Parameterized Deep Q-Network Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space

백승언

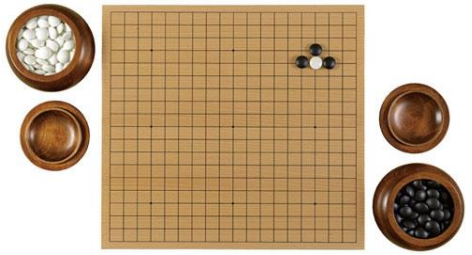
07 Mar, 2022

Contents

Introduction

Reinforcement learning in games (I)

- **Milestones**
  - Backgammon (1992) ➔ TD(λ) algorithm
  - Chess (1997) ➔ TD(λ) algorithm
  - Atari game (2013) ➔ DQN
  - Go (2016) ➔ MC Tree search



Reinforcement learning in games (II)

- **Recent achievement**
  - Dota II (2019) ➔ PPO
    - OpenAI “Five” model defeats world champions
    - Collaboration with **Valve** for training agent

VALVE

  - StarCraft II (2019) ➔ PPO, UPGO, V-trace, …
    - DeepMind “AlphaStar” achieves the GrandMaster level
    - Collaboration with **Blizzard** for training agent

BLIZZARD ENTERTAINMENT

  - 이쯤되면 떠오르는… League Of Legend(LOL!)
    - Where are **Tencent** and **Riot Games**?

Tencent 腾讯

RIOT GAMES

Dota II(OpenAI “Five”)

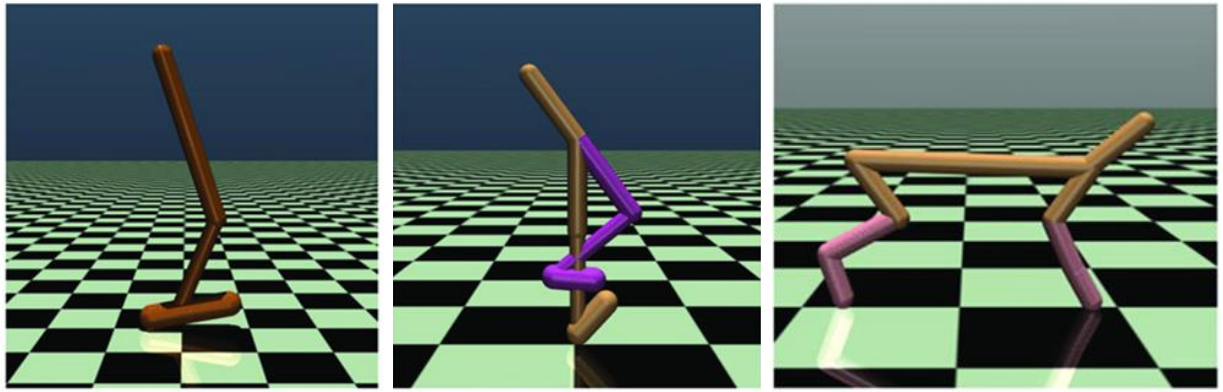
StarCraft II(DeepMind AlphaStar)

Parameterized Deep Q-Network Learning: RL with Hybrid Action Space

Backgrounds (I)

- **Most existing DRL methods require the action space to be either finite and discrete or continuous.**
  - The algorithms for discrete action space: DQN family, A3C, PPO, REDQ, …
  - The algorithms for continuous action space: DDPG, TD3, A3C, PPO, …
  - The authors consider the RL problem with a **discrete-continuous hybrid action space**

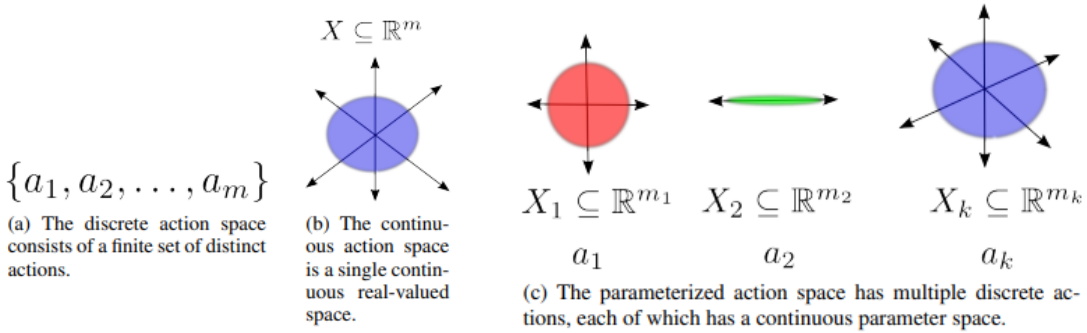
Discrete control
$\{A, B, X, Y, up, down\}$



Continuous control
$\{Vx, Vy, Vz\}, \quad V_i \in [0,1]$

Backgrounds (II)

- **Previous approaches for parameterized action space**
  - Masson et al.(2015): Q-PAMDP
    - Parameterized action $A = \mathrm{U}_{a \in A_d}\{(a, x) \mid x \in \chi_a\}$ and parameterized action space MDPs(PAMDPs) were proposed.
    - They proposed the learning framework that alternately updates the network weights for discrete actions using Q-learning and for continuous parameters using policy search
  - Hausknecht and Stone(2016): PA-DDPG
    - To handle such parameterized actions, DDPG algorithm was applied on the relaxed action space directly.
    - They defined parameterized action as follows:
      - $(Dash, p_{dash1}, p_{dash2}) \cup (Turn, p_{turn3}) \cup (Tackle, p_{tackle4}) \cup (Kick, p_{kick5}, p_{kick6})$.
      - The actor network factors the action into one output layer for 4 discrete actions and another for all 6 continuous params.

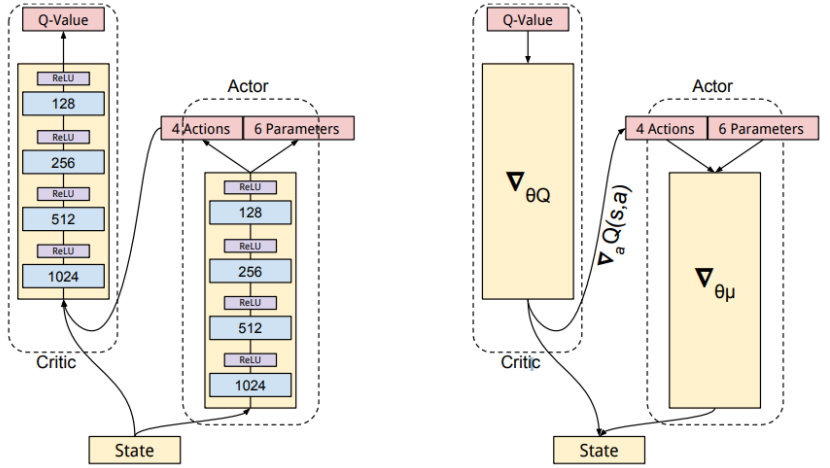


(a) The discrete action space consists of a finite set of distinct actions.

(b) The continuous action space is a single continuous real-valued space.

(c) The parameterized action space has multiple discrete actions, each of which has a continuous parameter space.

Three types of action spaces: discrete, continuous, and parameterized



Network architecture(left), Actor update procedure(right)

Methodology (I)

- **Hybrid action is defined by following hierarchical structure**
  - High level action k from discrete set $[k]$ was first chosen. ➔ $[k]$ $denotes$ $\{1, 2, \ldots, K\}$
  - Upon choosing k, low level parameter $x_k \in \chi_k$ was chosen
    - x_k is associated with the k-th high level action, χ_k is a continuous set for all $k \in [K]$
    - Ex) game내에서 사용할 skill을 고르고, skill을 적중 시킬 위치를 continuous하게 고르는 것
  - Therefore, the authors defined **discrete-continuous hybrid action space**
    - $\boldsymbol{A = \{(k, x_k) | x_k \in \chi_k \; for \; all \; k \in [K]\}}$
- **Bellman equation of P-DQN**
  - The authors considered MDP with parameterized action space $\boldsymbol{A}$
    - Action value function is denoted by $Q(s, a) = Q(s, k, x_k)$
  - Then, Bellman equation becomes
    - $\boldsymbol{Q(s_t, k_t, x_k) = \mathbb{E}_{r_t, s_{t+1}}\left[r_t + \gamma \max_{k \in [K]} \sup_{x_k \in \chi_k} Q(s_{t+1}, k, x_k) \mid s_t = s, a_t = (k_t, x_k)\right]}$
    - * Problem occurred! supremum over continuous space χ_k is intractable

Methodology (II)

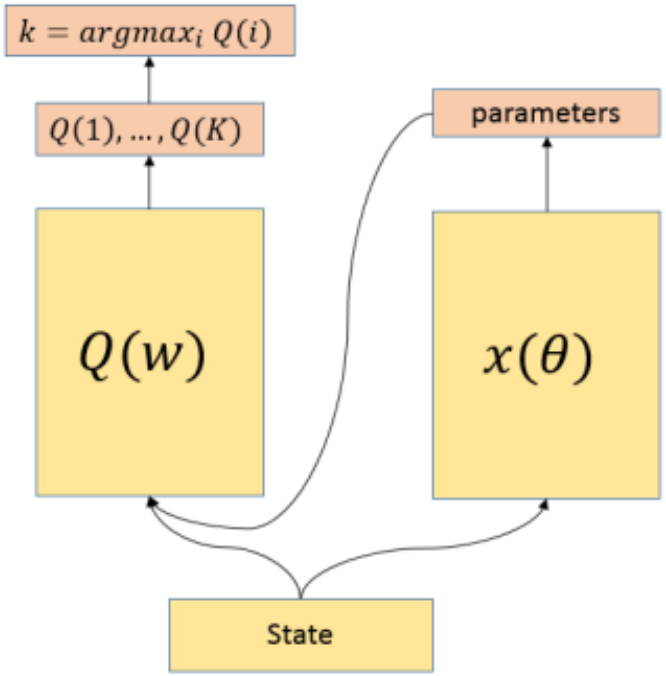
- **Approximation of parameterized action value function $Q(s, k, x_k)$**
  - Bellman equation with parametrized action $A \coloneqq (k, x_k)$
    - $Q(s_t, k_t, x_k) = \mathbb{E}_{r_t, s_{t+1}}\left[r_t + \gamma \max_{k \in [K]} \sup_{x_k \in \chi_k} Q(s_{t+1}, k, x_k) \mid s_t = s, a_t = (k_t, x_k)\right]$
  - The authors first solve $x_k^* = argsup_{x_k \in K} Q(s_{t+1}, k, x_k)$ for each $k \in [K]$ and take the largest $Q(s_{t+1}, k, x_k^*)$
    - When the function Q is fixed, for any $s \in S$ and $k \in [K]$, $argsup_{x_k \in \chi_k} Q(s, k, x_k)$ as function $x_k^Q : S \to \chi_k$.
  - Then, Bellman equation can be rewritten
    - $Q\left(s_t, k_t, x_{k_t}\right) = \mathbb{E}_{r_t, s_{t+1}}\left[r_t + \gamma \max_{k \in [K]} Q\left(s_{t+1}, k, x_k^Q(s_{t+1})\right) \middle| s_t = s\right]$
    - It resembles the classical Bellman equation with $A \coloneqq (k)$
  - Similar to the DQN, they used neural network for action value function
    - First, deep neural network $Q(s, k, x_k; \omega)$ used to approximate $Q(s, k, x_k)$
    - Moreover, for such a $Q\left(s, k, x_k; \omega_{,}\right)$, they approximate $x_k^Q(s_{t+1})$ with a deterministic policy network $x_k(s; \theta) : S \to \chi_k$
    - That is, when ω is fixed, θ is found such that $Q(s, k, x_k(s; \theta); \omega) \approx \sup_{x_k \in \chi_k} Q(s, k, x_k; \omega)$ for each $k \in [K]$

Methodology (III)

- **Loss function of ω and θ**
  - Bellman equation with parametrized action $A \coloneqq (k, x_k)$
    - $Q(s_t, k_t, x_k) = \mathbb{E}_{r_t, s_{t+1}}\left[r_t + \gamma \max_{k \in [K]} \sup_{x_k \in \chi_k} Q(s_{t+1}, k, x_k) \mid s_t = s, a_t = (k_t, x_k)\right]$
  - Approximation of parameterized action value function $\boldsymbol{Q(s, k, x_k)}$ with deep neural networks
    - $Q(s, k, x_k) \approx Q(s, k, x_k(s; \theta); \omega) \approx \sup_{x_k \in \chi_k} Q(s, k, x_k; \omega)$ for each $k \in [K]$
  - Like DQN, ω could be estimated by minimizing the TD error via gradient descent
    - $y_t = r_{t+1} + \gamma \max_{k \in [K]} Q(s_{t+1}, k, x_k(s_{t+1}, \theta_t); \omega_t)$, $Loss_t^Q(\omega) = \frac{1}{2}[Q(s_t, k_t, x_k; \omega) - y_t]^2$
  - Like DDPG, θ could be estimated by maximizing the deterministic policy gradient via gradient descent
    - $Loss_t^\Theta(\theta) = -\Sigma_{k=1}^k Q(s_k, k, x_k(s_t; \theta); \omega_t)$
  - In the ideal case, ω_t is fixed when $Loss_t^\Theta(\theta)$ is minimized.
    - But the authors could approximately achieve such a goal in an online fashion via two-timescale update rule.
    - In specific, they update ω with stepsize α_t that is asymptotically negligible compered with the stepsize β_t for θ

Methodology (IV)

- **Architecture of P-DQN**
  - Continuous action $x_k(s;\theta)$ was first outputted
  - After that, discrete action k was selected
  - Finally, parameterized action $A = (k, x_k)$ was combined



Network architecture of P-DQN

- **Total training sequence of P-DQN**
  - Both ϵ-greedy(DQN) and action noise(DDPG) was used for exploration
  - Experience replay was applied
  - n-step TD error was used

Algorithm 1 Parametrized Deep Q-Network (P-DQN) with Experience Replay

Input: Stepsizes $\{\alpha_t, \beta_t\}_{t\geq 0}$, exploration parameter ϵ, minibatch size B, a probability distribution ξ.
Initialize network weights ω_1 and θ_1.
for $t = 1, 2, \ldots, T$ **do**
Compute action parameters $x_k \leftarrow x_k(s_t, \theta_t)$.
Select action $a_t = (k_t, x_{k_t})$ according to the ϵ-greedy policy

$$a_t = \begin{cases} \text{a sample from distribution } \xi & \text{with probability } \epsilon, \\ (k_t, x_{k_t}) \ \text{ such that } \ k_t = \arg\max_{k\in[K]} Q(s_t, k, x_k; \omega_t) & \text{with probability } 1-\epsilon. \end{cases}$$

Take action a_t, observe reward r_t and the next state s_{t+1}.
Store transition $[s_t, a_t, r_t, s_{t+1}]$ into $\mathcal{D}$.
Sample B transitions $\{s_b, a_b, r_b, s_{b+1}\}_{b\in[B]}$ randomly from $\mathcal{D}$.
Define the target y_b by

$$y_b = \begin{cases} r_b & \text{if } s_{b+1} \text{ is the terminal state,} \\ r_b + \max_{k\in[K]} \gamma Q\big(s_{b+1}, k, x_k(s_{b+1}, \theta_t); \omega_t\big) & \text{if otherwise.} \end{cases}$$

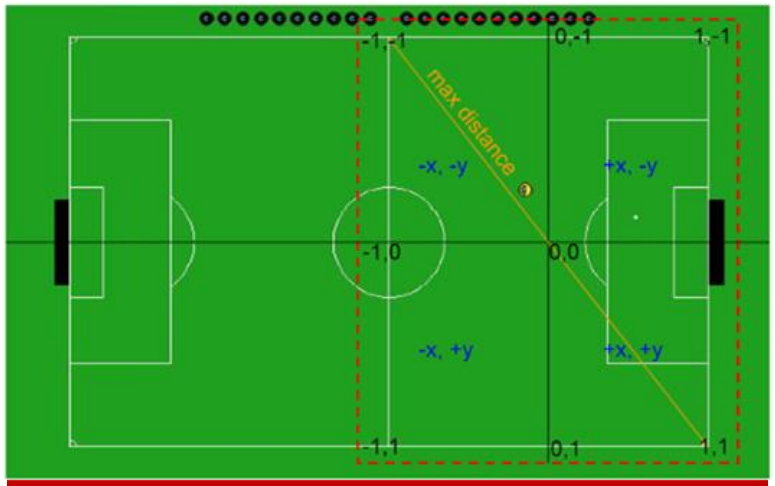
Use data $\{y_b, s_b, a_b\}_{b\in[B]}$ to compute the stochastic gradient $\nabla_\omega \ell_t^Q(\omega)$ and $\nabla_\theta \ell_t^\Theta(\theta)$.
Update the weights by $\omega_{t+1} \leftarrow \omega_t - \alpha_t \nabla_\omega \ell_t^Q(\omega_t)$ and $\theta_{t+1} \leftarrow \theta_t - \beta_t \nabla_\theta \ell_t^\Theta(\theta_t)$.
end for

Pseudo code of P-DQN

Experiments

Environments

- **P-DQN was validated in several simulation environments**
  - Half Field Offense domain in RoboCup soccer(HFO)
    - **State:** 58 continuously-valued features
    - **Action:** {Dash(power, direction), Kick(power, direction), Stop}
      - Direction ∈ [-180, 180], power ∈ [0, 100]
    - **Reward:** hand-crafted intensive reward
      - $r_t = \Delta d_t(a,b) + 3\Delta d_t(b,g) + \mathbb{I}_{t+1}^{kick} + 5\mathbb{I}_{t+1}^{goal}$
      - $d_t(a,b)$: dist to agent & ball, $d_t(b,g)$: dist to goal & ball



HFO environment

  - The solo mode in game King Of Glory(KOG)
    - **State:** 179-dimensional feature vector
    - **Action:** {Move(direction), Attack, skill1(x, y), skill2(direction), skill3(direction), retreat}
    - **Reward:** shaped reward
      - Gold difference, HP difference, Kill/Death, Tower HP difference, Tower destroyed, Winning game, Moving forward reward 등 고려

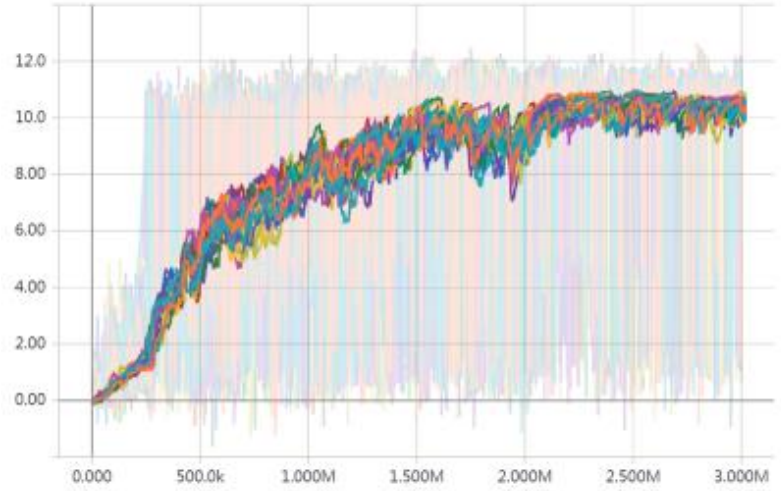
KOG environment

Results (I) – HFO environment

- **Comparison with PA-DDPG model**
  - Results in left column of table quoted from the Hausknecht and Stone(2016)
  - 실험군의 경우, 10개의 P-DQN에 대해, 각각 실험을 100번 수행하여 평균 낸 결과를 도시함
    - 실험 결과, 대조군인 PA-DDPG model이 안정적인 최적화 결과를 얻지 못하는 반면, P-DQN의 경우 대부분의 모델이 안정적으로 높은 점수를 학습하는 것을 확인
    - Hand-crafted agent인 Helios' Campion에 비해서도 모든 모델이 높은 점수를 얻은 것을 확인
    - Youtube를 통해 실험 결과 공개, https://www.youtube.com/watch?v=fwJGR-QJ9TE

	Scoring Percent	Avg. Step to Goal		Scoring Percent	Avg. Step to Goal
Helios' Champion	.962	72.0	P-DQN_1	.997	78.1
SARSA	.81	70.7	P-DQN_2	.997	78.1
DDPG_1	1	108.0	P-DQN_3	.996	78.1
DDPG_2	.99	107.1	P-DQN_4	.994	81.5
DDPG_3	.98	104.8	P-DQN_5	.992	78.7
DDPG_4	.96	112.3	P-DQN_6	.991	79.9
DDPG_5	.94	119.1	P-DQN_7	.985	82.2
DDPG_6	.84	113.2	P-DQN_8	.984	87.9
DDPG_7	.80	118.2	P-DQN_9	.979	78.5

Evaluation performance comparison with different methods



Episode reward sum in training

Results (II) – KOG environment

- **Comparison with PA-DDPG model**
  - 위 실험과 동일하게, 대조군으로써 PA-DDPG model을 선정
  - 실험 결과, 제안된 P-DQN model로 학습한 결과가, PA-DDPG model에 비해 더욱 빠른 시간에 적을 무찌르는 것을 확인(a1, b1)
  - 또한, 더욱 빠른 시간 안에 agent의 학습이 이뤄진다는 것을 확인(a2, b2)

(a1) Episode length in training

(b1) Episode length in training

(a2) Episode reward sum in training

(b2) Episode reward sum in training

Results of (a) P-DQN, (b) Hauscknecht and Stond DDPG model

Thank you!

Q&A

Implementation of P-DQN