Forward-Backward with Failure Arcs: Faster Inference for Variable-Order Conditional Random Fields

Tim Vieira* and **Ryan Cotterell*** and **Jason Eisner**
Johns Hopkins University
{timv,ryan.cotterell,jason}@cs.jhu.edu

Abstract

Variable-order conditional random fields (VoCRFs) offer an elegant manner to compromise between expressive higher-order CRFs and speedy lower-order CRFs—the model utilizes only those bits of higher-order structure that actually help performance, leaving out the rest. This approach yields a model with faster inference, avoiding the exponential runtime, when all contexts are considered. In this work, we introduce an asymptotically faster algorithm using failure arcs, a trick from the finite-state literature, for inference on variable-order CRFs and show empirically faster runtimes. We experiment on multi-lingual part-of-speech tagging.

1 Introduction

Linear-chain conditional random fields (CRFs) (Lafferty et al., 2001) have proven themselves a successful formalism for tagging tasks common in natural language processing (NLP). A CRF defines a probabilistic model of an output sequence conditioned on an input sequence. CRFs have retained their popularity over the years in large part due to the ability to consider arbitrary features of the input. In terms of output structure, however, inference in these models remains efficient only if the set of tag combinations is small. An order-k CRF (abbreviated k-CRF) is an extension to the original model that allows for higher-order tag interactions. Setting $k = 1$ yields the original model of Lafferty et al. (2001) and $k > 1$ yields the higher-order models of Sha and Pereira (2003). While more expressive models are often more accurate (Müller et al., 2013), inference in such models comes at an increased runtime—namely, it is exponential in the order k. One way around this increased runtime is to allow a (variable-order) subset of all order-k tag interactions; such a model is known as a variable-order CRF (VoCRF) (Ye et al., 2009). Crucially, it incorporates only those important higher-order interactions, while excluding the gratuitous ones.

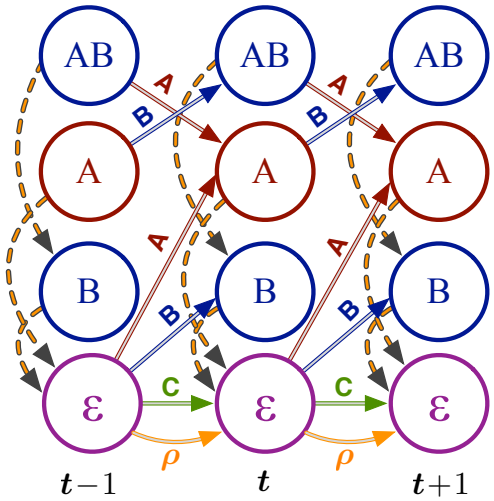

Figure 1: Here we depict a sample VoCRF lattice at three steps t–1, t and t+1 in medias res. The tag set is $Y = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}\}$ and $\mathcal{W} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{AB}, \texttt{ABA}\}$. Thus, $\mathcal{H} = \{\varepsilon, \texttt{A}, \texttt{B}, \texttt{AB}\}$. The ϕ-arcs are the only dotted arcs in the figure and show the backoff. We show the ρ-arc, which accepts the tags D and E that are not in $\mathcal{W}$.

In this work, we develop an asymptotically faster inference algorithm for VoCRFs. Indeed, given a tag set Y, our algorithm runs up to $|Y|$ times faster. For example, consider part-of-speech (POS) tagging on the Penn Treebank, where $|Y| = 36$. Our method draws upon failure arcs, a technique from the finite-state literature used to efficiently encode backoff n-gram language models as weighted finite-state machines; to the best of our knowledge, however, the use of failure arcs has not migrated to structured prediction. We review the relevant background material and provide a succinct exposition (with pseudocode) of our algorithm. Finally, we conduct a series of experiments that shows the practicality of our algorithm with empirical gains on multi-lingual POS tagging on the Universal Dependencies treebanks (Nivre et al., 2015).

2 *Faster* Variable-Order CRFs

In this section, we provide background on higher-order CRFs, motivating their refinement into

*Equal contribution

VoCRFs, and develop a faster inference algorithm by encoding VoCRFs compactly as weighted finite state machines (WFSAs) with failure arcs.

2.1 Higher-Order CRFs

A k-CRF is a conditional distribution of an output sequence given an input sequence:

$$p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}(\boldsymbol{x})} \exp\left(\sum_{t=1}^{n} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{w}_t)\right) \quad (1)$$

where $n = |\boldsymbol{x}| = |\boldsymbol{y}|$ is the input and output length, $\boldsymbol{w}_t = y_{t-k} \cdots y_t$ (or $\boldsymbol{w}_t = y_1 \cdots y_t$ if $t \leq k$), $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{w}_t) \in \mathbb{R}^d$ is an arbitrary user-defined feature vector (possibly computed by some neural network), and $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter vector.

Because each feature vector $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{w}_t)$ considers at most a length-$(k+1)$ substring of $\boldsymbol{y}$, inference over the set of $|Y|^n$ possible $\boldsymbol{y}$ values can be performed in time only $\mathcal{O}\left(n \cdot |Y|^{k+1}\right)$. Even so, it is wasteful to consider such an expansive set of substrings, as most $(k+1)$-grams are not idiosyncratic enough to require their own feature weights.

2.2 Variable-Order CRFs

To speed up the k-CRF, an important refinement is to consider variable-length substrings. A VoCRF (Ye et al., 2009; Nguyen et al., 2014) specifies a finite set $\mathcal{W} \subset Y^*$ of output substrings, and redefines $\boldsymbol{w}_t$ to be the longest suffix of $y_1 \cdots y_t$ that appears in $\mathcal{W}$.[1] Typically, $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{w}_t)$ will extract features from both $\boldsymbol{w}_t$ and its shorter suffixes. There is always some k large enough that $\mathcal{W} \subseteq Y^{k+1}$, but if $|\mathcal{W}| \ll |Y|^{k+1}$ we would like inference to be proportionately faster than in the k-CRF. This will let us trade accuracy for speed by reducing $\mathcal{W}$.

2.3 Finite-State Acceptors and Failure Arcs

Weighted finite-state acceptors (WFSAs) are ubiquitous in NLP (Mohri, 1997) and can be used to implement VoCRF models. Given $\mathcal{W}$ and an input $\boldsymbol{x}$, we can construct a deterministic WFSA (Fig. 1) whose accepting paths correspond to the possible outputs $\boldsymbol{y} \in Y^n$. Given also $\boldsymbol{f}$ and $\boldsymbol{\theta}$, the arcs of this DFA can be weighted so that the relative probabilities of the $\boldsymbol{y}$ sequences, from Eq. (1), are given by the weights of the corresponding paths (where a path's weight is the product of its arc weights). The normalizing constant $Z_{\boldsymbol{\theta}}(\boldsymbol{x})$ in Eq. (1) is the total weight of all paths (Mohri, 2002).

As Fig. 1 illustrates, an economical construction employs failure arcs. A failure arc (ϕ-arc) consumes no input and may be taken only if no ordinary arc is available (Allauzen et al., 2003). It serves as a default option that backs off to a simpler context state. While ϕ-arcs have mainly been used to compactly encode backoff language models, here we apply them to structured prediction.

We simplify our construction by assuming that $\mathcal{W}$ is closed under prefixes (as ensured by §3's algorithm for learning $\mathcal{W}$). Each nonempty string $\boldsymbol{w} \in \mathcal{W}$ can be written as $\boldsymbol{h}y$, where $\boldsymbol{h} \in Y^*$ is a "history" and $y \in Y$ is the next tag; let $\mathcal{H}$ denote the set of all histories. The WFSA states are $\{0, \ldots, n\} \times \mathcal{H}$, with $(0, \varepsilon)$ being the initial state and $\{n\} \times \mathcal{H}$ being the final states. Each nonempty[2] $\boldsymbol{w} = \boldsymbol{h}y \in \mathcal{W}$ generates an arc $(t-1, \boldsymbol{h}) \xrightarrow{y/e} (t, \boldsymbol{h}')$ for each $1 \leq t \leq n$, where $\boldsymbol{h}' \in \mathcal{H}$ is the longest history that is a proper suffix of $\boldsymbol{w}$. This arc will be traversed only if $\boldsymbol{w}_t = \boldsymbol{w}$, so we set the arc weight $e = \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{w}))$. To handle all actions $y' \in Y$ for which $\boldsymbol{h}y' \notin \mathcal{W}$, each state also has an outgoing failure arc $(t-1, \boldsymbol{h}) \xrightarrow{\phi/1} (t-1, \tilde{\boldsymbol{h}})$, where $\tilde{\boldsymbol{h}} \in \mathcal{H}$ is the longest history that is a proper suffix of $\boldsymbol{h}$.

This compact WFSA has $(n+1)(|\mathcal{W}| + |\mathcal{H}|) = \mathcal{O}(n \cdot |\mathcal{W}|)$ arcs. Without failure arcs, every state would have needed separate outgoing arcs for all $y \in Y$, for $(n+1)(|\mathcal{H}||Y|)$ arcs. Failure arcs therefore make us up to $|Y|/2$ times smaller.[3] For example, we improve over the prior work of Vieira et al. (2016), which learned a VoCRF without failure arcs, and which itself provided an $O(|\mathcal{H}|)$ speedup over (Ye et al., 2009; Nguyen et al., 2014).

2.4 Improved Inference in VoCRFs

Our modification of the forward-backward method is needed to run in time proportional to the number of arcs. The trick is to use *subtraction* to correct for double-counted paths (similarly to Roark et al. (2013, §4.1)).[4] This beats the naive algorithm that regards a failure arc as an abbreviation for up to $|Y|$

[1] To ensure this exists, we require either $\varepsilon \in \mathcal{W}$ or $Y \subseteq \mathcal{W}$.

[2] For the case $\boldsymbol{w} = \varepsilon$, we take $\boldsymbol{h} = \varepsilon$ (we include ε in $\mathcal{H}$) and $y = \rho$. This generates an arc $(t-1, \varepsilon) \xrightarrow{\rho/e} (t, \varepsilon)$. The special symbol ρ is similar to ϕ in that it matches "any other" symbol, but unlike ϕ it consumes that symbol. The ε state cannot back off further, so it has this ρ arc *instead of* a ϕ arc.

[3] In the case where $|\mathcal{W}| \approx |\mathcal{H}|$, i.e., most histories can be extended in only one way. For example, $\mathcal{W}$ might contain only a single long string and its prefixes. In general, failure arcs help at states that can only be extended in $\ll |Y|$ ways.

[4] Note that not all operations of interest efficiently lend themselves to this "subtraction", particularly Viterbi decoding. Viterbi would require an implementation of $\max$ that can delete subsets of its aggregands. This is possible with an additional $\log |Y|$ factor runtime, giving $\mathcal{O}(n \cdot |\mathcal{W}| \cdot \log |Y|)$, by using a Fenwick tree data structure (Fenwick, 1994).

Algorithm 1 Compute $\log Z_{\boldsymbol{\theta}}(\boldsymbol{x})$ where $|\boldsymbol{x}| = n$.

1: $\beta(\cdot, \cdot) = \mathbf{0}$; $\beta(n, \boldsymbol{h}) = 1$ **for** $\boldsymbol{h} \in \mathcal{H}$
2: **for** $t = n$ **down to** 1 :
3: **for** $\boldsymbol{h} \in \mathcal{H}$ from shortest to longest :
4: **for** $(t-1, \boldsymbol{h})$'s failure arc $\xrightarrow{\phi/1}$ $(t-1, \tilde{\boldsymbol{h}})$:
5: $\beta(t-1, \boldsymbol{h})$ += $\beta(t-1, \tilde{\boldsymbol{h}})$
6: **for** $(t-1, \boldsymbol{h})$'s outgoing arcs $\xrightarrow{y/e}$ $(t, \boldsymbol{h}')$:
7: $\beta(t-1, \boldsymbol{h})$ += $e \cdot \beta(t, \boldsymbol{h}')$ ▷ see footnote 4
8: $\tilde{\boldsymbol{h}}$ = longest proper suffix of $\boldsymbol{h}$ with $\tilde{\boldsymbol{h}}y \in \mathcal{W}$
9: **for** $(t-1, \tilde{\boldsymbol{h}})$'s outgoing arc $\xrightarrow{a/\tilde{e}}$ $(t, \tilde{\boldsymbol{h}}')$:
10: $\beta(t-1, \boldsymbol{h})$ -= $\tilde{e} \cdot \beta(t, \tilde{\boldsymbol{h}}')$
11: $Z = \beta(0, \varepsilon)$
12: **return** $\log Z$

Algorithm 2 Compute $\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}}(\boldsymbol{x})$.

1: $\boldsymbol{\Delta} = \mathbf{0}$; $\alpha(\cdot, \cdot) = \mathbf{0}$; $\alpha(0, \varepsilon) = 1/Z$
2: **for** $t = 1$ **to** n :
3: **for** $\boldsymbol{h} \in \mathcal{H}$ from longest to shortest :
4: **for** $(t-1, \boldsymbol{h})$'s failure arc $\xrightarrow{\phi/1}$ $(t-1, \tilde{\boldsymbol{h}})$:
5: $\alpha(t-1, \tilde{\boldsymbol{h}})$ += $\alpha(t-1, \boldsymbol{h})$
6: **for** $(t-1, \boldsymbol{h})$'s outgoing arcs $\xrightarrow{y/e}$ $(t, \boldsymbol{h}')$:
7: $\alpha(t, \boldsymbol{h}')$ += $e \cdot \alpha(t-1, \boldsymbol{h})$ ▷ see footnote 4
8: $\boldsymbol{\Delta}$ += $\alpha(t-1, \boldsymbol{h}) \cdot e \cdot \beta(t, \boldsymbol{h}') \cdot \boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{h}a)$
9: $\tilde{\boldsymbol{h}}$ = longest proper suffix of $\boldsymbol{h}$ with $\tilde{\boldsymbol{h}}y \in \mathcal{W}$
10: **for** $(t-1, \tilde{\boldsymbol{h}})$'s outgoing arc $\xrightarrow{a/\tilde{e}}$ $(t, \tilde{\boldsymbol{h}}')$:
11: $\alpha(t, \tilde{\boldsymbol{h}}')$ -= $\alpha(t-1, \boldsymbol{h}) \cdot \tilde{e}$
12: $\boldsymbol{\Delta}$ -= $\alpha(t-1, \boldsymbol{h}) \cdot \tilde{e} \cdot \beta(t, \tilde{\boldsymbol{h}}') \cdot \boldsymbol{f}(\boldsymbol{x}, t, \tilde{\boldsymbol{h}}a)$
13: **return** $\boldsymbol{\Delta}$

ordinary arcs that must be processed individually (thus saving space but not time). $|Y|$ is non-trivial for many applications in NLP, e.g., Penn Treebank tagging considers $|Y| = 36$ and Czech morphological tagging considers $|Y| > 1000$ (Hajič, 1998). Pseudocode that is more specialized to our VoCRF WFSA architecture is given as Algs. 1–2. Note that in practice, there is no need to build an explicit WFSA. A loop like line 6 of Alg. 1 is implemented simply by iterating through the $(y, e, \boldsymbol{h}')$ triples that *would* correspond to the outgoing arcs from $(t-1, \boldsymbol{h})$. These can be computed from $\mathcal{W}$ (and $\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{\theta}$) as described in the previous section.

Specifically, Alg. 1 is a backward algorithm to sum over all paths in the WFSA. We algorithmically differentiate this code (Griewank et al., 1989) to obtain the additional ("forward") code in Alg. 2. As Eisner (2016) explains, this is a generic way to derive forward-backward algorithms. The returned gradient vector $\boldsymbol{\Delta}$ is used for optimization in §3, but it also equals the vector of expected feature counts (used for MBR decoding in §4) that is traditionally presented as forward-backward's output.

3 Learning $|\mathcal{W}|$ from Data

A key concern in applying VoCRFs is learning a set $\mathcal{W}$ that result in a fast *and* accurate model. To do this, we minimize the following learning objective:

$$-\sum_{i=1}^{m} \underbrace{\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)})}_{\text{loss}} + \underbrace{\lambda ||\boldsymbol{\theta}||_2^2}_{\text{generalization}} + \gamma \underbrace{\mathcal{R}(\boldsymbol{\theta})}_{\text{runtime}}. \quad (2)$$

[4] In the special case $y = \rho$, the edge weight e must be multiplied by the number of tags in Y that will use the ρ-arc.

The first term is the loss, which we take as the log-likelihood of the training data $\{(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{m}$. The later terms encourage generalization (L_2 regularization) and a small runtime; each has a coefficient that controls its relative importance.

Since, our model parameterization assigns a unique weight $\boldsymbol{\theta}_{\boldsymbol{w}}$ to each string $\boldsymbol{w} \in Y^*$, $\mathcal{R}$ would ideally compute $|\mathcal{W}|$ by looking at which features have a nonzero weight in $\boldsymbol{\theta}$ taking account the fact that $\mathcal{W}$ must be prefixed closed. Unfortunately, minimization with this version of $\mathcal{R}$ is NP-hard. Thus, we use a convex relaxation known as **tree-structured group lasso** (Yuan and Lin, 2006; Nelakanti et al., 2013), which was shown effective for learning $\mathcal{W}$ for VoCRFs by Vieira et al. (2016). We define a group $G_{\boldsymbol{w}}$ for each $\boldsymbol{w}$, which consists of the indices of indicator features of all strings in $\mathcal{W}$ that have $\boldsymbol{w}$ as a prefix. We then define $\mathcal{R}(\boldsymbol{\theta}) = \sum_{\boldsymbol{w} \in Y^*} ||\boldsymbol{\theta}_{G_{\boldsymbol{w}}}||_2$, where $\boldsymbol{\theta}_{G_{\boldsymbol{w}}}$ is the subvector for the group $G_{\boldsymbol{w}}$. This group structure differs from Vieira et al. (2016) because they targeted $|\mathcal{H}|$ instead of $|\mathcal{W}|$, which is apt when $\mathcal{W}$ factorizes as $\mathcal{H} \times Y$. However, with failure arcs, this factorization is no longer necessary.

Optimization. Despite its convexity, optimizing Eq. (2) still constitutes a challenge as there are an infinite number of potential prefixes. Thus, we define an **active set** (Schmidt, 2010) of features that are allowed to be nonzero. To optimize the objective with respect to the parameters $\boldsymbol{\theta}$, we perform the following inductive procedure. Let $\mathcal{W}^{(k)}$ denote the active set at the k^{th} epoch. We define $\mathcal{W}^{(0)} = Y$. With a given $\mathcal{W}^{(k)}$, we build a VoCRF and perform a pass over the training data using

runtime	Basque		Bulgarian		Hindi		Norwegian		Slovenian	
$\lvert\mathcal{W}\rvert \leq \tau$	without-ϕ	with-ϕ	without-ϕ	with-ϕ	without-ϕ	with-ϕ	without-ϕ	with-ϕ	without-ϕ	with-ϕ
256	93.43	93.49	97.34	97.51	96.13	96.29	96.69	96.82	95.43	95.78
512	93.43	93.49	97.45	97.51	96.18	96.29	96.72	96.85	95.62	95.78
1024	93.43	93.49	97.45	97.51	96.22	96.29	96.76	97.03	95.78	95.85
2048	93.43	93.53	97.46	97.65	96.29	96.29	96.76	97.05	95.78	95.85
4096	93.48	93.55	97.48	97.65	96.29	96.31	96.89	97.08	95.78	95.93
8192	93.48	93.55	97.50	97.71	96.29	96.34	96.94	97.15	95.78	96.00

Table 1: Test set results. POS tagging in 5 languages. See §4 for experimental setup. Each row represents a different runtime determined by thresholding $\mathcal{W} \leq \tau$. For each language, a row compares a system *with* failure arcs and *without*. The failure arc version is *always* more accurate for the given runtime budget. This increase in accuracy is do the fact that with failure arcs we can select features that cover a wider output context, without incurring the cost last symbol closure.

the proximal gradient algorithm SPOM (Martins et al., 2011a) with learning rates determined by ADAGRAD (Duchi et al., 2011) with the proximal update applied every 25 steps for efficiency. At the end of this epoch, define $\mathcal{W}^{(k+1)} = \{\boldsymbol{w}y \mid \boldsymbol{w} \in \mathcal{W}^{(k)}, y \in Y \,\&\, \boldsymbol{\theta}_{\boldsymbol{w}} \neq 0\}$. We note that this procedure differs from that in Vieira et al. (2016), in that they require an additional closure step in order to run inference. At convergence, we set $\mathcal{W} = \{\boldsymbol{w} \mid \boldsymbol{w} \in W^{(final)} \,\&\, \boldsymbol{\theta}_{\boldsymbol{w}} \neq 0\}$ for decoding.

We make use of the budget-driven shrinkage heuristic Martins et al. (2011b). Rather than running optimization with a *fixed* parameter γ, it is instead determined in a *time-varying* manner: γ_t is the value that would ensure $|\mathcal{W}^{(k)}| \leq B$ after applying the proximal operator. We then use γ_t in the proximal update.[5] Here B is a *user-friendly* budget parameter. Furthermore, this trick guarantees that $|\mathcal{W}^{(k)}|$ remains $\leq B \cdot |Y|$ (after every proximal update). This is important because a bad setting for γ values will cause the active set to grow, increasing the runtime of computing gradients.

4 Experiments and Results[6]

Experimental Setup: We closely replicate the experimental setup in prior work by Vieira et al. (2016). We conduct experiments on POS tagging (Nivre et al., 2015) in five languages: Basque, Bulgarian, Hindi, Norwegian and Slovenian. For all languages $|Y| = 17$. We use precisely the same features as Vieira et al. (2016), which are given Appendix B for completeness. One important difference is that for decoding we use minimum Bayes risk decoding (MBR) (Bickel and Doksum, 2015) under hamming loss,[7] unlike the prior work. We found that MBR improves accuracy on development data for both methods.

We will compare two methods, VoCRF *with* failure arcs and *without*.[8] Using the procedure described in §3 determine a tags $\mathcal{W}$ for various (λ, B)-pairs using the development set (with early stopping). Prior work, swept γ directly (without the budget-trick). Additionally, they limited $|w| \leq 3$ for all $w \in \mathcal{W}$, we do not make such a restriction for either method. We train for 30 passes over the training data. We report the results in Tab. 1. Overall, the accuracy is comparable with popular open-source CRF taggers, e.g., MARMOT (Müller et al., 2013).

5 Conclusion

We have presented a novel algorithm for inference in VoCRF. We derive our algorithm using the concept of a failure arc, a popular technique in the finite-state language modeling literature. We provide detailed pseudocode and analysis of our scheme. The algorithm yields up to a $|Y|$ speed-up. We show that this improvement is also reflected empirically, we offer series of tagging experiments, using multi-lingual POS tagging.

[5] Of course, this approach is *heuristic* because γ_t *may* oscillate causing optimization to never fully converge.

[6] Our code is available at the following URL: http://github.com/timvieira/vocrf

[7] MBR straightforward to compute given Algs. 1 and 2 since it amounts to computing the expected counts (marginal probabilities in this case), $\frac{\partial \log Z_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_y} = p(Y_t = y|\boldsymbol{x})$, of each symbol y at each position t in the input and taking $\widehat{y}_t = \text{argmax}_y p(Y_t = y|\boldsymbol{x})$ and returning $\widehat{\boldsymbol{y}}$.

[8] In the final version, we will include paired-permutation significance tests.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sapporo, Japan, pages 40–47.

Peter J. Bickel and Kjell A. Doksum. 2015. *Mathematical statistics: basic ideas and selected topics*, volume 2. CRC Press.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12:2121–2159.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*. Association for Computational Linguistics, Austin, TX, pages 1–17.

Peter M. Fenwick. 1994. A new data structure for cumulative frequency tables. *Software: Practice and Experience* 24(3):327–336.

Andreas Griewank et al. 1989. On automatic differentiation. *Mathematical Programming: recent developments and applications* 6(6):83–107.

Jan Hajič. 1998. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, COLING-ACL '98, August 10-14, 1998, Université de Montréal, Montréal, Quebec, Canada. Proceedings of the Conference.*. pages 483–490.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*. pages 282–289.

André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A.T. Figueiredo. 2011a. Online learning of structured predictors with multiple kernels. In *AISTATS*. pages 507–515.

André FT Martins, Noah A Smith, Pedro MQ Aguiar, and Mário AT Figueiredo. 2011b. Structured sparsity in structured prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1500–1511.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational linguistics* 23(2):269–311.

Mehryar Mohri. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics* 7(3):321–350.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, pages 322–332.

Anil Nelakanti, Cedric Archambeau, Julien Mairal, Francis Bach, and Guillaume Bouchard. 2013. Structured penalties for log-linear language models. In *EMNLP*. pages 233–243.

Viet Cuong Nguyen, Nan Ye, Wee Sun Lee, and Hai Leong Chieu. 2014. Conditional random field with high-order dependencies for sequence labeling and segmentation. *Journal of Machine Learning Research* 15(1):981–1009.

Joakim Nivre, Cristina Bosco, Jinho Choi, Marie-Catherine de Marneffe, Timothy Dozat, Richárd Farkas, Jennifer Foster, Filip Ginter, Yoav Goldberg, Jan Hajič, Jenna Kanerva, Veronika Laippala, Alessandro Lenci, Teresa Lynn, Christopher Manning, Ryan McDonald, Anna Missilä, Simonetta Montemagni, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Maria Simi, Aaron Smith, Reut Tsarfaty, Veronika Vincze, and Daniel Zeman. 2015. Universal dependencies 1.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague. http://hdl.handle.net/11234/1-1464.

Brian Roark, Cyril Allauzen, and Michael Riley. 2013. Smoothed marginal distribution constraints for language modeling. In *ACL*. pages 43–52.

Mark Schmidt. 2010. *Graphical Model Structure Learning with ℓ_1-Regularization*. Ph.D. thesis, University of British Columbias.

Fei Sha and Fernando C. N. Pereira. 2003. Shallow parsing with conditional random fields. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*.

Tim Vieira, Ryan Cotterell, and Jason Eisner. 2016. Speed-accuracy tradeoffs in tagging with variable-order CRFs and structured sparsity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1973–1978.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning.

Nan Ye, Wee Sun Lee, Hai Leong Chieu, and Dan Wu. 2009. Conditional random fields with high-order features for sequence labeling. In *Advances in Neural Information Processing Systems 22: 23rd*

Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.. pages 2196–2204.

Ming Yuan and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68(1):49–67.

A General Forward-Backward Algorithm for WFSAs with Failure Arcs

Given an acyclic WFSA with failure arcs. This is more general than the specific VoCRF setting in §2. In that setting, the WFSA has a specific topology and is deterministic and all failure arcs have weight 1, but the method below doesn't require any of those things.

We now describe the algorithm. In addition to Alg. 1's $\beta(s)$, we also maintain $\beta(s, a)$ = total weight of all accepting paths from s that can be taken if the next symbol is a. $\beta(s)$ and $\beta(s, a)$ are analogous to $V(s)$ and $Q(s, a)$ in reinforcement learning. In the computation, we must memoize the $\beta(s)$ values to obtain polynomial runtime. For optimal asymptotic runtime, we also memoize the $\beta(s, a)$ values that are actually computed.

We will also maintain a third quantity, $\tilde{\beta}(s, a) = \sum_{e:s \xrightarrow{\phi} s'} \text{weight}(e) \cdot \beta(s', a)$ which sums over the backoff arcs from s, if any. (Usually there's exactly one backoff arc from s, or none in the case of the ϵ state. Writing the summation lets us handle the case where states have zero or more backoff arcs in a unified way.)

$\tilde{\beta}(s, a)$ is a kind of "default value" for $\beta(s, a)$. That is, if s has no outgoing arc labeled a, then the default is correct and $\beta(s, a) = \tilde{\beta}(s, a)$. But otherwise we must override the default and write $\beta(s, a) = \sum_{e:s \xrightarrow{a} s'} \text{weight}(e) \cdot \beta(s')$, which sums over a arcs from s. Finally, $\beta(s) = \left(\sum_{e:s \xrightarrow{\phi} s'} \text{weight}(e) \cdot \beta(s')\right) + \sum_a (\beta(s, a) - \tilde{\beta}(s, a))$, where the second sum ranges over $a \in Y$ such that s has an outgoing arc labeled a.

(Remark: The two cases for $\beta(s, a)$ each multiply an arc probability by another β value. In the first case, the symbol a is read within the first factor, whereas in the second case it is read within the second factor.)

State that the backward algorithm is now simply a loop that computes $\beta(s)$ at each WFSA state s, visiting the states in reverse toposorted order. Cyclic WFSA require solving a linear system, which we've outlined in this appendix, because there is no topological order.

As we stated in §2, computing the gradient can be derived by algorithmic differentiation on the backward algorithm we just outlined.

B Features

We use the exact features in Vieira et al. (2016). We include a description in this appendix for completeness. Thus, the following description is taken directly from their paper and lightly edited to match our notation. Our feature function $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{h}y)$ has indicator features which fire on the identity of the $\boldsymbol{h}y$ as well as features which are a conjunction of y and local properties of $\boldsymbol{x}$. We use the following language-agnostic properties of $(\boldsymbol{x}, t)$:

- The identities of the tokens $x_{t-3}, ..., x_{t+3}$, and the token bigrams (x_{t+1}, x_t), (x_t, x_{t-1}), (x_{t-1}, x_{t+1}). We use special boundary symbols for tokens at positions beyond the start or end of the sentence.
- Prefixes and suffixes of x_t, up to 4 characters long, that occur ≥ 5 times in the training data.
- Indicators for whether x_t is all caps, is lowercase, or has a digit.
- Word shape of x_t, which maps the token string into the following character classes (uppercase, lowercase, number) with punctuation unmodified (e.g., VoCRF-like ⇒ AaAAA-aaaa, $5,432.10 ⇒ $8,888.88).

For efficiency, we hash these properties into 2^{22} bins. The features are obtained by conjoining bins with y_t (Weinberger et al., 2009): e.g., there is a feature that returns 0 unless y_t = NOUN, in which case it counts the number of bin 1234567's properties that $(\boldsymbol{x}, t)$ has. The context features are not hashed.