Sequence Labelling SVMs Trained in One Pass

Antoine Bordes[1,2], Nicolas Usunier[1], and Léon Bottou[2]

[1] LIP6, Université Paris 6, 104 Avenue du Pdt Kennedy, 75016 Paris, France
[2] NEC Laboratories America, Inc., 4 Independence Way, Princeton, NJ08540, USA
{bordes,usunier}@poleia.lip6.fr, leonb@nec-labs.com

Abstract. This paper proposes an online solver of the dual formulation of support vector machines for structured output spaces. We apply it to sequence labelling using the exact and greedy inference schemes. In both cases, the per-sequence training time is the same as a perceptron based on the same inference procedure, up to a small multiplicative constant. Comparing the two inference schemes, the greedy version is much faster. It is also amenable to higher order Markov assumptions and performs similarly on test. In comparison to existing algorithms, both versions match the accuracies of batch solvers that use exact inference after a single pass over the training examples.

1 Introduction

The sequence labelling task consists in predicting a sequence of *labels* $(y^1 \ldots y^T)$ given an observed sequence of *tokens* $(x^1 \ldots x^T)$. This task is an example of a *structured output* learning system (see e.g. [1]). It appears in practical problems in computational linguistics and signal processing.

Two informal assumptions are crucial for this task. The first states that a label y^t depends only on the surrounding labels and tokens. The second states that this dependency is invariant with t. These assumptions are expressed through the parametric formulation of the models, and, in the case of probabilistic models, through conditional independence assumptions (e.g. Markov models). Part of the model specification is then the *inference procedure* that recovers the predicted labels for any input sequence. *Exact inference* can be carried out with the Viterbi algorithm. The more efficient *greedy inference*, which predicts the labels in the order of the sequence using the past predictions, can also be competitive in terms of accuracy by considering higher order Markov assumptions. The parameters for both inference schemes can be learned using structured output learning algorithms.

Batch sequence algorithms optimize a global objective function that depends on all training sequences or tokens [2–6]. They mainly consist of an iterative procedure that run several times over the entire dataset until some convergence criterion is met. The number of epochs of these algorithms usually increases with the number of examples, leading to training times that grow faster than the size of the training set.

A crucial issue with these algorithms is their *scalability*. When learning the parameters for exact inference, support vector methods (e.g. [3]) require the application of the costly Viterbi algorithm each time a sequence is visited in the iterative process. Output space factorization methods (e.g. [5]) solve an alternative problem with additional variables that encode the structure of the possible predicted sequences. In these methods, each sequence adds a number of such variables that is polynomial in the length of the sequence. Learning for greedy inference reduces to a smaller multiclass classification problem and is therefore much faster. However algorithms then focus on tokens rather than sequences, dramatically increasing the size of the training set. Batch sequence learning algorithms, having a computational cost that grows more than linearly with the number of sequences, are impracticable on large datasets because of the high per-sequence training cost.

Online sequence learning algorithms have been proposed as a scalable alternative to batch algorithms. They run a single pass on the training set, sequentially updating their parameters depending on the loss observed after each sequence (e.g. [7]) or token (e.g. [8]). Their computational cost therefore depends linearly on the number of observations.

Proponents of such algorithms often mention that generalization bounds for online algorithms are no worse than generalization bounds for batch algorithms [9], or that specific algorithms like the second order stochastic gradient descent (SOSGD) provably loose nothing relatively to the batch optimization of the same cost [10]. However, the error bounds are not tight, such theoretical guarantees are thus not very informative, and SOSGD algorithms requires an impractically large inverse Hessian matrix. In practice, it appears that online algorithms are still significantly less accurate than batch algorithms.[3]

In this paper, we propose an online algorithm for the optimization of the dual formulation of support vector methods for structured output spaces [2, 3]. Following recent works on the fast optimization of Support Vector Machines [11, 6], the algorithm performs SMO-like optimization steps over pairs of dual variables, alternating between unseen patterns and currently support patterns. It can be seen as an adaptation of LaRank ([6]), originally proposed for the batch optimization of multiclass SVMs, to online structured output learning.

The algorithm we propose shares the scalability property of other online algorithms, its training time increasing linearly with the number of examples. Similarly to [3, 6], its number of support vectors is conveniently bounded. Finally, using an extension of [12] to structured outputs, we show that our algorithm has at least the same theoretical guarantees in terms of regret (difference between the online error and the optimal train error) as *passive-aggressive* online algorithms. Its only drawback is to keep in memory the current vector expansion. However this memory usage grows at most linearly with the number of examples and

[3] A common workaround consists in performing several passes over the training examples. But this is no longer an online algorithm and it no longer enjoys the theoretical guarantees of online algorithms.

does not impact the computational cost. This drawback is shared by other online algorithms such as kernel or averaged perceptrons.

We present an empirical evaluation of our algorithm on standard benchmarks for sequence labelling. We test both exact and greedy inference. The performances are very close to state-of-the-art batch optimizers of the same dual, while being an order of magnitude faster. The new algorithm is then only a constant time slower than perceptron-like algorithms using the same inference scheme, while being significantly better in terms of accuracies. We therefore obtain new kinds of compromises in terms of training time/test performance. For example, the greedy version of our algorithm is approximately as fast as an online perceptron using exact inference, while being almost as accurate as a batch optimizer.

2 Representation and Inference

In the rest of this paper, we use bold characters for sequences such as the sequence of tokens $\mathbf{x} = (x^1 \ldots x^T)$ or the sequence of labels $\mathbf{y} = (y^1 \ldots y^T)$. Subsequences are denoted using superscripts, as in $\mathbf{y}^{\{t-k..t-1\}} = (y^{t-k} \ldots y^{t-1})$. We call $\mathcal{X}$ the set of possible tokens and $\mathcal{Y}$ the set of possible labels, augmented with a special symbol to represent the absence of a label. By convention, a label y^s is the special symbol whenever $s \leq 0$. Angle brackets $\langle .,. \rangle$ are used to represent the canonical dot product.

An *inference procedure* assigns a label y^t to each corresponding x^t taking into account the correlations between labels at different positions in the sequence. This work takes into account correlations between $k+1$ successive labels (Markov assumption of order k). More specifically, we assume that the inference procedure determines the predicted label sequence $\mathbf{y}$ on the sole basis of the scores

$$s^t(w, \mathbf{x}, \mathbf{y}) = \left\langle w, \Phi_{\mathrm{g}} \left(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t \right) \right\rangle \quad t = 1 \ldots T,$$

where $w \in \mathbb{R}^D$ is a parameter vector and function $\Phi_{\mathrm{g}} : \mathcal{X} \times \mathcal{Y}^k \times \mathcal{Y} \to \mathbb{R}^D$ determines the feature space.

2.1 Exact Inference

Exact inference maximizes the sum $\sum_{t=1}^{T} s^t(w, \mathbf{x}, \mathbf{y})$ over all possible label sequences $\mathbf{y}$. For a given input sequence $\mathbf{x}$, the prediction function $\mathbf{f}_{\mathrm{e}}(w, \mathbf{x})$ is then defined by

$$\begin{aligned} \mathbf{f}_{\mathrm{e}}(w, \mathbf{x}) &= \arg\max_{\mathbf{y} \in \mathcal{Y}^T} \sum_{t=1}^{T} s^t(w, \mathbf{x}, \mathbf{y}) \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}^T} \left\langle w, \Phi_{\mathrm{e}}(\mathbf{x}, \mathbf{y}) \right\rangle , \end{aligned} \tag{1}$$

where $\Phi_{\mathrm{e}}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} \Phi_{\mathrm{g}}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)$.

2.2 Greedy Inference

Greedy inference predicts the successive labels y^t in sequence by maximizing $\left\langle w, \Phi_{\mathrm{g}}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)\right\rangle$ where the previously predicted labels $\mathbf{y}^{\{t-k..t-1\}}$ are frozen. For a given input sequence $\mathbf{x}$, the prediction function $\mathbf{f}_{\mathrm{g}}(w, \mathbf{x})$ is then defined by the recursion

$$f^t_{\mathrm{g}}(w, \mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \left\langle w, \Phi_{\mathrm{g}}\left(x^t, \mathbf{f}_{\mathrm{g}}^{\{t-k..t-1\}}(w, \mathbf{x}), y\right) \right\rangle . \tag{2}$$

2.3 Comparison

Although greedy inference is an approximation of exact inference, their different computational complexity leads to a more nuanced picture. Exact inference solves (1) using the Viterbi algorithm. It requires a time proportional to $DT\mathrm{card}(\mathcal{Y})^{k+1}$ and becomes intractable when the order k of the Markov assumption increases. On the other hand, the recursion (2) runs in time proportional to $DT\mathrm{card}(\mathcal{Y})$. Therefore greedy inference is practicable with large k.

In practice, greedy inference with large k can sometimes achieve a higher accuracy than exact inference with Markov assumptions of lower order.

3 Training

In this section we write the convex optimization problem used for determining the parameter vector for both cases of exact and greedy inference. We first present a large margin formulation of the multiclass problem and show how it applies to both problems.

3.1 Large-Margin Multiclass Problem

We consider training patterns $p_1 \dots p_n$ and their classes $c_1 \dots c_n$. Following the formulation of large-margin learning with interdependent output spaces [2, 3], the parameters of a prediction function of the form $f(p) = \arg\max_c \langle w, \Phi(p, c)\rangle$ can be determined by minimizing the convex function

$$\min_w \quad \frac{1}{2}\langle w, w\rangle + C \sum_{i=1}^{n} \xi_i \tag{3}$$

$$\text{subject to} \begin{cases} \forall i \quad \xi_i \geq 0 \\ \forall i \quad \forall c \neq c_i \quad \langle w, \Phi(p_i, c_i) - \Phi(p_i, c)\rangle \geq \Delta(c_i, c) - \xi_i \,, \end{cases}$$

where $\Delta(c_i, c)$ is the true loss incurred by predicting class c instead of the true class c_i. Following [6], we rewrite this optimization problem in dual form by introducing one coefficient β_i^c for each pattern p_i and each class $c \in \mathcal{C}$.

$$\max_{\boldsymbol{\beta}} \quad -\sum_{i,c} \Delta(c, c_i)\beta_i^c - \frac{1}{2}\sum_{i,j,c,\bar{c}} \beta_i^c \beta_j^{\bar{c}} \langle \Phi(p_i, c), \Phi(p_j, \bar{c})\rangle$$

$$\text{subject to} \begin{cases} \forall i \quad \forall c \quad \beta_i^c \leq \delta(c, c_i)\, C \\ \forall i \quad \sum_c \beta_i^c = 0 \end{cases} \tag{4}$$

where $\delta(c, \bar{c})$ is 1 when $c = \bar{c}$ and 0 otherwise. The solution of the primal problem is then recovered from the optimal coefficients β_i^c as

$$w = \sum_{i,c} \beta_i^c \Phi(p_i, c)\,.$$

Multiple algorithms have been proposed to efficiently solve problem (4) even in cases where the number of classes is very large [3, 6]. As such, an optimizer of problem (4) can be used for learning the parameters of sequence labelling models for both exact and greedy inference. For clarity in the presentation, we give the instantiation of the dual objective for both problems using the notations introduced in section 2.

3.2 Training for Exact Inference

Since the exact inference prediction function (1) can be written under the form $\arg\max_c \langle w, \Phi(p, c)\rangle$, the above formulation applies directly. The patterns p_i are the token sequences $\mathbf{x}_i$ and the classes c are complete label sequences $\mathbf{y}$. The feature function $\Phi(p_i, c) = \Phi_\mathrm{e}(\mathbf{x}_i, \mathbf{y})$ has been defined in (1) and the loss $\Delta(\mathbf{y}, \bar{\mathbf{y}})$ is the Hamming distance between the sequences $\mathbf{y}$ and $\bar{\mathbf{y}}$.

The dual problem is then

$$\begin{aligned} \max_{\boldsymbol{\beta}} \quad & -\sum_{i,\mathbf{y}} \Delta(\mathbf{y}, \mathbf{y}_i)\beta_i^{\mathbf{y}} - \frac{1}{2}\sum_{ij}\sum_{\mathbf{y}\bar{\mathbf{y}}} \beta_i^{\mathbf{y}}\beta_j^{\bar{\mathbf{y}}} K_\mathrm{e}^{ij\mathbf{y}\bar{\mathbf{y}}} \\ & \text{subject to} \begin{cases} \forall i \;\; \forall \mathbf{y} \;\; \beta_i^{\mathbf{y}} \leq \delta(\mathbf{y}, \mathbf{y}_i)\, C \\ \forall i \;\; \sum_{\mathbf{y}} \beta_i^{\mathbf{y}} = 0\,. \end{cases} \end{aligned} \tag{5}$$

with the kernel matrix $K_\mathrm{e}^{ij\mathbf{y}\bar{\mathbf{y}}} = \langle \Phi_\mathrm{e}(\mathbf{x}_i, \mathbf{y}), \Phi_\mathrm{e}(\mathbf{x}_j, \bar{\mathbf{y}})\rangle$.

The solution is then $w = \sum_{i\mathbf{y}} \beta_i^{\mathbf{y}}\, \Phi_\mathrm{e}(\mathbf{x}_i, \mathbf{y})$.

3.3 Training for Greedy Inference

The greedy inference prediction function (2) does not readily have the form $\arg\max_c \langle w, \Phi(p, c)\rangle$ because of its recursive structure. However, each prediction f_g^t has the desired form with one pattern p_{it} for each training token x_i^t, and with classes c taken from the set of labels and compared with $\Delta(y, \bar{y}) = 1 - \delta(y, \bar{y})$.

This approach leads to difficulties because the feature function $\Phi(p_{it}, y) = \Phi_\mathrm{g}(x_i^t, \mathbf{f}_\mathrm{g}^{\{t-k..t-1\}}, y)$ depends on the prediction function. We avoid this difficulty by approximating the predicted labels $\mathbf{f}_\mathrm{g}^{\{t-k..t-1\}}$ with the true labels $\mathbf{y}_i^{\{t-k..t-1\}}$.

The corresponding dual problem is then

$$\begin{aligned} \max_{\boldsymbol{\beta}} \quad & -\sum_{ity} \Delta(y, y_i^t)\beta_{it}^{y} - \frac{1}{2}\sum_{itjr}\sum_{y\bar{y}} \beta_{it}^{y}\beta_{jr}^{\bar{y}} K_\mathrm{g}^{itjry\bar{y}} \\ & \text{subject to} \begin{cases} \forall i, t \;\; \forall y \;\; \beta_{it}^{y} \leq \delta(y, y_i^t)\, C \\ \forall i, t \;\; \sum_{y} \beta_{it}^{y} = 0\,. \end{cases} \end{aligned} \tag{6}$$

with the kernel matrix $K_\mathrm{g}^{itjry\bar{y}} = \left\langle \Phi_\mathrm{g}(x_i^t, \mathbf{y}_i^{\{t-k..t-1\}}, y)\,,\, \Phi_\mathrm{g}(x_j^r, \mathbf{y}_j^{\{r-k..r-1\}}, \bar{y})\right\rangle$.

The solution is then $w = \sum_{ity} \beta_{it}^{y}\, \Phi_\mathrm{g}(x_i^t, \mathbf{y}_i^{\{t-k..t-1\}}, y)$.

3.4 Discussion

Both dual problems (5) and (6) are defined using very different sets of coefficients $\boldsymbol{\beta}$. Experiments (section 6) show considerable differences in sparsity. Yet the two kernel matrices K_e and K_g generate parameter vectors w in the same feature space which is determined by the choice of the feature function Φ_g, or equivalently the choice of the kernel K_g.

We use the following kernels in the rest of this paper.

$$K_\mathrm{g}^{itjry\bar{y}} = \delta(y,\bar{y})\Big(k(x_i^t, x_j^r) + \sum_{s=1}^{k} \delta(y_i^{t-s}, \bar{y}_j^{r-s})\Big)\,,$$

$$K_\mathrm{e}^{ij\mathbf{y}\bar{\mathbf{y}}} = \sum_{tr} \delta(y^t, \bar{y}^r)\Big(k(x_i^t, x_j^r) + \sum_{s=1}^{k} \delta(y^{\,t-s}, \bar{y}^{\,r-s})\Big)\,,$$

where $k(x,\bar{x}) = \langle x, \bar{x}\rangle$ is a linear kernel defined on the tokens. These two kernels satisfy the identity $\Phi_\mathrm{e}(\mathbf{x},\mathbf{y}) = \sum_i \Phi_\mathrm{g}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)$ by construction. Furthermore, the exact inference kernel K_e is precisely the kernel proposed in [2].

The greedy kernel approximates the predicted labels with the true labels. The same approximation was used in LaSO [8] and in the first iteration of SEARN [4]. In the context of an online algorithm, other approximations would have been possible, such as the sequence of predicted labels for the previous values of the parameter. However, the simpler approximation works slightly better in our experiments.

4 Online Optimization

This section discusses the online optimization of problems (4), and therefore problems (5) and (6). We call our algorithm OLaRank (for Online LaRank), because it uses the same building blocks as the LaRank algorithm [6]. We first summarize these building blocks. Then, we describe how we adapted the original LaRank for our purposes.

4.1 OLaRank Building Blocks

The LaRank algorithm is a batch optimizer of the dual problem (4), for the specific case of multiclass classification with one representative vector per class. It can however be straightforwardly adapted to the general case of structured outputs we consider here. Assuming these adaptations done, LaRank can be applied to both problems (5) and (6).

Each elementary step of the LaRank algorithm maximizes the objective function (4) restricted to only two coefficients β_i^c and $\beta_i^{\bar{c}}$ associated with a same pattern p_i. Because these coefficients appear in the same equality constraint, they must move by opposite amounts. The maximization then simply becomes one-dimensional and can be solved analytically (see details in [6]). Each elementary step monotonically improves the dual objective function.

[6] define three ways to select pairs of coefficients, namely ProcessNew, ProcessOld, and Optimize. They prove that a reasonable mixture of these operations approaches the solution of problem (4) with a predefined accuracy, after a number of steps that grows linearly with the number of patterns and does not depend on the number of classes.

OLaRank uses exactly the same three basic operations, with a difference in the alternation between them. For clarity, we remind these definitions from [6]: *support vectors* are the pairs (p_i, c) such that $\beta_i^c \neq 0$, and *support patterns* are the patterns p_i for which at least one of the β_i^c coefficients is nonzero.

- ProcessNew randomly picks a fresh example (p_i, c_i) and chooses the best pair of coefficients β_i^c and $\beta_i^{\bar{c}}$ according to the gradient vector of the dual objective function.
- ProcessOld randomly picks a support pattern p_i and chooses the best pair of coefficient β_i^c and $\beta_i^{\bar{c}}$ according to the gradient vector.
- Optimize randomly picks a support pattern p_i and chooses the best pair of coefficient β_i^c and $\beta_i^{\bar{c}}$ according to the gradient vector but solely among the coefficients β_i^c that are not zero.

The ProcessOld and Optimize steps differ essentially in their computational costs. In the case of exact inference, ProcessOld requires to run the Viterbi algorithm to find the label sequence that maximizes the gradient for the chosen support pattern. Optimize only considers label sequences that are currently support vectors, avoiding the costly inference procedure. In the case of greedy inference, ProcessOld needs to compute the score of the current token for each possible label, while Optimize only considers the labels that are currently in the support vector expansion.

4.2 Scheduling

The LaRank algorithm schedules the three types of steps using a complex adaptive scheme that takes into account both the computing time and the progress of the objective function. This scheme works well for simple multiclass problems. However, we had mixed experiences with the exact inference models, because the ProcessOld operations incur a penalization in terms of computation time due to the Viterbi algorithm. In the end, ProcessOld was not sufficiently applied, leading to poor performance.

OLaRank implements a much simpler approach. We call Reprocess the combination of one ProcessOld step followed by ten Optimize steps. All our experiments are carried out by repeatedly performing one ProcessNew step followed by a predefined number n_{R} of Reprocess combinations. The number n_{R} depends on each problem and is determined like an hyper-parameter using a validation set (see table 1 and figure 3).

Notice that only performing ProcessNew steps (i.e. $n_{\mathrm{R}} = 0$) yields a typical *passive-aggressive* online algorithm [13]. Therefore, the Reprocess operation is the element that lets OLaRank match the test accuracy of batch optimization after a single sweep over the training data (see section 6).

5 Theoretical Analysis

This section displays theoretical results concerning the OLaRank algorithm presented in the previous section: a bound on the number of support vectors and another on the regret.

5.1 Sparsity Guarantee

All three basic operations (ProcessNew, ProcessOld and Optimize) do nothing unless they can find a search direction that fulfills two conditions: (1) the derivative of the objective function along this direction must be greater than $\tau > 0$, and, (2) a movement of size $\kappa > 0$ along that direction is possible without leaving the constraint polytope. Therefore OLaRank is an Approximate Stochastic Witness Direction Search (ASWDS) algorithm as defined in [11]. The number of support vectors it adds during learning can be bounded by the following proposition:

Proposition 1. *While training on n examples,* OLaRank *will add no more than*

$$\min\left\{n(2+n_{\mathrm{R}}),\ \ \max\{\frac{2\rho_{max}nC}{\tau^2}, \frac{2nC}{\kappa\tau}\}\right\}$$

support vectors, with $\rho_{max} = \max_{i,c} ||\Phi(p_i, c) - \Phi(p_i, c_i)||^2$.

The bound is actually a bound on the number of ProcessNew and ProcessOld iterations, since each ProcessNew adds either 0 or 2 support vectors, and each ProcessOld adds either 0 or 1 support vector. The term $n(2+n_{\mathrm{R}})$ is immediate considering the scheduling described in section 4. The term $\max\{\frac{2\rho_{max}nC}{\tau^2}, \frac{2nC}{\kappa\tau}\}$ is a bound on the maximal number of optimization steps carried out by LaRank (and therefore OLaRank). Its proof follows the theorem of [6].

In practice, the smaller term is obviously $n(2+n_{\mathrm{R}})$, since reasonable values of n_{R} are between 1 and 20. The interest of the second term of the bound is at the limit when n_{R} tends to infinity. Then OLaRank converges, at each round, to a $\kappa\tau$-approximate solution of problem (4), restricted to consider only the examples that are currently support patterns (see theorem 18 of [11] for the convergence of ASWDS algorithms). In that case, the bound proves that the number of support vectors still grows at most linearly with the number of examples given κ and τ.

5.2 Regret Bound

The OLaRank algorithm performs an iterative optimization of the dual, where only the parameters corresponding to already seen examples can be modified at each step. The primal-dual view of online learning of [12] allows to obtain online learning rates for that kind of algorithms in the case of SVMs for binary classification. In this section, we extend their result to structured predictors (i.e. online optimizers of equation (4)).

Regret Bound for Online Structured Predictors The learning rates are expressed with the notion of *regret* defined by the difference between the mean loss incurred by the algorithm on the course of learning and the empirical loss of a given weight vector,

$$\mathrm{regret}(n, w) = \frac{1}{n} \sum_{i=1}^{n} \ell(w_i, (p_i, c_i)) - \frac{1}{n} \sum_{i=1}^{n} \ell(w, (p_i, c_i))$$

with w_i the primal weight vector before seeing the i-th example, and $\ell(w, (p, c))$ the loss incurred by any weight vector w on the example (p, c). In our setup, the loss $\ell(w_i, (p_i, c_i))$ is defined as

$$\left[\max_{c \in \mathcal{C}} \Delta(c_i, c) - \langle w, \Phi(p_i, c_i) - \Phi(p_i, c) \rangle \right]_+$$

where $[x]_+ = \max(x, 0)$.

The following theorem gives a bound on the regret for any algorithm performing an online optimization of the dual of equation (4):

Theorem 1. *Assume that for all i, the dual increase after seeing example* (p_i, c_i) *is at least* $C\mu_\rho(\ell(w_i, (p_i, c_i)))$*, with*

$$\mu_\rho(x) = \frac{1}{\rho C} \min(x, \rho C) \left(x - \frac{1}{2} \min(x, \rho C) \right)$$

then, we have:

$$\forall w, \;\; \mathit{regret}(n, w) \leq \frac{||w||^2}{2nC} + \frac{\rho C}{2}.$$

The proof exactly follows section 5 of [12]. The crucial point of this theorem is to directly relate the dual increase when seeing an example and the regret bound: the more we can prove that the dual increases in the course of learning, the more we can have guarantees on the performance.

Application The following result allows to use theorem 1 to bound the regret for the OLaRank algorithm:

Proposition 2. *For a given i, the dual increase after performing a* ProcessNew *step on example* (p_i, c_i) *is equal to* $C\mu_{\rho^i}(\ell(w_i, (p_i, c_i)))$*, with* $\rho^i = ||\Phi(p_i, c_i) - \Phi(p_i, c_i^*)||^2$ *and* $c_i^* = \arg\max_{c \in \mathcal{C}} \left(\Delta(c_i, c) + \langle w_i, \Phi(p_i, c) \rangle \right)$.

This proposition is easily established by directly calculating the dual increase caused by ProcessNew step (see [6]) and expressing the result using the function μ_ρ. Since Reprocess cannot decrease the dual, the whole OLaRank algorithm increases the dual by at least $C\mu_{\rho^i}(\ell(w_i, (p_i, c_i)))$ after seeing example i. Moreover, as μ_ρ monotonically decreases with ρ theorem 1 can be applied to OLaRank with $\rho = \max_i \rho^i$.

Interpretation Proposition 2 first shows that OLaRank has the same guarantees (in terms of regret) than a typical *passive-aggressive* algorithm. Indeed he latter is equivalent to performing only ProcessNew operations.

In addition, Theorem 1 provides a partial justification of the Reprocess function. Indeed, it expresses that we can relate the dual increase to the regret. As such, if, for instance, Reprocess operations bring a dual increase of the same order of magnitude as ProcessNew operations at each round, then the regret of OLaRank would be typically two times smaller than the current bound. Although we do not have any analytical results concerning the dual increase ratio between ProcessNew and Reprocess operations, the theorem suggest that the true regret of OLaRank should be much smaller than the bound.

Finally, we can notice that the regret we consider here does not match the true applicative setting of greedy inference. Indeed, we consider in the regret bound a set of examples that is fixed independently of the parameter vector w with which we compare. But on test examples the greedy inference scheme uses the past predictions instead of the true labels. Nevertheless the bound is informative to compare online to batch learning. Indeed, if we consider the examples (p_i, c_i) in the regret bound to be the training set, Theorem 1 relates the online error with the error of the batch optimal. Then, we can claim that the online error of OLaRank will not be too far from the batch optimal trained with the same set of examples. The partial justification for the Reprocess function of the previous paragraph is still valid.

6 Experiments

This section reports experiments performed on various label sequence learning tasks to study the behavior of our online learning algorithm. We denote OLaRankGreedy, OLaRank using greedy inference and OLaRankExact when using exact inference. Since such tasks are common in the recent literature, we focus on fully supervised tasks where labels are provided for every time index. After presenting the experimental tasks we chose, we compare the performances of OLaRankExact and OLaRankGreedy to both batch and online methods to empirically validate their efficiency. We then investigate how the choice of the inference method influences the performances.

6.1 Experimental Setup

Experiments were carried out on three datasets. The Optical Character Recognition dataset (OCR) contains handwritten words, with average length of 8 characters, written by 150 human subjects and collected by [14]. This is a small dataset for which the performance evaluation is performed using 10-fold cross-validation. The Chunking dataset from the CoNLL 2000 shared task[4] consists of sentences divided in syntactically correlated segments or chunks. This dataset has more than 75,000 input features. The Wall Street Journal dataset[5] (WSJ) is a larger dataset with around 1 million words in more than 40,000 sentences and with a large number of features. The labels associated with each word are "part-of-speech" tags.

[4] `http://www.cnts.ua.ac.be/conll2000/chunking/`
[5] `http://www.cis.upenn.edu/~treebank/`

Table 1. Datasets and parameters used for the experiments.

	Training Set Sequences(Tokens)	Testing Set Sequences(Tokens)	Classes	Features	C	OLaRankGreedy n_{R}	OLaRankGreedy k	OLaRankExact n_{R}	OLaRankExact k
OCR	650 (∼4,600)	5500 (∼43,000)	26	128	0.1	5	10	10	1
Chunking	8,931 (∼212,000)	2,012 (∼47,000)	21	∼76,000	0.1	1	2	5	1
WSJ	42,466 (∼1,000,000)	2,155 (∼53,000)	44	∼130,000	0.1	1	2	5	1

Table 1 summarizes the main characteristics of these three datasets and specifies the parameters we have used for both batch and online algorithms: the constant C, the number n_{R} of Reprocess steps for each ProcessNew step, and the order k of the Markov assumptions. They have been chosen by cross-validation for the batch setting, online algorithms using the same parameters as their batch counterparts. Exact inference algorithm such as OLaRankExact are limited to first order Markov assumptions for tractability reasons.

6.2 General Performances

We report the training times for a number of algorithms as well as the percentage of correctly predicted labels on the test sets (For Chunking, we also provide F1 scores on test sets). Results for exact inference algorithms are reported in table 2. Results for greedy inference algorithms are reported in table 3.

Batch Counterparts As discussed in the introduction, our main points of comparison are the prediction accuracies achieved by batch algorithms that fully optimize the same dual problems as our online algorithms.

In the case of exact inference, our baseline is given by the SVM-HMM results using the cutting plane optimization algorithm described by [3]. In the case of greedy inference, we simply produced baseline results by running OLaRankGreedy several times over the training set until the Karush-Kuhn-Tucker conditions are satisfied. These results are labelled LaRankGreedyBatch.

Tables 2 and 3 show that both OLaRankGreedy and OLaRankExact reach competitive testing set performances relative to these baselines while saving a lot of training time.

Figure 1 depicts relative time increments. Denoting t_0 the running time of a model on a small portion of the training set of size s_0, the time increment on a training set of size s is defined as t_s/t_0. We also define the corresponding size increment as s/s_0. This allows to represent scaling in time for different models. Figure 1 thus shows that our models scale linearly in time while a common batch method as SVM-HMM does not.

The dual objective values reached by the online algorithms based on OLaRank and by their batch counterparts are quite similar as presented on table 4. OLaRankExact and OLaRankGreedy have good optimization abilities; they both reach a dual value close to the convergence point of their corresponding batch algorithms.

Table 2. Compared accuracies and times of methods using exact inference.

		OCR	Chunking *(F1 score)*	WSJ
CRF (batch)	Test. accuracy (%)	-	96.03 *(93.75)*	96.75
	Train. time (sec.)	-	568	3,400
SVM-HMM (batch)	Test. accuracy (%)	78.20	95.98 *(93.64)*	96.81
	Train. time (sec.)	180	48,000	350,000
CRF (online)	Test. accuracy (%)	-	95.26 *(92.47)*	94.42
	Train. time (sec.)	-	30	240
PerceptronExact (online)	Test. accuracy (%)	51.44	93.74 *(89.31)*	91.49
	Train. time (sec.)	0.2	10	180
PAExact (online)	Test. accuracy (%)	56.13	95.15 *(92.21)*	94.67
	Train. time (sec.)	0.5	15	185
OLaRankExact (online)	Test. accuracy (%)	75.77	95.82 *(93.34)*	96.65
	Train. time (sec.)	4	130	1380

Table 3. Compared accuracies and times of methods using greedy inference.

		OCR	Chunking *(F1 score)*	WSJ
LaRankGreedyBatch (batch)	Test. accuracy (%)	83.77	95.86 *(93.59)*	96.63
	Train. time (sec.)	15	490	9,000
PerceptronGreedy (online)	Test. accuracy (%)	51.82	93.24 *(88.84)*	92.70
	Train. time (sec.)	0.05	3	10
PAGreedy (online)	Test. accuracy (%)	61.23	94.61 *(91.55)*	94.15
	Train. time (sec.)	0.1	5	25
OLaRankGreedy (online)	Test. accuracy (%)	81.15	95.81 *(93.46)*	96.46
	Train. time (sec.)	1.4	20	175

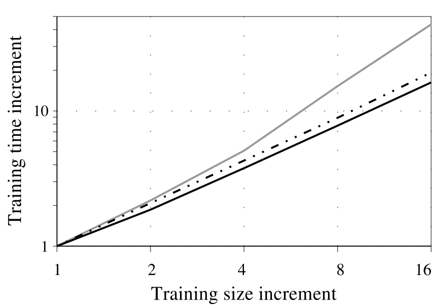

Fig. 1. Scaling with time on Chunking dataset, log-log plot. Solid black line: OLaRankGreedy, Dashed black line: OLaRankExact, Gray line: SVM-HMM.

	Chunking	WSJ
SVM-HMM (batch)	1360	9072
PAExact (online)	443	2122
OLaRankExact (online)	1195	7806
LaRankGreedyBatch (batch)	940	8913
PAGreedy (online)	410	2922
OLaRankGreedy (online)	905	8505

Table 4. Values of dual objective after training phase.

We also provide the dual of PAExact and PAGreedy, the *passive-aggressive* versions (i.e. without Reprocess) of OLaRankExact and OLaRankGreedy. These low values illustrate the crucial influence of Reprocess in the optimization process, independent of the inference method.

Other Comparisons We also provide comparisons with a number of popular algorithms for both exact and greedy inference.

For exact inference, the CRF results were obtained using a fast stochastic gradient descent implementation[6] of Conditional Random Fields: online results were obtained after one stochastic gradient pass over the training data; batch results were measured after reaching a performance peak on a validation set. The PerceptronExact results were obtained using the perceptron update described by [7] and the same exact inference scheme as OLaRankExact. The PAExact results were obtained with the *passive-aggressive* version of OLaRankExact, that is without Reprocess or Optimize steps.

For greedy inference, we report results for both PerceptronGreedy and PAGreedy. Like OLaRank, these algorithms were used in a strict online setup, performing only a single pass over the training examples.

Results in tables 2 and 3 clearly display a gap between the accuracies of these common online methods and the accuracies achieved by our two algorithms OLaRankGreedy and OLaRankExact. The OLaRank based algorithms are the only online algorithms able to match the accuracies of the batch algorithms. Although higher than those of other online algorithms, their training times remain reasonable. For example, on our largest dataset, WSJ, OLaRankGreedy reaches a test set accuracy competitive with the most accurate algorithms while requiring less training time than PerceptronExact (about four milliseconds per training sequence).

Results on the Chunking and WSJ benchmarks have been widely reported in the literature. Our Chunking results are competitive with the best results reported in the evaluation of the CoNLL 2000 shared task [15] (F1 score 93.48). More recent works include including [16] (F1 score 94.13, training time 800 seconds) and [**?**] (F1 score 94.19, training time 5000 seconds). The Stanford Tagger [17] reaches 97.24% accuracy on the WSJ task but requires 150,000 seconds of training. These state-of-the-art systems slightly exceed the performances reported in this work because they exploit highly engineered feature vectors. Both OLaRankExact and OLaRankGreedy need a fraction of these training times to achieve the full potential of our relatively simple feature vectors.

6.3 Comparing Greedy and Exact Inference

This section focuses on an empirical comparison of the differences caused by the inference scheme on learning.

[6] `http://leon.bottou.org/projects/sgd`

Inference Cost The same training set contains more training examples for an algorithm based on a greedy inference scheme. This has a computational cost. However the training time gap between PAExact and PAGreedy in tables 2 and 3 indicates that using exact inference entails much higher computational costs because the inference procedure is more complex (see section 2.3).

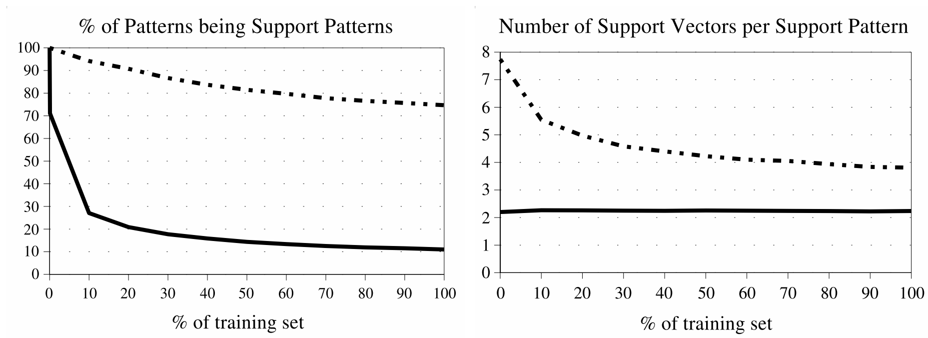

Fig. 2. Sparsity measures during learning on Chunking dataset. (Solid line: OLaRankGreedy, Dashed line: OLaRankExact.)

Sparsity As support vectors for OLaRankExact are complete sequences, local dependencies are not represented in an invariant fashion. OLaRankExact thus needs to store an important proportion of its training examples as support pattern while OLaRankGreedy only require a small fraction of them as illustrated in figure 2. Moreover, for each support pattern, OLaRankExact also need to store more support vectors.

Reprocess Figure 3 displays the gain in test accuracy that OLaRankGreedy and OLaRankExact get according to the number of Reprocess. This gain is computed relatively to the *passive-aggressive* algorithms which are similar but do not perform any Reprocess. OLaRankExact requires more Reprocess (10 on OCR) than OLaRankGreedy (only 5) to reach its best accuracy. This empirical result is verified on all three datasets. Using exact inference instead of greedy inference causes additional computations in the OLaRank algorithm.

Markov Assumption Length This section indicates that using exact inference in our setup involves both time and sparsity penalties. Moreover the loss in accuracy that could occur when using a greedy inference process and not an exact one can be compensated by using Markov assumptions of order higher than 1. As shown on figure 4 it can even lead to better generalization performances.

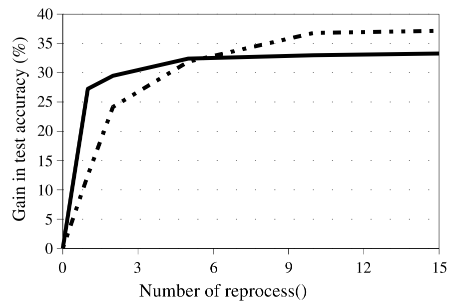

Fig. 3. Gain in test accuracy compared to the *passive-aggressives* according to n_R on OCR. (Solid line: OLaRankGreedy, Dashed line: OLaRankExact)

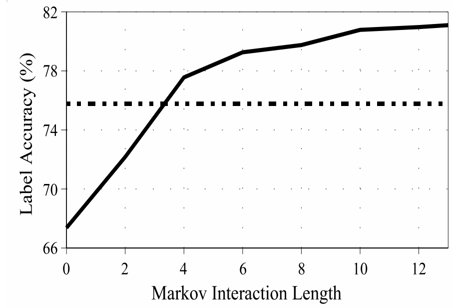

Fig. 4. Test accuracy according to the Markov interaction length on OCR. (Solid line: OLaRankGreedy, Dashed line: OLaRankExact for which $k = 1$)

7 Conclusion

The OLaRank algorithm applied to sequence labelling combines the linear scaling property of perceptrons and the accuracy of batch solvers.

Using the OLaRank algorithm with both exact and greedy inference leads to two competitive sequence labelling algorithm. Both learn in a single pass over the training data and reach the performance of equivalent batch algorithms. Both offer training times that scale linearly with the number of examples. Both have been shown to achieve good performances on well studied benchmark tasks.

Online learning and greedy inference offer the most attractive combination of short training time, high sparsity and accuracy.

Acknowledgments

Part of this work was funded by NSF grant CCR-0325463 and by the Network of Excellence IST-2002-506778 PASCAL. Antoine Bordes was also supported by the DGA.

References

1. Bakır, G., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., Vishwanathan, S.V.N., eds.: Predicting Structured Outputs. MIT Press (2007) in press.
2. Altun, Y., Tsochantaridis, I., Hofmann, T.: Hidden Markov support vector machines. In: Proceedings of the 20th International Conference on Machine Learning (ICML03), New York, NY, ACM Press (2003)
3. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research **6** (2005) 1453–1484

4. Daumé, III, H., Langford, J., Marcu, D.: Search-based structured prediction as classification. In: NIPS Workshop on Advances in Structured Learning for Text and Speech Processing, Whistler, Canada (2005)
5. Taskar, B., Guestrin, C., Koller, D.: Max-margin Markov networks. In: Advances in Neural Information Processing Systems. Volume 16. MIT Press (2004)
6. Bordes, A., Bottou, L., Gallinari, P., Weston, J.: Solving multiclass support vector machines with LaRank. In Ghahramani, Z., ed.: Proceedings of the 24th International Machine Learning Conference, Corvallis, Oregon, OmniPress (2007)
7. Collins, M.: Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In: EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing, Morristown, NJ, Association for Computational Linguistics (2002) 1–8
8. Daumé, III, H., Marcu, D.: Learning as search optimization: approximate large margin methods for structured prediction. In: Proceedings of the 22nd International Conference on Machine Learning (ICML05), New York, NY, ACM Press (2005) 169–176
9. Cesa-Bianchi, N., Conconi, A., Gentile, C.: On the generalization ability of on-line learning algorithms. IEEE Transactions on Information Theory **50**(9) (2004) 2050–2057
10. Murata, N., Amari, S.: Statistical analysis of learning dynamics. Signal Processing **74**(1) (1999) 3–28
11. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. Journal of Machine Learning Research **6** (2005) 1579–1619
12. Shalev-Shwartz, S., Singer, Y.: A primal-dual perspective of online learning algorithms. Machine Learning Journal **69:2/3** (2007) 115–142
13. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. Journal of Machine Learning Research **7** (2006) 551–585
14. Kassel, R.: A Comparison of Approaches on Online Handwritten Character Recognition. PhD thesis, MIT. Spoken Language System Group (1995)
15. Kudoh, T., Matsumoto, Y.: Use of support vector learning for chunk identification. In: Proceedings of CoNLL-2000 and LLL-2000. (2000) 252–259
16. Zhang, T., Damerau, F., Johnson, D.: Text chunking based on a generalization of winnow. Journal of Machine Learning Research **2** (2002) 615–637
17. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: HLT-NAACL. (2003) 252–259