TEXPLORE and APPLD Learning Presentation

Nice Wang
wangxiaonannice@gmail.com
2023.03

TEXPLORE: real-time sample-efficient reinforcement learning for robots

Challenges among RL to make it generally applicable to Robot control tasks:

1. Limited exploration: learn from few samples
2. Learn from continuous state representations
3. In face of sensor/actuator delays
4. Learn while taking actions continually in real-time (computationally efficient)

MDP formalism for the task:

$\mathcal{M}(S, A, R, T)$, $\quad S\!: states,$ $\quad A\!: actions,$ $\quad R(s,a)\!: reward,$ $\quad T\!: transition$

a. For transition, that is posterior: $T(s,a,s') = P(s'|s,a)$

b. Policies come from value function:

Bellman equation: $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$

Real-time

Learn delay information

Sample-efficient

The Optimal policy: $\pi(s) = \text{argmax}_a Q^*(s,a)$

Solutions in TEXPLORE:

1. Multi-threaded parallel architecture and MCTS -> RTMBA Parallel Architecture with UCT(λ) Planning Method
2. Model learning and predicting:
   a. supervised learners:
      n feature models: $s_i^{rel} = featModel_i(s,a)$ corresponds to ith feature of $s' - s$ learning relative transition
      one reward model: $r = rewardModel(s,a)$

Better select inputs with correct delay

   b. decision tree algorithm
   c. using supervised model's ability of generalization: make predictions for unseen or infrequently visited states
3. k-Markov approach: additionally use k previous actions for search
4. Linear regression trees to model continuous domains
5. Random forests to provide targeted, limited exploration:
   a. m trees for each model
   b. each tree update each with prob. ω } → randomness when choosing splits
   c. when planning: $Q(s,a) = \frac{1}{m}\sum_{i=1}^{m} R_i(s,a) + \gamma \frac{1}{m}\sum_{i=1}^{m}\sum_{s'} P_i(s'|s,a) \max_{a'} Q\ (s',a')$ → { toward $s-a$ with higher values explore optimistic; avoiding $s-a$ has low values avoid pessimistic }
   d. decision tree's ability of generalization

suboptimal

RTMBA Parallel Architecture with UCT(λ) Planning Method ⇨ Provide actions continually in real-time at whatever frequency is required.

Suppose that $maxDepth = 4$, rollout from state s_0 at depth 0 as following (Top-down and then Bottom-up):

eg. when $d = 0$

	Time Frame	t_0	t_1	t_2
Thread	Action	$s_0 \rightarrow S$	$s_0, Q_0^t \rightarrow a_0$	$\langle s_0, s_1, a_0, r\rangle \rightarrow updateList$
	Planning	wait	$s_0, M_0 \rightarrow s_1, r$	free
	Learning	training	free	wait
Mutex Lock	S	write-lock	read-lock	free
	Q	free	read-lock	free
	M	free -> write-lock	read-lock	free
	updateList	read-lock -> free	free	write-lock

eg. when $d = 0$

Planning	$Q_0^t \rightarrow Q_0^{t+1}$
Q	write-lock

Top Down

$d = 0$	$s_0, Q_0^t \rightarrow a_0$	$s_0, M_0 \rightarrow s_1$	$M_0, \langle s_0, s_1, a_0, r\rangle \rightarrow M_1$
$d = 1$	$s_1, Q_1^t \rightarrow a_1$	$s_1, M_1 \rightarrow s_2$	$M_1, \langle s_1, s_2, a_1, r\rangle \rightarrow M_2$
$d = 2$	$s_2, Q_2^t \rightarrow a_2$	$s_2, M_2 \rightarrow s_3$	$M_2, \langle s_2, s_3, a_2, r\rangle \rightarrow M_3$
$d = 3$	$s_3, Q_3^t \rightarrow a_3$	$s_3, M_3 \rightarrow s_4$	$M_3, \langle s_3, s_4, a_3, r\rangle \rightarrow M_4$

Bottom Up

$$Q_0^{t+1} = \alpha r + \alpha\gamma\{\lambda r + \lambda\gamma\{\lambda[r + \gamma(\lambda r + (1-\lambda)\boxed{a_{d=3}})] + (1-\lambda)\boxed{a_{d=2}}\} + (1-\lambda)\boxed{a_{d=1}}\} + (1-\alpha)\boxed{Q_0^t}$$

$$Q_1^{t+1} = \alpha r + \alpha\gamma\{\lambda[r + \gamma(\lambda r + (1-\lambda)\boxed{a_{d=3}})] + (1-\lambda)\boxed{a_{d=2}}\} + (1-\alpha)\boxed{Q_1^t}$$

$$Q_2^{t+1} = \alpha[r + \gamma(\lambda r + (1-\lambda)\boxed{a_{d=3}})] + (1-\alpha)\boxed{Q_2^t}$$

$$Q_3^{t+1} = \alpha r + (1-\alpha)\boxed{Q_3^t}$$

Re-planning from history Q
-> Speed up Q learning for new M

Note:

1. Q_3^t: $Q(s_3, -)$ after updating t times
2. $a_{d=3} = \max_a Q_3^{t+1}$

Thinking of Future Work of TEXPLORE

Supplementary Advantages of TEXPLORE:

1. Plan actions accordingly to a state-action visit-count to favor less-visited states

Shortages of TEXPLORE:

1. Random forests not quickly when effects of actions not generalization across states
2. Suboptimal:
    a. s-a pair which can't be predicted from neighbors will not be learned
    b. sometimes without high-rewarding s-a pairs
3. Partially observable

Some Thinking of Future Works:

1. For partially observable:
    a. stochastic transition for unobservable
    b. POMDPs
2. Dealing with continuous actions
3. Developmental and lifelong learning, external rewards
4. Resetting visit counts according to model change frequency
5. Further parallelization with multi-cores

summarized by the paper

6. Modeling delays themselves as part of the state-space (reference [1]):
    a. MDP -> RDMDP (random delay MDP):
        $RDMDP(MDP, p_\omega, p_\alpha)$, p_ω: observation delay, p_α: action delay
    b. Delay-Correcting Actor-Critic (DCRC): use off-policy multi-step value estimation
7. Gated decision trees applied in model-based reinforcement learning (eg. MoET, reference [2])
8. Dealing with task that has high dimensionality of the action space: refer to RTE (reference [3])

ideas from other literature

9. More suitable models (may such as time sequential prediction model) used in model-based reinforcement learning
10. Online and few-shot learning with few samples within the framework of robotic reinforcement learning

my own ideas

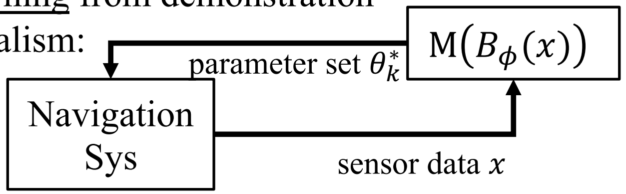
Reference:

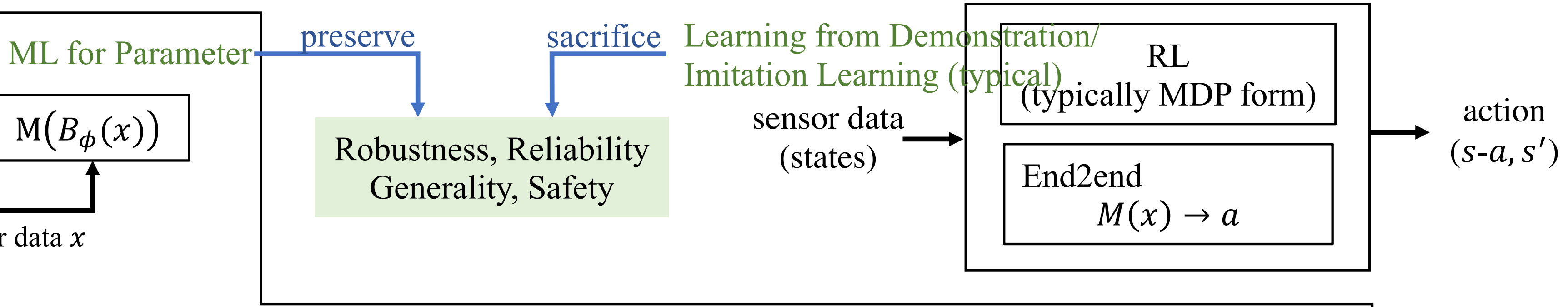
[1] Bouteiller, Yann, et al. "Reinforcement learning with random delays." International conference on learning representations. 2021.
[2] Vasić, Marko, et al. "MoËT: Mixture of Expert Trees and its application to verifiable reinforcement learning." Neural Networks 151 (2022): 34-47.
[3] Chatzilygeroudis, Konstantinos, Vassilis Vassiliades, and Jean-Baptiste Mouret. "Reset-free trial-and-error learning for robot damage recovery." Robotics and Autonomous Systems 100 (2018): 236-250.

APPLD: Adaptive Planner Parameter Learning from Demonstration

APPLD:

1. Parameter learning from demonstration
2. Problem formalism:





3. Two-Period learning:
   a. context learning – demonstration segmentation:
      1) changepoint detection: $D = \{x_i^D, a_i^D, t_i^D\}_{i=1}^N, A_{segment} \rightarrow \{D_k = \{x_i^D, a_i^D, t_i^D | \tau_{k-1} \le i < \tau_k\}\}_{k=1}^K$
      2) supervised multi-classification model for context target $k, k \in [1, K]$: $\{D_k, k\}_{k=1}^K \rightarrow B_\phi(x)$:

$$\phi^* = \mathrm{argmax}_\phi \sum_{i=1}^N \log \frac{\exp(B_\phi(x_i^D)[c_i])}{\sum_{c=1}^K \exp(B_\phi(x_i^D)[c])} \rightarrow \begin{cases} \text{maximum likelihood estimation} \\ \text{may change to min cross-entropy and use SGD for } \phi \text{ update} \end{cases}$$

   b. parameter learning with certain context:
      for context k from 1 to K, linear regression method for optimal θ_k^* learning: $\theta_k^* = \mathrm{argmin}_\theta \sum_{(x,a)\in D_k} ||a - G_\theta(x)||_H$
4. Online inference:

$$x, M(B_\phi(x)) \rightarrow \theta_k^* : \theta_t = \mathrm{M}\{\mathrm{argmax}_c B_\phi(x_i)[c], t - p < i \le t\}$$

Some Thinking of Future Works:

1. Clustering similar contexts together
2. Perform parameter learning and changepoint detection jointly
3. Param-learning from interventions/evaluative feedback (ref [1-2])
4. Reinforcement learning approach (for param or policy, ref [3-4])

Reference:

[1] Wang, Zizhao, et al. "Appli: Adaptive planner parameter learning from interventions." 2021 IEEE international conference on robotics and automation (ICRA). IEEE, 2021.

[2] Wang, Zizhao, et al. "Apple: Adaptive planner parameter learning from evaluative feedback." IEEE Robotics and Automation Letters 6.4 (2021): 7744-7749.

[3] Xu, Zifan, et al. "Applr: Adaptive planner parameter learning from reinforcement." 2021 IEEE international conference on robotics and automation (ICRA). IEEE, 2021.

[4] Spencer, Jonathan, et al. "Expert Intervention Learning: An online framework for robot learning from explicit and implicit human feedback." Autonomous Robots (2022): 1-15.