Reinforcement Learning for Cyber-Physical Systems

Anand Balakrishnan   Jyo Deshmukh

Spring 2019 CSCI 599: Autonomous Cyber-Physical Systems

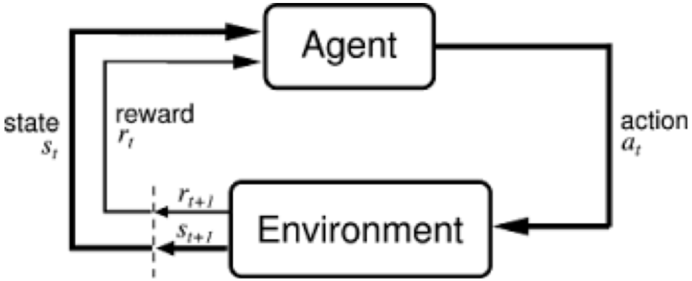
March 25, 2019

Outline

Outline

What is Reinforcement Learning?

- Reinforcement Learning is a framework for algorithms designed to train an *agent* on a task through repeated interaction with the environment.
- This method is inspired by ideas in cognitive sciences and behavioral psychology, as suggested by the titles of some of the early work done in this field (Barto, Sutton, and Anderson Bertsekas and Tsitsiklis).



What is Reinforcement Learning?

- RL is more related to *Optimal Control Theory* than it is to Machine Learning.
- It was initially developed as a method to learn controllers (or *policies*) for stochastic systems[1] (Kumar and Varaiya).
- In RL, we typically model the environment as a *Markov Decision Process (MDP)* and say that the goal is to learn a policy, π, that learns to “solve” the MDP.

[1] systems modeled with random noise in their observations and their control vectors

Outline

Definition (Markov Decision Process)

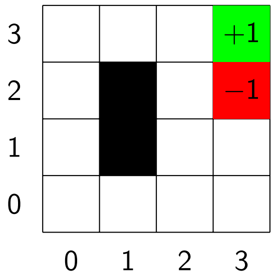
An MDP is a tuple $M = \langle S, A, P, R \rangle$ where

- S is the state space of the system;
- A is the set of actions that can be performed on the system;
- $P : S \times A \times S \to [0, 1]$ is the transition probability function such that, $P(s, a, s') = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s, a_t = a\right]$;
- R is a reward function that typically maps either some $s \in S$ or some transition $\delta \in S \times A \times S$ to $\mathbb{R}$.

- In the standard RL problem (Sutton and Barto), a learning agent interacts with a MDP.
- The state, action, and reward at each time $t \in \{0, 1, 2, \dots\}$ are denoted $s_t \in S$, $a_t \in A$, and $r_t \in \mathbb{R}$ respectively.
- We also denote the expected reward received from taking an action $a \in A$ in state $s \in S$ to be

$$R(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a]$$

Example: MDP



- $S = \{(i,j) | i,j \in [0,3]\}$,i.e., each cell in the above grid.
- $A = \{\text{UP, DOWN, LEFT, RIGHT}\}$.[2]
- $P(\text{success}) = 0.8$, else choose an orthogonal direction;
- R: $+1$ for goal (green) and -1 for fail (red), else 0.
- We will use this MDP as a running example.

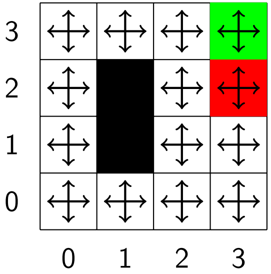
[2] if agent is trying to move into an obstacle, the agent shouldn't be allowed to move

Definition (Policy)

$\pi : S \to A$ is a function that outputs an action a given a state s.

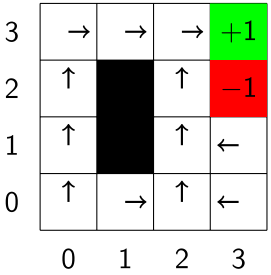
- We can also denote the policy to be a probability distribution $\pi : S \times A \to [0, 1]$ such that $\pi(s, a) = \mathbb{P}\left[a_t = a \mid s_t = s\right]$
- Thus, in the deterministic case, the chosen action will be the mode of the probability distribution represented by π.

Random Policy $\pi(s, a) = 0.25$



- A Random agent is one that uniformly picks an action from the action space A.
- Here, there is no use of the reward function.

Policy under deterministic MDP ($P(\text{success}) = 1$)



- In the deterministic MDP case, you can use your favorite path planning algorithm (Dijkstras, A*, D*, Bellman-Ford, etc.) to find a the optimal policy.
- We learn a policy π such that $\pi(s, a) = 1$ for correct action, 0 otherwise.

Definition (Value Function)

A function $V^\pi : S \to \mathbb{R}$ that outputs the total (expected) reward obtained from starting a MDP at a certain state s and choosing actions from a policy π, until a terminal state. [3].

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \;\middle|\; s_0 = s\right] \tag{1}$$

where $\gamma \in [0, 1]$ is the discounting factor.

Definition (Optimal Value Function)

Among all possible value functions, there exists an **optimal value function** such that

$$V^*(s) = \max_\pi V^\pi(s)$$

[3] In future, we will drop the subscript π without loss of generality

Definition (Action-Value or Q Function)

A function $Q^\pi : S \times A \rightarrow \mathbb{R}$ that outputs the expected value of the given state s, if we take action a.

$$Q^\pi(s, a) = \mathbb{E}_\pi[R(s, a) + V^\pi(s_{t+1}) \mid s_t = s, a_t = a] \tag{2}$$

where $R(s, a)$ is the expected random reward associated with the state-action pair (s, a).

- The Q-function is a measure of how good it is for an agent to pick an action a in state s so as to maximize $V(s)$.
- The optimal Q-function $Q^*(s, a)$ means the expected total reward received by an agent starting in sand picks action a, then will behave optimally afterwards.

Bellman Equation

- Richard Bellman showed that a dynamic optimization problem in discrete time can be stated in a recursive, step-by-step form known as backward induction by writing down the relationship between the value function in one period and the value function in the next period.
- This recursive relationship is called a **Bellman equation**.
- In *dynamic programming*, the existence of a Bellman equation is a necessary condition to learn an optimal solution to a problem.

Bellman equation for Value functions

We can provide a Bellman equation for the optimal Q-function by the following relation:

$$\begin{aligned} Q^{*}(s,a) &= R(s,a) + \gamma \mathbb{E}_{s'}\left[V^{*}(s')\right] \\ &= R(s,a) + \gamma \sum_{s' \in S} \mathbb{P}\left[s' \mid s,a\right] V^{*}(s') \end{aligned} \tag{3}$$

Since,

$$\begin{aligned} V^{*}(s) &= \max_{a} Q^{*}(s,a) \\ &= \max_{a} \left[R(s,a) + \gamma \sum_{s' \in S} \mathbb{P}\left[s' \mid s,a\right] V^{*}(s') \right] \end{aligned} \tag{4}$$

The above relation is very important for classical RL methods, and lay a foundation for designing objective functions modern RL.

Outline

Objective of an RL Agent

- Since $V^*(s)$ is the maximum expected total reward when starting from state s,

$$V^*(s) = \max_a Q^*(s,a) \quad \forall s \in S$$

- And since the policy attempts to maximize $V(s)$, the optimal policy π^* is

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Thus, the goal of a reinforcement learning agent is to learn a policy $\pi \approx \pi^*$.

Approaches to learning an Optimal Policy

We will go through the following algorithms for *Finite-state MDPs with finite action space*:

- Linear Programming formulation
- Value Iteration
- Policy Iteration
- Q-Learning

Linear Programming

- From (3) and (4), we can see that for finite-state MDPs, we can write $|S|$ linear equations for each state $s \in S$, with unknown $V^*(s)$.
- Thus, this linear programming formulation can easily be used to learn a optimal $Q^*(s, a)$ for all $(s, a) \in S \times A$ for the given MDP.

Value Iteration

Value iteration computes the optimal state value function by iteratively improving the estimate of $V(s)$.

Algorithm 1 Value Iteration

```
1: Initialize $V(s) = 0 \forall s \in S$
2: repeat
3:     for all $s \in S$ do
4:         for all $a \in A$ do
5:             $Q(s, a) \leftarrow \mathbb{E}[r \mid s, a] + \gamma \mathbb{E}_{s'}[V(s') \mid s, a]$
6:         end for
7:         $V(s) \leftarrow \max_a Q(s, a)$
8:     end for
9: until $V(s)$ converges
```

It can be shown that V converges to V^* in a finite number of iterations.

Value Iteration in Gridworld

noise $= 0.2$, $\gamma = 0.9$

3	0	0	0	$+1$
2	0	■	0	-1
1	0	■	0	0
0	0	0	0	0
	0	1	2	3

- Values after 1 iteration.

Value Iteration in Gridworld

noise $= 0.2$, $\gamma = 0.9$

3	0	0	0.72	$+1$
2	0	■	0	-1
1	0	■	0	0
0	0	0	0	0
	0	1	2	3

- Values after 2 iterations.

Value Iteration in Gridworld

noise $= 0.2$, $\gamma = 0.9$

3	0	0.52	0.78	$+1$
2	0	■	0.43	-1
1	0	■	0	0
0	0	0	0	0
	0	1	2	3

- Values after 3 iterations.

Value Iteration in Gridworld

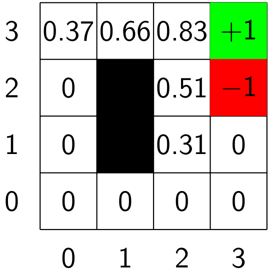
noise $= 0.2$, $\gamma = 0.9$

3	0.37	0.66	0.83	$+1$
2	0	⬛	0.51	-1
1	0	⬛	0.31	0
0	0	0	0	0
	0	1	2	3

- Values after 4 iterations.

Value Iteration in Gridworld

noise $= 0.2$, $\gamma = 0.9$



3	0.37	0.66	0.83	$+1$
2	0	■	0.51	-1
1	0	■	0.31	0
0	0	0	0	0
	0	1	2	3

- Values after iterations.
- Run for about 100 iterations to be sure of convergence.

Value Iteration: Effect of noise and γ

Noise ($\in [0, 0.5]$)[4]	$\gamma \in [0, 1]$	Effect
Low	High	Prefer to find the closest positive exit, while avoiding negative exits.
High	High	Prefer distant positive; Risk being close to negative.
Low	Low	Prefer distant positive; Avoid negative.
High	Low	Prefer close exit; Risk negative.

[4] Too much noise implies a bad model

Policy Iteration I

- In Value Iteration, we care about the convergence of the value function, but it is possibly for us to reach an optimal policy before the value function converges.
- In Policy Iteration, we update the policy at each iteration, rather than the value function.

Policy Iteration II

Algorithm 2 Policy Iteration

```
1: Initialize $\pi$ to arbitrary values
2: repeat
3:     $\pi' \leftarrow \pi$
4:     for all $s \in S$ do
5:         $V^{\pi'}(s) = \mathbb{E}[r \mid s, \pi'(s)] + \gamma \mathbb{E}_{s'}\left[V^{\pi'}(s') \,\middle|\, s, \pi'(s)\right]$
6:     end for    ▷ Using either linear programming or iterative algorithm
7:     $V = V^{\pi'}$
8:     $\pi(s) \leftarrow \arg\max_a \left(\mathbb{E}[r \mid s, a] + \gamma \mathbb{E}[V^{\pi}(s') \mid s, a]\right)$
9: until $\pi \approx \pi'$
```

Model-based vs. Model-free Learning

Model-based
: In model-based learning, the agent is assumed to have prior knowledge about the effects of its actions on the environment, that is, the transition probability function P of the MDP is known.

Model-free
: In model-free learning, the agent will not try to learn explicit models of the environment state transition and reward functions. However, it directly derives an optimal policy from the interactions with the environment.

Policy iteration and Value iteration are model-based methods as it is *necessary* to have knowledge of the probability of transitions in the MDP to compute the expected $V(s)$ at any given iteration of the algorithm.

Q-Learning (Watkins) I

Q-Learning is an example of model-free learning algorithm. It does not assume that agent knows anything about the state-transition and reward models. However, the agent will discover what are the good and bad actions by trial and error.

- The basic idea behind Q-learning is is to approximate the state-action pairs Q-function from the samples of $Q(s, a)$ that we observe during interaction with the environment.
- This approach is known as **Temporal-Difference (TD) Learning**.
- We combine this idea with **bootstrapping**, where TD learning methods update targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns as in Monte-Carlo methods.

Q-Learning (Watkins) II

- The update equation for the Q-function is:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a')\right) \quad (5)$$

where,

  - $\alpha \in [0,1]$ is the learning rate, such that if $\alpha \approx 0$ the Q value is updated very slowly and $\alpha \approx 1$ simply replaces the old Q with the new Q without any TD-learning taking place.
  - $\max_{a'} Q(s',a')$ is the estimate of the optimal state-value function.
- In Q-learning, we initialize the Q function as a table mapping all states $s \in S$ to the expected value for all actions $a \in A$.
- We then update this Q-table in an iterative (*episodic*) manner.

Outline

Limitations

- In the algorithms discussed up to this point, we have a general assumption that the environment we are acting upon has finite action and state spaces.
- This is (in general) a really bad assumption as most problems we see in the real world (especially in control theory) have either continuous action space or continuous state space or both.

Approaches: Continuous or Large State Space

- **Discretize** the state space
  - Here, we convert the continuous state space to a discrete grid or bins and use our favorite discrete state space RL agent.
  - But this leads to the **curse of dimensionality!** This refers to the explosion in state space with increase in the number of dimensions in the state space.
- Use **Function approximators**
  - We can define Q and value functions as functions that act upon a vector $\vec{s} \in S$.
  - Thus, the learning problem is that of optimizing the function using function approximation methods (convex optimization, etc.).

Approaches: Continuous Action Space I

To operate on continuous action spaces, we build on function approximators that map to a *action vector*. The following are a general class of algorithms used to learn a policy that operates in continuous action spaces:

- **Policy gradient** (Sutton et al.) methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to θ, $\pi_\theta(s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

Approaches: Continuous Action Space II

- **Actor-critic** (Konda and Tsitsiklis) methods contain two separate approximators defined as follows:
  - **Critic** updates the estimate of the value function (parametrized by w), and can either represent $Q_w(s, a)$ or $V_w(s, a)$
  - **Actor** outputs the direct action a and models the (stochastic) policy $\pi_\theta(s)$.

Note

Actor-critic methods typically use policy gradient methods to update the actor and TD-learning methods to update the critic.

Aside: On-policy and Off-policy Methods

On-policy	These methods use deterministic actions (or samples) from the target policy to train the algorithm. Examples of this are Monte-Carlo methods that compute V^π for the episode to update π.
Off-policy	They train on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy. Example of this is Q-Learning, where the TD error is computed against an existing (possibly older) $Q(s', a')$ to update the current $Q(s, a)$.

Exploration vs. Exploitation I

Example

Say there is a sheep grazing in a very patchy and large plot of land. The sheep can either approach a patch of grass that is somewhat small, and **exploit** this patch of grass until it is depleted, and then have no idea what to do. Otherwise, the sheep could **explore** the plot of land for a bit and then come back to the largest patch of grass it finds. Who knows, it could find some magical never-ending patch of grass! But too much exploration end up in the sheep going too far from any patch of grass and starve before it gets back to a known patch!

This is referred to as the **exploration vs. exploitation problem**.

- In the context of RL, we should train the learning agent to do both exploration and exploitation without being either too explorative or too greedy.

Exploration vs. Exploitation II

- This is typically done by performing $\epsilon-$greedy exploration, or sampling actions from a Boltzmann distribution.
- In $\epsilon-$greedy exploration, a decaying ϵ value is used as the probability of choosing a deterministic action (vs. a uniformly sampled action from A).
- In the Boltzmann distribution case, the action is sampled from a distribution

$$\mathbb{P}\left[a \mid s\right] = \frac{e^{Q(s,a)/k}}{\sum_{a' \in A} e^{Q(s,a')/k}},$$

where k is the *temperature* of the distribution. A large k implies uniform sampling of actions (explore) and a small k chooses the greedy strategy.

The Deadly Triad

- TD-learning + bootstrapping methods are a very efficient and flexible class of learning algorithms. But, when we combine Off-policy, bootstrapping methods with non-linear function approximation, the training could be (and is most likely) unstable and hard to converge.
- To tackle this, many architectures using deep learning models were proposed to resolve the problem, including DQN to stabilize the training with experience replay and occasionally frozen target network.

Outline

Deep Reinforcement Learning

- Deep RL = Deep Learning + RL.
- As mentioned in the previous section, deep RL architectures were developed to combat the so-called *deadly triad*, issue.
- Frameworks like AlphaZero (Silver et al.) and DQN (Mnih et al.) proposed the use of Convolutional Neural Networks to observe the state of games (Go and Atari games respectively) from raw pixel values and map them to discrete actions corresponding to either moves on a board or buttons on a video game controller.
- Both the above methods are off-policy methods that build off of TD-learning and bootstrapping methods like Q-learning.
- Many other works have also proposed the use of Deep RL in the context of continuous action and state spaces, like (Mnih et al. Schulman et al. Levine et al. Schulman et al.).

Deep Q Networks (Mnih et al.) I

- When using the DQN framework, we model the Q-function as a function approximator (deep neural network) $Q(s, a; \theta)$, where θ is the parameters for the NN.
- In the original Nature paper, the authors use a CNN that observes Atari games using their raw pixel values, and outputs the expected value of taking an action a at that state.

Deep Q Networks (Mnih et al.) II

- Moreover, to stabilize the problem of instability due to using TD-learning + bootstrapping with non-linear function approximators, the authors propose the following improvements to the vanilla Q-learning algorithm:
  - **Experience Replay:** All previous state transitions and their associated rewards $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in one replay memory $\mathcal{D} = \{e_1, \dots, e_t\}$. During update, a batch of experiences are sampled from this memory and used to compute the loss gradient for the NN.
  - **Periodically Update Targets:** The target for the loss is computed against a fixed (older) Q function, which is updated periodically. Essentially, The Q network is cloned and kept frozen as the optimization target every C steps (C is a hyperparameter). This modification makes the training more stable as it overcomes the short-term oscillations.

Deep Q Networks (Mnih et al.) III

- The loss function is computed as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta^{-}\right) - Q\left(s, a; \theta\right)\right)^2\right] \tag{6}$$

  where, $U(\mathcal{D})$ is a uniform distribution over the replay memory $\mathcal{D}$; θ^{-} is the parameters of the frozen target Q-network.

- Equation (6) is essentially computing the Mean-Squared Error (MSE) over a sampled batch of experiences, thus can be easily optimized (minimized) using Stochastic Gradient Descent (SGD).

Extensions to DQN

- Several extensions to DQN have been proposed, including (van Hasselt, Guez, and Silver Schaul et al. Z. Wang et al. Pritzel et al.).
- These build mainly on either the experience replay memory of the RL agent (with better, more intuitive sampling as opposed to uniform sampling) or they improve on the way the frozen target behaves in the update stage.

Policy Gradients I
(Sutton et al.)

- The general idea is that, instead of parameterizing the value function (as in the case of DQN) and doing policy improvement by greedily maximizing the expected value function, we parameterize the policy and do gradient decent along the direction that improves the policy.
- The intuition is that, sometimes, the policy is easier to approximate than the value function.
- The technique of policy gradients is a relatively old idea, but only recently has it been used along with non-linear function approximators (deep Neural Networks), like in (Mnih et al. Silver et al. Schulman et al. Schulman et al. Z. Wang et al. Lillicrap et al.) to name a few.

Policy Gradients II
(Sutton et al.)

The problem is generally formulated as so:

Let $\pi_\theta(a \mid s)$ be a stochastic policy that outputs the probability distribution on the action space for a given state s, where θ parameterizes the policy. Thus, the objective function for the optimization problem can be written as:

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a \mid s) Q^\pi(s, a), \qquad (7)$$

where $d^\pi(s)$ is the (on-policy)stationary distribution of the Markov chain induced under the policy π_θ.[5]

[5] For simplicity, we will omit θ under π when it is used in subscripts and superscripts (like in d^π, Q^π, V^π above).

Policy Gradient Theorem
(Sutton et al.)

Theorem (Policy Gradient Theorem)

The gradient of the objective function $\nabla_\theta J(\theta)$ can be simplified as follows:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s, a)\, \pi_\theta(a \mid s) \tag{8}$$

$$\propto \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s, a)\, \pi_\theta(a \mid s) d \tag{9}$$

This reduces the amount of computation required for the gradient significantly! But the approximation of d^π and $Q^\pi(s, a)$ still remains an issue. . .

Generalized Policy Gradients

(Schulman et al.)

Policy gradient methods maximize the objective function $J(\theta)$ by repeatedly estimating the gradient $g := \nabla_\theta J(\theta)$, which have the following general form:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \qquad (10)$$

where Ψ denotes the **target returns**, and may be one of:

1. $\sum_{t=0}^{\infty} r_t$: Total Reward;
2. $\sum_{t'=t}^{\infty} r_{t'}$: Reward following action a_t;
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of the previous form;
4. $Q^\pi(s_t, a_t)$;
5. $A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t)$: Advantage function;
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

Actor-Critic Models I
(Sutton and Barto)

- Two main components in PG are the policy model and the value function.
- It makes sense to learn the value function in addition to the policy, since knowing the value function can assist in the policy update.
  - This is mainly to reduce the gradient variance in vanilla policy gradients like REINFORCE (Williams).
- As mentioned earlier, an actor-critic model consists of two parts:
  - **Critic** updates the estimate of the value function (parametrized by w), and can either represent $Q_w(s, a)$ or $V_w(s, a)$
  - **Actor** outputs the direct action a and models the (stochastic) policy $\pi_\theta(a \mid s)$.

Actor-Critic Models II

(Sutton and Barto)

Algorithm 3 Simple Actor-Critic

```
1: Initialize $\theta, w$ arbitrarily.
2: Initialize $s$, sample $a \sim \pi_\theta(a \mid s)$
3: for $t = 1 \ldots T$ do
4:     $r_t = R(s, a), s' \sim \mathbb{P}[s' \mid s, a], a' \sim \pi_\theta(a' \mid s')$
5:     $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a \mid s)$    ▷ Update policy parameters
6:     $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$    ▷ Compute TD error
7:     $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$    ▷ Update critic parameters
8:     $a \leftarrow a'$ and $s \leftarrow s'$.
9: end for
```

- Here, two learning rates $(\alpha_\theta, \alpha_w)$ are predefined as training hyper-parameters.

Advantage Actor Critic (A2C) I
(Mnih et al.)

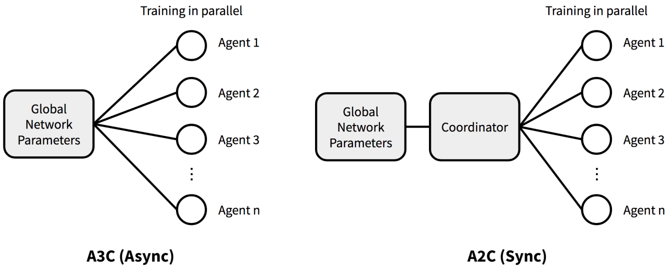


Figure: This algorithm is the same as Asynchronous Advantage Actor Critic (A3C) with the exception that A2C runs in a synchronous fashion. It is shown that A2C is generally better than A3C.

Advantage Actor Critic (A2C) II
(Mnih et al.)

- This algorithm basically leverages the power of modern distributed computation and has multiple actors running in parallel, and synchronize with a global critic every N steps.
- Each actor, a_i, runs for a specific number of steps, N, using its policy π_θ.
- In A2C, after *all actors complete N steps*, they synchronously provide the global coordinator with the total discounted rewards obtained from the individual runs. Then SGD is run on this batch of target returns (in this case the *Advantage Function*) to do the Actor-Critic updates as in Algorithm 3.
- In A3C, each actor talks to the global parameters independently, so it is possible sometimes the thread-specific actor would be playing with policies of different versions and therefore the aggregated update would not be optimal.

Other Extensions

- Deterministic Policy Gradient (DPG) (Silver et al.) extends the PG algorithm to *deterministic* policies (as opposed to stochastic).
- DDPG (Lillicrap et al.) extends the DPG algorithm with ideas from DQN, and thus extends DQN to the continuous domain.
- Trust Region Policy Optimization (Schulman et al. TRPO), Proximal Policy Optimization (Schulman et al. PPO) and ACKTR (Wu et al.) further improve on the PG algorithm by using so-called *natural gradients* that provide better approximation of the gradient direction.
- ACER (Z. Wang et al.) and Soft Actor-Critic (Haarnoja et al. SAC) both improve on the Actor-Critic algorithm by incorporating off-policy experience replay.
- SAC also incorporates the entropy measure of the stochastic policy into the returns to encourage exploration.

Outline

Applications of RL in CPS

CPS + Formal Methods Community

- Within the last few years, several works have been proposed that use ideas from Formal Methods in the context of Cyber-Physical Systems within RL frameworks.
- This includes incorporating:
  - Temporal Logics in RL algorithms, like (Aksaray et al. Li, Vasile, and Belta Hasanbeig, Abate, and Kroening)
  - Automata theory in RL (Li, Ma, and Belta);
  - Barrier-certificates for constrained training (Ohnishi et al. L. Wang, Theodorou, and Egerstedt)
  - Other such ideas like (Moarref and Kress-Gazit Alshiekh et al.)

Applications of RL in CPS

RL Community

- Inverse Reinforcement Learning:
  - Given policy π or behavior history sampled using a given policy, find a reward function for which the behavior is optimal.
  - The goal is to essentially derive a model that discriminates *good* behavior from *bad*, by looking at the demonstrations given by a "expert" (human or otherwise).
  - Some key works are:
    - "Algorithms for Inverse Reinforcement Learning"
    - "Apprenticeship Learning via Inverse Reinforcement Learning"
    - "Maximum Entropy Inverse Reinforcement Learning."
- Learning from Demonstrations/Apprenticeship Learning/Imitation Learning:
  - Goal is to learn a policy from demonstrations, using either direct mapping (deep learning) techniques or IRL.

Challenges: RL in Autonomous CPS

- Although many deep RL algorithms perform very well, they are very sensitive to hyperparameters and design of reward functions.
    - This introduces a lot of uncertainty in the training of the controller!
- Moreover, the RL algorithms we have discussed work on MDPs, that is, the environment is fully observable.
    - This is, in general, not true.
    - In CPS examples, there is uncertainty in states (sensor/actuation noise, state may be observable only by estimates, etc).
    - The approach to model this is by using Partially Observable MDPs (POMDPs).

POMDPs I

- POMDPs are incredibly powerful mathematical abstractions.
- They have been used in recent works for various tasks:
    - Navigating an office: (Åström).
    - Grasping with a robot arm: (Hsiao, Kaelbling, and Lozano-Pérez).
    - Wind Farm Management: (Memarzadeh Milad, Pozzi Matteo, and Zico Kolter J.).
    - Aircraft collision avoidance: (Mueller and Kochenderfer).

POMDPs II

Definition (POMDP)

It is a tuple $\langle S, A, P, R, \Omega, O \rangle$ where:

- S, A, P, R are the same as in and MDP;
- Ω is a set of observations;
- $O : S \times A \times \Omega \to [0, 1]$ is the observation function, which is a probability distribution such that $O(s, a, o) = \mathbb{P}\left[o \mid s, a\right], \forall o \in \Omega$ (the probability of observing o if action a is taken in state s).

POMDPs model the information available to the agent by specifying a function from the hidden state to the observables.

Planning for POMDPs

- Unfortunately, the observations are not Markov (because two different states might look the same), which invalidates all of the MDP solution techniques.
- The optimal solution to this problem is to construct a belief state MDP, where a belief state is a probability distribution over states.
- Control theory is concerned with solving POMDPs, but in practice, control theorists make strong assumptions about the nature of the model (typically linear-Gaussian) and reward function (typically negative quadratic loss) in order to be able to make theoretical guarantees of optimality, etc.
- By contrast, optimally solving a generic discrete POMDP is wildly intractable. Finding tractable special cases (e.g., structured models) is a hot research topic.

Approaches to solve POMDPs

- Research has been done since the **1960's!**
- POMDPs are MDPs if they are properly updated over **belief states** (Åström).
- Belief states can be computed with the following general idea:
  - Start in some initial belied b prior to any observations
  - Compute new belief state b' based on current belief state b, action a, and observation o.
- This is a very common approach used in popular algorithms in robotics, like:
  - Bayes Filters, Kalman Filters, and Particle Filters.

RL for POMDPs

Adapted Policies (Singh, Jaakkola, and Jordan)

Definition (Adapted Policy)

is a mapping $\pi : \Omega \times A \to [0, 1]$. That is, π is a stochastic policy that operates on the *observation space* Ω as opposed to the state space S.

Moreover, the value function of the POMDP is associated with a *distribution on states* as opposed to a single state.
This problem is then solved for relatively small, discrete, and low-dimensional state-spaces using a modified Q-learning algorithm.

RL for POMDPs

Point-based Value Iteration (Porta, Spaan, and Vlassis)

In this approach,

- A small set of reachable *belief points* is selected.
- A Bellman equation is defined for those points, keeping value and gradient.
- This approach is shown to be generalized to *continuous state spaces*, but not to continuous action and observation spaces.

RL for POMDPs

Deep RL Methods

1. Use tracking techniques for Deep Learning data:
   - Deep Variational Bayes Filter (Karl et al.) uses variational inference during batch updates to learn temporal and spatial dependencies.
2. Throw an Recurrent NN (LSTM/GRU) / Forget the math!:
   - DRQN (Hausknecht and Stone), RDPG (Heess et al.).
   - Recurrent Predictive State Policy (Hefny et al.).
3. Use human-like differentiable memory:
   - Neural Map (Parisotto and Salakhutdinov), MERLIN (Wayne et al.).

Future Work and Open Problems I

- Verifiable Robotics and RL:
  - There are several challenges in modern RL and AI, some of which are summarized in (Amodei et al. Leike et al.).
  - Most Deep RL systems are very sensitive to hyperparameter tuning, this a lot of empirical results are hard to reproduce in environments that differ from the environments used in a paper.
  - **Interpretability** of policies learned by Deep RL algorithms is another key issue, along with the interpretability of Deep Learning systems in general.

Future Work and Open Problems II

- Safety in CPS + Deep Learning systems:
  - This is a relatively new topic, where there have been attempts to use Formal Methods to prove the safety of systems that use deep learning controllers.
    - This is not an easy task even in standard cyber-physical systems and hybrid systems.
    - The addition of a non-linear, “black-box” component just compounds this problem.
  - There have also been attempts to incorporate ideas like Temporal Logic, Barrier Certificates, and Shields during the training of RL systems.
  - See: (Tian et al. Pei et al. S. Wang et al. Tuncali et al.)

Outline

References I

Abbeel, Pieter and Andrew Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: ACM, 2004. 1–. Web. ICML '04.

Aksaray, D., et al. "Q-Learning for Robust Satisfaction of Signal Temporal Logic Specifications". *2016 IEEE 55th Conference on Decision and Control (CDC)*. 2016. 6565–6570. Print.

Alshiekh, Mohammed, et al. "Safe Reinforcement Learning via Shielding". (2017). Print.

Amodei, Dario, et al. "Concrete Problems in AI Safety". (21 June 2016). arXiv: 1606.06565 [cs]. Web.

Åström, K. J. "Optimal Control of Markov Processes with Incomplete State Information". *Journal of Mathematical Analysis and Applications* 10.1 (1965): 174–205. Print.

Barto, A. G., R. S. Sutton, and C. W. Anderson. "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems". *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (Sept. 1983): 834–846. Print.

Bertsekas, Dimitri P. and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Belmont, Mass: Athena Scientific, 1996. Print. Optimization and Neural Computation Series.

Haarnoja, Tuomas, et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". (4 Jan. 2018). arXiv: 1801.01290 [cs, stat]. Web.

References II

Hasanbeig, Mohammadhosein, Alessandro Abate, and Daniel Kroening. "Logically-Constrained Reinforcement Learning". *arXiv:1801.08099 [cs]* (2018). arXiv: 1801.08099 [cs].

Hausknecht, Matthew and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". (23 July 2015). arXiv: 1507.06527 [cs]. Web.

Heess, Nicolas, et al. "Memory-Based Control with Recurrent Neural Networks". (14 Dec. 2015). Web.

Hefny, Ahmed, et al. "Recurrent Predictive State Policy Networks". (5 Mar. 2018). Web.

Hsiao, Kaijen, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Robust Grasping under Object Pose Uncertainty". *Autonomous Robots* 31.2 (2011): 253. Print.

Karl, Maximilian, et al. "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data". (20 May 2016). Web.

Konda, Vijay R. and John N. Tsitsiklis. "Actor-Critic Algorithms". *Advances in Neural Information Processing Systems 12*. Edited by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000. 1008–1014. Print.

Kumar, P. R. and P. P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Philadelphia: Society for Industrial and Applied Mathematics, 2016. Print. Classics in Applied Mathematics 75.

Leike, Jan, et al. "AI Safety Gridworlds". (27 Nov. 2017). arXiv: 1711.09883 [cs]. Web.

References III

Levine, Sergey, et al. “End-to-End Training of Deep Visuomotor Policies”. (2 Apr. 2015). arXiv: 1504.00702 [cs]. Web.

Li, X., C. Vasile, and C. Belta. “Reinforcement Learning with Temporal Logic Rewards”. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. 3834–3839. Print.

Li, Xiao, Yao Ma, and Calin Belta. “Automata-Guided Hierarchical Reinforcement Learning for Skill Composition”. *arXiv:1711.00129 [cs]* (2017). arXiv: 1711.00129 [cs].

Lillicrap, Timothy P., et al. “Continuous Control with Deep Reinforcement Learning”. (9 Sept. 2015). arXiv: 1509.02971 [cs, stat]. Web.

Memarzadeh Milad, Pozzi Matteo, and Zico Kolter J. “Optimal Planning and Learning in Uncertain Environments for the Management of Wind Farms”. *Journal of Computing in Civil Engineering* 29.5 (2015): 04014076. Print.

Mnih, Volodymyr, et al. “Asynchronous Methods for Deep Reinforcement Learning”. (4 Feb. 2016). arXiv: 1602.01783 [cs]. Web.

Mnih, Volodymyr, et al. “Human-Level Control through Deep Reinforcement Learning”. *Nature* 518.7540 (Feb. 2015): 529–533. Web.

Moarref, S. and H. Kress-Gazit. “Decentralized Control of Robotic Swarms from High-Level Temporal Logic Specifications”. *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. 2017. 17–23. Print.

References IV

Mueller, Eric R. and Mykel Kochenderfer. "Multi-Rotor Aircraft Collision Avoidance Using Partially Observable Markov Decision Processes". *AIAA Modeling and Simulation Technologies Conference*. American Institute of Aeronautics and Astronautics, 2016. Print. AIAA AVIATION Forum.

Ng, Andrew Y. and Stuart Russell. "Algorithms for Inverse Reinforcement Learning". *In Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, 2000. 663–670. Print.

Ohnishi, Motoya, et al. "Barrier-Certified Adaptive Reinforcement Learning with Applications to Brushbot Navigation". *arXiv:1801.09627 [cs]* (2018). arXiv: 1801.09627 [cs].

Parisotto, Emilio and Ruslan Salakhutdinov. "Neural Map: Structured Memory for Deep Reinforcement Learning". (27 Feb. 2017). Web.

Pei, Kexin, et al. "DeepXplore: Automated Whitebox Testing of Deep Learning Systems". *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17* (2017): 1–18. arXiv: 1705.06640.

Porta, Josep M., Matthijs T. J. Spaan, and Nikos A. Vlassis. "Robot Planning in Partially Observable Continuous Domains". *Robotics: Science and Systems*. 2005. Print.

Pritzel, Alexander, et al. "Neural Episodic Control". (6 Mar. 2017). arXiv: 1703.01988 [cs, stat]. Web.

Schaul, Tom, et al. "Prioritized Experience Replay". (18 Nov. 2015). arXiv: 1511.05952 [cs]. Web.

References V

Schulman, John, et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. (8 June 2015). arXiv: 1506.02438 [cs]. Web.

Schulman, John, et al. “Proximal Policy Optimization Algorithms”. (19 July 2017). arXiv: 1707.06347 [cs]. Web.

Schulman, John, et al. “Trust Region Policy Optimization”. (19 Feb. 2015). arXiv: 1502.05477 [cs]. Web.

Silver, David, et al. “Deterministic Policy Gradient Algorithms”. *ICML*. 2014. Print.

Silver, David, et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. *Nature* 529.7587 (Jan. 2016): 484–489. Web.

Singh, Satinder P., Tommi Jaakkola, and Michael I. Jordan. “Learning Without State-Estimation in Partially Observable Markovian Decision Processes”. *Machine Learning Proceedings 1994*. Edited by William W. Cohen and Haym Hirsh. San Francisco (CA): Morgan Kaufmann, 1994. 284–292. Print.

Sutton, Richard S. and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Cambridge, MA: The MIT Press, 2018. Print. Adaptive Computation and Machine Learning Series.

References VI

Sutton, Richard S., et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. *Advances in Neural Information Processing Systems*. 2000. 1057–1063. Print.

Tian, Yuchi, et al. “DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars”. *arXiv:1708.08559 [cs]* (2017). arXiv: 1708.08559 [cs].

Tuncali, Cumhur Erkan, et al. “Reasoning about Safety of Learning-Enabled Components in Autonomous Cyber-Physical Systems”. *arXiv:1804.03973 [cs]* (2018). arXiv: 1804.03973 [cs].

Van Hasselt, Hado, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. (22 Sept. 2015). arXiv: 1509.06461 [cs]. Web.

Wang, L., E. A. Theodorou, and M. Egerstedt. “Safe Learning of Quadrotor Dynamics Using Barrier Certificates”. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018. 2460–2465. Print.

Wang, Shiqi, et al. “Efficient Formal Safety Analysis of Neural Networks”. *arXiv:1809.08098 [cs, stat]* (2018). arXiv: 1809.08098 [cs, stat].

Wang, Ziyu, et al. “Dueling Network Architectures for Deep Reinforcement Learning”. (20 Nov. 2015). arXiv: 1511.06581 [cs]. Web.

Wang, Ziyu, et al. “Sample Efficient Actor-Critic with Experience Replay”. (3 Nov. 2016). arXiv: 1611.01224 [cs]. Web.

References VII

Watkins, Christopher John Cornish Hellaby. “Learning from Delayed Rewards”. Diss. King’s College, Cambridge, 1989. Print.

Wayne, Greg, et al. “Unsupervised Predictive Memory in a Goal-Directed Agent”. (28 Mar. 2018). Web.

Williams, Ronald J. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. *Machine Learning* 8.3 (May 1992): 229–256. Print.

Wu, Yuhuai, et al. “Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation”. (17 Aug. 2017). arXiv: 1708.05144 [cs]. Web.

Ziebart, Brian D., et al. “Maximum Entropy Inverse Reinforcement Learning.” *Aaai*. Chicago, IL, USA, 2008. 1433–1438. Print.