from this pool according to their posterior gains. If for a grammar, no AOFs can be found to increase its posterior gain, it is considered as a terminal grammar and we do backpropagation to update the value for previous grammars in the search tree. The reward used for backpropagation is the posterior difference between the terminal grammar and the initial grammar. We use different pool sizes for MCTS expansion step and simulation step.

3.3 Incremental Learning

Several additional procedures need to be done for the incremental learning setting. First, before every round of search, new data should be merged into the existing grammar. Before merging, it must be parsed using current rules. Usually, the new data is not fully parsable, so we amended the Earley parser (Hale, 2001) to recursively check for parsable portions of a sequence and return sub-parsings. Second, the parameters of the stochastic And-Or grammar need to be updated along the search. We took advantage of online EM to optimize $\prod_{x_i \in X} P(x_i, \Delta_{x_i}|G)$ with respect to the production probabilities of grammar G, where Δ_{x_i} is the set of all valid parse graphs of x_i (Nevado et al., 2000; Latombe et al., 2007). Denote $N(O \to O_i, pg_x)$ as the indicator function that a parse graph pg_x includes the rule $O \to O_i$, $\theta(O \to O_i), \eta(O \to O_i)$ as the production probability and the averaged count of parse graphs passing through the rule.

$$\eta(O \to O_i) = \sum_{x_i \in X} \frac{\sum_{pg_{x_i} \in \Delta_{x_i}} N(O \to O_i, pg_{x_i}) P(x, pg_{x_i}|G)}{P(x_i, \Delta_{x_i}|G)} \tag{3}$$

$$\theta'(O \to O_i) = \frac{\eta(O \to O_i)}{\sum_{j=1}^{m} \eta(O \to O_j)} \tag{4}$$

The E-step (3) estimates sufficient statistics η, and the M-step (4) updates production probabilities of Or-rules. m is the total number of Or-children of O. We do a stepwise update of the sufficient statistics η as $\eta = (1 - r_t)\eta + r_t\eta'$, where r_t is a decaying learning rate (Liang and Klein, 2009).

Finally, since the grammar keeps varying when new data comes in, some structures learned by the grammar may not be optimal and are up to change as more data are seen. Therefore, we allow learned AOFs to be deleted and recovered back to

	Geoquery	Robot
Solan et al. (2005)	0.437	0.554
Tu et al. (2013)	0.500	0.520
MCTS Batch	**0.584**	**0.798**
MCTS Incremental	0.437	0.589

Table 1: Best F-score for both dataset.

their configurations, so that further tweaking of the grammar structures is possible. Pseudocode and details for the complete algorithm can be found in Appendix B.

4 Experiments

We evaluate our grammar learning approach on two datasets. One is the **Geoquery** dataset (Wong, 2007) containing 880 unique natural language descriptions of facts about the United States geography. The other is a **robot command** dataset (Dukes, 2013) with 2713 unique natural language sentences of commanding a robot to do various in-house tasks. In our experiment, We randomly split a dataset into mutually exclusive training and testing set. For the Geoquery, half of the data is used for training and half for testing. For the robot command dataset, around 1000 sentences are used for training and the remainder for testing. We evaluate our approach under both the batch setting and the incremental setting (the model's memory is fixed as 4% of the training data size). We compare our approach with two previous approaches (Solan et al., 2005; Tu et al., 2013), which can only be applied to the batch learning.

We use a specific F1-score tailored for evaluating grammars learned from unlabeled data. We define the *precision* as the ratio of data generated from the learned grammar that can be parsed by the ground truth grammar, while the *recall* as the ratio of unseen testing data that can be parsed by the learned grammar. Since the ground truth grammars are often unavailable, we utilize BiLingual Evaluation Understudy (BLEU) score (Papineni et al., 2002) as an approximation of the precision metric, i.e., how close the sentences generated by the learned grammar is to the sentences generated by the true grammar. In our evaluation, we calculate the average precision, recall, and F1-score based on 10 rounds experiments with a random split of the datasets. Details about the experiments and hyperparameters can be found in Appendix E.

Table 1 summarizes the F1-scores of our approach applied under both batch (MCTS Batch) and incremental (MCTS Incremental) settings compared with the two previous approaches' performance under the batch setting. For both datasets, our batch learning results show significant improvement over the previous approaches, and our grammars learned incrementally have comparable performances as those learned by previous approaches in the batch learning setting. Figure 2 shows the detailed Precision (BLEU)-Recall curves of our approach and the two previous approaches in the batch learning setting. Figure 3 shows the F1-Score improvement w.r.t. the number of training examples received under the incremental learning setting. As shown by the figure, our approach can effectively learn more accurate grammar model as more data has been processed but not kept.

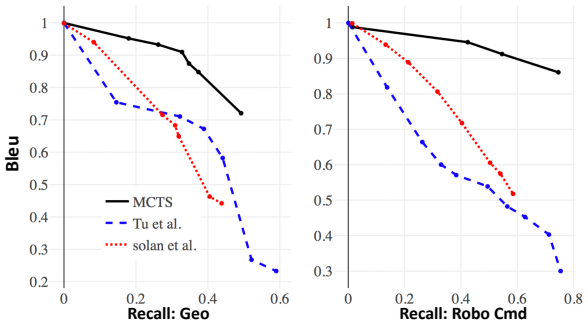


Figure 2: PR-Curves for Batch learning.

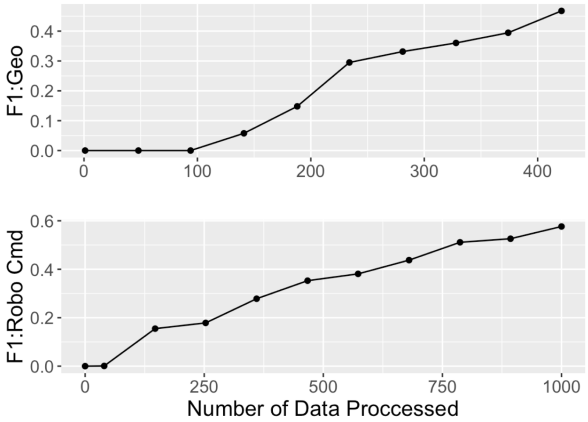


Figure 3: F-score increases as the model sees more data in incremental learning, even with fixed memory size.

5 Conclusion

In this paper, we propose an algorithm that learns the structure and parameters of stochastic And-Or grammars in a unified framework and employs MCTS to address the complexity of grammar learning. Our approach significantly improves the state-of-the-art performance for unsupervised batch learning, and supports incremental grammar learning and achieves comparable performance.

References

Claus Brabrand, Robert Giegerich, and Anders Møller. 2007. Analyzing ambiguity of context-free grammars. In *International Conference on Implementation and Application of Automata*, pages 214–225. Springer.

Eugene Charniak. 1997. Statistical techniques for natural language parsing. *AI magazine*, 18(4):33.

Nick Chater and Christopher D Manning. 2006. Probabilistic models of language processing and acquisition. *Trends in cognitive sciences*, 10(7):335–344.

Kais Dukes. 2013. Semantic annotation of robotic spatial commands. In *Language and Technology Conference (LTC)*.

E Mark Gold. 1978. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320.

John Hale. 2001. A probabilistic earley parser as a psycholinguistic model. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

I Hill III. 1979. *Introduction to automata theory, languages, and computation*. Addison Wesley, Boston, Ma.

Yuri A. Ivanov and Aaron F. Bobick. 2000. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):852–872.

Faizan Javed, Marjan Mernik, Barrett R Bryant, and Alan Sprague. 2008. An unsupervised incremental learning algorithm for domain-specific language development. *Applied Artificial Intelligence*, 22(7-8):707–729.

Guillaume Latombe, Eric Granger, and Fred A Dilkes. 2007. Incremental learning of stochastic grammars with graphical em in radar electronic support. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–301. IEEE.

Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress.

Percy Liang and Dan Klein. 2009. Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619. Association for Computational Linguistics.

Hangxin Liu, Yaofang Zhang, Wenwen Si, Xu Xie, Yixin Zhu, and Song-Chun Zhu. 2018. Interactive robot knowledge patching using augmented reality. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1947–1954. IEEE.

Katsuhiko Nakamura and Keita Imada. 2010. Incremental learning of cellular automata for parallel recognition of formal languages. In *International Conference on Discovery Science*, pages 117–131. Springer.

Katsuhiko Nakamura and Masashi Matsumoto. 2005. Incremental learning of context free grammars based on bottom-up parsing and search. *Pattern Recognition*, 38(9):1384–1392.

Francisco Nevado, Joan-Andreu Sánchez, and José-Miguel Benedí. 2000. Combination of estimation algorithms and grammatical inference techniques to learn stochastic context-free grammars. In *International Colloquium on Grammatical Inference*, pages 196–206. Springer.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Mingtao Pei, Yunde Jia, and Song-Chun Zhu. 2011. Parsing video events with goal inference and intent prediction. In *Computer vision (iccv), 2011 ieee international conference on*, pages 487–494. IEEE.

Zhangzhang Si and Song-Chun Zhu. 2013. Learning and-or templates for object recognition and detection. *IEEE transactions on pattern analysis and machine intelligence*, 35(9):2189–2205.

Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. 2005. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33):11629–11634.

Valentin I Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D Manning. 2010. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 9–17. Association for Computational Linguistics.

Andreas Stolcke and Stephen Omohundro. 1994. Inducing probabilistic grammars by bayesian model merging. In *International Colloquium on Grammatical Inference*, pages 106–118. Springer.

Kewei Tu and Vasant Honavar. 2012. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1324–1334. Association for Computational Linguistics.

Kewei Tu, Maria Pavlovskaia, and Song Chun Zhu. 2013. Unsupervised structure learning of stochastic and-or grammars. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1322–1330.

Yuk W Wong. 2007. Learning for semantic parsing and natural language generation using statistical machine translation techniques. Technical report, TEXAS UNIV AT AUSTIN DEPT OF COMPUTER SCIENCES.

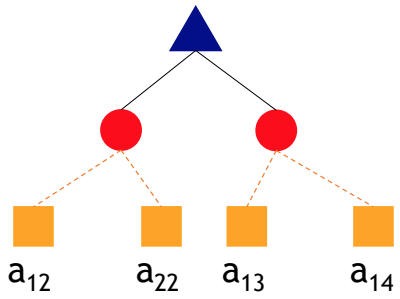
Unsupervised Incremental Structure Learning of Stochastic And-Or Grammars with Monte Carlo Tree Search (Appendix)

A Likelihood Gain

Substituting an AOF into a grammar changes the likelihood in two folds. Denote the set of replacements being made on the training samples R. First, a decrease in likelihood is caused by the introduction of new Or-nodes in the AOF. Suppose for reduction $r \in R$, a configuration $a_{1j_1}a_{2j_2}...a_{mj_m}$ is replaced by an m-gram AOF root A, where a_{ij_i} is an existing terminal or non-terminal node in the third-level of current grammar that can be generated by the new Or-node O_i in the AOF. That is, instead of generating the configuration directly without uncertainty, the new grammar needs to generate A first and then samples this configuration stochastically with a probability $P(A \rightarrow a_{1j_1}a_{2j_2}...a_{mj_m}) = \prod_{i=1}^{m} P(O_i \rightarrow a_{ij_i}) = \prod_{i=1}^{m} \theta(O_i \rightarrow a_{ij_i})$, where $\theta(O_i \rightarrow a_{ij_i})$ represents the production probability of the grammar rule $O_i \rightarrow a_{ij_i}$. Hence, the probability of generating the data including r has been reduced by a factor of $\prod_{i=1}^{m} \theta(O_i \rightarrow a_{ij_i})$. Meanwhile, after substituting an AOF in, some original distinctive data can become identical, thus merging with each other. This merging will increase the weights of some Or-branches coming out of the root of the grammar. The likelihoods for all affected data sequences are increased by the ratio of the new weight to the old weight. To facilitate the computation of this factor, we construct a context matrix CM where each row is a configuration of existing nodes covered by the AOF, each column is a context which is the surrounding patterns of a configuration, and each element is the number of times that the corresponding configuration and context co-occur in the training set. See figure 1 for a context matrix example. By combining these two factors together, we can calculate the total likelihood change of introducing an AOF, defining as **likelihood gain**:

$$\frac{P(X|G^{t+1})}{P(X|G^t)} = \frac{\prod_{i=1}^{n}\prod_{j=1}^{m_i} ||R_i(a_{ij})||^{||R_i(a_{ij})||}}{||R||^{n||R||}} \times \frac{\prod_c (\sum_e CM[e,c])^{\sum_e CM[e,c]}}{\prod_{e,c} CM[e,c]^{CM[e,c]}}$$

where G_t and G_{t+1} are the grammars before and after learning from the AOF, $R_i(a_{ij})$ denotes the subset of replacement in R in which the i-th node of the configuration being reduced is a_{ij}, e in the summation or product ranges over all the configurations covered by the And-Or fragment, and c in the product ranges over all the contexts that appear in CM. Refer to (Tu et al.) for detailed proof. An intuition is that the first fraction corresponds to the decreasing factor of the likelihood. For every Or-branch, the attached weight is the empirical ratio



Context / Config	$Context_1$	$Context_2$	.....	$Context_n$
a_{12} a_{13}	2	0	.....	3
a_{12} a_{14}	1	2	.....	1
a_{22} a_{13}	3	4	.....	0
a_{22} a_{14}	2	3	.....	1

Figure 1: Reduction and Context Matrix

of selecting a_{ij} for the $i-$th node in all configurations, which is $\frac{||R_i(a_{ij})||}{||R||}$. Multiplying over all reductions gives us the first fraction. After substitution, all data in the same column become one single new data, whose weight connecting with the root Or-node increases by a factor of $\frac{\sum_e CM[e,c]}{CM[e,c]}$. Again, multiplying over all reductions gives the second fraction.

B The Complete Algorithm

Pseudocode for both batch and incremental learning is showed in algorithm 1 - 3. One more detail is how to generate AOFs given current memory. We first generate a bi-gram AOF f by randomly sample two bi-grams from the memory and form an AOF out of them. Remember, the memory is always consistent with the third-level terminal/nonterminal nodes in the grammar. Then, on the basis of f, we optimize its posterior gain using greedy or beam search via adding/removing its Or-nodes and adding/removing its leaf nodes. As long as no operations can increase the posterior gain of f, we put f into the pool. To complete this pool, we repeat the above procedure K times, where K is a hyperparameter. For expansion steps in MCTS, we use $K = 20$ and 5 for simulation steps. Finally, we remove duplicates in the pool and sample one AOF from this pool based on their posterior gains.

C Partial Parser

We revised Earley parser (Hale, 2001) to enable partial parsing. For a given input data sequence, our partial parser starts from the first position of the data sequence, trying to parse using all rules in current grammar with the Earley parser and record all possible parsing that can form a full parse tree from this given position. The immediate position that cannot be parsed is returned. If no full parse tree can be formed given the current set of rules, we recursively delete the top level rules and obtain the subtrees of current grammar trees, until there is a grammar tree that can parse the sequence or until there is no rule left. Then we resume at the unparsed position we record previously and recursively parse the rest of the sequence. During the process, all possible parsing patterns are recorded and we select the one with the highest probabilities as the final partial parsing. Lastly, we concatenate the parsed and unparsed segments of the sequence to regenerate the data and merge it with the current stochastic grammar tree.

D Deletion of AOFs

Unlike batch learning, data in incremental learning varies during the learning process. As a result, previously learned structures may become obsolete or sub-optimal as more data are seen and the grammar has not yet been stable. To prevent our algorithm being trapped in sub-optimal, we randomly select among AOFs whose roots are in the third level of the grammar and decompose them back to their configurations, giving MCTS a chance to consider other possible arrangements of these symbols. We define a delete probability. When MCTS is doing expansions or simulations, it either tries to generate a new AOF or randomly delete an AOF following the delete probability. After an AOF been decomposed, both the grammar and the data memory need to be updated by inserting the configurations back to the position held by the root of the AOF. One thing to be noticed is that, even in the batch learning setting, deletion of AOFs can be included with much lower delete probability than that of incremental learning.

E Experiment Details

Here in table 1 and 2 we present some example sentences from the dataset together with sentences generated by our stochastic grammar.

In our experiments, we used several hyperparameters to guide the learning process. We chose $\alpha = 2.94$ for robotic commands dataset, and $\alpha = 2.2$ for Geoquery dataset, where α controls the balance between prior and likelihood as in formula (1) section 4.1. We set the deletion probability to be 0.2 for incremental learning of robotic command dataset, and 0.1 for incremental learning of Geoquery dataset. To prevent overfitting, we also restrict the simulation depth for MCTS. For batch learning, MCTS simulation is forced to stop after 20 steps for the robotic command dataset and 60 steps for the Geoyquery dataset. For incremental learning, we limit the simulation to be 10 for both datasets.

Give me the cities in virginia .
What are the high points of states surrounding mississippi ?
Count the states which have elevations lower than what alabama has .
Name the major cities in florida . [batch]
Which rivers do not flow through indiana? [incremental]
What state borders the least states excluding colorado and excluding dakota ? [incremental]

Table 1: Geoquery dataset, the top is the original data and the bottom is generated sentences.

Place green pyramid on top of red brick
Move the yellow tetrahedron on top of the red blocks in the corner nearest to the yellow tetrahedron
Pick the red pyramid which is on top of yellow brick and place it above the yellow block
Pick up the blue pyramid placed closest the green pyramid on the green pyramid [batch]
Place the red prism on top of the yellow cube [incremental]
Move the red pyramid in the white cube [incremental]

Table 2: Robot command dataset, the top is the original data and the bottom is generated sentences.

Input: Initial Grammar G_0, Dataset X, Computation Resource res
Output: Stochastic CFG G
$Gs = \{\}$
while $res > 0$ **do**
  $G_1 =$ UCT-Select(G_0) ▷ G_1 is a existing grammar in the MCTS search tree
  $G_2 =$ Expand(G_1, X) ▷ sample an AOF from G_1, X with large sample size and substitute in
  while G_2 *is not a terminal grammar* **do**
    $G_2 =$ Simulate(G_2, X) ▷ sample an AOF from G_2, X with small sample size and substitute in
  end
  $Gs \leftarrow Gs \cup \{G_2\}$
  $res \leftarrow res - 1$
end
return $\arg\max_{G \in Gs}$ Posterior(G)

Algorithm 1: MCTS searching process. Hyperparameters are omitted in this pseudocode. As described in section B, the only difference between expansion and simulation is the size of the AOF pool. Expansion steps have a larger pool size, correspondingly, take longer to run. A grammar will be considered as a terminal grammar if no AOF can be found to raise its posterior.

Input: Dataset X, Computation Resource res
Output: Stochastic CFG G
$G_0 \leftarrow$ initial grammar from X
$G \leftarrow$ MCTS(G_0, X, res)
return G

Algorithm 2: Batch learning algorithm.

References

John Hale. 2001. A probabilistic earley parser as a psycholinguistic model. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

Percy Liang and Dan Klein. 2009. Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619. Association for Computational Linguistics.

Kewei Tu, Maria Pavlovskaia, and Song-Chun Zhu. Unsupervised structure learning of stochastic and-or grammars (supplementary material).

Input: Data Stream S, Computation Resource res
Output: Stochastic CFG G
$t \leftarrow 0$
while *Data stream not finish* **do**
  $d \sim S$ ▷ incoming data d from data stream
  $G, d' \leftarrow$ PartialParse(G, d) ▷ partial parse d and merge it into G
  $\widehat{X} \leftarrow UpdateMemory(\widehat{X}, d')$ ▷ update memory by replacing old data with new data d'
  $G \leftarrow MCTS(G, \widehat{X}, res)$
  $\eta' \leftarrow$ E-Step($G, \widehat{X}$)▷ re-estimate sufficient statistics
  $G.\eta \leftarrow (1 - r_t)G.\eta + r_t\eta'$ ▷ stepwise update Or-branches' counts towards new counts
  $G \leftarrow$ M-Step(G) ▷ calculate Or-branches' weights with current counts
  $t \leftarrow t + 1$
  $res \rightarrow res - 1$
end
return G

Algorithm 3: Incremental learning algorithm. The E-Step and M-Step are implemented according to formula (3) and (4) in section 4.2. After the sufficient statistics of current memory is acquired, we use them to update the sufficient statistics of the grammar by intercepting them with previous suffcient statistics by a factor of r_t. We follow the statement from stochastic approximation literature by ensuring $\sum_{t=0}^{\infty} r_t = \infty, \sum_{t=0}^{\infty} r_t^2 = 0$ (Liang and Klein, 2009).