Scalability of Struct SVM approaches

Frederic Boileau Elyes Lamouchi

William St-Arnaud

https://github.com/SoliElvis/structuredPredictionProject

29th April 2019

Abstract

hello

Contents

1 Introduction

Structured prediction in machine learning is tasked with learning predictors where the labels on the datapoints are more than simple tags but have inherent structure which implies the following:

- The number of potential labels for a given feature vector can grow exponentially with the input which makes traditional classification procedures intractable
- A certain intelligibility of the structure; hence a hope to leverage it to improve tractability

It is often quite hard to know in advance wether we can tackle a structured prediction problem with known approaches.

In this paper we discuss two different approaches to solving structured prediction problems, both focusing on a *large-margin approach* which translates into a support-vector machine - like problem formulation.

Let us introduce some notation which we will use throughout the paper. Define the dataset to be

$$S = \{x^{(i)}, y^{(i)})\}_{i=1}^{n} \in (\mathcal{X} \times \mathcal{Y})^n \tag{1}$$

If one were to frame the machine learning goal in the concisest and simplest way possible we could say that the goal is to *learn* a (parametrized) *predictor* function:

$$\begin{aligned} h_w \quad &: \quad \mathcal{X} \to \mathcal{Y} \\ &: \quad \hat{x} \mapsto y \qquad y \in \mathcal{Y}(\hat{x}) \end{aligned}$$

where $\hat{x}$ is just some arbitrary sampled $\hat{x}$ and we abuse notation and mean the set valued mapping which outputs the feasible label set for a given x by $\mathcal{Y}(x)$

The traditional probabilistic approach is to compute h or its parameter by calculating the most likely parameter given conditionned on the observed data. However this is more often than not untractable in the structured prediction context as summed up in Taskar, Lacoste-Julien, and Jordan (2006)

In a large margin approach we wish to compute the predictor function the following way :

$$h_w(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle \tag{2}$$

where ϕ is just the feature map for the dataset.

This is a constrained optimization problem and the structure of $\mathcal{Y}$ clearly has a big influence on how well we can solve the problem as well as which method should work well.

The dual extragradient approach presented in Taskar, Lacoste-Julien, and Jordan (ibid.) leverages a saddle-point approach to the problem to tackle the problem with a first order method and moreover "this approach allows us to exploit the structure of W and Z separately, allowing for efficient solutions for a wider range of parameterizations and structures."(ibid.)

Despite its improvements at the time the dual extragradient approach suffers from two main computational draw-backs:

1. It is not seperable/'stochasizable' which is a severe problem as machine learning has seen much of its success in recent years by optimizing through first-order sampled method, i.e. first order method which use a probabilistic approximation of the gradient through sampling instead of computing the actual gradient (which might be unfeasible).
2. The algorithm requires a projection step which can be expensive depending on the structure of $\mathcal{Y}$.

This leads us to consider the Block Seperable Frank Wolfe algorithm presented in Lacoste-Julien et al. (2013)

2 Preliminaries

2.1 Structured SVM N-slack formulation

Let us recall the basic goal; to construct an accurate linear classifier[1]

$$h_w(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle \tag{3}$$

In general finding the optimal seperating hyperplane is an ill-defined problem. In the support vector machine (SVM) setting we want to find the seperating hyperplane with the *largest margin*. The support vectors are the datapoints which lie on the the defined margins. It is not obvious that we should strive to completely seperate the sets so we can include some slack variables. Indeed, some set of points might not be linearly separable but adding slack variables might allow one to misclassify some point but get a higher proportion of correctly classified points.

We call the following way to pose problem the n-slack formulation [2] of the problem :

$$\max_{\omega, \xi} \quad \frac{\lambda}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n} \varepsilon_i \tag{4}$$

$$s.t. \quad \langle w, \psi_i(y) \rangle \geq L(y_i, y) - \varepsilon_i, \quad \forall i, \forall y \in \mathcal{Y}(x) = \mathcal{Y}_i \tag{5}$$

2.2 Loss-Augmented Decoding

Define $\psi(y) := \phi(x^{(i)}, y^{(i)}) - \phi(x^{(i)}, y)$ and $L_i(y) = L(y^{(i)}, y)$

$$H_i = \max_{y \in \mathcal{Y}_i} \quad \{L_i(y) - \langle w, \psi_i(y) \rangle\} \tag{6}$$

$$\mathcal{H}_i = \arg\max_{y \in \mathcal{Y}_i} \quad \{L_i(y) - \langle w, \psi_i(y) \rangle\} \tag{7}$$

Proposition. The max oracle is a convex upper bound to the task loss.

Proof sketch: The maximum of two convex (linear) functions is convex, and

$$L(y_i, h_w(x_i)) \leq L(y_i, h_w(x_i)) + \underbrace{\langle w, \psi_i(y) \rangle}_{\geq 0 \ by \ definition} \tag{8}$$

$$\leq \max_{y \in \mathcal{Y}_i} L_i(y) - \langle w, \psi_i(y) \rangle \tag{9}$$

It is not hard to see thus that learning w amounts to the unconstrained problem,

$$\max_{w} \quad \frac{\lambda}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n} \tilde{H}_i(w) \tag{10}$$

[1] clearly we can easily extend to non-linear cases, with kernel maps for example

[2] see Moguerza and Muñoz (2006) for example

3 Extra Gradient Algorithm

3.1 Saddle Point Formualtion and Intuition

$$\min_{\mathbf{w}\in W}\max_{\mathbf{z}\in Z}\sum_i \left(\mathbf{w}^T\mathbf{F}_i\mathbf{z}_i + \mathbf{c}_i^T\mathbf{z}_i - \mathbf{w}^T\mathbf{f}_i(\mathbf{y}_i)\right) \tag{11}$$

where the $\mathbf{z}_i$'s can be identified with the edge and node potentials of a markov network and satisfy the constraints of the structured problem. The terms $\mathbf{F}_i$ correspond to the feature mapping for over all labels $\mathbf{y}_i$ when multiplied by $\mathbf{z}_i$'s. The $\mathbf{c}_i$'s correspond to the costs of a $\mathbf{z}_i$ and can be identified with the loss l for a label $\mathbf{y}_i'$.

In equation 11, the term that is opitmized is defined as:

$$L(\mathbf{w},\mathbf{z}) \triangleq \sum_i \mathbf{w}^T\mathbf{F}_i\mathbf{z}_i + \mathbf{c}_i^T - \mathbf{w}^T\mathbf{f}_i(\mathbf{y}_i) \tag{12}$$

It is bilinear in w and z. We can then imagine two players represented by $\mathbf{w}$ and $\mathbf{z}$ that play a zero-sum game. They perform updates using gradients of the objective w.r.t. their parameters. They then project the result to the set of feasible points given by the constraints imposed on the structure. We usually consider Euclidean projections, as there are well-known problems where they are efficient to compute. However, as seen later, this is not the case for all problem. This is why Bregman projections will be introduced. Going back to the zero-sum game, we have the following operator that is used to perform the updates for both players at the same time.

$$\begin{pmatrix} \nabla_{\mathbf{w}}L(\mathbf{w},\mathbf{z}) \\ -\nabla_{\mathbf{z}_1}L(\mathbf{w},\mathbf{z}) \\ \vdots \\ -\nabla_{\mathbf{z}_m}L(\mathbf{w},\mathbf{z}) \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & \mathbf{F}_1 & \dots & \mathbf{F}_m \\ -\mathbf{F}_1^T & & & \\ \vdots & & \mathbf{0} & \\ -\mathbf{F}_m^T & & & \end{pmatrix}}_{\mathbf{F}} \underbrace{\begin{pmatrix} \mathbf{w} \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_m \end{pmatrix}}_{\mathbf{u}} - \underbrace{\begin{pmatrix} \sum_i \mathbf{f}_i(\mathbf{y}_i) \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{pmatrix}}_{\mathbf{a}} = \mathbf{F}\mathbf{u} - \mathbf{a} \tag{13}$$

3.2 Duality and Gap function

We can measure the "goodness" of the parameters using the gap function G:

$$G(\mathbf{w}, \mathbf{z}) \triangleq \left[\max_{\mathbf{z}' \in Z} L(\mathbf{w}, \mathbf{z}') - L^*\right] + \left[L^* - \min_{\mathbf{w}' \in W} L(\mathbf{w}', \mathbf{z})\right] \tag{14}$$

where L^* gives the result of the min-max of the objective L. When we have a non-optimal point (i.e. not a saddle point), the gap is strictly positive. At at an optimal point, the gap is exaclty equal to 0. Now the restricted gap is exactly the same but the min and max are computed over a set of parameters that are within a certain distance of the start point $(\hat{\mathbf{u}}_\mathbf{w}, \hat{\mathbf{u}}_\mathbf{z}) \in U$:

$$G_{D_\mathbf{w}, D_\mathbf{z}}(\mathbf{w}, \mathbf{z}) = \max_{\mathbf{z}' \in Z} \left[L(\mathbf{w}', \mathbf{z}') : d(\mathbf{z}, \mathbf{z}') \leq D_\mathbf{z}\right] - \left[\min_{\mathbf{w}' \in W} L(\mathbf{w}', \mathbf{z}) : d(\mathbf{w}, \mathbf{w}') \leq D_{\mathbf{w}'}\right] \tag{15}$$

The motivation for using this restricted gap function is that if we start "close" to an optimal point, of course we will converge more rapidly to it. This can be seen in the convergence analysis of the method.

We mainly follow the intuition and proofs of Taskar, Lacoste-Julien, and Jordan, 2006.
The dual extragradient algorithm from Nesterov gives a convergence guarantee for the objective L.
We present a simple formulation of the algorithm:

Initialize: Choose $\hat{\mathbf{u}} \in U$, set $\mathbf{s}^{-1} = 0$.
for $t = 0$ to $t = \tau$ **do**
$\mathbf{v} = \mathbf{\Pi}_U(\hat{\mathbf{u}} + \eta \mathbf{s}^{t-1})$
$\mathbf{u}^t = \mathbf{\Pi}_U(\mathbf{v} - \eta(\mathbf{F}\mathbf{v} - \mathbf{a}))$
$\mathbf{s}^t = \mathbf{s}^{t-1} - (\mathbf{F}\mathbf{u}^t - \mathbf{a})$
end for
return $\overline{\mathbf{u}^\tau} = \frac{1}{1+\tau} \sum_{t=0}^{\tau} \mathbf{u}^t$

This algorithm has a lookahead step (i.e. v) that serves the peform the actual gradient update u^t. The intuition behind the lookahead step is that given a function to optimize that is Lipschitz, Nesterov was able to show that we can upper bound $f_D(\bar{u^n}) = \max_y \{\langle g(y), \bar{u^n} - y\rangle : d(\hat{u}, y) \leq D\}$, where $\bar{u^n}$ is the weighted average over all the updates u^t up to iteration n. The function g corresponds to the objective L in our setting. When value of $f_D(\bar{u^n})$ gets close to 0, we have that the value $g(y^*)$ for an optimal y^* is close to 0, which signifies that we have reached saddle point (i.e. what we wanted). Nesterov goes on to show that this upper bound indeed goes to 0. We then get convergence to a saddle point. Note that in the definition of f_D, we used a distance metric d. This corresponds to the Euclidean distance (or Bregman distance in non-Euclidean setting). The rojection operator Π_U in the algorithm simply projects a point back to the set U by finding the nearest point with respect to the distance metric used.

3.2.1 Non-Euclidean setting

The main problem with the Euclidean projection operator is that for many problems, it is hard to compute the projection. Indeed for min-cut, we need to compute the partition function first, which is #P-complete. Thus, the authors of the paper introduced the Bregman operator, which computes the projection using the Bregmand divergence. Using this operator has the great advantage of being easier to compute. We can see this for $L1$ regularization. Computing a projection using $L1$ distance is hard since it is not differentiable. Using the negative entropy, we get that the Bregman divergence is the KL divergence. This implies that we can differentiate the divergence to get the parameter that minimizes it.

3.2.2 Memory-efficient tweak

In the dual extragradient algorithm, both a vector s^t and a vector $\bar{u^t}$ are maintained. However, we can observe that the s_t's can be found using the running average $\bar{u^t}$ since $s^t = -(t+1)\sum_{i=0}^{t}(F\bar{u^t} - a)$. We only have to store the vector $\bar{u^t}$. We can even do better when $|Z| \gg |W|$ since $\bar{u^t} = \{\bar{u^t}_w, \bar{u^t}_z\}$ and we only care about the part that corresponds to w. $\bar{u^t}_z$ is maintained implicitly by storing a vector of size $|W|$ (although we now need to store s_w^t). It can be reconstructed using $\bar{u^t}_w$.

4 Frank Wolfe and Variants

4.1 Frank-Wolfe, the conditional gradient algorithm

Classical Frank-Wolfe

Consider a linear minimization oracle,

$$LMO_{\mathcal{A}}(\nabla f(x_t)) \in \arg\min_{s\prime \in \mathcal{A}} \langle s\prime, \nabla f(x_t)\rangle \tag{16}$$

Simply put; starting with an active set consisting of only an initial feasible point $S^0 = \{x_0\}$, the Frank-Wolfe algorithm adds an element $s_t = LMO_{\mathcal{A}}(\nabla f(x_t))$, to the active set by taking its convex combination with the previous iterate which allows for a sparse representation. Sparsity is a key feature of the BCFW variant of Frank Wolfe as it optimizes over the dual of struct-SVM which has a very large number of variables (since the primal has a very large number of constraints defined through the hinge loss)

Algorithm 1 Classical Frank-Wolf

Let $\alpha \in \mathcal{M}$

for $k = 0$ **to** K **do**

Compute $s = argmin_{s\in\mathcal{M}} \langle s, \nabla f(\alpha^k)\rangle$

Let $\gamma = \frac{2}{k+2}$, or optimize γ by line search

Update $\alpha^{k+1} = (1-\gamma)\alpha^k + \gamma s$

end for

Convergence results

We define the duality gap

$$g(\alpha^k) = \max_{s\in\mathcal{M}} \langle \alpha^k - s, \nabla f(\alpha^k)\rangle \tag{17}$$

By first order convexity of the objective, we have

$$f(s) \geq f(\alpha^k) + \langle \alpha^k - s, \nabla f(\alpha^k)\rangle \tag{18}$$

$$\implies g(\alpha^k) = \min_{s\in\mathcal{M}} \langle \alpha^k - s, \nabla f(\alpha^k)\rangle \geq f(\alpha^k) - f^* \tag{19}$$

We can thus see that the duality gap gives us a computable optimality guarantee.

Definition 1. *The curvature constant C_f is given by the maximum relative deviation of the objective function f from its linear aaporximations, over the domain $\mathcal{M}$,*

$$C_f = \sup_{\substack{x,s\in\mathcal{M} \\ \gamma\in[0,1], y=x+\gamma(s-x)}} \frac{2}{\gamma^2}\Big(f(y) - f(x) - \langle y-x, \nabla f(x)\rangle\Big) \tag{20}$$

Intuitively, the curvature constant can be seen as a measure of how flat the objective function is. For example, if the objective is linear, say $f(x) = ax + b$ and $x \in [e, f]$ then $\nabla f(x) = a$ and the curvature constant is zero:

$$C_f = \frac{2}{\gamma^2}\Big(ay + b - ax - b + (-ay + ax)\Big) = 0 \tag{21}$$

Moreover $s = argmin_{s\in[e,f]}\langle s, a\rangle = \frac{e}{a}$. Hence we reach the minimum in one F-W step. Thus, we can observe that for flatter functions, that is with smaller curvature constants, Frank-Wolfe should converge faster.

Theorem 1. *The duality gap obtained in the t^{th} iteraton of the Frank-Wolfe algorithm satisfies*

$$g(x_t) \leq 2\beta\frac{C_f}{t+2}(1+\delta) \tag{22}$$

Where $\beta = \frac{27}{8}$ and δ is the approximation error tolerated in the LMO.

4.2 Block Coordinate Frank Wolfe

Due to the exponential number of dual variables in the structured SVM setting, classical algorithms, like projected gradient are intractable. Stochastic subgradient methods, on the other hand, achieve a sublinear convergence rate while only requiring a single call to the maximization oracle every step. They are nonetheless very sensitive to the sequence of stepsizes and it is unclear when to terminate the iterations.

Frank-Wolfe methods address these problems by giving a stepsize which can be computed exactly analytically and a computable duality gap while still retaining a sublinear convergence rate. Moreover, despite the exponential number of constraints, the algorithm has sparse iterates alleviating the memory issues which come with the exponential number of dual variables.

Theorem 2. *Given a convex, differentiable objective $f : \mathcal{M}^1 \times ... \times \mathcal{M}^n \to \mathbb{R}$, where $\forall i \in \{1..n\}$, each factor $\mathcal{M}^i \subseteq \mathbb{R}^n$ is convex and compact, if we are at a point x such that $f(x)$ is minimized along each coordinate axis, then x is a global minimum.*

Proof. The objective function being differentiable and convex, if we are at a point α such that $f(\alpha)$ is minimized along each coordinate axis, then α is a global minimizer. Therefore,

$$\min_{s\in\mathcal{M}}\langle s, \nabla f(\alpha)\rangle = \sum_i \min_{s_i\in\Delta_{|\mathcal{Y}_i|}}\langle s_i, \nabla_i f(\alpha)\rangle \tag{23}$$

Moreover, with

$$w = A\alpha, A = \Big[\frac{1}{n\lambda}\psi_1(y)...\frac{1}{n\lambda}\psi_{\Sigma_i\,|\mathcal{Y}_i|}(y)\Big]$$
$$and \quad b = \Big(\frac{1}{n}L_i(y)\Big)_{i\in\Big[n\Big],y\in\mathcal{Y}_i}$$

The gradient of the dual would be,

$$\nabla f(\alpha) = \nabla\Big[\frac{\lambda}{2}||A\alpha||^2 - b^T\alpha\Big] = \lambda A^T A\alpha - b$$
$$= \lambda A^T w - b = \frac{1}{n}H_i(y, w)$$
$$\max_{y_i \in \mathcal{Y}_i} \quad \tilde{H}_i = -\min_{y_i \in \mathcal{Y}_i} \quad \tilde{H}_i = \min_{y_i \in \mathcal{Y}_i} \quad L_i - \langle w, \psi_i \rangle$$
$$= \min_{s_i \in \Delta_{|\mathcal{Y}_i|}} \langle s_i, \nabla_i f(\alpha) \rangle$$

Thus we can see that, if n = size of the training data, one Frank-Wolfe step is equivalent to n calls to the maximization oracle. □

Algorithm 2 Block Seperable Frank-Wolf

Let $\alpha \in \mathcal{M}$

Let $w^0 = 0$, $l^0 = 0$

for $k = 0, \ldots, K$ **do**

  for $i = 1, \ldots, n$ **do**

    Solve $y_i^* = \max_{y_i \in \mathcal{Y}_i} H_i(y, w^k)$

    Let $w_s = \sum_{i=1}^n \frac{1}{n\lambda}\psi_i(y_i^*)$, and $l_s = \frac{1}{n}\sum_{i=1}^n L_i(y_i^*)$

    Let $\gamma = \frac{\lambda(w^k - w_s)^T w^k - l^k + l_s}{\lambda||w^k - w_s||^2}$, and clip to $[0, 1]$

    Update $w^{k+1} = (1-\gamma)w^k + \gamma w_s$, and $l^{k+1} = (1-\gamma)l^k + \gamma l_s$

  end for

end for

5 Empirical

5.1 Experiments

In this section, we describe the implementation we performed for this project. The goal was to see how the Dual Extragradient algrithm compared to the Block-coordinate Frank-Wolfe algorithm. The task on which we performed the evaluation was word alignment in machine translation. We extracted the dataset from the Europarl dataset *Europarl Parallel Corpus* 2019. The data consisted of approximately 2 million sentence pairs in both english and french. Each the sentences in each pair were translations of one another. We extracted all sentences and performed a clean by splitting longer sentences into shorter ones. The goal of this step was to reduce the eventual number of matchings in training, which could take long to solve using a LP solver. Of course, each sentence was tokenized beforehand.

We then proceeded to implement the Dual Extragradient and BCFW algorithms using a SVM. We had to define a feature mapping for an input sentence pair. The features were extracted using the fastText library **fastText**. This library included a model that was previously trained to learn embeddings of words in both english and french. We later combined the embeddings in the two languages by applying a transformation found in Chojnacki and Kłopotek (2010). This transformation consisted in applying a matrix to each vector in each language (matrix W for english and Q for french). These matrices are in fact orthogonal (i.e. $Q^T W = I$). The idea behind such a transformation is that we sort of “put" or “align" both languages in the same vector space, a sort of “middle ground". This way we can better compare the words “cat" and “chat" by getting their cosine similarity measure. We combined the cosine measure of each pair of words in the alignment by summing. The following blog post *Aligning Vector Representations* (2017) provides a good intuition using maps that are aligned. As an example, consider the following two sentences:

- This assignment was hard
- Ce travail etait ardu

We would compute the cosine distance for each word tuple (e.g. this/ce, this/travail, . . . , hard/ardu). We were then able to get the highest match of each english word for for french translation. This was how we extracted the “labels". As the dataset was not annotated with alignments, we had to compute those according to the procedure mentioned.

For the features, we used the concatenated embeddings of each word pairs in the alignment. This gave us our edge score. Then, we simply performed a weighted combination of these vectors using the edge labels as weights. To clarify what we mean by edge labels, suppose that the edge linking “assignment” and “travail” has a value of 1. Then, we weight the vector extracted from these two words with 1. As another example, if the edge between “ce” and “ardu” has a value of 0 (i.e. no link between the words), we do not include the vector computed from the statistics of this word pair. This is exactly what was done in Taskar Taskar, Lacoste-Julien, and Jordan, n.d. modulo some other features.

In the implementation of BCFW, we used the solver from scipy with the simplex method. Since the constraint matrix given by the optimization of H_i in the algorithm is unimodular, when we relax the LP, we still get a solution to the ILP without relaxation. Thus, we take advantage of this fact and indeed use the LP solver. The loss that we used was the L_1 distance between the two labels, the proposal and the ground truth.

5.1.1 Results

Since running the experiments was computationally intensive and we did not dispose of a lot of computing power, we had to restrict the training set to a number a relatively small number of sentence pairs (100). This sanity check was simply a hint to the general applicability of the method as we scale up the number of training examples. We used the default λ value of 0.01 as our regularizer since, we did not want to train for too many iterations before getting decent results as per Theorem 3 found in Lacoste-Julien et al. (2013). We were able to get the following results for the BCFW algorithm:

To further motivate our intuition and to make sure the algorith ran properly, we used the local scene dataset Müller and Behnke, 2014 from PyStruct to train a SVM using BCFW. It consisted of vectors of size 294 for the features of each training example (images). Each image was labeld according to the type of objects that were present in the scene.
These were:

$$beach \cdot sunset \cdot fall \cdot foilage \cdot field \cdot mountain \cdot urban \tag{24}$$

These 6 classes were not mutually exclusive so we had a total of $2^6 = 64$ possible labels. Training using all possible edges allowed to fully capture the complex relationships between them. The duality gap on the training data is given by the figure 2.

For the dual extragradient, we were only able to run the algorithm on a dataset of images Vemulapalli and Agarwala (2018). These were tuples of 3 images and a person hand-picked two images in each 3-tuple that resembled each other the most. Using the extragradient algorithm, we were able to get convergence but the results were not satisfying. We only obtained 46% accuracy on the prediction, which would compare with 17% if we had a random predictor (3 choose 2 gives 6 possibilites hence 17%). Thus, it motivated our change of dataset to obtain better results. We also moved on to the BCFW algorithm as we had not implemented it yet.

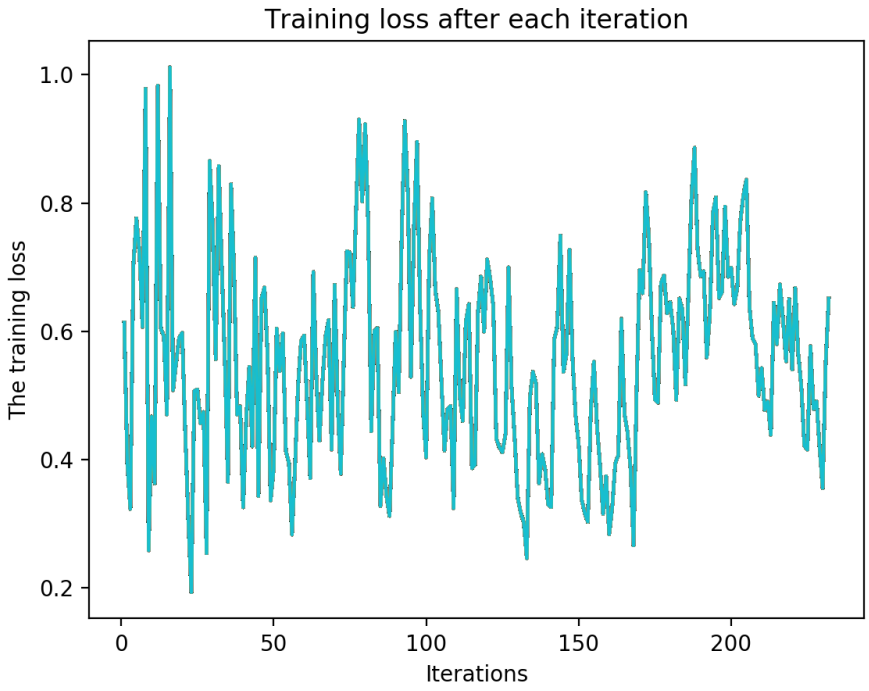

Figure 1: BCFW loss

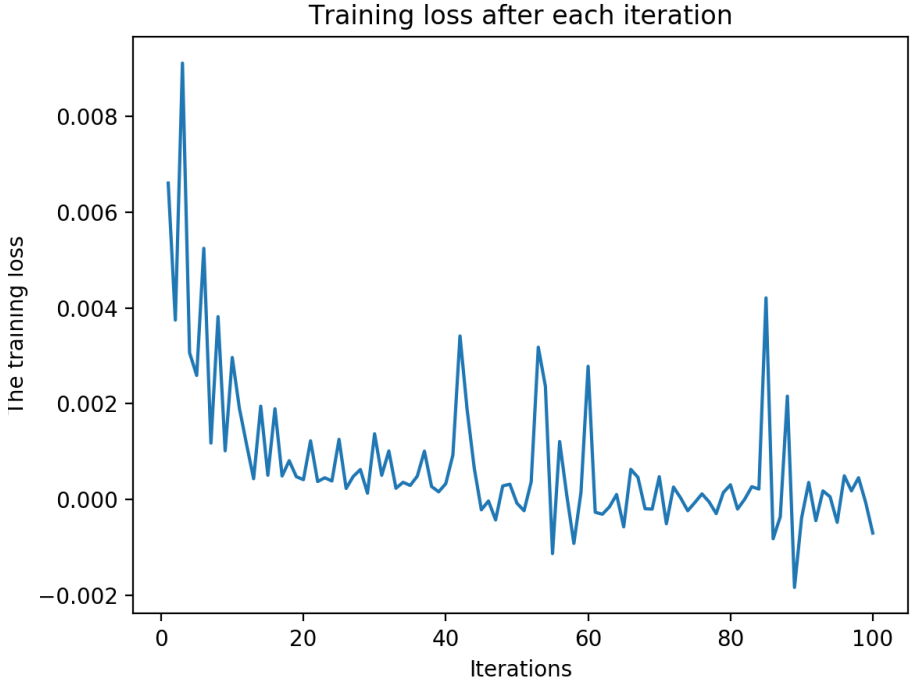

Figure 2: pystruct scene

6 Discussion

7 Conclusion and Further Work

8 Citations and References

References

[1] *Aligning Vector Representations*. May 27, 2017. URL: https://www.samtalksml.net/aligning-vector-representations/ (visited on 04/29/2019).

[2] Szymon Chojnacki and Mieczysław Kłopotek. "Random Graph Generator for Bipartite Networks Modeling". In: (Oct. 28, 2010). arXiv: 1010.5943 [physics]. URL: http://arxiv.org/abs/1010.5943 (visited on 04/29/2019).

[3] *Europarl Parallel Corpus*. URL: http://www.statmt.org/europarl/ (visited on 04/29/2019).

[4] Simon Lacoste-Julien et al. "Block-Coordinate Frank-Wolfe Optimization for Structural SVMs". In: (2013), p. 31.

[5] Javier M. Moguerza and Alberto Muñoz. "Support Vector Machines with Applications". In: *Statist. Sci.* 21.3 (Aug. 2006), pp. 322–336. ISSN: 0883-4237. DOI: 10.1214/088342306000000493. arXiv: math/0612817. URL: http://arxiv.org/abs/math/0612817 (visited on 04/29/2019).

[6] Andreas C. Müller and Sven Behnke. "Pystruct - Learning Structured Prediction in Python". In: *Journal of Machine Learning Research* 15 (2014), pp. 2055–2060. URL: http://jmlr.org/papers/v15/mueller14a.html.

[7] Ben Taskar, Simon Lacoste-Julien, and Michael I Jordan. "Structured Prediction via the Extragradient Method". In: (), p. 12.

[8] Ben Taskar, Simon Lacoste-Julien, and Michael I Jordan. "Structured Prediction, Dual Extragradient and Bregman Projections". In: (2006), p. 27.

[9] Raviteja Vemulapalli and Aseem Agarwala. "A Compact Embedding for Facial Expression Similarity". In: (Nov. 27, 2018). URL: https://arxiv.org/abs/1811.11283v2 (visited on 04/30/2019).

Appendices

Convex Analysis stuff

Proximal step operator

We define the proximal step operator as follows:

$$T_\eta(\mathbf{u}, \mathbf{s}) = \max_{\mathbf{u} \in U} \left\{ \langle \mathbf{s}, \mathbf{u}' - \mathbf{u} \rangle - \frac{1}{\eta} d(\mathbf{u}, \mathbf{u}') \leq D \right\} \tag{25}$$

The operator is useful to compute projections since when we have a strongly convex function $h(\mathbf{u})$, we can find its convex conjugate $h^*(\mathbf{u}) = \max_{\mathbf{u} \in U} [\langle \mathbf{s}, \mathbf{u} \rangle - h(\mathbf{u})]$. From the definition of a strongly convex function, we have that:

$$h(\mathbf{u}') \geq h(\mathbf{u}) + \langle \nabla h(\mathbf{u}), \mathbf{u}' - \mathbf{u} \rangle + \frac{\sigma}{2} \|\mathbf{u}' - \mathbf{u}\|^2 \tag{26}$$

where σ is the strong convexity parameter. Rearranging, we can define an upper bound on the squared norm of $\mathbf{u}' - \mathbf{u}$. This comes out as:

$$d(\mathbf{u}', \mathbf{u}) \triangleq h(\mathbf{u}') - h(\mathbf{u}) - \langle \nabla h(\mathbf{u}), \mathbf{u}' - \mathbf{u} \rangle \geq \frac{\sigma}{2} \|\mathbf{u}' - \mathbf{u}\|^2 \tag{27}$$

The distance metric d is called the Bregman divergence. The link between the Bregman diverence and the proximal step operator is that if we are given the function h inside the definition of the proximal step update, this induces the Bregman divergence, which in turn induces the update that is performed at each iteration of the extragradient algorithm. For example, if we have $h(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|_2^2$, the Bregman divergence becomes $d(\mathbf{u}', \mathbf{u}) = \frac{1}{2}\|\mathbf{u}' - \mathbf{u}'\|_2^2$. We might wonder why we care about the Bregman divergence when the definition still includes the usual norm. After all, we still optimize the term $\langle \mathbf{s}, \mathbf{u}' - \mathbf{u} \rangle - \frac{1}{\eta} d(\mathbf{u}', \mathbf{u})$. This is because $h*$ is differentiable at every point of its domain by the strong convexity of h. Thus, it is easy to compute a projection in the usual fashion: we can compute the derivative of the termn inside the projection operator and set it to 0. It is impossible to do for matchings for example as the distance is not even differentiable. We provide the steps to compute a projection:

$$\mathbf{s} - \nabla_{\mathbf{u}'} d(\mathbf{u}', \mathbf{u}) = \mathbf{s} - \frac{1}{\eta} \nabla_{\mathbf{u}'} d(\mathbf{u}, \mathbf{u}') = \mathbf{s} - \frac{1}{\eta} \left[\nabla h(\mathbf{u}') - \nabla h(\mathbf{u}) \right] \tag{28}$$

By setting this equation to 0, it is possible to recover the optimal $\mathbf{y}'$ when, let's say, $h(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}'\|^2$.

ExtraGradient

Convergence analysis

The restricted gap function $\mathcal{G}_{D_{\mathbf{v}},D_{\mathbf{z}}}$ is upper bounded by:

$$\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}(\overline{\mathbf{w}^{\tau}},\overline{\mathbf{z}^{\tau}}) \leq \frac{(D_{\mathbf{w}}+D_{\mathbf{z}})\,L}{\tau+1} \tag{29}$$

In his proof on the convergence of the extragradient algorithm, Nesterov (TOCITE) uses a function f_D instead of $\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}$, where f_D is defined as:

$$f_D(\mathbf{x}) = \max_{\mathbf{y}\in\mathcal{Q}} \{\langle g(\mathbf{y}), \mathbf{x}-\mathbf{y}\rangle : d(\mathbf{x},\mathbf{y})\} \tag{30}$$

where the set $\mathcal{Q}$ is the set of parameters and g is a monotone operator. We can already see a link between the function f_D and the gap $\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}$.

$$\begin{aligned}
\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}(\mathbf{w},\mathbf{z}) &= \sum_i \mathbf{w}^T\mathbf{F}_i\mathbf{z}_i^* - (\mathbf{w}^*)^T\mathbf{F}_i\mathbf{z}_i - \sum_i (\mathbf{w}^T - (\mathbf{w}^*)^T)\mathbf{f}_i(\mathbf{y}_i) - \sum_i \mathbf{c}_i^T(\mathbf{z}_i-\mathbf{z}_i^*) \\
&= \sum_i (\mathbf{z}_i^*)^T\mathbf{F}_i^T\mathbf{w} - (\mathbf{w}^*)^T\mathbf{F}_i\mathbf{z}_i - \sum_i (\mathbf{f}_i(\mathbf{y}_i))^T(\mathbf{w}-\mathbf{w}^*) - \sum_i \mathbf{c}_i^T(\mathbf{z}_i-\mathbf{z}_i^*) \\
&= \sum_i (\mathbf{z}_i^*)^T\mathbf{F}_i^T(\mathbf{w}-\mathbf{w}^*) - (\mathbf{w}^*)^T\mathbf{F}_i(\mathbf{z}_i-\mathbf{z}_i^*) - \sum_i (\mathbf{f}_i(\mathbf{y}_i))^T(\mathbf{w}-\mathbf{w}^*) - \sum_i \mathbf{c}_i^T(\mathbf{z}_i-\mathbf{z}_i^*) \\
&= \begin{pmatrix} \sum_i \mathbf{F}_i\mathbf{z}_i^* \\ -\mathbf{F}_1^T\mathbf{w}^* \\ \vdots \\ -\mathbf{F}_m^T\mathbf{w}^* \end{pmatrix}^T \begin{pmatrix} \mathbf{w}-\mathbf{w}^* \\ \mathbf{z}_1-\mathbf{z}_1^* \\ \vdots \\ \mathbf{z}_m-\mathbf{z}_m^* \end{pmatrix} - \begin{pmatrix} \sum_i \mathbf{f}_i(\mathbf{y}_i) \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{pmatrix}^T \begin{pmatrix} \mathbf{w}-\mathbf{w}^* \\ \mathbf{z}_1-\mathbf{z}_1^* \\ \vdots \\ \mathbf{z}_m-\mathbf{z}_m^* \end{pmatrix}
\end{aligned}$$

From this, we deduce that the function g from the definition of f_D corresponds to:

$$g(\mathbf{w},\mathbf{z}) = \begin{pmatrix} \sum_i \mathbf{F}_i\mathbf{z}_i^* \\ -\mathbf{F}_i^T\mathbf{w}^* \\ \vdots \\ -\mathbf{F}_m^T\mathbf{w}^* \end{pmatrix} - \begin{pmatrix} \sum_i \mathbf{f}_i(\mathbf{y}_i) \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{pmatrix} = \mathbf{F}\mathbf{u}^* - \mathbf{a} \tag{31}$$

It is constant and thus monotone as required by Nesterov's proof of the convergence of the algorithm. Its Lipschitz constant L is equal to $\max_{\mathbf{u}\in\mathcal{U}}\|\mathbf{F}(\mathbf{u}-\mathbf{u}')\|_2/\|\mathbf{u}-\mathbf{u}'\|_2 \leq \|\mathbf{F}\|_2$. Of course, a point $\mathbf{w},\mathbf{z}$ that satisifies $\|\mathbf{w}\|_2 \leq D_{\mathbf{w}}$ and $\|\mathbf{z}\|_2 \leq D_{\mathbf{z}}$ also satisifies $\|(\mathbf{w},\mathbf{z})\|_2 \leq D$ when $D = \sqrt{D_{\mathbf{w}}^2 + D_{\mathbf{z}}^2}$ since $(\mathbf{w},0) \perp (0,\mathbf{z})$. It is then easy to see that $f_D \geq \mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}$. Thus, the function $\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}$ is upper bounded by the right-hand side of equation 29. We can also observe that the function $g(\mathbf{w},\mathbf{z})$ is exactly the gradient of the objective $\mathcal{L}(\mathbf{w},\mathbf{z})$ at the point $\mathbf{w},\mathbf{z}$.