Efficient Algorithms for Exact Inference in Sequence Labeling SVMs

Alexander Bauer, Nico Görnitz, Franziska Biegler, Klaus-Robert Müller*, and Marius Kloft

***Abstract*—The task of structured output prediction deals with learning general functional dependencies between arbitrary input and output spaces. In this context two loss-sensitive formulations for maximum-margin training have been proposed in the literature, which are referred to as margin and slack rescaling, respectively. The latter is believed to be more accurate and easier to handle. Nevertheless, it is not popular due to the lack of known efficient inference algorithms; therefore, margin rescaling — which requires a similar type of inference as normal structured prediction — is the most often used approach. Focusing on the task of label sequence learning, we here define a general framework that can handle a large class of inference problems based on Hamming-like loss functions and the concept of decomposability of the underlying joint feature map. In particular, we present an efficient generic algorithm that can handle both rescaling approaches and is guaranteed to find an optimal solution in polynomial time.**

***Index Terms*—Inference, structured output, label sequence learning, structural support vector machines (SVM), hidden Markov SVM, margin rescaling, slack rescaling, gene finding, dynamic programming,.**

I. Introduction

WHILE many efficient and reliable algorithms have been designed for typical supervised tasks such as regression and classification, learning general functional dependencies between arbitrary input and output spaces is still a considerable challenge. The task of *structured output prediction* extends the well studied classification and regression tasks to a most general level, where the outputs can be complex structured objects like graphs, alignments, sequences, or sets [1]–[3]. One of the most difficult questions arising in this context is the one of how to efficiently learn the prediction function.

In this paper, we focus on a special task of structured output prediction called *label sequence learning* [4]–[7], which concerns the problem of predicting a state sequence for an observation sequence, where the states may encode any labeling information, e.g., an annotation or segmentation of the sequence. A proven methodology to this end consists in the *structured support vector machine* [1]–[3], [8]–[10] — a generalized type of support vector machine (SVM). While margin rescaling has been studied extensively since it can often be handled by a generalized version of *Viterbi's* algorithm [11]–[16], slack rescaling is conjectured to be more effective, generally providing a higher prediction accuracy. Up to date, the latter is, however, not popular due to the lack of efficient algorithms solving the corresponding inference task. Even margin rescaling is applicable only to a small fraction of existing learning problems, where the loss function can be decomposed in the same way as the feature map.

In this paper, we present a general framework for both rescaling approaches that can handle a very large class of inference problems, that is, the ones that are based on the *Hamming-type* loss functions and whose joint feature map fulfills a decomposability requirement. We develop a generic algorithm and prove that it always finds an optimal solution in time polynomial in the size of the input data, which is a considerable improvement over more straightforward algorithms that have exponential runtime.

The remainder of the paper is organized as follows: In Section II we introduce the task of structured output prediction on example of label sequence learning. Section III is concerned with corresponding inference tasks. Here, we derive a general representation for different inference problems leading to a generic formulation of algorithms that is independent of a specific learning task. In particular, we define the notion of decomposability of a joint feature map that characterizes a class of inference problems, which can be dealt with by means of dynamic programming. Finally, we show how such inference problems can be solved algorithmically. In order to assess the resulting behaviour of the presented algorithm w.r.t. the learning process, in Section IV, we perform experiments on synthetic data simulating the computational biology problem of gene finding by using a cutting-plane algorithm for training. In the last Section V we summarize the achieved results followed by a discussion.

II. Label Sequence Learning in a Nutshell

Structured Output Prediction is a task in the general context of supervised learning, where we aim at learning a functional relationship $f : X \to Y$ between an input space X and a discrete output space Y, based on a training sample of input-output pairs $(x_1, y_1), ..., (x_n, y_n) \in X \times Y$, independently drawn from some fixed but unknown probability distribution over $X \times Y$. For instance, the elements $y \in Y$ may be complex structured objects such as sequences, sets, trees, or graphs. A

A. Bauer is with Machine Learning Group, Technische Universität Berlin Marchstr. 23, Berlin, Germany.

N. Görnitz is with Machine Learning Group, Technische Universität Berlin Marchstr. 23, Berlin, Germany.

F. Biegler is with Machine Learning Group, Technische Universität Berlin Marchstr. 23, Berlin, Germany.

K.-R. Müller is with Machine Learning Group, Technische Universität Berlin, and with Department of Brain and Cognitive Engineering, Korea University, Anam-dong, Seongbuk-ku, Seoul 136-713, Korea.

M. Kloft is with Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012, USA, and with Memorial Sloan-Kettering Cancer Center, 415 E 68th street, New York.

The correspondence should be addressed to klaus-robert.mueller@tu-berlin.de.

common approach is to learn a *joint discriminant function* $F : X \times Y \to \mathbb{R}$ that assigns each pair (x, y) with a real number encoding a degree of compatibility between input x and output y. Hence, by maximizing the function F over all $y \in Y$, we can determine, for a given input x, the most compatible output $f_w(x)$, that is,

$$f_w(x) = \arg\max_{y \in Y} \; F_w(x, y). \tag{1}$$

Here F is assumed to be linear in a joint feature representation of inputs and outputs and parameterized by a vector w, i.e.,

$$F_w(x, y) = \langle w, \Psi(x, y) \rangle, \tag{2}$$

with Ψ being a vector-valued mapping called *joint feature map*, whose concrete form depends on the nature of the underlying learning task.

From an abstract level, structured output prediction is closely related to multi-task learning, i.e., when $Y = \{1, \ldots, K\}$ [17]. Multi-task learning can be understood as a specific instance of (2) by putting

$$\Psi(x, y) = \Phi(x) \otimes \Lambda(y), \tag{3}$$

where

$$\Lambda(y) = (\delta(y_1, y), \delta(y_2, y), ..., \delta(y_K, y))^\top \in \{0, 1\}^K, \tag{4}$$

and δ denotes the Kronecker delta function, yielding 1 if the arguments are equal and 0 otherwise. To classify an instance x in the multi-class setting, we simply use the prediction rule (1).

Although it seems to be a natural approach to view such structured prediction tasks as multi-class problems, a major limitation consists in that all classes are treated equally and similarities as well as inter-dependencies between the classes are not taken into account. As a remedy, a prevalent line of research [1] considers functions

$$\Delta : Y \times Y \to \mathbb{R}, \;\; (y, \hat{y}) \mapsto \Delta(y, \hat{y}) \tag{5}$$

satisfying $\Delta(y, y) = 0$ and $\Delta(y, \hat{y}) > 0$ for $y \neq \hat{y}$. Intuitively, $\Delta(y, \hat{y})$ quantifies the discrepancy or distance between two outputs y and $\hat{y}$.

A. Structural SVMs

In order to learn the prediction function f_w we can train a generalized type of support vector machine, the so-called *structural SVM* [2], [8]–[10]

$$\begin{aligned} &\min_{w,\, \xi \geqslant 0} \;\; \tfrac{1}{2}\|w\|^2 + \tfrac{C}{n} \textstyle\sum_{i=1}^n \xi_i \\ &s.t. \;\; \forall \hat{y} \in Y \backslash \{y_1\} : \langle w, \delta\Psi_1(\hat{y}) \rangle \geqslant 1 - \xi_1 \\ &\quad \vdots \\ &s.t. \;\; \forall \hat{y} \in Y \backslash \{y_n\} : \langle w, \delta\Psi_n(\hat{y}) \rangle \geqslant 1 - \xi_n \,, \end{aligned} \tag{6}$$

where, for notational convenience, we define $\delta\Psi_i(\hat{y}) = \Psi(x_i, y_i) - \Psi(x_i, \hat{y})$. As for the standard SVM, the constant $C > 0$ controls the trade-off between minimizing the sum of slacks ξ_i and the regularizer $\frac{1}{2}\|w\|$. In the optimal point we have

$$\xi_i^*(w) = \max\left\{0, \max_{\hat{y} \neq y_i} \{1 - \langle w, \delta\Psi_i(\hat{y}) \rangle\}\right\} \tag{7}$$

so that, equivalently, we may write

$$\min_w \;\; \frac{1}{2}\|w\|^2 + \frac{C}{n} \sum_{i=1}^n \max_{\hat{y} \neq y_i} \; l(\langle w, \delta\Psi_i(\hat{y}) \rangle) \tag{8}$$

where $l(t) = \max\{0, 1 - t\}$ denotes the *hinge loss*. This formulation reveals a major limitation of (6), that is, any two outputs $\hat{y}_1, \hat{y}_2 \in Y \backslash \{y_i\}$ violating the margin constraints to the same extent, *contribute by the same value to the objective function*, independently of their individual form and structure. This is very unintuitive, as margin violations should be penalized less severely for outputs that are very similar to the training output.

In this paper, we consider two approaches to overcome this problem, either *margin* or *slack rescaling* [1], [3], respectively, which both are based on including an additional loss term Δ into the optimization problem (6).

B. Slack Rescaling

We can encode more information about the structure of the outputs by rescaling the slack variables ξ_i according to the loss incurred in each of the linear constraints. This gives rise to the following optimization problem:

$$\begin{aligned} &\min_{w,\, \xi \geqslant 0} \;\; \tfrac{1}{2}\|w\|^2 + \tfrac{C}{n} \textstyle\sum_{i=1}^n \xi_i \\ &s.t. \;\; \forall \hat{y} \in Y \backslash \{y_1\} : \langle w, \delta\Psi_1(\hat{y}) \rangle \geqslant 1 - \tfrac{\xi_1}{\Delta(y_1, \hat{y})} \\ &\quad \vdots \\ &s.t. \;\; \forall \hat{y} \in Y \backslash \{y_n\} : \langle w, \delta\Psi_n(\hat{y}) \rangle \geqslant 1 - \tfrac{\xi_n}{\Delta(y_n, \hat{y})} \end{aligned} \tag{9}$$

The values of slack variables, for which the true output is already well separated before rescaling, remain unchanged, but all other slack variables are rescaled according to the corresponding loss value. This is illustrated in Fig. 1. We observe that the outputs y_2 and y_3 violate the margin to the same extent $\xi_{y_2}/\Delta(y, y_2) = \xi_{y_3}/\Delta(y, y_3)$, but, since their corresponding loss values are different, the output y_2

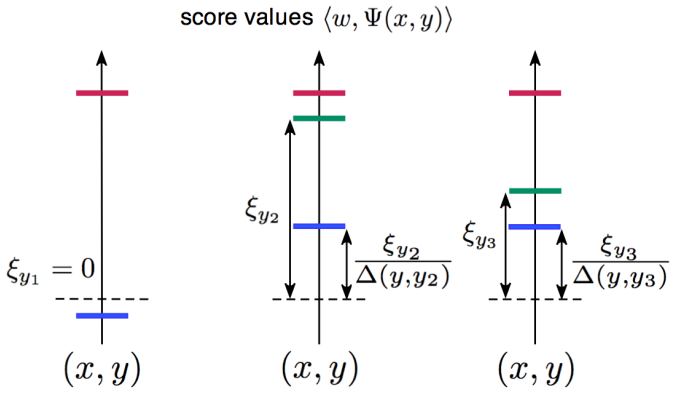


Fig. 1. Illustration of the behaviour of slack variables $\xi_i \geqslant 0$ due to the slack rescaling. Each vertical axis corresponds to a subset of constraints from (9) w.r.t. to the same data point (x, y). The score value of the true output y is marked by a red bar, while the blue bars mark the score values of three other outputs y_1, y_2 and y_3 with $\Delta(y, y_2) > \Delta(y, y_3)$. The distance between a red bar and the dotted line corresponds to the margin. The green bars illustrate the effect of rescaling on the corresponding slack variables.

contributes a greater value ξ_{y_2} to the objective function than the output y_3. For the optimal solutions of the slack variables for a given weight vector w we get:

$$\xi_i^*(w) = \max\left\{0, \max_{\hat{y}\neq y_i}\left\{\Delta(y_i,\hat{y})\cdot(1-\langle w, \delta\Psi_i(\hat{y})\rangle\right\}\right\} \quad (10)$$

with the sum $\frac{1}{n}\sum_{i=1}^n \xi_i^*$ being an upper bound on the empirical risk (see [1, p. 1458]).

C. Margin Rescaling

Another way to include more information about the structure of the outputs into the problem (6) is to rescale the margin according to incurred loss as follows:

$$\begin{array}{l} \min\limits_{w,\,\xi\geqslant 0} \ \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^n \xi_i \\ s.t.\ \forall \hat{y}\in Y\backslash\{y_1\} : \langle w, \delta\Psi_1(\hat{y})\rangle \geqslant \Delta(y_1,\hat{y}) - \xi_1 \\ \quad\vdots \\ s.t.\ \forall \hat{y}\in Y\backslash\{y_n\} : \langle w, \delta\Psi_n(\hat{y})\rangle \geqslant \Delta(y_n,\hat{y}) - \xi_n \end{array} \quad (11)$$

The behavior of the slack variables is illustrated in Fig. 2. We observer that, although the outputs y_2 and y_3 violate the margin by the same value $\xi_{y_2} = \xi_{y_3}$ (before rescaling), because of $\Delta(y,y_2) > \Delta(y,y_3)$, the output y_2 contributes a greater value $\Delta(y,y_2) - 1 + \xi_{y_2}$ to the objective function than the the output y_3 (which contributes $\Delta(y,y_3) - 1 + \xi_{y_3}$). Note that the output y_1 is already well separated from the true output, so there is no margin violation before and after the rescaling since $\Delta(y,y_1) \leqslant \langle w, \Psi(x,y) - \Psi(x,y_1)\rangle$, although the margin width has changed.

A potential disadvantage of the margin rescaling (when contrasted to slack rescaling) is that it seems to give more importance to labelings having large errors — even after they are well separated, sometimes at the expense of instances that are not even separated. For example, the labeling y_4 before rescaling has a score value that lies well below the margin line with $\xi_{y_4} = 0$. We thus cannot improve the separability of any other labeling of this instance having a smaller margin violation than y_4 since the optimal value $\xi_i^*(w)$ of the corresponding slack variable for a given weight vector w is determined by a single maximum violating labeling, where

$$\xi_i^*(w) = \max\left\{0, \max_{\hat{y}\neq y_i}\left\{\Delta(y_i,\hat{y}) - \langle w, \delta\Psi_i(\hat{y})\rangle\right\}\right\}. \quad (12)$$

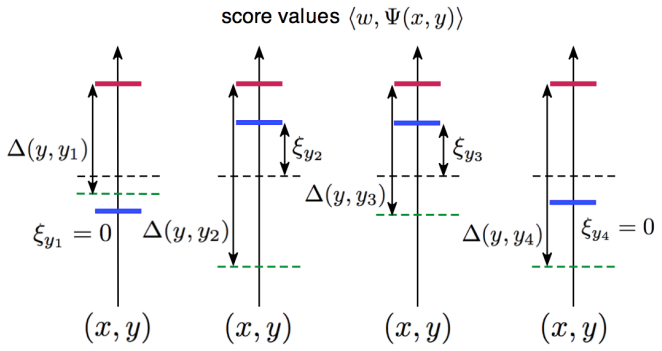


Fig. 2. Illustration of the behaviour of slack variables $\xi_i \geqslant 0$ due to the margin rescaling. Each vertical axis corresponds to the same subset of constraints from (11) w.r.t. a fixed data point (x,y). The score value of the true output y is marked by a red bar, while the blue bars mark the score values of some other outputs y_1, y_2, y_3 and y_4. The distance between a red bar and the black dotted line corresponds to the margin of the length 1 and the distance to the green dotted line is the rescaled margin being equal to the corresponding loss value. For the corresponding slack variables before rescaling we have: $\xi_{y_1} = 0$, $\xi_{y_2} = \xi_{y_3} > 0$, $\xi_{y_4} = 0$ with $\Delta(y_i,y_2) > \Delta(y_i,y_3)$.

D. Label Sequence Learning

Label sequence learning is an important instance of structured output prediction. Here, we aim at predicting a label sequence $y = (y_1, ..., y_T)$ for a given observation sequence $x = (x_1, ..., x_T)$ by using the functions f and F defined in (1) and (2). In order to simplify the presentation we assume that all sequences are of the same length. In particular, we define the input space to be $X = \mathbb{R}^{r\times T}$ with $r, T \in \mathbb{N}$ and the output space Y to consist of all possible label sequences over a finite alphabet Σ with $|\Sigma| = N \in \mathbb{N}$, i.e. $Y := \{(y_1, ..., y_T) \mid \forall t \in \{1, ..., T\} : y_t \in \Sigma\} = \Sigma^T$, where for notational convenience we denote the possible states for single sites $\hat{y}_i \in \Sigma$ in the sequence $\hat{y}$ by the numbers $1, ..., N$.

One possible choice of the discriminant function F for this task is based on the concept of hidden Markov models (HMMs), which restrict possible dependencies between states in a label sequence by treating it as a Markov chain. This results in a formulation, which can be addressed by dynamic programming enabling an efficient inference. Following this chain of reasoning we can specify a joint feature map Ψ including the interactions between input features $\Phi(x_t)$ (for some mapping Φ) and labels y_t as well as the interactions between nearby label variables y_{t-1}, y_t similar to the approach of HMMs (cf. [1, p. 1475]):

$$\Psi(x,y) = \begin{pmatrix} \Lambda(y_1) \\ \sum_{t=2}^T \Lambda(y_{t-1}) \otimes \Lambda(y_t) \\ \sum_{t=1}^T \Phi(x_t) \otimes \Lambda(y_t) \end{pmatrix} \quad (13)$$

E. Cutting-Plane Algorithm

In order to find an optimal solution w of the problems like (6), (9) and (11) we can use the following cutting-plane algorithm:

Cutting Plane Algorithm for training structural SVMs.

Input: training examples $(x_1, y_1), ..., (x_n, y_n)$
precision ϵ and regularization parameter C

Output: (w, ξ) : the optimal solution

1: $W \leftarrow \emptyset$, $w \leftarrow 0$, $\xi_i \leftarrow 0$ for all $i = 1, ..., n$
2: **repeat**
3: **for** $i = 1$ **to** n **do**
4: $y \leftarrow \arg\max\limits_{\hat{y}\in Y} H(\hat{y})$
5: **if** $H(y) > \xi_i + \epsilon$ **then**
6: $W \leftarrow W \cup \{y\}$
7: $(w, \xi) \leftarrow \min\limits_{w,\xi\geqslant 0} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^n \xi_i$ s.t. W
8: **end if**

9: **end for**
10: **until** W has not changed during iteration
11: **return** (w, ξ)

The above algorithm iterates through the training examples, where for each example a separation oracle (line 4) delivers a constraint that is most violated by the current solution (w, ξ), where

$$H(\hat{y}) := \begin{cases} 1 - \langle w, \delta\Psi_i(\hat{y})\rangle, & \text{no rescaling} \\ \Delta(y_i, \hat{y}) - \langle w, \delta\Psi_i(\hat{y})\rangle, & \text{margin rescaling} \\ (1 - \langle w, \delta\Psi_i(\hat{y})\rangle) \cdot \Delta(y_i, \hat{y}), & \text{slack rescaling} \end{cases} \quad (14)$$

If this constraint is violated by more than ϵ, we add it to the current set W and compute a new solution by using a QP solver over W (lines 5 - 8). The algorithm terminates if W did not change between iterations. It is provably efficient whenever the most violated constraint can be found efficiently. For correctness and complexity analysis see [1, p. 1464-1469]. The separation oracle is given in a generic form; in order to apply it to a concrete structured prediction problem, one needs to supply the functions

$$\Psi(x, y), \quad \Delta(y, \hat{y}), \quad \arg\max_{y \in Y} H(y).$$

In the next section we take a closer look on how to solve such a problem efficiently.

III. Inference Problems

In this section we consider a special class of optimization problems occurring in structured prediction tasks on example of label sequence learning, which are known as *inference* or *decoding* problems. We restrict our attention to two special forms of the general inference problem, which are referred here to as *linear* and *loss augmented inference*. For the case where the underlying problem has special factoring properties, we describe efficient algorithms, which find an optimal solution in polynomial time.

A. Linear Inference

As already mentioned above, in label sequence learning we aim at learning a prediction function of the form:

$$y^* = \arg\max_{\hat{y} \in Y} \langle w, \Psi(x, \hat{y})\rangle \quad (15)$$

Such optimization problems, where we aim at determining an optimal assignment $y^* \in Y$ given an input $x \in X$, are typically referred to as the *inference* or *decoding* problems. Here, we call the problems as in (15) *linear inference* problems indicating that the objective function is linear in the joint feature vector. Since the cardinality $|Y|$ of the output space grows exponentially in the description length of $\hat{y} \in Y$, the exhaustive searching is computational infeasible. However, combinatorial optimization problems with *optimal substructure*, i.e. such problems of which each optimal solution contains within it optimal solutions to the subproblems, can often be solved efficiently by dynamic programming [18, p. 359-397]. For the case of inference problems of the form (15) it exploits certain *factoring* properties of Ψ, which we define formally in a moment. For notational convenience we first introduce the following projections: for each $i \in \{1, ..., T\}$ and each $\ell \in \mathbb{N}$ we define a map

$$\pi_i^\ell(y) := (y_{\max\{i-\ell,1\}}, ..., y_{i-1}, y_i) \quad (16)$$

the *local projection* of order ℓ at position i. The main goal of these projection maps is to reduce the size of the formulas in order to keep the notation compact. So, for a given sequence y the map π_i^ℓ cuts out a consecutive subsequence of the length at most $\ell + 1$ consisting of term y_i and its ℓ predecessors if they exist.

Example 1. *For projection order $\ell \in \{0, 1, 2\}$ and position $i \in \{1, 2, ..., T\}$ we get:*

$$\begin{array}{lll} \pi_1^0(y) = y_1 & \pi_1^1(y) = y_1 & \pi_1^2(y) = y_1 \\ \pi_2^0(y) = y_2 & \pi_2^1(y) = (y_1, y_2) & \pi_2^2(y) = (y_1, y_2) \\ \pi_3^0(y) = y_3 & \pi_3^1(y) = (y_2, y_3) & \pi_3^2(y) = (y_1, y_2, y_3) \\ \vdots & \vdots & \vdots \\ \pi_T^0(y) = y_T & \pi_T^1(y) = (y_{T-1}, y_T) & \pi_T^2(y) = (y_{T-2}, y_{T-1}, y_T) \end{array}$$

Now, building on the notation of local projections, we define the term of the *decomposability* of a joint feature map.

Definition 1 (Decomposability of joint feature maps)**.** *We say a joint feature map*

$$\Psi : X \times Y \to \mathbb{R}^d, \ \ (x, y) \mapsto \Psi(x, y) \quad (17)$$

*for $d \in \mathbb{N}$ is **additively decomposable** of order $\ell \in \mathbb{N}$ if for each $i \in \{1, ..., T\}$ there is a map*

$$\psi_i : (x, \pi_i^\ell(y)) \mapsto \psi_i(x, \pi_i^\ell(y)) \in \mathbb{R}^d \quad (18)$$

such that for all x, y the following equality holds:

$$\Psi(x, y) = \sum_{i=1}^{T} \psi_i(x, \pi_i^\ell(y)). \quad (19)$$

Next we give a short example in order to clarify the above definition.

Example 2. *Consider the joint feature map in (13). It is additively decomposable of order $\ell = 1$, since we can write*

$$\Psi(x, y) = \begin{pmatrix} \Lambda(y_1) \\ \sum_{i=2}^T \Lambda(y_{i-1}) \otimes \Lambda(y_i) \\ \sum_{i=1}^T \Phi(x_i) \otimes \Lambda(y_i) \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} \Lambda(y_1) \\ 0 \\ \Phi(x_1) \otimes \Lambda(y_1) \end{pmatrix}}_{=:\psi_1(x, y_1) = \psi_1(x, \pi_1^1(y))} + \sum_{i=2}^T \underbrace{\begin{pmatrix} 0 \\ \Lambda(y_{i-1}) \otimes \Lambda(y_i) \\ \Phi(x_i) \otimes \Lambda(y_i) \end{pmatrix}}_{=:\psi_i(x, y_{i-1}, y_i) = \psi_i(x, \pi_i^1(y))}$$

$$= \sum_{i=1}^T \psi_i(x, \pi_i^1(y))$$

So, if a joint feature map is additively decomposable of order ℓ, it can be written as a sum of vectors, where each vector depends only "locally" on the sequence y in the sense

of the map $\pi_i^\ell(y)$. For such a Ψ we can reformulate the right-hand side of the inference problem (15) as follows:

$$\begin{aligned}\langle w, \Psi(x,\hat{y})\rangle &= \langle w, \textstyle\sum_{i=1}^T \psi_i(x, \pi_i^\ell(\hat{y}))\rangle \\ &= \textstyle\sum_{i=1}^T \underbrace{\langle w, \psi_i(x,\pi_i^\ell(\hat{y}))\rangle}_{=:\ g_i(\pi_i^\ell(\hat{y}))} = \sum_{i=1}^T g_i(\pi_i^\ell(\hat{y}))\end{aligned}$$

for some real-valued functions g_i. Inference problems having a general representation

$$y^* = \operatorname*{arg\,max}_{\hat{y}\in Y} \ \sum_{i=1}^T g_i(\pi_i^\ell(\hat{y})) \tag{20}$$

can be solved via dynamic programming in $O(T\cdot N^2\cdot \ell)$ time, similar to the well known Viterbi's algorithm for hidden Markov models. To do so we stepwise compute quantities $V_t(j)$ for all $1\leqslant t\leqslant T$ and $1\leqslant j\leqslant N$ according to the following equation

$$V_t(j) := \max_{\hat{y}=(\hat{y}_1,\ldots,\hat{y}_{t-1},j)} \ \sum_{i=1}^t g_i(\pi_i^\ell(\hat{y})) \tag{21}$$

We interpret $V_t(j)$ as the optimal value of a subproblem of (20), where $t\leqslant T$ and the last state in each solution is fixed being j. Furthermore, we save the preceding state of the last state j in a structure S as follows:

$$\forall t\in\{2,\ldots,T\}\ \forall j\in\{1,\ldots,N\}: \quad S_t(j) := y_{t-1}^{t,j},$$

where

$$y^{t,j} = (y_1^{t,j},\ldots,y_t^{t,j}) := \operatorname*{arg\,max}_{\hat{y}=(\hat{y}_1,\ldots,\hat{y}_{t-1},j)} \ \textstyle\sum_{i=1}^t g_i(\pi_i^\ell(\hat{y})).$$

If the solution $y^{t,j}$ is not unique, so that there are several predecessors of the last state j, we choose one having the smallest value for $y_{t-1}^{t,j}$ as a tie-breaker. In this notation the state y_T^* of the last site in y^* is given by

$$y_T^* = \operatorname*{arg\,max}_{1\leqslant i\leqslant N}\{V_T(i)\}$$

and the remaining sites can be computed in a top-down manner by

$$y_t^* = S_{t+1}(y_{t+1}^*), \qquad t = T-1,\ldots,1.$$

The pseudocode of the above algorithm is given in the appendix A. Since we are initially interested in solving the problem (15), we have to take the time needed to compute the values $g_t(i,j)$ into account. However, the good news is that usually we will deploy a mask algorithm in an outer loop to solve (15) (e.g. a cutting-plane algorithm) so that we need to compute the values $g_t(i,j)$ only once at the beginning (as a preprocessing step) and therefore can access the cached values $g_t(i,j)$ in subsequent iterations. So that, we can often assume the time needed to compute $g_t(i,j)$ to be constant $O(1)$.

B. Loss Augmented Inference

Loss augmented inference is similar to the linear inference problems, but the objective function is extended by a loss mapping Δ as follows:

$$y^* = \operatorname*{arg\,max}_{\hat{y}\in Y} \ \langle w, \Psi(x,\hat{y})\rangle \odot \Delta(y,\hat{y}), \quad \odot\in\{+,\cdot\} \tag{22}$$

where y is a fixed (true) output for a given input x. In order to train a structural SVM with margin or slack rescaling using the cutting-plane algorithm we have to solve an appropriate loss augmented inference problem. For the general case of an arbitrary loss function Δ it seems that we cannot do better than exhaustive search even if the joint feature map is decomposable. However, for the special case of the *Hamming loss*, which can be decomposed additively over the sites of the sequences, we can develop algorithms, which find an optimal solution for such inference problems in polynomial time. More precisely, we consider here loss functions of the form $\Delta = h\circ\Delta_H$, where $h:\mathbb{R}\to\mathbb{R}$ is an arbitrary function and Δ_H denotes the Hamming loss, which is defined as

$$\Delta_H : Y\times Y\to\mathbb{N}, \quad (y,\hat{y})\mapsto \sum_{i=1}^T \mathbb{1}_{[y_i\neq\hat{y}_i]}, \tag{23}$$

where $\mathbb{1}_{[y_i\neq\hat{y}_i]}$ is an indicator function yielding 1 if the expression $[y_i\neq\hat{y}_i]$ is true and 0 otherwise. Furthermore, we assume the joint feature map Ψ to be additively decomposable of order ℓ, i.e. there are real-valued functions $g_1,\ldots,g_T$ with $\langle w,\Psi(x,\hat{y})\rangle = \sum_{i=1}^T g_i(\pi_i^\ell(\hat{y}))$. Using this terminology, (22) writes

$$y^* = \operatorname*{arg\,max}_{\hat{y}\in Y} \ \left[\sum_{i=1}^T g_i(\pi_i^\ell(\hat{y}))\right] \odot h(\Delta_H(y,\hat{y})), \ \ \odot\in\{+,\cdot\} \tag{24}$$

In the following we show how to solve inference problems in the general form (24) algorithmically. One important observation here is that the Hamming loss only takes on a **finite** number of non-negative integer values so that we can define the following term:

$$L(t,j,k) := \sup_{\hat{y}=(\hat{y}_1,\ldots,\hat{y}_{t-1},j)\,:\,\Delta_H(y,\hat{y})=k} \ \sum_{i=1}^t g_i(\pi_i^\ell(\hat{y})) \tag{25}$$

We interpret $L(t,j,k)$ as a corresponding sum-value of an optimal solution of a subproblem of (24), where $t\in\{1,2,\ldots,T\}$ denotes the length of the sequence $\hat{y}$, $j\in\{1,2,\ldots,N\}$ is the last state in this sequence, and $k\in\{0,1,\ldots,T\}$ is the value of the Hamming loss $\Delta_H(y,\hat{y})$ with y being reduced to the length of $\hat{y}$, i.e. $y=(y_1,\ldots,y_t)$. For the cases

$$t<k \quad \text{or} \quad t=k, j=y_t \quad \text{or} \quad k=0, j\neq y_t$$

the set $\{\hat{y}=(\hat{y}_1,\ldots,\hat{y}_{t-1},j):\Delta_H(y,\hat{y})=k\}$ is empty and the sum-value of the corresponding subproblem is $L(t,j,k) = \sup\emptyset = -\infty$. In this notation the following holds:

$$\begin{aligned}&\max_{1\leqslant j\leqslant N,\ 0\leqslant k\leqslant T} L(T,j,k)\odot h(k) = \\ &\max_{\hat{y}\in Y} \ \left[\textstyle\sum_{i=1}^T g_i(\pi_i^\ell(\hat{y}))\right]\odot h(\Delta_H(y,\hat{y}))\end{aligned} \tag{26}$$

That is, if we know the optimal sum-values $L(T,j,k)$ of the subproblems of (24) for all $1\leqslant j\leqslant N$ and $0\leqslant k\leqslant T$, we can determine the optimal value of the base problem (24) too, by solving (26). The next theorem shows how to compute the optimal sum-values $L(t,j,k)$ of the subproblems recursively. For simplicity, we first restrict our attention to the case $\ell=1$, i.e. Ψ is additively decomposable of order 1.

Theorem 1. *Let $\ell = 1$, $t \in \{2,3,...,T\}$, $j \in \{1,2,...,N\}$, $k \in \{0,1,...,T\}$ and a fixed sequence $y = (y_1,...,y_T) \in Y$ be given. Then the following equality holds:*

$$L(t,j,k) = \begin{cases} \sup\limits_{1 \leqslant i \leqslant N} L(t-1,i,k-1) + g_t(i,j), & j \neq y_t, k > 1, t \geqslant k \\ \sum_{i=1}^{t-1} g_i(\pi_i^1(y)) + g_t(y_{t-1},j), & j \neq y_t, k = 1, t \geqslant k \\ \sup\limits_{1 \leqslant i \leqslant N} L(t-1,i,k) + g_t(i,j), & j = y_t, t > k \\ -\infty, & \textit{otherwise} \end{cases}$$

A proof of the above theorem is given in appendix B. Finally, we can solve the inference problem (24) algorithmically as follows. After the initialization part, we compute in a bottom-up manner (i.e. all combinations $(t,\cdot,\cdot)$ before $(t+1,\cdot,\cdot)$) the values $L(t,j,k)$ using the appropriate formula in the Theorem 1, so that we can determine the optimal value L^* according to equation (26). Simultaneously, we store the information about the predecessor of the last state $\hat{y}_t = j$ of the sequence $\hat{y}$ (in a current iteration) in the variable $S_{t,k}(j)$, which we use later to compute a corresponding optimal solution, i.e.

$$\forall t \in \{2,...,T\}\ \forall k \in \{0,...,t\}\ \forall j \in \{1,...,N\} : S_{t,k}(j) := y_{t-1}^{t,j,k}$$

where

$$(y_1^{t,j,k},...,y_t^{t,j,k}) := \underset{\hat{y}=(\hat{y}_1,...,\hat{y}_{t-1},j):\ \Delta_H(y,\hat{y})=k}{\arg\max} \sum_{i=1}^{t} g_i(\pi_i^\ell(\hat{y}))$$

if the set $\{\hat{y} = (\hat{y}_1,...,\hat{y}_{t-1},j) : \Delta_H(y,\hat{y}) = k\}$ is not empty. If the solution $y^{t,j,k}$ is not unique so that there are several potential predecessors of the last state j, we can choose any of them. Here, we take as a tie-breaker the state having the smallest value for $y_{t-1}^{t,j,k}$. In this notation the state y_T^* of the last site in y^* is given by

$$y_T^* = \underset{1 \leqslant j \leqslant N}{\arg\max} \left\{ \max_{1 \leqslant k \leqslant T} L(T,j,k) \right\}$$

and the states of the remaining sites can be computed in a top-down manner by $y_t^* = S_{t+1,K}(y_{t+1}^*)$ for $t = T-1,...,1$, where $K = \Delta_H((y_1,...,y_{t+1}),(y_1^*,...,y_{t+1}^*))$. This way we obtain an efficient algorithm for solving optimization problem (24), which is summarized in Algorithm Table 1 (for $\ell = 1$).

Figure 3 presents a simulation of the above algorithm for the case "$\odot = \cdot$", $T = 4$, $N = 3$ and $y = (3,2,1,3)$ with h being the identity. It is illustrated, how the variables $L(t,j,k)$ and $S_{t,k}(j)$ evolve over the iterations $t = 1,...,T$. Each block corresponding to a value $t \in \{1,2,3,4\}$ in the recursion part represents one iteration step of the outer **for**-loop in lines 5-12 and depicts the content of the variable S represented by the edges between nodes with red edges marking the decision in the current iteration step: the nodes $(i,t-1)$ and (j,t) belonging to the same loss value $k \in \{1,2,3,4\}$ are connected by an edge if and only if $S_{t,k}(j) = i$ [1]. For example, after the second iteration (t = 2) we have : $S_{2,1}(1) = 3$, $S_{2,1}(2) = 1$, $S_{2,1}(3) = 3$, $S_{2,2}(1) = 1$ and $S_{2,2}(3) = 2$. The corresponding values $g_t(i,j)$ represented by a matrix are shown on the right:

[1] We omitted the value $k = 0$ in Fig. 3 for simplicity, since for that case the algorithm constructs only one path reproducing the true sequence y.

Algorithm 1 (Loss Augmented Viterbi) Find an optimal solution y^* of the problem (24) for $\ell = 1$

Input: $g_t(\pi_t^\ell(\hat{y}))$ for all $t \in \{1,...,T\}$ and $\hat{y} \in \{1,...,N\}^T$
true output sequence $y = (y_1,...,y_T)$
Output: y^* : an output sequence with highest value
L^* : the corresponding value of y^*

{Initialization: set the values $L(1,\cdot,\cdot)$ according to the definition (25)}

1: **for** $j = 1$ **to** N **do**
2: $L(1,j,0) \leftarrow \begin{cases} g_1(j), & y_1 = j \\ -\infty, & y_1 \neq j \end{cases}$
3: $L(1,j,1) \leftarrow \begin{cases} g_1(j), & y_1 \neq j \\ -\infty, & y_1 = j \end{cases}$
4: **end for**

{Recursion: compute in a bottom-up manner the values $L(t,j,k)$ and store the predecessors i of the last state j in $S_{t,k}(j)$}

5: **for** $t = 2$ **to** T **do**
6: **for** $j = 1$ **to** N **do**
7: **for** $k = 0$ **to** t **do**
8: compute the value $L(t,j,k)$ using Theorem 1
9: save the corresponding predecessor of j in $S_{t,k}(j)$
10: **end for**
11: **end for**
12: **end for**

{Termination: determine the last site y_T^* of an optimal sequence y^* and the corresponding optimal value L^*}

13: $L^* \leftarrow \max\limits_{1 \leqslant j \leqslant N, 1 \leqslant k \leqslant T} L(T,j,k) \odot h(k)$
14: $K \leftarrow \underset{1 \leqslant k \leqslant T}{\arg\max} \left\{ \max\limits_{1 \leqslant j \leqslant N} L(T,j,k) \odot h(k) \right\}$
15: $y_T^* \leftarrow \underset{1 \leqslant j \leqslant N}{\arg\max}\ L(T,j,K)$

{Back-Tracking: reconstruct the remaining sites y_t^* from y_T^*, S and K}

16: **for** $t = T-1$ **to** 1 **do**
17: $y_t^* \leftarrow S_{t+1,K}(y_{t+1}^*)$
18: **if** $y_{t+1}^* \neq y_{t+1}$ **then**
19: **if** $K = 1$ **then**
20: $(y_1^*,...,y_{t-1}^*) = (y_1,...,y_{t-1})$
21: **return** y^*, L^*
22: **else**
23: $K \leftarrow K - 1$
24: **end if**
25: **end if**
26: **end for**
27: **return** y^*, L^*

for example $g_2(1,2) = 5$ and $g_2(2,1) = 2$. The circled numbers represent the current values $L(t,j,k)$, where the values of the missing circles are set to $-\infty$.

After the initialization part we have a situation, where $L(1,1,1) = 2$, $L(1,2,1) = 1$, and $L(1,j,k) = -\infty$ for the remaining combinations of j and k. In the second iteration of the outer **for**-loop (t = 2) we get the following values of the variables $L(t,j,k)$ and $S_{t,k}(j)$:

Case 1: $(t,j,k) = (2,1,1)$ i.e. $j \neq y_t$, $k = 1$ and $t \geqslant k$

$$L(2,1,1) = g_1(3) + g_2(3,1) = 3 + 1 = 4$$

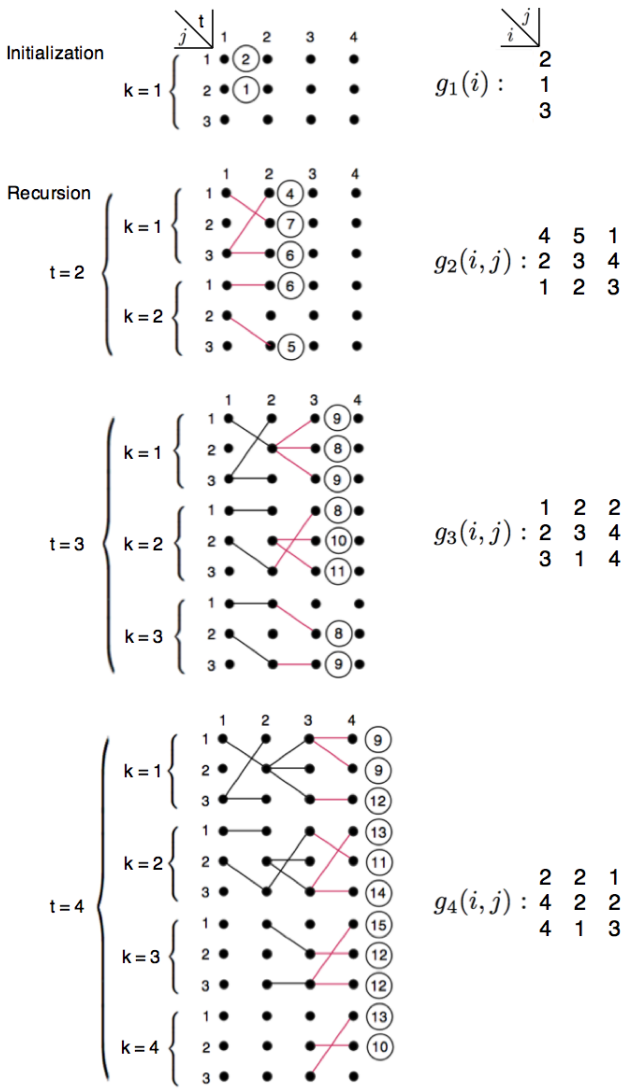


Fig. 3. This Fig. shows a simulation of the Algorithm 1 for the case "$\odot = \cdot$", $T = 4$, $N = 3$ and $y = (3, 2, 1, 3)$ with h being the identity. On the left the values $S_{t,k}(j)$ of the variable S are presented, where the edges between nodes denote the predecessor-relationship. The red edges mark the decision in the current iteration step and the circled numbers correspond to the values $L(t, j, k)$. On the right the values $g_t(i, j)$ are shown in a matrix form.

$S_{2,1}(1) = 3$

Case 2: $(t, j, k) = (2, 1, 2)$ i.e. $j \neq y_t$ and $k > 1$

$L(2, 1, 2) = \max_{1 \leqslant i \leqslant N} L(1, i, 1) + g_2(i, 1) =$
$L(1, 1, 1) + g_2(1, 1) = 2 + 4 = 6$

$S_{2,2}(1) = 1$

Case 3: $(t, j, k) = (2, 2, 1)$ i.e. $j = y_t$ and $t > k$

$L(2, 2, 1) = \max_{1 \leqslant i \leqslant N} L(1, i, 1) + g_2(i, 2) =$
$L(1, 1, 1) + g_2(1, 2) = 2 + 5 = 7$

$S_{2,1}(2) = 1$

Case 4: $(t, j, k) = (2, 2, 2)$ i.e. $j = y_t$ and $t = k$

$L(2, 2, 2) = -\infty$

Case 5: $(t, j, k) = (2, 3, 1)$ i.e. $j \neq y_t$ and $k = 1$

$L(2, 3, 1) = g_1(3) + g_2(3, 3) = 3 + 3 = 6$

$S_{2,1}(3) = 3$

Case 6: $(t, j, k) = (2, 3, 2)$ i.e. $j \neq y_t$ and $k > 1$

$L(2, 3, 2) = \max_{1 \leqslant i \leqslant N} L(1, i, 1) + g_2(i, 3) =$
$L(1, 2, 1) + g_2(2, 3) = 1 + 4 = 5$

$S_{2,2}(3) = 2$

The remaining iterations ($t = 3$ and $t = 4$) are handled in a similar way. The corresponding back-tracking part yielding an optimal solution $y^* = (2, 3, 3, 1)$ with the optimal sum-value $L^* = 13$ is illustrated in Fig. 4.

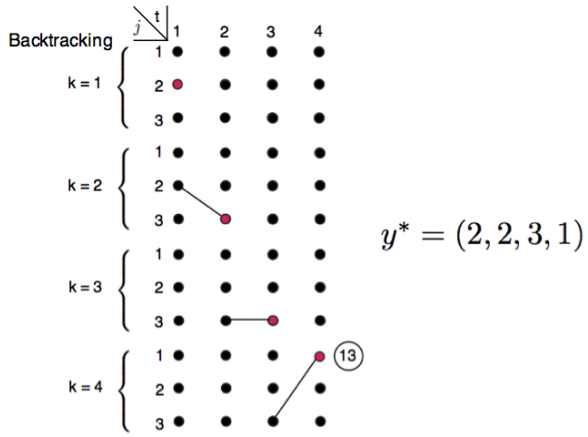


Fig. 4. This Fig. illustrates the back-tracking part of the simulation of the Algorithm 1 in the Fig. 3, where the red nodes mark an optimal solution $y^* = (2, 3, 3, 1)$. The corresponding sum-value is $L = 13$ yielding the optimal value $L^* = L \odot h(4) = 13 \cdot 4 = 52$.

Remark 1. *For the case $j \neq y_t, k = 1, t \geqslant k$ we can compute the value $L(t, j, 1)$ in a constant time without consuming additional memory space as follows. If $t = 2$ we set $L(t, j, 1) = g_1(y_1) + g_2(y_1, j)$ and if $t > 2$ we set $L(t, j, 1) = L(t-1, \tilde{j}, 1) - g_{t-1}(y_{t-2}, \tilde{j}) + g_{t-1}(y_{t-2}, y_{t-1}) + g_t(y_{t-1}, j)$, where $\tilde{j} \in \{1, ..., N\}$ is an arbitrary value with $\tilde{j} \neq y_{t-1}$.*

Equivalently, we could compute all partial sums $\sum_{i=1}^{t} g_i(\pi_i^\ell(y))$ for $t = 1, ..., T-1$ before the first iteration in $O(T)$ time and store them in an array A for further use. Since querying the values then takes a constant time, we can compute $L(t, j, 1)$ by $L(t, j, 1) = A(t-1) + g_t(y_{t-1}, j)$.

The run-time complexity of the algorithm above is dominated by the recursion part in lines 4 - 11 and is $O(T^2 \cdot N^2)$, since each computation inside the loops takes $O(N)$ time. The entire running time is therefore dominated by the term, that is quadratic in both the length of the sequence T and the number of possible states N resulting in the overall complexity $O(T^2 \cdot N^2)$. The correctness follows from the Theorem 1 and the equation (26). An extended variant of the above algorithm for the case $\ell \geqslant 1$ running in $O(T^2 \cdot N^2 \cdot \ell)$ time is discussed in the appendix C.

Finally, we can use the above results to solve the corresponding inference problems when training a structural SVM by either margin or slack rescaling.

Proposition 1. *Let $h : \mathbb{R} \to \mathbb{R}$ be an arbitrary function and $(x, y) \in X \times Y$ a given data pair. If the joint feature map Ψ is additively decomposable of order ℓ, then we can solve the corresponding inference problem (for $\Delta = h \circ \Delta_H$) resulting from either slack or margin rescaling by using Algorithm 1 in $O(T^2 \cdot N^2 \cdot \ell)$ time.*

Proof: During training via slack rescaling the corresponding separation oracle has to solve the following loss augmented inference problem

$$y^* = \operatorname*{arg\,max}_{\hat{y} \in Y} \ (1 - \langle w, \Psi(x, y) - \Psi(x, \hat{y})\rangle) \cdot \Delta(y, \hat{y}),$$

which can be written in more compact form as follows:

$$\begin{aligned} & (1 - \langle w, \Psi(x, y) - \Psi(x, \hat{y})\rangle) \cdot \Delta(y, \hat{y}) \\ = \ & (1 - \langle w, \Psi(x, y)\rangle + \langle w, \Psi(x, \hat{y})\rangle) \cdot \Delta(y, \hat{y}) \\ = \ & \langle \underbrace{\begin{pmatrix} w \\ 1 \end{pmatrix}}_{\tilde{w}}, \underbrace{\begin{pmatrix} \Psi(x, \hat{y}) \\ 1 - \langle w, \Psi(x, y)\rangle \end{pmatrix}}_{\tilde{\Psi}(x, \hat{y})} \rangle \cdot \Delta(y, \hat{y}) \\ = \ & \langle \tilde{w}, \tilde{\Psi}(x, \hat{y})\rangle \cdot \Delta(y, \hat{y}) \end{aligned}$$

In particular, we further can assume that $\tilde{\Psi}$ is additively decomposable of order ℓ: if there are maps ψ_i as in Definition 1, we easily can construct the corresponding maps $\tilde{\psi}_i$ by setting

$$\tilde{\Psi}(x, \hat{y}) = \begin{pmatrix} \Psi(x, \hat{y}) \\ 1 - \langle w, \Psi(x, y)\rangle \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^T \psi_i(x, \pi_i^\ell(\hat{y})) \\ 1 - \langle w, \Psi(x, y)\rangle \end{pmatrix} = \sum_{i=1}^T \underbrace{\begin{pmatrix} \psi_i(x, \pi_i^\ell(\hat{y})) \\ \frac{1 - \langle w, \Psi(x, y)\rangle}{T} \end{pmatrix}}_{\tilde{\psi}(x, \pi_i^\ell(\hat{y})) :=}$$

On the other hand, for margin rescaling we have to solve the following inference problem

$$y^* = \operatorname*{arg\,max}_{\hat{y} \in Y} \ \Delta(y, \hat{y}) - \langle w, \Psi(x, y) - \Psi(x, \hat{y})\rangle,$$

which is equivalent to

$$y^* = \operatorname*{arg\,max}_{\hat{y} \in Y} \ \Delta(y, \hat{y}) + \langle w, \Psi(x, \hat{y})\rangle$$

since $\langle w, \Psi(x, y)\rangle$ is constant over all $\hat{y} \in Y$ and has therefore no influence on the optimal value. So, for both scaling methods we can consider a more general problem instead:

$$y^* = \operatorname*{arg\,max}_{\hat{y} \in Y} \ \langle w, \Psi(x, \hat{y})\rangle \odot \Delta(y, \hat{y}), \quad \odot \in \{\cdot, +\}$$

which for $\Delta = h \circ \Delta_H$ can be solved by using Algorithm 1 in $O(T^2 \cdot N^2 \cdot \ell)$ time.

□

It is also worth to mention that for special case "$\odot = +$" if h is an affine function, we can always find an optimal solution of the problem (24) in linear time w.r.t. the length of the sequences. For that case we can even use more general loss functions instead of the Hamming loss provided this functions can be decomposed in the same fashion as the feature map.

Definition 2 (Decomposability of loss functions)**.** *We say a loss function*

$$\Delta : Y \times Y \to \mathbb{R}, \ \ (x, y) \mapsto \Delta(y, \hat{y}) \tag{27}$$

*is **additively decomposable** of order ℓ, if for each $i \in \{1, ..., T\}$, there is a map*

$$\Delta_i : (y, \pi_i^\ell(\hat{y})) \mapsto \Delta_i(y, \pi_i^\ell(\hat{y})) \in \mathbb{R} \tag{28}$$

such that for all $y, \hat{y}$ the following equality holds:

$$\Delta(y, \hat{y}) = \sum_{i=1}^T \Delta_i(y, \pi_i^\ell(\hat{y})). \tag{29}$$

An obvious example for such functions is the Hamming loss, where $\ell = 1$ and $\Delta_i(y, \pi_i^1(\hat{y})) = \mathbb{1}_{[y_i \neq \hat{y}_i]}$. A more general function, which cannot be handled by the Algorithm 1, is the so called *weighted* Hamming loss.

Example 3. *The weighted Hamming loss on segments of the length ℓ*

$$\Delta_{WH} : Y \times Y \to \mathbb{R}, \quad (y, \hat{y}) \mapsto \sum_{i=1}^T w_i(\pi_i^\ell(y), \pi_i^\ell(\hat{y})) \tag{30}$$

where for $i = 1, ..., T$

$$w_i : (\pi_i^\ell(y), \pi_i^\ell(\hat{y})) \mapsto w_i(\pi_i^\ell(y), \pi_i^\ell(\hat{y}))$$

are arbitrary functions, obviously is additively decomposable of order ℓ.

Using the above definition we can now formulate the following statement.

Proposition 2. *Let $h : \mathbb{R} \to \mathbb{R}$ be an arbitrary affine function and $(x, y) \in X \times Y$ a given data pair. If the joint feature map Ψ as well as the loss function Δ both are additively decomposable of orders ℓ_1 and ℓ_2 respectively, then we can solve the problem*

$$y^* = \operatorname*{arg\,max}_{\hat{y} \in Y} \ \langle w, \Psi(x, \hat{y})\rangle + h(\Delta(y, \hat{y})) \tag{31}$$

in $O(T \cdot N^2 \cdot \max\{\ell_1, \ell_2\})$ time by using the generalized Viterbi's algorithm.

Proof: We note that since h is an affine function, there are $a, b \in \mathbb{R}$ so that $h(x) = a \cdot x + b$. It suffices to show that we can write the problem (31) in the form (20) as follows:

$$\begin{aligned} & \langle w, \Psi(x, \hat{y})\rangle + h(\Delta(y, \hat{y})) \\ \overset{(*)}{=} \ & \langle w, \sum_{j=1}^T \psi_j(x, \pi_j^{\ell_1}(\hat{y}))\rangle + a \cdot \sum_{i=1}^T \Delta_i(y, \pi_i^{\ell_2}(\hat{y})) + b \\ = \ & \sum_{i=1}^T \underbrace{\langle w, \psi_i(x, \pi_i^{\ell_1}(\hat{y}))\rangle + a \cdot \Delta_i(y, \pi_i^{\ell_2}(\hat{y}))}_{=: g_i(\pi_i^{\max\{\ell_1, \ell_2\}}(\hat{y}))} + b \\ = \ & \sum_{i=1}^T g_i(\pi_i^{\max(\ell_1, \ell_2)}(\hat{y})) + b \end{aligned}$$

where in $(*)$ we used the corresponding assumption about the decomposability of Ψ and Δ. Since b is a constant, i.e. it has

no influence on the optimal value, we can use the generalized Viterbi's algorithm (see appendix A) to find an optimal solution of the corresponding problem in $O(T \cdot N^2 \cdot \max\{\ell_1, \ell_2\})$ time.

□

IV. Experiments and Empirical Analysis

Let us first take a moment to look at the inference problems (24) independently of the structured output learning. To get an idea how the presented algorithms consume the computational time in practice we show the corresponding running time as a function of the sequence length in Fig. 5. As one can see, for the sequence length $T = 2000$ it takes about one minute to compute a maximum violated constraint via Algorithm 1 restricting the number of effectively handleable training points to a few hundreds examples.

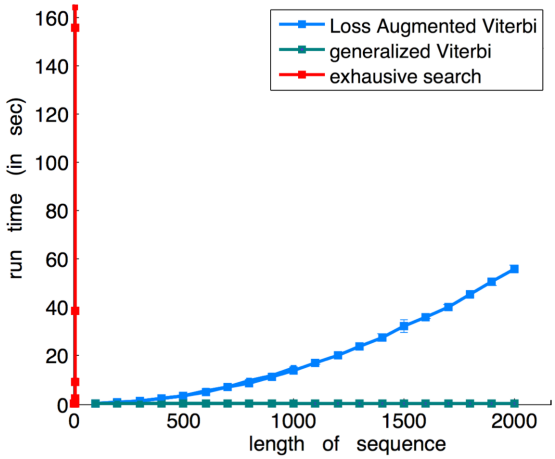


Fig. 5. Illustration of the running time in seconds of Loss Augmented Viterbi's algorithm (for slack or margin rescaling), generalized Viterbi's algorithm (for margin rescaling) and a brute force algorithm (for both scaling methods) via exhaustive searching in dependence on the length of the involved sequences. The exhaustive searching requires already for the short sequences of the length $T = 10$ about 160 seconds, while the computation time of the generalized Viterbi's algorithm for sequence length $T = 2000$ is less than 0.2 seconds.

Furthermore, we note the difference between the slack and margin rescaling due to the fact that the corresponding loss function contributes differently to the objective value. It is illustrated in Fig. 6 how the objective values are distributed over the length of the involved sequences and the values of the corresponding Hamming loss w.r.t. the true output. As one could expect, the objective value of both scaling methods is growing with increased sequence length and the Hamming loss. The difference here is that the highest objective values for slack rescaling (due to the multiplicative contribution of the loss) are more concentrated in the right lower corner of the heat map corresponding to longest sequences with biggest Hamming loss. For margin rescaling, where the loss contributes additively, the distribution of corresponding objective values is more smooth. Also the range of taken values differs significantly: for margin rescaling the values vary between 0 and 10^5 and for slack rescaling between 0 and 10^8 indicating that the objective value for the latter grows much faster with increase in the sequence length.

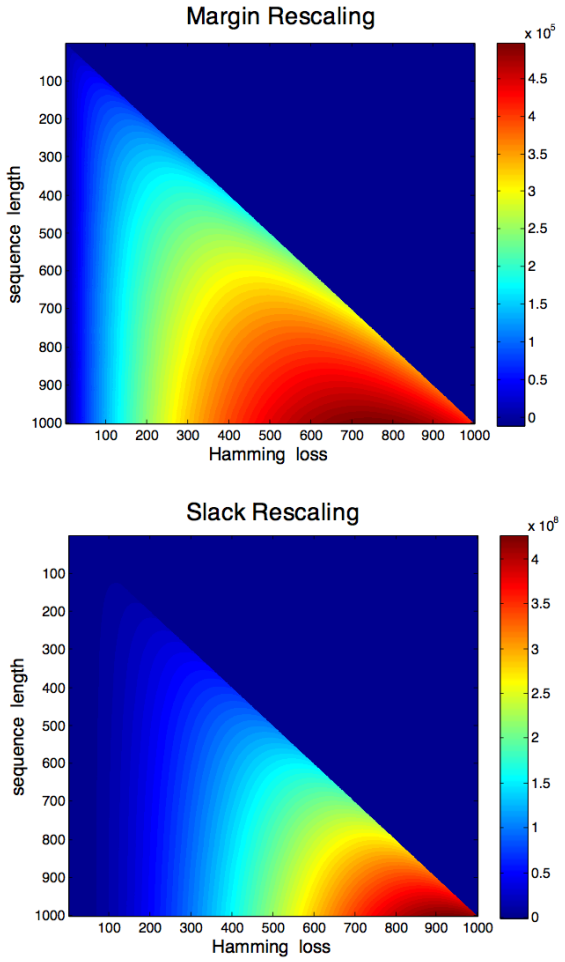


Fig. 6. Illustration of the distribution of the highest objective values of the problem (24) for margin ("$\odot = +$") and slack rescaling ("$\odot = \cdot$") in dependence on the sequence length (the vertical axis) and the Hamming loss w.r.t the true output (the horizontal axis). The pictures were constructed by averaging over 10 randomly generated data sets, where the values $g_t(i, j)$ were drawn uniformly at random from an interval $[-50, 50]$ with $N = 4$ and h being identity.

In the following we present the empirical results of experiments on synthetic data simulating the computational biology problem of gene finding. In particular, since we know the underlying distribution generating the data, we investigate the effects of presented algorithm in dependence on different problem settings by using a cutting-plane algorithm for training.

A. Data generation

We generate artificial data sets with $X = \mathbb{R}^{r \times T}$ and $Y = \{1, -1\}^T$, where $r = 4$ is the number of observation features and $T = 100$ is the length of the sequences. Each data point $(x, y) \in X \times Y$ was constructed in the following way. We first initialize a label sequence $y \in Y$ with entries -1 and then introduce positive blocks, where we choose the number, the position and the length of the latter uniformly at random from a finite set of values: minimal and maximal number of positive blocks are 0 and 4; minimal and maximal length of positive blocks are 10 and 40 respectively. Furthermore, the example sequences generated in this way are extended by start

and end states resulting in a data model represented by a state diagram in Fig. 7 on the right. The corresponding observation sequences were generated by adding a gaussian noise with zero mean and standard deviation $\sigma = 1, 2, 5, 7, 10, 12$ to the label sequence.

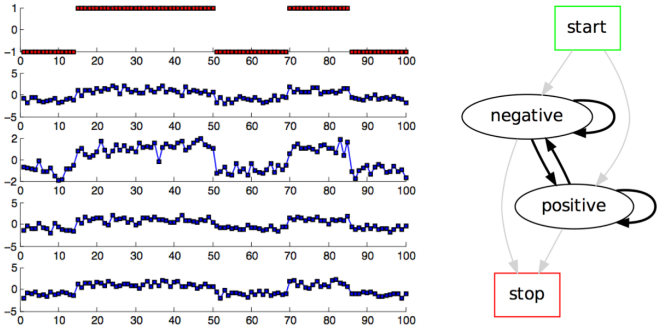


Fig. 7. Illustration of data model for synthetic experiment. The graphic on the left shows a visualization of an example from the synthetic data. The first row (red) represents the label sequence consisting of two positive $(+1)$ and three negative (-1) blocks. The remaining rows (blue) are the corresponding observation sequences generated by adding a gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 1$ to the label sequence. The graph on the right provides a corresponding state diagram describing the data model. There can be arbitrary many positive and negative states in a label sequence, while the first site always corresponds to the start and the last site to the end state.

B. Experimental Setting

We performed a number of experiments by training via cutting-plane algorithm on different data sets generated by varying the standard deviation σ of the gaussian noise. As a joint feature map we used one similar to that in (13), where we subdivided the continuous signals of observation sequences by additionally using histogram features with 10 bins. For each value of σ we first generated 10 different data sets each consisting of 3000 examples, which we then partitioned in training, validation and test sets of equal size (1000 examples in each set). During the training phase we also changed systematic the size of the training set by monotonically increasing the number of the training examples. Soft margin hyperparameter C was tuned by grid search over a range of values $\{10^i \mid i = -8, -7, ..., 6\}$ on the validation set (optimal C values are attained in the interior of the grid).

C. Results

The two upper graphics 8a and 8b in Fig. 8 present the averaged accuracy (over 10 different data sets) of the slack and margin rescaling in dependence on the number of training examples and on the standard deviation value σ of the noise. The both scaling methods show a consistent behavior in the sense that the accuracy of a prediction increases with the growing number of training examples and decreases with increasing amount of noise. The graphics 8c and 8d present the averaged number of iterations of cutting-plane algorithm as a function of the number of training examples. Here, we can make two observations holding for both scaling approaches. First, the number of iterations obviously decreases with an increasing number of training examples. And second, the number of iterations seems to increase with increasing amount of noise in the data. Finally, we can see that on average margin rescaling needs less iterations than slack rescaling to converge. The total training time of both approaches is given in the graphics 8e and 8f.

D. Interpretation

In our synthetic example, clearly slack and margin rescaling yield approximately the same performance. The slack rescaling still provides a higher prediction accuracy than margin rescaling, although the performance difference is insignificant. More precisely, the difference of averaged accuracies (in %) for different values σ between slack and margin rescaling (by training on 500 examples) is

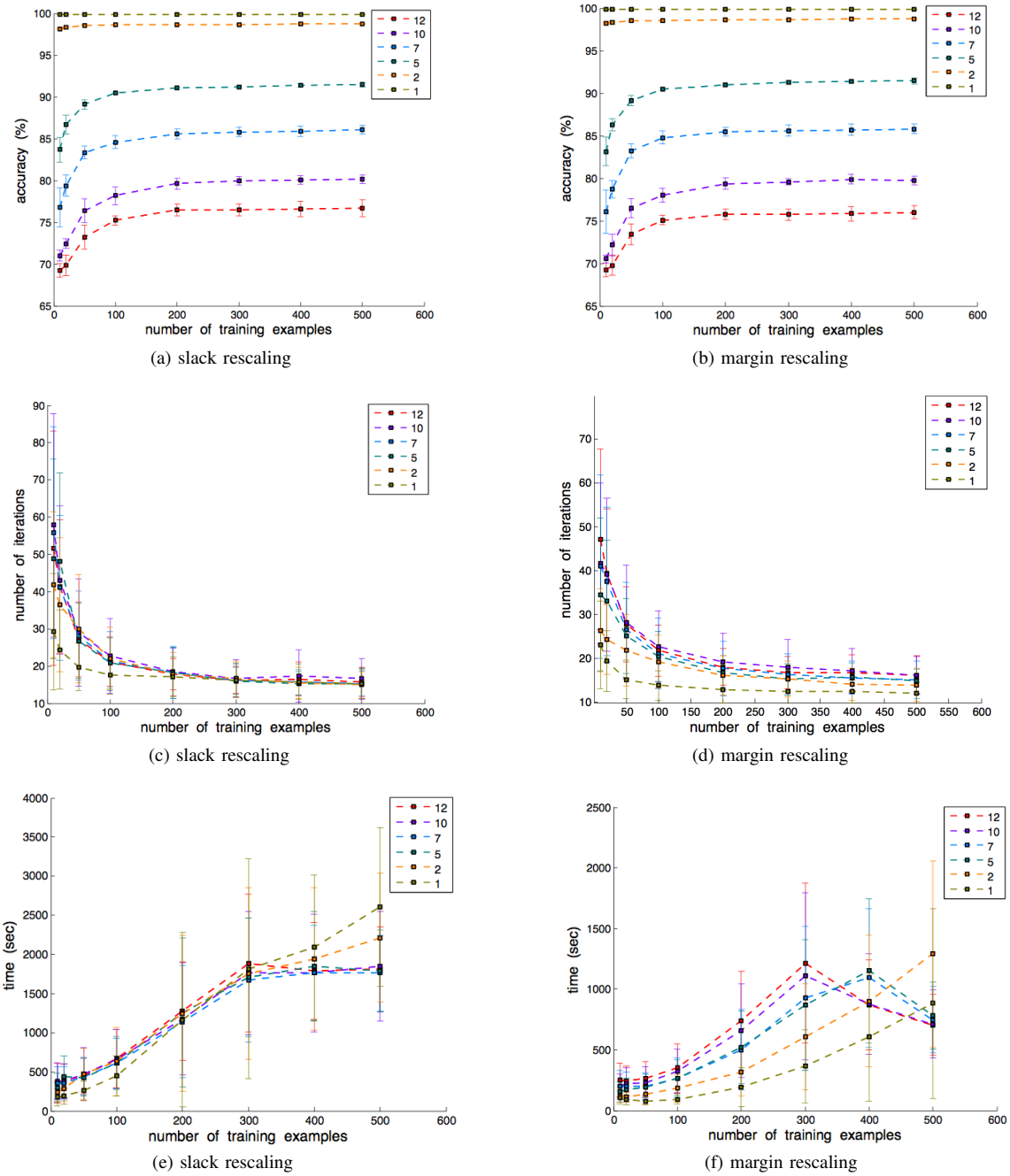
$\sigma = 1$	$\sigma = 2$	$\sigma = 5$	$\sigma = 7$	$\sigma = 10$	$\sigma = 12$
0	0	0	0.3	0.4	0.7

The numbers of iterations taken by the cutting-plane algorithm for both scaling methods are also very close. Nevertheless, due to the difference in running time of the corresponding inference algorithms, the training via slack rescaling is slower than via margin rescaling.

V. Conclusion and Discussion

Label sequence learning is a versatile but costly to optimize tool for learning when a joint feature map can define the dependencies between inputs and outputs. We have addressed the computational problems inherent to its inference algorithms. We contributed by defining a general framework (including Algorithm 1, the Loss Augmented Viterbi) based on the decomposability of the underlying joint feature map. Combined with a cutting-plane algorithm it enables an efficient training of structural SVMs for label sequence learning via slack rescaling in quadratic time w.r.t. the length of the sequences. Although we observed a similar prediction performance of margin and slack rescaling in our experiments on synthetic data, there are many references in the literature pointing out that slack rescaling is potentially more accurate yielding generally higher prediction accuracy. Furthermore, the most frequently used learning approach (via margin rescaling) is limited to the cases where the loss function can be decomposed in the same manner as the joint feature map. Algorithm 1 extends the set of manageable loss functions by a Hamming loss rescaled in an arbitrary way, which in general yields a non decomposable loss function and therefore cannot be handled by previously used inference approach via generalized Viterbi's algorithm.

Our approach thus presents an abstract unified view on inference problems, which is decoupled from the underlying learning task. Once the decomposability property of the joint feature map holds true, we can handle different problem types independently of the concrete form of the joint feature map. In label sequence learning the decomposability is naturally fulfilled; in addition, since we aim at predicting a sequence of labels, a natural choice for the loss function Δ is the number of incorrect labels or label segments. So, the Hamming loss already seems to be appropriate for many generic tasks, for example, gene finding/annotation, named entity recognition or



(a) slack rescaling (b) margin rescaling

(c) slack rescaling (d) margin rescaling

(e) slack rescaling (f) margin rescaling

Fig. 8. The graphics (a) and (b) present the averaged accuracy of slack and margin rescaling over 10 different synthetic data sets as a function of the number of training examples. Each curve corresponds to a fixed value of the standard deviation $\sigma \in \{1, 2, 5, 7, 10, 12\}$ of the Gaussian noise in the data. The graphics (c) and (d) present the averaged number of iterations of cutting-plane algorithm, where the total training time in seconds is given in (e) and (f).

part-of-speech tagging, just to mention a few. Use of an additional wrapper-function h also extends the set of manageable loss functions, so that the Hamming loss can further be scaled in an arbitrary (especially non-linear) way.

The reason for the efficiency lies in the form of dependencies between different labels in sequences modeled by a joint feature map. The concept of the decomposability enables us to divide a given inference problem in smaller overlapping subproblems, where we need to compute them only once and can save the results for reuse. This dramatically reduces the computational time. Finally, we can compute the actual solution from solutions to the subproblems based on the notion of optimal substructure of the the underlying task. In other words, the concept of decomposability of joint feature maps makes the technique of dynamic programming applicable to the task of label sequence learning.

The range of potential applications, which can be handled by our approach, is limited by the length of the involved sequences. Due to the quadratic running time (w.r.t. the sequence length) of the presented algorithm for the loss augmented inference, in practice, we only can effective address the inference tasks, where the corresponding sequences are no

longer than 2000 - 3000 sites. Otherwise longer computation and training times are to be expected.

Future work will apply the novel techniques developed here to other structured learning scenarios and also study gene finding in computational biology where slack rescaling is thought beneficial.

APPENDIX A
GENERALIZED VITERBI'S ALGORITHM

As mentioned in Section III we can solve the problem (20) by a generalized version of Viterbi's algorithm given below. For simplicity reasons we only consider the case $\ell = 1$ (the extension to the general case $\ell \geqslant 1$ is straightforward).

Generalized Viterbi's Algorithm: finds an optimal solution y^* of the problem (20) for $\ell = 1$.

Input: $g_t(\pi_t^\ell(\hat{y}))$ for all $t \in \{1, ..., T\}$ and $\hat{y} \in \{1, ..., N\}^T$

Output: y^* : an output sequence with highest value
V^* : the corresponding value of y^*

{Initialization: set the values $V_1(j)$ according to the definition (21)}

for $j = 1$ **to** N **do**
  $V_1(j) \leftarrow g_1(j)$
end for

{Recursion: compute in a bottom-up manner the values $V_t(j)$ and store the predecessor of the last state j in $S_t(j)$}

for $t = 2$ **to** T **do**
  for $j = 1$ **to** N **do**
    $V_t(j) \leftarrow \max\limits_{1 \leqslant i \leqslant N} \{V_{t-1}(i) + g_t(i, j)\}$
    $S_t(j) \leftarrow \arg\max\limits_{1 \leqslant i \leqslant N} \{V_{t-1}(i) + g_t(i, j)\}$
  end for
end for

{Termination: determine the last site y_T^* of optimal sequence y^* and the corresponding value V^*}

$V^* \leftarrow \max\limits_{1 \leqslant i \leqslant N} \{V_T(i)\}$
$y_T^* \leftarrow \arg\max\limits_{1 \leqslant i \leqslant N} \{V_T(i)\}$

{Back-Tracking: reconstruct the remaining sites y_t^* from y_T^* and S}

for $t = T - 1$ **to** 1 **do**
  $y_t^* \leftarrow S_{t+1}(y_{t+1}^*)$
end for
return y^*, V^*

APPENDIX B
PROOF OF THE THEOREM 1

Proof:
For this proof we introduce the following short notation:

$$\overline{\sup_{t,j,k}} := \sup_{\hat{y}=(\hat{y}_1,\ldots,\hat{y}_{t-1},j):\, \Delta_H(y,\hat{y})=k},$$

where we optimize over the variables $\hat{y} = (\hat{y}_1, ..., \hat{y}_t)$ restricted to the subset with $\hat{y}_t = j$ and $\Delta_H(y, \hat{y}) = k$ w.r.t. the given sequence y.

Case 1: $j \neq y_t$, $k > 1$ and $t \geqslant k$
By definition of the term $L(t-1, i, k-1)$ in (25), we have

$$\sup_{1 \leqslant i \leqslant N} L(t-1, i, k-1) + g_t(i, j) \overset{(25)}{=}$$

$$\sup_{1 \leqslant i \leqslant N} \left\{ \overline{\sup_{t-1,i,k-1}} \left\{ \textstyle\sum_{m=1}^{t-2} g_m(\pi_m^1(\hat{y})) + g_{t-1}(\hat{y}_{t-2}, i) \right\} + g_t(i, j) \right\}$$

$$\overset{(*)}{=} \overline{\sup_{t,j,k}} \ \textstyle\sum_{i=1}^{t-1} g_i(\pi_i^1(\hat{y})) + g_t(\hat{y}_{t-1}, j) = L(t, j, k)$$

where for $(*)$ we merged the two sup-terms using that by assumption $j \neq y_n$, which implies that the value of the Hamming loss increases from $k-1$ to k.

Case 2: $j \neq y_t$, $k = 1$ and $t \geqslant k$

This case is straightforward, since the assumption $j \neq y_t$ and $k = 1$ implies that the sequences y and $\hat{y}$ are equal up to the last element $\hat{y}_t$, which is fixed: $\hat{y}_t = j$.

Case 3: $j = y_t$ and $t > k$
By definition of the term $L(t-1, i, k)$ in (25), we have

$$\sup_{1 \leqslant i \leqslant N} L(t-1, i, k) + g_t(i, j) \overset{(25)}{=}$$

$$\sup_{1 \leqslant i \leqslant N} \left\{ \overline{\sup_{t-1,i,k}} \left\{ \textstyle\sum_{m=1}^{t-2} g_m(\pi_m^1(\hat{y})) + g_{t-1}(\hat{y}_{t-2}, i) \right\} + g_t(i, j) \right\}$$

$$\overset{(*)}{=} \overline{\sup_{t,j,k}} \ \textstyle\sum_{i=1}^{t-1} g_i(\pi_i^1(\hat{y})) + g_t(\hat{y}_{t-1}, j) = L(t, j, k)$$

where for $(*)$ we merged the two sup-terms using that by assumption $j = y_t$, which implies that the value of the Hamming loss doesn't change.

Case 4: $t < k$ or $j = y_t$, $t = k$

The statement of the theorem for this case follows from the definition (25) because $\{\hat{y} = (\hat{y}_1, ..., \hat{y}_{t-1}, j) : \Delta_H(y, \hat{y}) = k\}$ is empty.

□

APPENDIX C
SUPPLEMENTS TO THE ALGORITHM FOR LOSS AUGMENTED INFERENCE

Here, we discuss an extended variant of the Algorithm 1 for the general case of decomposability order $\ell \geqslant 1$. For that purpose we first give a general formulation of the Theorem 1 for $\ell \geqslant 1$.

Theorem 2. *Let $\ell \geqslant 1$, $t \in \{2, 3, ..., T\}$, $j \in \{1, 2, ..., N\}$, $k \in \{0, 1, ..., T\}$ and a fixed sequence $y = (y_1, ..., y_T) \in Y$ be given. Then the following equality holds:*

$$L(t,j,k) = \begin{cases} \sup\limits_{1\leqslant i\leqslant N} L(t-1,i,k-1) + \\ g_t(y^{t-1,i,k-1}_{\max\{1,t-\ell\}},...,y^{t-1,i,k-1}_{t-2},i,j), & j\neq y_t, k>1, t\geqslant k \\ \\ \sum_{i=1}^{t-1} g_i(\pi_i^\ell(y)) + \\ g_t(y_{\max\{1,t-\ell\}},...,y_{t-1},j), & j\neq y_t, k=1, t\geqslant k \\ \\ \sup\limits_{1\leqslant i\leqslant N} L(t-1,i,k) + \\ g_t(y^{t-1,i,k}_{\max\{1,t-\ell\}},...,y^{t-1,i,k}_{t-2},i,j), & j= y_t, t> k \\ \\ -\infty, & \textit{otherwise} \end{cases}$$

where

$$(y_1^{t,j,k},...,y_{t-1}^{t,j,k}) := \operatorname*{arg\,max}_{\hat{y}=(\hat{y}_1,...,\hat{y}_{t-1},j):\ \Delta_H(y,\hat{y})=k} \sum_{i=1}^{t} g_i(\pi_i^\ell(\hat{y}))$$

Proof:
For this proof we introduce the following short notation:

$$\overline{\sup_{t,j,k}} := \sup_{\hat{y}=(\hat{y}_1,...,\hat{y}_{t-1},j):\ \Delta(y,\hat{y})=k},$$

where we optimize over the variables $\hat{y}=(\hat{y}_1,...,\hat{y}_t)$ restricted to the subset with $\hat{y}_t = j$ and $\Delta_H(y,\hat{y}) = k$ w.r.t. the given sequence y. Furthermore we use the following abbreviations:

$$g_t^\ell(i,j) := g_t(y^{t-1,i,k-1}_{\max\{1,t-\ell\}},...,y^{t-1,i,k-1}_{t-2},i,j)$$
$$\hat{g}_t^\ell(i,j) := g_t(\hat{y}_{\max\{1,t-\ell\}},...,\hat{y}_{t-2},i,j)$$

Case 1: $j\neq y_t$, $k>1$ and $t\geqslant k$
By definition of the term $L(t-1,i,k-1)$ in (25), we have

$$\sup_{1\leqslant i\leqslant N}\left\{L(t-1,i,k-1)+g_t^\ell(i,j)\right\} \overset{(25)}{=}$$
$$\sup_{1\leqslant i\leqslant N}\left\{\overline{\sup_{t-1,i,k-1}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))\right\}+g_t^\ell(i,j)\right\} \overset{(*)}{=}$$
$$\sup_{1\leqslant i\leqslant N}\left\{\overline{\sup_{t-1,i,k-1}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))+\hat{g}_t^\ell(i,j)\right\}\right\} \overset{(**)}{=}$$
$$\overline{\sup_{t,j,k}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))+g_t(\hat{y}_{\max\{1,t-\ell\}},...,\hat{y}_{t-1},j)\right\} = L(t,j,k)$$

where for $(*)$ we used the definition of $(y_1^{t,j,k},...,y_{t-1}^{t,j,k})$ and in $(**)$ we merged the two sup-terms and used that by assumption $j\neq y_n$, which implies that the value of the Hamming loss increases from $k-1$ to k.

Case 2: $j\neq y_t$, $k=1$ and $t\geqslant k$

This case is straightforward, since the assumption $j\neq y_t$ and $k=1$ implies that the sequences y and $\hat{y}$ are equal up to the last element $\hat{y}_t$, which is fixed: $\hat{y}_t = j$.

Case 3: $j=y_t$ and $t>k$
By definition of the term $L(t-1,i,k)$ in (25), we have

$$\sup_{1\leqslant i\leqslant N}\left\{L(t-1,i,k)+g_t^\ell(i,j)\right\} \overset{(25)}{=}$$
$$\sup_{1\leqslant i\leqslant N}\left\{\overline{\sup_{t-1,i,k}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))\right\}+g_t^\ell(i,j)\right\} \overset{(*)}{=}$$
$$\sup_{1\leqslant i\leqslant N}\left\{\overline{\sup_{t-1,i,k}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))+\hat{g}_t^\ell(i,j)\right\}\right\} \overset{(**)}{=}$$
$$\overline{\sup_{t,j,k}}\left\{\textstyle\sum_{i=1}^{t-1} g_i(\pi_i^\ell(\hat{y}))+g_t(\hat{y}_{\max\{1,t-\ell\}},...,\hat{y}_{t-1},j)\right\} = L(t,j,k)$$

where for $(*)$ we used the definition of $(y_1^{t,j,k},...,y_{t-1}^{t,j,k})$ and in $(**)$ we merged the two sup-terms and used that by assumption $j=y_t$, which implies that the value of the Hamming loss doesn't change.

Case 4: $t<k$ or $j=y_t$, $t=k$

The statement of the theorem for this case follows from the definition (25) because $\{\hat{y}=(\hat{y}_1,...,\hat{y}_{t-1},j):\ \Delta_H(y,\hat{y})=k\}$ is empty.

□

Finally, we get an extended algorithm for the case $\ell\geqslant 1$ if we compute the values $L(t,j,k)$ in line 6 of Algorithm 1 by using the (extended) Theorem 2. The run-time is still dominated by the recursion part in the lines 4 - 11 having the complexity $O(T^2\cdot N^2\cdot\ell)$, which for $\ell=1$ is just $O(T^2\cdot N^2)$.

ACKNOWLEDGMENT

We thank Gunnar Rätsch for helpful discussions. This work was supported by the German Science Foundation (DFG MU 987/6-1, RA 1894/1-1) and the World Class University Program of the Korea Research Foundation (R31-10008). We would like to acknowledge support for this project from the BMBF project ALICE (01IB10003B). Marius Kloft acknowledges a postdoctoral fellowship by the German Research Foundation (DFG).

REFERENCES

[1] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *Journal of Machine Learning Research*, vol. 6, pp. 1453–1484, 2005.
[2] T. Joachims, T. Hofmann, Y. Yue, and C. nam Yu, "Predicting structured objects with support vector machines," *Communications of the ACM - Scratch Programming for All*, vol. 52, no. 11, pp. 97–104, Nov 2009.
[3] S. Sarawagi and R. Gupta, "Accurate max-margin training for structured output spaces," in *ICML*, 2008.
[4] Y. Altun, I. Tsochantaridis, and T. Hofmann, "Hidden markov support vector machines," in *ICML*, 2003.
[5] Y. Altun and T. Hofmann, "Large margin methods for label sequence learning," in *In Proc. of 8th European Conference on Speech Communication and Technology (EuroSpeech)*, 2003.
[6] Y. Altun, T. Hofmann, and M. Johnson, "Discriminative learning for label sequences via boosting," in *Advances in Neural Information Processing Systems 15*. MIT Press, 2002, pp. 977–984.
[7] Y. Altun, M. Johnson, and T. Hofmann, "Investigating loss functions and optimization methods for discriminative learning of label sequences," in *In Proc. EMNLP*, 2003.

[8] T. Joachims, T. Finley, and C.-N. J. Yu, "Cutting-plane training of structural svms," *Machine Learning*, vol. 77, no. 1, pp. 27–59, Oct 2009.

[9] M. Schmidt, "A note on structural extensions of svms," 2009.

[10] H. Xue, S. Chen, and Q. Yang, "Structural support vector machine," in *ISNN '08 Proceedings of the 5th international symposium on Neural Networks: Advances in Neural Networks*, 2008, pp. 501–511.

[11] G. Swaminathan, S. Dr, and K. Gupta, "The viterbi algorithm," in *Proc IEEE, Vol. 61, no. 3, Mai*, 1973.

[12] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proceedings of the IEEE*, Feb 1989, pp. 257–286.

[13] R. Esposito and D. P. Radicioni, "Carpediem: Optimizing the viterbi algorithm and applications to supervised sequential learning," *The Journal of Machine Learning Research*, vol. 10, pp. 1851–1880, Jan 2009.

[14] J. S. Reeve, "A parallel viterbi decoding algorithm," *Concurrency and Computation*, vol. 13, pp. 95–102, 2000.

[15] L. R. Rabiner and B. H. Juang, "An introduction to hidden markov models," *IEEE ASSp Magazine*, 1986.

[16] A. Krogh, M. Brown, I. S. Mian, K. Sjlander, and D. Haussler, "Hidden markov models in computational biology: applications to protein modeling," *Journal of Molecular Biology*, vol. 235, pp. 1501–1531, 1994.

[17] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2001.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Eddition*. The MIT Press 2009.

[19] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *ICML*, 2004.

[20] Y. Altun and D. M. A. Chicago, "Maximum margin semi-supervised learning for structured variables," in *Advances in Neural Information Processing Systems 18*, 2005, p. 18.

[21] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *NIPS*. MIT Press, 2003.

[22] H. D. Iii and D. Marcu, "Learning as search optimization: Approximate large margin methods for structured prediction," in *ICML*, 2005, pp. 169–176.

[23] J. Lafferty, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *ICML*. Morgan Kaufmann, 2001, pp. 282–289.

[24] C. H. Lampert, "Maximum margin multi-label structured prediction," in *NIPS'11*, 2011, pp. 289–297.

[25] P. Pletscher, C. S. Ong, and J. M. Buhmann, "Entropy and margin maximization for structured output learning," in *ECML PKDD'10 Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part III*, 2010, pp. 83–98.

[26] C. nam John Yu and T. Joachims, "Learning structural svms with latent variables," in *ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 1169–1176.

[27] G. Rätsch and S. Sonnenburg, "Large scale hidden semi-markov svms," in *NIPS*, 2006.

[28] W. Zhong, W. Pan, J. T. Kwok, and I. W. Tsang, "Incorporating the loss function into discriminative clustering of structured outputs," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1564–1575, Oct 2010.

[29] B. Zhao, J. Kwok, and C. Zhang, "Maximum margin clustering with multivariate loss function," in *ICDM '09 Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 637–646.

[30] C. Cortes, M. Mohri, and J. Weston, "A general regression technique for learning transductions," in *Proceedings of ICML 2005*. ACM Press, 2005, pp. 153–160.

[31] U. Brefeld and T. Scheffer, "Semi-supervised learning for structured output variables," in *ICML06, 23rd International Conference on Machine Learning*, 2006.

[32] A. Zien, U. Brefeld, and T. Scheffer, "Transductive support vector machines for structured variables," in *ICML '07 Proceedings of the 24th international conference on Machine learning*, 2007, pp. 1183–1190.

[33] N. Kaji, Y. Fujiwara, N. Yoshinaga, and M. Kitsuregawa, "Efficient staggered decoding for sequence labeling," in *ACL '10 Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010, pp. 485–494.

[34] S. Chatterjee and S. Russell, "A temporally abstracted viterbi algorithm," in *UAI'11*, 2011, pp. 96–104.

[35] A. Bordes, N. Usunier, and L. Bottou, "Sequence labelling svms trained in one pass," in *ECML PKDD '08 Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, 2008, pp. 146–161.

Alexander Bauer received a B.Sc. in mathematics 2011 and a diploma in computer science in 2012, both from Technical University of Berlin, Germany. He is currently pursuing the Doctoral degree in the machine learning group of the Berlin Institute of Technology headed by Klaus-Robert Müller. His scientific interests include different methods in the field of machine learning, in particular, learning with structured outputs.

Nico Görnitz received a diploma (MSc equivalent) in computer engineering from the Technische Universität Berlin. Currently, he is enrolled as a PhD student in the machine learning program at the Berlin Institute of Technology headed by Klaus-Robert Müller. From 2010-2011, he was also affiliated with the Friedrich Miescher Laboratory of the Max Planck Society in Tübingen, Germany where he was co-advised by Gunnar Rätsch. His primary research interests cover machine learning and it's applications in computational biology and computer security. This includes especially structured output prediction, specifically large-scale label sequence learning for transcript prediction. Furthermore, he had been working on level-set estimation, semi-supervised learning, anomaly detection as well as corresponding optimization methods.

Franziska Biegler received a diploma (M.Sc. equivalent) in computer science in 2005 from the University of Potsdam, Germany. She received her Ph.D. in computer science from The University of Western Ontario, Canada, in 2009. Her diploma as well as Ph.D. research was the area of formal languages and combinatorics on words. Since 2011, Franziska is a postdoctoral fellow in Klaus-Robert Müller's research group at TU Berlin. Her primary research interests are accurate predictions of molecular properties as well as at the crossroads of machine learning and formal languages, e.g. automatic learning and analysis of sequences.

Klaus-Robert Müller has been Professor for Computer Science at Technische Universität Berlin since 2006; at the same time he is directing the Bernstein Focus on Neurotechnology Berlin. Since 2012 he is distinguished professor at Korea University within the WCU Program. He studied physics in Karlsruhe from 1984-89 and obtained his PhD in Computer Science at TU Karlsruhe in 1992. After a PostDoc at GMD-FIRST, Berlin, he was a Research Fellow at University of Tokyo from 1994-1995. From 1995 he built up the Intelligent Data Analysis (IDA) group at GMD-FIRST (later Fraunhofer FIRST) and directed it until 2008. 1999-2006 he was a Professor at University of Potsdam. In 1999, he was awarded the Olympus Prize by the German Pattern Recognition Society, DAGM and in 2006 he received the SEL Alcatel Communication Award. In 2012 he was elected to be a member of the German National Academy of Sciences Leopoldina. His research interests are intelligent data analysis, machine learning, signal processing and Brain Computer Interfaces.

Marius Kloft received a diploma in mathematics from Philipps-Universitat Marburg with a thesis in algebraic geometry and a PhD in computer science from Technische Universitat Berlin, where he was co-advised by Klaus-Robert Muller and Gilles Blanchard. During his PhD, he spend a year at UC Berkeley, where he advised by with Peter Bartlett. After research visits to the Friedrich-Miescher-Laboratory of the Max Planck Society, Tubingen, and Korea University, Seoul, he is now a postdoctoral fellow, jointly appointed at Courant Institute of Mathematical Sciences and Memorial Sloan-Kettering Cancer Center, New York, working with Mehryar Mohri and Gunnar Ratsch, respectively. His current research focus lies in the field of machine learning with non-i.i.d. data and applications to genomic cancer data.