DDPG and Practice in RL

RL Discuss Group - Week 2

Zhengfei Wang

October 18, 2018

Outline

Pre-Knowledge

- Actor-Critic: **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$ to compute action. **Critic** updates the value function parameters w to evaluate the action-value or state-value.
- DPG: Previous policy function $\pi(.|s)$ is modeled as a probability distribution, therefore the action is stochastic. **Deterministic Policy Gradient** models the policy function $a = \mu(s)$ and prove the gradient of deterministic policy.
- DQN: **Replay buffer** minimize correlations between samples. **Target update** makes the algorithm more stable. Limitation: continuous and high dimensional action spaces.

Overview

A model-free, off-policy, actor-critic algorithm, deep function approximators for high-dimensional, continuous action spaces.

Continuous control with deep reinforcement learning. ICLR 2016. DeepMind.

Core Ideas

- Experience replay store transitions (s_t, a_t, r_t, s_{t+1})
- Target network update follows $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
- Add noise $\mathcal{N}$ for exploration $\mu'(s) = \mu_\theta(s) + \mathcal{N}$

Algorithm

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

PyTorch DDPG

https://github.com/ailab-pku/rl-framework/tree/master/DDPG

Some Tips

- Parameter initialization is **IMPORTANT**
- Network architecture is always small (400-300)
- Noise level **SHOULD** decay during training
- RunningMeanStd normalize observation and reward
- Another exploration is ϵ-greedy style
- About **Batch Normalization**: I'm NOT sure currently...

Advances for DDPG

- Prioritized Experience Replay
- Parameter Noise
- Distributed Version

Priority Experience Relay

Prioritized Experience Replay. ICLR 2016. DeepMind.

Intuition: replay important transitions more frequently to learn more efficiently.
Criteria:

- based on TD-error δ, for how 'surprising' or unexpected the transition is
- stochastic priority: $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$
  - proportional, $p_i = |\delta_i| + \epsilon$
  - rank-based, $p_i = \frac{1}{rank(i)}$
- importance sampling to anneal the bias

PER Experiment

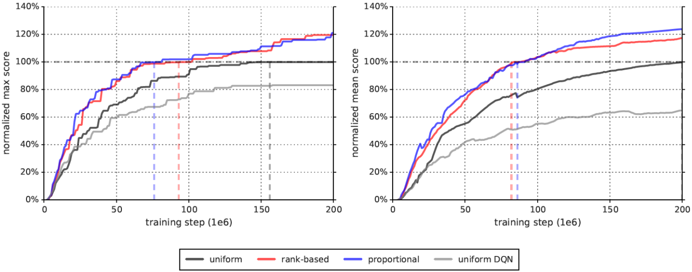

Figure: median (left) or mean (right) over 57 games of maximum baseline-normalized score achieved so far

Conclusion: PER speed up learning by factor 2 and new SOA Atari benchmark.

Parameter Noise

Parameter Space Noise for Exploration. ICLR 2018. OpenAI.

Intuition: action noise may obtain different action a for a *fixed* state s.
Solution: add noise to policy network level (for DDPG and off-policy algorithm)

Schematic Diagram

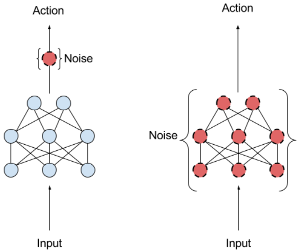

Figure: Action space noise (left), compared to parameter space noise (right)

Conclusion: Parameter Noise rarely decrease performance, often result in improved performance and allow solving environments with sparse rewards. It is an interesting alternative.

Distributed Version for DDPG

- Multiple Distributed Parallel Actors
- Prioritized Experience Replay
- N-step returns
- Distributional Critic

last two items belong to D4PG (Distributed Distributional DDPG), a paper published in ICLR 2018 by DeepMind.

Shortages Remain in DDPG and RL

DDPG

- can not apply large network
- depend on effective exploration
- update always too big for network

RL

- computational consuming (mostly CPUs)
- good reward design
- always feature engineering

References

- Deterministic Policy Gradient on ICML
- DQN on Nature
- Deep Deterministic Policy Gradient on ICLR
- Priority Experience Replay on ICLR
- Parameter Noise on ICLR
- Distributed Distributional DDPG on ICLR
- Discuss on Batch Normalization in DDPG - Zhihu
- PKU AI Lab DDPG Implementation - GitHub