Actor-Critic Methods for Control

Presenter: Jake Grigsby

University of Virginia
https://qdata.github.io/deep2Read/

202008

Control Tasks in RL

Control Tasks in RL:

- Involve fast, accurate interpretation of sensory data to take actions that have an immediate effect
  - Simple credit assignment
- Do not involve significant amounts of exploration, long-term memory or planning
  - Partially observed tasks can be solved with a window of recent frames
- Have dense reward signals
  - Meaningful variance in returns near the randomly initialized policy

Control Tasks in RL

- Classic RL Problems

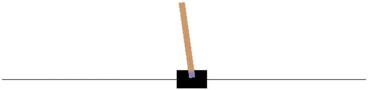


Figure: CartPole task involves balancing a pole on a moving platform

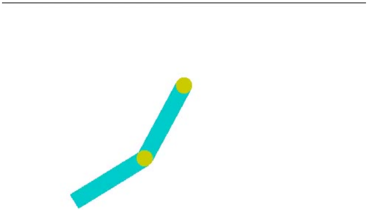
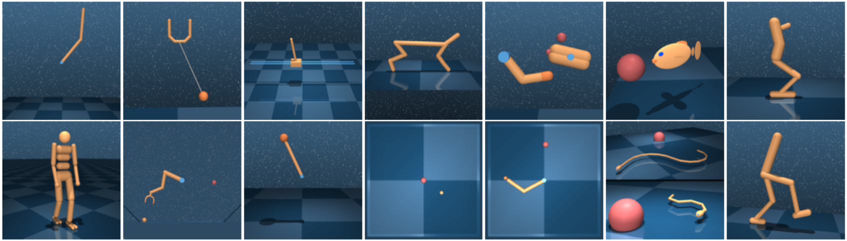


Figure: The Acrobat task involves swinging a hinged arm

Control Tasks in RL



- Continuous Control
  - Tasks with continuous (and usually multi-dimensional) action spaces

Control Tasks in RL

Continuous Control MDPs:

- States:
  - Representations of the essential dynamics (joint position, velocity, angle...). Low-dimensional, "State Based", Fully Observed
  - 3rd person camera views of the rendered scene. High-dimensional, "Pixel Based", Partially Observed
- Actions:
  - Real-valued scalars that apply to specific components of the model's control interface, such as the torque on each joint/motor
- Rewards:
  - Typically based on some measure of efficient motion, such as velocity or distance from the starting point

Control Tasks in RL

Locomotion: control limbs and joints to move a 3D model efficiently in a simulated physics engine.

Hopper (14, 4, 15): The planar one-legged hopper introduced in (Lillicrap et al., 2015), initialised in a random configuration. In the `stand` task it is rewarded for bringing its torso to a minimal height. In the `hop` task it is rewarded for torso height and forward velocity.

Cheetah (18, 6, 17): A running planar biped based on (Wawrzyński, 2009). The reward r is linearly proportional to the forward velocity v up to a maximum of 10m/s i.e. $r(v) = \max(0, \min(v/10, 1))$.

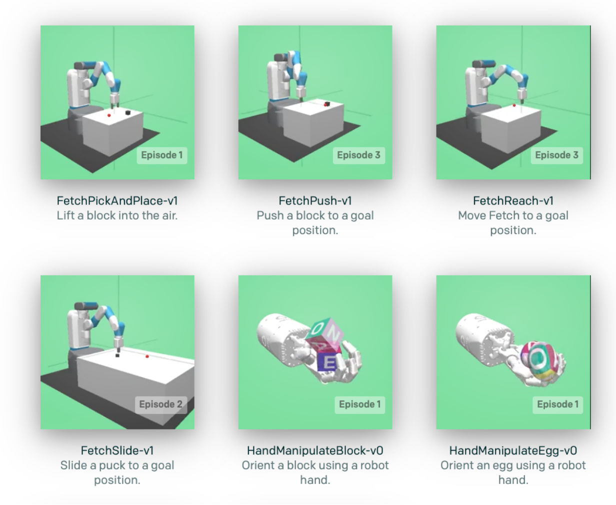
Control Tasks in RL

Walker (18, 6, 24): An improved planar walker based on the one introduced in (Lillicrap et al., 2015). In the `stand` task reward is a combination of terms encouraging an upright torso and some minimal torso height. The `walk` and `run` tasks include a component encouraging forward velocity.

Humanoid (54, 21, 67): A simplified humanoid with 21 joints, based on the model in (Tassa et al., 2012). Three tasks: `stand`, `walk` and `run` are differentiated by the desired horizontal speed of 0, 1 and 10m/s, respectively. Observations are in an egocentric frame and many movement styles are possible solutions e.g. running backwards or sideways. This facilitates exploration of local optima.

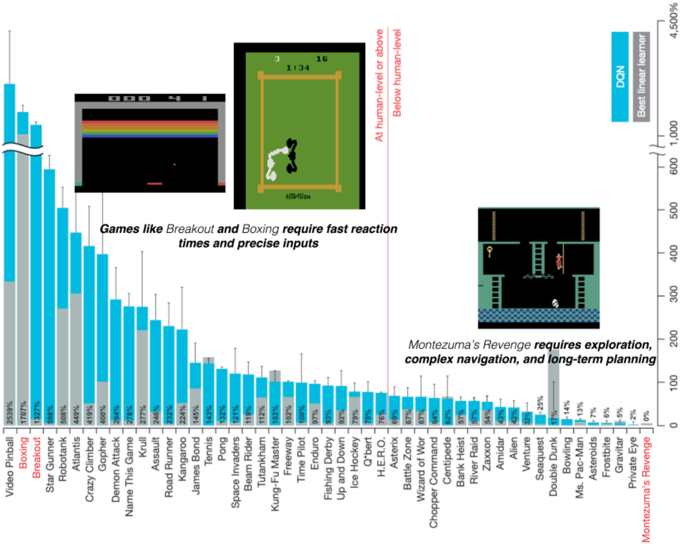
Control Tasks in RL

Manipulation: continuous control tasks meant to emulate fine motor control on modern robotics systems.



Control Tasks in RL

- The Atari 2600 Benchmark (ALE/Atari 57)



Control Tasks in RL

Most of the problems (Deep) RL algorithms solve can be loosely classified as Control Tasks.

Some notable exceptions:

- Two player board games
  - AlphaGo and AlphaZero augment Deep RL with planning (MCTS)
- Dota 2 (OpenAI Five), Starcraft II (AlphaStar)
  - Model-free agents do start to show long-term planning... with recurrent architectures and thousands of years of gameplay
  - But these games have a ton of control-like subproblems (combat, item selection...). Professional human players make hundreds of split-second decisions per minute.

Markov Decision Process

Definition

A Markov Decision Process **(MDP)** consists of:

- $\mathcal{S}$, a set of states
- $\mathcal{A}$, a set of actions
- $\mathcal{R} \subseteq \mathbb{R}$, a set of rewards
- a dynamics function $p : \mathcal{S}x\mathcal{R}x\mathcal{S}x\mathcal{A} \rightarrow [0, 1]$

$$p(s', r|s, a) = Pr\{\mathcal{S}_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

- an initial state distribution, ρ_0

It's common to break the dynamics function p up into a **Transition Function** $T(s, a, s') = \sum_{r \in \mathcal{R}} p(s', r|s, a)$, and a **Reward Function**

$$R(s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

The RL Problem

The goal of RL agents is to find a **policy**[1] $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the *expected discounted return*

$$\pi^* = \underset{\pi}{argmax} \underset{\tau \sim \pi}{\mathbb{E}} \left[\sum_{t=0}^{t=\infty} \gamma^t R_t \right]$$

where $\gamma \in [0, 1)$ is the *discount factor* that lets us deal with non-episodic tasks and τ is a *trajectory* (a sequence of states and actions that describe the agent's experience)

[1] Policies can also be stochastic, in which case they're written $\pi(a|s) : \mathcal{S} x \mathcal{A} \rightarrow [0, 1]$

The Challenges of Continuous Control

Value methods (SARSA, Q-Learning, ...) build their policy functions by maxing over the state space:

$$\pi(s) = \underset{a}{argmax}\, Q_\theta(s, a)$$

When actions are continuous and high-dimensional, we skip the *max* operation by directly parameterizing the policy (π_θ).

Outputs are either the deterministic action choice, or the mean and std of a distribution to sample from:

$$\pi_\theta(s) = a$$
$$\pi(s) \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$$

[1] In practice, the std parameters are learned, but often state-independent.

Actor-Critic Algorithms

Actor-Critic algorithms bring Policy Iteration to continuous action spaces

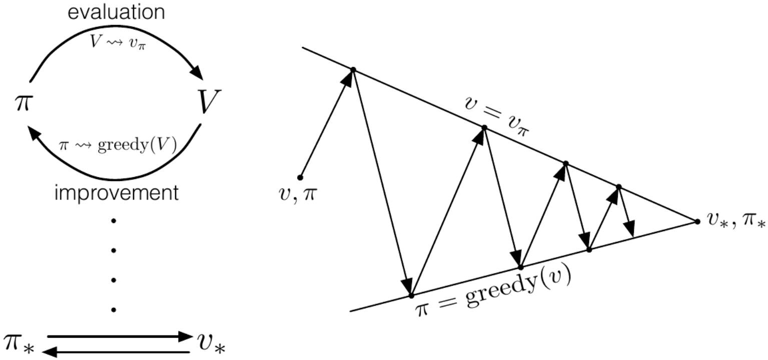


Figure: Policy Iteration iteratively evaluates and improves and a policy until convergence

Actor-Critic Algorithms

General Actor-Critic approach:

Initialize an actor network μ_{θ_0}, and a critic network q_{ϕ_0}
For $k = 1, 2, 3, \ldots$

1. Sample trajectories $(s_t, a_t, r_t, s_{t+1}, d_{t+1})$ from the environment using the current μ_{θ_k}
2. Use sampled trajectories to construct Temporal Difference targets (y) and optimize q_ϕ

$$q_{\phi_{k+1}} \leftarrow q_{\phi k} - \alpha \nabla \frac{1}{N} \sum_{i=0}^{N} (y^{(i)} - q_{\phi_k}(s^{(i)}, a^{(i)}))^2$$

3. Replay the experiences and move actions in the direction recommended by the critic

$$\mu_{\theta_{k+1}} \leftarrow \mu_{\theta_k} + \alpha \nabla \frac{1}{N} \sum_{i=0}^{N} q_{\phi_{k+1}}(s^{(i)}, \mu_{\theta_k}(s^{(i)}))$$

Deep Deterministic Policy Gradient

- Use a replay buffer to store experience between updates, increasing sample efficiency
  - Off-policy learning can hurt stability
- Use target networks to stabilize loss functions where the same parameters appear twice
- Deterministic policy - use action-space noise during experience collection to increase exploration.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for however many updates **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

    end for

  end if

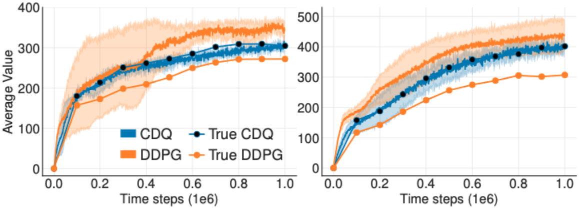
until convergence

Overestimation Bias

- By updating the actor based on the critic's estimations, we risk learning to exploit (s, a) pairs whose value the critic *overestimates*

$$\mathbb{E}[q_\phi(s, \pi_\theta(s))] \geq \mathbb{E}[q^\phi(s, \pi_\theta(s))]$$

  - See Section 4.1 of [3]



(a) Hopper-v1 (b) Walker2d-v1

Figure 1. Measuring overestimation bias in the value estimates of DDPG and our proposed method, Clipped Double Q-learning (CDQ), on MuJoCo environments over 1 million time steps.

Twin Delayed Deep Deterministic Policy Gradient (TD3)

- Clipped Double Q Learning to address overestimation

$$y = r_t + \gamma \min_{i=1,2} q_{\phi_i}(s_{t+1}, \pi_{\theta_{targ}}(s_{t+1}))$$

- Delayed policy updates
  - The learning speed of each network is a difficult thing to tune (similar to GANs[2])
- Target Smoothing
  - Add noise to the target network's actions when computing y. Smooths the function's surface by fitting the value of a small neighborhood around the target.

$$y = r_t + \gamma \min_{i=1,2} q_{\phi_i}(s_{t+1}, \pi_{\theta_{targ}}(s_{t+1})) + \epsilon$$

$$\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

[1] There are actually lots of similarities between AC and GANs, see [2]

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i && \text{for } i = 1, 2 \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta \end{aligned}$$

      end if
    end for
  end if

Soft Actor Critic (SAC)

- Stochastic policy
  - No need for extra exploration noise - just sample from the action distribution. We introduce an entropy regularization term to encourage diversity.
- Changes to target updates
  - Use active actor network for TD targets

[2] [4]

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

repeat

  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for j in range(however many updates) **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

      Update target networks with

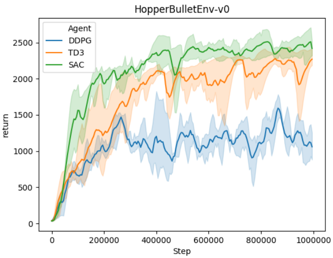
$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

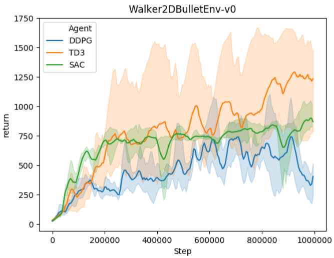
    end for

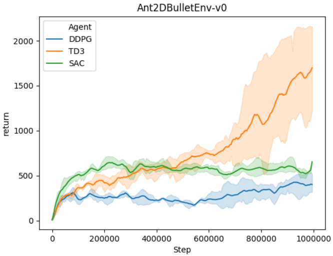
  end if

until convergence

Locomotion Results







[2] Link to implementations

References I

- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: 1509.02971 [cs.LG].
- David Pfau and Oriol Vinyals. "Connecting Generative Adversarial Networks and Actor-Critic Methods". In: *CoRR* abs/1610.01945 (2016). arXiv: 1610.01945. URL: http://arxiv.org/abs/1610.01945.
- Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *arXiv preprint arXiv:1802.09477* (2018).
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.