Enforcing robust control guarantees within neural network policies

Carnegie Mellon University

Priya L. Donti[1], Melrose Roderick[1], Mahyar Fazlyab[2], and J. Zico Kolter[1,3]

[1] Carnegie Mellon University [2] Johns Hopkins University [3] Bosch Center for AI

Johns Hopkins University

We present a method for provably robust control via deep RL, which embeds a differentiable projection layer into a neural network policy in order to enforce robust control stability criteria.

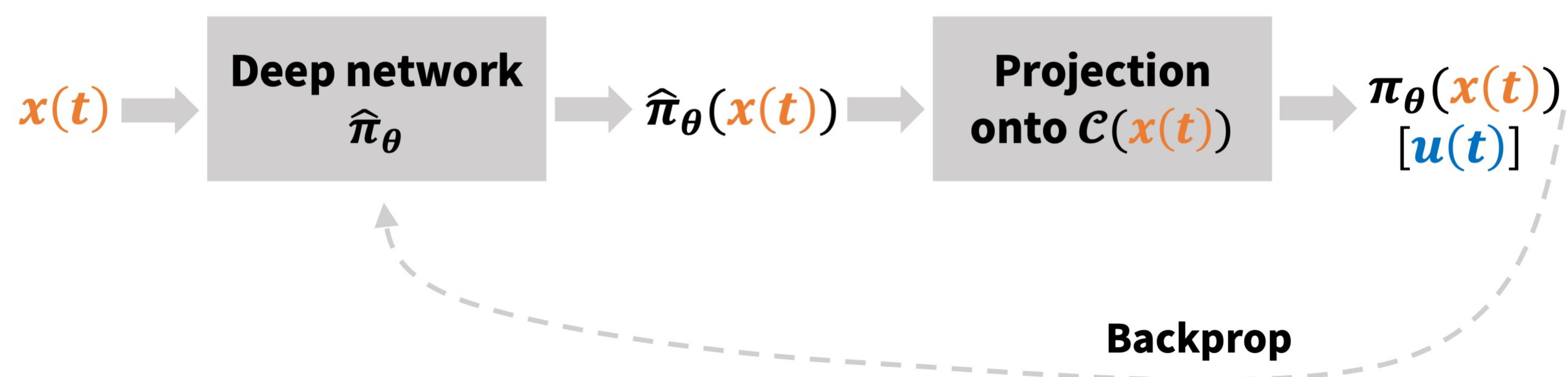
Motivation

Deep RL methods often give no safety or **stability guarantees**
→ Dealbreaker for safety-critical systems (e.g., airplanes, power grids)

Robust control gives provably stabilizing policies, but they are simple (e.g., linear)
→ Limited overall **performance**

Goal: **Bridge the gap** by enforcing robust control criteria within neural network policies trained via RL

Approach



System: Unknown, nonlinear with uncertainty bound, e.g. NLDI:
$$\dot{x}(t) \in Ax(t) + Bu(t) + Gw(t) \text{ s.t. } \|w(t)\|_2 \le \|Cx(t) + Du(t)\|_2$$

Step 1: Construct set of stabilizing actions
- Obtain Lyapunov function V via robust control
- Compute $\mathcal{C}(x(t)) \coloneqq \{\, u(t) \mid \dot{V}(x(t)) \le -\alpha V(x(t)) \;\forall t \,\}$

Step 2: Construct policy π_θ
- Construct deep network $\hat{\pi}_\theta$
- Policy is then $\pi_\theta(x(t)) = \mathrm{Proj}_{\mathcal{C}(x(t))}(\hat{\pi}_\theta(x(t)))$

Step 3: Train end-to-end using deep RL techniques
- Gradient through projection via implicit function thm

Related work

Safe RL: Aims to learn "safe" control policies by making smoothness assumptions about dynamics; no provable guarantees

Robust control + RL: Efforts combining control-theoretic ideas with RL. Predominantly limited to H_∞ control.

Differentiable optimization layers: NN layers with optimization problem as forward pass, and backward pass via implicit function theorem. We employ such layers in our work.

Results

Environment		LQR	MBP	PPO	Robust LQR	Robust MPC	RARL	Robust MBP*	Robust PPO*
		Non-robust methods			*Robust methods*				
Generic NLDI ($D=0$)	O	373	**16**	21	253	253	**27**	69	33
	A		——— *unstable* ———		1009	873	*unstable*	1111	2321
Generic NLDI ($D \neq 0$)	O	278	**15**	82	199	199	147	**69**	80
	A		——— *unstable* ———		1900	1667	*unstable*	1855	1669
Cart-pole	O	36.3	**3.6**	7.2	10.2	10.2	**8.3**	9.7	8.4
	A	— *unstable* —		172.1	42.2	47.8	41.2	50.0	16.3
Quadrotor	O	5.4	**2.5**	7.7	13.8	13.8	12.2	11.0	**8.3**
	A	*unstable*	545.7	137.6	64.8	*unstable*†	63.1	25.7	26.5
Microgrid	O	4.59	**0.60**	0.61	0.73	0.73	0.67	**0.61**	**0.61**
	A		——— *unstable* ———		0.99	0.92	2.17	7.68	8.91

We test our method under two settings:
- Original dynamics ("average case")
- Adversarial dynamics ("worst case")

Our method
- **improves "average-case" performance** over robust baselines
- **remains stable under "worst-case" dynamics** (unlike non-robust baselines)