BCFW for Structured SVMs

Optimization for Data Science

Ian Solagna, Giovanna Pilot, Michele Zanatta

June 25, 2022

Introduction

This project is devolved to implementing a Frank-Wolfe (FW) algorithm for a structural Support Vector Machines (SVM) prediction task. We aim to implement an alternative version of the FW algorithm able to deal with structured SVM. To do that, we focus on the Block-Coordinate variant of the FW algorithm (BCFW). In particular, we will develop a BCFW algorithm where the block sample will be done through a probability distribution related to the values of the blocks' gaps, i.e. the blocks that give larger duality gaps are weighted more, and thus sampled with greater probability.

This paper is divided into two parts.

In section (1) we explore the theory behind the Frank-Wolfe algorithm for structured SVMs. In particular, we describe the structural SVMs prediction task (1.1) and why we need duality to make the problem feasible. We used the theory seen during the 'Optimization for Data Science' to develop the dual problem, along with the Karush-Kuhn-Tucker optimality conditions. Subsection (1.2) goes through the FW algorithm theory, first for a general problem, then for a structured SVMs task (1.2.1), while Subsection (1.3) refers to the Block-Coordinate FW algorithms, maintaining the same structure of the previous one. For this part we referred to Lacoste-Julien et al., 2012 for algorithms' theory, and to Boyd and Vandenberghe, 2004 for duality theory. Finally, for this first module, we describe the adaptive sampling BCFW in the Subsection (1.4), which has been introduced by Osokin et al., 2016.

Through this section, we have proved the results related to the equivalence between FW, and BCFW, algorithms, and their structured counterparts, among with the construction of the Lagrange duality problem. We also reported the results about the convergence of the algorithm. For further details, refer to the already cited papers.

In the second Section (2), we explain the implementation on MATLAB of the algorithms described in the first section, using as a basis the one given by Osokin. In particular, we experimented with these algorithms on a real dataset, named CoNLL 2000, which is described in the first paragraph of subsection (2.1). Also, in this part, we explain the feature map used and the algorithm within the max oracle. Finally, we give the results of our experiment on a training set composed of 1'000 sentences. Due to computational cost, we have not been able to use all the 8'936 sentences in the CoNLL dataset, and for this reason, we could not use the test set. As explained below, the fact that all of the dataset is not used means that not all the feature values and states are explored. In the test dataset, however, all the feature values and states are present. Still, our results are aligned with the one obtained by Osokin et al., 2016.

1 Frank-Wolfe Algorithm for Structured SVMs

In this section, we report the theory of structural Support Vector Machines and the Frank-Wolfe algorithm with the variations analyzed.

1.1 Structural Support Vector Machines

In structured SVMs prediction, the goal is to predict a structured object $y \in \mathcal{Y}(x)$ for a given input $x \in \mathcal{X}$. In the standard approach, a structured feature map $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ encodes the relevant information for input/output pairs, and a linear classifier with parameter w is defined by the linear classifier $h_w(x) = \mathrm{argmax}_{y \in \mathcal{Y}(x)} \langle w, \phi(x, y)\rangle$. Given a labeled training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, w is estimated by solving a convex non-smooth optimization problem

$$\begin{aligned} \min_{w,\xi} \quad & \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & \langle w, \psi_i(y)\rangle \geq L(y_i, y) - \xi_i \quad \forall i, \forall y \in \mathcal{Y}(x_i), \end{aligned} \tag{1}$$

where $\psi_i(y) := \phi(x_i, y_i) - \phi(x_i, y)$ and $L_i(y) := L(y_i, y)$ denotes the task-dependent structured error of predicting output y instead of the observed output y_i. The slack variable ξ_i measures the surrogate loss for the i-th datapoint and λ is the regularization parameter. For the sake of simplicity, we will denote $\mathcal{Y}_i := \mathcal{Y}(x_i)$.

Due to the combinatorial nature of $\mathcal{Y}$, we the problem (1) can have an exponential number of constraints. We can replace the $\sum_i |\mathcal{Y}_i|$ linear constraints with n piecewise-linear ones by defining the structured hinge-loss:

$$\tilde{H}_i(w) := \max_{y \in \mathcal{Y}_\rangle} L_i(y) - \langle w, \psi_i(y)\rangle. \tag{2}$$

The constraints in (1) can thus be replace with the non-linear ones $\xi_i \geq \tilde{H}_i(w)$. In fact,

$$\begin{aligned} & \langle w, \psi_i(y)\rangle \geq L(y_i, y) - \xi_i \quad \forall y \in \mathcal{Y}_i \\ \iff & \xi_i \geq L(y_i, y) - \langle w, \psi_i(y)\rangle \quad \forall y \in \mathcal{Y}_i \\ \iff & \xi_i \geq \max_{y \in \mathcal{Y}_i} L(y_i, y) - \langle w, \psi_i(y)\rangle. \end{aligned}$$

The computation of the structured hinge-loss for each i amounts to finding the most 'violating' output $\boldsymbol{y}$ for a given input $\boldsymbol{x_i}$, a task which can be carried out efficiently in many structured prediction settings, for example, the MAP decoding algorithm which we have used for the experimental results. This problem is called the *loss-augmented decoding* subproblem. For the theoretical part, we will assume to have access to an efficient solver for this subproblem, and we call such solver *maximization oracle.*

Thus, the equivalent non-smooth unconstrained formulation of (1) is:

$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^n \tilde{H}_i(w). \tag{3}$$

Having a maximization oracle allows us to apply subgradient methods to this problem, as a subgradient of $\tilde{H}_i(w)$ with respect to $\boldsymbol{w}$ is $-\boldsymbol{\psi_i(y_i^*)}$, where $\boldsymbol{y_i^*}$ is any maximizer of the loss-augmented decoding subproblem (2).

The Dual In this paragraph we will explain how to derive the dual of (1). We define the Lagrange function

$$\mathcal{L}(x, \alpha) = \underbrace{\frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^n \xi_i}_{f_0(w)} + \sum_{i=1}^n \frac{1}{n}\alpha_i \underbrace{[L(y_i, y) - \xi_i - \langle w, \psi_i(y)\rangle]}_{f_i(W)},$$

where $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{R}^{|\mathcal{Y}_1|} \times ... \times \mathbb{R}^{|\mathcal{Y}_n|} = \mathbb{R}^m$ are the corresponding non-negative Lagrange multipliers or rather the dual variables and we have re-scaled the multipliers by a constant of $\frac{1}{n}$ without loss of generality.

Since the objective as well as the constraints are assumed continuously differentiable with respect to (w, ξ), the Lagrangian $\mathcal{L}$ will attain its finite minimum over α when $\nabla_{(w,xi)}\mathcal{L}(w, \xi, \alpha) = 0$. Thus we have:

$$\frac{\partial}{\partial w}\mathcal{L}(w,\xi,\alpha) = \lambda w - \sum_{i\in[n],y\in\mathcal{Y}_i} \frac{1}{n}\alpha_i(y)\psi(y) = 0$$
$$\iff \lambda w = \sum_{i\in[n],y\in\mathcal{Y}_i} \frac{1}{n}\alpha_i(y)\psi(y)$$

$$\frac{\partial}{\partial \xi}\mathcal{L}(w,\xi,\alpha) = \frac{1}{n} - \sum_{i\in[n],y\in\mathcal{Y}_i} \frac{1}{n}\alpha_i(y) = 0$$
$$\iff \sum_{i\in[n],y\in\mathcal{Y}_i} \alpha_i(y) = 1.$$

Notice that if this last condition is not satisfied, because the Lagrangian is linear in ξ_i, we have that the minimization of the Lagrangian in ξ_i yield $-\infty$ and so these points can be excluded.

At this point we can define the Lagrange dual function $g(\alpha) = \inf_{w\in D}\mathcal{L}(w,\alpha)$ where $D = \cap_{i=0}^m \mathrm{dom} f_i$:

$$\begin{aligned} g(\alpha) &= \frac{\lambda}{2}\| \sum_{i\in[n],y\in\mathcal{Y}_i} \alpha_i(y)\frac{\psi(y)}{\lambda n}\|^2 + \sum_{i\in[n],y\in\mathcal{Y}_i}\frac{1}{n}\alpha_i(y)L_i(y) \\ &= \frac{\lambda}{2}\|A\alpha\|^2 - b^T\alpha \end{aligned}$$

where the matrix $A \in \mathbb{R}^{d\times m}$ consists of the m columns $A := \left\{\frac{1}{\lambda n}\psi_i(y) \in \mathbb{R}^d | i \in [n], y \in \mathcal{Y}_i\right\}$, and the vector $b \in \mathbb{R}^m$ is given by $b := \left(\frac{1}{n}L_i(y)\right)_{i\in[n],y\in\mathcal{Y}_i}$.

Finally, we can write the Lagrange dual problem

$$\begin{aligned} &\min_{\alpha\in\mathbb{R}^m,\alpha\geq 0} f(\alpha) := \frac{\lambda}{2}||A\alpha||^2 - b^T\alpha \\ &\text{s.t.} \sum_{y\in\mathcal{Y}_i}\alpha_i(\mathbf{y}) = 1 \ \forall i \in [n] \\ &\quad \alpha_i(y) \geq 0 \ \forall i \in [n] \quad \text{(condition 3 Theorem 1.1).} \end{aligned} \tag{4}$$

We state the following theorem:

Theorem 1.1 (Karush-Kuhn-Tucker optimality conditions)**.** *Consider the following non linear minimization problem*

$$\begin{aligned} &\max_x f(x) \\ \texttt{s.t.}\quad &g_i(x) \leq 0 \quad i = 1,\dots,m \\ &h_i(x) = 0 \quad i = 1,\dots,p \end{aligned}$$

where f, h and g are differentiable. Let x^* and (λ^*,ν^*) be any primal and dual optimal points with zero duality gap. Since x^* minimizes $\mathcal{L}(x,\lambda^*,\nu^*)$ over x, it follows that:

$$\begin{aligned} g_i(x^*) &\leq 0, \quad i = 1,\dots,m \\ h_i(x^*) &= 0, \quad i = 1,\dots,p \\ \lambda_i^* &\geq 0, \quad i = 1,\dots,m \\ \lambda^* g_i(x^*) &= 0, \quad i = 1,\dots,m \\ \nabla f(x^*) + \sum_{i=1}^m \lambda^*\nabla g_i(x^*) + \sum_{i=1}^p &\nu_i^*\nabla h_i(x^*) = 0. \end{aligned} \tag{5}$$

We note that, given a dual variable vector α, we can use the Karush-Kuhn-Tucker optimality conditions (condition 5 with observation above about the partial derivative w.r.t. ξ_i) to obtain the corresponding variables $w = A\alpha$. Note that the domain $\mathcal{M} \subset \mathbb{R}^m$ is the product of n probability simplices $\mathcal{M} := \Delta_{|\mathcal{Y}_1|} \times \ldots \times \Delta_{|\mathcal{Y}_n|}$.

In the end, the gradient of f is:

$$\nabla f(\alpha) = \lambda A^T A\alpha - b = \lambda A^T w - b. \tag{6}$$

1.2 The Frank-Wolfe algorithm

The Frank-Wolfe iterative optimizers, also known as the conditional gradient method, address the convex optimization problem of the form

$$\min_{\alpha \in \mathcal{M}} f(\alpha) \tag{7}$$

where the convex feasible set $\mathcal{M}$ is compact and the convex objective function f is continuously differentiable. The Frank-Wolfe algorithm is presented as a great alternative to projected gradient algorithms. Indeed, F-W at each iteration considers the linearization of the objective function f and minimizes it over $\mathcal{M}$ searching for a feasible corner $\mathbf{s}$ at the current iterate α. The next iterate is obtained as a convex combination of s and the previous iterate, with step-size γ.

Algorithm 1 Frank-Wolfe on a compact domain

```
Let $\alpha^{(0)} \in \mathcal{M}$
for $k \leftarrow 0$ to $K$ do
    Compute $s := \text{argmin}_{s' \in \mathcal{M}} \langle s', \nabla f(\alpha^{(k)}) \rangle$
    Let $\gamma := \frac{2}{k+2}$ or optimize $\gamma$ by line-search
    Update $\alpha^{(k+1)} := (1-\lambda)\alpha^{(k)} + \gamma s$
end for
```

The explained procedure of the algorithm guarantees some interesting and handy properties. Foremost, every iterates $\alpha^{(k)}$ can be written as a convex combination of the starting point $\alpha^{(0)}$ and the search corners s previously found. Therefore, we obtain a sparse representation of the parameter $\alpha^{(k)}$ which is computationally useful when the dimension of α is exponential. Moreover, convexity of f implies that the linearization $f(\alpha) + \langle s - \alpha, \nabla f(\alpha) \rangle$ always lies below the graph of the function f. We defined the 'linearization duality gap' as:

$$g(\alpha) := \max_{s' \in \mathcal{M}} \langle \alpha - s', \nabla f(\alpha) \rangle = \langle \alpha - s, \nabla f(\alpha) \rangle. \tag{8}$$

Following what we just mentioned above about the lower bound on the value of the yet unknown optimal solution $f(\alpha^*)$ and noticing that at each step the F-W algorithm already calculates the linearization duality gap, this last one turns out to be a certificate for the current approximation quality, i.e. $g(\alpha) \geq f(\alpha) - f(\alpha^*)$. So whenever s minimizes the linearized problem at an arbitrary point α, then this s gives at each iteration the current duality gap $g(\alpha) = \langle \alpha - s, \nabla f(\alpha) \rangle$ as a certificate for the current approximation quality, allowing us to monitor the convergence and helps us to choose the right stopping criterion $g(\alpha^{(k)}) \leq \epsilon$.

1.2.1 Frank-Wolfe for Structured SVMs

An important property of the Frank-Wolfe method is that the linear subproblem employed by Frank-Wolfe is directly equivalent to the loss-augmented decoding subproblem (2) for each data point, which can be solved efficiently. In other words, we define a new algorithm (2) which corresponds to the standard Frank-Wolfe algorithm (1) applied to the SVM dual problem (4). Let's see how. First of all, the following lemma holds:

Lemma 1.1. *The sparse vector $s \in \mathbb{R}^{\mathsf{K}}$ constructed in the inner for-loop of algorithm (2) is an exact solution to $s = argmin_{s' \in \mathcal{M}} \langle s', \nabla f(\alpha^{(k)}) \rangle$.*

Proof. We have to prove that the sparse vector $y_i^* := \text{argmax}_{y \in \mathcal{Y}_\rangle} H_i(y, w^{(k)})$ for $i \in [n]$ is an exact solution of $s = \text{argmin}_{s' \in \mathcal{M}} \langle s', \nabla f(\alpha^{(k)}) \rangle$ for the problem (4).

Because $\mathcal{M} = \Delta_{|\mathcal{Y}_1|} \times \ldots \times \Delta_{|\mathcal{Y}_n|}$ is a product of separable simplices, we can decompose $\min_{s' \in \mathcal{M}} \langle s', \nabla f(\alpha) \rangle$ as $\sum_{i \in [n]} \min_{s_i \in \Delta_{|\mathcal{Y}_i|}} \langle s_i, \nabla f(\alpha) \rangle$ over $\mathcal{M}$, where $\Delta_{|\mathcal{Y}_i|}$ is the simplex with regard to $\mathcal{Y}_i$.

We note now that $\min_{s_i \in \Delta_{|\mathcal{Y}_i|}} \langle s_i, \nabla f(\alpha) \rangle$ is a minimization problem over a simplex of a linear function, so it reduces to search over its corners. Thus, because we have

$$\begin{aligned} f(\alpha) &= \frac{\lambda}{2}\|A\alpha\|^2 - b^T\alpha \\ \nabla f(\alpha) &= \lambda A^T A\alpha - b \\ &= \lambda A^T w - b = -\frac{1}{n} H_i(y; w) \end{aligned}$$

for $i \in [n], y \in \mathcal{Y}_i$, the search amounts for each i to find the minimal component of $-H_i(y; w)$ over $y \in \mathcal{Y}_i$. This is means exactly that we have to solve the loss-augmented decoding problem as used in Algorithm (2) to construct the domain vertex s. □

Thus, applying Algorithm (1) with line search to the 4 but only maintaining the corresponding primal iterates $w^{(k)} := A\alpha^{(k)}$ we obtain the Algorithm (2).

In other words, we have seen that Frank-Wolfe can efficiently be applied to the dual of the structural SVM (4). We recall that the optimization domain for the dual variables α is the product of n simplices $\mathcal{M} = \Delta_{|\mathcal{Y}_1|} \times \ldots \times \Delta_{|\mathcal{Y}_n|}$. Since each simplex has an exponential number of dual variables $|\mathcal{Y}_i|$, we can not maintain a dense vector for α. Frank-Wolfe algorithm already solves this problem because each iterate $\alpha^{(k)}$ is a sparse convex combination of the previously visited corners s and the starting point $\alpha^{(0)}$, and so we only need to maintain the list of previously seen solutions to the loss-augmented decoding subproblems to keep track of the non-zero coordinates of α, avoiding the problem of its exponential size.

Algorithm 2 Batch Primal-Dual Frank-Wolfe for the Structural SVM

```
Let $w^{(0)} := 0, \ell^{(0)} := 0$
for $k \leftarrow 0$ to $K$ do
    for $i \leftarrow 1$ to $n$ do
        Solve $y_i^* := \text{argmax}_{y \in \mathcal{Y}_i} H_i(y; w^{(k)})$
    end for
    Let $w_s := \sum_{i=1}^n \frac{1}{\lambda n} \psi(y_i^*)$ and $\ell_s := \frac{1}{n} \sum_{i=1}^n L_i(y_i^*)$
    Let $\gamma := \frac{\lambda(w^{(k)} - w_s)^T w^{(k)} - \ell^{(k)} + \ell_s}{\lambda||w^{(k)} - w_s||^2}$ and clip to [0,1]
    Update $w^{(k+1)} := (1-\gamma) w^{(k)} + \gamma w_s$
    and $\ell^{(k+1)} := (1-\gamma)\ell^{(k)} + \gamma \ell_s$
end for
```

The 'linearization' gap for the structural SVM dual formulation 4 is given by

$$\begin{aligned} g(\alpha) &:= \max_{s' \in \mathcal{M}} \langle \alpha - s', \nabla f(\alpha) \rangle \\ &= (\alpha - s)^T (\lambda A^T A \alpha - b) \\ &= \lambda (w - As)^T w - b^T \alpha + b^T s \end{aligned} \tag{9}$$

where s is an exact minimizer of the linearized problem given at point α. We have the following remark.

Remark 1.1. *The 'linearization' gap for the structural SVM (Eq. 9) is equivalent to the standard Lagrangian duality gap for the structural primal objective (1).*

Proof. The duality gap is the difference between the values of any primal solutions and any dual solutions. So consider the difference of our objective function at $w := A\alpha$ in the primal problem (3) and the dual objective at α in problem (4) in its maximization version. This difference is

$$\begin{aligned} g_{lag}(w,\alpha) &= \underbrace{\frac{\lambda}{2}w^Tw + \frac{1}{n}\sum_{i=1}^{n}\tilde{H}_i(w)}_{\text{Primal solution}} - \underbrace{\left(b^T\alpha - \frac{\lambda}{2}w^Tw\right)}_{\text{Dual solution}} \\ &= \lambda w^Tw - b^T\alpha + \frac{1}{n}\sum_{i=1}^{n}\max_{y\in\mathcal{Y}_i} H_i(y;w). \end{aligned}$$

Now we recall that by definition of A and b we have that $\frac{1}{n}H_i(y;w) = (b - \lambda A^Tw)_{(i,y)} = (-\nabla f(\alpha))_{(i,y)}$. By summing up over all points and re-using a similar argument as in Lemma (1.1), we get that

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n}\max_{y\in\mathcal{Y}_i} H_i(y;w) &= \sum_{i=1}^{n}\max(-\nabla f(\alpha))_{(i,y)} \\ &= \max_{s'\in\mathcal{M}}\langle s', -\nabla f(\alpha)\rangle, \end{aligned}$$

thus,

$$\begin{aligned} g_{lag}(w,\alpha) &= (\lambda w^TA - b^T)\alpha + \frac{1}{n}\sum_{i=1}^{n}\max_{y\in\mathcal{Y}_i} H_i(y;w) \\ &= \langle\nabla f(\alpha),\alpha\rangle + \max_{s'\in\mathcal{M}}\langle -s', \nabla f(\alpha)\rangle = \langle\alpha - s, \nabla f(\alpha)\rangle = g(\alpha), \end{aligned}$$

as defined in equation (9).

□

Thus, the 'linearization' gap gives us a direct handle on the suboptimality of $w^{(k)}$ for the primal problem (3). Using $w_s := As$ and $l_s := b^Ts$ we observe that the gap is efficient to compute given the primal variables $w := A\alpha$ and $l := b^Ts$, which are maintaned during the run of Algorithm (2). Therefore, we can use the duality gap $g(\alpha^{(k)}) \leq \epsilon$ as a proper stopping criterion.

Concerning the convergence of the Algorithm (2), it obtains an ϵ-approximate solution to the structural SVM dual problem (4) and duality gap $g(\alpha^{(k)}) \leq \epsilon$ after at most $O\big(\frac{R^2}{\lambda\epsilon}\big)$ iterations, where each iteration costs n oracle calls.

R is the maximal length of a difference feature vector, i.e. $R := \max_{i\in[n],y\in\mathcal{Y}_i}||\psi_i(y)||$.

We state that the duality gap is smaller than ϵ which implies that the original SVM primal objective (3) is solved to accuracy ϵ as well.

1.3 Block Coordinate Frank-Wolfe method (BCFW)

One of the major disadvantage of the batch Frank-Wolfe applied to the Structural SVM is that each iteration requires a full pass through the data. A block-coordinate generalization of the Frank-Wolfe can help reaching much cheaper iterations since it requires only one call to the maximization oracle instead of n calls.

The algorithm presented in this section applies to any constrained convex optimization problem of the form

$$\min_{\alpha\in\mathcal{M}} f(\alpha) \tag{10}$$

where the domain $\mathcal{M}$ has the structure of a Cartesian product $\mathcal{M} = \mathcal{M}^{(1)} \times ... \times \mathcal{M}^{(n)} \subset \mathbb{R}^m$. We assume that each factor $\mathcal{M}^{(i)} \subseteq R^{m_i}$ is convex and compact, with $m = \sum_{i=1}^{n} m_i$. We will write $\alpha_{(i)} \in \mathbb{R}^{m_i}$ for the i-th block of coordinates of a vector $\alpha \in \mathbb{R}^m$.

Algorithm 3 Block-Coordinate Frank-Wolfe Algorithm on Product Domain

```
Let $\alpha^{(0)} \in \mathcal{M} = \mathcal{M}^{(1)} \times ... \times \mathcal{M}^{(n)}$
for $k \leftarrow 0$ to $K$ do
    Pick $i$ at random in $\{1, ..., n\}$
    Find $s_i := \text{argmin}_{s'_i \in \mathcal{M}^{(i)}} < s'_{(i)}, \nabla_{(i)} f(\alpha^{(k)}) >$
    Let $\gamma := \frac{2n}{k+2n}$ or optimize $\gamma$ by line-search
    Update
```

In each step, Algorithm (3) picks one of the n blocks uniformly at random and leaves all other blocks unchanged. If there is only one block meaning $n = 1$, then Algorithm (3) becomes the batch Frank-Wolfe Algorithm (1).
It can be shown that Algorithm (3) still has a convergence rate of $\mathcal{O}(\frac{1}{\epsilon})$ obtaining an ϵ-approximate solution to (10) with an ϵ-small duality gap. For further details on the convergence of BCFW, please refer to Lacoste-Julien et al., 2012.

1.3.1 BCFW for the Structural SVMs

As for the case of the regular Frank-Wolfe algorithm, the block coordinate version can easily be applied to the structural SVM dual 4, maintaining only the primal variables w.

Algorithm 4 Block-Coordinate Primal-Dual Frank- Wolfe Algorithm for the Structural SVM

```
Let $w^{(0)} := w_i^{(0)} := \bar{w}^{(0)} := 0, l^{(0)} := l_i^{(0)} := 0$
for $k \leftarrow 0$ to $K$ do
    Pick $i$ at random in $\{1, ..., n\}$
    Solve $y_i^* := \text{argmax}_{y \in \mathcal{Y}_i} H_i(y; w^{(k)})$
    Let $w_s := \frac{1}{\lambda n} \psi(y_i^*)$ and $l_s := \frac{1}{n} L_i(y_i^*)$
    Let $\gamma := \frac{\lambda(w_i^{(k)} - w_s)^T w^{(k)} - l_i^{(k)} + l_s)}{\lambda ||w_i^{(k)} - w_s||^2}$ and clip to [0,1]
    Update $w_i^{(k+1)} := (1-\gamma) w_i^{(k)} + \gamma w_s$ and $l_i^{(k+1)} := (1-\gamma) l_i^{(k)} + \gamma l_s$
    Update $w^{(k+1)} := w^{(k)} + w_i^{(k+1)} w_i^{(k)}$ and $l^{(k+1)} := l^{(k)} + l_i^{(k+1)} l_i^{(k)}$

    (Optionally: Update $\bar{w}^{(k+1)} := \frac{k}{k+2} \bar{w}^{(k)} + \frac{2}{k+2} w^{(k+1)}$)
```

We see that Algorithm 4 is equivalent to Algorithm 3, by observing that the corresponding primal updates become $w_s = A s_{[i]}$ and $l_s = b^T s_{[i]}$, where $s_{[i]}$ is the zero-padding of $s_{(i)} := e^{y_i^*} \in \mathcal{M}^{(i)}$ so that $s_{[i]} \in \mathcal{M}$.

We get that the number of iterations needed for Algorithm (4) to obtain a specific accuracy ϵ is the same as for the batch version in Algorithm (2), even though each iteration takes n times fewer oracle calls.

1.4 Adaptive non-uniform sampling BCFW

The adaptive sampling version of the block coordinate Frank-Wolfe arises from the fact that the standard BCFW algorithm uniformly selects at each iterate a training object and performs the block-coordinate step w.r.t. the corresponding dual variables. If these variables are already close to being optimal, then BCFW does not make significant progress at this iteration. Instead, the intuition behind adaptive non-uniform sampling BCFW is to randomly choose a block at each iteration in such a way that the blocks with larger duality gaps so with larger suboptimality are sampled more often. In this way, we obtain more information from the sampled blocks.

Algorithm 5 Block-coordinate Frank-Wolfe with gap sampling for structured SVM

```
Let $w^{(0)} := w_i^{(0)} := 0;\ l^{(0)} := l_i^{(0)} := 0;\ g_i^{(0)} := +\infty;$
$k_i := 0$   // the last time $g_i$ was computed
for $k \leftarrow 0$ to  do
    Pick $i$ at random with probability $\propto g_i^{(k_i)}$
    Solve $y_i^* = \text{argmax}_{y \in \mathcal{Y}_i} H_i(y; w^{(k)})$
    Let $k_i = k$
    Let $w_s := \frac{1}{\lambda n}\psi_i(y_i^*)$ and $l_s := \frac{1}{n}L_i(y_i^*)$
    Let $g_i^{(k_i)} := \lambda(w_i^{(k)} - w_s)^T w^{(k)} - l_i^{(k)} + l_s$
    Let $\gamma := \frac{g^{(k_i)}}{\lambda ||w_i^{(k)} - w_s||^2}$ and clip to [0,1]
    Update $w_i^{(k+1)} := (1-\gamma)w_i^{(k)} + \gamma w_s$ and $l_i^{(k+1)} := (1-\gamma)l_i^{(k)} + \gamma l_s$
    Update $w^{(k+1)} := w^{(k)} + w_i^{(k+1)} - w_i^{(k)}$ and $l^{(k+1)} := l^{(k)} + l_i^{(k+1)} - l_i^{(k)}$
    if update global gap then
        for $i \leftarrow 0$ to $n$ do
            Let $k_i := k + 1$
            Solve $y_i^* := \text{argmax}_{y \in \mathcal{Y}_i} H_i(y; w^{(k_i)})$
            Let $w_s := \frac{1}{\lambda n}\psi_i(y_i^*)$ and $l_s := \frac{1}{n}L_i(y_i^*)$
            $g_i^{(k_i)} := \lambda(w_i^{(k_i)} - w_s)^T w^{(k_i)} - l_i^{(k_i)} + l_s$
        end for
    end if
end for
```

Intuitively, by adapting the probabilities $p^{(k)}$, we can obtain a better bound on the expected improvement of f but having access to the exact block gaps at each step is prohibitively expensive. However, the values of the block gaps obtained at the previous iterations can serve as estimates of the block gaps at the current iteration. We use them in the following non-uniform gap sampling scheme: $p_i^{(k)} \propto g_i(\alpha^{(k_i)})$ where k_i records the last iteration at which the gap i was computed.

So having access to the exact block gaps is intractable, however, we have access to the block gaps computed from past oracle calls on each block. Anyway, such estimates can be outdated and therefore might be quite far from the current values of the block gaps. In this case, we talk about 'staleness' which can be overcome by doing a full gap computation after several block-coordinate passes. We must find a suitable trade-off in the number of blocks we let pass before refreshing the block gaps. From Osokin et al., 2016 emerges that when the gap computation is never run, the gap becomes significantly underestimated and the algorithm does not converge. On another extreme, when performing the gap computation after each pass of BCFW, the algorithm wastes too many computations and converges slowly.

2 Experiment

In the following section, we implement the algorithms explained above and we run the code on a real dataset CoNLL 2000, comparing the performances among them.

2.1 CoNLL 2000

Dataset In our experiment, we used the CoNLL dataset, released by and which poses the task of text chunking. Text chunking, also known as shallow parsing, consists in dividing the input text into syntactically related non-overlapping groups of words, called phrases or chunks. This means that one word can only be a member of one chunk. The task of text chunking can be cast as a sequence labeling where a sequence of labels y is predicted from an input of tokens x. For a given token x_t (a word with its corresponding part-of-speech tag), the associated label y_t gives the type of phrase the token belongs to, i.e., says whether or not it corresponds to the beginning of a chunk, or encodes the fact that the token does not belong to a chunk.

We now present an example of sentence:

[$_{\text{NP}}$ He] [$_{\text{VP}}$ reckons] [$_{\text{NP}}$ the current account deficit] [$_{\text{VP}}$ will narrow] [$_{\text{PP}}$ to] [$_{\text{NP}}$ only £ 1.8 billion] [$_{\text{PP}}$ in] [$_{\text{NP}}$ September].

Here chunks have been represented as groups of words between square brackets. A tag next to the open bracket denotes the type of the chunk. For example, NP indicates the noun phrase, VP the verb phrase, and PP the prepositional phrase (Tjong Kim Sang and Buchholz, 2000)

The dataset contains tokens (words and punctuation marks), information about the location of sentence boundaries, and information about chunk boundaries. Additionally, a part-of-speech (POS) tag was assigned to each token by a standard POS tagger (Marcus et al., 1993).

The original CoNLL dataset contains 8'936 training English sentences extracted from the Wall Street Journal part of the Penn Treebank II (Marcus et al., 1993); and each output label y_t can take up to 22 different values. The dataset is ordered. This means that starting from the first sentence and going to the last, the first state encountered will be state number 1, the second will be state number 2, and so on. The same is applied to the values of the feature of a word.

For computational reasons, we restrict the dataset to 1'000 sentences. By doing that, not only do we decrease the number of observations in the dataset, but also we decrease the number of values each feature of a word can take and the number of states for the reason stated above. In this case, we have that the features have 38'647 different values, and the word can take 20 different state values. This will come in handy in the implementation of the algorithm.

Feature Map We use the feature map $\phi(x, y)$ proposed by Sha and Pereira, 2003 and used also by Osokin et al., 2016. The dataset is given such that, for each position t of the input sequence x, we have a unary feature representation, containing the local information. We transformed these unary feature representations into binary representations to get a sparse binary vector of dimensionality 38'647 each.

Given a labeling y and the unary representations, the feature map ϕ is constructed by concatenating features of three types: emission, transition, and bias. We construct a $38'647 \times 20$ matrix representing the emission features (*unit* in the code), which count the number of times each coordinate of the unary representation of token x_t is nonzero and the corresponding output variable y_t is assigned a particular value.

The transition map of size 20×20 (*bin* in the code) encodes the number of times one label follows another in the output y.

The 20×3 bias features encode biases for all the possible values of the output variables, and, specifically, biases for the first (*gr_start*) and last (*gr_end*) variables. Note that one bias row is already encode in the *unit* matrix in our code.

Finally, we create a vector concatenating horizontally the *unit* rows, *gr_start*, *gr_end* and the matrix *bin* and transformed it into a sparse space feature.

Task-dependent structured error The structured error $L(y_i, y)$ is the normalized Hamming distance.

Gap Sampling As already mentioned, *Gap Sampling* consists of sampling with a higher probability of the blocks that have a big *Block Gap*. To do this, we create a vector whose length corresponds to the number of blocks and set all of its values to ∞. Then, after each iteration performed on the block i, we update the *Block Gap* related to that variable. This way we make sure that, in the beginning, every block is chosen at least once, and then, from that point on, those blocks with larger suboptimality are sampled more often.

Oracle Since our structured error corresponds to the normalized Hamming distance, the max oracle consists of the Viterbi algorithm (Viterbi, 1967). Briefly, the Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states - called the Viterbi path - that results in a sequence of observed events.

For each sentence the max oracle is applied, we construct a transition matrix T (*nodePot* in *makeLogNodePotentials* in the code) with size #words $\times$ #states. We denote this matrix T. This matrix stores the a posteriori probability, that is $T(i,j) = P(y_j|x_i)$, where x_i is a word of the sentence, and y_j a state.

Then, we compared each path likelihood function and take the path with the greatest likelihood function, denoted as the survivor (first `for` cycle in *logDecode*). Only the survivors are preserved for further consideration, while the remaining paths are discarded.

Finally, for each survivor, we compute the likelihood functions and store the maximum values, which correspond to the most likely sequence (second `for` cycle in *logDecode*).

Results To evaluate the three algorithms we train the model on 1000 of the 8936 sentences of the CoNLL dataset (for computational reasons). Furthermore, we put 1000 seconds as time limit (which is enough to understand how the three methods compare) and 0.01 as the gap threshold below which the algorithms stop. Then, we produce four different plots (Fig 1):

- **Duality Gap** against the **Number of Iterations**.
- **Training Error** against the **Number of Iterations**.
- **Duality Gap** against **CPU Time**.
- **Training Error** against **CPU Time**.

Unfortunately, because we only use a subset of the Training Set, we are not able to evaluate the model on the Test Set. This is because of the way the feature map is created: as we explained in 2.1 the number of states is determined incrementally and, in the 1000 sentences chosen, not every single one of them appears. In the test set, however, they do and, as a result, the feature maps have different sizes which makes them incompatible. Nevertheless, the results on the Test Set are consistent, as shown in Osokin et al., 2016.

First, we notice that only the gap sampling method is able to reach convergence in the 1000 seconds we put as threshold, this makes it the fastest out of the three. Second, we can see right away that the number of iterations needed to reach a small value for the duality gap and a low error on the training set is much smaller on the Batch FW. This is expected because in one iteration the batch algorithm explores all the dataset, while the BCFW algorithms explore just one observation at each iteration. However, it is interesting to see that the Gap BCFW takes a considerably smaller amount of iterations than the Uniform BCFW to reach convergence (in our experiments we noticed it usually needs 30-50% less, depending on the number of sentences). Even though it is less noticeable, the gap method also reaches higher accuracy in fewer iterations than the uniform method, making it the better alternative.

Analyzing the computational time, the superiority of Block Coordinate methods is recognizable. Both the Uniform BCFW and Gap BCFW take much less time than the Batch FW to reach a low duality gap. The former, however, is not able to keep up with the improvements of the non-uniform BCFW with regards to the duality gap and, after a while, convergence becomes much slower. The difference between the three methods is particularly striking in the case of gap sampling which can reach convergence in an astonishing smaller time. In our experiments we also observed that this effect is stronger when the number of sentences grows, making it the superior algorithm for this task.

One thing to notice is that, in the case of Block Coordinate methods, the computation of the duality gap is not free like in the Batch case. This is because we need a full pass over all the examples to compute it. In our case, we decide to do a gap check every 100 iterations which means that faster convergence is very easily achievable by simply lowering the number of checks (which are still necessary to make sure that convergence is reached).

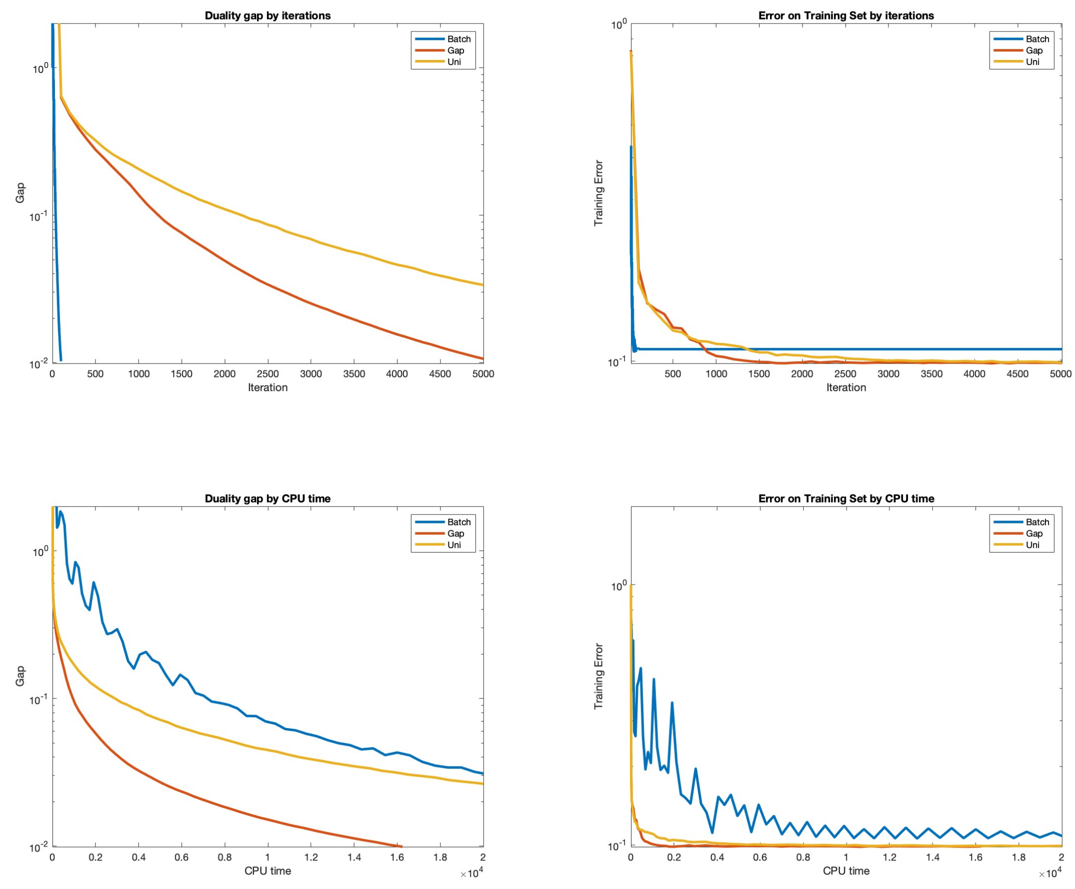

Figure 1: Summary of the results on the CoNLL dataset. In blue Batch FW, in yellow Uniform BCFW and in red Gap BCFW. Duality Gap (Left) and Train Error (Right) against the number of iterations (Top). Duality Gap (Left) and Train Error (Right) against CPU time (Bottom).

As it can be seen in Osokin et al., 2016 (*fig. 5*) the results obtained with the full Training Set are very similar to the ones we got with only 1000 sentences: gap sampling is in general faster and works better in this kind of setting.

Conclusion

In this paper, we discussed the use of the Frank-Wolfe algorithm for structural SVMs prediction task.

We provided the theoretical results about FW, BCFW, and BCFW with gap sampling, explaining how we can construct the dual problem for the minimization structural SVMs problem and how to develop each algorithm for the considered task.

First, we analyzed the standard Frank-Wolfe algorithm which, due to its sparse iterations, has proven quite effective applied to the dual structural SVM objective with an exponential number of variables. Then, we considered a randomized block-coordinate generalization of Frank-Wolfe proposed by Lacoste-Julien et al., 2012 which, despite the much lower iteration cost, reaches almost the same convergence rate in the duality gap. Finally, we talked about a variant of the Block Coordinate Frank-Wolfe introduced by Osokin et al., 2016 which uses the information contained in the block gaps to sample the blocks that have a larger suboptimality.

Then, we implemented and evaluated the three algorithms on a text chunking task using the ConLL 2000 Dataset: the experiments confirm that the block coordinate methods are faster at reaching convergence. Moreover, we can see that the use of gap sampling makes the BCFW algorithm faster to converge.

Finally, we noticed that the results obtained by our experiments using just 1'000 sentences are in line with the ones obtained by Osokin et al., 2016 which uses the complete dataset.

References

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex optimization.* Cambridge university press.

Lacoste-Julien, Simon, Martin Jaggi, Mark Schmidt, and Patrick Pletscher (2012). "Stochastic Block-Coordinate Frank-Wolfe Optimization for Structural SVMs". In: *CoRR* abs/1207.4747. arXiv: `1207.4747`. URL: `http://arxiv.org/abs/1207.4747`.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2, pp. 313–330. URL: `https://aclanthology.org/J93-2004`.

Osokin, Anton, Jean-Baptiste Alayrac, Isabella Lukasewitz, Puneet Kumar Dokania, and Simon Lacoste-Julien (2016). "Minding the Gaps for Block Frank-Wolfe Optimization of Structured SVMs". In: URL: `http://arxiv.org/abs/1605.09346`.

Sha, Fei and Fernando Pereira (2003). "Shallow Parsing with Conditional Random Fields". In: USA: Association for Computational Linguistics. URL: `https://doi.org/10.3115/1073445.1073473`.

Tjong Kim Sang, Erik F. and Sabine Buchholz (2000). "Introduction to the CoNLL-2000 Shared Task Chunking". In: *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop.* URL: `https://aclanthology.org/W00-0726`.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13.2, pp. 260–269.