Adaptive Importance Sampling from Probabilistic Tree Automata

Christoph Teichmann
University of Potsdam

Kasimir Wansing
Leipzig University

Alexander Koller
University of Potsdam

{chriteich|akoller}@uni-potsdam.de
kasimir.wansing@uni-leipzig.de

Abstract

We present a general importance sampling technique for approximating expected values based on samples from probabilistic finite tree automata. The algorithm uses the samples it produces to adapt rule probabilities for the automaton in order to improve sample quality.

1 Introduction

Natural language processing (NLP) often requires the computation of expected values $\mathbb{E}_{G(T)}[f(T)] = \sum_{t \in L_A} G(T = t) f(t)$ where the random variable T takes values from the language L_A of a probabilistic regular tree automaton (pRTA) A, f measures a quantity of interest and G is a probability distribution on L_A. A tree automaton provides a natural generalization of acyclic hypergraphs and latent variable grammars that are often used in natural language processing to express e.g. a parse chart or all the ways a word could be decomposed into morphemes. For different choices of G and f we obtain e.g. (Li and Eisner, 2009):

Feature Expectations for conditional random fields.

Kullback-Leibler Divergence to compare the predictions of different probability models.

Expected Loss for minimum risk decoding.

Log-Likelihood for predicting the next word in language models.

Gradients of those quantities for optimization.

Exact computation of these values is feasible if L_A is small or additional assumptions can be made, e.g. if the expected value is defined via semiring operations on the automaton defining L_A (Li and Eisner, 2009).

Li and Eisner (2009) give an exact semiring solution if two key assumptions can be made. First the definition of the probability $G(T)$ must decompose into smaller derivation steps along the rules of A. Second the number of rules of A cannot be too large, as they must all be visited. The first assumption is violated when e.g. non-local features (Huang, 2008) are used to define probabilities or when probabilities are defined by recurrent neural nets that use hidden states derived from whole subtrees (Socher et al., 2013; Dyer et al., 2016). The second assumption is violated when e.g. tree automata are used to represent parse charts for combinatorially complex objects like in graph parsing (Groschwitz et al., 2015).

When semiring techniques are not applicable, it is necessary to use approximation techniques. One popular technique is the use of Monte Carlo methods, i.e. sampling. It is often based on Markov Chain Monte Carlo (Gamerman and Lopes, 2006) or Particle Monte Carlo (Cappé et al., 2007) approaches and requires minimal knowledge about the expected value being approximated. In this work we develop an importance sampler based on pRTAs which can be used to approximate expected values in settings where exact solutions are infeasible. One can efficiently sample from a pRTA making it a suitable tool for generating proposals for importance sampling, as we show in Section 3.2.

Good performance of importance sampling requires choosing rule probabilities for the pRTA to closely approximate the target distribution. One can attempt to derive rule probabilities that achieve this by analyzing the target distribution a priori or by using a proposal that is known to be a good fit (Dyer et al., 2016). We present a technique for self-adaption in Section 4 that allows the sampler to learn a good proposal distribution on its own. Following recent advances in adaptive importance sampling (Ryu and Boyd, 2014; Douc et al., 2007) our technique picks the best possible rule probabilities for the pRTA according to an appropriate quality measure.

Both the generation of proposals from a pRTA and the adaption procedure we propose allow for a lazy implementation that only needs to visit states seen when drawing a sample. Therefore our algorithm can deal with very large automata.

2 Tree Automata

A signature Σ is a set of *functors*, for which any $l \in \Sigma$ is assigned a positive integer as an *arity* denoted by $rank\,(l)$. A *well formed tree* over Σ has the form $l(t_1, \dots, t_{rank(l)})$ where $l \in \Sigma$ and each t_i is also a well formed tree.

2.1 Regular Tree Automata

A *probabilistic regular tree automaton* (pRTA) (Comon et al., 2007) is a tuple $A = \langle \Sigma, Q, R, q_S, \theta \rangle$ with Σ a signature, Q a finite set of *states* and $q_S \in Q$ the *start state*. R is a finite set of *rules* that have the form $q_0 \to l(q_1, \dots, q_{rank(l)})$ with each q_i taken from Q and l from Σ.

The rules in R are mapped one to one to integers in $[1, |R|]$ so that we can say, e.g. r_i is the i-th rule. $\theta \in \mathbb{R}^{|R|}$ is a vector of *parameters* for the rules and θ_i is the parameter for r_i. We call the values of θ parameters because we use them indirectly to derive probabilities. By using a vector θ and defining probabilities indirectly, we deviate from standard notation for pRTA. We do this in order to more easily express the optimization problem we will later have to solve during adaption. For a rule $r = q \to l(\dots)$ we call $lhs(r) = q$ the *left hand side* of r and use $sis\,(r)$ to denote the set of rules with the same left hand side as r. For pRTA A with parameters θ we define the probability $P^q_A(R = r_i)$ of using a rule r_i given a state q as 0 if $lhs(r_i) \neq q$ and otherwise as:

$$\frac{e^{\theta_i}}{\sum_{r_j \in sis(r)} e^{\theta_j}} \tag{1}$$

The probabilities sum to 1 for q if there is at least one rule r with $lhs(r) = q$ and therefore θ can take any value in $\mathbb{R}^{|R|}$. We shorten $P(X = x_1, \dots | Y_1 = y_1, \dots)$ to $P(x_1, \dots | y_1, \dots)$ if random variables are clear from the context.

As an example for a pRTA we will use $A_{ex} = \langle \{l, l'\}, \{q_0\}, \{r_1 = q_0 \to l(q_0, q_0), r_2 = q_0 \to l'()\}, q_0, \langle 1, 2 \rangle \rangle$. By our definition $P^{q_0}_{A_{ex}}(r_1) = \frac{e^1}{e^1 + e^2}$.

2.2 Derivations

At the core of our paper are *derivations* for a pRTA A, which are easily sampled. They are well formed trees for a signature Σ_R that is derived by using the rules of A as functors with $rank\,(q_0 \to l(q_1, \dots)) = rank\,(l)$. A derivation d for A must have a rule r with $lhs(r) = q_S$ as its root functor and any node in d with a functor of the form $q_0 \to l(q_1, \dots, q_n)$, must have an i-th child with a root functor of the form $q_i \to l'(\dots)$. For A_{ex} one derivation is $d_{ex} = r_1(r_2, r_2)$.

We say a derivation *maps* to a tree t, denoted by $m(d) = t$, if replacing all the rules in d by their functor produces t. For A_{ex} we have $m(d_{ex}) = l(l', l')$. The set of derivations from a pRTA A that map to a tree t is denoted as Δ^t_A.

The language of A is $L_A = \{t | \Delta^t_A \neq \emptyset\}$. If $|\Delta^t_A| \leq 1$ for every t then A is said to be *unambiguous*. Assuming unambiguous automata simplifies most of our discussion and we do so throughout. Therefore we can denote the single derivation for a tree t as d_t. Many tree automata used in NLP, e.g. parse charts, are unambiguous. An extension to the ambiguous case is straightforward but laborious and is left for future work.

We denote by $R(d)$ the multi-set of all the rules that occur in the derivation d, e.g. $R(d_{ex}) = \{r_1, r_2, r_2\}$. For $A = \langle \Sigma, Q, R, q_S, \theta \rangle$ we define the joint probability $P_A(T = t, D = d)$ of a derivation d and a tree t as 0 if $m(d) \neq t$ and otherwise as:

$$\prod_{r \in R(d)} P^{lhs(r)}_A(r) \tag{2}$$

Lack of ambiguity implies $P_A(t) = P_A(t, d_t)$.

2.3 Sampling Derivations

For almost all Monte Carlo techniques it is of great importance that one can efficiently generate proposals and compute their probability. This is very easy for pRTAs which is why we use them here. One can draw a sample $\langle t, d \rangle$ from $P_A(T, D)$ if it is possible to sample rules given their left hand state. Begin with a tree consisting of only the start state q_S. Until we have a derivation we replace the leftmost state q_l in our tree with a rule

$q_l \to l(q_1,\ldots,q_n)$ drawn according to $P_A^{q_l}(R)$ and make $q_1,\ldots,q_n$ its children. This produces a derivation d and tree t given by $m(d)$. Note that the probability of generating any derivation-tree pair $\langle t,d\rangle$ this way is given by the multiplication of rule probabilities matching the definition of $P_A(T,D)$. For our case this means that we can also sample $P_A(t)$.

The sampling of derivations can fail in three ways. First, there may be states for which there are no rules to expand them. The sampling procedure we have outlined is undefined when such a state is reached. Secondly, it is possible that a pRTA is deficient[1] (Nederhof and Satta, 2006). This means that there is a non-zero probability that the sampling procedure we have outlined keeps expanding without ever stopping. Finally the number of rules for a single state might be so large that it is impossible to iterate over them efficiently to select one for a sample. The techniques for solving these problems are outside the scope of this paper and we therefore make three additional assumptions in addition to our condition that a pRTA A is unambiguous:

1. For every state q in A there is at least one rule r with $lhs(r) = q$.
2. A is cycle free, i.e. no state can be expanded to reach the same state. This restricts us to finite tree languages and removes the possibility of deficiency.
3. For each state q in A we can efficiently list all the rules r with $lhs(r)$.

3 Importance Sampling

3.1 Problem Setting

The formulation for the problem we want to solve is as follows. We have access to some pRTA A, the language L_A of which is the domain over which we are trying to compute an expected value. We are also given a probability function $G(T)$ defined over L_A and some function f, which measures some quantity of interest, e.g. the number of times a certain functor occurs as a leaf in a tree, whether a constituent appears in a parse tree or the count of a feature used for training a model. We want to compute the value of $\mathbb{E}_{G(T)}[f(T)]$. Note that many problem domains in NLP can be expressed as the language of a pRTA as they can express charts that occur e.g. in context free parsing and also graph parsing (Groschwitz et al., 2015).

Li and Eisner (2009) give an algorithm for this problem that computes the expected frequency of seeing every rule in L_A under G and then exactly computes $\mathbb{E}_{G(T)}[f(T)]$ rule by rule. The algorithm they present is not applicable if G does not decompose according to the rules of the A or if A is too large to compute expectations efficiently. It can also be the case that we cannot compute $G(T)$ and instead have to use a proportional function $g(t)$ with $G(t) = \frac{g(t)}{Z}$ where Z is some normalization constant. This is the case if we are using e.g. a discriminate model that defines probabilities based on global normalization such as Conditional Random Fields (Lafferty et al., 2001). It also happens whenever we are only interested in the probability of a tree given an observation, e.g. when we want to know the probability of a parse tree given an observed sentence for EM training. If g is defined using non-local features (Huang, 2008) or using e.g. recurrent neural network structures (Dyer et al., 2016; Vinyals et al., 2014), then it is often infeasible to compute either Z or G.

The algorithm by Li and Eisner (2009) also requires f to decompose with the rules of A, which may not be the case if we are computing e.g. complex losses in loss aware training (Gimpel and Smith, 2010).

There are therefore situations in which we need a sampling algorithm that uses A to generate an approximation for $\mathbb{E}_{\frac{g(T)}{Z}}[f(T)]$ using only evaluations of g and f on single trees. This algorithm should work without computing Z or visiting all rules of A.

3.2 Solution with Importance Sampling

Importance sampling (Robert and Casella, 2004) is a well known technique based on the following fundamental equalities that hold for any set L and *auxiliary* probability distribution $P(T)$:[2]

$$\mathbb{E}_{G(T)}[f(T)] = \sum_{t\in L} \frac{G(t)f(t)}{P(t)} P(t) \quad (3)$$

$$= \mathbb{E}_{P(T)}\left[\frac{G(T)f(T)}{P(T)}\right] \quad (4)$$

On this basis we can approximate any expected value we are interested in by generating independent samples $s = t_1,\ldots,t_n$ from P, defining:

[1]Also called inconsistent.

[2]We assume $P(t) \neq 0$ for all $t \in L$

$$\mathbb{S}_{\langle t_1,\ldots,t_n\rangle}\left[h(t_i)\right] = \frac{1}{n}\sum_{i=1}^{n} h(t_i) \tag{5}$$

and using $\mathbb{S}_s\left[\frac{G(t_i)f(t_i)}{P(t_i)}\right]$ as our approximation. Under mild conditions the law of large numbers tells us that this approximation will almost surely become arbitrarily close to the correct value as the number of samples increases.

As stated before, we might be unable to compute the normalizer Z to derive G from g. We can also estimate Z from samples. Note that:

$$\begin{aligned} Z = & \sum_{t\in L} \frac{g(t)}{P(t)} P(t) & (6)\\ = & \mathbb{E}_{P(T)}\left[\frac{g(T)}{P(t)}\right] & (7)\end{aligned}$$

and we can estimate Z in a similar way to how we estimate $\mathbb{E}_{G(T)}\left[f(T)\right]$. In practice Z is estimated from the same sample s used to compute $\mathbb{S}_s\left[\frac{G(t_i)f(t_i)}{P(t_i)}\right]$. For a sample $s = t_1,\ldots,t_n$ from P we therefore approximate $\mathbb{E}_{G(T)}\left[f(T)\right]$ through the self normalized importance sampling estimate (Cappé et al., 2004) defined by :

$$\mathbb{S}_s^{\text{norm}}\left[\frac{g(t_i)f(t_i)}{P(t_i)}\right] = \sum_{i=1}^{n} \frac{f(t_i)g(t_i)\sum_{j=0}^{n}\frac{P(t_j)}{g(t_j)}}{P(t)} \tag{8}$$

If P^n is the probability of generating a sample $s = t_1,\ldots,t_n$ through independent draws from P then $\mathbb{E}_{P^n(S)}\left[\mathbb{S}_S^{\text{norm}}\left[\frac{g(t_i)f(t_i)}{P_A(t_i)}\right]\right]$ may not equal $\mathbb{E}_{G(T)}\left[f(T)\right]$ as nested estimates can create systematic bias. But the law of large numbers ensures that Z converges to its true value with increasing n and the self normalized importance sampling estimate converges on $\mathbb{E}_{G(T)}\left[f(T)\right]$.

If we want to use importance sampling over a set of trees then we need a way to specify P that allows us easy sampling and evaluation of $P(t)$ for any tree t. As we saw in Section 2 pRTAs meet these requirements. Therefore we will use P_A with respect to some unambiguous pRTA A as our proposal distribution. It must be true that $L \subseteq L_A$ and such an automaton is usually given by the problem setting, e.g. one would directly use a parse chart when computing expected values over parses. We draw samples from P_A and compute an approximation according to equation 8. Note that this algorithm does not require us to know much about f or g, we only have be be able to evaluate them both on the trees that occur in our sample, which will usually cover a small fraction of L_A. Most importantly g and f do not need to decompose for any sub-parts of t, as is often required in dynamic programming based techniques.

While the technique we have outline works no matter how we set rule probabilities for A, the term:

$$\frac{G(t_i)f(t_i)}{P_A(t_i)} \tag{9}$$

may be become comparatively small when $P_A(t_i) >> G(t_i)$ so t_i contributes almost nothing to the estimate. It may also become relatively large if $P_A(t_i) << G(t_i)$ which can lead to a single t_i "dominating" the sample, again allowing other t_j almost no contribution. In both cases we will have to draw much more samples to obtain a good estimate (Robert and Casella, 2004). Therefore we would prefer P_A to not be too different from G. Choosing rule weights to set P_A to be close to G through analysis of G may be difficult and would have to be solved for every new problem instance. Therefore we develop an algorithm that automatically adapts P_A to improve sample quality.

4 Proposal Optimization

In this section we will discuss how to learn the auxiliary distribution P_A. We will use one out of a family A_θ of pFTAs with the form $\langle \Sigma, Q, R, q_S, \theta\rangle$ for which Σ, Q, R and q_S are fixed and we only adjust θ. We write $\theta^{(i)}$ with $i \in \mathbb{N}$ to denote a fixed choice for the parameters. $A_{\theta^{(i)}}$ denotes the member of the family with $\theta = \theta^{(i)}$. As in other recent research on adaptive importance sampling (Lian, 2011; Douc et al., 2007; Gu et al., 2015) we try to find a $\theta^{(i)}$ that minimizes a measure for how bad the fit between $P_{A_{\theta^{(i)}}}$ and G is. A natural measure from information theory is the Kullback-Leibler Divergence between P_{A_θ} and G:

$$D_{KL}(G||P_{A_\theta}) = \mathbb{E}_{G(T)}\left[\log\left(\frac{G(T)}{P_{A_\theta}(T)}\right)\right] \tag{10}$$

If we subtract $G(t)\log\left(G(t)\right)$ – which does not vary with θ – and replace $G(t) = \frac{g(t)}{Z}$ with just $g(t)$ we do not change the values of θ for which minima are obtained. A simpler function with the same minima would therefore be:

$$\mathbf{o}(\theta) = \sum_{t \in L_A} \left(-g(t) \left(\sum_{r_i \in R(d_t)} \theta_i - \log \left(\sum_{r_j \in sis(r_i)} e^{\theta_j} \right) \right) \right) + \frac{\|\theta\|^2}{2\lambda} \tag{11}$$

$$\frac{\partial \mathbf{o}\left(\theta^{(n)}\right)}{\partial \theta_k} = \sum_{t \in L_A} \left(-g(t) \left(\sum_{r_i \in R(d_t)} 1(i = k) - P^{lhs(r_i)}_{A_{\theta^{(n)}}}(r_k) \right) \right) + \frac{\theta_k^{(n)}}{\lambda} \tag{12}$$

$$o_{\langle \theta^{(n)}, k \rangle}(t) = -g(t) \left(\sum_{r_i \in R(d_t)} 1(i = k) - P^{lhs(r_i)}_{A_{\theta^{(n)}}}(r_k) \right) \tag{13}$$

$$\sum_{t \in L_A} -g(t) \left(\log \left(P_{A_\theta}(t) \right) \right) \tag{14}$$

Note that this function is no longer dependent on the complicated normalization constant Z and we can therefore ignore Z for our optimization algorithm. We will attempt to find the optimal θ iteratively. To ensure that any estimate $P_{A_{\theta^{(i)}}}$ assigns nonzero weight to any tree in L_A we add a regularization term to our objective. As a result we are going to attempt to find parameters θ that minimize the objective function given by (11). λ is a configuration parameter used to trade off between fitting G and spreading out rule probabilities evenly. Note that unambiguous automata are important to derive this convex objective function.

4.1 Optimization

Like Ryu and Boyd (2014) we use *stochastic gradient descent* (SGD) (Bottou, 1998). In SGD we attempt to find a minimum for $\mathbf{o}(\theta)$ by generating a series $\theta^{(1)}, \ldots, \theta^{(m)}$ of approximations to a stationary point according to:

$$\theta^{(n+1)} = \theta^{(n)} - \left(\alpha^{(n)} \odot \overline{\nabla \mathbf{o}\left(\theta^{(n)}\right)} \right) \tag{15}$$

Here $\overline{\nabla \mathbf{o}\left(\theta^{(n)}\right)}$ is an approximation of the gradient of $\mathbf{o}(\theta)$ at the point $\theta^{(n)}$, $\alpha^{(n)}$ is a vector of pre-set learning rates and $\odot$ denotes element-wise multiplication of the two arguments to the operator. If we can efficiently obtain an estimate for $\nabla \mathbf{o}\left(\theta^{(n)}\right)$ then we can run SGD for a number of iterations and optimize θ. Standard derivative rules show that the kth dimension of $\nabla \mathbf{o}\left(\theta^{(n)}\right)$ takes the form given by (12), where $1(x)$ is 1 if x is true and 0 otherwise. Note that, if we ignore the contribution of the regularization term, the gradient becomes smaller the more the probability of a rule given its left hand side matches the same value according to $G(T)$. Intuitively this means that the rule weights adapt to match the expected frequency of a rule given its parent which is reminiscent of the use of expected rule counts in the technique of (Li and Eisner, 2009).

```
Given $G$, $f$, $A_\theta$ and population size $ps$
  1. set $\theta^{(1)} = \mathbf{0}$
  2. for $n \in [1, m]$ do:
      (a) generate a sample of trees
          $s^{(n)} = t_1^n, \ldots, t_{ps}^n$ from $P_{A_{\theta^{(n)}}}$
      (b) estimate gradient $\overline{\nabla \mathbf{o}\left(\theta^{(n)}\right)}_k$ as
          $\mathbb{S}_{s^{(n)}} \left[ \frac{o_{\langle \theta^{(n)}, k \rangle}(t_i)}{P_{A_{\theta^{(n)}}}(t_i)} \right] + \frac{\theta_k^{(n)}}{\lambda}$
      (c) generate learning rates $\alpha^{(n)}$
      (d) set $\theta^{(n+1)}$ according to (15)
  3. compute an importance estimate for
     $\mathbb{E}_{G(T)}\left[f(T)\right]$.
```

Algorithm 1: The adaptive importance sampling algorithm for pRTAs.

The gradient is a sum over a fairly simple function of trees and we can denote the "contribution" from a single tree t to a gradient dimension k by $o_{\langle \theta^{(n)}, k \rangle}(t)$ given by (13). We therefore use the iterative procedure given in Figure 1 for to chose a good $A_{\theta^{(n)}}$ and produce an estimate.

For the last step in our algorithm we can generate an additional final sample from $A_{\theta^{(m)}}$. A more efficient use of computation resources is to use all the samples $s^{(n)}$ to create m importance sampling estimates for $\mathbb{E}_{G(T)}\left[f(T)\right]$ and then take a weighted average. The use of a weighed average is due to the fact that early values for $\theta^{(n)}$ are likely to be of poorer quality than the final ones and there might be less merit to their contribution. The number of iteration steps m, the number ps of samples drawn in each iteration and the learning rates $\alpha^{(n)}$ must be specified externally. The parameters that need to be configured are usually much easier to set than values for θ, e.g. ps and m should simply be set as large as computationally feasible for optimal performance.

At first it may seem as if Step 2d requires us to adjust all parameters of the automaton at the same time. This would make it expensive to apply our algorithm to large automata. However we can add the gradient contribution of the regularization in a lazy fashion as in Carpenter (2008). This means that we only update rules seen in the sample. The regularization also forces parameters that have not been updated for a while towards 0, allowing us to remove these from explicit storage. As a result we can sample with very large pRTAs as proposal distributions.

4.2 Convergence

Before we give evaluation results we should make some statements on the convergence behavior of our algorithm. By the law of large numbers we can state:

Theorem 1. *In Algorithm 1 we almost surely have for each n:* $\lim_{ps\rightarrow\infty} \mathbb{S}^{norm}_{s^{(n)}}\left[\frac{g(t_i)f(t_i)}{P_{A_{\theta^{(n)}}}}\right] = \mathbb{E}_{G(T)}\left[f(T)\right]$.

Estimates will converge to $\mathbb{E}_{G(T)}\left[f(T)\right]$ as stated by Theorem 1 even if SGD did not improve the fit between P_{A_θ} and G. But it would be reassuring if we could show that the $\theta^{(n)}$ tend towards a minimum for $\mathbf{o}\left(\theta\right)$ as n increases towards infinity. This can be shown by standard convergence conditions for SGD. The objective function given by equation 11 is actually strictly convex. This is the case because the square norm is strictly convex and a logarithm over a sum of exponentials is convex. All other terms in the formula are either constant or linear in θ and therefore also convex. The following conditions ensure convergence to an optimal setting of θ (for details see Bottou (1998)):

1. For every dimension k and sample size ps:
$$\mathbb{E}_{P^{ps}_{A^{\theta^{(n)}}}(S)}\left[\mathbb{S}_{s^{(n)}}\left[\frac{o_{\langle\theta^{(n)},k\rangle}(t_i)}{P_{A_{\theta^{(n)}}}(t_i)}\right]\right]$$
equals $\overline{\nabla\mathbf{o}\left(\theta^{(n)}\right)}_k$
2. $\mathbf{o}\left(\theta\right)$ is bounded from below.
3. For every dimension k it is true that $\alpha^{(n)}_k > 0$ for all n and $\sum_{n=0}^{\infty}(\alpha^{(n)}_k)^2 < \infty$, $\sum_{n=0}^{\infty}\alpha^{(n)}_k = \infty$.
4. The second moment of our estimate for $\nabla\mathbf{o}\left(\theta^{(n)}\right)$ is bounded by a linear function of $||\theta^{(n)}||^2$.

Condition 3 depends only on the α which may be chosen by the user to be i.e. $\frac{1}{n}$ to meet the requirement. Condition 2 is true as the Kullback-Leibler Divergence is never smaller than 0 and the same is true for the Euclidean norm. Condition 4 can be verified by inspecting the gradient and noting that L_A is finite by the assumptions we made in Section 2 and that all the terms in the sum are bounded by either 1,-1 or $\theta^{(n)}$. Condition 1 can be verified by simple algebra from our exposition of importance sampling. As a result we can state:

Theorem 2. *For Algorithm 1 we almost surely have:*
$$\lim_{n\rightarrow\infty}\theta^{(n)} = \operatorname*{argmin}_{\theta}\boldsymbol{o}\left(\theta\right)$$

This means that the rule probabilities converge towards values that are optimal as measured by the Kullback-Leibler Divergence.

5 Evaluation

We created an implementation of our approach in order to investigate its behavior. It, along with all evaluation data and documentation, is available via `https://bitbucket.org/tclup/alto/wiki/AdaptiveImportanceSampling` as part of Alto, an Interpreted Regular Tree Grammar (Koller and Kuhlmann, 2011) toolkit. We encourage readers to use the sampler in their own experiments. In this section we will evaluate the ability of our algorithm to learn good proposal weights in an artificial data experiment that specifically focuses on this aspect. This will show whether the theoretical guarantees correspond to practical benefits. In future work we will evaluate the algorithm in End-to-End NLP experiments to see how it interacts with a larger tool chain.

5.1 Evaluation Problem

We tested our approach by computing a constant expected value $\mathbb{E}_{G(T)}\left[1\right]$. The ratio $\frac{1*G(T)}{P_{A_\theta}(T)}$ be-

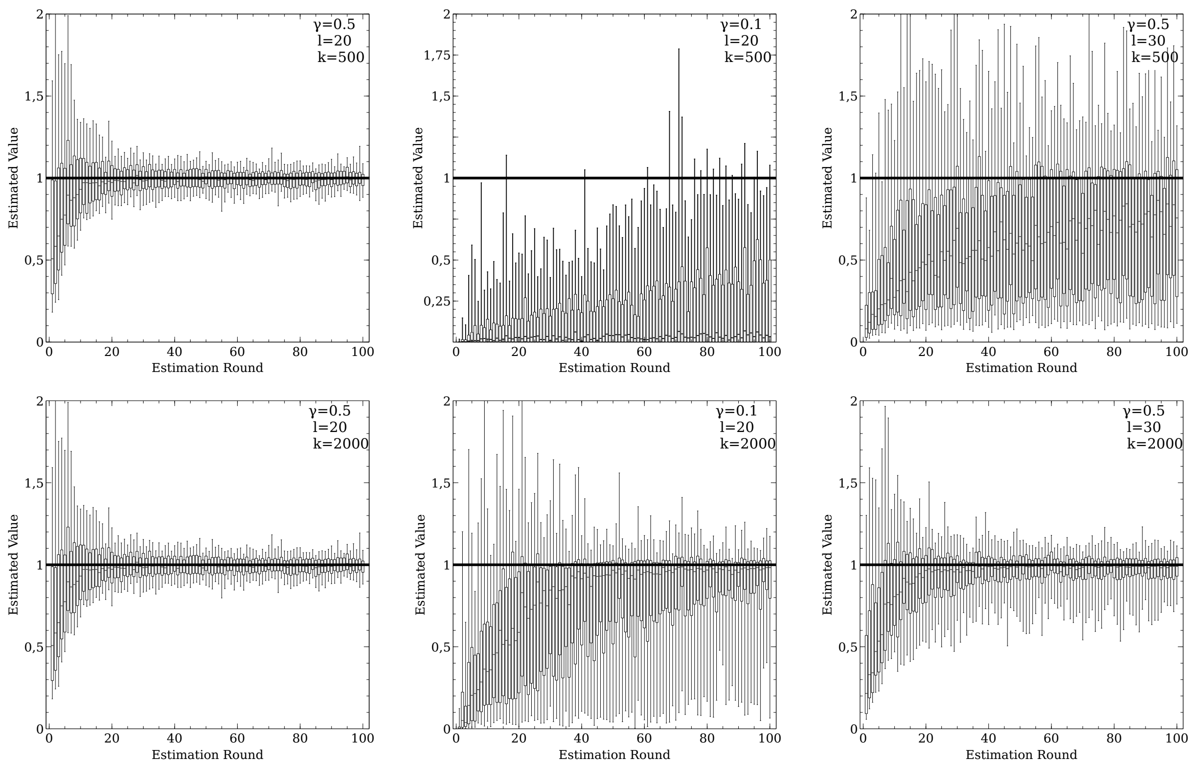

Figure 1: Convergence plots for different problem settings.

comes 1 if P_{A_θ} perfectly matches G and a single sample would suffice for an exact estimate. Therefore any error in the estimate is directly due to a mismatch between G and P_{A_θ}. We defined G through a pRTA with the same structure as the automaton we are sampling from. Therefore it is possible for P_{A_θ} to exactly match G. If we were told the correct parameters for θ then we would obtain a perfect sample in the first step and a good sample should be generated if the SGD steps successfully move θ from $\mathbf{0}$ to the values defining G.

The parameters for G were randomly chosen so that the resulting probabilities for rules given their left hand sides where distributed according to a symmetric Dirichlet Distribution. This is a good model for probability distributions that describe natural language processes (Goldwater et al., 2011). The symmetric Dirichlet Distribution is parametrized by a concentration parameter γ. The rule weights become more likely to be concentrated on a few of the rules for each left hand side as γ goes toward 0. Therefore many trees will be improbable according to G.

We obtain a complete evaluation problem by giving the structure of the automata used. As stated, we use the same automaton to specify G and A_θ save for θ which we initialize to be 0 for all entries. We chose a structure similar to a CKY parse chart for the underlying rules and states (Younger, 1967). Given a *length parameter* $l \in \mathbb{N}$ we added to the automaton all states of the form $\langle i, j\rangle$ with $0 \leq i < j \leq l$. The automaton has all rules of the form $\langle i, i+1\rangle \rightarrow i()$ and all rules $\langle i, j\rangle \rightarrow *(\langle i, h\rangle, \langle h, j\rangle)$ with $0 \leq i < h < j \leq l$ and $*$ an arbitrary functor. Therefore the parameters for our evaluation problems are l and γ.

A central concern in making SGD based algorithms efficient is the choice of $\alpha^{(n)}$. We use an adaptive strategy for configuring the learning rates (Duchi et al., 2010; Schaul et al., 2012). These schemes usually have convergence proofs that require much stricter conditions than "vanilla" SGD and we therefore cannot claim that $\theta^{(n)}$ will converge with these approaches, but in practice they often perform well. Concretely, we use the technique for setting learning rates that was introduced by Duchi et al. (2010) which uses α' and divides it by the sum of all the gradient estimates seen so far to obtain the learning rate for each dimension. α' is fixed ahead of time – we chose 0.5 and we found that values between 1.0 and 0.1 could be used interchangeably to obtain the best performance. We set the regularization parameter λ in our objective $\mathbf{o}(\theta)$ to 100 for comparatively weak regularization.

5.2 Evaluation Results

Figure 1 plots the convergence behavior of the algorithm for different problem settings with fixed parameters l and γ. The upper row shows experiments with 500 samples and the lower uses 2000 samples. Each plot gives results for a single example automaton. Experiments with different automata showed the same trends. Each graph shows 100 repetitions of the experiment as box plots. Different random number seed were used for the repetitions. The box plots show how well the expected value was approximated in each of the $m = 100$ rounds of adaption. The value in each round/repetition n is computed only for the sample $s^{(n)}$ drawn in that round/repetition. Whiskers indicate the 9th and 91th percentile value.

Note the tendency towards underestimation in all experiments. This indicates that the algorithm proposes many trees with low probability under G and has to adapt in order to find more likely trees.

For $l = 20$ and $\gamma = 0.5$ and $k = 500$ the sampler converges in 40 iterations. Note that performance after four steps with $k = 500$ is better than with one step of $k = 2000$. This shows that adapting parameters provides a benefit over simply increasing the sample size. With $\gamma = 0.1$, $l = 20$ and $k = 500$ the samples improve much more slowly. This is to be expected as there are more than 1 billion trees in L_A and only very few of them will be assigned large probabilities by the more peaky rule probabilities. Therefore the algorithm has to randomly find these few trees to produce a good estimate both for the evaluated value and for the gradient used in adaptation. When $k = 2000$ there are faster improvements as the algorithm has a better gradient estimate.

Convergence is also slower when l is increased to 30 as the number of trees to consider rises and the amount of parameters in θ grows in the order of $O(l^3)$. Convergence speed again increases if we set the sample size $k = 2000$.

Overall we can see that the adaption steps improve the quality of our importance sampler and lead to a simple, yet versatile algorithm for approximating expected values.

6 Related Work

Sampling in NLP is most often implemented via Markov Chain Monte Carlo methods that either have to move through the relevant domain with small steps (Chung et al., 2013) or use a good proposal distribution in order to generate new trees (Johnson et al., 2007). Because it is difficult to adapt Markov Chain Monte Carlo algorithms (Liang et al., 2010) the proposal distribution for generating new trees needs to be specified by the user in advance. Particle Monte Carlo Methods (Cappé et al., 2007; Börschinger et al., 2012) are related to importance sampling and would allow for more adaptive proposals, but have not been used this way for the structured outputs used in natural language processing. The idea of using adaptive versions of importance sampling has become much more prevalent in the last years (Douc et al., 2007; Lian, 2011; Ryu and Boyd, 2014). Ryu and Boyd (2014) discussed the use of SGD to optimize a convex function in order to improve an importance sampler. They discussed applications where the proposal distribution is from the exponential family of distributions and used the variance of their sampler as the optimization objective. Douc et al. (2007) and Lian (2011) used an objective based on Kullback-Leibler divergence, but they used techniques other than SGD to optimize the objective. It is also possible to use more complex models to generate proposals (Gu et al., 2015) at the price of less efficient training.

7 Conclusion

We have presented an adaptive importance sampler that can be used to approximate expected values taken over the languages of probabilistic regular tree automata. These values play a central role in many natural language processing applications and cannot always be computed analytically. Our sampler adapts itself for improved performance and only requires the ability to evaluate all involved functions on single trees. To achieve adaptiveness, we have introduced a convex objective function which does not depend on a complex normalization term. We hope that this simple technique will allow researchers to use more complex models in their research.

Acknowledgments. We thank the anonymous reviewers for their comments. We thank the workshop organizers for their patients in waiting for our final version. We received valuable feedback from Jonas Groschwitz and Martn Villalba. This work was supported by the DFG grant KO 2916/2-1.

References

Benjamin Börschinger, Katherine Demuth, and Mark Johnson. 2012. Studying the effect of input size for Bayesian Word Segmentation on the Providence Corpus. In *Proceedings of COLING 2012*, pages 325–340.

Léon Bottou. 1998. Online Algorithms and stochastic approximations. In David Saad, editor, *Online Learning in Neural Networks*, pages 9–13. Cambridge University Press. revised, oct 2012.

Olivier Cappé, Arnaud Guillin, Jean-Michel Marin, and Christian Robert. 2004. Population Monte Carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929.

Olivier Cappé, Simon J. Godsill, and Eric Moulines. 2007. An overview of existing methods and recent advances in Sequential Monte Carlo. In *Proceedings of the IEEE*, volume 95, pages 899–924.

Bob Carpenter. 2008. Lazy sparse Stochastic Gradient Descent for Regularized Multinomial Logistic Regression. Technical report, Alias-i, Inc.

Tagyoung Chung, Licheng Fang, Daniel Gildea, and Daniel Štefankovič. 2013. Sampling tree fragments from forests. *Computational Linguistics*, 40(1):203–229.

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi, and Christof Löding. 2007. *Tree Automata techniques and applications*. published online - `http://tata.gforge.inria.fr/`.

Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian Robert. 2007. Convergence of adaptive mixtures of Importance Sampling schemes. *The Annals of Statistics*, 35(1):420–448.

John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive Subgradient methods for Online Learning and stochastic optimization. In *COLT 2010 - The 23rd Conference on Learning Theory*, pages 257–269.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent Neural Network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209.

Dani Gamerman and Hedibert F. Lopes. 2006. *Markov Chain Monte Carlo - Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC.

Kevin Gimpel and Noah A. Smith. 2010. Softmax-Margin Training for structured Log-Linear Models. Technical report, Carnegie Mellon University.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2011. Producing Power-Law Distributions and damping Word Frequencies with two-stage language models. *Journal of Machine Learning Research*, 12:2335–2382.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with S-graph Grammars. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1481–1490.

Shixiang Gu, Zoubin Ghahramani, and Richard E. Turner. 2015. Neural adaptive Sequential Monte Carlo. In *Advances in Neural Information Processing Systems 28*, pages 2629–2637.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. 2007. Bayesian Inference for PCFGs via Markov Chain Monte Carlo. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 139–146. The Association for Computational Linguistics.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 2–13.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Zhifei Li and Jason Eisner. 2009. First- and second-order Expectation Semirings with applications to Minimum-Risk training on Translation Forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 40–51.

Heng Lian. 2011. Stochastic adaptation of Importance Sampler. *Statistics*, 46(6):777–785.

Faming Liang, Chuanhai Liu, and Raymond Carroll. 2010. *Advanced Markov Chain Monte Carlo Methods*. John Wiley and Sons Ltd.

Mark-Jan Nederhof and Giorgio Satta. 2006. Estimation of consistent Probabilistic Context-free Grammars. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 343–350.

Christian P. Robert and George Casella. 2004. *Monte Carlo statistical methods*. Springer, 2 edition.

Ernest K. Ryu and Stephen P. Boyd. 2014. Adaptive Importance Sampling via stochastic Convex Programming. *CoRR, abs/1412.4845*.

Tom Schaul, Sixin Zhang, and Yann LeCun. 2012. No more pesky learning rates. *CoRR*, abs/1206.1106.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2014. Grammar as a foreign language. *CoRR*, abs/1412.7449.

Daniel H. Younger. 1967. Recognition and parsing of Context-Free Languages in time n^3. *Information and Control*, 10:189–208.