More Advanced RL Algorithms

Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap,* Jonat
Tom Erez, Yuval Tassa, Dav
Google Deepmind
London, UK
{countzero, jjhunt,
etom, tassa, david

Addressing Function Approximation Error in Actor-Critic Methods

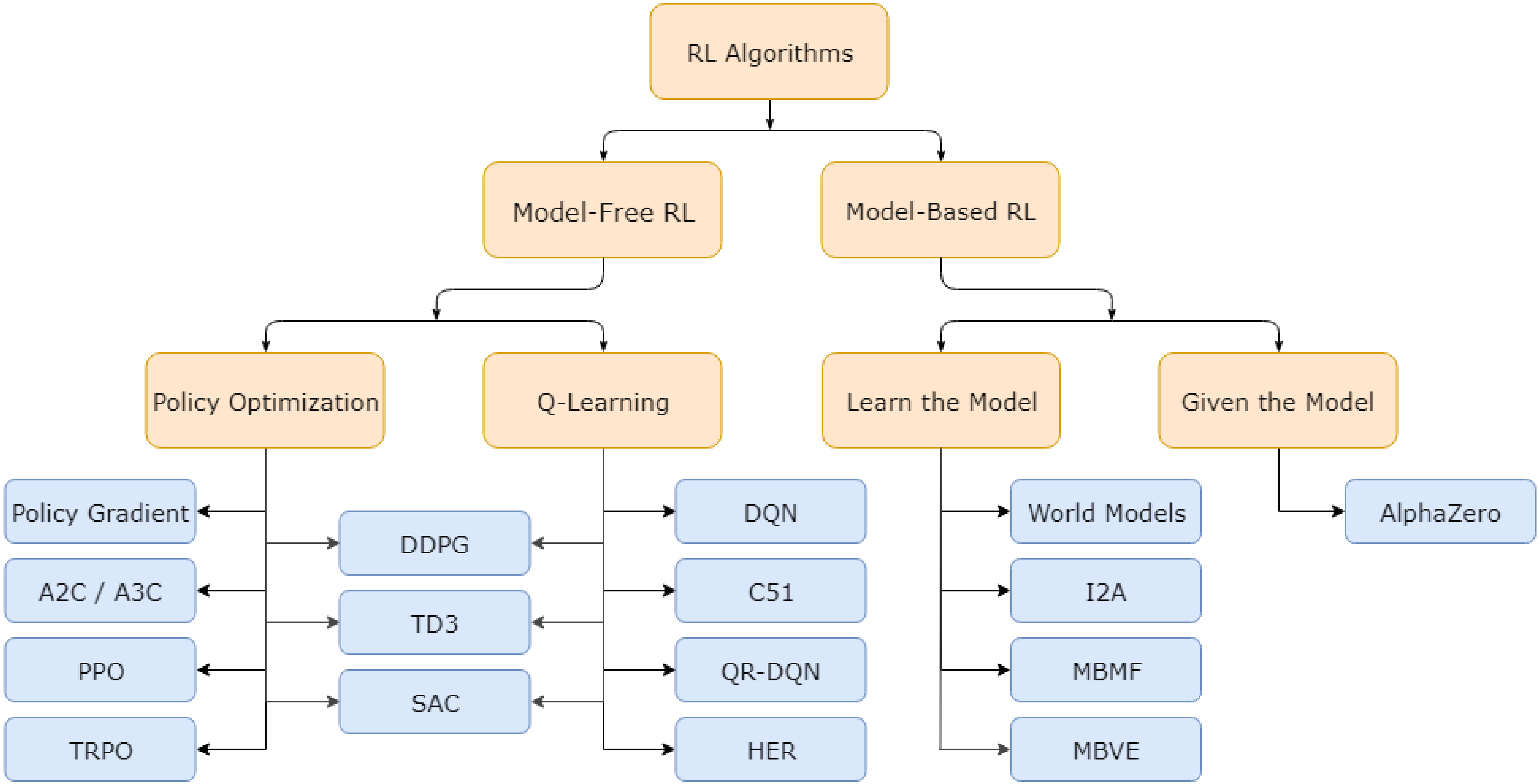
Scott Fujimoto [1] **Herke van H**

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja [1] **Aurick Zhou** [1] **Pieter Abbeel** [1] **Sergey Levine** [1]

Instructor: Daniel Brown --- University of Utah

Rough Taxonomy of RL Algorithms



Deep Deterministic Policy Gradients (DDPG)

Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap,* Jonathan J. Hunt,* Alexander Pritzel, Nicolas Heess,
Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK
{countzero, jjhunt, apritzel, heess,
etom, tassa, davidsilver, wierstra} @ google.com

DDPG Core Ideas

- Learn both a Q-Function and a Policy
- Uses off-policy data to learn a Q-function via the Bellman equations
- Related to Q-Learning but only works with continuous action spaces.
- Given Q* we ha

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

Key idea is to alternate learning a model of Q^* and learning a model of $a^*(s)$

How to deal with continuous actions?

- Solving $a^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$ is trivial if there are finite actions, but in continuous spaces this is a non-trivial and complex optimization problem that would have to be repeatedly solved perhaps millions of times!

Learning a Q-function

- Our old friend, the Bellman equation

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

- To train a neural net to approximate Q* we usually use an MSE loss based on the Bellman equation

$$L(\phi, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d) \sim \mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

Where d = 1 if "done" (terminal state reached) and d = 0 otherwise.

To stabilize training Q-functions

- We do the same things as done in DQN…
  - Replay buffer to store experience (s,a,r,s’,d)
    - The optimal Q-function should satisfy the Bellman equation for any transition so we can train on any data (DDPG is an Off-Policy RL algorithm).
  - Use a target network to stabilize MSE loss
    - DDPG uses Polyak averaging like the DQN tutorial for HW4

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

How is this different than DQN?

- We also learn a target policy network to approximate the argmax

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right]$$

But how??

- Policy learning in Deep **Deterministic** Policy Gradients (DDPG)?

- Just use gradient ascent (freezing Q-function weights)

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_{\phi}(s, \mu_{\theta}(s)) \right]$$

- To improve exploration, it is typical to add Gaussian noise to the actions during training.

DDPG Overview

- Environment interaction during training:
  - Take actions according to $a \sim \mu_{\theta_{target}}(s) + noise$
  - Store (s,a,s',r,d) in Buffer
- In parallel or periodically train Q-function and policy

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

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

Twin Delayed DDPG (TD3)

Addressing Function Approximation Error in Actor-Critic Methods

Scott Fujimoto [1] **Herke van Hoof** [2] **David Meger** [1]

- TD3 is an off-policy algorithm.
- TD3 only works with continuous action spaces.

Motivation

- What might go wrong in DDPG?

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_{\phi}(s, \mu_{\theta}(s)) \right]$$

- The policy is incentivized to exploit any errors in the Q-function!
  - Leads to bad policy if Q-function ever overestimates Q-values.

Twin Delayed DDPG (TD3) Tricks

- **Target Policy Smoothing**
  - Adds noise to the target action, to make it harder for the policy to exploit Q-function errors.
- **Clipped Double-Q Learning**
  - Learn two Q-functions instead of one (“twin”)
  - Conservatively choose the smaller of the two Q-values when computing the Bellman error
- **Delayed Policy Updates**
  - Update the policy less frequently than the Q-function.
  - Recommends one policy update for every two Q-function updates.

More details

- Adding noise to target actions
  - $a(s) = \mu_{\theta_{target}}(s) + noise$
  - Also usually clipped within any continuous action limits to prevent impossible actions (same with DDPG)
- Double Q-Learning
  - Compute pessimistic target $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s'))$
  - Update both Q-functions via Bellman MSE loss

$$L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi_1}(s,a) - y(r,s',d)\right)^2\right] \qquad L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi_2}(s,a) - y(r,s',d)\right)^2\right]$$

Policy Learning

- Basically the same as DDPG

$$\max_{\theta} \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_{\phi_1}(s, \mu_\theta(s))\right]$$

- But policy updates, and target policy updates, are less frequent than Q-function updates for improved stability.
- Exploration still done by adding noise to rollouts.
  - Another common trick is to start with uniform random policy to collect a bunch of diverse data in the replay buffer

Soft Actor Critic

Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja[1] **Aurick Zhou**[1] **Pieter Abbeel**[1] **Sergey Levine**[1]

- Optimizes a **Stochastic** policy in an **Off-Policy** way.
- Makes use of **entropy regularization** to help with exploration and stability.
- There are both continuous and discrete action versions.

Entropy Regularized RL

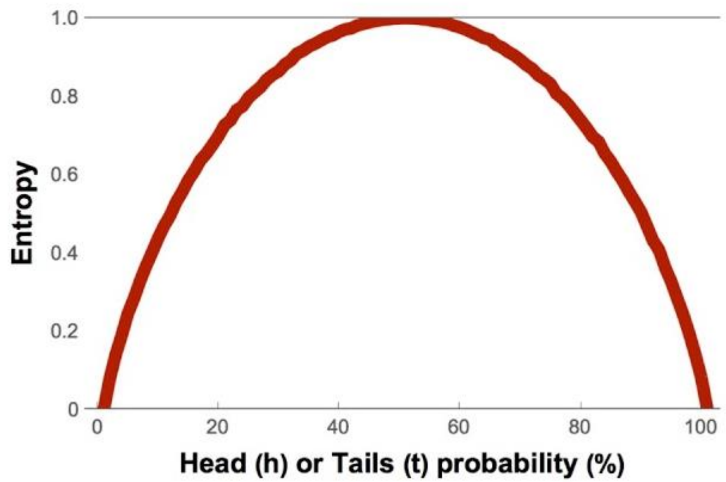
- Entropy strikes again!

$$\mathrm{H}(X) := -\sum_{x\in\mathcal{X}} p(x)\log p(x)$$

$$H(\pi) = -\sum_a \pi(a|s)\log\pi(a|s)$$

$$H(\pi) = -\int \pi(a|s)\log\pi(a|s)da$$

$$P(X = heads) = \frac{1}{2} \qquad P(X = tails) = \frac{1}{2}$$



Entropy Regularized RL

- Key idea: Give the policy a bonus for having high entropy.

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \right]$$

- The parameter α gives some control over exploration vs. exploitation

Entropy Regularized RL

- We now can define entropy regularized value functions

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty} \gamma^t \Bigg(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Bigg) \middle| s_0 = s\right]$$

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \middle| s_0 = s, a_0 = a\right]$$

where

$$V^{\pi}(s) = \underset{a \sim \pi}{\mathrm{E}}\left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right)$$

Entropy Regularized Bellman Equation

- We now have a new Bellman Equation

$$Q^{\pi}(s,a) = \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right) \right) \right] = \underset{s' \sim P}{\mathrm{E}} \left[R(s,a,s') + \gamma V^{\pi}(s') \right].$$

and with some rewriting we have

$$= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') - \alpha \log \pi(a'|s') \right) \right]$$

Because $\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x)$

Entropy Regularized Bellman Equation

- We now have a new Bellman Equation

$$Q^\pi(s,a) = \underset{\substack{s'\sim P \\ a'\sim\pi}}{\mathrm{E}}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') - \alpha\log\pi(a'|s')\right)\right]$$

- What do we do with expectations in RL?
  - Approximate them with samples!! (s,a,r,s')

$$Q^\pi(s,a) \approx r + \gamma\left(Q^\pi(s',\tilde{a}') - \alpha\log\pi(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi(\cdot|s')$$

Sampled from current policy (not from replay buffer)

Soft Actor Critic High-Level

- Learns a policy and two Q-functions
  - Takes minimum over Q-functions like TD3 but with extra entropy term.
- Optimizes a policy to maximize Q-function
  - Similar to TD3, but with additional bonus for policy entropy

Applications

- https://sites.google.com/view/sac-and-applications