Reified Context Models

Jacob Steinhardt JSTEINHARDT@CS.STANFORD.EDU
Percy Liang PLIANG@CS.STANFORD.EDU
Stanford University, 353 Serra Street, Stanford, CA 94305 USA

Abstract

A classic tension exists between exact inference in a simple model and approximate inference in a complex model. The latter offers expressivity and thus accuracy, but the former provides coverage of the space, an important property for confidence estimation and learning with indirect supervision. In this work, we introduce a new approach, *reified context models*, to reconcile this tension. Specifically, we let the choice of factors in a graphical model (the contexts) be random variables inside the model itself. In this sense, the contexts are reified and can be chosen in a data-dependent way. Empirically, we show that our approach obtains expressivity and coverage on three sequence modeling tasks.

1. Introduction

Many structured prediction tasks across natural language processing, computer vision, and computational biology can be formulated as that of learning an exponential family distribution over outputs $y_{1:L} = (y_1, \ldots, y_L) \in \mathcal{Y}_{1:L}$ given input x:

$$p_\theta(y_{1:L} \mid x) \propto \exp\left(\sum_{i=1}^{L} \theta^\top \phi_i(y_{1:i-1}, y_i, x)\right), \qquad (1)$$

where ϕ_i are the features and θ are the parameters. The thirst for expressive models (e.g., where y_i depends heavily on its context $y_{1:i-1}$) often leads one down the route of approximate inference, for example, to Markov chain Monte Carlo (Brooks et al., 2011), sequential Monte Carlo (Cappé et al., 2007), or beam search (Koehn et al., 2003). While these methods can operate on models with arbitrary amounts of context (ϕ_i can be any function of $y_{1:i-1}$), they use a set of concrete points to approximate the distribution p_θ, and thus they can fundamentally cover only a small part of the output space $\mathcal{Y}_{1:L}$. This inadequate *coverage* leads to many issues; the two that we highlight in this paper are the following:

Figure 1. In handwriting recognition, we want to output a letter at each position. Our model also chooses a *context* at each position that summarizes the outputs so far (e.g., ⋆olc remembers only the last three letters) and is used for predicting the next letter. By selecting contexts at multiple levels of resolution, our model can place probability mass over the entire output space while still capturing complex dependencies.

- **precision**: In user-facing applications, it is important to only predict on inputs where the system is confident, leaving hard decisions to the user (Zhang et al., 2014). Lack of coverage means failing to consider all alternative outputs, which leads to overconfidence and poor estimates of uncertainty.
- **indirect supervision**: When only part of the output $y_{1:L}$ is observed, lack of coverage is even more problematic than in the fully-supervised setting. An approximate inference algorithm might not even consider the true y (whereas one always has the true y in a fully-supervised setting), leading to invalid parameter updates (Yu et al., 2013).

Of course, lower-order models admit exact inference and ensure coverage, but these models have unacceptably low expressive power. Ideally, we would like a model that varies the amount of context in a judicious way, allocating modeling power to parts of the input that demand it. Therein lies the principal challenge: How can we adaptively choose the amount of context for each variable i in a data-dependent way while maintaining tractability?

In this paper, we introduce a new approach, which we call *reified context models*. The key idea is inspired by *reification*, a general idea in logic and programming languages,

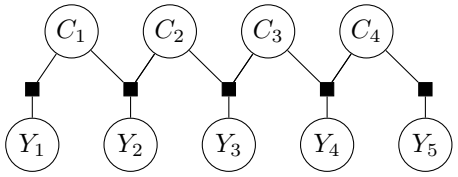
Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

which refers to making something previously unaccessible (e.g., functions or metadata of functions) a "first-class citizen" and therefore available (e.g., via lambda abstraction or reflection) to formal manipulation. In our probabilistic modeling setting, we propose reifying the contexts as random variables in the model so that we can perform inference to choose the contexts in a data-dependent way.

Specifically, suppose we have a collection of *contexts* $\mathcal{C}_i$ for each $i \in \{1, \ldots, L-1\}$. Each context $c \in \mathcal{C}_i$ is a subset of $\mathcal{Y}_{1:i}$ representing what we remember about the past (see Figure 1 for a full example). We define a joint model over (y, c). Suppressing x for brevity:

$$p_\theta(y_{1:L}, c_{1:L-1}) \propto \exp\left(\sum_{i=1}^{L} \theta^\top \phi_i(c_{i-1}, y_i)\right) \kappa(y, c), \tag{2}$$

where κ is a *consistency* potential, to be explained later. The features ϕ_i now depend on the current context c_{i-1}, rather than the full history $y_{1:i-1}$. The distribution over (y, c) factorizes according to the graphical model below:



(3)

The factorization (3) implies that the family in (2) admits efficient exact inference via the forward-backward algorithm as long as each collection $\mathcal{C}_i$ has small cardinality.

Adaptive selection of context sets. But how do we select the context sets $\mathcal{C}_i$? First, we will show that various static choices of $\mathcal{C}_i$ recover classic dynamic programming and beam search algorithms. But ideally, we would like to choose $\mathcal{C}_i$ in a data-dependent way. We propose a method, called RCMS (Section 4), which performs a forward pass similar to beam search to choose the most promising context sets. Importantly, unlike beam search, we still obtain coverage of the output space because we are selecting *contexts* rather than setting individual variables.

The goal of this paper is to flesh out the framework described above, providing intuitions about its use and exploring its properties empirically. To this end, we start in Section 2 by defining some tasks that motivate this work. In Sections 3 and 4 we introduce reified context models formally, together with an algorithm, RCMS, for selecting context sets in a data-dependent way. Sections 5-7 explore the empirical properties of the RCMS method. We discuss future research directions in Section 9. Finally, to showcase the ease of implementation of our method, we provide implementation details and runtime comparisons in the supplementary material, as well as runnable source code in our CodaLab worksheet.

2. Description of Tasks

To better understand the motivation for our work, we present three tasks of interest, which are also the tasks used in our empirical evaluation later. These tasks are handwriting recognition (a fully supervised task), speech recognition (an indirectly supervised task), and decipherment (an unsupervised task). The first of these tasks is relatively easy while the latter two are harder. We use handwriting recognition to study the precision properties of our method, and the other two tasks to explore learning under indirect supervision.

Handwriting recognition. The first task is the handwriting recognition task from Kassel (1995); we use the "clean" version of the dataset from Weiss & Taskar (2010). This contains 6,876 examples, split into 10 folds (numbered 0 to 9); we used fold 1 for testing and the rest for training. Each input is a sequence of 16×8 binary images; the output is the corresponding sequence of characters. These characters form a word without its first letter (to avoid dealing with capitalization). Since this task ended up being too easy as given, we downsampled each image to be 8×4 (by taking all pixels whose coordinates were both odd). An example input and output is given below:

input x										
output y	r	o	j	e	c	t	i	o	n	s

Each individual image is too noisy to interpret in isolation, and so leveraging the context of the surrounding characters is crucial to achieving high accuracy.

Speech recognition (decoding). Our second task is from the Switchboard speech transcription project (Greenberg et al., 1996). Our dataset consists of 999 utterances, split into two chunks of sizes 746 and 253; we used the latter chunk as a test set. Each utterance is a phonetic input and textual output:

input x	h# y ae ax s w ih r dcl d h#
latent z	(alignment)
output y	yeah_it's_weird

Note that the alignment between the input and output is unobserved. The average input length is 26 phonemes, or 2.5 seconds of speech. We removed most punctuation from the output, except for spaces, apostrophes, dashes, and dots.

Decipherment. Our final task is a decipherment task similar to that described in Nuhn & Ney (2014). In decipherment, one is given a large amount of plain text and a smaller amount of cipher text; the latter is drawn from the same distribution as the former but is then passed through a 1-to-1 substitution cipher. For instance, the plain text sen-

tence "I am what I am" might be enciphered as "13 5 54 13 5":

latent z	I	am	what	I	am
output y	13	5	54	13	5

The task is to reverse the substitution cipher, e.g. determine that $13 \mapsto I$, $5 \mapsto am$, etc.

We created a dataset from the English Gigaword corpus (Graff & Cieri, 2003) by finding the 500 most common words and filtering for sentences that only contained those words. This left us with 24,666 utterances, of which 2,000 were enciphered and the rest were left as plain text.

While this task is unsupervised, we can hope to gain information about the cipher by looking at various statistics of the plaintext and ciphertext. For instance, a very basic idea would be to match words based on their frequency. This alone doesn't work very well, but by considering bigram and trigram statistics over the latent plaintext z, we can do much better.

3. Reified Context Models

We now formally introduce reified context models. Our setup is structured prediction, where we want to predict an output $(y_1, \dots, y_L) \in \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_L$ (we abbreviate this as $y_{1:L} \in \mathcal{Y}_{1:L}$) given an input x (which we elide to simplify notation).

The central concept of our paper is that of a *context*. Suppose we want to predict y_i given $y_{1:i-1}$. Informally, a context c_{i-1} is the information that we remember about $y_{1:i-1}$. Formally, a context can be represented as a subset of the past output history: $c_{i-1} \subseteq \mathcal{Y}_{1:i-1}$. Intuitively, c_{i-1} only remembers that $y_{1:i-1} \in c_{i-1}$.

We define a *canonical context set* $\mathcal{C}_i$ to be a collection of subsets of $\mathcal{Y}_{1:i}$ satisfying two properties:[1]

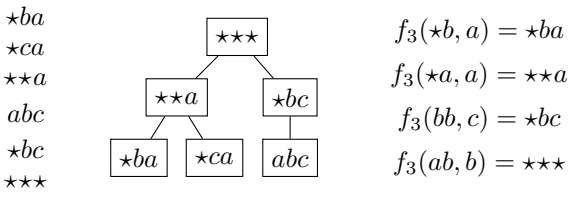
- **coverage:** $\mathcal{Y}_{1:i} \in \mathcal{C}_i$
- **closure:** for $c, c' \in \mathcal{C}_i$, $c \cap c' \in \mathcal{C}_i \cup \{\emptyset\}$

An example of such a context set is given in Figure 2; as in Section 1, notation such as $\star\star a$ denotes the subset of $\mathcal{Y}_{1:3}$ where $y_3 = a$.

These conditions allow us to operate on the context in a consistent way: After predicting y_i given c_{i-1}, we want to be able to form c_i based on c_{i-1} and y_i alone, so that if $y_{1:i-1} \in c_{i-1}$, then $y_{1:i} \in c_i$. The coverage property ensures that such a c_i always exists: we can always take $c_i = \mathcal{Y}_{1:i}$. In reality, we would like to use the smallest (most precise) context c_i possible: Given a context $c_{i-1} \in \mathcal{C}_{i-1}$ and a value $y_i \in \mathcal{Y}_i$, we define $c_i = f_i(c_{i-1}, y_i)$ to be the intersection of all $c \in \mathcal{C}_i$ that contain $c_{i-1} \times \{y_i\}$. This intersection is in $\mathcal{C}_i$ by the closure property. Example evaluations of f_i are provided in Figure 2.

[1] This is similar to the definition of a *hierarchical decomposition* from Steinhardt & Liang (2014), which used a more restrictive closure condition, namely that $c \cap c' \in \{c, c', \emptyset\}$.

Figure 2. Illustration of a context set $\mathcal{C}_3$, where each context $c \in \mathcal{C}_3$ is a subset of $\mathcal{Y}_{1:3}$; for example, $\star ba$ denotes $\mathcal{Y}_1 \times \{b\} \times \{a\}$. The contexts form a partial order (based on the subset relation). The function f_3 takes a context representing the first two letters (e.g., bb), a new letter (e.g., c), and returns a context for the first three letters (e.g., $\star bc$, which forgets the first letter).



We now define a joint model over the outputs $y_{1:L}$ and contexts $c_{1:L-1}$:

$$p_\theta(y_{1:L}, c_{1:L-1}) \propto \exp\left(\sum_{i=1}^{L} \theta^\top \phi_i(c_{i-1}, y_i)\right) \kappa(y, c),$$

where $\kappa(y, c) \stackrel{\text{def}}{=} \prod_{i=2}^{L-1} \mathbb{I}[c_i = f_i(c_{i-1}, y_i)]$ enforces consistency of the contexts. The distribution p_θ factors according to (3). One consequence of this is that the variables $y_{1:L}$ are jointly independent given $c_{1:L-1}$: the contexts capture all the dependencies between the y_i's.

Example: 2nd-order Markov chain. To provide more intuition, we construct a 2nd-order Markov chain using our framework (we can construct nth-order Markov chains in the same way). We would like $\mathcal{C}_i$ to "remember" the previous 2 values, i.e. (y_{i-1}, y_i). To do this, we let $\mathcal{C}_i$ consist of all sets of the form $\mathcal{Y}_{1:i-2} \times \{(y_{i-1}, y_i)\}$; these sets fix the value of (y_{i-1}, y_i) while allowing $y_{1:i-2}$ to vary freely. Then $f_i(c_{i-1}, y_i) = \mathcal{Y}_{1:i-2} \times \{(y_{i-1}, y_i)\}$, which is well-defined since y_{i-1} can be determined from c_{i-1}.

If $|\mathcal{Y}_i| = V$, then $|\mathcal{C}_i| = V^2$ (or V^n for nth-order chains), reflecting the true cost of inference in such models.

As a technical note, we also need to include $\mathcal{Y}_{1:i}$ in $\mathcal{C}_i$ to satisfy the coverage condition. However, $\mathcal{Y}_{1:i}$ will never actually appear as a context, as can be seen by the preceding definition of f_i.

To finish the construction, suppose we have a family of 2nd-order Markov chains parameterized as

$$p_\theta(y_{1:n}) \propto \exp\left(\sum_{i=1}^{L} \theta^\top \phi_i((y_{i-2}, y_{i-1}), y_i)\right). \quad (4)$$

Since ϕ_i depends only on (y_{i-2}, y_{i-1}), which can be determined from c_{i-1}, we can define an equivalent function

$\tilde{\phi}_i(c_{i-1}, y_i)$. Doing so, we recover a model family equivalent to (4) after marginalizing out $c_{1:L-1}$ (since $c_{1:L-1}$ is a deterministic function of $y_{1:L}$, this last step is trivial).

4. Adaptive Context Sets

The previous section showed how to define a tractable model given a collection of canonical context sets $\mathcal{C}_i$. How do we choose these context sets? Recall the intuition from Figures 1 and 2: We want to keep both coarse contexts (such as $\star\star\star$ in Figure 2) to achieve coverage of the space as well as much finer contexts (such as abc) to model complex dependencies. While finer contexts expose more information about $y_{1:i}$ and thus allow for more accurate modeling, they only cover a small part of the space. Since we must keep $\mathcal{C}_i$ small for tractability, we would like to choose contexts strategically to cover high probability regions.

We consider contexts c of the form $\mathcal{Y}_{1:i-n} \times y_{i-n+1:i}$, which are those that remember the last n outputs. We use a heuristic motivated by beam search. Beam search greedily chooses the highest-scoring configurations $y_{1:i}$ based on an estimate of their mass. We work at one higher level of abstraction, choosing *contexts* instead of *configurations*.

To assess contexts, we define the following *partial model*:

$$q^i_\theta(y_{1:i}, c_{1:i-1}) \propto \exp\left(\sum_{j=1}^{i} \theta^\top \phi_j(c_{j-1}, y_j)\right) \kappa(y, c).$$

We then select context sets of inductively as follows:

- Let $\tilde{\mathcal{C}}_i = \{c_{i-1} \times \{y_i\} \mid c_{i-1} \in \mathcal{C}_{i-1}, y_i \in \mathcal{Y}_i\}$.
- Compute the mass of each element of $\tilde{\mathcal{C}}_i$ under q^i_θ.
- Let $\mathcal{C}_i$ be the B elements of $\tilde{\mathcal{C}}_i$ with highest mass, together with the set $\mathcal{Y}_{1:i}$.

Each element of $\tilde{\mathcal{C}}_i \backslash \mathcal{C}_i$ is effectively represented by its least ancestor in $\mathcal{C}_i$. Also note that each $c \in \mathcal{C}_i$ fixes the value of some suffix $y_{j:i}$ of $y_{1:i}$, and allows $y_{1:j-1}$ to vary freely across $\mathcal{Y}_{1:j-1}$. Any such collection will automatically satisfy the closure property.

The above procedure can be performed during the forward pass of inference and is computationally cheap. Implementation details can be found in the supplementary material. We call this procedure RCMS (short for "Reified Context Model Selection").

Caveat. While we can perform exact inference over outputs $y_{1:L}$ and contexts $c_{1:L}$ given the context sets $\mathcal{C}_{1:L}$, the choice of the context sets is admittedly heuristic. While this heuristic works well empirically, an open question is to find a more principled approach.

4.1. Relationship to beam search

The idea of greedily selecting contexts based on q^i_θ is similar in spirit to beam search, an approximate inference algorithm that greedily selects individual values of $y_{1:i}$ based on q^i_θ. More formally, beam search maintains a *beam* $\mathcal{B}_i \subseteq \mathcal{Y}_{1:i}$ of size B, constructed as follows:

- Let $\tilde{\mathcal{B}}_i = \mathcal{B}_{i-1} \times \mathcal{Y}_i$.
- Compute the mass of each element of $\tilde{\mathcal{B}}_i$ under q^i_θ.
- Let $\mathcal{B}_i$ be the B elements of $\tilde{\mathcal{B}}_i$ with highest mass.

The similarity can be made precise: beam search is a degenerate instance of our procedure. Given $\mathcal{B}_i$, let $\mathcal{C}_i = \{\{b\} \mid b \in \mathcal{B}_i\} \cup \{\mathcal{Y}_{1:i}\}$. Then $\mathcal{C}_i$ consists of singleton sets for each element of $\mathcal{B}_i$, together with $\mathcal{Y}_{1:i}$ in order to ensure coverage. To get back to beam search (which doesn't have coverage), we add an additional feature to our model: $\mathbb{I}[c_i = \mathcal{Y}_{1:i}]$. We set the weight of this feature to $-\infty$, which assigns zero mass to everything outside of $\mathcal{B}_i$.

Given any algorithm based on beam search, we can improve it simply by allowing the weight on this additional feature to be learned from data. This allows us to reason about when beam search has made a search error.

4.2. Featurizations

We end this section with a recipe for choosing features $\phi_i(c_{i-1}, y_i)$. We focus on n-gram and alignment features, which are what we use in our experiments.

n-gram features. We consider nth-order Markov chains over text, typically featurized by $(n+1)$-grams:

$$\phi_i(y_{1:i-1}, y_i) = \big(\mathbb{I}[y_{i-n:i} = \hat{y}]\big)_{\hat{y} \in \mathcal{Y}_{i-n:i}}. \tag{5}$$

To extend this to our setting, define $\overline{\mathcal{Y}}_i = \mathcal{Y}_i \cup \{\star\}$ and $\overline{\mathcal{Y}}_{1:i} = \prod_{j=1}^{i} \overline{\mathcal{Y}}_j$. We can identify each pair (c_{i-1}, y_i) with a sequence $s = \sigma(c_{i-1}, y_i) \in \overline{\mathcal{Y}}_i$ in the same way as before: in each position $j \leq i$ where y_j is determined by (c_{i-1}, y_i), $s_j = y_j$; otherwise, $s_j = \star$. We then define our n-gram model on the extended space $\overline{\mathcal{Y}}_{i-n:i}$:

$$\phi_i(c_{i-1}, y_i) = \big(\mathbb{I}[\sigma(c_{i-1}, y_i) = \hat{y}]\big)_{\hat{y} \in \overline{\mathcal{Y}}_{i-n:i}}. \tag{6}$$

Alignments. In the speech task from Section 2, we have an input $x_{1:L'}$ and output $y_{1:L}$, where x and y have different lengths and need to be aligned. To capture this, we add an *alignment* z to the model, such as the one below:

aligned input	h#	y	ae	_	ax	,	s	_	w	ih	r	dcl d	h#
aligned output		y	eah	_	it		s	_	w	ei	r	d	

We represent z as a bipartite graph between $\{1, \ldots, L\}$ and $\{1, \ldots, L'\}$ with no crossing edges, and where every node

has degree at least one. The non-crossing condition allows one phoneme to align to multiple characters, or one character to align to multiple phonemes, but not many-to-many alignments. Our goal is to model $p_\theta(y, z \mid x)$.

To featurize the alignment models, we place n-gram features on the output y_i, as well as on every group of n consecutive edges. In addition, the context $c_{i',i}$ now tracks the alignment of some joint prefix $x_{1:i'}$ and $y_{1:i}$, i.e. it keeps track of the subgraph of z involving only those positions. Though this is no longer a chain structure, it is still amenable to dynamic programming.

5. Generating High Precision Predictions

Recall that one symptom stemming from a lack of coverage is poor estimates of uncertainty and the inability to generate high precision predictions. In this section, we show that the coverage offered by RCMS mitigates this issue compared to beam search.

Specifically, we are interested in whether an algorithm can find a large subset of test examples that it can classify with high ($\approx 99\%$) accuracy. Formally, assume a method outputs a prediction y with confidence $p \in [0, 1]$ for each example. We sort the examples by confidence, and see what fraction R of examples we can answer before our accuracy drops below a given threshold P. In this case, P is the *precision* and R is the *recall*.

Having good recall at high levels of precision (e.g., $P = 0.99$) is useful in applications where we need to pass on predictions below the precision threshold for a human to verify, but where we would still like to classify as many examples as possible automatically.

We ran an experiment on the handwriting recognition dataset described in Section 2. We used a 4-gram model, training both beam search (with a beam size of 10) and RCMS (with 10 contexts per position, not counting $\mathcal{Y}_{1:i}$). In addition, we used beam search with a beam size of 200 as a surrogate for exact inference. To train the models, we maximized the approximate log-likelihood using AdaGrad (Duchi et al., 2010) with a step size $\eta = 0.2$ and $\delta = 10^{-4}$.

The precision-recall curve for each method is plotted in Figure 3; confidence is the probability the model assigns to the predicted output. Note that while beam search and RCMS achieve similar accuracies (precision at $R = 1$) on the full test set (87.1% and 88.5%, respectively), RCMS is much better at separating out examples that are likely to be correct. The flat region in the precision-recall curve for beam search means that the model confidence and actual error probability are unrelated across that region.

This shows that beam search has a *precision ceiling*, where it is simply impossible to obtain high precision at any reasonable level of recall. To quantify this effect, note that the recall at 99% precision for beam search is only 16%, while for RCMS it is 82%. For comparison, the recall for exact inference is only 4% higher (86%). Therefore, RCMS is nearly as effective as exact inference on this metric while requiring substantially fewer computational resources.

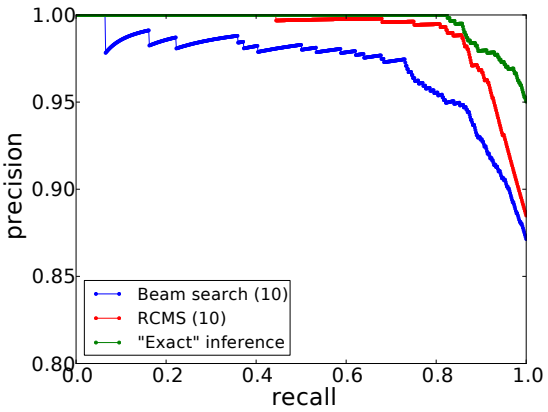


Figure 3. On handwriting recognition, precision-recall curve of beam search with beam size 10, RCMS with 10 contexts per position, and almost-exact inference simulated by beam search with a beam of size 200. Beam search makes errors even on its most confident predictions, while RCMS is able to separate out a large number of nearly error-free predictions.

6. Learning with Indirect Supervision

The second symptom of lack of coverage is the inability to learn from indirect supervision. In this setting, we have an exponential family model $p_\theta(y, z \mid x) \propto \exp(\theta^\top \phi(x, y, z))$, where x and y are observed during training but z is unobserved. The gradient of the (marginal) log-likelihood is:

$$\nabla \log p_\theta(y \mid x) = \mathbb{E}_{\hat{z} \sim p_\theta(z \mid x, y)} \left[\phi(x, y, \hat{z})\right] - \mathbb{E}_{\hat{y}, \hat{z} \sim p_\theta(y, z \mid x)} \left[\phi(x, \hat{y}, \hat{z})\right], \tag{7}$$

which is the difference between the expected features with respect to the *target* distribution $p_\theta(z \mid x, y)$ and the *model* distribution $p_\theta(y, z \mid x)$. In the fully supervised case, where we observe z, the target term is simply $\phi(x, y, z)$, which provides a clear training signal without any inference. With indirect supervision, even obtaining a training signal requires inference with respect to $p_\theta(z \mid x, y)$, which is generally intractable.

In the context of beam search, there are several strategies to inferring z for computing gradients:

- **Select-by-model:** select beams based on $q^i_\theta(z \mid x)$, then re-weight at the end by $p_\theta(y \mid z, x)$. This only works if the weights are high for at least some "easy" examples, from which learning can then bootstrap.

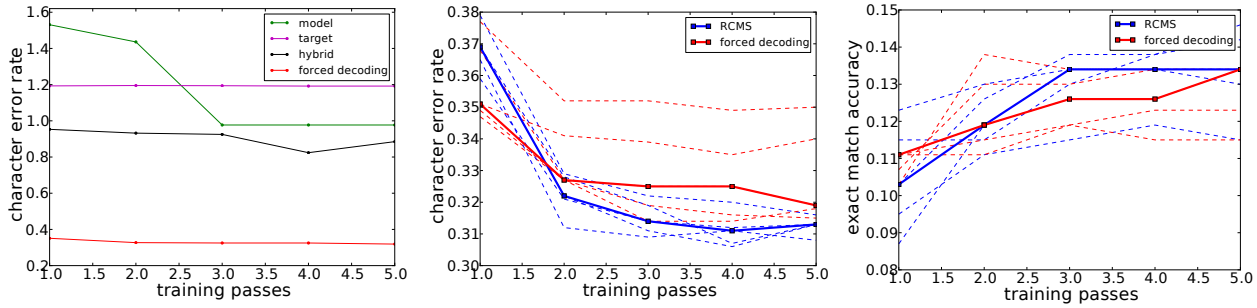


Figure 4. Left: character error rate (CER) of all beam search-based methods on the speech task, for 5 passes of the training data; note that an empty output always has a CER of 1.0. Middle: CER of forced decoding and RCMS over 5 random permutations of the data; the solid line is the median. Right: exact-match accuracy over the same 5 permutations.

- **Select-by-target:** select beams based on $q_\theta^i(z \mid x, y)$. Since y is not available at test time, parameters θ learned conditioned on y do not generalize well.
- **Hybrid:** take the union of beams based on both the model and target distributions.
- **Forced decoding (Gorman et al., 2011):** first train a simple model for which exact inference is tractable to infer the most likely z, conditioned on x and y. Then simply clamp z; this then becomes a fully-supervised problem.

To understand the behavior of these methods, we used them all to train a model on the speech recognition dataset from Section 2. The model places 5-gram indicator features on the output as well as on the alignments. We trained using AdaGrad with step size $\eta = 0.2$ and $\delta = 10^{-4}$. For each method, we set the beam size to 20. For forced decoding, we used a bigram model with exact inference to impute z.

The results are shown in Figure 4(a). Select-by-model doesn't learn at all: it only finds valid alignments for 2 out of the 746 training examples; for the rest, $p_\theta(y \mid z, x)$ is zero for all alignments considered, thus providing no signal for learning. Select-by-target quickly reaches high training accuracy, but generalizes extremely poorly because it doesn't learn to keep the right answer on the beam. The hybrid approach does better but still not very well. The only method that learns effectively is forced decoding.

While forced decoding works well, it relies on the idea that a simple model can effectively determine z given access to x and y. This will not always be the case, so we would like methods that work well even without such a model. Reified context models provide a natural way of doing this: we simply compute $p_\theta(z \mid x, y)$ under the contexts selected by RCMS, and perform learning updates in the natural way.

To test RCMS, we trained it in the same way using 20 contexts per position. Without any need for special initialization, we obtain a model whose test accuracy is better than that of forced decoding (see Figures 4(b),4(c)).

Decipherment: Unsupervised Learning. We now turn our attention to an unsupervised problem: the decipherment task from Section 2. We model decipherment as a hidden Markov model (HMM): the hidden plain text evolves according to an n-th order Markov chain, and the cipher text is emitted based on a deterministic but unknown 1:1 substitution cipher (Ravi & Knight, 2011).

Of the baseline method described above, only select-by-model can run in the absence of supervision. We therefore compare only three methods: select-by-model (beam search), RCMS, and exact inference. We trained a 1st-order (bigram) HMM using all 3 methods, and a 2nd-order (trigram) HMM using only beam search and RCMS, as exact inference was too slow (the vocabulary size is 500). We used the given plain text to learn the transition probabilities, using absolute discounting (Ney et al., 1994) for smoothing. Then, we used EM to learn the emission probabilities; we used Laplace smoothing for these updates.

The results are shown in Figure 5. We evaluated using mapping accuracy: the fraction of unique symbols that are correctly mapped (Nuhn et al., 2013). First, we compared the overall accuracy of all methods, setting the beam size and context size both to 60. We see that all 2nd-order models outperform all 1st-order models, and that beam search barely learns at all for the 1st-order model.

Restricting attention to 2nd-order models, we measure the effect of beam size and context size on accuracy, plotting learning curves for sizes of 10, 20, 30, and 60. In all cases, RCMS learns more quickly and converges to a more accurate solution than beam search. The shapes of the learning curves are also different: RCMS learns quickly after a few initial iterations, while beam search slowly accrues information at a roughly constant rate over time.

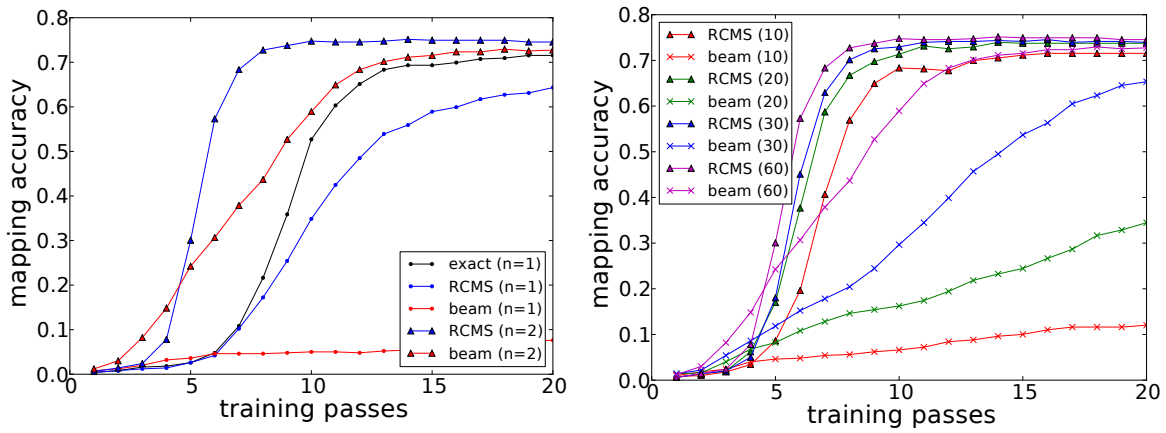


Figure 5. Results on the decipherment task. Left: accuracy for a fixed beam/context size as the model order varies; approximate inference with a 2nd-order HMM using RCMS outperforms both beam search in the same model and exact inference in a simpler model. Right: effect of beam/context size on accuracy for the 2nd-order HMM. RCMS is much more robust to changes in beam/context size.

7. Refinement of Contexts During Training

When learning with indirect supervision and approximate inference, one intuition is that we can "bootstrap" by first learning from easy examples, and then using the information gained from these examples to make better inferences about the remaining ones (Liang et al., 2011). However, this can fail if there are insufficiently many easy examples (as in the speech task), if the examples are hard to identify, or if they differ statistically from the remaining examples.

We think of the above as "vertical bootstrapping": using the full model on an increasing number of examples. RCMS instead performs "horizontal bootstrapping": for each example, it selects a model (via the context sets) based on the information available. As training progresses, we expect these contexts to become increasingly fine as the parameters improve.

To measure this quantitatively, we define the *length* of a context c_{i-1} to be the number of positions of $y_{1:i-1}$ that can be determined from c_{i-1} (number of non-$\star$'s). We plot the average length (weighted by mass under q_θ^i) as training progresses. The averages are updated every 50 and 100 training examples respectively for handwriting and speech recognition. For decipherment, they are computed once for each EM update (i.e., for each full pass over the training data).

Figure 6 shows that the broad trend is an increase in the context length over time. For both the handwriting and speech tasks, there is an initial overshoot at the beginning; this is because the handwriting and speech tasks are trained with stochastic gradient methods, which often overshoot (in contrast, for decipherment, we use the more stable EM algorithm).

Since we start by using coarse contexts and move to finer contexts by the end of training, RCMS can be thought of as a coarse-to-fine training procedure (Petrov et al., 2006). However, instead of using a sequence of pre-defined, discrete models, we organically adapt the amount of context on a per-example basis.

8. Related work

Kulesza & Pereira (2007) studied the interaction between approximate inference and learning, showing that even in the fully supervised case, approximate inference can be seriously detrimental. Finley & Joachims (2008) show that approximate inference algorithms which over-generate possible outputs interact best with learning; this further supports the need for coverage when learning.

Four major approaches have been taken to address the problem of learning with inexact inference. The first modifies the learning updates to account for the inference procedure, as in the max-violation perceptron and related algorithms (Huang et al., 2012; Zhang et al., 2013; Yu et al., 2013); a related idea is to view approximate inference as part of the model, either by reinforcement learning (Daume et al., 2009; Shi et al., 2015) or by propagating the approximations through the gradient updates (Barbu, 2009; Domke, 2011; Stoyanov et al., 2011; Steinhardt & Liang, 2015).

A second approach modifies the inference algorithm to obtain better coverage, as in coarse-to-fine inference (Petrov et al., 2006; Weiss et al., 2010), where simple models are used to direct the focus of more complex models. Pal et al. (2006) encourage coverage for beam search by adaptively increasing the beam size. A third approach is to use inference procedures with certificates of optimality, based on either duality gaps (Sontag, 2010) or variational bounds (Xing et al., 2002; Wainwright et al., 2005).

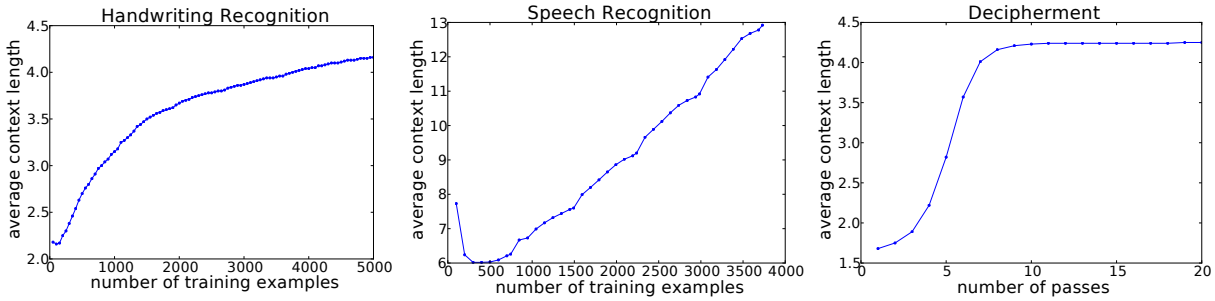


Figure 6. Average context length vs. number of learning updates for the handwriting recognition, speech recognition, and decipherment tasks. For handwriting and speech recognition we take a cumulative average (to reduce noise).

Finally, one can learn a model that is tractable. While classical tractable model families based on low treewidth are often insufficiently expressive, more modern families have shown promise; for instance, sum-product networks (Poon & Domingos, 2011), exchangeable variable models (Niepert & Domingos, 2014), and mean-field networks (Li & Zemel, 2014).

Our method RCMS also attempts to define tractable model families, in our case, via a parsimonious choice of latent context variables, even though the actual distribution over $y_{1:L}$ may have arbitrarily high treewidth. We adaptively choose the model structure for each example at "run-time", which distinguishes our approach from the aforementioned methods (though sum-product networks have some capacity for expressing adaptivity implicitly).

Certain smoothing techniques in natural language processing also interpolate between contexts of different order, such as absolute discounting (Ney et al., 1994) and Kneser-Ney smoothing (Kneser & Ney, 1995). However, in such cases *all* observed contexts are used in the model; to get the same tractability gains as we do, it would be necessary to adaptively sparsify the model for each example at runtime. Some Bayesian nonparametric approaches such as infinite contingent Bayesian networks (Milch et al., 2005) and hierarchical Pitman-Yor processes (Teh, 2006; Wood et al., 2009) also reason about contexts. Our model is also similar to the variable-length Markov chains of Bühlmann & Wyner (1999). In a different vein, some work focuses on controlling the factors present in a model, similar to our selection of contexts; this includes tightening relaxations (Riedel & Smith, 2010; Sontag et al., 2008) as well as evidence-specific MRFs (Stoyanov & Eisner, 2012).

9. Discussion

We have presented a new framework, *reified context models*, that reifies context as a random variable, thereby defining a family of expressive but tractable probability distributions. By adaptively choosing context sets per-example, our RCMS method is able to use short contexts in regions of high uncertainty and long contexts in regions of low uncertainty, thereby reproducing the behavior of coarse-to-fine training methods in a more organic and fine-grained manner. In addition, because RCMS maintains full coverage of the space, it is able to break through the precision ceiling faced by beam search. Coverage also helps with training under indirect supervision, since we can better identify settings of latent variables that assign high likelihood to the data.

At a high level, our method provides a framework for structuring inference around contexts; because the contexts are reified in the model, we can also support queries about how much mass lies in each context. These two properties together open up intriguing possibilities. For instance, we could use small context sets for each location and add finer contexts at locations where there is high uncertainty.

Another direction is to extend our construction beyond sequences. In principle, we can use any collection of contexts that induce a graphical model with low treewidth, rather than only considering the chain structure in (3). For problems such as image segmentation where the natural structure is a grid, such extensions may be necessary.

Finally, while we currently learn how much weight to assign to each context, we could go one step further and learn which contexts to propose and include in the context sets $\mathcal{C}_i$ (rather than relying on a fixed procedure as in the RCMS algorithm). Ideally, one could specify a large number of possible strategies for building context sets, and the best strategy to use for a given example would be learned from data. This would move us one step closer to being able to employ arbitrarily expressive models with the assurance of an automatic inference procedure that can reliably take advantage of this expressivity.

Reproducibility. The code, data, and the experiments for this paper are available on CodaLab at https://www.codalab.org/worksheets/0x8967960a7c644492974871ee60198401/.

Acknowledgments. The first author was supported by a Fannie & John Hertz Fellowship and an NSF Graduate Research Fellowship. The second author was supported by a Microsoft Research Faculty Fellowship. We are also grateful to the referees for their valuable comments.

References

Barbu, A. Training an active random field for real-time image denoising. *IEEE Transactions on Image Processing*, 18(11):2451–2462, 2009.

Brooks, S., Gelman, A., Jones, G., and Meng, X. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

Bühlmann, P. and Wyner, A. J. Variable length Markov chains. *Annals of Statistics*, 27(2):480–513, 1999.

Cappé, O., Godsill, S. J., and Moulines, E. An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

Daume, H., Langford, J., and Marcu, D. Search-based structured prediction. *Machine Learning*, 75:297–325, 2009.

Domke, J. Parameter learning with truncated message-passing. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 2937–2943, 2011.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*, 2010.

Finley, T. and Joachims, T. Training structural SVMs when exact inference is intractable. In *International Conference on Machine Learning (ICML)*, pp. 304–311, 2008.

Gorman, K., Howell, J., and Wagner, M. Prosodylab-aligner: A tool for forced alignment of laboratory speech. *Canadian Acoustics*, 39(3):192–193, 2011.

Graff, D. and Cieri, C. *English Gigaword LDC2003T05*, 2003.

Greenberg, S., Hollenback, J., and Ellis, D. Insights into spoken language gleaned from phonetic transcription of the Switchboard corpus. In *International Conference on Spoken Language Processing (ICSLP)*, 1996.

Huang, L., Fayong, S., and Guo, Y. Structured Perceptron with inexact search. In *Association for Computational Linguistics (ACL)*, pp. 142–151, 2012.

Kassel, R. H. *A comparison of approaches to on-line handwritten character recognition*. PhD thesis, Massachusetts Institute of Technology, 1995.

Kneser, R. and Ney, H. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pp. 181–184, 1995.

Koehn, P., Och, F. J., and Marcu, D. Statistical phrase-based translation. In *Association for Computational Linguistics (ACL)*, pp. 48–54, 2003.

Kulesza, A. and Pereira, F. Structured learning with approximate inference. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 785–792, 2007.

Li, Y. and Zemel, R. Mean-field networks. *arXiv preprint arXiv:1410.5884*, 2014.

Liang, P., Jordan, M. I., and Klein, D. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pp. 590–599, 2011.

Milch, B., Marthi, B., Sontag, D., Russell, S., Ong, D. L., and Kolobov, A. Approximate inference for infinite contingent Bayesian networks. In *Artificial Intelligence and Statistics (AISTATS)*, pp. 238–245, 2005.

Ney, H., Essen, U., and Kneser, R. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language*, 8(1):1–38, 1994.

Niepert, M. and Domingos, P. Exchangeable variable models. In *International Conference on Machine Learning (ICML)*, 2014.

Nuhn, M. and Ney, H. Improved decipherment of homophonic ciphers. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Nuhn, M., Schamper, J., and Ney, H. Beam search for solving substitution ciphers. In *Association for Computational Linguistics (ACL)*, pp. 1569–1576, 2013.

Pal, C., Sutton, C., and McCallum, A. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, 2006.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. Learning accurate, compact, and interpretable tree annotation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*, pp. 433–440, 2006.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 337–346, 2011.

Ravi, S. and Knight, K. Deciphering foreign language. In *Association for Computational Linguistics (ACL)*, pp. 12–21, 2011.

Recht, B., Ré, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 693–701, 2011.

Riedel, S. and Smith, D. A. Relaxed marginal inference and its application to dependency parsing. In *Association for Computational Linguistics (ACL)*, pp. 760–768, 2010.

Shi, T., Steinhardt, J., and Liang, P. Learning where to sample in structured prediction. In *Artificial Intelligence and Statistics (AISTATS)*, pp. 875–884, 2015.

Sontag, D. *Approximate inference in graphical models using LP relaxations*. PhD thesis, Massachusetts Institute of Technology, 2010.

Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., and Jaakkola, T. Tightening LP relaxations for MAP using message-passing. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 503–510, 2008.

Steinhardt, J. and Liang, P. Filtering with abstract particles. In *International Conference on Machine Learning (ICML)*, pp. 727–735, 2014.

Steinhardt, J. and Liang, P. Learning fast-mixing models for structured prediction. In *International Conference on Machine Learning (ICML)*, 2015.

Stoyanov, V. and Eisner, J. Fast and accurate prediction via evidence-specific MRF structure. In *ICML Workshop on Inferning: Interactions between Inference and Learning*, 2012.

Stoyanov, V., Ropson, A., and Eisner, J. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Artificial Intelligence and Statistics (AISTATS)*, pp. 725–733, 2011.

Teh, Y. W. A hierarchical Bayesian language model based on Pitman-Yor processes. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*, pp. 985–992, 2006.

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.

Weiss, D. and Taskar, B. Structured prediction cascades. In *Artificial Intelligence and Statistics (AISTATS)*, 2010.

Weiss, D., Sapp, B., and Taskar, B. Sidestepping intractable inference with structured ensemble cascades. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2415–2423, 2010.

Wood, F., Archambeau, C., Gasthaus, J., James, L., and Teh, Y. W. A stochastic memoizer for sequence data. In *International Conference on Machine Learning (ICML)*, pp. 1129–1136, 2009.

Xing, E. P., Jordan, M. I., and Russell, S. A generalized mean field algorithm for variational inference in exponential families. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 583–591, 2002.

Yu, H., Huang, L., Mi, H., and Zhao, K. Max-violation Perceptron and forced decoding for scalable MT training. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1112–1123, 2013.

Zhang, H., Huang, L., Zhao, K., and McDonald, R. Online learning for inexact hypergraph search. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Zhang, L., Kalashnikov, D. V., and Mehrotra, S. Context-assisted face clustering framework with human-in-the-loop. *International Journal of Multimedia Information Retrieval*, 3(2):69–88, 2014.

A. Implementation Details

Recall that to implement the RCMS method, we need to perform the following steps:

1. Let $\tilde{\mathcal{C}}_i = \{c_{i-1} \times \{y_i\} \mid c_{i-1} \in \mathcal{C}_{i-1}, y_i \in \mathcal{Y}_i\}$.
2. Compute what the mass of each element of $\tilde{\mathcal{C}}_i$ would be if we used q_θ^i as the model and $\tilde{\mathcal{C}}_i$ as the collection of contexts.
3. Let $\mathcal{C}_i$ be the B elements of $\tilde{\mathcal{C}}_i$ with highest mass, together with the set $\mathcal{Y}_{1:i}$.

As in Section 4, each context in $\mathcal{C}_i$ can be represented by a string $s_{1:i}$, where $s_j \in \mathcal{Y}_j \cup \{\star\}$. We will also assume an arbitrary ordering on $\mathcal{Y}_j \cup \{\star\}$ that has $\star$ as its maximum element.

In addition, we use two datatypes: `E` (for "expand"), which keeps track of elements of $\tilde{\mathcal{C}}_i$, and `M` (for "merge"), which keeps track of elements of $\mathcal{C}_i$. More precisely, if c_{i-1} is represented by an object m_{i-1} of type `M`, then $\texttt{E}(m_{i-1}, y_i)$ represents $c_{i-1} \times \{y_i\}$; and $\texttt{M}(\texttt{E}(m_{i-1}, y_i))$ represents $c_{i-1} \times \{y_i\}$ as well, with the distinction that it is a member of $\mathcal{C}_i$ rather than $\tilde{\mathcal{C}}_i$. The distinction is important because we will also want to merge smaller contexts into objects of type `M`. For both `E` and `M` objects, we maintain a field len, which is the length of the suffix of $y_{1:i}$ that is specified (e.g., if an object represents $\mathcal{Y}_{1:3} \times \{y_{4:5}\}$, then its len is 2).

Throughout our algorithm, we will maintain 2 invariants:

- $\tilde{\mathcal{C}}_i$ and $\mathcal{C}_i$ will be sorted lexicographically (e.g. based first on s_i, then s_{i-1}, etc.)
- A list $\widetilde{\text{lcs}}_i$ of length $\text{len}(\tilde{\mathcal{C}}_i)$ is maintained, such that the longest common suffix of $\tilde{\mathcal{C}}_i[a]$ and $\tilde{\mathcal{C}}_i[b]$ is $\min_{c \in [a,b)} \widetilde{\text{lcs}}_i[c]$. A similar list lcs_i is maintained for $\mathcal{C}_i$.

Step 1. To perform step 1 above, we just do:

$\tilde{\mathcal{C}}_i = []$
for $j = 0$ **to** $\text{len}(\mathcal{Y}_i) - 1$ **do**
  for $k = 0$ **to** $\text{len}(\mathcal{C}_{i-1}) - 1$ **do**
    if $k + 1 < \text{len}(\mathcal{C}_{i-1})$ **then**
      $\widetilde{\text{lcs}}_i$.append($\text{lcs}_{i-1}[k] + 1$)
    else
      $\widetilde{\text{lcs}}_i$.append(0)
    end if
    $\tilde{\mathcal{C}}_i$.append($\texttt{E}(\mathcal{C}_i[k], \mathcal{Y}_i[j])$)
  end for
end for

The important observation is that if two sequences end in the same character, their lcs is one greater than the lcs of the remaining sequence without that character; and if they end in different characters, their lcs is 0.

Each `E` keeps track of a forward score, defined as

$$\texttt{E}(m, y).\text{forward} = m.\text{forward} \times \exp(\theta^\top \phi(m, y)). \quad (8)$$

Step 2. For step 2, we find the B elements $\tilde{c}$ of $\tilde{\mathcal{C}}_i$ with the largest forward score; we set a flag $\tilde{c}$.active to true for each such $\tilde{c}$.

Step 3. Step 3 contains the main algorithm challenge, which is to efficiently merge each element of $\tilde{\mathcal{C}}_i$ into its least ancestor in $\mathcal{C}_i$. If we think of $\mathcal{C}_i$ as a tree (as in Figure 2), we can do this by essentially performing a depth-first-search of the tree. The DFS goes backwards in the lexicographic ordering, so we need to reverse the lists $\mathcal{C}_i$ and lcs_i at the end.

▷ merge and update lcs
stack = []
$\mathcal{C}_i = []$
$\text{lcs}_i = []$
$l \leftarrow \infty$
for $j = \text{len}(\tilde{\mathcal{C}}_i) - 1$ **to** 0 **do**
  $l \leftarrow \min(l, \widetilde{\text{lcs}}_i[j])$
  while $l < \text{stack}[-1].\text{len}$ **do**
    ▷ then current top of stack is not an ancestor of $\tilde{\mathcal{C}}_i[j]$
    stack.pop()
  end while
  if $\tilde{\mathcal{C}}_i[j]$.active **then**
    $m = \texttt{M}(\tilde{\mathcal{C}}_i[j])$
    lcs_i.append(l)
    $\mathcal{C}_i$.append(m)
    stack.push(m)
    $l \leftarrow \infty$
  else
    ▷ merge $\tilde{\mathcal{C}}_i[j]$ into its least ancestor
    stack$[-1]$.absorb($\tilde{\mathcal{C}}_i[j]$)
  end if
end for
lcs_i.reverse()
$\mathcal{C}_i$.reverse()

If $m \in \mathcal{C}_i$ has absorbed elements $e_1, \ldots, e_k$, then we compute m.forward as $\sum_{j=1}^k e_j$.forward.

After we have constructed $\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}$, we also need to send backward messages for inference. If $e \in \tilde{\mathcal{C}}_i$ is merged into $m \in \mathcal{C}_i$, then $e.\text{backward} = m.\text{backward}$. If $m \in \mathcal{C}_i$ expands to $\texttt{E}(m, y)$ for $y \in \mathcal{Y}_{i+1}$, then $m.\text{backward} = \sum_{y \in \mathcal{Y}_{i+1}} \texttt{E}(m, y).\text{backward} \times \exp(\theta^\top \phi(m, y))$. The (unnormalized) probability mass of an object is then simply the product of its forward and backward scores; we can compute the normalization constant by summing over $\mathcal{C}_i$.

In summary, our method can be coded in three steps; first, during the forward pass of inference, we:

1. Expand to $\tilde{\mathcal{C}}_i$ and construct $\widetilde{\text{lcs}}_i$.
2. Sort by forward score and mark active nodes in $\tilde{\mathcal{C}}_i$ for inclusion in $\mathcal{C}_i$.
3. Merge each node in $\tilde{\mathcal{C}}_i$ into its least ancestor in $\mathcal{C}_i$, using a depth-first-search.

Finally, once all of the $\mathcal{C}_i$ are constructed, we perform the backward pass:

4. Propagate backward messages and compute the normalization constant.

B. Further Details of Experimental Setup

We include here a few experimental details that did not fit into the main text. When training with AdaGrad, we performed several stochastic gradient updates in parallel, similar to the approach described in Recht et al. (2011) (although we parallelized even more aggressively at the expense of theoretical guarantees). We also used a randomized truncation scheme to round most small coordinates of the gradients to zero, which substantially reduces memory usage as well as concurrency overhead.

For decipherment, we used absolute discounting with discount 0.25 and smoothing 0.01, and Laplace smoothing with parameter 0.01. For the 1st-order model, beam search performs better if we use Laplace smoothing instead of absolute discounting (though still worse than RCMS). In order to maintain a uniform experimental setup, we excluded this result from the main text.

For the hybrid selection algorithm in the speech experiments, we take the union of the beams at every step (as opposed to computing two sets of beams separately and then taking a single union at the end).

C. Runtime

Asymptotically, both RCMS and beam search have the same runtime: given a beam size b and alphabet size s, beam search and RCMS both run in time $\mathcal{O}(bs \log(bs))$ at each position y_i, and require $\mathcal{O}(bs)$ feature extractions. However, in general the constant factor for RCMS will be larger, and in this section we characterize this more precisely.

First, we make the caveat that the relative speed of RCMS versus beam search is sensitive to many factors, including the time required for feature extraction and model evaluation, and how much one is willing to trade off runtime for memory.

With this caveat in mind, we obtained a rough runtime comparison by recording the runtime of both RCMS and beam search on the decipherment task, and attempted to optimize both methods to about the same degree. In this case, RCMS was approximately 2 to 3 times slower than beam search.

The main factors causing RCMS to be slower are the more complicated merging step (which requires the LCS data structure described in Appendix A) as well as the backward pass, which requires recomputing all of the features if they were not cached. Also, the gradient updates are denser than beam search, by a factor of L (the length of y).

In cases where feature extraction is the dominant cost, and assuming that features are cached for the backwards pass, the runtimes of RCMS and beam search should be similar.

D. Additional Files

In the supplementary material, we also include the source code and datasets for the decipherment task. A `README` is included to explain how to run these experiments.

In addition, a fully runnable version of all the experiments (including data, source code, dependencies, and run commands) can be found on CodaLab, at https://www.codalab.org/worksheets/0x8967960a7c644492974871ee60198401/.