Reinforcement Learning Journal Club

Algorithms for inverse reinforcement learning

A. Y. Ng and S. Russell

Inverse reinforcement learning via convex optimization

H. Zhu, Y. Zhang, and J. Boedecker

February 20, 2025

About this talk

- a (very) brief introduction to convex optimization
- an (old) convex formulation of inverse reinforcement learning (CIRL) problems
  - used for behavioral scientific research: *reward fitting* given subject behavior
  - (sadly) not much applications in engineering
- sloppy math
- examples and opinions (some controversial)

Outline

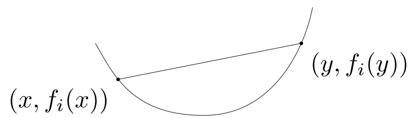
Convex optimization problem

$$
\begin{array}{ll}
\text{minimize} & f_0(x) \\
\text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& Ax = b
\end{array}
$$

- variable $x \in \mathbf{R}^n$
- equality constraints are affine
- $f_0, f_1, \ldots, f_m$ are *convex*: for $0 \leq \theta \leq 1$,

  $$f_i(\theta x + (1-\theta)y) \leq \theta f_i(x) + (1-\theta) f_i(y)$$

  $i.e.$, f_i have *nonnegative (upward) curvature*



why

- effective algorithms, methods (in theory and practice)
- get global solution and *optimality certificate*

Modeling languages for convex optimization

- domain specific languages (DSLs) for convex optimization
  - describe problem in high level human readable language, close to the math
  - can automatically verify problem as convex
  - can automatically transform problem to standard form, then solve

- enables rapid prototyping
- it's now much easier to develop an optimization-based application
- ideal for teaching and research (can do a lot with short scripts)

- gets close to the basic idea: **say what you want, not how to get it**

Implementation

- **CVXPY** (Python): Diamond and Boyd, 2016 [DB16]
- **Convex.jl** (Julia): Udell et al., 2014 [UMZ+14]
- **CVXR** (R): Fu, Narasimhan, and Boyd, 2017 [FNB20]
- **CVX** (Matlab): Grant and Boyd, 2006 [GB14]
- **YALMIP** (Matlab): Lofberg, 2004 [Lof04]

CVXPY example: Non-negative least squares

math:

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 \\ \text{subject to} & x \succeq 0 \end{array}$$

- problem variable: x
- problem data (given): A, b
- $\succeq$: componentwise inequality

CVXPY code:

```
import cvxpy as cp

A, b = ...

x = cp.Variable(n)
obj = cp.norm2(A @ x - b) ** 2
constr = [x >= 0]
prob = cp.Problem(cp.Minimize(obj),
                  constr)
prob.solve()
```

Outline

Introduction to convex optimization

Inverse reinforcement learning via convex optimization

Summary

Preliminaries

Markov decision processes (MDPs)

$$(\mathcal{S},\ \mathcal{A},\ \{P_a\}_{a\in\mathcal{A}},\ r,\ \gamma)$$

- $\mathcal{S}$, $\mathcal{A}$: finite sets of states and actions, with $|\mathcal{S}| = m$, $|\mathcal{A}| = k$
- $\{P_a \in \mathbf{R}_+^{m\times m} \mid P_a\mathbf{1} = \mathbf{1},\ a \in \mathcal{A}\}$: transition probability matrices for all $a \in \mathcal{A}$
- $r \in \mathbf{R}^m$: reward function (or really, vector), which is the *problem variable* and is assumed to be bounded by some positive number $r^{\max} \in \mathbf{R}_{++}$
- $\gamma \in [0, 1)$: discount factor

expert policy $\pi\colon \mathcal{S} \to \mathcal{A}$, assumed to be deterministic with $\pi(s) = a^\star$

value function $v \in \mathbf{R}^m$:

$$v = r + \gamma P_{a^\star} v \implies v = (I - \gamma P_{a^\star})^{-1} r$$

optimality condition

$$
\begin{aligned}
& \pi(s_i) = a^\star \in \operatorname*{argmax}_{a\in\mathcal{A}} p_{i,a}^T v, \quad i = 1,\dots,m \\
\iff \quad & P_{a^\star} v \succeq P_a v, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
\iff \quad & P_{a^\star}(I - \gamma P_{a^\star})^{-1} r \succeq P_a (I - \gamma P_{a^\star})^{-1} r, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
\iff \quad & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\},
\end{aligned}
$$

where $p_{i,a}^T$ denotes the ith row of P_a

The CIRL problem

a trivial formulation of CIRL could be the feasibility problem

$$\begin{array}{ll} \text{find} & r \\ \text{subject to} & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\ & r^{\max} \succeq r \succeq -r^{\max} \end{array}$$

with variable $r \in \mathbf{R}^m$; data $\{P_a\}_{a \in \mathcal{A}}$, γ; hyperparameter $r^{\max}$

- contains trivial ('meaningless') solutions, *e.g.*, $r = c \in \mathbf{R}^m$ with $c_1 = \cdots = c_m$

to find a 'meaningful' reward, consider

$$\begin{array}{ll} \text{minimize} & J(r) + \lambda \phi(r) \\ \text{subject to} & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\ & r^{\max} \succeq r \succeq -r^{\max} \end{array}$$

where $J(r)$, $\phi(r)$ are two criteria that a reward function is considered to be meaningful, and $\lambda \geq 0$ is a hyperparameter

primary objective

$$J(r) = -\sum_{i=1}^{m}\left(p_{i,a^\star}^T v - \sup_{a\in\mathcal{A}\setminus\{a^\star\}} p_{i,a}^T v\right) = -\sum_{i=1}^{m}\inf_{a\in\mathcal{A}\setminus\{a^\star\}}(p_{i,a^\star}^T - p_{i,a}^T)v$$
$$= -\sum_{i=1}^{m}\inf_{a\in\mathcal{A}\setminus\{a^\star\}}\left((p_{i,a^\star}^T - p_{i,a}^T)(I-\gamma P_{a^\star})^{-1}r\right)$$

where $p_{i,a}^T$ denotes the ith row of P_a

- favor reward functions that maximize the margin between the observed expert policy π and all other possible policies at all states

penalty function: ℓ_1-norm, $i.e.$, $\phi(r) = \|r\|_1$

- reward function should be as sparse as possible

put together, we have the convex problem

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{m}\inf_{a\in\mathcal{A}\setminus\{a^\star\}}\left((p_{i,a^\star}^T - p_{i,a}^T)(I-\gamma P_{a^\star})^{-1}r\right) + \lambda\|r\|_1 \\ \text{subject to} & (P_{a^\star} - P_a)(I-\gamma P_{a^\star})^{-1}r \succeq 0, \quad \text{for all } a\in\mathcal{A}\setminus\{a^\star\} \\ & r^{\max} \succeq r \succeq -r^{\max}, \end{array} \qquad (*)$$

Hyperparameter selection

- trade off between J and ϕ by varying λ in $[0, \infty)$
- there exists a value
$$\lambda^{\max} = \inf_{z \in \partial J(0)} \|z\|_\infty$$
(where $\partial J(0)$ is the subdifferential of $J(r)$ at $r = 0$), such that if $\lambda \geq \lambda^{\max}$, the optimal of (*) is achieved at $r = 0$
  - $\lambda^{\max}$ can be obtained via iterative methods, such as bisection
- find the 'simplest' r given the problem data by setting $\lambda = \lambda^{\max}$

Implementation

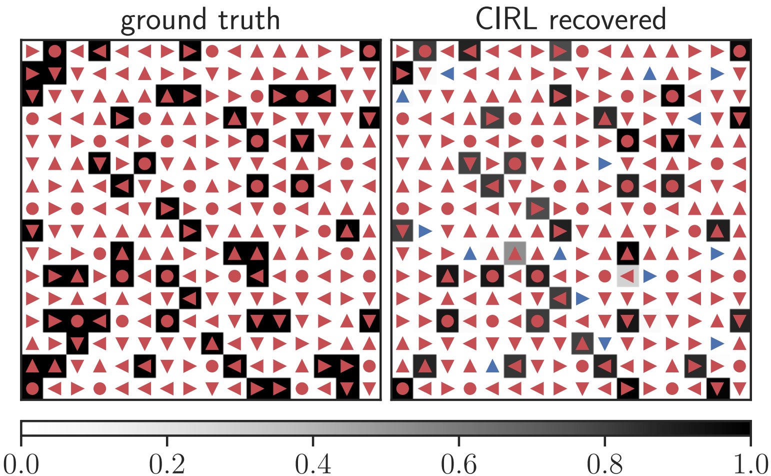
transforming $(*)$ into epigraph form

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s + \lambda \|r\|_1 \\ \text{subject to} & \begin{bmatrix} p_{i,a^\star}^T - p_{i,\tilde{a}_1}^T \\ \vdots \\ p_{i,a^\star}^T - p_{i,\tilde{a}_{k-1}}^T \end{bmatrix} (I - \gamma P_{a^\star})^{-1} r + s_i \succeq 0, \quad i = 1, \ldots, m \\ & (P_{a^\star} - P_{\tilde{a}_i})(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad i = 1, \ldots, k-1 \\ & r^{\max} \succeq r \succeq -r^{\max} \end{array}$$

- ℓ_1-regularized linear program over variables: $r \in \mathbf{R}^m$ and $s \in \mathbf{R}^m$

```python
import numpy as np
import cvxpy as cp
# problem information (input from user)
m = None  # number of states
gamma = None  # discount factor
Pastr = None  # transition matrix of the optimal action
lPa = []  # list of transition matrices of the other actions
# hyperparameters (input from user)
rmax = None  # reward function bound
lbd = None  # scalarization weight
r = cp.Variable(m)
s = cp.Variable(m)
constraints = []
H = np.linalg.inv(np.identity(m) - gamma * Pastr)
D = np.array([[Pastr[i] - Pa[i] for Pa in lPa] for i in range(m)])
for i in range(m):
    constraints.append(D[i] @ H @ r + s[i] >= 0)
for Pa in lPa:
    constraints.append((Pastr - Pa) @ H @ r >= 0)
constraints.append(rmax >= r)
constraints.append(r >= -rmax)
obj = cp.Minimize(cp.sum(s) + lbd * cp.norm(r, 1))
prob = cp.Problem(obj, constraints)
prob.solve()
```

Example: Gridworld



- $\lambda = 2$, $r^{\max} = 100$
- 0.85 cosine similarity between true and recovered reward
- solved in 1.65 seconds

Outline

Summary

advantages of CIRL formulation:

- global optimality guarantee with certificate
- very easy to implement (though might need some effort to figure out the math)
- very fast for moderate dense problems (can be fast for large problems if sparsity pattern exists in the data)

limitations:

- require deterministic expert policy
- infeasible (or feasible only for $r = 0$) if expert policy is strongly suboptimal

in the paper but not covered here:

- incorporating function approximations ($e.g.$, for continuous $\mathcal{S}$ and $\mathcal{A}$)
- learning from trajectories ($i.e.$, no analytical expert policy available)

Reference

[BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[FNB20] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94:1–34, 2020.

[GB14] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, 2014.

[Lof04] J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No. 04CH37508)*, pages 284–289. IEEE, 2004.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE, 2014.