Bigram Constrained Linear Chain Conditional Random Fields

Emmanuel Liossis (emmanuel.liossis@gmail.com)

***Abstract*—It is known that Linear Chain Conditional Random Fields have quadratic time and space complexity in terms of the output tags set cardinality. This fact poses a prohibitive performance penalty when the tag set is large, such as in language applications where the language has a rich set of morphosyntactic tags. However, knowledge of the allowed tag bigram combinations can lead to significant speedup and memory savings, often by several orders of magnitude, for both training and inference. This theoretical exposition presents how to exploit this knowledge by introducing steps and data structures to the ordinary Linear Chain Conditional Random Field implementation in order to achieve these savings.**

I. Introduction

Linear Chain Conditional Random Fields [1] have been used successfully in many sequence classification tasks, including language processing applications such as part-of-speech tagging when the cardinality of the output tag set is no more than a few dozen. However, certain highly inflective languages have a rich set of morphosyntactic tags where the cardinality exceeds one thousand. Bearing in mind that Linear Chain Conditional Random Fields have quadratic time and space complexity with respect to this cardinality, application to such languages becomes impractical.

This poses the need to exploit any convenient sparsity pattern known about the data. We observe that, in the graphical model of the Linear Chain Conditional Random Field, adjacent elements of the output sequence have pairwise connections. This pairwise connectivity drives message passing and it is responsible for the quadratic complexity of the operations. The above suggest that knowldge about the possible tag bigrams is a good fit for boosting message passing.

We will derive a Bigram Constrained Linear Chain Conditional Random Field (BC-LCCRF) which will take advantage of the allowed tag bigrams. It will be a general approach which will contain the original "full" Linear Chain Conditional Random Field (F-LCCRF) as a special case, falling back to the original performance with only a small amount of overhead. We proceed as follows: First we review relevant previous work. Then we intruduce the data structures facilitating the formulation and its implementation, and consequently we establish the formulation. Finally, there is a theoretical comparison against previous work and some notes on applications. An implementation in C# can be found at https://github.com/grammophone/Grammophone.CRF.

II. Previous work

The cardinality of the tag set of highly inflectional languages has always been a concern when processed using Conditional Random Fields. Waszczuk [2] has faced this challenge for the Polish language which also presents a tag set with cardinality of more than a thousand, and produced a form called Constrained Conditional Random Field, which accepts as an input the constraint r of the possible tags per output sequence position:

$$r = (r_1 \subseteq Y, \ldots, r_n \subseteq Y,) \tag{1}$$

The above constraint specification favors languages and domains in general where tags can be easily expressed as position-dependent. However, this specification becomes difficult with languages such as Ancient Greek, where the various parts of speech can have many valid juxtapositons in a sentence.

Expressing sparsity in terms of bigrams is a more general, less domain-dependent approach which is better suited for such languages. Bigram sparsity is proposed by Sokolovska et al in [3]. They consider feature functions falling into two special groups, unigram functions and bigram sparse boolean functions, and they take advantage of the sparsity of the "true" values of the latter in order to achieve efficiency.

Moving towards the same direction of bigram sparsity as [3], we will introduce a scheme which allows for general feature functions but which also has the benefits of [2] through implicit inference of the sequence of possible tags (1).

III. Data structures prerequiesites

Before we describe the method, we introduce the necessary data structures. Their simplified diagram is shown in Figure 1.

A. Bigram: The `Tuple<F, S>` class

This is the class which represents a bigram, shown in middle left of Figure 1. Its two components are held in properties `Item1` and `Item2` having generic types `F` and `S` correspondingly. The class provides implementations of the `Equals` and `GetHashCode` methods using the respective implementations of its two components which are expected to run in O(1) time, thus it is automatically ready for equality comparison and for hashing algorithms.

Figure 1. The simplified UML diagram of the auxilliary data structures.

B. Bigram container: The `BiGramSet<Y>` class

The `BiGramSet<Y>` class (bottom right in Figure 1) is a collection of `Tuple<Y, Y>` bigram items, with the additional capabilities:

- It can fetch all bigrams having a given first element in approximate O(1) time (method `GetByFirst`).
- It can fetch all bigrams having a given second element in approximate O(1) time (method `GetBySecond`).
- It can check whether a bigram is a member of the collection in approximate O(1) time (method `Contains`).

The implementation of the bigram set relies partly on a hash set for storing all bigrams and for member checking in approximate O(1) time. This functionality is inherited by the parent `Bag<Tuple<Y, Y>>`. Additionally, the class contains two private hash multi-dictionaries (not shown), allowing many items under the same key, in order to support the `GetByFirst` and `GetBySecond` methods with approximate O(1) performance.

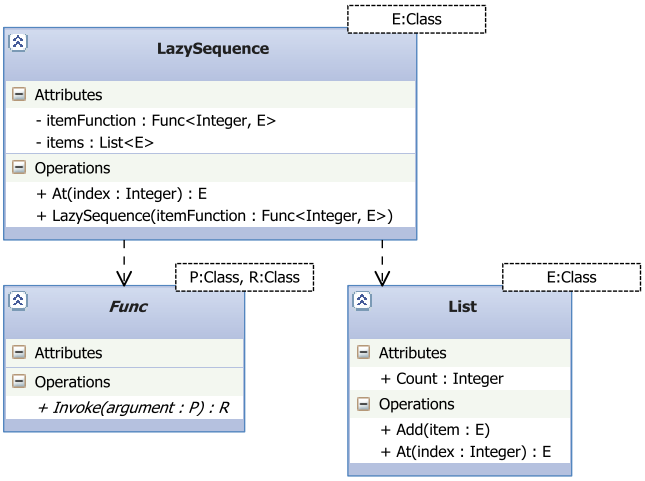

Figure 2. The `LazySequence<E>` diagram.

C. Infinite sequence: The `LazySequence<E>` class

This is a thread-safe, lazily evaluated sequence of elements of generic type `E`, in the style of the lists found in functional languages. It differs though in having O(1) indexed random access time for an element which has already been computed. When computing a missing element, any previous missing elements are computed recursively as necessary. The elements of the sequence are implicitly defined by a function `Func<Integer, E>`, taking the index as an argument and returning the corresponding element.

That function may access any previous element, as the above recursive behavior guarantees that it will be present.

IV. Formulation

We can now describe the Bigram Constrained Conditional Random Field. The previously reviewed data structures will facilitate the implementation of the scheme.

A. Definition

We establish some notation. Let Y be the set of tags, $Y_+ = Y \cup \{S, E\}$ be the tag set extended with the special start and end tags. Let X be the set from where input x takes its values. The input x is not constrained to be a sequence, it can be anything. Let $B \subseteq Y_+^2$ be the set of possible bigrams. Let $f : \left(X, Y_+^2, \mathbb{N}\right) \to \mathbb{R}^d$ be d effective feature functions packed as a single function returning a vector with d dimensions. The probability of an output sequence $y \in Y_+^n$ given an input $x \in X$ and model parameters vector $w \in \mathbb{R}^d$ according to Bigram Constrained Linear Chain Conditional Random Field (BC-LCCRF) follows:

$$p\left(y|x;w\right) = \frac{\exp\left[w^T \sum_{i=0}^{n} f\left(x, y_{i-1}, y_i, i\right)\right]}{Z\left(x, w\right)} \cdot I\left[y \in Y^p\right] \quad (2)$$

where

$$Z\left(x, w\right) = \sum_{y \in Y^p} \exp\left[w^T \sum_{i=0}^{n} f\left(x, y_{i-1}, y_i, i\right)\right] \quad (3)$$

The output sequence y has n non-trivial elements and has zero based indices, always implying the boundary conditions $y_{-1} = S$ and $y_n = E$. The Y^p is the set of all possible tag sequences of length n, according to the set of allowed bigrams B, which additionally satisfy:

- The previous boundary conditions $y_{-1} = S$ and $y_n = E$.
- $y_i \notin \{S, E\}$ when $i = 0 \ldots n-1$.

The Y^p set, being constrained by bigrams in B, is what discriminates the scheme from the ordinary Full Linear Chain Conditional Random Field (F-LCCRF). But if we specify as bigrams the complete cartesian product of tags, the scheme falls back to F-LCCRF.

The feature functions are expressed as a vector function f. Implementation-wise, this vector can be sparse in order to speed up the inner product with the model parameters vector w. We observe that we have no restriction upon feature functions other than accessing only a pair of neighboring output tags at a time. Feature functions don't need to be boolean functions of the input and the tags, they don't need to be boolean functions of the tag bigrams either; they can return any real value and can make use of the input and the tag pair in any way. The input doesn't need to be a sequence, and any part of it or all of it can be considered during a feature function evaluation.

B. Inference

The most likely output sequence $\hat{y} \in Y^p$ given an input x and the model parameters w is the one which maximizes the numerator of (2), as the denominator is not dependent on the output:

$$\begin{aligned} \hat{y} &= \arg\max_{y \in Y^p} \left[w^T \sum_{i=0}^{n} f\left(x, y_{i-1}, y_i, i\right)\right] \\ &= \arg\max_{y \in Y^p} \sum_{i=0}^{n} g_i\left(y_{i-1}, y_i\right) \end{aligned} \quad (4)$$

where

$$g_i\left(y_{i-1}, y_i\right) = w^T f\left(x, y_{i-1}, y_i, i\right) \quad (5)$$

The $g_i\left(y_{i-1}, y_i\right)$ can be implemented as an array of $n+1$ hash dictionaries having as key the bigram and storing numeric values, ie `Dictionary<Tuple<Y, Y>, Double>[*]`. Thus the access time in terms of a bigram is approximately O(1).

Before we move on to sequence definitions, let us define the functions prev and next, which return the possible tags preceding or following a given tag according to B:

$$\begin{aligned} \text{prev}\left(v\right) &= \{u|\left(u, v\right) \in B\} \\ \text{next}\left(u\right) &= \{v|\left(u, v\right) \in B\} \end{aligned}$$

We now define the forward open sequence of possible tags, which is infinite and unique in terms of B. This is the sequence of possible tags per position, assuming that they stem from $\{S\}$ according to B, but making no assumption about the end of the sequence:

$$\begin{aligned} t &= \left(t_{-1} \subseteq Y_+, t_0 \subseteq Y_+, t_1 \subseteq Y_+, \ldots, t_i \subseteq Y_+, \ldots\right) \\ t_{-1} &= \{S\} \\ t_i &= \{y_s|\ \exists y_f \in t_{i-1} :\ y_s \in \text{next}\left(y_f\right)\} \end{aligned}$$

Similarly, we define the backward open sequence of possible tags, which is the reverse infinite sequence of possible tags ending to $\{E\}$ according to B:

$$\begin{aligned} q &= \left(q_0 \subseteq Y_+, q_1 \subseteq Y_+, \ldots, q_i \subseteq Y_+, \ldots\right) \\ q_0 &= \{E\} \\ q_i &= \{y_f|\ \exists y_s \in q_{i-1} :\ y_f \in \text{prev}\left(y_s\right)\} \end{aligned}$$

These infinite sequences can be readily implemented as `LazySequence<Y[*]>`. The function for creating an element of either sequence takes advantage of set B being implemented as a `BiGramSet<Y>`, and uses its `GetByFirst` and `GetBySecond` methods to compute next and prev functions correspondingly.

The allowed tags for a sequence of length n can now be defined:

$$\begin{aligned} r &= \left(r_{-1}, r_0, \ldots, r_n\right) \\ r_i &= t_i \cap q_{n-i} \end{aligned}$$

The above sequence depends only on B, which is constant for a given domain, and n. Its computation can thus be effectively cached. We could also further restrict the r_i sets if we have prior positional knowledge as in [2].

Resuming the inference computation in (4), we proceed in a Viterbi-like fashion normally as in F-LCCRF by defining the maximum score of sequences ending with tag $v \in r_k$ at position k:

$$\begin{aligned} U_k(v) &= \max_{y_0,\ldots,y_{k-1}} \sum_{i=0}^{k-1} g_i(y_{i-1}, y_i) + g_k(y_{k-1}, v) \\ &= \max_{\substack{y_{k-1} \in \text{prev}(v) \\ y_{k-1} \in r_{k-1}}} \max_{y_0,\ldots,y_{k-2}} \sum_{i=0}^{k-2} g_i(y_{i-1}, y_i) \\ &+ g_{k-1}(y_{k-2}, y_{k-1}) + g_k(y_{k-1}, v) \\ &= \max_{\substack{y_{k-1} \in \text{prev}(v) \\ y_{k-1} \in r_{k-1}}} U_{k-1}(y_{k-1}) + g_k(y_{k-1}, v) \end{aligned} \quad (6)$$

The base of the above recursion is

$$U_0(v) = g_0(S, v)$$

These observations allow us to achieve fast computation of the expression (6). For each stage k of the recursion, we only iterate over $v \in r_k$ instead of all tags. Then, for each tag v, we only maximize over $u \in \text{prev}(v) \cap r_{k-1}$ instead of all tags.

Implementation-wise, these steps imply which values to store for each g_k function: For every k we only store values for bigrams (y_{k-1}, y_k) by selecting all $y_k \in r_k$, then for each y_k selecting all $y_{k-1} \in \text{prev}(y_k) \cap r_{k-1}$. For every element in $\text{prev}(y_k)$, the intersection test with r_{k-1} can execute in approximate $O(1)$ time if the collections of tags returned by each element of r are based on hash sets. Thus formally, g would be realized as an array of dictionaries having bigrams as keys and scalar values, `Dictionary<Tuple<Y, Y>, Double>[*]`. Similarly to g, we can realize U as an array of dictionaries having tags as keys and scalar values, `Dictionary<Y, Double>[*]`.

In order to infer the best sequence of tags, we set $\hat{y}_n = E$ and move backwards recursively:

$$\hat{y}_{k-1} = \underset{u \in \text{prev}(\hat{y}_k) \cap r_{k-1}}{\arg\max} U_{k-1}(u) + g_k(u, \hat{y}_k)$$

If the above domain $\text{prev}(\hat{y}_k) \cap r_{k-1}$ of the $\arg\max$ operator is found empty, it means that there is no bigram leading to the tag E at $\hat{y}_{n+1}$, thus we report that a sequence of the given length is infeasible according the the bigrams specified in B.

C. Training

Training methods may consist of maximizing the log conditional likelihood of the training data. These methods typically require an expression of the gradient of the log conditional likelihood. Considering the outputs as conditionally independent with respect to the inputs, this is reduced to the sum of the logarithms of the conditional probabilities defined in (2) for each input/output pair. Let us define:

$$F(x, y) = \sum_{i=0}^{n} f(x, y_{i-1}, y_i, i)$$

We proceed similarly as in F-LCCRF by finding the gradient of the log of the conditional probability (2), assuming $y \in Y^p$:

$$\begin{aligned} \nabla_w \log p(y|x; w) &= F(x, y) - \nabla_w \log Z(x, w) \\ &= F(x, y) - \frac{1}{Z(x, w)} \nabla_w Z(x, w) \end{aligned} \quad (7)$$

We elaborate on the the last term, the gradient of the partition function, $\nabla_w Z(x, w)$, using its definition in (3):

$$\begin{aligned} \nabla_w Z(x, w) &= \nabla_w \sum_{y' \in Y^p} \exp\left[w^T F(x, y')\right] \\ &= \sum_{y' \in Y^p} \nabla_w \exp\left[w^T F(x, y')\right] \\ &= \sum_{y' \in Y^p} \exp\left[w^T F(x, y')\right] F(x, y') \end{aligned}$$

Replacing the above in (7) we get:

$$\begin{aligned} \nabla_w \log p(y|x; w) &= F(x, y) \\ &- \sum_{y' \in Y^p} \frac{\exp\left[w^T F(x, y')\right]}{Z(x, w)} F(x, y') \\ &= F(x, y) - E_{y' \sim p(y'|x;w)}\left[F(x, y')\right] \end{aligned} \quad (8)$$

We will focus on computing the second term of (8) efficiently. The numerator in (8) is the unnormalized probability of output y' given an input x and model parameters w. In order to compute it, we define $\alpha_k(y_k)$ to be the unnormalized probability of an output sequence ending at position k with a tag $y_k \in r_k$, always implying $y_{-1} = S$. This is the forward vector:

$$\alpha_k(y_k) = \sum_{y_0 \ldots y_{k-1}} \exp\left[w^T \sum_{i=0}^{k} f(x, y_{i-1}, y_i, i)\right]$$

Using the definition in (5), we can rewrite the above

$$\begin{aligned} \alpha_k(y_k) &= \sum_{y_0 \ldots y_{k-1}} \exp\left[\sum_{i=0}^{k} g_i(y_{i-1}, y_i)\right] \\ &= \sum_{y_0 \ldots y_{k-1}} \exp\left[\sum_{i=0}^{k-1} g_i(y_{i-1}, y_i) + g_k(y_{k-1}, y_k)\right] \\ &= \sum_{y_{k-1}} \sum_{y_0 \ldots y_{k-2}} \exp \sum_{i=0}^{k-1} g_i(y_{i-1}, y_i) \\ &\cdot \exp g_k(y_{k-1}, y_k) \\ &= \sum_{\substack{y_{k-1} \in \text{prev}(y_k) \\ y_{k-1} \in r_{k-1}}} \alpha_{k-1}(y_{k-1}) \exp g_k(y_{k-1}, y_k) \end{aligned} \quad (9)$$

This recursion has the following basis:

$$\alpha_0 (y_0) = \exp g_0 (S, y_0)$$

In a similar manner, we define the unnormalized probability of an output sequence having tag $y_k \in r_k$ at position k and ending with tag $y_n = E$ as the backward vector:

$$\beta_k (y_k) = \sum_{\substack{y_{k+1} \in \text{next}(y_k) \\ y_{k+1} \in r_{k+1}}} \beta_{k+1} (y_{k+1}) \exp g_{k+1} (y_k, y_{k+1}) \tag{10}$$

The basis for the recursion of the backward vector is:

$$\beta_{n-1} (y_{n-1}) = \exp g_n (y_{n-1}, E)$$

We could move on defining the partition function $Z(x, w)$ and the output probabilities in terms of (9) and (10). These equations though suffer from arithmetic error propagation leading to underflow and overflow risk. Some implementations come around this difficulty by keeping the logarithm of α_k and β_k and re-exponiantiating for each addition in the formulae using the identity:

$$\log (\exp q + \exp r) = q + \log (1 + \exp (r - q))$$

This incurs a performance hit because of the train of log-exp calls. We will resort to a different approach instead. The contents of both α and β vectors are unnormalized probabilities, which means that we can scale them as we deem fit; they will be normalized anyway. We will adopt the scaling proposed by Rabiner for Hidden Markov Models in [4], incorporating the corrections pointed out by Rahimi [5].

We will produce a scaled version $\hat{\alpha}$ of the forward vector α such that:

$$\hat{\alpha}_k (u) = \frac{1}{\sum_{v \in r_k} \alpha_k (v)} \alpha_k (u) = C_k \alpha_k (u) \tag{11}$$

This is achieved by setting:

$$\begin{aligned} \bar{\alpha}_0 (y_0) &= \alpha_0 (y_0) = \exp g_0 (S, y_0) \\ c_k &= \frac{1}{\sum_{y_k \in r_k} \bar{\alpha}_k (y_k)} \\ \hat{\alpha}_k (y_k) &= c_k \bar{\alpha}_k (y_k) \\ \bar{\alpha}_{k+1} (y_{k+1}) &= \sum_{\substack{y_k \in \text{prev}(y_{k+1}) \\ y_k \in r_k}} \hat{\alpha}_k (y_k) \exp g_{k+1} (y_k, y_{k+1}) \end{aligned} \tag{12}$$

If at any k we find that $r_k = \oslash$, we stop as there is no possible output sequence having $k+1$ elements under the bigram constraints B.

The above scaling verifies (11) by induction. Indeed, the base case is satisfied, resulting to $C_0 = c_0$:

$$\begin{aligned} \bar{\alpha} (y_0) &= \alpha_0 (y_0) \\ \hat{\alpha}_0 (y_0) &= c_0 \alpha_0 (y_0) \\ &= \frac{1}{\sum_{u \in r_0} \alpha_0 (u)} \alpha_0 (y_0) \\ &= C_0 \alpha_0 (y_0) \end{aligned}$$

If we assume $\hat{\alpha}_k (y_k) = C_k \alpha_k (y_k)$, then we complete the induction:

$$\begin{aligned} \bar{\alpha}_{k+1} (y_{k+1}) &= C_k \sum_{y_k \in \text{prev}(y_{k+1}) \cap r_k} \alpha_k (y_k) \exp (y_k, y_{k+1}) \\ &= C_k \alpha_{k+1} (y_{k+1}) \\ c_{k+1} &= \frac{1}{\sum_{u \in r_{k+1}} \bar{\alpha}_{k+1} (u)} \\ &= \frac{1}{C_k \sum_{u \in r_{k+1}} \alpha_{k+1} (u)} \\ \hat{\alpha}_{k+1} (y_{k+1}) &= c_{k+1} \bar{\alpha}_{k+1} (y_{k+1}) \\ &= \frac{C_k \alpha_{k+1} (y_{k+1})}{C_k \sum_{u \in r_{k+1}} \alpha_{k+1} (u)} \\ &= C_{k+1} \alpha_{k+1} (y_{k+1}) \end{aligned} \tag{13}$$

From (13) and the definition in (11), we obtain a useful relationship for the scaling coefficients:

$$\begin{aligned} C_k &= \frac{1}{c_{k+1} \sum_{u \in r_{k+1}} \alpha_{k+1} (u)} \\ &= \frac{C_{k+1}}{c_{k+1}} \Rightarrow \\ \Rightarrow C_k &= C_{k-1} c_k = \prod_{i=0}^{k} c_i \end{aligned}$$

We also define the term D_k, which will be used to scale the backward vector β:

$$D_k = \prod_{i=k}^{n-1} c_i \tag{14}$$

$$C_k D_{k+1} = \prod_{i=0}^{k} c_i \prod_{i=k+1}^{n-1} c_i = C_{n-1} \tag{15}$$

Thus the scaled backward vector is set to be:

$$\hat{\beta}_k (y_k) = D_k \beta_k (y_k) \tag{16}$$

The required scaling is achieved as follows:

$$\begin{aligned} \bar{\beta}_{n-1} (y_{n-1}) &= \beta_{n-1} (y_{n-1}) \\ \hat{\beta}_k (y_k) &= c_k \bar{\beta}_k (y_k) \\ \bar{\beta}_k (y_k) &= \sum_{\substack{y_{k+1} \in \text{next}(y_k) \\ y_{k+1} \in r_{k+1}}} \hat{\beta}_{k+1} (y_{k+1}) \exp g_{k+1} (y_k, y_{k+1}) \end{aligned} \tag{17}$$

Again, we can show by induction that, by the above scaling, (16) is enforced. The basis of the induction is satisfied as $D_{n-1} = c_{n-1}$. Assuming $\hat{\beta}_{k+1}(y_{k+1}) = D_{k+1}\beta_{k+1}(y_{k+1})$, we complete the induction:

$$\begin{aligned}
\bar{\beta}_k(y_k) &= D_{k+1} \sum_{\substack{y_{k+1}\in \text{next}(y_k) \\ y_{k+1}\in r_{k+1}}} \beta_{k+1}(y_{k+1}) \exp g_{k+1}(y_k, y_{k+1}) \\
&= D_{k+1}\beta_k(y_k) \\
\hat{\beta}_k(y_k) &= c_k \bar{\beta}_k(y_k) \\
&= c_k D_{k+1}\beta_k(y_k) \\
&= D_k \beta_k(y_k)
\end{aligned}$$

We proceed now with expressing the probability of a bigram at a position in a feasible output sequence. Starting from the definition in (2) and substituting the g_i quantities introduced in (5), this probability is:

$$p(Y_k = u, Y_{k+1} = v|x; w) = \frac{\sum_{\substack{y'\in Y^p \\ y_k = u \\ y_{k+1}=v}} \exp \sum_{i=0}^{n} g_i\left(y'_{i-1}, y'_i\right)}{Z(x; w)} \tag{18}$$

Initially, we use the unscaled forward and backward vectors described in (9) and (10) to rewrite the above expression. For $0 \leq k \leq n-2$, the numerator becomes:

$$p(Y_k = u, Y_{k+1} = v|x; w) = \frac{\alpha_k(u) \exp g_{k+1}(u, v)\, \beta_{k+1}(v)}{Z(x; w)} \tag{19}$$

In order to cover the extreme cases $k = -1$ and $k = n-1$ under the same formulae, we define:

$$\alpha_{-1}(S) = 1 \qquad\qquad \beta_n(E) = 1$$

The denominator is $Z(x; w)$ defined in (3). We substitute g_i in it as well and we proceed:

$$\begin{aligned}
Z(x; w) &= \sum_{y'\in Y^p} \exp \sum_{i=0}^{n} g_i\left(y'_{i-1}, y'_i\right) \\
&= \sum_{y'_k \in r_k} \alpha_k(y'_k)\, \beta_k(y'_k) \\
&= \sum_{y'_0 \in r_0} \beta_0(y'_0) \exp g_0(S, y'_0) \\
&= \sum_{y'_{n-1}\in r_{n-1}} \alpha_{n-1}\left(y'_{n-1}\right) \exp g_n\left(y'_{n-1}, E\right)
\end{aligned} \tag{20}$$

Looking at the definition of scaling coefficients C_k in (11), D_k in (16) and using (15), we substitute in the above expression(20) for $Z(x; w)$:

$$\begin{aligned}
Z(x; w) &= \sum_{y'_k \in r_k} \alpha_k(y'_k)\, \beta_k(y'_k) \\
&= \frac{1}{C_k D_k} \sum_{y'_k \in r_k} \hat{\alpha}_k(y'_k)\, \hat{\beta}_k(y'_k) \\
&= \frac{1}{C_{k-1} c_k D_k} \sum_{y'_k \in r_k} \hat{\alpha}_k(y'_k)\, \hat{\beta}_k(y'_k) \\
&= \frac{1}{c_k C_{n-1}} \sum_{y'_k \in r_k} \hat{\alpha}_k(y'_k)\, \hat{\beta}_k(y'_k)
\end{aligned}$$

We define the rescaled partition function $\hat{Z}(x; w)$:

$$\begin{aligned}
\hat{Z}(x; w) &= C_{n-1} Z(x; w) \\
&= \frac{1}{c_k} \sum_{y'_k \in r_k} \hat{\alpha}_k(y'_k)\, \hat{\beta}_k(y'_k)
\end{aligned} \tag{21}$$

Moving on to the bigram probability (19), assuming $0 \leq k \leq n-2$ and using (12), (17) and (21), we get:

$$\begin{aligned}
p(Y_k = u, Y_{k+1} = v|x; w) &= \frac{\alpha_k(u) \exp g_{k+1}(u, v)\, \beta_{k+1}(v)}{Z(x; w)} \\
&= \frac{\hat{\alpha}_k(u) \exp g_{k+1}(u, v)\, \hat{\beta}_{k+1}(v)}{C_k D_{k+1} Z(x; w)} \\
&= \frac{\hat{\alpha}_k(u) \exp g_{k+1}(u, v)\, \hat{\beta}_{k+1}(v)}{C_{n-1} Z(x; w)} \\
&= \frac{\hat{\alpha}_k(u) \exp g_{k+1}(u, v)\, \hat{\beta}_{k+1}(v)}{\hat{Z}(x; w)}
\end{aligned} \tag{22}$$

Again, in order to extend the validity of the above relationship for $k = -1$ and $k = n-1$, we make the following assumptions:

$$\begin{aligned}
c_{-1} &= 1 & c_n &= 1 \\
C_n &= C_{n-1} & D_n &= 1
\end{aligned}$$

These assumptions lead to:

$$\hat{\alpha}_{-1}(S) = 1 \qquad\qquad \hat{\beta}_n(E) = 1$$

Now that we have the probability of a bigram at a position in a feasible output sequence $p(Y_k = u, Y_{k+1} = v|x; w)$ given by (22), we can compute the gradient of the log of the conditional probability $\nabla_w \log p(y|x; w)$ displayed in (8), focusing on its $E_{y'\sim p(y|x;w)}[F(x, y')]$ term:

$$\begin{aligned}
E_{y\sim p(y|x;w)}[F(x, y)] &= E_{y\sim p(y|x;w)}\left[\sum_{i=0}^{n} f(x, y_{i-1}, y_i, i)\right] \\
&= \sum_{i=0}^{n} E_{y\sim p(y|x;w)}[f(x, y_{i-1}, y_i, i)] \\
&= \sum_{i=0}^{n} E_{y_{i-1}, y_i}[f(x, y_{i-1}, y_i, i)]
\end{aligned}$$

$$= \sum_{i=0}^{n} \sum_{u \in r_{i-1}} \sum_{\substack{v \in r_i \\ v \in \text{next}(u)}} p\left(Y_{i-1} = u, Y_i = v\right) f\left(x, u, v, i\right)$$

$$= \frac{1}{\hat{Z}\left(x; w\right)} \sum_{i=0}^{n} \sum_{u \in r_{i-1}} \sum_{\substack{v \in r_i \\ v \in \text{next}(u)}} \hat{\alpha}_{k-1}\left(u\right) \exp g_k\left(u, v\right) \hat{\beta}_k\left(v\right) \cdot f\left(x, u, v, i\right) \tag{23}$$

From the implementation standpoint, the forward and backward vectors can be realized as arrays of dictionaries of real numbers by tags, `Dictionary<Y, Double>[*]`. Their computation is analogous to the one for $U_k\left(v\right)$ in (6).

By obtaining the gradient of the log of the conditional probability with respect to w in (8) through (21) and (23), we can perform training using an on-line method such as Stochastic Gradient Ascent or, since the log of (2) is concave with respect to parameters vector w, we can form the gradient of the joint conditional probability of all training samples and use any standard concave optimization technique.

V. Theoretical comparison

The standard F-LCCRF has $O\left(n\left|Y\right|^2\right)$ time and space complexity for both training and inference, neglecting the effect of the feature functions computation. For BC-LCCRF, observing (6), (12), (17) and (23), we conclude that all of them perform n steps, in each step k they enumerate the tags in r_k, and for each tag $u \in r_k$ they enumerate the previous or the next tags to u accoding to the bigram set B. If we could estimate $m = E_i\left[r_i\right]$ the expected number of possible output tags in a position, $b = \max\left(E_i\left[\left|\text{prev}\left(r_i\right)\right|\right], E_i\left[\left|\text{next}\left(r_i\right)\right|\right]\right)$ the expected number of bigrams stemming from a possible tag in a position, then we could sketch a rough estimate of the complexity as $O\left(nmb\right)$. In sparse domains, we expect $b \ll \left|Y\right|$ and, especially for shorter sequences, $m < \left|Y\right|$. Thus the performance gain is expected from replacing the cardinality of the full cartesian product $\left|Y^2\right|$ with mb.

In comparison, the domain-specific approach in [2], taking advantage of the constraint of the possible output tags only, seems to yield an $O\left(nm\left|Y\right|\right)$ complexity. On the other hand, the bigram approach in [3] exploits only bigram sparsity directly, yielding an $O\left(n\left|Y\right|b\right)$ complexity. The proposed scheme combines both speedups while remaining a general approach supporting the most generic possible feature functions. The price we pay is that, in contrast to [3] which in theory can infer outputs having bigrams outside the specified bigram set when unigram feature functions return a sufficiently strong signal, the proposed approach can never yield a bigram outside the specified set B. This may be a desirable behavior in some contexts, but in others it may pose a generalization penalty.

VI. Applications

The presented methodology was developed for part-of-speech tagging of ancient Greek texts, where the number of possible tags is 1400. The ordinary "full" Conditional Random Field simply didn't even complete processing a single sentence in reasonable time. Using the described method, training over 32000 sentences takes 6 hours on a standard PC running on an Intel i7 processor, despite having expensive feature functions relying, among others, on the dense scalar outputs of 12000 support vector machines (SVMs) per word, all running on complex generalized string kernels. Inference for an average sentence takes about half a second, which is an acceptable performance for interactive applications.

Despite having this specific mission though, the methodology is generic and not bound to a particular problem. For example, it can be used in speech recognition over larger vocabularies, special cases in bioinformatics or any sequence labeling task requiring a large number of output tags. For this reason, the C# implementation offered at https://github.com/grammophone/Grammophone.CRF relies heavily on language generics, working with any type of input (not just sequences), any type of output tags and arbitrary unigram and bigram feature functions.

References

[1] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655813

[2] J. Waszczuk, "Harnessing the CRF complexity with domain-specific constraints. the case of morphosyntactic tagging of a highly inflected language," in *Proceedings of COLING 2012*. Mumbai, India: The COLING 2012 Organizing Committee, December 2012, pp. 2789–2804. [Online]. Available: http://www.aclweb.org/anthology/C12-1170

[3] N. Sokolovska, T. Lavergne, O. Cappé, and F. Yvon, "Efficient learning of sparse conditional random fields for supervised sequence labelling," *CoRR*, vol. abs/0909.1308, 2009.

[4] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[5] A. Rahimi. An erratum for "a tutorial on hidden markov models and selected applications in speech recognition". [Online]. Available: http://alumni.media.mit.edu/~{}rahimi/rabiner/rabiner-errata/rabiner-errata.html