Speed-Accuracy Tradeoffs in Tagging with Variable-Order CRFs and Structured Sparsity

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

Abstract

We propose a method for learning the structure of variable-order CRFs, a more flexible variant of higher-order linear-chain CRFs. Variable-order CRFs achieve faster inference by including features for only some of the tag n-grams. Our learning method discovers the useful higher-order features at the same time as it trains their weights, by maximizing an objective that combines log-likelihood with a structured-sparsity regularizer. An active-set outer loop allows the feature set to grow as far as needed. On part-of-speech tagging in 5 randomly chosen languages from the Universal Dependencies dataset, our method of shrinking the model achieved a 2–6x speedup over a baseline, with no significant drop in accuracy.

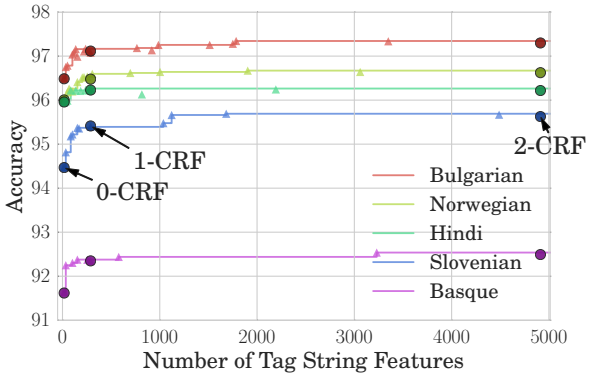

Figure 1: Speed-accuracy tradeoff curves on test data for the 5 languages. Large dark circles represent the k-CRFs of ascending orders along x-axis (marked on for Slovenian). Smaller triangles each represent a VoCRF discovered by sweeping the speed parameters γ. We find faster models at similar accuracy to the best k-CRFs (§5).

1 Introduction

Conditional Random Fields (CRFs) (Lafferty et al., 2001) are a convenient formalism for sequence labeling tasks common in NLP. A CRF defines a feature-rich conditional distribution over tag sequences (output) given an observed word sequence (input).

The key advantage of the CRF framework is the flexibility to consider arbitrary features of the input, as well as enough features over the output structure to encourage it to be well-formed and consistent. However, inference in CRFs is fast only if the features over the output structure are limited. For example, an order-k CRF (or "k-CRF" for short, with $k > 1$ being "higher-order") allows expressive features over a window of $k+1$ adjacent tags (as well as the input), and then inference takes time $\mathcal{O}(n \cdot |Y|^{k+1})$, where Y is the set of tags and n is the length of the input.

How large does k need to be? Typically $k = 2$ works well, with big gains from $0 \rightarrow 1$ and modest gains from $1 \rightarrow 2$ (Fig. 1). Small k may be sufficient when there is enough training data to allow the model to attend to many fine-grained features of the input (Toutanova et al., 2003; Liang et al., 2008). For example, when predicting POS tags in morphologically-rich languages, certain words are easily tagged based on their spelling without considering the context ($k = 0$). In fact, such languages tend to have a more free word order, making tag context less useful.

We investigate a hybrid approach that gives the accuracy of higher-order models while reducing runtime. We build on variable-order CRFs (Ye et al., 2009) (VoCRF), which support features on tag subsequences of mixed orders. Since only modest gains are obtained from moving to higher-order models, we posit that only a small fraction of the higher-order features are necessary. We introduce a hyperparameter γ that *discourages* the model from using many higher-order features (= faster inference) and a hyperparameter λ that *encourages* generalization. Thus, sweeping a range of values for γ and λ gives rise to a

*Equal contribution

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1973–1978,
Austin, Texas, November 1-5, 2016.

number of operating points along the speed-accuracy curve (triangle points in Fig. 1).

We present three contributions: (1) A simplified exposition of VoCRFs, including an algorithm for computing gradients that is asymptotically more efficient than prior art (Cuong et al., 2014). (2) We develop a structure learning algorithm for *discovering* the essential set of higher-order dependencies so that inference is fast *and* accurate. (3) We investigate the effectiveness of our approach on POS tagging in five diverse languages. We find that the amount of required context for accurate prediction is highly language-dependent. In all languages, however, our approach meets the accuracy of fixed-order models at a fraction of the runtime.

2 Variable-Order CRFs

An order-k CRF (k-CRF, for short) is a conditional probability distribution of the form

$$p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}(\boldsymbol{x})} \exp\Big(\sum_{t=1}^{n+1} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, t, y_{t-k} \dots y_t)\Big)$$

where n is the length of the input $\boldsymbol{x}$, $\boldsymbol{\theta} \in \mathbb{R}^d$ is the model parameter, and $\boldsymbol{f}$ is an arbitrary user-defined function that computes a vector in $\mathbb{R}^d$ of features of the tag substring $\boldsymbol{s} = y_{t-k} \dots y_t$ when it appears at position t of input $\boldsymbol{x}$. We define y_i to be a distinguished boundary tag # when $i \notin [1, n]$.

A variable-order CRF or VoCRF is a refinement of the k-CRF, in which $\boldsymbol{f}$ may not always depend on all $k+1$ of the tags that it has access to. The features of a particular tag substring $\boldsymbol{s}$ may sometimes be determined by a shorter suffix of $\boldsymbol{s}$.

To be precise, a VoCRF specifies a finite set $\mathcal{W} \subset Y^*$ that is sufficient for feature computation (where Y^* denotes the set of all tag sequences).[1] The VoCRF's featurization function $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{s})$ is then defined as $\boldsymbol{f}'(\boldsymbol{x}, t, \boldsymbol{w}(\boldsymbol{s}))$ where $\boldsymbol{f}'$ can be any function and $\boldsymbol{w}(\boldsymbol{s}) \in Y^*$ is the longest suffix of $\boldsymbol{s}$ that appears in $\mathcal{W}$ (or ε if none exists). The full power of a k-CRF can be obtained by specifying $\mathcal{W} = Y^{k+1}$, but smaller $\mathcal{W}$ will in general allow speedups.

To support our algorithms, we define $\overline{\mathcal{W}}$ to be the closure of $\mathcal{W}$ under prefixes and last-character substitution. Formally, $\overline{\mathcal{W}}$ is the smallest nonempty superset of $\mathcal{W}$ such that if $\boldsymbol{h}y \in \overline{\mathcal{W}}$ for some $\boldsymbol{h} \in Y^*$ and $y \in Y$, then $\boldsymbol{h} \in \overline{\mathcal{W}}$ and also $\boldsymbol{h}y' \in \overline{\mathcal{W}}$ for all $y' \in Y$. This implies that we can factor $\overline{\mathcal{W}}$ as $\mathcal{H} \times Y$, where $\mathcal{H} \subset Y^*$ is called the set of *histories*.

We now define $\textsc{Next}(h, y)$ to return the longest suffix of hy that is in $\mathcal{H}$ (which may be hy itself, or even ε). We may regard NEXT as the transition function of a deterministic finite-state automaton (DFA) with state set $\mathcal{H}$ and alphabet Y. If this DFA is used to read any tag sequence $\boldsymbol{y} \in Y^*$, then the arc that reads y_t comes from a state $\boldsymbol{h}$ such that $\boldsymbol{h}y_t$ is the longest suffix of $\boldsymbol{s} = y_{t-k} \dots y_t$ that appears in $\overline{\mathcal{W}}$—and thus $\boldsymbol{w}(\boldsymbol{h}y_t) = \boldsymbol{w}(\boldsymbol{s}) \in \mathcal{W}$ and provides sufficient information to compute $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{s})$.[2]

For a given $\boldsymbol{x}$ of length n and given parameters $\boldsymbol{\theta}$, the log-normalizer $\log Z_{\boldsymbol{\theta}}(\boldsymbol{x})$—which will be needed to compute the log-probability in eq. (1) below—can be found in time $\mathcal{O}(|\overline{\mathcal{W}}|\, n)$ by dynamic programming. Concise pseudocode is in Alg. 1. In effect, this

Algorithm 1 FORWARD: Compute $\log Z_{\boldsymbol{\theta}}(\boldsymbol{x})$.

$\alpha(\cdot, \cdot) = 0$; $\alpha(0, \#) = 1$ ▷ *initialization*
for $t = 1$ **to** $n+1$:
  if $t = n+1$ **then** $Y_t = \{\#\}$ **else** $y_t = Y \backslash \{\#\}$
  for $\boldsymbol{h} \in \mathcal{H}, y_t \in Y_t$:
    $\boldsymbol{h}' = \textsc{Next}(\boldsymbol{h}, y_t)$
    $z = \exp\big(\boldsymbol{\theta}^\top \boldsymbol{f}'(\boldsymbol{x}, t, \boldsymbol{w}(\boldsymbol{h}y_t))\big)$
    $\alpha(t, \boldsymbol{h}')$ += $\alpha(t-1, \boldsymbol{h}) \cdot z$
$Z = \sum_{\boldsymbol{h} \in \mathcal{H}} \alpha(n+1, \boldsymbol{h})$ ▷ *sum over final states*
return $\log Z$

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

$\beta(\cdot, \cdot) = 0$; $\boldsymbol{\Delta} = \mathbf{0}$
$\beta(n+1, \boldsymbol{h}) = 1$ for all $\boldsymbol{h} \in \mathcal{H}$ ▷ *initialization*
for $t = n+1$ **downto** 1 :
  for $\boldsymbol{h} \in \mathcal{H}, y_t \in Y_t$:
    $\boldsymbol{h}' = \textsc{Next}(\boldsymbol{h}, y_t)$
    $z = \exp\big(\boldsymbol{\theta}^\top \boldsymbol{f}'(\boldsymbol{x}, t, \boldsymbol{w}(\boldsymbol{h}y_t))\big)$
    $\boldsymbol{\Delta}$ += $\boldsymbol{f}'(\boldsymbol{x}, t, \boldsymbol{w}(\boldsymbol{h}y_t)) \cdot \alpha(t-1, \boldsymbol{h}) \cdot z \cdot \beta(t, \boldsymbol{h}')$
    $\beta(t-1, \boldsymbol{h})$ += $z \cdot \beta(t, \boldsymbol{h}')$
return $\boldsymbol{\Delta}/Z$

[1] The constructions given in this section assume that $\mathcal{W}$ does not contain ε nor any sequence having ## as a proper prefix.

[2] Our DFA construction is essentially that of Cotterell and Eisner (2015, Appendix B.5). However, Appendix B of that paper also gives a construction that obtains an even smaller DFA by using failure arcs (Allauzen et al., 2003), which remove the requirement that $\overline{\mathcal{W}}$ be closed under last-character substitution. This would yield a further speedup to our Alg. 1 (replacing it with the efficient backward algorithm in footnote 16 of that paper) and similarly to our Alg. 2 (by differentiating the new Alg. 1).

runs the forward algorithm on the lattice of taggings given by length-n paths through the DFA.

For finding the parameters $\boldsymbol{\theta}$ that minimize eq. (1) below, we want the gradient $\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}}(\boldsymbol{x})$. By applying algorithmic differentiation to Alg. 1, we obtain Alg. 2, which uses back-propagation to compute the gradient (asymptotically) as fast as Alg. 1 and $|\mathcal{H}|$ times faster than Cuong et al. (2014)'s algorithm—a significant speedup since $|\mathcal{H}|$ is often quite large (up to 300 in our experiments). Algs. 1–2 together effectively run the forward-backward algorithm on the lattice of taggings.[3]

It is straightforward to modify Alg. 1 to obtain a Viterbi decoder that finds the most-likely tag sequence under $p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})$. It is also straightforward to modify Alg. 2 to compute the marginal probabilities of tag substrings occurring at particular positions.

3 Structured Sparsity and Active Sets

We begin with a k-CRF model whose feature vector $\boldsymbol{f}(\boldsymbol{x}, t, y_{t-k} \dots y_t)$ is partitioned into non-stationary local features $\boldsymbol{f}^{(1)}(\boldsymbol{x}, t, y_t)$ and stationary higher-order features $\boldsymbol{f}^{(2)}(y_{t-k} \dots y_t)$. Specifically, $\boldsymbol{f}^{(2)}$ includes an indicator feature for each tag string $\boldsymbol{w} \in Y^*$ with $1 \leq |\boldsymbol{w}| \leq k+1$, where $f^{(2)}_{\boldsymbol{w}}(y_{t-k} \dots y_t)$ is 1 if $\boldsymbol{w}$ is a suffix of $y_{t-k} \dots y_t$ and is 0 otherwise.[4]

To obtain the advantages of a VoCRF, we merely have to choose a *sparse* weight vector $\boldsymbol{\theta}$. The set $\mathcal{W}$ can then be defined to be the set of strings in Y^* whose features have nonzero weight. Prior work (Cuong et al., 2014) has left the construction of $\mathcal{W}$ to domain experts or "one size fits all" strategies (e.g., k-CRF). Our goal is to choose $\boldsymbol{\theta}$—and thus $\mathcal{W}$—so that inference is accurate *and* fast.

Our approach is to modify the usual L_2-regularized log-likelihood training criterion with a carefully defined runtime penalty scaled by a parameter γ to balance competing objectives: likelihood on the data $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^m$ vs. efficiency (small $\mathcal{W}$).

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

Recall that the runtime of inference on a given sentence is proportional to the size of $\overline{\mathcal{W}}$, the *closure* of $\mathcal{W}$ under prefixes and last-character replacement. (Any tag strings in $\overline{\mathcal{W}} \backslash \mathcal{W}$ can get nonzero weight without increasing runtime.) Thus, $\mathcal{R}(\boldsymbol{\theta})$ would ideally measure $|\overline{\mathcal{W}}|$, or proportionately, $|\mathcal{H}|$. Experimentally, we find that $|\overline{\mathcal{W}}|$ has $> 99\%$ Pearson correlation with wallclock time, making it an excellent proxy for wallclock time while being more replicable.

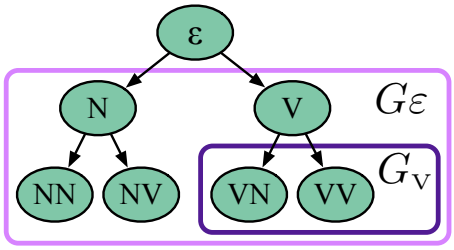

Figure 2: A visual depiction of the tree-structured group lasso penalty. Each node represents a tag string feature. The group indexed by a node's tag string is defined as the *set* of features that are *proper descendants* of the node. For example, the lavender box indicates the largest group G_ε and the aubergine box indicates a smaller group G_{V}. To avoid clutter, not all groups are marked.

We relax this regularizer to a convex function—a tree-structured *group lasso* objective (Yuan and Lin, 2006; Nelakanti et al., 2013). For each string $\boldsymbol{h} \in Y^*$, we have a group $G_{\boldsymbol{h}}$ consisting of the indicator features (in $\boldsymbol{f}^{(2)}$) for all strings $w \in \overline{\mathcal{W}}$ that have $\boldsymbol{h}$ as a proper prefix. Fig. 2 gives a visual depiction. We now define $\mathcal{R}(\boldsymbol{\theta}) = \sum_{\boldsymbol{h} \in Y^*} ||\boldsymbol{\theta}_{G_{\boldsymbol{h}}}||_2$. This penalty encourages each group of weights to remain all at zero (thereby conserving runtime, in our setting, because it means that $\boldsymbol{h}$ does not need to be added to $\mathcal{H}$). Once a single weight in a group becomes nonzero, the "initial inertia" induced by the group lasso penalty is overcome, and other features in the group can be more cheaply adjusted away from zero.

Although eq. (1) is now convex, directly optimizing it would be expensive for large k, since $\boldsymbol{\theta}$ then contains very many parameters. We thus use a heuristic optimization algorithm, the *active set* method (Schmidt, 2010), which starts with a low-dimensional $\boldsymbol{\theta}$ and *incrementally* adds features to the model. This also frees us from needing to specify a limit k. Rather, $\mathcal{W}$ grows until further extensions are unhelpful, and then implicitly $k = \max_{\boldsymbol{w} \in \mathcal{W}} |\boldsymbol{w}| - 1$.

The method defines $\boldsymbol{f}^{(2)}$ to include indicator features for all tag sequences $\boldsymbol{w}$ in an *active set* $\mathcal{W}_{\text{active}}$. Thus, $\boldsymbol{\theta}^{(2)}$ is always a vector of $|\mathcal{W}_{\text{active}}|$ real numbers. Initially, we take $\mathcal{W}_{\text{active}} = Y$ and $\boldsymbol{\theta} = \mathbf{0}$. At each

[3] Eisner (2016) explains the connection between algorithmic differentiation and the forward-backward algorithm.

[4] Extensions to richer sets of higher-order features are possible, such as conjunctions with properties of the words at position t.

active set iteration, we fully optimize eq. (1) to obtain a sparse $\boldsymbol{\theta}$ and a set $\mathcal{W} = \{\boldsymbol{w} \in \mathcal{W}_{\text{active}} \mid \boldsymbol{\theta}^{(2)}_{\boldsymbol{w}} \neq 0\}$ of features that are known to be "useful."[5] We then update $\mathcal{W}_{\text{active}}$ to $\{\boldsymbol{w}y \mid \boldsymbol{w} \in \mathcal{W}, y \in Y\}$, so that it includes single-tag extensions of these useful features; this expands $\boldsymbol{\theta}$ to consider additional features that plausibly might prove useful. Finally, we complete the iteration by updating $\mathcal{W}_{\text{active}}$ to its closure $\overline{\mathcal{W}_{\text{active}}}$, simply because this further expansion of the feature set will not slow down our algorithms. When eq. (1) is re-optimized at the next iteration, some of these newly added features in $\mathcal{W}_{\text{active}}$ may acquire nonzero weights and thus enter $\mathcal{W}$, allowing further extensions. We can halt once $\mathcal{W}$ no longer changes.

As a final step, we follow common practice by running "debiasing" (Martins et al., 2011a), where we fix our $\boldsymbol{f}^{(2)}$ feature set to be given by the final $\overline{\mathcal{W}}$, and retrain $\boldsymbol{\theta}$ without the group lasso penalty term.

In practice, we optimized eq. (1) using the online proximal gradient algorithm SPOM (Martins et al., 2011b) and Adagrad (Duchi et al., 2011) with $\eta = 0.01$ and 15 inner epochs. We limited to 3 active set iterations, and as a result, our final $\overline{\mathcal{W}}$ contained at most tag trigrams.

4 Related Work

Our paper can be seen as transferring methods of Cotterell and Eisner (2015) to the CRF setting. They too used tree-structured group lasso and active set to select variable-order n-gram features $\mathcal{W}$ for globally-normalized sequence models (in their case, to rapidly and accurately approximate beliefs during message-passing inference). Similarly, Nelakanti et al. (2013) used tree-structured group lasso to regularize a variable-order language model (though their focus was *training* speed). Here we apply these techniques to *conditional* models for tagging.

Our work directly builds on the variable-order CRF of Cuong et al. (2014), with a speedup in Alg. 2, but our approach also learns the VoCRF structure. Our method is also related to the *generative* variable-order tagger of Schütze and Singer (1994).

Our static feature selection chooses a single model that permits fast exact marginal inference, similar to learning a low-treewidth graphical model (Bach and Jordan, 2001; Elidan and Gould, 2008). This contrasts with recent papers that learn to do approximate 1-best inference using a sequence of models, whether by dynamic feature selection within a greedy inference algorithm (Strubell et al., 2015), or by gradually increasing the feature set of a 1-best global inference algorithm and pruning its hypothesis space after each increase (Weiss and Taskar, 2010; He et al., 2013).

Schmidt (2010) explores the use of group lasso penalties and the active set method for learning the structure of a graphical model, but does not consider learning repeated structures (in our setting, $\overline{\mathcal{W}}$ defines a structure that is reused at each position). Steinhardt and Liang (2015) jointly modeled the amount of context to use in a variable-order model that dynamically determines how much context to use in a beam search decoder.

5 Experiments[6]

Data: We conduct experiments on multilingual POS tagging. The task is to label each word in a sentence with one of $|Y| = 17$ labels. We train on five typologically-diverse languages from the Universal Dependencies (UD) corpora (Petrov et al., 2012): Basque, Bulgarian, Hindi, Norwegian and Slovenian. For each language, we start with the original train / dev / test split in the UD dataset, then move random sentences from train into dev until the dev set has 3000 sentences. This ensures more stable hyperparameter tuning. We use these new splits below.

Eval: We train models with $(\lambda, \gamma) \in \{10^{-4} \cdot m, 10^{-3} \cdot m, 10^{-2} \cdot m\} \times \{0, 0.1 \cdot m, 0.2 \cdot m, \ldots, m\}$, where m is the number of training sentences. To tag a dev or test sentence, we choose its most probable tag sequence. For each of several model sizes, Table 1 selects the model of that size that achieved the highest per-token tagging accuracy on the dev set, and reports that model's accuracy on the test set.

Features: Recall from §3 that our features include non-stationary zeroth-order features $\boldsymbol{f}^{(1)}$ as well as the stationary features based on $\mathcal{W}$. For $\boldsymbol{f}^{(1)}(\boldsymbol{x}, t, y_t)$ we consider 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}),

[5] Each gradient computation in this inner optimization takes time $\mathcal{O}(|\overline{\mathcal{W}_{\text{active}}}|\, n)$, which is especially fast at early iterations.

[6] Code and data are available at the following URLs:
`http://github.com/timvieira/vocrf`
`http://universaldependencies.org`

	k-CRF ($\lvert\overline{\mathcal{W}}\rvert = 17^{k+1}$)			VoCRF at different model sizes $\lvert\overline{\mathcal{W}}\rvert$ (which is proportional to runtime)									
	0 (17)	1 (289)	2 (4913)	≤ 34	≤ 85	≤ 170	≤ 340	≤ 850	≤ 1700	≤ 2550	≤ 3400	≤ 4250	≤ 5100
Ba	$91.61^{1,2}$	$\underline{92.35}^{0}$	92.49^{0}	$92.25^{0,2}$	$92.25^{0,2}$	$\underline{92.38}^{0}$	92.34^{0}	92.44^{0}	92.44^{0}	92.44^{0}	$\mathbf{92.54}^{0}$	92.54^{0}	92.54^{0}
Bu	$96.48^{1,2}$	$97.11^{0,2}$	$\underline{97.29}^{0,1}$	$96.75^{0,1,2}$	$96.78^{0,1,2}$	$96.99^{0,1,2}$	$97.08^{0,2}$	$\underline{97.18}^{0,1}$	$97.25^{0,1}$	$\mathbf{97.34}^{0,1}$	$97.34^{0,1}$	$97.34^{0,1}$	$97.34^{0,1}$
Hi	$95.96^{1,2}$	$\underline{96.22}^{0}$	96.21^{0}	$95.97^{1,2}$	$\underline{96.22}^{0}$	96.22^{0}	96.26^{0}	96.13^{0}	96.13^{0}	$\mathbf{96.24}^{0}$	96.24^{0}	96.24^{0}	96.24^{0}
No	$96.00^{1,2}$	$\underline{96.64}^{0}$	96.66^{0}	$96.07^{1,2}$	$96.26^{0,1,2}$	$\underline{96.41}^{0}$	96.60^{0}	96.62^{0}	96.64^{0}	$\mathbf{96.67}^{0}$	96.64^{0}	96.64^{0}	96.64^{0}
Sl	$94.46^{1,2}$	$95.41^{0,2}$	$\underline{95.62}^{0,1}$	$94.82^{1,2}$	$95.18^{0,2}$	$95.36^{0,2}$	$95.39^{0,2}$	$95.39^{0,2}$	$\mathbf{\underline{95.69}}^{0,1}$	$95.69^{0,1}$	$95.69^{0,1}$	$95.69^{0,1}$	$95.67^{0,1}$

Table 1: Part-of-speech tagging with Universal Tags: This table shows test results on 5 languages at different target runtimes. Each row's best results are in **boldface**, where ties in accuracy are broken in favor of faster models. Superscript k indicates that the accuracy is significantly different from the k-CRF (paired permutation test, $p < 0.05$) and this superscript is in blue/red if the accuracy is higher/lower than the k-CRF. In all cases, we find a VoCRF (underlined) that is about as accurate as the 2-CRF (i.e., not significantly less accurate) and far faster, since the 2-CRF has $\lvert\overline{\mathcal{W}}\rvert = 4913$. Fig. 1 plots the Pareto frontiers.

(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 $\Rightarrow$ AaAAA-aaaa, $5,432.10 $\Rightarrow$ $8,888.88).

For efficiency, we hash these properties into 2^{22} bins. The $\boldsymbol{f}^{(1)}$ features are obtained by conjoining these bins with y_t (Weinberger et al., 2009): e.g., there is a feature that returns 0 unless $y_t = \textsf{NOUN}$, in which case it counts the number of bin 1234567's properties that $(\boldsymbol{x}, t)$ has. (The $\boldsymbol{f}^{(2)}$ features are not hashed.)

Results: Our results are presented in Fig. 1 and Table 1. We highlight two key points: (i) Across *all languages* we learned a tagger about as accurate as a 2-CRF, but much faster. (ii) The size of the set $\overline{\mathcal{W}}$ required is highly language-dependent. For many languages, learning a full k-CRF is wasteful; our method resolves this problem.

In each language, the fastest "good" VoCRF is rather faster than the fastest "good" k-CRF (where "good" means statistically indistinguishable from the 2-CRF). These two systems are underlined; the underlined VoCRF systems are smaller than the underlined k-CRF systems (for the 5 languages respectively) by factors of 1.9, 6.4, 3.4, 1.9, and 2.9. In every language, we learn a VoCRF with $\lvert\overline{\mathcal{W}}\rvert \leq 850$ that is not significantly worse than a 2-CRF with $\lvert\overline{\mathcal{W}}\rvert = 17^3 = 4913$.

We also notice an interesting language-dependent effect, whereby certain languages require a small number of tag strings in order to perform well. For example, Hindi has a competitive model that ignores the previous tag y_{t-1} unless it is in $\{\textsf{NOUN}, \textsf{VERB}, \textsf{ADP}, \textsf{PROPN}\}$: thus the stationary features are 17 unigrams plus 4×17 bigrams, for a total of $\lvert\overline{\mathcal{W}}\rvert = 85$. At the other extreme, the Slavic languages Slovenian and Bulgarian seem to require more expressive models over the tag space, remembering as many as 98 useful left-context histories (unigrams and bigrams) for the current tag. An interesting direction for future research would be to determine which morpho-syntactic properties of a language tend to increase the complexity of tagging.

6 Conclusion

We presented a structured sparsity approach for structure learning in VoCRFs, which achieves the accuracy of higher-order CRFs at a fraction of the runtime. Additionally, we derive an asymptotically faster algorithm for the gradients necessary to train a VoCRF than prior work. Our method provides an effective speed-accuracy tradeoff for POS tagging across five languages—confirming that significant speed-ups are possible with little-to-no loss in accuracy.

Acknowledgments: This material is based in part on research sponsored by DARPA under agreement number FA8750-13-2-0017 (DEFT program) and the National Science Foundation under Grant No. 1423276. The second author was funded by a DAAD Long-term research grant and an NDSEG fellowship.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of ACL*, pages 40–47.

F. R. Bach and M. I. Jordan. 2001. Thin junction trees. In *NIPS*, pages 569–576.

Ryan Cotterell and Jason Eisner. 2015. Penalized expectation propagation for graphical models over strings. In *NAACL-HLT*, pages 932–942.

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

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. In *Proceedings of the EMNLP 16 Workshop on Structured Prediction for NLP*, Austin, TX, November.

G. Elidan and S. Gould. 2008. Learning bounded treewidth Bayesian networks. In *NIPS*, pages 417–424.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *EMNLP*, pages 1455–1464.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289.

Percy Liang, Hal Daumé III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *ICML*, pages 592–599.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011a. Structured sparsity in structured prediction. In *EMNLP*, pages 1500–1511.

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

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

Slav Petrov, Dipanjan Das, and Ryan T. McDonald. 2012. A universal part-of-speech tagset. In *LREC*, pages 2089–2096.

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

Hinrich Schütze and Yoram Singer. 1994. Part-of-speech tagging using a variable memory Markov model. In *ACL*, pages 181–187.

Jacob Steinhardt and Percy Liang. 2015. Reified context models. In *ICML*, pages 1043–1052.

Emma Strubell, Luke Vilnis, Kate Silverstein, and Andrew McCallum. 2015. Learning dynamic feature selection for fast sequential prediction. In *ACL*, pages 146–155.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *ACL*, pages 173–180.

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

David J. Weiss and Benjamin Taskar. 2010. Structured prediction cascades. In *AISTATS*, pages 916–923.

Nan Ye, Wee S. Lee, Hai L. Chieu, and Dan Wu. 2009. Conditional random fields with high-order features for sequence labeling. In *NIPS*, 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.