

Gradient Methods for Solving Stackelberg Games

Roi Naveiro and David Ríos Insua
ADT Conference

Motivation - Adversarial Regression

- R_J and R_D are two competing wine brands.
- R_D has a system to automatically measure wine quality training a regression over some quality indicators. (Response value: wine quality, Covariates: quality indicators).
- R_J , aware of the actual superiority of its competitor's wines, decides to **hack** R_D 's system by manipulating the value of several quality indicators **at operation time**, to artificially decrease R_D 's quality rates.

Motivation - Adversarial Regression

- R_D is **aware** of the possibility of being hacked and decides to train its regression in an **adversarial robust** manner.
- R_D models this **conflict** as a game between a *learner* (R_D) and a *data generator* (R_J). (Brückner and Scheffer, 2011).
- The *data generator* tries to fool the learner **modifying input data at application time**, inducing a change between the data distribution at training $[p(x, y)]$ and test $[\bar{p}(x, y)]$ times.

The Learner Problem

- Given a feature vector $\mathbf{x} \in \mathbb{R}^p$ and target $y \in \mathbb{R}$, the learner's decision is to choose the weight vector of a linear model $f_w(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$, minimizing **theoretical costs at application time**

$$\theta_l(w, \bar{p}, c_l) = \int c_l(x, y)(f_w(x) - y)^2 d\bar{p}(x, y),$$

- To do so, the learner has a training matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ and a vector of target values $\mathbf{y} \in \mathbb{R}^n$ (a sample from distribution $p(\mathbf{x}, y)$ at training time).

The Data Generator Problem

- The data generator aims at **changing features of test instances** to induce a transformation from $p(x, y)$ to $\bar{p}(x, y)$.
- $z(x, y)$ is the data generator's target value for instance x with real value y
- The data generator aims at **choosing the data transformation** that minimizes the theoretical costs given by

$$\theta_d(w, \bar{p}, c_d) = \int c_d(x, y)(f_w(x) - z(x, y))^2 d\bar{p}(x, y) + \Omega_d(p, \bar{p})$$

Regularized Empirical Costs

- Theoretical costs defined above depend on the unknown distributions p and \bar{p} .
- We focus on their regularized empirical counterparts, given by

$$\hat{\theta}_l(w, \bar{X}, c_l) = \sum_{i=1}^n c_{l,i} (f_w(\bar{x}_i) - y_i)^2 + \Omega_l(f_w),$$
$$\hat{\theta}_d(w, \bar{X}, c_d) = \sum_{i=1}^n c_{d,i} (f_w(\bar{x}_i) - z_i)^2 + \Omega_d(X, \bar{X}).$$

Resulting Stackelberg Game

- We assume the learner acts first, choosing a weight vector w . Then the data generator, after observing w , chooses his optimal data transformation.

$$\begin{aligned} & \arg \min_w \quad \hat{\theta}_l(w, T(X, w, c_d), c_l) \\ \text{s.t.} \quad & T(X, w, c_d) = \arg \min_{X'} \hat{\theta}_d(w, X', c_d) \end{aligned}$$

The general problem

Defender (D) makes decision $\alpha \in \mathbb{R}^n$. Attacker (A), after observing α , makes decision $\beta \in \mathbb{R}^m$

$$\begin{aligned} & \arg \max_{\alpha} \quad u_D[\alpha, \beta^*(\alpha)] \\ \text{s.t.} \quad & \beta^*(\alpha) = \arg \max_{\beta} u_A(\alpha, \beta) \end{aligned}$$

- Many problems formulated this way (not just Stackelberg Games!)
- Our interest is in Algorithmic methods for solving **Stackelberg Games arising in Adversarial Machine Learning** (AML).
- In classical games, decision spaces are small. In AML, α and β usually **high dimensional** and **continuous**.

Gradient Methods

- Forget about analytical solutions!
- **Gradient methods** require computing $\mathbf{d}_\alpha u_D$ (and moving α in the direction of increasing gradient...)

$$\begin{aligned}\mathbf{d}_\alpha u_D &= \partial_\alpha u_D + \partial_{\beta^*} u_D \mathbf{d}_\alpha \beta^*(\alpha) \\ &= \partial_\alpha u_D - \partial_{\beta^*} u_D \left[\partial_\beta^2 u_A \right]^{-1} \cdot \partial_{\alpha\beta}^2 u_A\end{aligned}$$

- Inverting the Hessian has cubic complexity!
- We need a different strategy...

Backward Solution

- Under **certain conditions** (Bottou, 1998), we can approximate our problem by

$$\begin{aligned} & \arg \max_{\alpha} \quad u_D [\alpha, \beta(\alpha, T)] \\ \text{s.t.} \quad & \partial_t \beta(\alpha, t) = \partial_{\beta} u_A [\alpha, \beta(\alpha, t)] \\ & \beta(\alpha, 0) = 0. \end{aligned}$$

- Where $T \gg 1$.
- $\lim_{t \rightarrow \infty} \beta(\alpha, t) = \beta^*(\alpha)$.
- Let's try to solve this problem instead.

Backward Solution

- It can be proved that (Naveiro and Ríos, 2019)

$$d_{\alpha}u_D[\alpha, \beta(\alpha, T)] = \partial_{\alpha}u_D[\alpha, \beta(\alpha, T)] - \int_0^T \lambda(t) \partial_{\alpha} \partial_{\beta} u_A[\alpha, \beta(\alpha, t)] dt$$

- Provided that λ satisfies the **adjoint equation**

$$d_t \lambda(t) = -\lambda(t) \partial_{\beta}^2 u_A[\alpha, \beta(\alpha, t)]$$

- With initial conditions $\lambda(T) = -\partial_{\beta} u_D(\alpha, \beta)$.

Backward Solution

Algorithm 1 Approximate total derivative of defender utility function with respect to her decision using the backward solution

```
1: procedure APPROXIMATE DERIVATIVE USING BACKWARD METHOD( $\alpha, T$ )
2:    $\beta_0(\alpha) = 0$ 
3:   for  $t = 1, 2, \dots, T$  do
4:      $\beta_t(\alpha) = \beta_{t-1}(\alpha) + \eta \partial_\beta u_A(\alpha, \beta) \Big|_{\beta_{t-1}}$ 
5:   end for
6:    $\lambda_T = -\partial_\beta u_D(\alpha, \beta) \Big|_{\beta_T}$ 
7:    $d_\alpha u_D = \partial_\alpha u_D[\alpha, \beta_T(\alpha)]$ 
8:   for  $t = T - 1, T - 2, \dots, 0$  do
9:      $d_\alpha u_D = d_\alpha u_D - \eta \lambda_{t+1} \partial_\alpha \partial_\beta u_A(\alpha, \beta) \Big|_{\beta_{t+1}}$ 
10:     $\lambda_t = \lambda_{t+1} \left[ I + \eta \partial_\beta^2 u_A(\alpha, \beta) \Big|_{\beta_{t+1}} \right]$ 
11:   end for
12:   return  $d_\alpha u_D$ 
13: end procedure
```

Backward Solution - Complexity Analysis

Time complexity

- If $\tau(n, m)$ is the time required to evaluate $u_D(\alpha, \beta)$ and $u_A(\alpha, \beta)$, computing their derivatives requires time $\mathcal{O}(\tau(n, m))$.
- First loop $\mathcal{O}(T\tau(n, m))$.
- Second loop needs computing Hessian Vector Products, by basic results of AD, they have same complexity as function evaluations!
- Thus, overall time complexity is $\mathcal{O}(T\tau(n, m))$.

Space complexity

- We need to store $\beta_t(\alpha)$ for all t .
- $\sigma(n, m)$ is the space requirement for storing each $\beta_t(\alpha)$.
- Overall space complexity $\mathcal{O}(T\sigma(n, m))$.

Forward Solution

- Under **certain conditions**, we can approximate our problem by

$$\begin{aligned} & \arg \max_{\alpha} \quad u_D [\alpha, \beta_T(\alpha)] \\ \text{s.t} \quad & \beta_t(\alpha) = \beta_{t-1}(\alpha) + \eta_t \partial_{\beta} u_A(\alpha, \beta) \Big|_{\beta_{t-1}} \quad t = 1, \dots, T \\ & \beta_0(\alpha) = 0. \end{aligned}$$

- Again, $T \gg 1$.
- $\lim_{t \rightarrow \infty} \beta_t(\alpha) = \beta^*(\alpha)$.

Forward Solution

- Using the chain rule

$$\mathbf{d}_\alpha u_D[\alpha, \beta_T(\alpha)] = \partial_\alpha u_D[\alpha, \beta_T(\alpha)] + \partial_{\beta_T} u_D[\alpha, \beta_T(\alpha)] \mathbf{d}_\alpha \beta_T(\alpha)$$

- To obtain $\mathbf{d}_\alpha \beta_T(\alpha)$ we can sequentially compute

$$\mathbf{d}_\alpha \beta_t(\alpha) = \mathbf{d}_\alpha \beta_{t-1}(\alpha) + \eta_{t-1} \left[\partial_\alpha \partial_\beta u_A(\alpha, \beta) \Big|_{\beta_{t-1}} + \partial_\beta^2 u_A(\alpha, \beta) \Big|_{\beta_{t-1}} \mathbf{d}_\alpha \beta_{t-1}(\alpha) \right]$$

- This induces a dynamical system in $\mathbf{d}_\alpha \beta_t(\alpha)$ that can be iterated in parallel to the dynamical system in $\beta_t(\alpha)$!

Forward Solution

Algorithm 2 Approximate total derivative of defender utility function with respect to her decision using the forward solution.

```
1: procedure APPROXIMATE DERIVATIVE USING FORWARD METHOD( $\alpha, T$ )
2:    $\beta_0(\alpha) = 0$ 
3:    $d_\alpha \beta_0(\alpha) = 0$ 
4:   for  $t = 1, 2, \dots, T$  do
5:      $\beta_t(\alpha) = \beta_{t-1}(\alpha) + \eta \partial_\beta u_A(\alpha, \beta) \Big|_{\beta_{t-1}}$ 
6:      $d_\alpha \beta_t(\alpha) = d_\alpha \beta_{t-1}(\alpha) + \eta_{t-1} \left[ \partial_\alpha \partial_\beta u_A(\alpha, \beta) \Big|_{\beta_{t-1}} + \partial_\beta^2 u_A(\alpha, \beta) \Big|_{\beta_{t-1}} d_\alpha \beta_{t-1}(\alpha) \right]$ 
7:   end for
8:    $d_\alpha u_D = \partial_\alpha u_D[\alpha, \beta_T(\alpha)] + \partial_{\beta_T} u_D[\alpha, \beta_T(\alpha)] d_\alpha \beta_T(\alpha)$ 
9:   return  $d_\alpha u_D$ 
10: end procedure
```

Forward Solution - Complexity Analysis

Time complexity

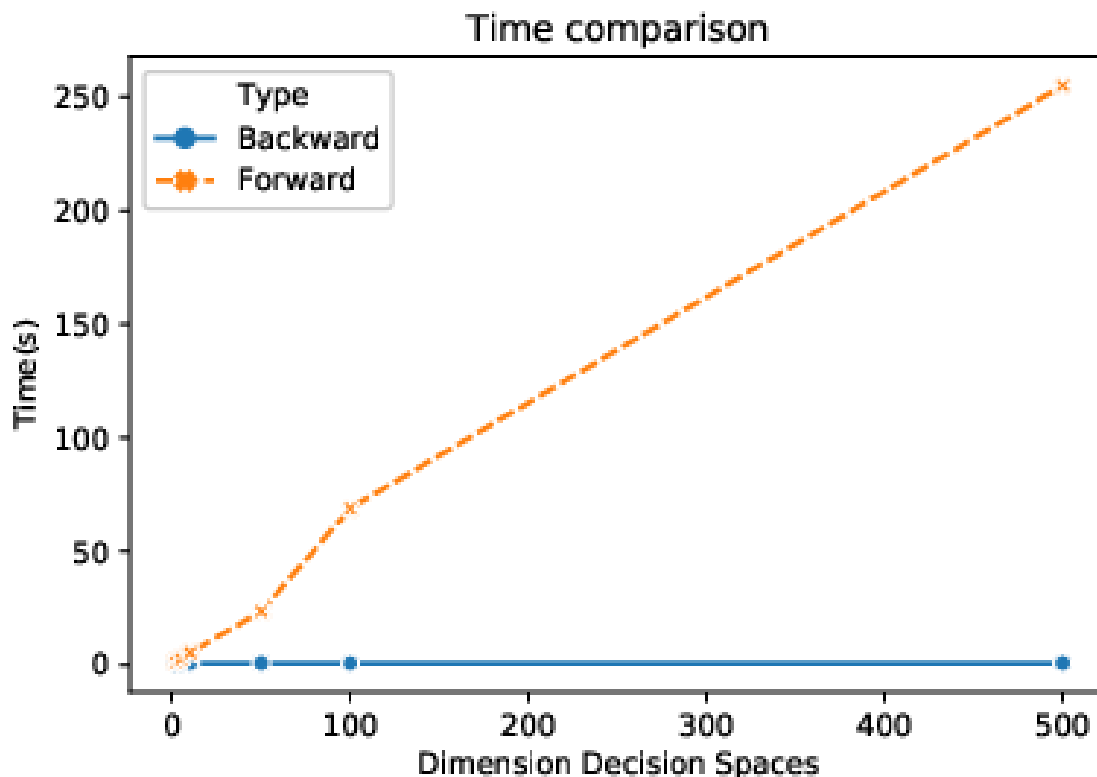
- Computing $\partial_{\beta}^2 u_A(\alpha, \beta)$ requires time $\mathcal{O}(m\tau(m, n))$ as it requires computing m Hessian vector products.
- Computing $\partial_{\alpha}\partial_{\beta}u_A(\alpha, \beta)$ requires computing n Hessian vector products and thus time $\mathcal{O}(n\tau(m, n))$.
- If we compute the derivative in the other way, first we derive with respect to β and then with respect to α , the time complexity is $\mathcal{O}(m\tau(m, n))$.
- Thus, computing $\partial_{\alpha}\partial_{\beta}u_A(\alpha, \beta)$ requires $\mathcal{O}(\min(n, m)\tau(m, n))$.
- Overall, $\mathcal{O}(\max[\min(n, m), m]T\tau(m, n)) = \mathcal{O}(mT\tau(m, n))$.

Space complexity

- The values $\beta_t(\alpha)$ are overwritten at each iteration.
- Overall space complexity is $\mathcal{O}(\sigma(m, n))$.

Conceptual Example

- Attacker's utility is $u_A(\alpha, \beta) = -\sum_{i=1}^n 3(\beta_i - \alpha_i)^2$ and the defender's one is $u_D(\alpha, \beta) = -\sum_{i=1}^n (7\alpha_i + \beta_i^2)$.
- $\mathcal{O}(T\tau(m, n))$ vs $\mathcal{O}(mT\tau(m, n))$.



Application - Adversarial Regression

- We compare ridge regression versus *adversarial robust regression* in the wine problem.
- For ridge regression, we compute the weights in the usual way, and test them in data attacked using those weights.
- For *adversarial robust regression* we compute the weights solving

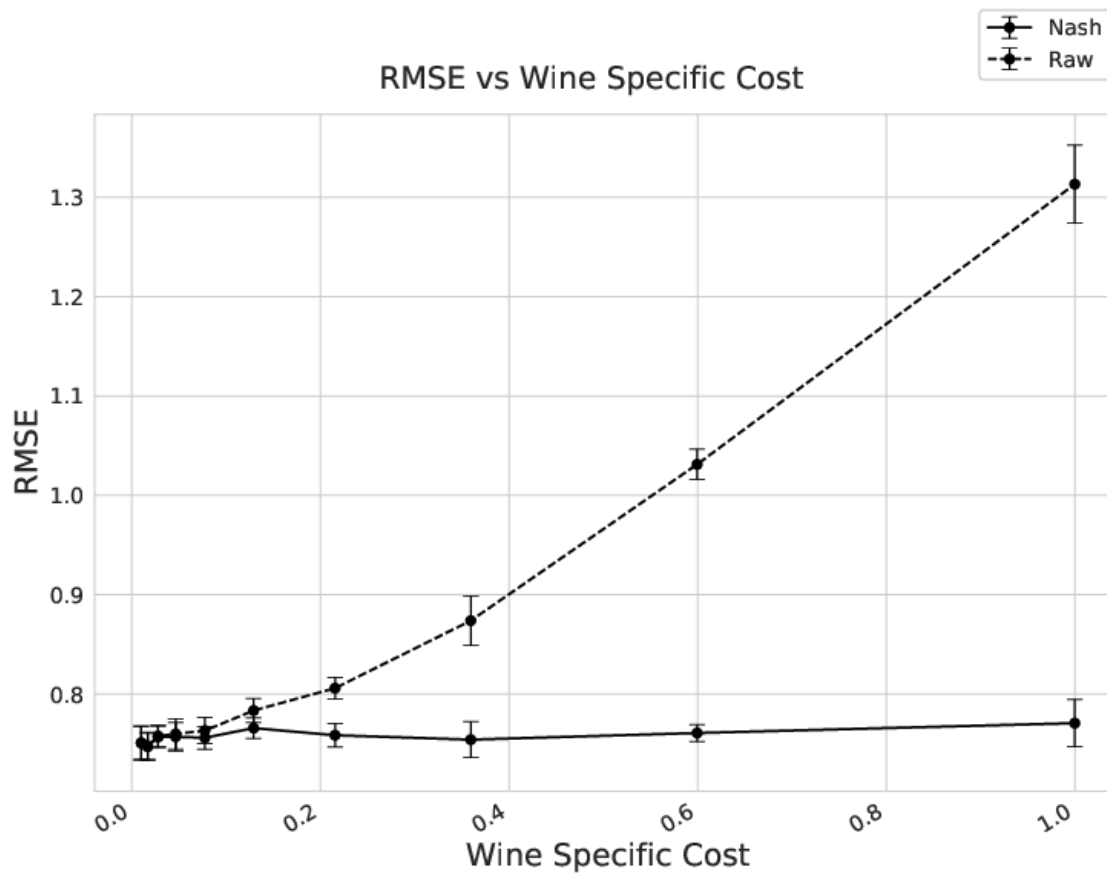
$$\arg \min_w \quad \hat{\theta}_l(w, T(X, w, c_d), c_l)$$

$$\text{s.t.} \quad T(X, w, c_d) = \arg \min_{X'} \hat{\theta}_d(w, X', c_d)$$

and test them in data attacked using those weights.

- Note the dimension of the attacker's decision space is huge! He needs to modify $k = 3263$ data points each with $n = 11$ components!

Adversarial Regression



Conclusions and future work

- New algorithmic method able to solve **huge Stackelberg Games** (dimension of decision sets of the order of 10^4).
- Could be implemented in any **Automatic Differentiation** library (Pytorch, tensorflow...).
- Novel derivation of the backward solution formulating the Stackelberg game as a PDE-constrained optimization problem.
- Application to games with uncertain outcomes.
- Application to Bayesian Stackelberg Games and ARA.
- Several attackers?

Thank you!!

roi.naveiro@icmat.es

www.github.com/roinaveiro/GM_SG