Learning Constraint Approximations to Hard Constrained Problems

Preliminary Notes and Results

James Kotary Ferdinando Fioretto
Syracuse Universty
jkotary@syr.edu ffiorett@syr.edu

1 Introduction

Consider an optimization problem of the form

$$P(\boldsymbol{d}): \quad \underset{x}{\operatorname{argmin}}\, f(x,\boldsymbol{d}) \quad \text{subject to:} \quad H(x,\boldsymbol{d})=0; \;\; G(x,\boldsymbol{d})\leq 0; \;\; x\in\mathcal{X}, \tag{1}$$

where $x\in\mathcal{X}$ is a n-dimensional vector of variables and $\mathcal{X}$ arbitrarily complicated, $\boldsymbol{d}\in\mathcal{D}$ is an m-dimensional vector of input data, and $f:\mathbb{R}^{n+m}\rightarrow\mathbb{R}_+$ is the problem objective. Additionally, H and G describe p equality and q inequality constraints, respectively, $h_i(x,\boldsymbol{d})=0$ $(i\in[p])$ and $g_i(x,\boldsymbol{d})\leq 0$ $(i\in[q])$, each defined on $\mathbb{R}^n\times\mathbb{R}^m$.

The training task is given a dataset $D=\{(\boldsymbol{d}_i,\mathring{x}_i\in P(\boldsymbol{d}_i))\}_{i=1}^N$ of N samples, where $\mathring{x}_i$ is a (possibly) optimal solution to Problem (1) for input $\boldsymbol{d}_i$. The dataset is used to estimate a parametrized model represented by the function $\mathcal{M}$. Given its parameters vector θ and input $\boldsymbol{d}_i$, the model predicts $\hat{x}_i=\mathcal{M}_\theta(\boldsymbol{d}_i)\in\mathbb{R}^n$, a vector of values from the output space of x. In this document, $\mathcal{M}$ represents a deep neural network (DNN).

To be solved efficiently, many constrained optimization problems rely on the development of relaxations or heuristics (referred to as *approximations* here). In power systems, for example, the DC-OPF is a linear approximation to the AC model which is widely adopted in the industry; in scheduling, variants of the Job Shop Scheduling Problem (e.g., with preemptive task processing times) have polynomial-time approximations which allow the (approximate) resolution of large problems. These approximations are important for solving hard constrained optimization problems as well as for facilitating training the learning of constrained problems. While important, building these approximations is a non-trivial task requiring extensive domain and optimization knowledge.

Given a problem P of the form (1), this thread proposes to learn to approximate the nonconvex feasibility region of P by solving the following constrained empirical risk problem:

$$\tilde{P}(D)=\underset{\theta}{\text{Minimize}}\sum_{i=1}^{N}\mathcal{L}(\hat{x}_i,\mathring{x}_i) \quad \text{subject to:} \quad A_\theta\hat{X}\leq b_\theta \tag{2}$$

where $\hat{X}=(\hat{x}_1,\ldots,\hat{x}_N)$ is the vector of prediction $\hat{x}_i$ associated with inputs $\boldsymbol{d}_i$, and A_θ and b_θ are a matrix and vector of appropriate dimensions parametrized by the learnable parameters θ.

The prediction model predicts linear constraints to and solves the resulting optimization problem in the training loop. The solution is targeted to precomputed solutions to $P(\boldsymbol{d})$, using minimum squared error (MSE) loss. The end-to-end model learns to adapt the predicted linear constraints A_θ and b_θ so that resulting solutions $\hat{x}$ are as close to the solutions of $\mathring{x}=P(\boldsymbol{d})$ as possible.

The optimization problem used in-the-loop is currently restricted to Quadratic Programming, using the solution proposed in [1], but general differentiable convex solvers are also available. With this restriction, the predicted problem has the form:

$$\hat{\boldsymbol{x}}=\underset{\boldsymbol{x}}{\operatorname{argmax}} \;\; \boldsymbol{x}^TQ\boldsymbol{x}+q^T\boldsymbol{x} \tag{L1}$$

$$\text{subject to:} \;\; A_\theta\boldsymbol{x}\leq b_\theta \tag{L2}$$

With the above QP prediction, we can capture approximations for the constraint regions of hard optimization problems which have either linear or positive semidefinite quadratic objectives. The parameters of the model are the elements $A_{i,j}$ and b_k of the inequality constraint matrix and right-hand-side vector, al-

lowing 3 parameters to approximate each linear inquality constraint (each of which defines a half-plane).[1]

2 Unit Ball Projection

A simple problem considers projections of points p in 2D onto the unit circle:

$$P = \underset{x}{\text{Minimize}} \ \|x - p\|^2 \ \text{ subject to: } \ \|x\|^2 \leq 1 \tag{3}$$

This problem constitute a good test case for the proposed idea. Indeed, (1) The objective is quadratic, (2) The constrained region is nonlinear but convex, (3) Despite being nonlinear, the projection onto the unit cirle also has a trivial solution method (by diving each point p by its norm) so that accurate targets can be produced.

The predicted problem approximation is:

$$\hat{P}(p) = \underset{x}{\text{Minimize}} \ \|x - p\|^2 \ \text{ subject to: } \ A_\theta \leq b_\theta, \tag{4}$$

where A_θ and b_θ are *learned* parameters, and p, the point to be projected onto, is treated as the input to this optimization. With notation as in (L1-L2), this corresponds to $Q = \frac{1}{2}I$ and $q = -p$. For each training sample p_i, the corresponding label is computed as $\frac{p_i}{\|p_i\|}$. When $P(p_i)$ is the optimal solution solution to (4) with input p_i, we train to minimize

$$\frac{1}{N}\sum_{i=1}^{N} \ell(\hat{P}(p_i), \frac{p_i}{\|p_i\|})$$

where ℓ is the MSE function and the minimization is performed by gradient descent. The loss function above is differentiated only with respect to parameters A_θ and b_θ.

The result of this process is illustrated in Figure 1. The figure illustrates the constraints (gray lines) learned during the early training stages (left) vs the converged results (right) showing how they approximate the feasibility region of the original problem P (highlighted red).

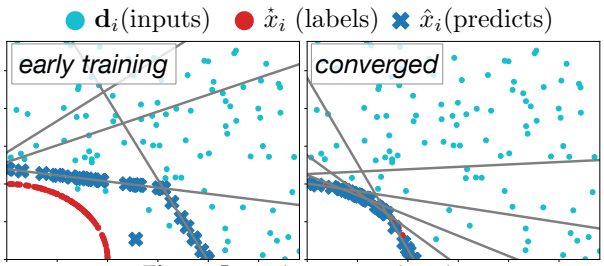

Fig. 1: Learning approximations.

3 Constraint Blocking

We observe that among a set predicted of linear constraints, some do not receive parameter updates during gradient descent, and optimal constraint approximations are not reached. To see why, suppose two parameter triples defining two different constraints $(A_{i,1}, A_{i,2}, b_i)$ and $(A_{j,1}, A_{j,2}, b_j)$. For $k = \{i, j\}$, let

$$S_k = \{x | A_{k,1}x_1 + A_{k,2}x_2 \leq b_k\}.$$

If every point in the half-plane described by constraint i is also captured by constraint j ($S_j \subset S_i$) then the removal of constraint j does not affect (L1 - L2), and likewise neither do small perturbations to the parameters of constraint j. Therefore, the loss gradients with respect to those parameters are 0 (plateau).

The solution adopted recur to the use of Lagrange multipliers to enforce a constraint which requires all linear constraints to have their boundaries tangent to the unit circle at some point. The resulting constraint penalty, applied to constraint i for example, is written as

$$\left| \min_{x \in S_i} \|x - 0\|^2 - 1 \right|.$$

[1] One of the 3 parameters in each line can be dispensed since a line is defined by only 2 values.

This says that the line defining constraint S_i must be at distance 1 from the origin, which is equivalent to saying that the distance from the line to the unit circle is 0 (tangent line). The constraint is enforced by training with the Lagrangian type loss function, with a penalty on the violation of tangentness for each predicted line [2].

The motivation for enforcing these constraints is to overcome the constraint blocking effect, and attract more constraints closer to their proper positions despite their vanishing gradient with respect to the primary training loss function. We observe that the lagrangian penalties can prevent constraints from drifting away from their optimal positions, when reached.

References

[1] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. pages 136–145. PMLR, 2017.

[2] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W.K. Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. A lagrangian dual framework for deep neural networks with constraints. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 18–135, 2020.