A Multi-Plane Block-Coordinate Frank-Wolfe Algorithm for Structural SVMs with a Costly $\max$-Oracle

Neel Shah, Vladimir Kolmogorov, Christoph H. Lampert
IST Austria, 3400 Klosterneuburg, Austria

Abstract

Structural support vector machines (SSVMs) are amongst the best performing models for many structured computer vision tasks, such as semantic image segmentation or human pose estimation. Training SSVMs, however, is computationally costly, since it requires repeated calls to a structured prediction subroutine (called $\max$-oracle), which requires solving an optimization problem itself, e.g. a graph cut. In this work, we introduce a new technique for SSVM training that is more efficient than earlier techniques when the $\max$-oracle is computationally expensive, as it is frequently the case in computer vision tasks. The main idea is to combine the recent stochastic Block-Coordinate Frank-Wolfe method with efficient hyperplane caching and to use an automatic selection rule for deciding whether to call the $\max$-oracle or to rely on one of the cached hyperplanes. We show experimentally that this strategy leads to faster convergence to the optimum with respect to the number of requires oracle calls, and that this also translates into faster convergence with respect to the total runtime for cases where the $\max$-oracle is slow compared to the other steps of the algorithm. A publicly available C++ implementation is provided.

1 Introduction

Many computer vision problems have a natural formulation as structured prediction tasks: given an input image the goal is to predict a structured output object, for example a segmentation mask or a human pose. Structural support vector machines (SSVMs) [24, 26], are currently one of the most popular methods for learning models that can perform this task from training data. In contrast to ordinary support vector machines (SVMs) [5], which only predict single numbers, e.g. class labels, SSVMs are designed such that, in principle, they can predict arbitrary structured objects. However, this increased flexibility comes at a cost: SSVM training requires solving a more difficult optimization problem than ordinary SVM training. In particular, it requires repeated runs of the structured prediction step (the so called $\max$-oracle) across the training set. Each of these steps is an optimization problem itself, e.g. finding the minimum energy labeling of a graph for image segmentation tasks, and often computationally costly. The more challenging the problem, the more the $\max$-oracle calls become a computational bottleneck. This is the main reason why SSVMs are typically only used for problems with small and medium-sized training sets, not for large scale training as it is common these days, for example, in object categorization [7] and object detection [6].

In this work, we introduce a new variant of the Frank-Wolfe algorithm [8] that is specifically designed for training SSVMs in situations where the $\max$-oracle calls are the computational bottleneck. It extends the recently proposed block-coordinate Frank-Wolfe (BCFW) algorithm [15] by introducing a caching mechanism that keeps multiple potential hyperplanes in memory. In each step of the optimization, the algorithm decides whether to call the exact, but slow, $\max$-oracle, or to reuse one of the hyperplanes from the cache. The first option might allow the algorithm to make a larger steps towards the optimum, while the second option will make smaller steps, but this can be justified because many more steps can be made within the same time window. This suggests that a
trade-off between both options would be optimal, and one of the contribution of the manuscript is a geometrically motivated criterion for dynamically deciding which choice is more promising at any time.

We report on experiments on three different datasets that reflect a range of structured prediction scenarios: first, multiclass classification, where the max-oracle is a trivial lookup. Second, sequence labeling, where the max-oracle can be solved using dynamic programming and is therefore still very efficient. Third, semantic image segmentation, where the max-oracle consists requires minimizing a submodular energy, which is a computationally costly step, even when relying on the efficient graphcut algorithm [4].

2 Preliminaries and Related Work

In structured prediction, the task is to predict structured output objects, \( y \in \mathcal{Y} \), for given inputs, \( x \in \mathcal{X} \). Structural Support Vector Machines (SSVMs) [24, 26] provide a principled method for learning a structured prediction function, \( h : \mathcal{X} \rightarrow \mathcal{Y} \), by generalizing the maximum margin training of binary SVMs to structured spaces.\(^1\) Let \( \phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d \) be a joint feature function of inputs and outputs. Then we parameterize \( h(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle \), for a weight weights, \( w \in \mathbb{R}^d \), that can be learned from a training set, \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \).

The SSVM does so by solving the following convex optimization problem:

\[
\min_w \lambda \|w\|^2 + \sum_{i=1}^{n} H_i(w),
\]

where \( \lambda \geq 0 \) is a regularization parameter. \( H_i(w) \) is the (scaled) structured Hinge loss that generalizes the SVM’s Hinge loss. It is defined as

\[
H_i(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \left\{ \Delta(y_i, y) - \langle w, \phi(x_i, y) \rangle - \phi(x_i, y) \right\},
\]

for a task-specific loss function, \( \Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \), for example, the Hamming loss for many image segmentation tasks. Computing the value of \( H_i(w) \), or the label that realizes this value, requires solving an optimization problem over the label space. We refer to the procedure to do so as the max-oracle, or just oracle. Other names for this in the literature are loss-augmented inference, or just the arg max step. It depends on the problem at hand how the max-oracle is implemented. We discuss three choices and their properties in the appendix.

Structured SVMs are amongst the state-of-the-art choice for numerous complex computer vision tasks, including human pose estimation [11, 28], semantic image segmentation [24, 18, 3], scene reconstruction [9, 20] and tracking [10, 17]. In this work, we concentrate not on the question if SSVMs learn better predictors than other methods, but we study Equation (1) purely from the point of a challenging optimization problem. We are interested in the question how fast for given data and parameters we can find the optimal, or a close-to-optimal, solution vector \( w \). This is relevant, since it has been observed that training structured SVMs is computationally very costly, especially in a computer vision context where the output space \( \mathcal{Y} \) is very large and has a combinatoric structure [18], and this is one of the main reasons why SSVM training is not performed on a similar large scale regime as, for example, ordinary SVM training.

2.1 Related Work

Many algorithms have been proposed to solve the optimization problem [4] or equivalent formulations. In [26], where the problem was originally introduced, Tsochantaridis et al. relied on a formulation as quadratic program

---

\(^1\)In principle, different inputs, \( x \in \mathcal{X} \), could have different output spaces, \( \mathcal{Y}(x) \). This allows, e.g., the prediction of graph labelings for graphs of different sizes. We suppress this aspect and write just \( \mathcal{Y} \) to keep the notation more compact.
(QP) that is equivalent to (1) but contains slack variables and a large number of linear constraints. The authors introduce a cutting-plane algorithm that alternates between calling the max-oracle once for each training example and solving a QP with a subset of contraints (cutting planes) obtained from the oracle. The algorithm was proved to reach a solution $\epsilon$-close to the optimal within $O\left(\frac{1}{\epsilon^2}\right)$ step, i.e. $O\left(\frac{1}{\epsilon^2}\right)$ calls to the max-oracle. Joachims et al. in [13] improve this bound by introducing the so-called one-slack formulation. It is also based on finding cutting planes, but keeps their number much smaller, achieving an improved convergence rate of $O\left(\frac{1}{\epsilon^2}\right)$. The same convergence rate can also be achieved using bundle methods [22].

An alternative approach lies in applying the subgradient method [22] directly to the objective (1), which also allows stochastic and online training [19]. As is common for (stochastic) subgradient-based optimization, one can prove fast convergence to a close-to-optimal solutions. However, the speed of convergence depends crucially on the choice of a learning rate, and this often makes subgradient-based SSVM training less appealing for practical tasks.

The Frank-Wolfe algorithm (FW) [8] is an elegant alternative: it resembles subgradient methods in the simplicity of its updates, but does not require a manual selection of the step size. More recently, Lacoste-Julien et al. introduced a block-coordinate variant of the Frank-Wolfe algorithm [15]. It is more efficient for SSVM training, because it makes use of the fact that the SSVM objective is not arbitrary but can be decomposed additively into $n$ terms, each of which is structured. The authors showed experimentally that the BCFW algorithm is usually much more efficient than the original FW algorithm as well as the earlier cutting plane or subgradient-based optimization, and it can be considered the current state-of-the-art of SSVM training.

In the next section, we show how to further improve the BCFW algorithm, when the algorithm’s runtime is limited by the calls to the max-oracle, as it is commonly the case for computer vision tasks.

## 3 Multi-Plane Block-Coordinate Frank-Wolfe Algorithm

In this section we introduce our main contribution, the multi-plane block-coordinate Frank-Wolfe (MP-BCFW) algorithm for SSVM training. Since it is based on the FW and BCFW methods, we start by giving a more detailed explanation of the working mechanisms of these two algorithms, and then we highlight the improvements we make to tackle the situation when the max-oracle is computationally very costly.

First, we rewrite the structured Hinge loss term, $H_i(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \{\Delta(y_i, y) - \langle w, \phi(x_i, y_i) - \phi(x_i, y) \rangle\}$, more compactly as

$$H_i(w) = \max_{y \in \mathcal{Y}} \langle \varphi^y, [w \ 1] \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathbb{R}^{d+1}$ and $[w \ 1]$ is the concatenation of $w$ with a single 1 entry. For a vector $\varphi \in \mathbb{R}^{d+1}$ we denote its first $d$ components as $\varphi_* \in \mathbb{R}^d$ and its last component as $\varphi_0 \in \mathbb{R}$. The data vector $\varphi^y$ in (3) for $i = 1, \ldots, n$ and $y \in \mathcal{Y}$ is given by $\varphi^y_* = \frac{1}{n}(\phi(x_i, y) - \phi(x_i, y_i))$ and $\varphi^y_0 = \frac{1}{n}\Delta(y, y)$. Note that $\langle \varphi^y, [w \ 1] \rangle = \langle \varphi^y_*, w \rangle + \varphi^y_0$.

### 3.1 Frank-Wolfe algorithm

We start by reviewing the Frank-Wolfe (FW) algorithm that solves the SSVM training problem in the dual. Writing $H(w) = \sum_{i=1}^n H_i(w)$ and introducing concatenated vectors $\varphi^{y_1, \ldots, y_n} = \sum_{i=1}^n \varphi^y_i$, the primal problem can be written as

$$\min_w \frac{\lambda}{2} ||w||^2 + H(w), \quad \text{with} \quad H(w) = \max_{y \in \mathcal{Y}} \langle \varphi^y, [w \ 1] \rangle,$$

for $\mathcal{Y} = \mathcal{Y} \times \cdots \times \mathcal{Y}$. Note that evaluating $H(w)$ for a given $w$ requires $n$ calls to the max-oracle, one for each of the terms $H_1(w), \ldots, H_n(w)$.
Algorithm 1 \[\text{Frank-Wolfe algorithm for the dual of } (4)\]

1. set \( \varphi \leftarrow \varphi^y \) for some \( y \in \mathcal{Y} \)
2. repeat until some stopping criterion
3. compute \( w \leftarrow \arg \min_w \frac{\lambda}{2}||w||^2 + \langle \varphi, [w \ 1] \rangle \); the solution is given by \( w = -\frac{1}{\lambda}\varphi^* \)
4. call oracle for vector \( w \): compute \( \hat{\varphi} \leftarrow \arg \max_{\varphi^y: y \in \mathcal{Y}} \langle \varphi^y, [w \ 1] \rangle \)
5. compute \( \gamma \leftarrow \arg \max_{\gamma \in [0,1]} \mathcal{F}((1 - \gamma)\varphi + \gamma \hat{\varphi}) \) as follows:
   - set \( \gamma \leftarrow \langle \varphi^* - \hat{\varphi}, \varphi \rangle - \lambda \langle \varphi^* - \hat{\varphi}, \varphi \rangle ||\varphi^* - \hat{\varphi}||^2 \) and clip \( \gamma \) to \([0, 1] \)
   - set \( \varphi \leftarrow (1 - \gamma)\varphi + \gamma \hat{\varphi} \)
6. end repeat

The FW algorithm maintains a plane specified by a vector \( \varphi \in \mathbb{R}^{d+1} \) that is a lower bound on \( H(\cdot) \): \( \langle \varphi, [w \ 1] \rangle \leq H(w) \) for all \( w \in \mathbb{R}^d \). Such a plane exists, because \( H \) is convex, and it can in fact be obtained by taking any convex combination of planes \( \varphi^y \) for \( y \in \mathcal{Y} \). We thus have \( \varphi = \sum_{y \in \mathcal{Y}} \alpha_y \varphi^y \) where \( \sum_{y \in \mathcal{Y}} \alpha_y = 1 \) and \( \alpha \geq 0 \). Vectors \( \varphi \) of this form will be called feasible. Note, that in an implementation the coefficients \( \alpha_y \) do not need to be stored, it suffices to have access to the resulting vector \( \varphi \). Any feasible \( \varphi = [\varphi^*, \varphi_0] \) gives a lower bound on problem \((4)\) as

\[
\mathcal{F}(\varphi) = \min_w \left\{ \frac{\lambda}{2}||w||^2 + \langle \varphi, [w \ 1] \rangle \right\} = -\frac{1}{2\lambda}||\varphi^*||^2 + \varphi_0.
\]

To obtain the tightest bound, we need to maximize \( \mathcal{F}(\varphi) \) over all feasible vectors \( \varphi \). This maximization problem is the dual to \((4)\). The Frank-Wolfe algorithm in pseudo code is shown in Algorithm 1 and is illustrated in Fig. 1. Each iteration monotonically increases \( \mathcal{F}(\varphi) \) (unless the maximum is reached), and the algorithm converges to an optimal solution with the rate \( O(\frac{1}{k}) \) with respect to the number of iterations, i.e. \( O(\frac{n}{k}) \) oracle calls \([16]\).

### 3.2 Block-coordinate Frank-Wolfe algorithm

The recent block-coordinate Frank-Wolfe algorithm \([15]\) for solving problem \((1)\) improves over the FW algorithm by making use of the additive structure of the objective \((4)\). Instead of keeping a single plane, \( \varphi \), it maintains \( n \) planes, \( \varphi^1, \ldots, \varphi^n \), such that the \( i \)-th plane is a lower bound on \( H_i : \langle \varphi^i, [w \ 1] \rangle \leq H_i(w) \) for all \( w \in \mathbb{R}^d \). Each such plane is obtained as a convex combination of the planes that define \( H_i(\cdot) \), i.e. \( \varphi^i = \sum_{y \in \mathcal{Y}} \alpha_{iy} \varphi^y \) where \( \sum_{y \in \mathcal{Y}} \alpha_{iy} = 1 \) and \( \alpha \geq 0 \).

A vector \((\varphi^1, \ldots, \varphi^n)\) where each \( \varphi^i \) as above is called feasible. For a feasible vector the sum \( \varphi = \sum_{i=1}^n \varphi^i \) defines a plane which is a lower bound on \( H(w) = \sum_{i=1}^n H_i(w) \), i.e. \( \langle \varphi, [w \ 1] \rangle \leq H(w) \) for all \( w \in \mathbb{R}^d \).
bound $F$ vector $\Theta$(efforts in each BCFW step are very unbalanced: for each oracle call (line 5 in Alg. 2) there is only
However, it still has one suboptimal feature that suggests that it can be improved upon. Namely, the computation
than the FW algorithm as well as earlier approaches, such as the cutting plane and stochastic subgradient methods.
It has been shown in [15] that for training SSVMs, BCFW needs much fewer passes through the training data
of additional work (lines 4 and 6), and this is often negligible compared to the time taken by the oracle. We could
easily afford to do more work per oracle call without significantly changing the running time of one iteration. Our
goal is therefore to exploit this extra freedom to accelerate convergence, thereby decreasing the number of required
oracle calls and the overall runtime of the algorithm.

The main insight is that BCFW acts suboptimally by discarding the plane $\hat{\varphi}^i$ after completing the iteration for
term $H_i$, even though it had made an expensive call to the max-oracle to obtain $\hat{\varphi}^i$. We propose to retain some of

Algorithm 2

| Line | Description |
|------|-------------|
| 1:   | for each $i \in [n]$ set $\varphi^i \leftarrow \varphi^{iy}$ for some $y \in \mathcal{Y}$, and set $\varphi = \sum_{i=1}^{n} \varphi^i$ |
| 2:   | repeat until some stopping criterion |
| 3:   | pick $i \in [n]$, e.g. uniformly at random |
| 4:   | compute $w \leftarrow \arg\min \frac{1}{2}||w||^2 + \langle \varphi, [w \ 1] \rangle$; the solution is given by $w = -\frac{1}{\lambda} \varphi_\ast$ |
| 5:   | call $i$-th oracle for vector $w$: compute $\hat{\varphi}^i \leftarrow \arg\max_{\varphi^{iy}, y \in \mathcal{Y}} \langle \varphi^{iy}, [w \ 1] \rangle$ |
| 6:   | compute $\gamma \leftarrow \arg\max_{\gamma \in [0,1]} \mathcal{F}(\varphi - \varphi^i + (1 - \gamma)\varphi^i + \gamma \hat{\varphi}^i)$ as follows: |
|      | $\bullet$ set $\gamma \leftarrow \frac{\langle \varphi^i - \hat{\varphi}^i, \varphi^i \rangle - \lambda \langle \varphi^i - \hat{\varphi}^i \rangle}{||\varphi^i - \hat{\varphi}^i||^2}$ and clip $\gamma$ to $[0, 1]$ |
|      | set $\varphi^{i\text{old}} \leftarrow \varphi^i$, $\varphi^i \leftarrow (1 - \gamma)\varphi^i + \gamma \hat{\varphi}^i$, $\varphi \leftarrow \varphi + \varphi^i - \varphi^{i\text{old}}$ |
| 7:   | end repeat |

Therefore, $\mathcal{F}(\varphi)$ is a lower bound on problem (1), where the function $\mathcal{F}$ is defined by (5). The goal is now to maximize this bound over feasible vectors $(\varphi^1, \ldots, \varphi^n)$.

The block-coordinate strategy from [15] picks an index $i \in [n] = \{1, \ldots, n\}$ and updates the component $\varphi^i$ while keeping all other components fixed. During this step the terms $H_j(w)$ for $j \neq i$ are approximated by linear functions $\langle \varphi^j, [w \ 1] \rangle$. The algorithm then tries to find a different linear approximation for $H_i(\cdot)$ that gives a larger bound $\mathcal{F}(\sum_{j=1}^{n} \varphi^j)$. Pseudo code for BCFW is given in Algorithm 2. For efficiency reasons it also maintains a vector $\varphi = \sum_{i=1}^{n} \varphi^i$.

3.3 Multi-Plane Block-Coordinate Frank-Wolfe

It has been shown in [15] that for training SSVMs, BCFW needs much fewer passes through the training data than the FW algorithm as well as earlier approaches, such as the cutting plane and stochastic subgradient methods. However, it still has one suboptimal feature that suggests that it can be improved upon. Namely, the computation efforts in each BCFW step are very unbalanced: for each oracle call (line 5 in Alg. 2) there is only $\Theta(d)$ amount of additional work (lines 4 and 6), and this is often negligible compared to the time taken by the oracle. We could easily afford to do more work per oracle call without significantly changing the running time of one iteration. Our goal is therefore to exploit this extra freedom to accelerate convergence, thereby decreasing the number of required oracle calls and the overall runtime of the algorithm.

The main insight is that BCFW acts suboptimally by discarding the plane $\hat{\varphi}^i$ after completing the iteration for term $H_i$, even though it had made an expensive call to the max-oracle to obtain $\hat{\varphi}^i$. We propose to retain some of
these planes, maintaining a “working set” \( \mathcal{W}_i \) for each \( i = 1, \ldots, n \). Similar to \([13]\), whenever the oracle for \( H_i \) is called, the obtained plane \( \tilde{\varphi}^i \) is added to \( \mathcal{W}_i \). After a certain time, planes are removed again from \( \mathcal{W}_i \), unless in the mean time they have become active (see below). Consequently, at any iteration the algorithm has access to multiple planes instead of just one, which is why we name the proposed algorithm multi-plane block-coordinate Frank-Wolfe (MP-BCFW).

The working set, \( \mathcal{W}_i \), allows us to define an alternative mechanism for increasing the value of \( \mathcal{F} \), i.e. bringing the parameter vector closer to its optimal value, that does not require a call to the costly max-oracle. We define an approximation \( \tilde{H}_i(w) \) of \( H_i(w) \) by

\[
\tilde{H}_i(w) = \max_{\tilde{\varphi}^i \in \mathcal{W}_i} \langle \tilde{\varphi}^i, [w \, 1] \rangle
\]

Note that \( \tilde{H}_i(w) \leq H_i(w) \) for all \( w \in \mathbb{R}^d \), since the maximization is performed over a smaller set of planes. For any \( i \), we can run BCFW updates where term \( H_i \) is replaced with term \( \tilde{H}_i \), i.e. in step 5 of Algorithm\([2]\) we set \( \tilde{\varphi}^i \leftarrow \arg \max_{\tilde{\varphi}^i \in \mathcal{W}_i} \langle \tilde{\varphi}^i, [w \, 1] \rangle \). Such approximate oracle steps will increase \( \mathcal{F}(\varphi) \), but potentially less so than a BCFW update using the exact expression.\(^2\)

We propose to interleave the approximate updates steps with exact updates. The order of operations in our current implementation is shown in Algorithm\([3]\). We refer to one pass through the data in lines 3 and 4 as an exact and an approximate pass respectively, and to steps 3-5 as an (outer) iteration. Thus, each iteration contains 1 exact pass and up to \( M \) approximate passes. The parameter \( N \) bounds the number of stored planes per term: \( |\mathcal{W}_i| \leq N \) for each \( i \). In this algorithm a plane is considered active at a given moment if it returned as optimal by either an exact or an approximate oracle call.

The complexity of one approximate update in step 4 is \( O(Nd) \), therefore the algorithm performs \( O(MNd) \) additional work per each oracle call. For \( M = N = 0 \), MP-BCFW reduces to the standard BCFW algorithm. Since MP-BCFW in particular performs all steps that BCFW does, it inherits all convergence guarantees from the earlier algorithm, such as a convergence rate of \( O(\frac{1}{t}) \) towards the optimum, as well as the guarantee of convergence even when the max-oracle can solve the problem only approximately (see \([15]\) for details). However, as we will show in Section\([4]\) it gets more “work done” per iteration and therefore converges faster with respect to the number of max-oracle calls.

### 3.4 Automatic Parameter Selection

The optimal number of planes to keep per term as well as the optimal number of efficient approximate passes to run depends on multiple factors, in particular the number of support vectors and how far the current solution still is from the optimal one. Therefore, we propose not to set these parameters to fixed values but to adapt them dynamically over time in a data-dependent way. The first criterion described below is fairly standard \([13]\), but the second one is specific to MP-BCFW and is a second contribution of this work.

**Parameter \( N \)** First, we notice that size of the working set is bounded not only by \( N \), but also by the mechanisms of automatically removing inactive planes. We therefore set \( N \) to a very large value, and rely on \( T \) to control the working set. This has the effect that the actual number of planes is adjusted in a data-dependent way for each training instance. In particular, for easy terms with just a few relevant planes (support vectors) the algorithm uses only those planes. This also accelerates the algorithm, since the approximate oracle calls have runtime proportional to the actual number of planes in their working set, rather than to the maximal number.

\(^2\)Note that the function \( \langle \varphi^i, [w \, 1] \rangle \) may not be a lower bound on \( \tilde{H}(w) \): the plane \( \varphi^i \) is a convex combination of planes \( \{ \varphi^{iy} : y \in \mathcal{Y} \} \), but some of these planes may have been removed from the working set \( \mathcal{W}_i \). However, this property is not required for the correctness of the method. Indeed, it follows from the construction that in Algorithm\([3]\) vector \( \varphi^i \) is always convex combination of planes \( \{ \varphi^{iy} : y \in \mathcal{Y} \} \) for each \( i \), and each step is guaranteed not to decrease the bound \( \mathcal{F}(\sum_{i=1}^n \varphi^i) \). In particular, the convergence properties proven in \([13]\) still hold.
Algorithm 3 Multi-Plane Block-Coordinate Frank-Wolfe (MP-BCFW). Parameters: $N, M, T$.

1: for each $i \in [n]$ set $\varphi^i \leftarrow \varphi^i y$ for some $y \in \mathcal{Y}$, and set $\varphi = \sum_{i=1}^{n} \varphi^i$

   set $\mathcal{W}_i = \{\varphi^i\}$ for each $i \in [n]$ (or $\mathcal{W}_i = \emptyset$ if $N = 0$)

2: repeat until some stopping criterion

3: do one pass through $i \in [n]$ in random order, and for each $i$ do the following:
   • run BCFW update using original term $H_i(w)$
   • add obtained vector $\hat{\varphi}^i$ to $\mathcal{W}_i$
   • if $|\mathcal{W}_i| > N$ then remove plane from $\mathcal{W}_i$ that was inactive the longest time

4: do $M$ passes (or less - see text) through $i \in [n]$ in random order, and for each $i$ do the following:
   • run BCFW update using approximate term $\tilde{H}_i(w) = \max_{\hat{\varphi}^i \in \mathcal{W}_i} \langle \hat{\varphi}^i, [w 1] \rangle$
   • remove planes from $\mathcal{W}_i$ that have not been active during the last $T$ outer iterations

5: end repeat

Parameter $M$ This parameter controls the number of approximate passes in a single iteration. We replace it by the following geometrically motivated criterion. After each approximate pass we compute two quantities: 1) the increase in $F(\varphi)$ per time unit (i.e. the difference in function value divided by the runtime) of the last approximate pass, and 2) the increase in $F(\varphi)$ per unit time of the complete sequence of steps since the beginning of the current iteration (which includes the last exact pass). If the former value is smaller than the latter, we end the approximate passes, and start a new iteration with an exact pass.

This criterion can be best understood as an extrapolation of the recent behavior of the runtime-vs-function value graph into the future. If the slope of the last segment is higher than the slope of the concatenated segments since the beginning of the current iteration, then the expected increase from another approximate pass is high enough. Otherwise, it is more promising to start a new cycle from the beginning, i.e. perform an exact pass next.

3.5 Implementation with inner products

Computer vision problem often have not only many training examples, but also high-dimensional and dense feature vectors. In MP-BCFW one can try to mitigate the effect of high feature dimensionality by caching the values of inner products $\langle \hat{\varphi}^i, \tilde{\varphi}^i \rangle$ for $\hat{\varphi}^i, \tilde{\varphi}^i \in \mathcal{W}_i$.

To make use of these products, we implemented the following scheme. We run BCFW updates with approximate term $\tilde{H}_i$ for a given $i$ not once but multiple times (10 in our experiments). In the first step we compute products $\langle \varphi^i, \varphi^i \rangle$, $\langle \varphi^i, \varphi^i \rangle$, $\langle \varphi^i, \varphi^i \rangle$, $\langle \varphi^i, \varphi^i \rangle$ for all $\hat{\varphi}^i \in \mathcal{W}_i$. These values are then incrementally recomputed using products $\langle \hat{\varphi}^i, \tilde{\varphi}^i \rangle$; the latter are computed on demand and cached. Once all products $\langle \hat{\varphi}^i, \tilde{\varphi}^i \rangle$ have been cached, each step becomes very efficient, namely $\Theta(|\mathcal{W}_i|)$.

Since the inner product values could also be the result of kernelization, we refer to this procedure also as the caching of kernel values, as is it common for accelerating non-linear SVMs [12].

3.6 Weighted averaging of iterates

Following [15], we also tested taking weighted averages of iterates. If $\varphi^{(k)}$ for $k = 1, 2, \ldots$ is the vector produced after the $k$-th oracle call in Algorithm 2 (BCFW) then the $k$-th averaged vector is defined by $\bar{\varphi}^{(k)} = \sum_{i=1}^{k} t_i \varphi^{(i)}$. Note that it can be computed incrementally via $\bar{\varphi}^{(k+1)} = \frac{k}{k+1} \bar{\varphi}^{(k)} + \frac{1}{k+1} \varphi^{(k+1)}$ for $k \geq 1$.

According to experiments in [15], the primal objective of the iterates $\bar{\varphi}^{(k)} = -\frac{1}{\lambda} \varphi^{(k)}$ typically converges to the optimum significantly faster than that of the iterates $\tilde{w}^{(k)} = -\frac{1}{\lambda} \varphi^{(k)}$ (though the effect of averaging on the dual objective function was detrimental, as we observed in our tests). In the context of the stochastic subgradient descent
method it has also been shown in [16, 21] that such averaging improves the convergence rate from $O((\log k)/k)$ to $O(1/k)$.

We extended this scheme to MP-BCFW by maintaining two vectors, $\tilde{\varphi}(k)$ and $\tilde{\varphi}(k')$. They are updated after every exact and every approximate oracle call respectively, using the formula above. When we need to extract a solution, we compute the interpolation between $\tilde{\varphi}(k)$ and $\tilde{\varphi}(k')$ that gives the best lower bound. Such scheme was motivated by the fact that the two types of oracle calls have quite different characteristics, and thus may require different weights.

We will refer to the versions that use the averaging as BCFW-avg and MP-BCFW-avg.

4 Experiments

We analyze the effectiveness of the MP-BCFW algorithm by performing experiments in three different settings. The first two are based on standard datasets that were used previously to benchmark SSVM training (multi-class classification on the USPS dataset and sequence labeling on the OCR dataset). The third task (semantic image segmentation on part of the HorseSeg dataset) shows the characteristic properties of computer vision tasks, in particular a max-oracle that requires much more computation than in the previous two cases, forming a strong computational bottleneck for training. Exact details of dataset characteristics and feature representations for the three scenarios are provided in the appendix.

We focus on comparing MP-BCFW with BCFW (with and without averaging), since [15] already showed a significant improvement of BCFW over earlier algorithms, in particular ordinary FW, cutting plane training and stochastic subgradient training. Results for the variant of MP-BCFW with precomputed inner products did not differ much from the regular algorithm, so we do not report them here. Since all algorithms solve the same convex optimization problem and will ultimately arrive at the same solution, we are mainly interested in the convergence speed, not in the error rates of the resulting predictors. This allows an easy experimental setup in which we can make use of all available training data for learning and do not have to perform model selection, such as by cross-validation. Furthermore, it is known that for a reasonably chosen regularization parameter, $\lambda$, the test error usually decreases monotonically during the optimization, such that a faster converging method is preferable, as it allow reaching a better predictor within a given time budget. For our experiments we use $\lambda = 1/n$, which is consistent with the earlier work [15].

For all algorithms we measure three quantities: 1) the primal suboptimality (the difference between primal objective and the highest observed lower bound we observe during any of our experiments), 2) the dual suboptimality (the highest observed lower bound minus the dual objective), and 3) the duality gap (the difference between primal and dual objective). We report the results in coordinate frames: 1) with respect to the number of calls to the max-oracle, and 2) with respect to the actual runtime. The first quantity, which we refer to as oracle convergence, measures how efficiently the algorithm uses the statistical information that is present in the training examples. It is implementation independent and therefore comparable between publications. The second value, called runtime convergence, is of practical interest, because it directly reflects the computational resources required to achieve a certain solution quality. However, it depends on the concrete implementation and hardware. To nevertheless obtain fair runtime comparisons, we use the same code base for both methods, making use of the fact that BCFW can be recovered from MP-BCFW with minimal overhead by deactivating the working sets and approximate passes ($N = 0, M = 0$). For MP-BCFW we use $T = 10$, $N = 1000$, $M = 1000$ as parameters, where the latter two are just large values that do not influence the system’s behaviour, because of the automatic selection criteria.

---

[16] http://www-i6.informatik.rwth-aachen.de/~keysers/usps.html
[17] http://www.seas.upenn.edu/~taskar/ocr/
[18] http://www.ist.ac.at/~akolesnikov/HDSeg/

*Our C++ implementation is available at [1]. All experiments are performed on the same workstation with 3.3GHz Intel Xeon CPU.
4.1 Results

Figure 3 shows the oracle convergence results for the three datasets. One can see that with the same number of calls to the max-oracle, MP-BCFW is achieves lower primal and dual suboptimality as well as duality gap than BCFW is. This effect is stronger for HorseSeg and OCR than for USPS, which makes sense, since the latter has a very small label space (|Y| = 10), so the number of support vectors per example is small, and the benefit of having access to more than one plane is also limited. OCR and HorseSeg, as graph labeling tasks, have a larger label space, so one can expect more support vectors to contribute to the score. Reusing planes that have been found earlier during the optimization can be seen to have a significantly beneficial effect.

Figure 4 illustrates the runtime convergence, i.e. the horizontal axis shows the actual runtime instead of the number of oracle calls. Here one can see that for the USPS and OCR dataset, the better oracle convergence actually does not translate to actually faster convergence. This can be explained by the characteristics of the optimization problem: in both cases, the max-oracle calls are not the absolute computational bottlenecks.

In the USPS case, each call to max-oracle takes only approximate 20ms on the hardware we used. Overall the
Figure 4: Runtime convergence on the USPS (top), OCR (middle) and HorseSeg (bottom) datasets. Shaded areas indicate minimum and maximum values over 10 repeats.

BCFW algorithm spends approximately 15% of its runtime on oracle calls. In the OCR case, each max-oracle call takes approximate 300ms, for an overall 60% of BCFW’s runtime. In both settings, the ratio of time spent for the oracle calls and time spent elsewhere is not high enough to introduce the overhead of keeping the working sets and repeatedly solving using the approximate oracle. Instead, one achieve the same effect by simply running BCFW for more iterations. Note, however, that MP-BCFW is not slower in its convergence than BCFW, either. We interpret this fact as a sign that the automatic parameter selection of Section 3.4 does its job as intended: if the overhead of a large working set and many approximate passes would be bigger than the benefit they offer, the selection rule falls back to the established (and efficient) baseline.

The situation is very different for the HorseSeg dataset that we take as a more realistic example of a typical computer vision task. Its max-oracle consists of solving a network flow problem, which, even when implemented efficiently, requires on average 2.2 seconds per call. Overall, calls to the max-oracle account for almost 99% of the training time when using the BCFW algorithm. With MP-BCFW, this ratio drops to approximately 25%, because the parameter selection mechanism determines that the time it takes to make one exact approximate oracle call is
often spent better making several calls to the approximate oracle. At the same time, MP-BCFW achieves a much larger reduction of the duality gap per time unit than BCFW does.

4.2 Effect of weight vector averaging

Figure 3 and 4 also give us insight into the effect of weight vector averaging for BCFW and MP-BCFW. Indeed, one can see clearly that for the USPS and OCR data averaging significantly reduces the primal suboptimality. However, this comes at the cost of increasing the dual suboptimality, which is understandable, since the MP-BCFW and BCFW are designed to find solution with highest possible dual value, and averaging moves the coefficient away from this. Overall, the positive and negative effects of averaging are on the same order of magnitude, so in terms of the duality gap, there is little difference between averaging and non-averaging techniques.

For the HorseSeg dataset, averaging had little effect on the primal objective, but the negative effects on the dual were still there, leading to an apparent reduction in convergence speed. In practice, however, we still recommend to activate averaging, since in the context of machine learning, the primal objective is the main object of interest. A second advantage of averaging is that the primal objective of the averaged weight vectors decreases smoothly and monotonically in our experiments (even though there is no guarantee for this), whereas without averaging the typical fluctuations stochastic optimization algorithms are apparent.

4.3 Automatical parameter selection for MP-BCFW

Figure 5 and 6 illustrate the dynamic behavior of the parameter selection for MP-BCFW in more detail. The first figure shows the average size of the working set per term over the course of the optimization. One can see that for the USPS and HorseSeg dataset, after an initial exploration phase the algorithm identifies that only a small number of planes are relevant, and the working set size is reduced accordingly. As a consequence, calls to the approximate oracle are very fast in later iterations, and one can see in the second figure that this leads to the number of approximate passes growing as the algorithm progresses. In the OCR example, a larger number of planes remains relevant throughout the optimization, but the number of approximate passes per exact pass stays relatively low.

5 Summary and Conclusion

We have presented a new algorithm for training structured SVMs that is specifically designed for the situation of computationally costly max-oracles, as they are common in computer vision tasks. The main improvement is the option to re-use previously computed oracle results by means of a per-example working set, and to alternate...
between calls to the exact but slow oracle and calls to an approximate but fast oracle. We also introduce a rule for dynamically choosing the number of approximate passes depending on the algorithm’s runtime behaviour and its progress towards the optimum. Together with the procedure for automatically adjusting the size of the working set, this gives an easy-to-use method whose default settings work for different scenarios. Its C++ implementation is available at [1].

Our experiments showed that the working set optimization always achieves faster convergence with respect to the number of oracle calls. In a situation where the max oracle is the computational bottleneck, this translates to significantly faster convergence also with respect to the actual runtime.

We furthermore introduce the possibility to cache inner products between cutting planes in order to further accelerate calls to the approximate oracle and open the door for kernelization. In our experiments (not reported here), the performances of the schemes with and without caching were similar, but we expect it to be more beneficial for very high dimensional feature spaces. We plan to explore this in future work.

Acknowledgements

We thank Alexander Kolesnikov for his help with the HorseSeg dataset. This work was funded by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 308036 and 616160.

A Appendix

We give details of the three different applications scenarios with max-oracles of increasing complexity.

A.1 Multiclass Classification – USPS Dataset

The USPS dataset is a standard multiclass dataset with 7291 samples of 10 classes, \( \mathcal{Y} = \{0, 1, \ldots, 9\} \). Each sample, \( x \), comes with a 256-dimensional feature vector, \( \psi(x) \), from which we build a 2560-dimensional joint feature map

\[
\phi(x, y) = \left( \psi(x)[y = 0], \ldots, \psi(x)[y = 9] \right).
\] (7)

As loss function we use the ordinary multiclass loss, \( \delta(y, \hat{y}) = [y \neq \hat{y}] \). Overall, the structured Hinge loss is

\[
H_x(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \left\{ [y_i \neq \hat{y}] + \langle w, \phi(x_i, y) - \phi(x_i, y_i) \rangle \right\}
\] (8)

Because of the small label set the max oracle can be performed efficiently by an explicit search over all labels.
A.2 Sequence Labeling - OCR dataset

The OCR dataset consists of 6877 data samples that are sequences of hand-written letters, \( x = (x^1, x^2, \ldots, x^L) \), where for each part, \( x^l \), a feature vector \( \psi(x^l) \in \mathbb{R}^{128} \) is available. The outputs are sequences of labels, \( y = (y^1, y^2, \ldots, y^L) \) of the same length, where \( y^l \in \{a, \ldots, z\} \) for each \( l = 1, \ldots, L \). The length of the sequences in fact differs for different examples with an average of 7.6. We do not indicate this explicitly in order to keep the notation simple.

We define a joint feature map, \( \phi(x, y) = (\phi_u(x, y), \phi_p(x, y)) \), consisting of a unary and a pairwise component:

\[
\phi_u(x, y) = \sum_{l=1}^{L} \phi(x^l, y^l), \quad \phi_p(x, y) = \sum_{l=1}^{L-1} e_{y^l, y^{l+1}},
\]

where \( \phi(x^l, y^l) \) is a multiclass encoding of the feature \( \psi(x^l) \) as defined above for the USPS dataset, and \( e_{a,b} \) is the \( 26^2 \)-dimensional binary vector indicating the label pair \( (a, b) \) out of all possible label transitions. As loss function we use the normalized Hamming loss, \( \Delta(y, \tilde{y}) = \frac{1}{L} \sum_{l=1}^{L} [y^l \neq \tilde{y}^l] \).

The structured Hinge loss can be written using a summation over all positions and transitions:

\[
H_i(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \left\{ \sum_{l=1}^{L} \frac{1}{L} [y_i^l \neq \tilde{y}_i^l] + \langle w_u, \phi(x^l_i, y^l_i) - \phi(x^l_i, \tilde{y}^l_i) \rangle + \sum_{l=1}^{L-1} (w_p, e_{y^l_i, y^{l+1}_i} - e_{y^l_i, \tilde{y}^{l+1}_i}) \right\}, \tag{9}
\]

where \( w = (w_u, w_p) \) is a decomposition of the weight vector into parts acting on the unary and pairwise part of the joint feature map, accordingly. Even though the size of the label space grows exponentially in the length of the sequence, \( |\mathcal{Y}| = 26^L \), the additive structure of the problem makes it possible to solve the max-oracle efficiently using dynamic programming, namely by the Viterbi algorithm [27].

A.3 Graph Labeling – HorseSeg Dataset

We use a subset of 2376 image of the HorseSeg dataset. Each image, \( x \), is decomposed into a variable number of superpixels, \( x^1, \ldots, x^L \), which where obtained from the SLIC algorithm [2]. The average number of superpixels per image is 265. Each superpixel comes with a 649-dimensional feature vector \( \phi(x^l) \). The task consists of predicting a segmentation mask for the image, i.e. a binary label for each superpixel, \( y = (y^1, \ldots, y^L) \) with \( y^l \in \{0, 1\} \) for every \( l = 1, \ldots, L \).

We construct a joint feature map using the same construction as for the unary term in the OCR dataset,

\[
\phi(x, y) = \sum_{l=1}^{L} \phi(x^l, y^l),
\]

In addition, we add a pairwise term to the SSVM prediction function,

\[
\Theta(x, y) = \sum_{k \sim l} [y^k \neq y^l]
\]

where \( k \sim l \) denotes that the superpixels \( k \) and \( l \) are neighbors in the image. For this term, we do not learn a weight vector, but assign it a constant weight of \( 1 \). Formally, the term therefore is not part of the feature vector but contributes to the \( \varphi_o \) component of the parameterization (see Section 3).

As loss function we again use the normalized Hamming loss and we obtain the following structured Hinge loss

\[
H_i(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \left\{ \sum_{l=1}^{L} \frac{1}{L} [y_i^l \neq \tilde{y}_i^l] + \langle w_u, \phi(x^l_i, y^l_i) - \phi(x^l_i, \tilde{y}^l_i) \rangle + \sum_{k \sim l} [y^k \neq y^l] \right\}. \tag{10}
\]
The objective in this discrete optimization is submodular, so the max-oracle can be implemented using the mincut algorithm \cite{4}. This is also the reason that we fix the weight of the pairwise term instead of learning it: in order to ensure submodularity, the pairwise term must stay nonnegative during the training \cite{14}. In practice, this can also be enforced by a positivity constraint on the weight vector component, but it would require a modification to the algorithm to perform constrained instead of unconstrained optimization, which in this work we want to avoid for the sake of simplicity.

References

\begin{itemize}
\item \cite{1} [http://pub.ist.ac.at/~vnk/papers/SVM.html] Our implementation of MP-BCFW.
\item \cite{2} R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. \textit{PAMI}, 34(11):2274–2282, 2012.
\item \cite{3} L. Bertelli, T. Yu, D. Vu, and B. Gokturk. Kernelized structural SVM learning for supervised object segmentation. In \textit{CVPR}, pages 2153–2160, 2011.
\item \cite{4} Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. \textit{PAMI}, 2004.
\item \cite{5} C. Cortes and V. Vapnik. Support-vector networks. \textit{Machine Learning}, 20(3), 1995.
\item \cite{6} T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In \textit{CVPR}, pages 1814–1821. IEEE, 2013.
\item \cite{7} J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale hierarchical image database. In \textit{CVPR}, 2009.
\item \cite{8} M. Frank and P. Wolfe. An algorithm for quadratic programming. \textit{Naval Research Logistics Quarterly}, 3:149–154, 1956.
\item \cite{9} A. Gupta, M. Hebert, T. Kanade, and D. M. Blei. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In \textit{NIPS}, pages 1288–1296, 2010.
\item \cite{10} S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In \textit{ICCV}, pages 263–270, 2011.
\item \cite{11} C. Ionescu, L. Bo, and C. Sminchisescu. Structural SVM for visual localization and continuous state estimation. In \textit{ICCV}, pages 1157–1164, 2009.
\item \cite{12} T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, \textit{Advances in Kernel Methods}, pages 169–184. MIT Press, 1999.
\item \cite{13} T. Joachims, T. Finley, and C. Yu. Cutting-plane training of structural SVMs. \textit{Machine Learning}, 1:2759, 2009.
\item \cite{14} V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? \textit{PAMI}, 2004.
\item \cite{15} S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In \textit{ICML}, 2014.
\end{itemize}
16] S. Lacoste-Julien, M. W. Schmidt, and F. Bach. A simpler approach to obtaining an O(1/t) convergence rate for the projected stochastic subgradient method. CoRR, abs/1212.2002, 2012.

17] X. Lou and F. A. Hamprecht. Structured learning for cell tracking. In NIPS, pages 1296–1304, 2011.

18] S. Nowozin, P. V. Gehler, and C. H. Lampert. On parameter learning in CRF-based approaches to object class image segmentation. In ECCV, pages 98–111. Springer, 2010.

19] N. D. Ratliff, J. A. Bagnell, and M. Zinkevich. Large margin methods for structured and interdependent output variables. In AISTATS, 2007.

20] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun. Efficient structured prediction for 3d indoor scene understanding. In CVPR, pages 2815–2822, 2012.

21] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In ICML, 2013.

22] N. Z. Shor. Minimization methods for nondifferentiable functions. Springer-Verlag, 1985.

23] M. Szummer, P. Kohli, and D. Hoiem. Learning CRFs using graph cuts. In ECCV, pages 582–595, 2008.

24] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In NIPS, 2004.

25] C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. JMLR, 11:311–365, 2010.

26] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. JMLR, 2005.

27] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory, 1967.

28] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In CVPR, pages 2030–2037, 2010.