Extragradient Algorithm for structured predictions

William St-Arnaud

April 15, 2019

1 Preliminaries

In the typical setting for SVM's, we aim to optimize a linear objective over a set of constraints. There are many alernatives to solve that problem, the most popular ones being gradient-based methods. One of the issues with this type of algorithm is that they cannot be applied to matchings and min-cuts. In addition, variants that aim at generating constraints on the fly and incrementally solving new QP's do not typically scale well. The method proposed in this paper, which is in essence a modification of the extra-gradient method by Nesterov, leverages the min-max formulation and takes the dua ; the exponential number of constraints in then transformed an exponential number of variables. However, the paper highlights that we can use only a tractable number of these dual variables to solve the optimization problem. The method prososed in this paper also generalizes the applicability of the usual extra-gradient method to non-euclidean, Bergman projections. We thus end up with a framework that is more flexible and also efficient to implememt.

Since estimating the maximum likelihood over graphical models is often impractical or infeasible for a wide class of problems, we focus our attention on large margin estimation. For a dataset $S = \{(x_i, y_i)\}_{i=1}^m$, where each x_i is an object with a structure (e.g. sequence of words in french), we attempt to find the optimal parameter w of a linear classifier:

$$\mathbf{y}_i = \arg\max_{\mathbf{y}_i' \in \mathcal{Y}} \mathbf{w}^T \mathbf{f}(\mathbf{x}_i, \mathbf{y}_i') \tag{1}$$

The function f gives a feature mapping of a structured object with its corresponding label y_i. The error of a prediction is mesured by a loss function l. To make the loss convex, another term is introduced in the form a hinge loss. Since this gives an upper bound to the loss, it is natural to minimize it. We end up with a problem of the form:

$$\min_{\mathbf{w} \in \mathcal{W}} \sum_i \max_{\mathbf{y}_i' \in \mathcal{Y}_i} \left[\mathbf{w}^T \mathbf{f}_i(\mathbf{y}_i') + l_i(\mathbf{y}_i') \right] - \mathbf{w}^T \mathbf{f}_i(\mathbf{y}_i) \tag{2}$$

The parameters w are also regularized with parameter λ. Since we are optimizing over y_i', we can drop the term from equation 1 and we end up with a loss-augmented inference problem inside the min function. The three types of structure that are presented in the paper have a general formulation that can better be expressed as:

$$\min_{\mathbf{w} \in \mathcal{W}} \max_{\mathbf{z} \in \mathcal{Z}} \sum_i \left(\mathbf{w}^T \mathbf{F}_i \mathbf{z}_i + \mathbf{c}_i^T \mathbf{z}_i - \mathbf{w}^T \mathbf{f}_i(\mathbf{y}_i) \right) \tag{3}$$

where the 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 F_i correspond to the feature mapping for over all labels y_i when multiplied by z_i's. The c_i's correspond to the costs of a z_i and can be identified with the loss l for a label y_i'. Taking the dual, we end up with the following:

$$\begin{aligned} \min_{\mathbf{w} \in \mathcal{W}, (\lambda, \mu) \geq \mathbf{0}} \quad & \sum_i \left(\mathbf{b}_i^T \lambda_i + \mathbf{1}^T \mu_i - \mathbf{w}^T \mathbf{f}_i(\mathbf{y}_i) \right) \\ \text{s.t.} \quad & \mathbf{F}_i^T \mathbf{w} + \mathbf{c}_i \leq \mathbf{A}_i^T \lambda_i + \mu_i \quad i = 1, \ldots, m \end{aligned} \tag{4}$$

The number of variables and constraints is linear in the number of paramters and training data. We already see that this formulation is much more efficient. We do have a set of constraints that is tractable, as is the number of parameters to update. In equation 3, the term that is opitmized is defined as:

$$\mathcal{L}(\mathbf{w}, \mathbf{z}) \triangleq \sum_i \mathbf{w}^T \mathbf{F}_i \mathbf{z}_i + \mathbf{c}_i^T - \mathbf{w}^T \mathbf{f}_i(\mathbf{y}_i) \tag{5}$$

It is bilinear in w and z. We can then imagine two players represented by w and 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.

$$\begin{pmatrix} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \mathbf{z}) \\ -\nabla_{\mathbf{z}_1} \mathcal{L}(\mathbf{w}, \mathbf{z}) \\ \vdots \\ -\nabla_{\mathbf{z}_m} \mathcal{L}(\mathbf{w}, \mathbf{z}) \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & \mathbf{F}_1 & \dots & \mathbf{F}_m \\ -\mathbf{F}_1^T & & & \\ \vdots & & \mathbf{0} & \\ -\mathbf{F}_m^T & & & \end{pmatrix}}_{\mathbf{F}} \underbrace{\begin{pmatrix} \mathbf{w} \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_m \end{pmatrix}}_{\mathbf{u}} - \underbrace{\begin{pmatrix} \sum_i \mathbf{f}_i(\mathbf{y}_i) \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{pmatrix}}_{\mathbf{a}} = \mathbf{Fu} - \mathbf{a} \tag{6}$$

We can measure the "goodness" of the parameters using the gap function $\mathcal{G}$:

$$\mathcal{G}(\mathbf{w}, \mathbf{z}) \triangleq \left[\max_{\mathbf{z}' \in \mathcal{Z}} \mathcal{L}(\mathbf{w}, \mathbf{z}') - \mathcal{L}^* \right] + \left[\mathcal{L}^* - \min_{\mathbf{w}' \in \mathcal{W}} \mathcal{L}(\mathbf{w}', \mathbf{z}) \right] \tag{7}$$

where $\mathcal{L}^*$ gives the result of the min-max of the objective $\mathcal{L}$. 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 $(\hat{u}_{\mathbf{w}}, \hat{u}_{\mathbf{z}}) \in \mathcal{U}$:

$$\mathcal{G}_{D_{\mathbf{w}}, D_{\mathbf{z}}}(\mathbf{w}, \mathbf{z}) = \max_{\mathbf{z}' \in \mathcal{Z}} \left[\mathcal{L}(\mathbf{w}', \mathbf{z}') : d(\mathbf{z}, \mathbf{z}') \leq D_{\mathbf{z}} \right] - \left[\min_{\mathbf{w}' \in \mathcal{W}} \mathcal{L}(\mathbf{w}', \mathbf{z}) : d(\mathbf{w}, \mathbf{w}') \leq D_{\mathbf{w}'} \right] \tag{8}$$

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.

2 Dual Extragradient algorithm

The dual extragradient algorithm from Nesterov gives a convergence guarantee for the objective $\mathcal{L}$. The algorithm can be given by:

```
Initialize: Choose $\hat{\mathbf{u}} \in \mathcal{U}$, set $\mathbf{s}^{-1} = 0$.
**for** $t = 0$ to $t = \tau$ **do**
    $vecv = \mathbf{\Pi}_{\mathcal{U}}(\hat{\mathbf{u}} + \eta \mathbf{s}^{t-1})$
    $\mathbf{u}^t = \mathbf{\Pi}_{\mathcal{U}}(\mathbf{v} - \eta(\mathbf{Fv} - \mathbf{a}))$
    $\mathbf{s}^t = \mathbf{s}^{t-1} - (\mathbf{Fu}^t - \mathbf{a})$
**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 $\mathcal{L}$ 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 $\Pi_{\mathcal{U}}$ in the algorithm simply projects a point back to the set $\mathcal{U}$ by finding the nearest point with respect to the distance metric used.

2.1 Convergence analysis

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

$$\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}(\overline{\mathbf{w}^{\tau}},\overline{\mathbf{z}^{\tau}}) \leq \frac{(D_{\mathbf{w}}+D_{\mathbf{z}})\,L}{\tau+1} \tag{9}$$

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

$$f_D(\mathbf{x}) = \max_{\mathbf{y}\in\mathcal{Q}} \{\langle g(\mathbf{y}), \mathbf{x}-\mathbf{y}\rangle : d(\mathbf{x},\mathbf{y}) \leq D\} \tag{10}$$

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 $\mathcal{G}_{D_{\mathbf{w}},D_{\mathbf{z}}}$.

2.2 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.

2.3 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 $|\mathcal{Z}|gg|\mathcal{W}|$ 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 $|\mathcal{W}|$ (although we now need to store s^t_w). It can be reconstructed using $\bar{u}^t{}_w$. The following figure (**ADD FIGURE 5 FROM PAPER**) illustrates the various dependencies.