Exact Sampling and Decoding in High-Order Hidden Markov Models

Simon Carter[*]
ISLA, University of Amsterdam
Science Park 904, 1098 XH Amsterdam,
The Netherlands
s.c.carter@uva.nl

Marc Dymetman **Guillaume Bouchard**
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan, France
{first.last}@xrce.xerox.com

Abstract

We present a method for exact optimization and sampling from high order Hidden Markov Models (HMMs), which are generally handled by approximation techniques. Motivated by adaptive rejection sampling and heuristic search, we propose a strategy based on sequentially refining a lower-order language model that is an upper bound on the true model we wish to decode and sample from. This allows us to build tractable variable-order HMMs. The ARPA format for language models is extended to enable an efficient use of the *max-backoff* quantities required to compute the upper bound. We evaluate our approach on two problems: a SMS-retrieval task and a POS tagging experiment using 5-gram models. Results show that the same approach can be used for exact optimization and sampling, while explicitly constructing only a fraction of the total implicit state-space.

1 Introduction

In NLP, sampling is important for many real tasks, such as: (i) diversity in language generation or machine translation (proposing multiple alternatives which are not clustered around a single maximum); (ii) Bayes error minimization, for instance in Statistical Machine Translation (Kumar and Byrne, 2004); (iii) learning of parametric and non-parametric Bayesian models (Teh, 2006).

However, most practical sampling algorithms are based on MCMC, i.e. they are based on local moves starting from an initial valid configuration. Often, these algorithms are stuck in local minima, i.e. in a basin of attraction close to the initialization, and the method does not really sample the whole state space. This is a problem when there are ambiguities in the distribution we want to sample from: by having a local approach such as MCMC, we might only explore states that are close to a given configuration.

The necessity of exact sampling can be questioned in practice. Approximate sampling techniques have been developed over the last century and seem sufficient for most purposes. However, the cases where one actually knows the quality of a sampling algorithm are very rare, and it is common practice to forget about the approximation and simply treat the result of a sampler as a set of i.i.d. data. Exact sampling provides a de-facto guarantee that the samples are truly independent. This is particularly relevant when one uses a cascade of algorithms in complex NLP processing chains, as shown by (Finkel et al., 2006) in their work on linguistic annotation pipelines.

In this paper, we present an approach for exact decoding and sampling with an HMM whose hidden layer is a high-order language model (LM), which innovates on existing techniques in the following ways. First, it is a *joint* approach to sampling and optimization (i.e. decoding), which is based on introducing a simplified, "optimistic", version $q(x)$ of the underlying language model $p(x)$, for which it is tractable to use standard dynamic programming techniques both for sampling and optimization. We then formulate the problem of sampling/optimization with the original model $p(x)$ in

[*]This work was conducted during an internship at XRCE.

terms of a novel algorithm which can be viewed as a form of *adaptive rejection sampling* (Gilks and Wild, 1992; Gorur and Teh, 2008), in which a low acceptance rate (in sampling) or a low ratio $p(x^*)/q(x^*)$ (in optimization, with x^* the argmax of q) leads to a refinement of q, i.e., a slightly more complex and less optimistic q but with a higher acceptance rate or ratio.

Second, it is the first technique that we are aware of which is able to perform *exact sampling* with such models. Known techniques for sampling in such situations have to rely on approximation techniques such as Gibbs or Beam sampling (see e.g. (Teh et al., 2006; Van Gael et al., 2008)). By contrast, our technique produces exact samples from the start, although in principle, the first sample may be obtained only after a long series of rejections (and therefore refinements). In practice, our experiments indicate that a good acceptance rate is obtained after a relatively small number of refinements. It should be noted that, in the case of exact optimization, a similar technique to ours has been proposed in an image processing context (Kam and Kopec, 1996), but without any connection to sampling. That paper, written in the context of image processing, appears to be little known in the NLP community.

Overall, our method is of particular interest because it allows for exact decoding and sampling from HMMs where the applications of existing dynamic programming algorithms such as Viterbi decoding (Rabiner, 1989) or Forward-Backward sampling (Scott, 2002) are not feasible, due to a large state space.

In Section 2, we present our approach and describe our joint algorithm for HMM sampling/optimization, giving details about our extension of the ARPA format and refinement procedure. In Section 3 we define our two experimental tasks, SMS-retrieval and POS tagging, for which we present the results of our joint algorithm. We finally discuss perspectives and conclude in Section 4.

2 Adaptive rejection sampling and heuristic search for high-order HMMs

Notation Let $x = \{x_1, x_2, ...x_\ell\}$ be a given hidden state sequence (e.g. each x_i is an English word) which takes values in $\mathcal{X} = \{1, \cdots, N\}^\ell$ where ℓ is the length of the sequence and N is the number of latent symbols. Subsequences $(x_a, x_{a+1}, \cdots, x_b)$ are denoted by x_a^b, where $1 \leq a \leq b \leq \ell$. Let $o = \{o_1, o_2, ...o_\ell\}$ be the set of observations associated to these words (e.g. o_i is an acoustic realization of x_i). The notations p, q and q' refer to unnormalized densities, i.e. non-negative measures on $\mathcal{X}$. Since only discrete spaces are considered, we use for short $p(x) = p(\{x\})$. When the context is not ambiguous, sampling according to p means sampling according to the distribution with density $\bar{p}(x) = \frac{p(x)}{p(\mathcal{X})}$, where $p(\mathcal{X}) = \int_{\mathcal{X}} p(x)dx$ is the total mass of the unnormalized distribution p.

Sampling The objective is to sample a sequence with density $\bar{p}(x)$ proportional to $p(x) = p_{\text{lm}}(x)p_{\text{obs}}(o|x)$ where p_{lm} is the probability of the sequence x under a n-gram model and $p_{\text{obs}}(o|x)$ is the probability of observing the noisy sequence o given that the correct/latent sequence is x. Assuming the observations depend only on the current state, this probability becomes

$$p(x) = \prod_{i=1}^{\ell} p_{\text{lm}}(x_i|x_{i-n+1}^{i-1})p_{\text{obs}}(o_i|x_i) \ . \quad (1)$$

To find the most likely sequence given an observation, or to sample sequences from Equation 1, standard dynamic programming techniques are used (Rabiner, 1989; Scott, 2002) by expanding the state space at each position. However, as the transition order n increases, or the number of latent tokens N that can emit to each observation o_l increases, the dynamic programming approach becomes intractable, as the number of operations increases exponentially in the order of $\mathcal{O}(\ell N^n)$.

If one can find a proposal distribution q which is an upper bound of p — i.e such that $q(x) \geq p(x)$ for all sequences $x \in \mathcal{X}$ — and which it is easy to sample from, the standard *rejection sampling* algorithm can be used:

1. Sample $x \sim q/q(\mathcal{X})$, with $q(\mathcal{X}) = \int_{\mathcal{X}} q(x)dx$;
2. Accept x with probability $p(x)/q(x)$, otherwise reject x;

To obtain multiple samples, the algorithm is repeated several times. However, for simple bounds,

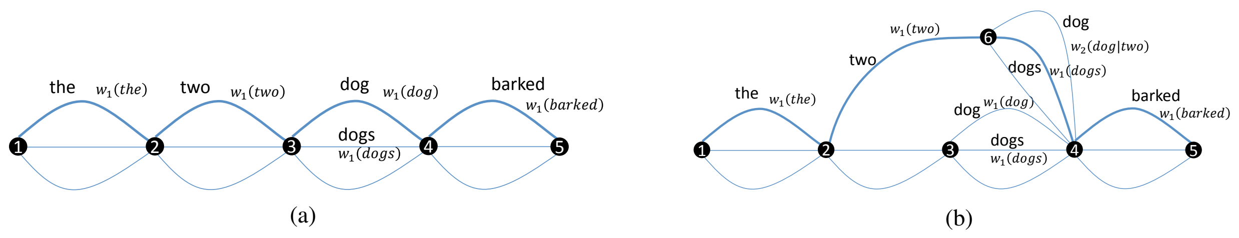

Figure 1: *An example of an initial q-automaton (a), and the refined q-automaton (b) Each state corresponds to a context (only state 6 has a non-empty context) and each edge represents the emission of a symbol. Thick edges are representing the path for the sampling/decoding of* `two dog(s) barked`, *thin edges corresponding to alternative symbols. By construction,* $w_1(\texttt{dog}) \geq w_2(\texttt{dog}\,|\,\texttt{two})$ *so that the total weight of (b) is smaller than the total weight of (a).*

the average acceptance rate — which is equal to $p(\mathcal{X})/q(\mathcal{X})$ — can be so large that rejection sampling is not practical. In *adaptive rejection sampling* (ARS), the initial bound q is incrementally improved based on the values of the rejected elements. While often based on log-concave distributions which are easy to bound, ARS is valid for any type of bound, and in particular can be applied to the upper bounds on n-gram models introduced by (Kam and Kopec, 1996) in the context of optimization. When a sample is rejected, our algorithm assumes that a small set of refined proposals is available, say $q'_1, \cdots, q'_m$, where m is a small integer value. These refinements are *improved* versions of the current proposal q in the sense that they still upper-bound the target distribution p, but their mass is strictly smaller than the mass of q, i.e. $q'(\mathcal{X}) < q(\mathcal{X})$. Thus, each such refinement q', while still being optimistic relative to the target distribution p, has higher average acceptance rate than the previous upper bound q. A bound on the n-gram LM will be presented in Section 2.1.

Optimization In the case of optimization, the objective is to find the sequence maximizing $p(x)$. Viterbi on high-order HMMs is intractable but we have access to an upper bound q, for which Viterbi is tractable. Sampling from q is then replaced by finding the maximum point x of q, looking at the ratio $r(x) = p(x)/q(x)$, and accepting x if this ratio is equal to 1, otherwise refining q into q' exactly as in the sampling case. This technique is able to find the exact maximum of p, similarly to standard heuristic search algorithms based on optimistic bounds. We stop the process when q and p agree at the value maximizing q which implies that we have found the global maximum.

2.1 Upper bounds for n-gram models

To apply ARS on the target density given by Equation 1 we need to define a random sequence of proposal distributions $\{q^{(t)}\}_{t=1}^{\infty}$ such that $q^{(t)}(x) \geq p(x), \forall x \in \mathcal{X}, \forall t \in \{0, 1, \cdots\}$. Each n-gram $x_{i-n+1}, ..., x_i$ in the hidden layer contributes an n-th order factor $w_n(x_i|x_{i-n+1}^{i-1}) \equiv p_{\text{lm}}(x_i|x_{i-n+1}^{i-1})p_{\text{obs}}(o_i|x_i)$. The key idea is that these n-th order factors can be upper bounded by factors of order $n-k$ by maximizing over the head (i.e. prefix) of the context, as if part of the context was "forgotten". Formally, we define the *max-backoff weights* as:

$$w_{n-k}(x_i|x_{i-n+1+k}^{i-1}) \equiv \max_{x_{i-n+1}^{i-n+k}} w_n(x_i|x_{i-n+1}^{i-1}), \tag{2}$$

By construction, the max-backoff weights w_{n-k} are factors of order $n-k$ and can be used as surrogates to the original n-th order factors of Equation (1), leading to a nested sequence of upper bounds until reaching binary or unary factors:

$$\begin{aligned} p(x) &= \Pi_{i=1}^{\ell} w_n(x_i|x_{i-n+1}^{i-1}) && (3)\\ &\leq \Pi_{i=1}^{\ell} w_{n-1}(x_i|x_{i-n+2}^{i-1}) && (4)\\ &\cdots \\ &\leq \Pi_{i=1}^{\ell} w_2(x_i|x_{i-1}) && (5)\\ &\leq \Pi_{i=1}^{\ell} w_1(x_i) := q^{(0)}(x) \ . && (6) \end{aligned}$$

Now, one can see that the loosest bound (6) based on unigrams corresponds to a completely factorized distribution which is straightforward to sample and optimize. The bigram bound (5) corresponds to a standard HMM probability that can be efficiently decoded (using Viterbi algorithm) and sampled (using backward filtering-forward sampling). [1] In the context of ARS, our initial proposal $q^{(0)}(x)$ is set to

[1]Backward filtering-forward sampling (Scott, 2002) refers to the process of running the Forward algorithm (Rabiner,

the unigram bound (6). The bound is then incrementally improved by adaptively refining the max-backoff weights based on the values of the rejected samples. Here, a refinement refers to the increase of the order of *some* of the max-backoff weights in the current proposal (thus most refinements consist of n-grams with heterogeneous max-backoff orders, not only those shown in equations (3)-(6)). This operation tends to tighten the bound and therefore increase the acceptance probability of the rejection sampler, at the price of a higher sampling complexity. The are several possible ways of choosing the weights to refine; in Section 2.2 different refinement strategies will be discussed, but the main technical difficulty remains in the efficient exact optimization and sampling of a HMM with n-grams of variable orders. The construction of the refinement sequence $\{q^{(t)}\}_{t\geq 0}$ can be easily explained and implemented through a Weighted Finite State Automaton (WFSA) referred as a *q-automaton*, as illustrated in the following example.

Example We give now a high-level description of the refinement process to give a better intuition of our method. In Fig. 1(a), we show a WFSA representing the initial proposal $q^{(0)}$ corresponding to an example with an acoustic realization of the sequence of words (`the`, `two`, `dogs`, `barked`). The weights on edges of this q-automaton correspond to the unigram max-backoffs, so that the total weight corresponds to Equation (6). Considering sampling, we suppose that the first sample from $q^{(0)}$ produces $x_1 =$ (`the`, `two`, `dog`, `barked`), marked with bold edges in the drawing. Now, computing the ratio $p(x_1)/q^{(0)}(x_1)$ gives a result much below 1, because from the viewpoint of the full model p, the trigram (`the two dog`) is very unlikely; in other words the ratio $w_3(\texttt{dog}|\texttt{the two})/w_1(\texttt{dog})$ (and, in fact, already the ratio $w_2(\texttt{dog}|\texttt{two})/w_1(\texttt{dog})$) is very low. Thus, with high probability, x_1 is rejected. When this is the case, we produce a refined proposal $q^{(1)}$, represented by the WFSA in Fig. 1(b), which takes into account the more realistic weight $w_2(\texttt{dog}|\texttt{two})$ by adding a node (node 6) for the context `two`. We then perform a sampling trial with $q^{(1)}$, which this time tends to avoid producing `dog` in the context of `two`; if the new sample is rejected, the refinement process continues until we start observing that the acceptance rate reaches a fixed threshold value. The case of optimization is similar. Suppose that with $q^{(0)}$ the maximum is x_1, then we observe that $p(x_1)$ is lower than $q^{(0)}(x_1)$, reject suboptimal x_1 and refine $q^{(0)}$ into $q^{(1)}$.

1989), which creates a lattice of forward probabilities that contains the probability of ending in a latent state at a specific time t, given the subsequence of previous observations o_1^t, for all the previous latent sub-sequences x_1^{t-1}, and then recursively moving backwards, sampling a latent state based on these probabilities.

Algorithm 1 ARS for HMM algorithm.

```
1:  while not Stop($h$) do
2:      if Optimisation then
3:          Viterbi $x \sim q$
4:      else
5:          Sample $x \sim q$
6:      $r \leftarrow p(x)/q(x)$
7:      Accept-or-Reject($x, r$)
8:      Update($h, x$)
9:      if Rejected($x$) then
10:         for all $i \in \{2, \cdots, \ell\}$ do
11:             $q \leftarrow$ UpdateHMM ($q, x, i$)
12: return $q$ along with accepted $x$'s in $h$
```

Algorithm 2 UpdateHMM

```
Input: A triplet $(q, x, i)$ where $q$ is a WFSA, $x$ is a sequence determining a unique path in the WFSA and $i$ is a position at which a refinement must be done.
1:  $n$ :=ORDER$_i(x_1^i) + 1$  #implies $x_{i-n+2}^{i-1} \in S_{i-1}$
2:  if $x_{i-n+1}^{i-1} \notin S_{i-1}$ then
3:      CREATE-STATE($x_{i-n+1}^{i-1}, i-1$)
4:      #move incoming edges, keeping WFSA deterministic
5:      for all s $\in$ SUF$_{i-2}(x_{i-n+1}^{i-2})$ do
6:          $e$ := EDGE($s, x_{i-1}$)
7:          MOVE-EDGE-END(e,$x_{i-n+1}^{i-1}$)
8:      #create outgoing edges
9:      for all $(s, l, \omega) \in T_i(x_{i-n+2}^{i-1})$ do
10:         CREATE-EDGE($x_{i-n+1}^{i-1}$,$s$,$l$,$\omega$)
11: #update weights
12: for all s $\in$ SUF$_{i-1}(x_{i-n+1}^{i-1})$ do
13:     weight of EDGE($s, x_i$) := $w_n(x_i|x_{i-n+1}^{i-1})$
14: return
```

2.2 Algorithm

We describe in detail the algorithm and procedure for updating a q-automaton with a max-backoff of longer context.

Algorithm 1 gives the pseudo-code of the sampling/optimization strategy. On line 1, h represents the history of all trials so far, where the stopping criterion for decoding is whether the last trial in the history has been accepted, and for sampling whether the ratio of accepted trials relative to all trials exceeds a certain threshold. The WFSA is initialized so that all transitions only take into account the $w_1(x_i)$ max-backoffs, i.e. the initial optimistic-bound ignores all contexts. Then depending on whether we are sampling or decoding, in lines 2-5, we draw an event from our automaton using either the Viterbi algorithm or Forward-Backward sampling. If the sequence is rejected at line 7, then the q-automaton is updated in lines 10 and 11. This is done by expanding all the factors involved in the sampling/decoding of the rejected sequence x to a higher order. That is, while sampling or decoding the automaton using the current proposal $q^{(t)}$, the contexts used in the path of the rejected sequence are replaced with higher order contexts in the new refined proposal $q^{t+1}(x)$.

The update process of the q-automaton represented as a WFSA is described in Algorithm 2. This procedure guarantees that a lower, more realistic weight is used in *all* paths containing the n-gram x^i_{i-n+1} while decoding/sampling the q-automaton, where n is the order at which x^i_{i-n+1} has been expanded so far. The algorithm takes as input a max-backoff function, and refines the WFSA such that any paths that include this n-gram have a smaller weight thanks to the fact that higher-order max-backoff have automatically smaller weights.

The algorithm requires the following functions:

- ORDER$_i(x)$ returns the order at which the n-gram has been expanded so far at position i.
- S_i returns the states at a position i.
- $T_i(s)$ returns end states, labels and weights of all edges that originate from this state.
- SUF$_i(x)$ returns the states at i which have a suffix matching the given context x. For empty contexts, all states at i are returned.
- EDGE(s, l) returns the edge which originates from s and has label l. Deterministic WFSA, such as those used here, can only have a single transition with a label l leaving from a state s.
- CREATE-STATE(s, i) creates a state with name s at position i, CREATE-EDGE(s_1, s_2, l, ω) creates an edge (s_1, s_2) between s_1 and s_2 with weight ω and label l, and MOVE-EDGE-END(e, s) sets the end of edge e to be the state s, keeping the same starting state, weight and label.

At line 1, the expansion of the current n-gram is increased by one so that we only need to expand contexts of size $n-2$. Line 2 checks whether the context state exists. If it doesn't it is created at lines 3-10. Finally, the weight of the edges that could be involved in the decoding of this n-gram are updated to a smaller value given by a higher-order max-backoff weight.

The creation of a new state in lines 3-10 is straightforward: At lines 5-7, incoming edges are moved from states at position $i-2$ with a matching context to the newly created edge. At lines 9-10 edges heading out of the context state are created. They are simply copied over from all edges that originate from the suffix of the context state, as we know these will be legitimate transitions (i.e we will always transition to a state of the same order or lower).

Note that we can derive many other variants of Algorithm 2 which also guarantee a smaller total weight for the q-automaton. We chose to present this version because it is relatively simple to implement, and numerical experiments comparing different refinement approaches (including replacing the max-backoffs with the highest-possible context, or picking a single "culprit" to refine) showed that this approach gives a good trade-off between model complexity and running time.

2.3 Computing Max-Backoff Factors

An interesting property of the max-backoff weights is that they can be computed recursively; taking a 3-gram LM as an example, we have:

$$\begin{aligned} w_1(x_i) &= \max_{x_i-1} w_2(x_i|x_{i-1}) \\ w_2(x_i|x_{i-1}) &= \max_{x_i-2} w_3(x_i|x^{i-1}_{i-2}) \\ w_3(x_i|x^{i-1}_{i-2}) &= p(x_i|x^{i-1}_{i-2})\, p(o_i|x_i). \end{aligned}$$

The final $w_3(x_i|x^{i-1}_{i-2})$ upper bound function is simply equal to the true probability (multiplied by the conditional probability of the observation), as any extra context is discarded by the 3-gram language model. It's easy to see that as we refine $q^{(t)}$ by replacing existing max-backoff weights with more specific contexts, the $q^{(t)}$ tends to p at t tends to infinity.

In the HMM formulation, we need to be able to efficiently compute at run-time the max-backoffs $w_1(\texttt{the})$, $w_2(\texttt{dog}|\texttt{the})$, $\cdots$, taking into account smoothing. To do so, we present a novel method for converting language models in the standard ARPA format used by common toolkits such as (Stolcke, 2002) into a format that we can use. The ARPA file format is a table T composed of three columns: (1) an n-gram which has been observed in the training corpus, (2) the log of the conditional probability of the last word in the n-gram given the previous words (log f(.)), and (3) a backoff weight (bow(.)) used when unseen $n-$grams 'backoff' to this n-gram. [2]

The probability of any n-gram x_{i-n}^{i} (in the previous sense, i.e. writing $p(x_{i-n}^{i})$ for $p(x_i|x_{i-n}^{i-1})$) is then computed recursively as:

$$p(x_{i-n}^{i}) = \begin{cases} \mathrm{f}(x_{i-n}^{i}) & \text{if } x_{i-n}^{i} \in T \\ \mathrm{bow}(x_{i-n}^{i-1})\, p(x_{i-n+1}^{i}) & \text{otherwise.} \end{cases} \tag{7}$$

Here, it is understood that if x_{i-n}^{i-1} is in T, then its bow(.) is read from the table, otherwise it is taken to be 1.

Different smoothing techniques will lead to different calculations of $\mathrm{f}(x_{i-n}^{i})$ and $\mathrm{bow}(x_{i-n}^{i-1})$, however both backoff and linear-interpolation methods can be formulated using the above equation.

Starting from the ARPA format, we pre-compute a new table MAX-ARPA, which has the same lines as ARPA, each corresponding to an n-gram x_{i-n}^{i} observed in the corpus, and the same f and bow, but with two additional columns: (4) a max log probability (log $\mathrm{mf}(x_{i-n}^{i})$), which is equal to the maximum log probability over all the n-grams extending the context of x_{i-n}^{i}, i.e. which have x_{i-n}^{i} as a suffix; (5) a "max backoff" weight ($\mathrm{mbow}(x_{i-n}^{i})$), which is a number used for computing the max log probability of an n-gram not listed in the table. From the MAX-ARPA table, the max probability w of any n-gram x_{i-n}^{i}, i.e the maximum of $p(x_{i-n-k}^{i})$ over all n-grams extending the context of x_{i-n}^{i}, can then be computed recursively (again very quickly) as:

$$w(x_{i-n}^{i}) = \begin{cases} \mathrm{mf}(x_{i-n}^{i}) & \text{if } x_{i-n}^{i} \in T \\ \mathrm{mbow}(x_{i-n}^{i-1})\, p(x_{i-n}^{i}) & \text{otherwise.} \end{cases} \tag{8}$$

Here, if x_{i-n}^{i-1} is in T, then its mbow(.) is read from the table, otherwise it is taken to be 1. Also note that the procedure calls p, which is computed as described in Equation 7. [3]

3 Experiments

In this section we empirically evaluate our joint, exact decoder and sampler on two tasks; SMS-retrieval (Section 3.1), and supervised POS tagging (Section 3.2).

3.1 SMS-Retrieval

We evaluate our approach on an SMS-message retrieval task. A latent variable $x \in \{1, \cdots, N\}^{\ell}$ represents a sentence represented as a sequence of words: N is the number of possible words in the vocabulary and ℓ is the number of words in the sentence. Each word is converted into a sequence of numbers based on a mobile phone numeric keypad. The standard character-to-numeric function num : $\{\texttt{a}, \texttt{b}, \cdots, \texttt{z}, \texttt{.}, \cdots, \texttt{?}\} \rightarrow \{1, 2, \cdots, 9, 0\}$ is used. For example, the words dog and fog are represented by the sequence $(3, 6, 4)$ because num(d)=num(f)=3, num(o)=6 and num(g)=4. Hence, observed sequences are sequences of numeric strings separated by white spaces. To take into account typing errors, we assume we observe a noisy version of the correct numeric sequence $(\mathrm{num}(x_{i1}), \cdots, \mathrm{num}(x_{i|x_i|}))$ that encodes the word x_i at the i-th position of the sentence x. The noise model is:

$$p(o_i|x_i) \propto \prod_{t=1}^{|x_i|} \frac{1}{k * d(o_{it}, \mathrm{num}(x_{it})) + 1}, \tag{9}$$

where $d(a, b)$ is the physical distance between the numeric keys a and b and k is a user provided con-

[2]See `www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html`, last accessed at 1/3/2012, for further details.

[3]In this discussion of the MAX-ARPA table we have ignored the contribution of the observation $p(o_i|x_i)$, which is a constant factor over the different max-backoffs for the same x_i and does not impact the computation of the table.

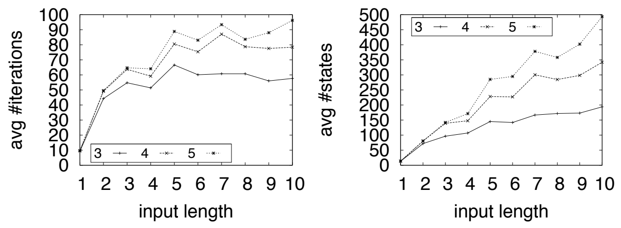

Figure 2: *On the left we report the average # of iterations taken to decode given different LMs over input sentences of different lengths, and on the right we show the average # of states in the final q-automaton once decoding is completed.*

stant that controls the ambiguity in the distribution; we use 64 to obtain moderately noisy sequences.

We used the English side of the Europarl corpus (Koehn, 2005). The language model was trained using SRILM (Stolcke, 2002) on 90% of the sentences. On the remaining 10%, we randomly selected 100 sequences for lengths 1 to 10 to obtain 1000 sequences from which we removed the ones containing numbers, obtaining a test set of size 926.

Decoding Algorithm 1 was run in the optimization mode. In the left plot of Fig. 2, we show the number of iterations (running Viterbi then updating q) that the different n-gram models of size 3, 4 and 5 take to do exact decoding of the test-set. For a fixed sentence length, we can see that decoding with larger n-gram models leads to a sub-linear increase w.r.t. n in the number of iterations taken. In the right plot of Fig. 2, we show the average number of states in our variable-order HMMs.

To demonstrate the reduced nature of our q-automaton, we show in Tab. 1 the distribution of n-grams in our final model for a specific input sentence of length 10. The number of n-grams in the full model is $\sim 3.0 \times 10^{15}$. Exact decoding here is not tractable using existing techniques. Our HMM has only 9008 n-grams in total, including 118 5-grams.

n:	1	2	3	4	5
q:	7868	615	231	176	118

Table 1: *# of n-grams in our variable-order HMM.*

Finally, we show in Tab. 2 an example run of our algorithm in the optimization setting for a given input. Note that the weight according to our q-automaton for the first path returned by the Viterbi algorithm is high in comparison to the true log probability according to p.

Sampling For the sampling experiments, we limit the number of latent tokens to 100. We refine our q-automaton until we reach a certain fixed cumulative acceptance rate (AR). We also compute a rate based only on the last 100 trials (AR-100), as this tends to better reflect the current acceptance rate.

In Fig. 3a, we plot a running average of the ratio at each iteration over the last 10 trials, for a single sampling run using a 5-gram model for an example input. The ratios start off at 10^{-20}, but gradually increase as we refine our HMM. After $\sim$ 500 trials, we start accepting samples from p. In Fig. 3b, we show the respective ARs (bottom and top curves respectively), and the cumulative # of accepts (middle curve), for the same input. Because the cumulative accept ratio takes into account all trials, the final AR of 17.7% is an underestimate of the true accept ratio at the final iteration; this final accept ratio can be better estimated on the basis of the last 100 trials, for which we read AR-100 to be at around 60%.

We note that there is a trade-off between the time needed to construct the forward probability lattice needed for sampling, and the time it takes to adapt the variable-order HMM. To resolve this, we propose to use batch-updates: making B trials from the same q-automaton, and then updating our model in one step. By doing this, we noted significant speed-ups in sampling times. In Tab. 3, we show various

input:			3637 843 66639 39478 *
oracle:			does the money exist ?
best:			does the money exist .
Viterbi paths		log q(x)	log p(x)
q^1	does the money exist)	-0.11	-17.42
q^{50}	does the **owned** exist **.**	-11.71	-23.54
q^{100}	**ends** the money exist **.**	-12.76	-17.09
q^{150}	does **vis** money exist **.**	-13.45	-23.74
q^{170}	does the money exist **.**	-13.70	-13.70

Table 2: *Viterbi paths given different q^t. Here, for the given input, it took 170 iterations to find the best sequence according to p, so we only show every 50th path.*

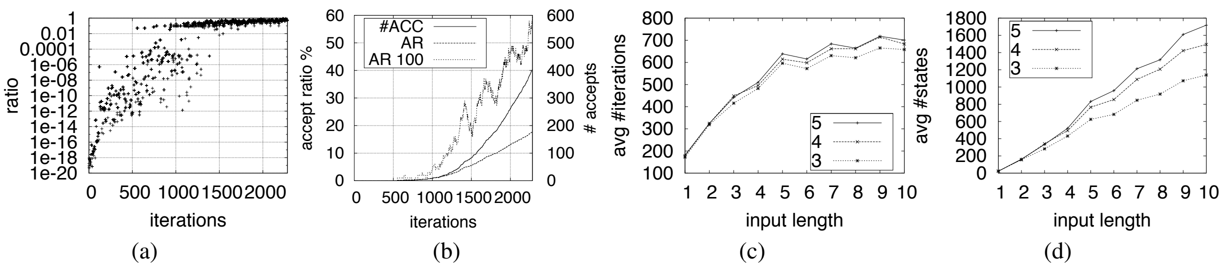

Figure 3: *In 3a, we plot the running average over the last 10 trials of the ratio. In 3b, we plot the cumulative # of accepts (middle curve), the accept rate (bottom curve), and the accept rate based on the last 100 samples (top curve). In 3c, we plot the average number of iterations needed to sample up to an AR of 20% for sentences of different lengths in our test set, and in 3d, we show the average number of states in our HMMs for the same experiment.*

B:	1	10	20	30	40	50	100
time:	97.5	19.9	15.0	13.9	12.8	12.5	11.4
iter:	453	456	480	516	536	568	700

Table 3: *In this table we show the average amount of time in seconds and the average number of iterations (iter) taken to sample sentences of length 10 given different values of B.*

input:	3637 843 66639 39478 *		
oracle:	does the money exist ?		
best:	does the money exist .		
samples	#	log q(x)	log p(x)
does the money exist .	429	-13.70	-13.70
does the money exist **?**	211	-14.51	-14.51
does the money exist **!**	72	-15.49	-15.49
does the **moody** exist .	45	-15.70	-15.70
does the money exist **:**	25	-16.73	-16.73

Table 4: *Top-5 ranked samples for an example input. We highlight in bold the words which are different to the Viterbi best of the model. The oracle and best are not the same for this input.*

statistics for sampling up to AR-100 $= 20$ given different values for B. We ran this experiment using the set of sentences of length 10. A value of 1 means that we refine our automaton after each rejected trial, a value of 10 means we wait until rejecting 10 trials before updating our automaton in one step. We can see that while higher values of B lead to more iterations, as we do not need to re-compute the forward trellis needed for sampling, the time needed to reach the specific AR threshold actually decreases, from 97.5 seconds to 11.4 seconds, an 8.5% speedup. Unless explicitly stated otherwise, further experiments use a $B = 100$.

We now present the full sampling results on our test-set in Fig. 3c and 3d, where we show the average number of iterations and states in the final models once refinements are finished (AR-100=20%) for different orders n over different lengths. We note a sub-linear increase in the average number of trials and states when moving to higher n; thus, for length=10, and for $n = 3, 4, 5$, # trials: 3-658.16, 4-683.3, 5-700.9, and # states: 3-1139.5, 4-1494.0, 5-1718.3.

Finally, we show in Tab. 4, the ranked samples drawn from an input sentence, according to a 5-gram LM. After refining our model up to AR-100 $= 20\%$, we continued drawing samples until we had 1000 exact samples from p (out of $\sim$ 4.7k trials). We show the count of each sequence in the 1000 samples, and the log probability according to p for that event. We only present the top-five samples, though in total there were 90 unique sequences sampled, 50 of which were only sampled once.

3.2 POS-tagging

Our HMM is the same as that used in (Brants, 2001); the emission probability of a word given a POS tag x_i is calculated using maximum likelihood techniques. That is, $p(o_i|x_i) = \frac{c(o_i,x_i)}{c(x_i)}$. Unseen words are handled by interpolating longer suffixes with shorter, more general suffixes. To train our language model, we use the SRILM toolkit (Stolcke, 2002) We build LMs of up to size 9. We present results on the WSJ Penn Treebank corpus (Marcus et al., 1993). We use sections 0-18 to train our emission and transitions probabilities, and report results on

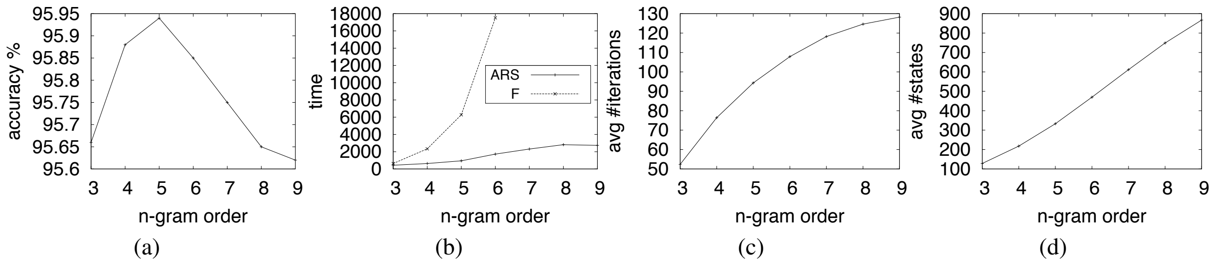

Figure 4: *In 4a, we report the accuracy results given different n-gram models on the WSJ test-set. In 4b, we show the time taken (seconds) to decode the WSJ test-set given our method (ARS), and compare this to the full model (F). In 4c, the average number of iterations needed to sample the test-set given different n-gram language models is given, and 4d shows the average number of states in the variable-order HMMs.*

sections 22-24.

We first present results for our decoding experiments. In Fig. 4a we show the accuracy results of our different models on the WSJ test-set. We see that the best result is achieved with the 5-gram LM giving an accuracy of 95.94%. After that, results start to drop, most likely due to over-fitting of the LM during training and an inability for the smoothing technique to correctly handle this.

In Fig. 4b, we compare the time it takes in seconds to decode the test-set with the full model at each n-gram size; that is a WFSA with all context states and weights representing the true language model log probabilities. We can see that while increasing the n-gram model size, our method (ARS) exhibits a linear increase in decoding time, in contrast to the exponential factor exhibited when running the Viterbi algorithm over the full WFSA (F). Note for n-gram models of order 7 and higher, we could not decode the entire test set as creating the full WFSA was taking too long.

Finally in both Figs 4c and 4d, we show the average number of iterations taken to sample from the entire test-test, and the average number of states in our variable-order HMMs, with AR-100=60%. Again we note a linear increase in both Fig., in contrast to the exponential nature of standard techniques applied to the full HMM.

4 Conclusion and Perspectives

We have presented a dual-purpose algorithm that can be used for performing exact decoding and sampling on high-order HMMs. We demonstrated the validity of our method on SMS-retrieval and POS examples, showing that the "proposals" that we obtain require only a fraction of the space that would result from explicitly representing the HMM. We believe that this ability to support exact inference (both approximation and sampling) at a reasonable cost has important applications, in particular when moving from inference to learning tasks, which we see as a natural extension of this work.

By proposing a common framework for sampling and optimization our approach has the advantage that we do not need separate skills or expertise to solve the two problems. In several situations, we might be interested not only in the most probable sequence, but also in the distribution of the sequences, especially when diversity is important or in the presence of underlying ambiguities.

The interplay between optimization and sampling is a fruitful area of research that can lead to state-of-the art performances on inference and decoding tasks in the special case of high-order HMM decoding, but the method is generic enough to be generalized to many others models of interest for NLP applications. One family of models is provided by agreement-based models, for example HMM+PCFG, where distribution p takes the form of a product: $p(x) = p_{\text{HMM}}(x)p_{\text{PCFG}}(x)$. Even if the factors $p_{\text{HMM}}(x)$ and $p_{\text{PCFG}}(x)$ can be decoded and sampled efficiently, the product of them is intractable. Dual decomposition is a generic method that has been proposed for handling decoding (i.e. optimization) with such models, by decoupling the problem into two alternating steps that can each be handled by dynamic programming or other polynomial-time algorithms (Rush et al., 2010), an approach that has been applied to Statistical Machine Translation (phrase-based (Chang and Collins,

2011) and hierarchical (Rush and Collins, 2011)) among others. However, sampling such distributions remains a difficult problem. We are currently extending the approach described in this paper to handle such applications. Again, using ARS on a sequence of upper bounds to the target distribution, our idea is to express one of the two models as a context free grammar and incrementally compute the intersection with the second model, maintaining a good trade-off between computational tractability of the refinement and a reasonable acceptance rate.

References

Thorsten Brants. 2001. Tnt - a statistical part-of-speech tagger. In *Proceedings of the Sixth conference of Applied Natural Language Processing (ANLP 2001)*, pages 224–231.

Yin-Wen Chang and Michael Collins. 2011. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP 2011)*.

Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: approximate bayesian inference for linguistic annotation pipelines. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2006)*, pages 618–626.

W. R. Gilks and P. Wild. 1992. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, 42(2):337–348.

Dilan Gorur and Yee Whye Teh. 2008. Concave convex adaptive rejection sampling. Technical report, Gatsby Computational Neuroscience Unit.

Anthony C. Kam and Gary E. Kopec. 1996. Document image decoding by heuristic search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:945–950.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit (MT-Summit 2005)*, pages 79–86.

Shankar Kumar and William Byrne. 2004. Minimum bayes risk decoding for statistical machine translation. In *Joint Conference of Human Language Technologies and the North American chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: the penn treebank. *Computional Linguistics*, 19:313–330.

Lawrence R. Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February.

Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP 2011)*, pages 26–37.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP 2010)*.

Steven L. Scott. 2002. Bayesian methods for hidden markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, 97:337–351.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. In *Proceedings of the International Conference of Spoken Language Processing (INTERSPEECH 2002)*, pages 257–286.

Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Yee Whye Teh. 2006. A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (ACL 2006)*, pages 985–992.

Jurgen Van Gael, Yunus Saatci, Yee Whye Teh, and Zoubin Ghahramani. 2008. Beam sampling for the infinite hidden Markov model. In *Proceedings of the International Conference on Machine Learning (ICML 2008)*, volume 25.