Supplementary Material for ECCV 2016 paper: Partial Linearization based Optimization for Multi-class SVM

Pritish Mohapatra[1], Puneet Kumar Dokania[2], C.V. Jawahar[1]
and M. Pawan Kumar[3]

[1] IIIT-Hyderabad, [2] CentraleSupelec & INRIA Saclay, [3] University of Oxford

1 Appendix

1.1 Proof for Proposition 1

Proposition 1. *If the surrogate function is defined as*

$$f(\boldsymbol{\alpha}, \boldsymbol{\alpha}^k) = \frac{\tau}{n} \sum_{i \in [n]} \sum_{y \in \mathcal{Y}} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}),$$

then the update direction $\mathbf{s}^k$ *in iteration* k *for given* i *and* $y \in \mathcal{Y}$ *can be computed as*

$$\mathbf{s}^k_{iy} = \frac{\exp\left(\log(\boldsymbol{\alpha}^{k-1}_{iy}) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k-1\top} \Psi_i(y))\right)}{z_i}. \tag{1}$$

Proof. Given the choice of the surrogate function as,

$$f(\boldsymbol{\alpha}, \boldsymbol{\alpha}^k) = \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}), \tag{2}$$

the objective of the optimization problem that has to be solved in the k^{th} iteration becomes,

$$\begin{aligned} T^k(\boldsymbol{\alpha}) \; &= \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}) - b^\top \boldsymbol{\alpha}^k \\ &+ \frac{\lambda}{2} \boldsymbol{\alpha}^{k\top} A^\top A \boldsymbol{\alpha}^k \\ &- \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}^k_{iy} \log(\boldsymbol{\alpha}^k_{iy}) \\ &+ ((-b + \lambda A^\top A \boldsymbol{\alpha}^k) - \frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}^k)))^\top (\boldsymbol{\alpha} - \boldsymbol{\alpha}^k) \end{aligned}$$

Therefore, the approximate optimization problem to be solved in the k^{th} iteration looks like,

$$\begin{aligned} &\min_{\alpha \geq 0} \; T^k(\boldsymbol{\alpha}) \\ &\text{s.t.} \quad \sum_{y \in Y_i} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n] \end{aligned} \tag{3}$$

The Lagrangian of problem 3, with $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_n]^\top$ as the set of Lagranian multipliers can be defined as,

$$L(\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}) = T^k(\boldsymbol{\alpha}) + \sum_{i \in [n]} \boldsymbol{\beta}_i \left(\sum_{y \in Y_i} \boldsymbol{\alpha}_i - 1 \right)$$

Differentiating the Lagrangian with respect to the variables,

$$\begin{aligned} & \frac{\partial L}{\partial \boldsymbol{\alpha}_{iy}} = 0 \\ \Rightarrow \; & \frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}_{iy})) + (-b_{iy} + \lambda (A^T A)_{iy} \boldsymbol{\alpha}^k) \\ & -\frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}^k_{iy})) + \boldsymbol{\beta}_i = 0 \\ \Rightarrow \; & \frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}_{iy})) + (-b_{iy} + \frac{1}{n} {\mathbf{w}^k}^\top \Psi_i(\mathbf{y})) \\ & -\frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}^k_{iy})) + \boldsymbol{\beta}_i = 0 \\ \Rightarrow \; & \frac{\tau}{n} \log(\boldsymbol{\alpha}_{iy}) = \frac{\tau}{n} \log(\boldsymbol{\alpha}^k_{iy}) + b_{iy} - \frac{1}{n} {\mathbf{w}^k}^\top \Psi_i(\mathbf{y}) - \boldsymbol{\beta}_i \\ \Rightarrow \; & \boldsymbol{\alpha}_{iy} = \exp\left(\log(\boldsymbol{\alpha}^k_{iy}) + \frac{n}{\tau}(b_{iy} - \frac{1}{n} {\mathbf{w}^k}^\top \Psi_i(\mathbf{y}) - \boldsymbol{\beta}_i) \right) \\ \Rightarrow \; & \boldsymbol{\alpha}_{iy} = \frac{\exp\left(\log(\alpha^k_{iy}) + \frac{1}{\tau}(\Delta_i(y) - {\mathbf{w}^k}^\top \Psi_i(\mathbf{y}) \right)}{\exp\left(\frac{n}{\tau}\beta_i\right)} \end{aligned} \tag{4}$$

$$\begin{aligned} & \frac{\partial L}{\partial \boldsymbol{\beta}_i} = 1 \; \Rightarrow \; \sum_{y \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} = 0 \\ \Rightarrow \; & \sum_{y \in \mathcal{Y}_i} \frac{\exp\left(\log(\boldsymbol{\alpha}^k_{iy}) + \frac{1}{\tau}(\Delta_i(y) - {\mathbf{w}^k}^\top \Psi_i(\mathbf{y})) \right)}{\exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right)} = 1 \\ \Rightarrow \; & \sum_{y \in \mathcal{Y}_i} \exp\left(\log(\boldsymbol{\alpha}^k_{iy}) + \frac{1}{\tau}(\Delta_i(y) - {\mathbf{w}^k}^\top \Psi_i(\mathbf{y})) \right) = \exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right) \\ \Rightarrow \; & z_i = \exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right) \end{aligned} \tag{5}$$

Hence, using 4 and 5, the optimal solution is,

$$\mathbf{s}_{iy}^k = \frac{\exp\left(\log(\alpha_{iy}^k) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k\top}\Psi_i(\mathbf{y}))\right)}{z_i}$$

1.2 Proof for Proposition 3

Proposition 3. *The block-coordinate partial linearization algorithm is guaranteed to converge to the global optima of the multi-class* SVM *learning problem.*

Proof. Consider the following optimization problem whose feasible domain and objective function are a cross-section are a cross-section of the original dual SSVM learning problem.

$$\begin{aligned} \min_{\boldsymbol{\alpha}_k \geq 0} \quad & T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_{\mathbf{y}\in\mathcal{Y}_j} \boldsymbol{\alpha}_{jy} = 1, \\ & \boldsymbol{\alpha}_i = \bar{\boldsymbol{\alpha}}_i, \forall i \in [n] - j, \end{aligned} \tag{6}$$

where $\bar{\boldsymbol{\alpha}}_i$'s are fixed and satisfy $\sum_{\mathbf{y}\in\mathcal{Y}_i} \bar{\boldsymbol{\alpha}}_{iy} = 1, \forall i \in [n] - j$. We can devise a partial-linearization algorithm for solving this optimization problem using $f(\boldsymbol{\alpha}_j, {\boldsymbol{\alpha}_j}^k) = \frac{\tau}{n}\sum_{\mathbf{y}\in\mathcal{Y}_j} \boldsymbol{\alpha}_{jy}\log(\boldsymbol{\alpha}_{jy})$, as the surrogate function. Following the procedure similar to the proof for proposition 1, it can be shown that the update direction in the k^{th} iteration of the algorithm would take the following form.

$$s_{iy}^k = \frac{\exp\left(\log(\alpha_{jy}^{k-1}) + \frac{1}{\tau}(\Delta_j(y) - {\mathbf{w}^{k-1}}^T\Psi_{jy})\right)}{z_j}. \tag{7}$$

It follows from the work of Patriksson in [1] that this update direction is guaranteed to be a feasible descent direction of the objective function of problem 6. As the objective function in 6 is a cross-section of the objective function of problem (2) over $\boldsymbol{\alpha}_j$, this update direction would also be a feasible descent direction of the original objective of the dual SSVM learning problem. It can be easily verified that in the k^{th} iteration of the block-coordinate partial-linearization (BCPL) algorithm, the update direction is of the form of 7 with $\bar{\boldsymbol{\alpha}}_{iy} = \boldsymbol{\alpha}_{iy}^k, \forall i \in [n] - j$. Therefore, the update direction in each iteration of BCPL is guaranteed to reduce the original dual SSVM objective. Since the dual SSVM learning problem is convex, the BCPL algorithm is guaranteed to converge to the global optima.

1.3 Partial linearization with primal update for optimization of low tree-width SSVM

We provide a brief overview of the structured SVM optimization problem. Given an input $\mathbf{x} \in \mathcal{X}$ the aim of structured prediction is to predict the output $\mathbf{y}$ that

belongs to a structured space $\mathcal{Y}(\mathbf{x})$. A joint feature map $\Phi(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}^d$, encodes the relationship between an input $\mathbf{x}$ and an output $\mathbf{y}$. A structured SVM, parameterized by $\mathbf{w}$, provides a linear prediction rule as follows: $h_{\mathbf{w}}(\mathbf{x}) = \text{argmax}_{\mathbf{y} \in \mathcal{Y}} \left(\mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y}) \right)$, which is parameterized by $\mathbf{w}$. Given a set of labelled samples $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, the parameter vector $\mathbf{w}$ is learnt by solving the following convex optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \ & \frac{\lambda}{2} ||\mathbf{w}||^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \ & \mathbf{w}^\top \Psi_i(\mathbf{y}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i, \forall i \in [n], \forall \mathbf{y} \in \mathcal{Y}(\mathbf{x}_i) \end{aligned} \tag{8}$$

Here $\Psi_i(\mathbf{y}) = \Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \mathbf{y})$ and $\Delta(\mathbf{y}_i, \mathbf{y})$ is the loss incurred for predicting $\mathbf{y}$ given the ground truth $\mathbf{y}_i$ for the sample $\mathbf{x}_i$. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and shall use $\mathcal{Y}_i$ and $\Delta_i(\mathbf{y})$ as a short hand for $\mathcal{Y}(\mathbf{x}_i)$and $\Delta(\mathbf{y}, \mathbf{y}_i)$ respectively. The Lagrangian dual of the problem (8) is given by:

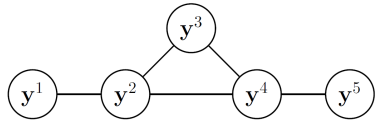
$$\begin{aligned} \min_{\boldsymbol{\alpha} \geq 0} \ & T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha} \\ \text{s.t.} \ & \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n]. \end{aligned} \tag{9}$$

Here the dual variable vector $\boldsymbol{\alpha}$ is of size $m = \sum_{i=1}^{n} |\mathcal{Y}_i|$; $\mathbf{b} \in \mathcal{R}^m$ is defined as $\mathbf{b} = \{b_{iy} = \frac{1}{n} \Delta_i(\mathbf{y}) \mid i \in [n], \mathbf{y} \in \mathcal{Y}_i\}$ and the matrix $A \in \mathcal{R}^{d \times m}$ is defined as $A = \{A_{iy} = \frac{1}{\lambda n} \Psi_i(\mathbf{y}) \in \mathcal{R}^d \mid i \in [n], \mathbf{y} \in \mathcal{Y}_i\}$.

In general the size of output space can be exponential in the number of output variables. This would result in exponentially large number of primal constraints and dual variables which can be hard to deal with. For example, consider the problem of recognizing the handwritten word depicted in an image, which has been segmented into p letters. The total number of possible words is 26^p. In other words, given the current set of parameters, we would require $O(n26^p)$ time in order to compute the objective function of problem (8), since we would have to find the value of the slack variable ξ_i for each sample. In order to alleviate this deficiency, the structure of the output space is often exploited. For example, in the handwritten word recognition example, we can consider the output space to consist of a set of p parts. Each part corresponds to a letter of the word. Consider a joint feature vector $\Psi_i(\mathbf{y})$ that decomposes over the parts, that is, $\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1); \Psi_i(\mathbf{y}^2); ...; \Psi_i(\mathbf{y}^p)]$. It can be verified that, for a given set of parameters, we can compute the objective function value of problem (8) in $O(np)$ time. However, the above joint feature vector fails to capture the interdependency between the letters. For example, if the q^{th} letter is a 'q' then the likelihood of the $(q+1)^{th}$ letter to be 'u' is very high. To capture this, one can use a slightly more complex joint feature vector of the form:

$$\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1, \mathbf{y}^2); ...; \Psi_i(\mathbf{y}^q, \mathbf{y}^{q+1}); ...; \Psi_i(\mathbf{y}^{p-1}, \mathbf{y}^p)].$$

More generally, one can visually represent the output space as a graph where the vertices are the parts of the output and the edges represent their interdependency. Using a "Markovian" style argument, we define a joint feature vector



.

Fig. 1: *An example of a plausible underlying graph structure of the output space. Here, the graph has 3 maximal cliques:* $\mathcal{C}_1 = \{\mathbf{y}^1, \mathbf{y}^2\}$, $\mathcal{C}_2 = \{\mathbf{y}^2, \mathbf{y}^3, \mathbf{y}^4\}$, $\mathcal{C}_3 = \{\mathbf{y}^4, \mathbf{y}^5\}$

that decomposes into features defined over the maximal cliques of the graph. In other words, $\Psi_i(\mathbf{y}) = [\Psi_i(\mathcal{C}_1); \Psi_i(\mathcal{C}_2); ...; \Psi_i(\mathcal{C}_h)]$, where $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_h\}$ is the set of maximal cliques of the graph. For example, consider the graph shown in Figure 1. In this case, the graph consists of 3 maximal cliques and the joint feature vector decomposes as:

$$\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1, \mathbf{y}^2); \Psi_i(\mathbf{y}^2, \mathbf{y}^3, \mathbf{y}^4); \Psi_i(\mathbf{y}^4, \mathbf{y}^5)].$$

When the underlying graphical structure of the output space has a low tree-width, it is still possible to efficiently evaluate the objective of problem (8), since the following problem lends itself to efficient optimization:

$$\bar{\mathbf{y}}_i = \underset{\mathbf{y} \in \mathcal{Y}_i}{\operatorname{argmax}} \, \Delta_i(\mathbf{y}) - \mathbf{w}^\top \Psi_i(\mathbf{y}) \tag{10}$$

Note that the value of ξ_i can be computed using $\overline{\mathbf{y}}_i$ as $\xi_i = \Delta_i(\overline{\mathbf{y}}_i) - \mathbf{w}^\top \Psi_i(\overline{\mathbf{y}}_i)$. We refer to the above problem as the *max-oracle*. Let $P(\mathbf{y})$ denote the probability distribution represented by the graph parameterized by the loss augmented scores as potential functions. The max-oracle gives the most probable configuration of this distribution $P(\mathbf{y})$. It has been shown through several works, including cutting-plane algorithms [2], subgradient descent [3] and Frank-Wolfe [4], that an efficient solution to the above problem is sufficient to minimize problem (8) and/or its Lagrangian dual (9) efficiently. As we will see shortly, our work exploits the fact that, for low tree-width graphs, a related problem known as the *expectation-oracle* can be solved efficiently as well (with the same time complexity as the max-oracle). While the max-oracle gives the most probable configuration, the expectation-oracle gives the marginals for the parts over which the output space decomposes. For example, the marginal for part $\mathbf{y}^q$ can be represented as $\mu_i(\mathbf{y}^q) = \sum_{\mathbf{y}^l \neq \mathbf{y}^q} P(\mathbf{y}^1, ..., \mathbf{y}^p)$. Similarly, the marginal for a clique $\mathcal{C} = \{\mathbf{y}^q, \mathbf{y}^r\}$ would be $\mu_i(\mathbf{y}^q, \mathbf{y}^r) = \sum_{\mathbf{y}^l \notin \{\mathbf{y}^q, \mathbf{y}^r\}} P(\mathbf{y}^1, ..., \mathbf{y}^p)$. By cleverly exploiting this observation, we obtain a natural generalization of the Frank-Wolfe algorithm that retains many of its desirable properties such as guaranteed descent direction, analytically computable optimal step size and guaranteed convergence even in block-coordinate mode, while allowing the use of the expectation-oracle to find a valid descent direction that can often lead to improved performance in practice.

The partial-linearization algorithm for optimizing the dual multi-class SVM problem is outlined in Algorithm 1. Step 6 in Algorithm 1, requires us to explicitly compute the update direction corresponding to each dual variable. When the number of dual variables are small as in the case of multiclass classification, this step can be performed efficiently. However, in general, the number of dual variables is proportional to the size of the output space $\mathcal{Y}$. So, when $|\mathcal{Y}|$ is big, it becomes computationally infeasible to explicitly compute the update direction for each dual variable. When the temperature hyperparameter $\tau = 0$, that is when partial-linearization reduces to Frank-Wolfe, this deficiency can be alleviated using an equivalent algorithm that updates the primal variables only. We now show that such an efficient implementation is actually possible for all values of the temperature hyperparameter.

The key observation behind an efficient implementation of partial-linearization using primal variables is that we are only required to compute the marginals of the output variables. Recall from the above discussion that when the output space $\mathcal{Y}_i$ has the underlying graph structure of a tree or a low tree-width graph, it is possible to efficiently compute the exact marginals of the parts of output space, over which it decomposes, by solving the *expectation-oracle* problem. This can be done using a message passing algorithm over a junction tree corresponding to the underlying graph of the output space [5]. For higher tree-width graph structures, although it is not generally possible to efficiently compute the exact marginals, we can get good approximations efficiently.

In order to compute the required marginals with respect to the distribution $\mathbf{s}_i^k$, we parameterize the underlying graph of the output space by node potentials:

$$\theta_i^{s^k}(\mathbf{y}^q) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q) - \mathbf{w}^{k-1\top}\Psi_i(\mathbf{y}^q))$$

and edge potentials:

$$\begin{aligned}\theta_i^{s^k}(\mathbf{y}^q, \mathbf{y}^r) = \quad & \theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) \\ & + \tfrac{1}{\tau}(\Delta_i(\mathbf{y}^q, \mathbf{y}^r) - \mathbf{w}^{k-1\top}\Psi_i(\mathbf{y}^q, \mathbf{y}^r)).\end{aligned}$$

Where, $\theta_i^{\alpha^{k-1}}$ are the potential functions corresponding to the marginals from the $(k-1)^{th}$ iteration. They can be computed from the marginals as $\theta_i^{\alpha^{k-1}}(\mathbf{y}^q) = (1 - \deg(\mathbf{y}_q))\log(\mu_i^k(\mathbf{y}_q))$ for the nodes and as $\theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) = \log(\mu_i^k(\mathbf{y}_q, \mathbf{y}_r))$ for the edges. Here, $\deg(\mathbf{y}_q)$ denotes the degree of the node $\mathbf{y}^q$.

Proposition 4. *For tree structured output space, given the marginals for the parts, the parameter vector* $\mathbf{w}$ *can be computed as,*

$$\begin{aligned}\mathbf{w} = \quad & \textstyle\sum_i \sum_{\mathbf{y}^q} \mu_i(\mathbf{y}^q)\Psi_i(\mathbf{y}^q) \\ & + \textstyle\sum_i \sum_{\mathbf{y}^q, \mathbf{y}^r} \mu_i(\mathbf{y}^q, \mathbf{y}^r)\Psi_i(\mathbf{y}^q, \mathbf{y}^r)\end{aligned} \tag{11}$$

The proof of the above proposition is provided in the next section.

The complete partial linearization based optimization algorithm involving iterative update of the primal variables is outlined in Algorithm 1. In the k^{th} iteration of the algorithm, we compute the marginals μ_i^k for a chosen sample i (steps 5-7). We first compute the potentials θ_i^k to parameterize the graph underlying the output space. The marginals μ_i^k are then computed by a message passing algorithm over the graph. In step 8, making use of proposition 3, we compute the primal vector $\mathbf{w}_s$ in the update direction from the marginals. We also compute the expected loss l_s using the marginals in step 9. Next, we compute the optimal update step γ (step 10) and update the primal variable vector $\mathbf{w}$, the loss $\mathbf{l}$ and the marignals μ in steps 11-15. The epochs are repeated until the convergence of the dual objective.

Algorithm 1 *Block-Coordinate Partial-linearization with primal variable update for optimizing* SSVM

1: $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$
2: Initialize marginals μ^0 such that $\mathbf{w}(\mu^0) \sim [0]^d$, $k \leftarrow 1$, $l^0 \leftarrow 0$ and $\forall i \in [n]$, $l_i^0 \leftarrow 0$
3: Initialize a $(d \times n)$ matrix W such that i^{th} column of W, $\mathbf{w}_i = \mathbf{w}(\mu_i^0)$
4: **repeat**
5: Chose a random $i \in [n]$
6: Compute potential functions to parameterize the graph:
$\forall$ nodes $\mathbf{y}^q$,
$\theta_i^{s^k}(\mathbf{y}^q) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q) - \mathbf{w}^{k-1\top}\Psi_i(\mathbf{y}^q))$
and $\forall$ edges $(\mathbf{y}^q, \mathbf{y}^r)$,
$\theta_i^{s^k}(\mathbf{y}^q, \mathbf{y}^r) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q, \mathbf{y}^r) - \mathbf{w}^{k-1\top}\Psi_i(\mathbf{y}^q, \mathbf{y}^r))$.
7: Compute marginals $\mu_s^k(i)$ by doing message passing over the graph parameterized by potentials $\theta_i^{s^k}$.
8: $\mathbf{w}_s \leftarrow \sum_{\mathbf{y}^q} \mu_i^k(\mathbf{y}^q)\Psi_i(\mathbf{y}^q) + \sum_{\mathbf{y}^q, \mathbf{y}^r} \mu_i^k(\mathbf{y}^q, \mathbf{y}^r)\Psi_i(\mathbf{y}^q, \mathbf{y}^r)$
9: $l_s \leftarrow \sum_{\mathbf{y}}^q \mu_i^k(\mathbf{y}^q)\Delta_i(\mathbf{y}^q)$
10: Optimal step size,
$\gamma \leftarrow \frac{\lambda < \mathbf{w}_i^k, \mathbf{w}_i^k - \mathbf{w}_s > - (l_i^k - l_s)}{\lambda ||\mathbf{w}_i^k - \mathbf{w}_s||^2}$
11: Update $\mathbf{w}_i$: $\mathbf{w}_i^k \leftarrow (1-\gamma)\mathbf{w}_i^{k-1} + (\gamma)\mathbf{w}_s^k$
12: Update l_i: $l_i^k \leftarrow (1-\gamma)l_i^{k-1} + (\gamma)l_s^k$
13: Update $\mathbf{w}$: $\mathbf{w}^k \leftarrow \mathbf{w}^{k-1} - \mathbf{w}_i^{k-1} + \mathbf{w}_i^k$
14: Update l: $l^k \leftarrow l^{k-1} - l_i^{k-1} + l_i^k$
15: Update the marginals: $\mu^k = (\gamma)\mu_s^k + (1-\gamma)\mu^{k-1}$
16: $k \leftarrow k+1$
17: **until** Convergence
18: Optimal parameter, $\mathbf{w}$

1.4 Proof for Proposition 4

Proposition 4 *For tree structured output space, given the marginals for the parts, the parameter vector* $\mathbf{w}$ *can be computed as,*

$$\begin{aligned}\mathbf{w} = \quad & \textstyle\sum_i \sum_{\mathbf{y}^q} \mu_i(\mathbf{y}^q)\Psi_i(\mathbf{y}^q) \\ & + \textstyle\sum_i \sum_{\mathbf{y}^q,\mathbf{y}^r} \mu_i(\mathbf{y}^q,\mathbf{y}^r)\Psi_i(\mathbf{y}^q,\mathbf{y}^r)\end{aligned} \tag{12}$$

Proof. As we discuss the case in which the output space has a tree structure, any output can be decomposed on to a set of parts $p \in P$. P being the set of parts. Let r_p be a configuration a part can have from a set of all possible configurations R_p for the part p. Then any output vector $\mathbf{y}$ can be composed of a combination of $r \in R_p$ for the different parts $p \in P$. The feature vector $\Psi_i(\mathbf{y})$ can also be decomposed into a sum of feature vectors for individual parts.

$$\Psi_i(\mathbf{y}) = \sum_{p \in P} \Psi_i(r_p)$$

We define θ_i^k as the potentials computed at iteration k for sample i and σ_y as the mapping from potentials to the distribution.

$$\boldsymbol{\alpha}_{iy} = \sigma_y(\theta_i^k)$$

We can compute the weight vector $\mathbf{w}$ for the corresponding vector $\mathbf{s}^k$ in the dual space as follows.

$$\begin{aligned}
\mathbf{w}(\mathbf{s}^k) &= \sum_i \sum_y s_{iy}^k \left(\Psi_i(\mathbf{y})\right) \\
&= \sum_i \sum_y s_{iy}^k \Psi_i(\mathbf{y}) \\
&= \sum_i \sum_y \sigma_y(\theta_i^k)\Psi_i(\mathbf{y}) \\
&= \sum_i \sum_y \sigma_y(\theta_i^k) \sum_{r_p \in y} \Psi_i(r_p) \\
&= \sum_i \sum_y \sum_{r_p \in y} \sigma_y(\theta_i^k)\Psi_i(r_p) \\
&= \sum_i \sum_{p \in P} \sum_{r_p \in R_p} \left(\sum_{y : r_p \in y} \sigma_y(\theta_i^k) \right) \Psi_i(r_p) \\
&= \sum_i \sum_{p \in P} \sum_{r_p \in R_p} \mu_{i,r_p}(\theta_i^k)\Psi_i(r_p)
\end{aligned} \tag{13}$$

Here, $\mu_{i,r_p}(\theta_i^k)$ is the marginal probability of the configuration r_p of part p for sample i in the k^{th} iteration.

1.5 Numerically stable implementation of sum-product Belief Propagation

During the training process, we can encounter situations in which the range of the values of the potentials associated with the graph structure is very big. Computing factors by exponentiating these potentials would inadvertently lead to underflow or overflow. Overflow leads to numerical infinities which are difficult to handle. Whereas, underflow leads to factors getting truncated to 0, which leads to dual variables getting stuck on facets of the domain polytope. In order to avoid such kind of numerical instability, we try to do all our computations in the log-space or the potential space, as far as possible.

We shall illustrate our method with a simple example. Consider a tree with 2 nodes $\{u_1, u_2\}$ and a single edge e_{12}. Let x_i denote the random variable associated with node u_i. Then the unary potentials associated with node u_i is denoted by $\theta(x_i)$ and the pairwise potential associated with the edge e_{12} is denoted by $\theta(x_1, x_2)$. In the belief propagation algorithm, let the message that is passed from node u_i to node u_j be denoted by $msg_{ij}(x_j)$. We compute the message $msg_{21}(x_1)$ as follows,

$$msg_{21}(x_1) = \sum_{x_2} \exp\left(\theta_2(x_2) + \theta_{12}(x_1, x_2) - C_1(x_1)\right) \tag{14}$$

Where, $C_1(x_1) = \max_{x_2}\left(\theta_2(x_2) + \theta_{12}(x_1, x_2)\right) - M$, M being a number chosen according to the maximum representational capacity of the machine. This clamping operation before exponentiation, guarantees that at least one element inside the summation in equation 14 is equal to M. Hence, the message never gets truncated to 0. Once all the messages are computed, the log-marginals for example $\log(P(x_1))$ are computed as follows,

$$\log\left(P(x_1)\right) = \theta_1(x_1) + \log\left(msg_{21}(x_1)\right) + C_1(x_1) - \log(z) \tag{15}$$

Where, $\log(z)$ is the log-partition function. This method requires us to store the clamping constants $C_i(x_i)$'s along with the messages. This implementation allows us to run our optimization algorithm for very low values of lambda and temperature without any numerical instability.

1.6 Comparison between Partial-linearization at low temperatures and Frank-Wolfe

For very low values ($\leq 10^{-5}$) of temperature, as can be seen in figure 2, BCPL behaves almost exactly same as BCFW when change in objective function value is considered over number of iterations. The small gap observed between the BCFW and BCPL at low temperature is because of the difference in running time of the oracles for the respective algorithms.

1.7 Experimental results for scene-text recognition

Dataset. We use the IIIT-5k dataset [6] for the scene-text recognition problem. The dataset consists of 5000 word images collected from natural scenes. It includes 2000 'trainval' and 3000 'test' images. We are also given the bounding

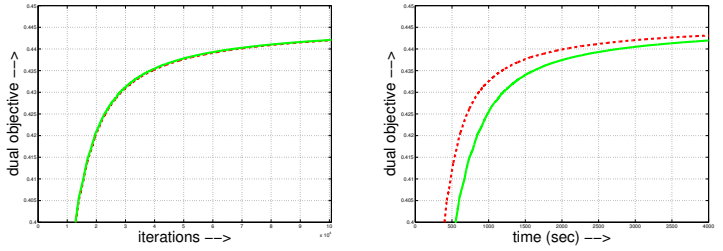


Fig. 2: *Comparison of the Frank-Wolfe algorithm with Partial-linearization at very low temperature ($\tau = 10^{-10}$). Left plot shows change in dual objective over training iterations where as right plot change in dual objective over time. Here, the broken red curves correspond to the Frank-Wolfe algorithm, solid green to our partial-linearization algorithm with $\tau = 10^{-10}$.*

boxes of the characters in each of word image. Each character can be of one of the 62 classes: $\{a, ..., z, A, ..., Z, 0, ..., 9\}$;

Features. The word images in the dataset are of different size. Consequently, the size of the character bounding boxes vary over a wide range. We run 2 sets of experiments by resizing each of the segmented character images first to the size of 50×25 pixels and then to 70×35 pixels. We represent each character image using either a 1250 sized or a 2450 sized feature vector constructed from its gray-scale pixel values. Similar to the handwritten word recognition problem, we use indicator basis functions to represent the correlation between adjacent characters, location independent bias for each of the characters and additional bias for the first and the last characters of any word. This makes the overall size of the feature vector equal to $(1250 \times 26 + 26 \times 26 + 26 + 26 \times 2) = 33254$ for character image size of 50×25 and $(2450 \times 26 + 26 \times 26 + 26 + 26 \times 2) = 64454$ for character image size of 70×35.

Methods. We compare our BCPL algorithm with the BCFW algorithm. We run each method for 2 values $(0.1, 0.01)$ of λ and repeat the BCPL algorithm for 5 values $(1, 0.1, 0.01, 0.001, 10^{-5})$ of τ. We repeat the same set of experiments for both character image size of 50×25 as well as 70×35.

Results. We report the performance of the different optimization algorithms for the structured SVM learning problem. Figure 3 shows the progress of the optimization algorithms over time. It can be observed that while for character image size of 50×25 BCFW has a faster rate of convergence than our method, for 70×35, our method performs better and beats BCFW for $\lambda = 0.1$.

In general for any problem, our method performs better when the size of the feature vector large. It's quite common to have large feature vectors for many practical problems particularly those related to computer vision and are potential candidates for application of our optimization algorithm.

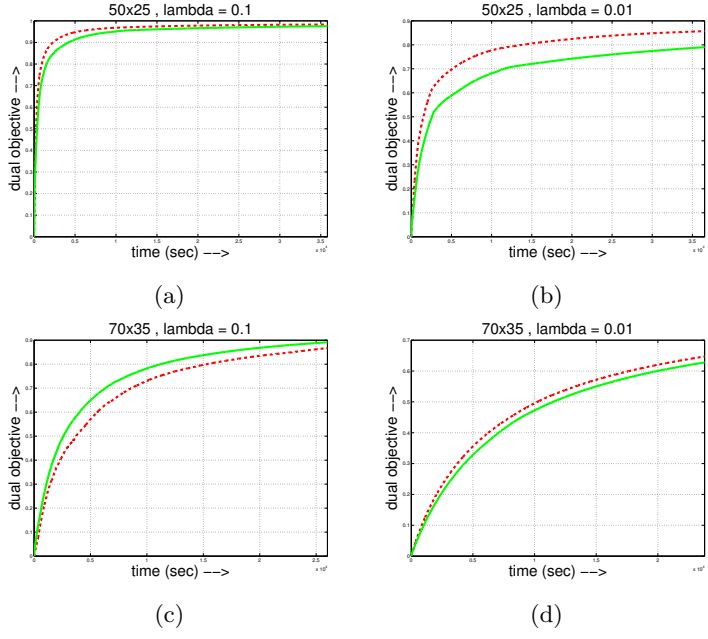


Fig. 3: *Comparison of optimization algorithms for the scene-text recognition problem. Plots show change in the dual objective over training time in seconds. The left column corresponds to a character image size of* 50×25 *and the right column to that of* 70×35*. The top row shows the results for* $\lambda = 0.1$ *and the bottom row for* $\lambda = 0.01$*. Here, the broken red curves correspond to the Frank-Wolfe algorithm, solid green to our partial-linearization algorithm. The curves representing the partial-linearization algorithm correspond to the best values of the temperature parameter.*

References

1. Patriksson, M.: Partial linearization methods in nonlinear programming. Journal of Optimization Theory and Applications, Springer, 1993
2. Joachims, T., Finley, T., Yu, C.J.: Cutting-plane training of structural svms. Machine Learning, Springer, 2009
3. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: Primal estimated sub-gradient solver for svm. Mathematical programming, Springer, 2011
4. Lacoste-Julien, S., Jaggi, M., Schmidt, M., Pletscher, P.: Block-coordinate frank-wolfe for structural svms. In: ICML. (2012)
5. Wainwright, M.J., Jordan, M.: Graphical models, exponential families, and variational inference. Foundations and Trends® in Machine Learning (2008)
6. Mishra, A., Alahari, K., Jawahar, C.V.: Scene text recognition using higher order language priors. In: BMVC. (2012)