Gradient-Based Planning with Learned Forward Models

Mikael Henaff, Will Whitney and Yann LeCun

New York University

Introduction

- Model-free RL has become very popular for action planning
- But suffers from limitations
  - High sample complexity
  - Unstable
  - Need to query environment
- Planning with forward models has several advantages
  - More informative gradient
  - Can learn on outside data
  - Reuse of model across different tasks

Introduction

- Model-based planning has been successful when actions are continuous
- Cannot easily be used for discrete actions
- Introduce a method to perform optimization in discrete action spaces
- Also show some results when action space is continuous

Training a Forward Model

- Given
  - initial state s_0
  - action a
  - model $f(s_0, a, \theta)$
- Forward Model maps state-action pairs to future states
- Loss $\mathcal{L}(s, s')$ measures difference between predicted and true future states
- Train Forward Model:

$$\arg\min_{\theta} \quad \mathbb{E}_{(s_0,a,s')\sim\mathcal{E}}\left[\mathcal{L}(f(s_0, a, \theta), s')\right]$$

Planning with Forward Models

- After the model is trained, it can be used for planning
- Fix initial state s_0 and desired future state s'
- Find required action(s)

$$\arg\min_{a} \quad \mathcal{L}(f(s_0, a, \theta), s')$$

- If f is differentiable, can get gradients wrt actions

Navigation Task (Discrete)

Table : Modification of a QA task to become a planning task. Asterisks indicate elements which the method must fill in.

Forward Modeling Task	Planning Task
agent1 is at (8,4)	agent1 is at (8,4)
agent1 faces-E	agent1 faces-E
agent1 moves-2	*
agent1 moves-5	*
agent1 faces-S	*
agent1 moves-5	*
agent1 faces-S	*
agent1 moves-1	*
agent1 moves-1	*
agent1 moves-1	*
where is agent1?	where is agent1?
*	(10,1)

Navigation Task (Discrete)

- Used Recurrent Entity Network (EntNet) as forward model
- Bank of gated RNNs with shared weights
- 1-hop Memory Net on top
- Gets $< 1\%$ error on forward modeling task

Planning Discrete Actions with Backprop

- Discrete actions can be encoded as 1-hot vectors

$$\mathcal{A} = \{e_1, ..., e_d\} \tag{1}$$

- If we just optimize using gradients wrt actions, the solution may not be a unit vector

Planning Discrete Actions with Backprop

- Need to solve:

$$\begin{aligned} \underset{a=(a_1,\ldots,a_T)}{\arg\min} \quad & \mathcal{L}(f(s_0, a, \theta), s') \\ \text{subject to} \quad & a_t \in \{e_1, \ldots, e_d\} \end{aligned} \tag{2}$$

- Rewrite

$$\begin{aligned} \mathcal{A} &= \{e_1, \ldots, e_d\} \\ &= \{p : \mathcal{H}(p) = 0\} \\ &= \{\sigma(z) : z \in \mathbb{R}^d, \mathcal{H}(\sigma(z)) = 0\} \end{aligned}$$

- σ is softmax, $\mathcal{H}$ is entropy

Planning Discrete Actions with Backprop

- Problem is now:

$$\begin{aligned} \underset{z=(z_1,\ldots,z_T)}{\arg\min} \quad & \mathcal{L}(f(s_0, \sigma(z), \theta), s') \\ \text{subject to} \quad & \mathcal{H}(\sigma(z_t)) = 0 \end{aligned}$$

- Relaxing constraints gives:

$$\underset{z=(z_1,\ldots,z_T)}{\arg\min} \quad \mathcal{L}(f(s_0, \sigma(z), \theta), s') + \sum_t \mathcal{H}(\sigma(z_t))$$

- After optimizing, quantize each $\sigma(z_t)$ to closest unit vector

Planning Discrete Actions with Backprop

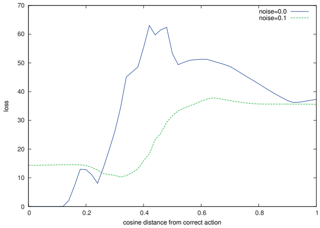
- There may points with low loss very close to unit vectors with high loss
- I.e. "adversarial examples"
- Add noise to one-hot vectors when training forward model
  - Expand low loss region around good actions
  - Expand high loss region around bad actions

Planning Discrete Actions with Backprop

Table : Success rate for Navigation Planning Task, for different length sequences

T	5	10	20
Random	0.195	0.063	0.041
EntNet, $\sigma = 0$	0.517	0.282	0.223
EntNet, $\sigma = 0.01$	0.544	0.324	0.253
EntNet, $\sigma = 0.1$	**0.621**	0.473	**0.390**
Q-learner	0.583	**0.492**	0.352

Planning Discrete Actions with Backprop



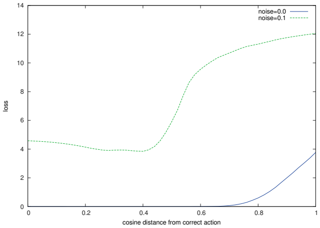


Figure : Loss surface along the curve connecting a correct action (left) to an incorrect action (right). Blue represents model trained without noise, green with noise.

Future work

- More complex discrete planning tasks
- Modify bAbI tasks to become planning
- Others?
- Different distributions for training forward model and planning

Planning for Pool (Continuous)

- Two balls on a board
- Can collide and bounce off walls
- Described by positions $\{s_t^i\}$ and velocities $\{v_t^i\}$
- Forward Model:
  - Given initial positions $s_0 = (s_0^1, s_0^2)$ and velocities $a = (v_0^1, v_0^2)$
  - Predict positions $s = (s_t^1, s_t^2)$ for next 50 timesteps

Planning for Pool (Continuous)

- Planning Task: sink ball 2 using ball 1
  - Initial positions of both balls $s_0 = (s_0^1, s_0^2)$
  - Target position of ball 2 at given timestep s^*
  - Find velocity to apply to ball 1 to minimize

$$\mathcal{L} = ||s_{35}^2 - s^*||_2 \tag{3}$$

- Must make use of collision, get angle and timing right

Planning for Pool

- Forward model: Interaction Network, 150 hidden units
- Can apply backprop easily since action space in continuous
- Baseline: REINFORCE agent with 4 layers, 150 hidden units.

Planning for Pool

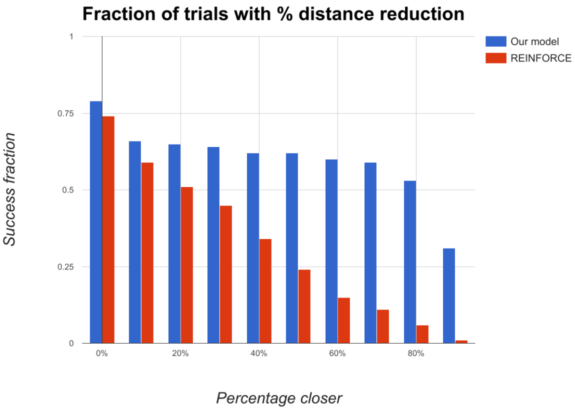


Figure : Comparison of the performance of our method versus REINFORCE. Column height represents the fraction of trials where the agent moved the ball $x\%$ of the way to the target. REINFORCE is evaluated by its maximum-likelihood action. Our model manages to get the ball very close to the target much more often than REINFORCE.

Planning for Pool

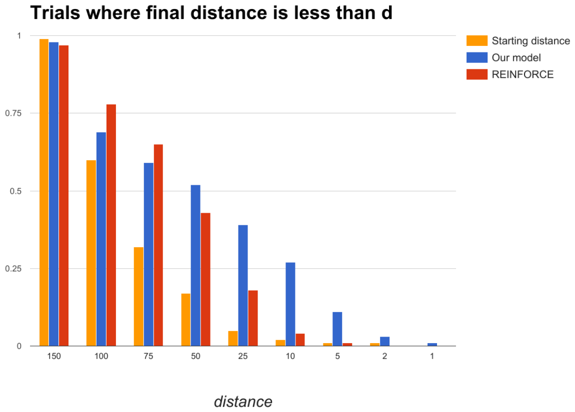


Figure : Comparison of the performance of our method versus REINFORCE. Column height represents the fraction of trials where the agent got the ball within *d* pixels. REINFORCE is evaluated by its maximum-likelihood action. Our model gets the ball within 10px of the target 25% of the time, while REINFORCE succeeds at this level in only 4% of trials.

Planning for Pool