Deep Reinforcement Learning

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

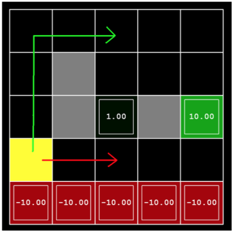
Outline

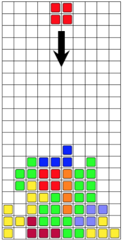
(Tabular) RL

- Q-learning:
  $Q^*(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q^*(\boldsymbol{s},\boldsymbol{a}) + \eta\left[\left(R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}',\boldsymbol{a}')\right) - Q^*(\boldsymbol{s},\boldsymbol{a})\right]$
- SARSA:
  $Q_\pi(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q_\pi(\boldsymbol{s},\boldsymbol{a}) + \eta\left[\left(R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma Q_\pi(\boldsymbol{s}',\pi(\boldsymbol{s}'))\right) - Q_\pi(\boldsymbol{s},\boldsymbol{a})\right]$

(Tabular) RL

- Q-learning:
  $Q^*(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q^*(\boldsymbol{s},\boldsymbol{a}) + \eta\left[\left(R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}',\boldsymbol{a}')\right) - Q^*(\boldsymbol{s},\boldsymbol{a})\right]$
- SARSA:
  $Q_\pi(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q_\pi(\boldsymbol{s},\boldsymbol{a}) + \eta\left[\left(R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma Q_\pi(\boldsymbol{s}',\pi(\boldsymbol{s}'))\right) - Q_\pi(\boldsymbol{s},\boldsymbol{a})\right]$
- In realistic environments with large state/action space, requires a large table to store Q^*/Q_π values
  - Maze: $O(10^1)$, Tetris: $O(10^{60})$, Atari: $O(10^{16922})$ pixels
  - Continuous states/actions?
- May not be able to visit all $(\boldsymbol{s},\boldsymbol{a})$'s in limited training time





Generalizing across States

- Idea: to learn a function $f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)$ (resp. f_{Q_π}) that approximates $Q^*(\boldsymbol{s},\boldsymbol{a})$ (resp. $Q_\pi(\boldsymbol{s},\boldsymbol{a})$), $\forall \boldsymbol{s},\boldsymbol{a}$
  - Trained by a small number (millions) of samples
  - Generalizes to unseen states/actions
  - Smaller Θ to store

Generalizing across States

- Idea: to learn a function $f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)$ (resp. f_{Q_π}) that approximates $Q^*(\boldsymbol{s},\boldsymbol{a})$ (resp. $Q_\pi(\boldsymbol{s},\boldsymbol{a})$), $\forall \boldsymbol{s},\boldsymbol{a}$
  - Trained by a small number (millions) of samples
  - Generalizes to unseen states/actions
  - Smaller Θ to store
- E.g., in Q-learning, Q^* should satisfy Bellman optimality equation:

$$Q^*(\boldsymbol{s},\boldsymbol{a}) \leftarrow \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}',\boldsymbol{a}')], \forall \boldsymbol{s},\boldsymbol{a}$$

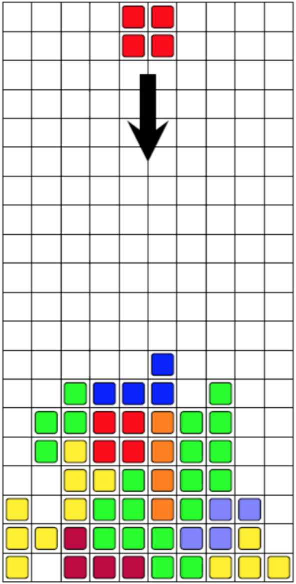
- Algorithm (TD estimate): initialize Θ arbitrarily, iterate until converge:
  1. Take action $\boldsymbol{a}$ from $\boldsymbol{s}$ using some exploration policy π' derived from f_{Q^*} (e.g., ε-greedy)
  2. Observe $\boldsymbol{s}'$ and reward $R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')$, update Θ using SGD:

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta C, \text{ where}$$

$$C(\Theta) = \left[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}',\boldsymbol{a}';\Theta) - f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)\right]^2$$

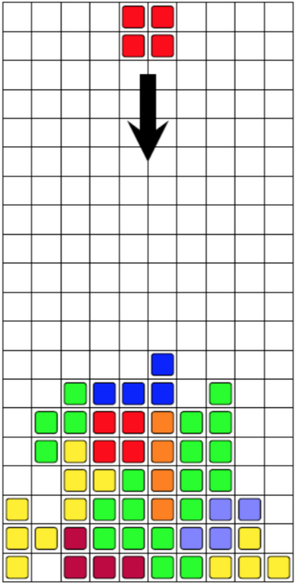
Works If with Careful Feature Engineering

- Tetris: [1]
  - States: $O(10^{60})$ configurations
  - Actions: rotation and translation to falling piece
- $f(\boldsymbol{s},\boldsymbol{a};\Theta)$ and $C(\Theta)$ modeled as an approximated linear programming problem
- Hand-crafted features (22 in total)



Works If with Careful Feature Engineering

- Tetris: [1]
  - States: $O(10^{60})$ configurations
  - Actions: rotation and translation to falling piece
- $f(\boldsymbol{s},\boldsymbol{a};\Theta)$ and $C(\Theta)$ modeled as an approximated linear programming problem
- Hand-crafted features (22 in total)
- Why not use a deep neural network to represent Q_Θ?
  - One model for different tasks
  - Automatically learned features



Deep RL

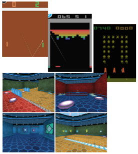
- ***Value-based***: use DNNs to represent ***value/Q-function***
  - E.g., DQN
  - $\pi^*(\boldsymbol{s}) \leftarrow \arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s},\boldsymbol{a})$ only feasible if actions are discrete
- ***Policy-based***: use DNNs to represent ***policy π***
  - E.g., DDPG, Action-Critic, A3C, TRPO, PPO
- **Model-based**: deep RL when **MDP/env. model** is known
  - E.g., AlphaGo

Kohl and Stone, 2004

Ng et al, 2004

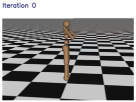
Tedrake et al, 2005

Kober and Peters, 2009

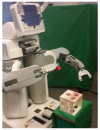


Mnih et al 2013 (DQN)
Mnih et al, 2015 (A3C)

Silver et al, 2014 (DPG)
Lillicrap et al, 2015 (DDPG)



Schulman et al, 2016 (TRPO + GAE)



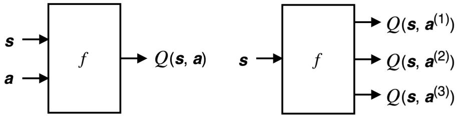
Levine*, Finn*, et al, 2016 (GPS)

Silver*, Huang*, et al, 2016 (AlphaGo)

Outline

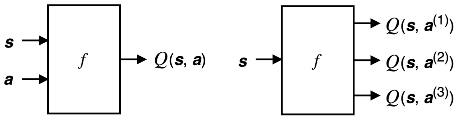
DNNs for Q^*

- Use a DNN $f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)$ to represent $Q^*(\boldsymbol{s},\boldsymbol{a})$



DNNs for Q^*

- Use a DNN $f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)$ to represent $Q^*(\boldsymbol{s},\boldsymbol{a})$



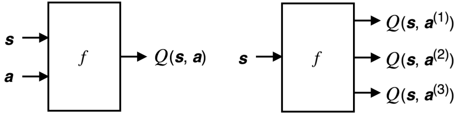
- Algorithm (TD): initialize Θ arbitrarily, iterate until converge:
  1. Take action $\boldsymbol{a}$ from $\boldsymbol{s}$ using some exploration policy π' derived from f_{Q^*} (e.g., ε-greedy)
  2. Observe $\boldsymbol{s}'$ and reward $R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')$, update Θ using SGD:

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} C, \text{ where}$$

$$C(\Theta) = \left[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}',\boldsymbol{a}';\Theta) - f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta) \right]^2$$

DNNs for Q^*

- Use a DNN $f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)$ to represent $Q^*(\boldsymbol{s},\boldsymbol{a})$



- Algorithm (TD): initialize Θ arbitrarily, iterate until converge:
  1. Take action $\boldsymbol{a}$ from $\boldsymbol{s}$ using some exploration policy π' derived from f_{Q^*} (e.g., ε-greedy)
  2. Observe $\boldsymbol{s}'$ and reward $R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')$, update Θ using SGD:

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta C, \text{ where}$$

$$C(\Theta) = \left[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}',\boldsymbol{a}';\Theta) - f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta) \right]^2$$

- However, ***diverges*** due to
  - Samples are correlated (violates i.i.d. assumption of training examples)
  - Non-stationary target ($f_{Q^*}(\boldsymbol{s}',\boldsymbol{a}')$ changes as Θ is updated for current $\boldsymbol{a}$)

Outline

Deep Q-Network (DQN)

- Naive TD algorithm diverges due to:
  - Samples are correlated
  - Non-stationary target

Deep Q-Network (DQN)

- Naive TD algorithm diverges due to:
  - Samples are correlated
  - Non-stationary target
- Stabilization techniques proposed by (Nature) DQN [5]:
  - ***Experience replay***
  - ***Delayed target network***

Experience Replay

- Use a replay memory $\mathbb{D}$ to store recently seen transitions $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')$'s
- Sample a mini-batch from $\mathbb{D}$ and update Θ

Experience Replay

- Use a replay memory $\mathbb{D}$ to store recently seen transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$'s
- Sample a mini-batch from $\mathbb{D}$ and update Θ
- Algorithm (TD): initialize Θ arbitrarily, iterate until converge:
  1. Take action $\boldsymbol{a}$ from $\boldsymbol{s}$ using π' derived from f_{Q^*} (e.g., ε-greedy)
  2. Observe $\boldsymbol{s}'$ and reward R, add $(\boldsymbol{s}, \boldsymbol{a}, R, \boldsymbol{s}')$ to $\mathbb{D}$
  3. Sample a mini-batch of $(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}, R^{(i)}, \boldsymbol{s}^{(i+1)})$'s from $\mathbb{D}$, do:

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} C, \text{ where}$$

$$C(\Theta) = \sum_i \left[R^{(i)} + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}^{(i+1)}, \boldsymbol{a}'; \Theta) - f_{Q^*}(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}; \Theta) \right]^2$$

Delayed Target Network

- To avoid chasing a moving target, set the target value at network output parametrized by ***old*** Θ^-

Delayed Target Network

- To avoid chasing a moving target, set the target value at network output parametrized by ***old*** Θ^-
- Algorithm (TD): initialize Θ arbitrarily and $\Theta^- = \Theta$, iterate until converge:
  1. Take action $\boldsymbol{a}$ from $\boldsymbol{s}$ using π' derived from f_{Q^*} (e.g., ε-greedy)
  2. Observe $\boldsymbol{s}'$ and reward R, add $(\boldsymbol{s},\boldsymbol{a},R,\boldsymbol{s}')$ to $\mathbb{D}$
  3. Sample a mini-batch of $(\boldsymbol{s}^{(i)},\boldsymbol{a}^{(i)},R^{(i)},\boldsymbol{s}^{(i+1)})$'s from $\mathbb{D}$, do:

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} C, \text{ where}$$

$$C(\Theta) = \sum_i \left[R^{(i)} + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}^{(i+1)},\boldsymbol{a}';\Theta^-) - f_{Q^*}(\boldsymbol{s}^{(i)},\boldsymbol{a}^{(i)};\Theta) \right]^2$$

  4. Update $\Theta^- \leftarrow \Theta$ every K iterations

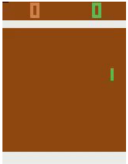
Other Tricks

- Optimization techniques matter in deep RL
    - Optimization error may lead to wrong traditions (trajectory)
    - And bad final policy

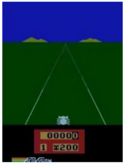
Other Tricks

- Optimization techniques matter in deep RL
  - Optimization error may lead to wrong traditions (trajectory)
  - And bad final policy
- ***Reward clipping*** for better conditioned gradients
  - Can't differentiate between small and large rewards
  - Better use batch normalization
- Use ***RMSProp*** instead of vanilla SGD for adaptive learning rate

DQN on Atari

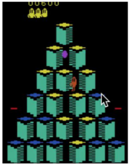


Pong



Enduro

Beamrider

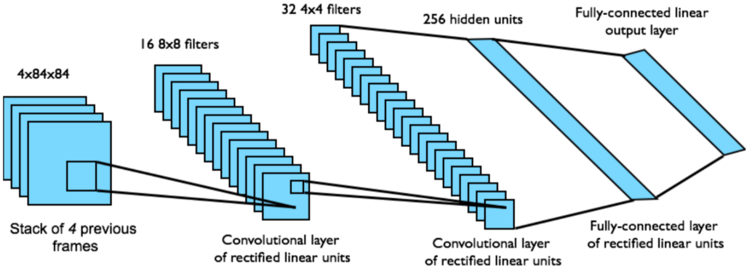


Q*bert

- 49 Atari 2600 games
- States: raw pixels
- Actions: 18 joystick/button positions
- Rewards: changes in score

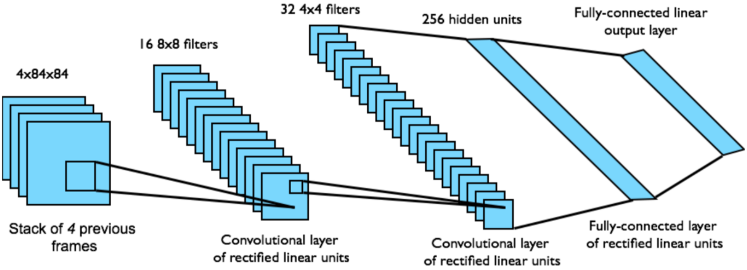
Network Architecture

- End-to-end from raw pixels to $Q^*(\boldsymbol{s},\boldsymbol{a})$
- CNN + fully connected layers
- Input: state $\boldsymbol{s}$ a stack of raw pixels from last 4 frames
- Output: 18 $Q^*(\boldsymbol{s},\boldsymbol{a})$'s (one for each action)



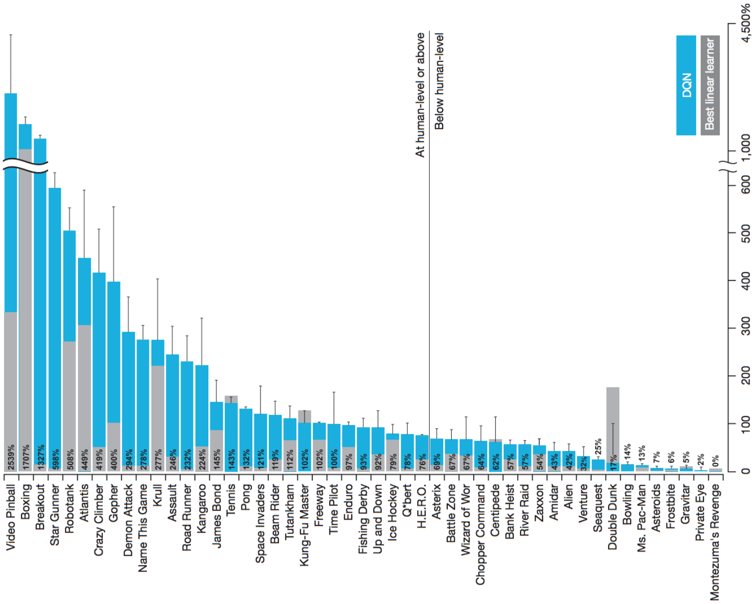
Network Architecture

- End-to-end from raw pixels to $Q^*(\boldsymbol{s},\boldsymbol{a})$
- CNN + fully connected layers
- Input: state $\boldsymbol{s}$ a stack of raw pixels from last 4 frames
- Output: 18 $Q^*(\boldsymbol{s},\boldsymbol{a})$'s (one for each action)



- Network architecture is ***fixed across all games***

Results



At human-level or above
Below human-level
DQN
Best linear learner
4,500%
1,000
600
500
400
300
200
100
0
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

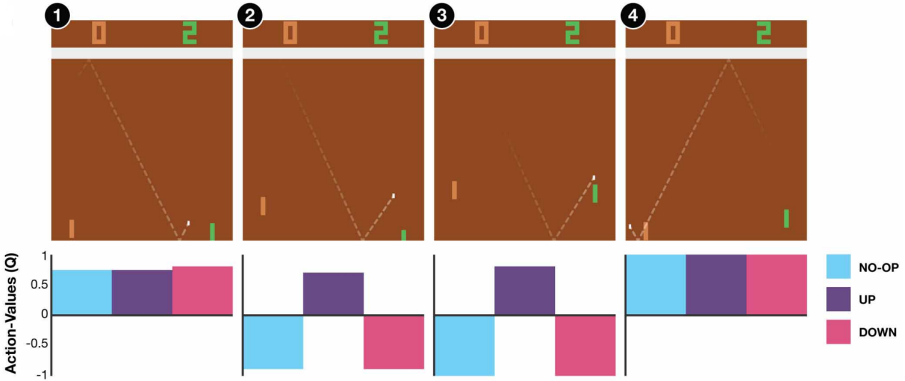
Effect of Stability Techniques

DQN

	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

- Delayed target network is less useful for large networks

Predicted Q^* Values for Pong



Outline

Improvements since DQN

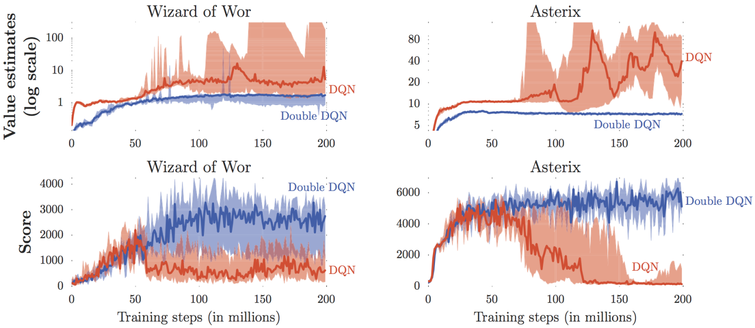
- Stabilization:
  - Double DQN [9]
  - Prioritized replay [7]
- Modeling additional prior:
  - Duelling network [10]
- Exploration:
  - NoisyNet [2]
- Large-scale implementation

Double DQN I

- DQN update rule: $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} C$, where

$$C(\Theta) = \sum_i \left[R^{(i)} + \gamma \max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}^{(i+1)}, \boldsymbol{a}'; \Theta^-) - f_{Q^*}(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}; \Theta) \right]^2$$

- There is an upward bias in $\max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}^{(i+1)}, \boldsymbol{a}'; \Theta^-)$
  - $f_{Q^*}(\boldsymbol{s}^{(i+1)}, \boldsymbol{a}'; \Theta^-)$ with high positive error is preferred
- At each step, the positive error is added to $f_{Q^*}(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}; \Theta)$

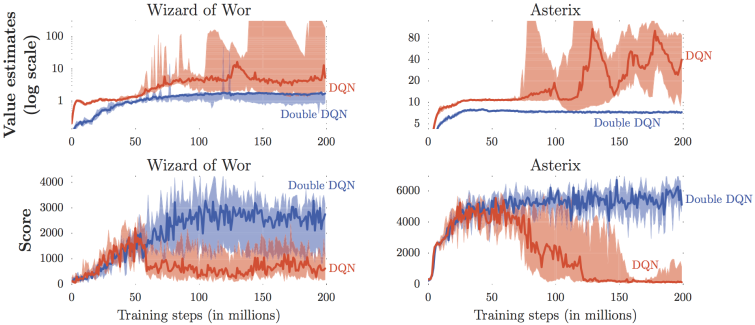


Double DQN II

- Double DQN (DDQN) [9]:

$$C(\Theta) = \sum_i \left[R^{(i)} + \gamma f_{Q^*}(\boldsymbol{s}^{(i+1)}, \arg\max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}^{(i+1)}, \boldsymbol{a}'; \Theta); \Theta^-) - f_{Q^*}(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}; \Theta) \right]^2$$

  - Uses Θ to select the best action
  - Uses Θ^- to evaluate the best action
- Random (unbiased) error added to $f_{Q^*}(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}; \Theta)$ at each step

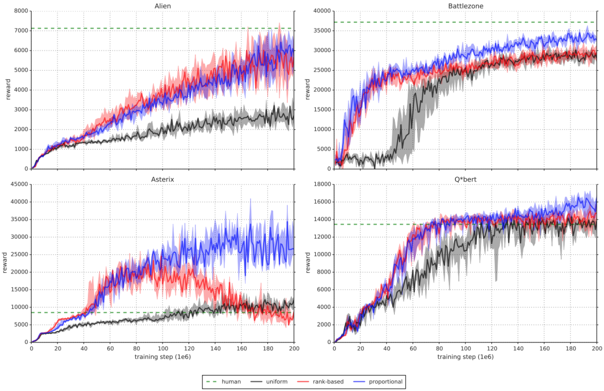


Prioritized Replay [7]

- Not all $(\boldsymbol{s},\boldsymbol{a},R,\boldsymbol{s}')$'s from $\mathbb{D}$ are equally helpful to training f_{Q^*}
- Sample $(\boldsymbol{s},\boldsymbol{a},R,\boldsymbol{s})$'s with probability proportional to "surprise" in terms of Bellman equation:

$$\left|R+\gamma\max_{\boldsymbol{a}'} f_{Q^*}(\boldsymbol{s}',\boldsymbol{a}';\Theta^-) - f_{Q^*}(\boldsymbol{s},\boldsymbol{a};\Theta)\right|$$

- Rank-based alternative: $\mathbb{D}$ a priority queue

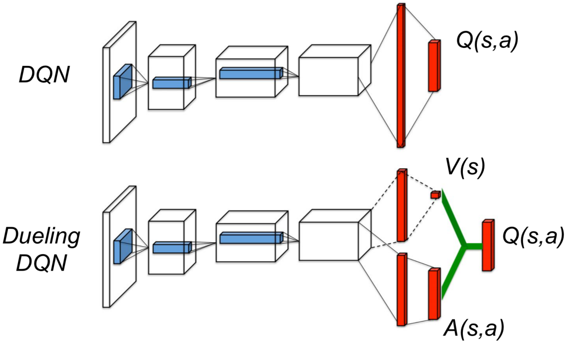


Dueling Network [10]

- $Q^*(\boldsymbol{s},\boldsymbol{a}) = V^*(\boldsymbol{s}) + A^*(\boldsymbol{s},\boldsymbol{a})$
  - $A^*(\boldsymbol{s},\boldsymbol{a})$ the ***advantage function*** of $\boldsymbol{a}$

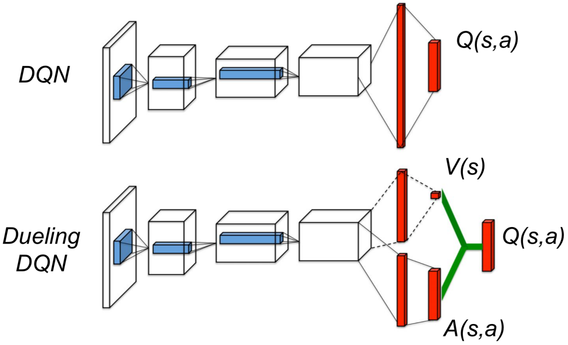
Dueling Network [10]

- $Q^*(\boldsymbol{s},\boldsymbol{a}) = V^*(\boldsymbol{s}) + A^*(\boldsymbol{s},\boldsymbol{a})$
  - $A^*(\boldsymbol{s},\boldsymbol{a})$ the ***advantage function*** of $\boldsymbol{a}$
- Idea: to model this prior and learn $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + f_{A^*}(\boldsymbol{s},\boldsymbol{a})$



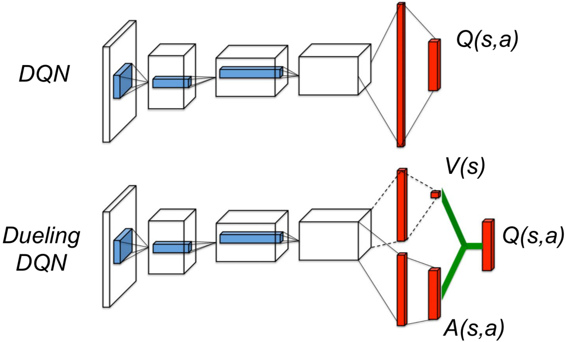
Dueling Network [10]

- $Q^*(\boldsymbol{s},\boldsymbol{a}) = V^*(\boldsymbol{s}) + A^*(\boldsymbol{s},\boldsymbol{a})$
  - $A^*(\boldsymbol{s},\boldsymbol{a})$ the ***advantage function*** of $\boldsymbol{a}$
- Idea: to model this prior and learn $f_{Q*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + f_{A^*}(\boldsymbol{s},\boldsymbol{a})$
  - Not well-defined: $f_{Q*}(\boldsymbol{s},\boldsymbol{a}) = (f_{V^*}(\boldsymbol{s}) + c) + (f_{A^*}(\boldsymbol{s},\boldsymbol{a}) - c)$ for any c
- Dueling DQN: $f_{Q*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + (f_{A^*}(\boldsymbol{s},\boldsymbol{a}) - \max_{\boldsymbol{a}'} f_{A^*}(\boldsymbol{s},\boldsymbol{a}'))$
  - The best action $\boldsymbol{a}^*$ has zero advantage and $f_{Q*}(\boldsymbol{s},\boldsymbol{a}^*) = f_{V^*}(\boldsymbol{s})$

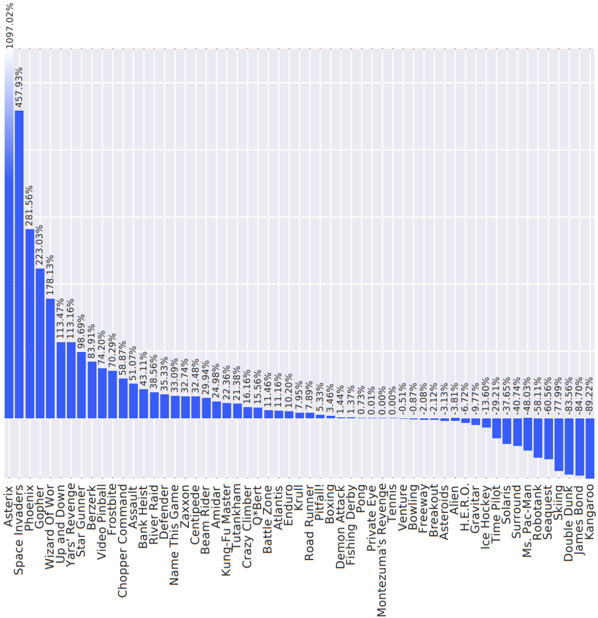


Dueling Network [10]

- $Q^*(\boldsymbol{s},\boldsymbol{a}) = V^*(\boldsymbol{s}) + A^*(\boldsymbol{s},\boldsymbol{a})$
  - $A^*(\boldsymbol{s},\boldsymbol{a})$ the ***advantage function*** of $\boldsymbol{a}$
- Idea: to model this prior and learn $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + f_{A^*}(\boldsymbol{s},\boldsymbol{a})$
  - Not well-defined: $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = (f_{V^*}(\boldsymbol{s}) + c) + (f_{A^*}(\boldsymbol{s},\boldsymbol{a}) - c)$ for any c
- Dueling DQN: $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + (f_{A^*}(\boldsymbol{s},\boldsymbol{a}) - \max_{\boldsymbol{a}'} f_{A^*}(\boldsymbol{s},\boldsymbol{a}'))$
  - The best action $\boldsymbol{a}^*$ has zero advantage and $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}^*) = f_{V^*}(\boldsymbol{s})$
- Stabilized version: $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + (f_{A^*}(\boldsymbol{s},\boldsymbol{a}) - \frac{1}{|\mathbb{A}|}\sum_{\boldsymbol{a}'} f_{A^*}(\boldsymbol{s},\boldsymbol{a}'))$
  - f_{V^*} and f_{A^*} are off-target (by a constant) but f_{A^*} changes more slowly

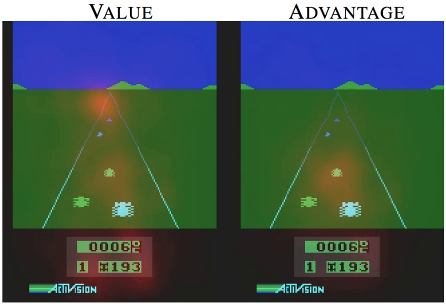


Improvement over Prioritized DDQN



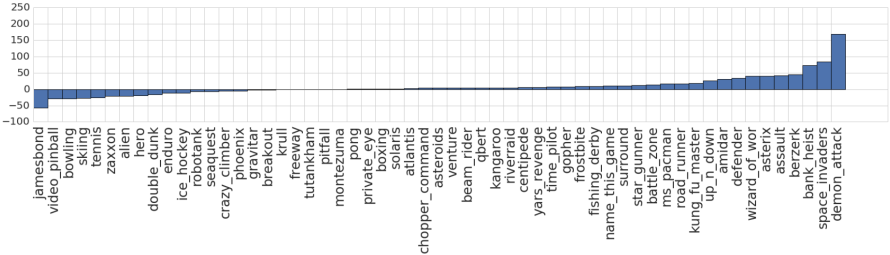
See, Attend, and Drive

- Atari game: Enduro
- Attention mask: $\frac{\partial f_{V^*}}{\partial \boldsymbol{s}}(\boldsymbol{s})$ and $\frac{\partial f_{A^*}}{\partial \boldsymbol{s}}(\boldsymbol{s})$
- $f_{Q^*}(\boldsymbol{s},\boldsymbol{a}) = f_{V^*}(\boldsymbol{s}) + f_{A^*}(\boldsymbol{s},\boldsymbol{a})$
    - f_{V^*} pays attention to the road
    - f_{A^*} pays attention only when there's obstacles in front



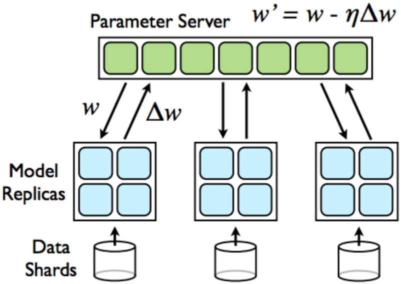
NoisyNet [2]

- Instead of using using ε-greedy for exploration, add noise to Θ
- The level of noise is learned by SGD along with Θ
- Improvement over Dueling DQN:



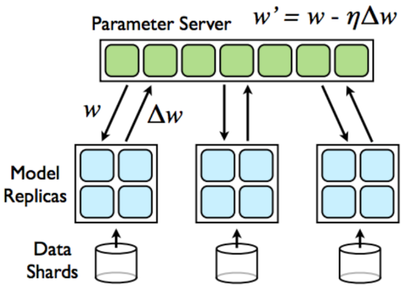
Scaling Up DQN on Single Machine

- Exploits multi-threading of modern CPUs/GPUs
- Run/train multiple agents in parallel (one per thread/GPU)
  - Θ shared between threads in main memory
  - Data-parallelism



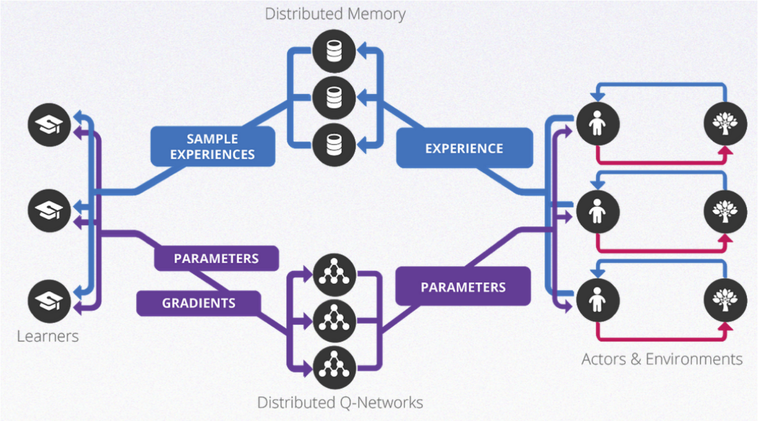
Scaling Up DQN on Single Machine

- Exploits multi-threading of modern CPUs/GPUs
- Run/train multiple agents in parallel (one per thread/GPU)
  - Θ shared between threads in main memory
  - Data-parallelism
- Parallelism ***decorrelates samples***
  - Alternative to experience replay



Scaling Out DQN with Gorila [6]

- Distributed system architecture for large-scale RL
- 10x faster than Nature DQN
- Applied to recommender systems in Google



Outline

Why Policy Network?

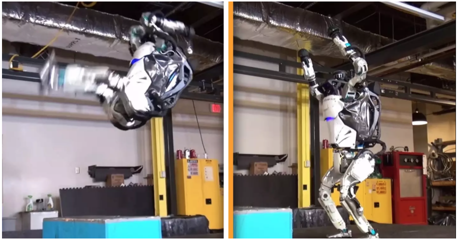
- Policy-based deep RL: use a DNN $g_\pi(\boldsymbol{s};\Phi)$ to approximate $\pi(\boldsymbol{s})$
- Why?

Why Policy Network?

- Policy-based deep RL: use a DNN $g_\pi(\boldsymbol{s};\Phi)$ to approximate $\pi(\boldsymbol{s})$
- Why?
- In DQN (or any method based on value/policy iteration), one needs to solve

$$\pi^* = \arg\max_{\boldsymbol{a}'} Q^*(\boldsymbol{s},\boldsymbol{a}') \text{ or } \hat{\pi} = \arg\max_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s},\boldsymbol{a}')$$

  - Not applicable to continuous action space $\mathbb{A}$ common in, e.g., robotics

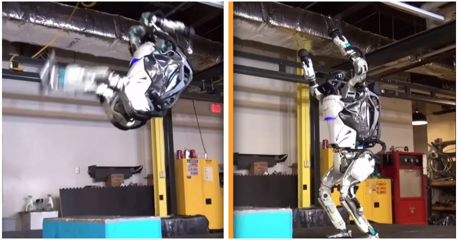


Why Policy Network?

- Policy-based deep RL: use a DNN $g_\pi(\boldsymbol{s};\Phi)$ to approximate $\pi(\boldsymbol{s})$
- Why?
- In DQN (or any method based on value/policy iteration), one needs to solve

$$\pi^* = \arg\max_{\boldsymbol{a}'} Q^*(\boldsymbol{s},\boldsymbol{a}') \text{ or } \hat{\pi} = \arg\max_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s},\boldsymbol{a}')$$

  - Not applicable to continuous action space $\mathbb{A}$ common in, e.g., robotics
- π may be easier to learn than Q or V

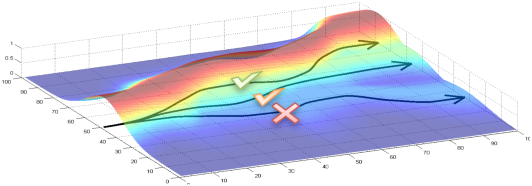


Modeling π

- π can be either
  - deterministic: $g_\pi(\boldsymbol{s};\Phi) = \boldsymbol{a}$, or
  - stochastic: $g_\pi(\boldsymbol{s};\Phi) = \mathrm{P}(\boldsymbol{a}\,|\,\boldsymbol{s})$
- ***Pathwise derivative methods***
  - For deterministic π and continuous $\mathbb{A}$
- ***Policy gradient***/***optimization*** methods
  - For stochastic π

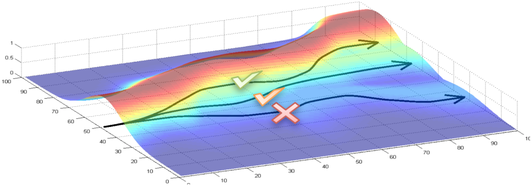
Modeling π

- π can be either
  - deterministic: $g_\pi(\boldsymbol{s};\Phi) = \boldsymbol{a}$, or
  - stochastic: $g_\pi(\boldsymbol{s};\Phi) = \mathrm{P}(\boldsymbol{a}|\boldsymbol{s})$
- *Pathwise derivative methods*
  - For deterministic π and continuous $\mathbb{A}$
  - To find Φ such that $g_\pi(\boldsymbol{s};\Phi)$ gives action $\boldsymbol{a}$ maximizing $Q^*(\boldsymbol{s},\boldsymbol{a})$
  - Changes the trajectory of an episode in the graph of accumulative rewards
- *Policy gradient*/*optimization* methods
  - For stochastic π



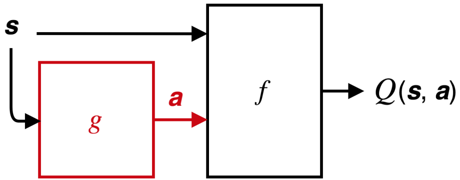
Modeling π

- π can be either
  - deterministic: $g_\pi(\boldsymbol{s};\Phi) = \boldsymbol{a}$, or
  - stochastic: $g_\pi(\boldsymbol{s};\Phi) = \mathrm{P}(\boldsymbol{a}|\boldsymbol{s})$
- ***Pathwise derivative methods***
  - For deterministic π and continuous $\mathbb{A}$
  - To find Φ such that $g_\pi(\boldsymbol{s};\Phi)$ gives action $\boldsymbol{a}$ maximizing $Q^*(\boldsymbol{s},\boldsymbol{a})$
  - Changes the trajectory of an episode in the graph of accumulative rewards
- ***Policy gradient***/***optimization*** methods
  - For stochastic π
  - To find Φ such that gives trajectory of high accumulative rewards
  - Do ***not*** change the trajectory (but its probability) of an episode



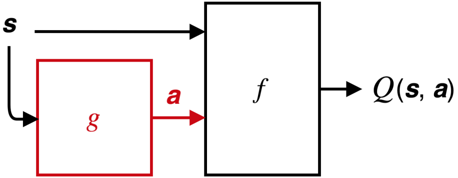
Outline

Deep Deterministic Policy Gradient (DDPG) [4]



- Based on DQN
  - Q-learning is off-policy and works with changing exploration strategies
- Deterministic policy: $g_{\pi^*}(\boldsymbol{s};\Phi) = \boldsymbol{a} \in \mathbb{R}$

Deep Deterministic Policy Gradient (DDPG) [4]



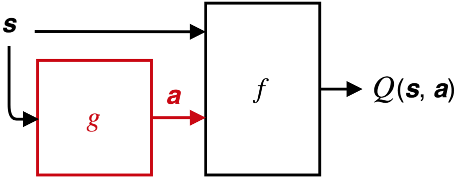
- Based on DQN
  - Q-learning is off-policy and works with changing exploration strategies
- Deterministic policy: $g_{\pi^*}(\boldsymbol{s};\Phi) = \boldsymbol{a} \in \mathbb{R}$
- Goal: to find Φ maximizing $\mathrm{E}_{\mathbf{s}}\left[f_{Q^*}(\mathbf{s},\mathbf{a};\Theta)\right]$, where $\mathbf{a} = g_{\pi^*}(\mathbf{s};\Phi)$
- SGD update rule:

$$\begin{aligned}\Phi \quad & \leftarrow \Phi + \eta \frac{\partial \mathrm{E}_{\mathbf{s}}\left[f_{Q^*}(\mathbf{s},\mathbf{a};\Theta)\right]}{\partial \Phi} \\ & = \Phi + \eta \mathrm{E}_{\mathbf{s}}\left[\frac{\partial f_{Q^*}}{\partial \boldsymbol{a}}(\mathbf{s},\mathbf{a};\Theta) \cdot \frac{\partial g_{\pi^*}}{\partial \Phi}(\mathbf{s};\Phi)\right]\end{aligned}$$

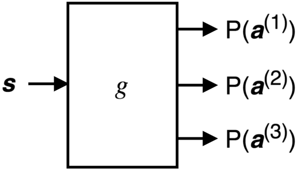
DDPG Algorithm (TD)

- Initialize Θ and Φ arbitrarily, set $\Theta^- = \Theta$ and $\Phi^- = \Phi$, iterate until converge:
  1. Take action $\boldsymbol{a} = g_{\pi^*}(\boldsymbol{s};\Phi) + z$ from $\boldsymbol{s}$, where z is a random noise for exploration
  2. Observe $\boldsymbol{s}'$ and reward R, add $(\boldsymbol{s},\boldsymbol{a},R,\boldsymbol{s}')$ to $\mathbb{D}$
  3. Sample a mini-batch of $(\boldsymbol{s}^{(i)},\boldsymbol{a}^{(i)},R^{(i)},\boldsymbol{s}^{(i+1)})$'s from $\mathbb{D}$
  4. Update Θ:
$$\Theta \leftarrow \Theta - \eta \nabla_\Theta C, \text{ where}$$
$$C(\Theta) = \sum_i \left[R^{(i)} + \gamma f_{Q^*}(\boldsymbol{s}^{(i+1)}, g_{\pi^*}(\boldsymbol{s};\Phi^-);\Theta^-) - f_{Q^*}(\boldsymbol{s}^{(i)},\boldsymbol{a}^{(i)};\Theta) \right]^2$$
  5. Update Φ:
$$\Phi \leftarrow \Phi + \lambda \sum_i \frac{\partial f_{Q^*}}{\partial \boldsymbol{a}}(\boldsymbol{s}^{(i)}, g_{\pi^*}(\boldsymbol{s}^{(i)};\Phi);\Theta) \cdot \frac{\partial g_{\pi^*}}{\partial \Phi}(\boldsymbol{s}^{(i)};\Phi)$$
  6. Update $\Theta^- \leftarrow \tau\Theta + (1-\tau)\Theta^-$ and $\Phi^- \leftarrow \tau\Phi + (1-\tau)\Phi^-$

Limitations



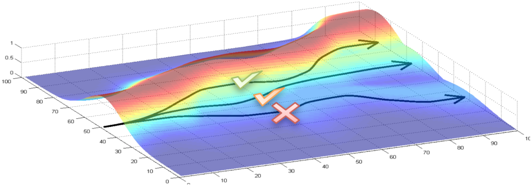
- Only applicable to continuous action space $\mathbb{A}$
- Cannot backprop through samples when calculating $\frac{\partial \mathrm{E}_{\mathbf{s}}\left[f_{Q^*}(\mathbf{s}, g_{\pi^*}(\mathbf{s};\Phi);\Theta)\right]}{\partial \Phi}$
- For discrete $\mathbb{A}$, it's more natural to use a DNN to model a stochastic policy: $g_\pi(\boldsymbol{s}) = \mathrm{P}(\boldsymbol{a}|\boldsymbol{s}), \forall \boldsymbol{a}$



Outline

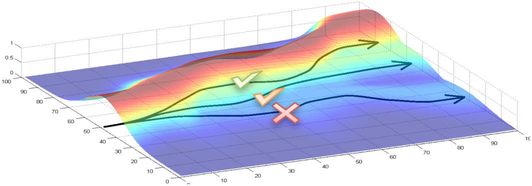
Episodic Policy Gradient

- ***Policy gradient***/***optimization*** methods
  - For stochastic policy: $g_\pi(\boldsymbol{s}) = \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi), \forall \boldsymbol{a}$ (discrete or continuous)
  - Do ***not*** change the trajectory (but its probability) of an episode



Episodic Policy Gradient

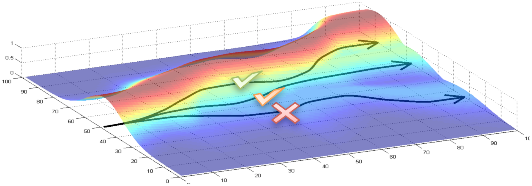
- ***Policy gradient***/***optimization*** methods
  - For stochastic policy: $g_\pi(\boldsymbol{s}) = \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi), \forall \boldsymbol{a}$ (discrete or continuous)
  - Do ***not*** change the trajectory (but its probability) of an episode
- Given an episode, let $\tau = \{(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}, R^{(t)}, \boldsymbol{s}^{(t+1)})\}_t$ be the sequence of state-action transitions
  - Action $\boldsymbol{a}^{(t)}$ sampled from $g_\pi(\boldsymbol{s}^{(t)})$



Episodic Policy Gradient

- ***Policy gradient/optimization*** methods
  - For stochastic policy: $g_\pi(\boldsymbol{s}) = \mathrm{P}(\boldsymbol{a}\,|\,\boldsymbol{s};\Phi), \forall \boldsymbol{a}$ (discrete or continuous)
  - Do ***not*** change the trajectory (but its probability) of an episode
- Given an episode, let $\tau = \{(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}, R^{(t)}, \boldsymbol{s}^{(t+1)})\}_t$ be the sequence of state-action transitions
  - Action $\boldsymbol{a}^{(t)}$ sampled from $g_\pi(\boldsymbol{s}^{(t)})$
- Let $R(\tau) = \sum_t \gamma^t R^{(t)}$, our goal:

$$\arg\max_{\Phi} \mathrm{E}_\tau \left[R(\tau);\Phi\right] = \arg\max_{\Phi} \sum_\tau \mathrm{P}(\tau;\Phi) R(\tau)$$



Policy Gradient

- Let $J(\Phi) = \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau)$, we have:

$$\begin{aligned}
\nabla_\Phi J(\Phi) &= \nabla_\Phi \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau) = \sum_\tau \nabla_\Phi \mathrm{P}(\tau;\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\frac{\nabla_\Phi \mathrm{P}(\tau;\Phi)}{\mathrm{P}(\tau;\Phi)}R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \log \mathrm{P}(\tau;\Phi)R(\tau)
\end{aligned}$$

Policy Gradient

- Let $J(\Phi) = \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau)$, we have:

$$\begin{aligned}
\nabla_\Phi J(\Phi) &= \nabla_\Phi \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau) = \sum_\tau \nabla_\Phi \mathrm{P}(\tau;\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\frac{\nabla_\Phi \mathrm{P}(\tau;\Phi)}{\mathrm{P}(\tau;\Phi)}R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \log \mathrm{P}(\tau;\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \log \prod_t \mathrm{P}(\boldsymbol{s}^{(t+1)}|\boldsymbol{s}^{(t)},\boldsymbol{a}^{(t)})\mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \sum_t \left[\log \mathrm{P}(\boldsymbol{s}^{(t+1)}|\boldsymbol{s}^{(t)},\boldsymbol{a}^{(t)}) + \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)\right]R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)R(\tau)
\end{aligned}$$

Policy Gradient

- Let $J(\Phi) = \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau)$, we have:

$$\begin{aligned}
\nabla_\Phi J(\Phi) &= \nabla_\Phi \sum_\tau \mathrm{P}(\tau;\Phi)R(\tau) = \sum_\tau \nabla_\Phi \mathrm{P}(\tau;\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\frac{\nabla_\Phi \mathrm{P}(\tau;\Phi)}{\mathrm{P}(\tau;\Phi)}R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \log \mathrm{P}(\tau;\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \log \prod_t \mathrm{P}(\boldsymbol{s}^{(t+1)}|\boldsymbol{s}^{(t)},\boldsymbol{a}^{(t)})\mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\nabla_\Phi \sum_t \left[\log \mathrm{P}(\boldsymbol{s}^{(t+1)}|\boldsymbol{s}^{(t)},\boldsymbol{a}^{(t)}) + \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)\right] R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)R(\tau) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)\sum_{t'} \gamma^{t'} R^{(t')}(\boldsymbol{s}^{(t)},\boldsymbol{a}^{(t)},\boldsymbol{s}^{(t+1)}) \\
&= \sum_\tau \mathrm{P}(\tau;\Phi)\sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)}|\boldsymbol{s}^{(t)};\Phi)\sum_{t'=t}^{H} \gamma^{t'} R^{(t')}
\end{aligned}$$

- Assumes that the environment is MDP-alike
  - But no need for the exact model

REINFORCE Algorithm

$$\nabla_\Phi J(\Phi) = \sum_\tau \mathrm{P}(\tau;\Phi) \sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)};\Phi) \sum_{t'=t}^{H} \gamma^{t'} R^{(t')}$$

- REINFORCE (MC estimate): initialize Φ arbitrarily, iterate until converge:
    1. Run episodes $\{\tau^{(i)}\}_i$ by sampling actions from $g(\cdot;\Phi)$
    2. For each time step t in an episode, compute $R^{(i,t)} = \sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$
    3. Update Φ using SGD:

$$\Phi \leftarrow \Phi + \eta \nabla_\Phi \hat{J}, \text{ where}$$

$$\nabla_\Phi \hat{J}(\Phi) = \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)};\Phi) R^{(i,t)}.$$

REINFORCE Algorithm

$$\nabla_\Phi J(\Phi) = \sum_\tau \mathrm{P}(\tau;\Phi) \sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)};\Phi) \sum_{t'=t}^{H} \gamma^{t'} R^{(t')}$$

- REINFORCE (MC estimate): initialize Φ arbitrarily, iterate until converge:
  1. Run episodes $\{\tau^{(i)}\}_i$ by sampling actions from $g(\cdot;\Phi)$
  2. For each time step t in an episode, compute $R^{(i,t)} = \sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$
  3. Update Φ using SGD:

$$\Phi \leftarrow \Phi + \eta \nabla_\Phi \hat{J}, \text{ where}$$

$$\nabla_\Phi \hat{J}(\Phi) = \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)};\Phi) R^{(i,t)}.$$

- REINFORCE-style policy gradient: ∇ log prob. of actions $\times$ episodic rewards

Outline

Variance

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} \,|\, \boldsymbol{s}^{(i,t)};\Phi) \sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ is an MC estimate of $Q_\pi(\boldsymbol{s}^{(i,t)},\boldsymbol{a}^{(i,t)}) = \mathrm{E}_{\{\mathbf{s}^{(t')},\mathbf{a}^{(t')}\}_{t'}}\left[\sum_{t'} \gamma^{t'} R^{(t')} \,|\, \mathbf{s}^{(0)} = \boldsymbol{s}^{(i,t)}, \mathbf{a}^{(0)} = \boldsymbol{a}^{(i,t)}\right]$
  - using samples rolled out from single episode

Variance

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} \,|\, \boldsymbol{s}^{(i,t)}; \Phi) \sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ is an MC estimate of $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)}) = \mathrm{E}_{\{\mathbf{s}^{(t')}, \mathbf{a}^{(t')}\}_{t'}} \left[\sum_{t'} \gamma^{t'} R^{(t')} \,|\, \mathbf{s}^{(0)} = \boldsymbol{s}^{(i,t)}, \mathbf{a}^{(0)} = \boldsymbol{a}^{(i,t)} \right]$
    - using samples rolled out from single episode
- TD vs. MC estimate:
    - TD: biased, but low variance
    - MC: unbiased, but ***high variance***
- How to lower the variance of vanilla policy gradient algorithm?

Variance

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi) \sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ is an MC estimate of $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)}) = \mathrm{E}_{\{\mathbf{s}^{(t')}, \mathbf{a}^{(t')}\}_{t'}} \left[\sum_{t'} \gamma^{t'} R^{(t')} | \mathbf{s}^{(0)} = \boldsymbol{s}^{(i,t)}, \mathbf{a}^{(0)} = \boldsymbol{a}^{(i,t)} \right]$
  - using samples rolled out from single episode
- TD vs. MC estimate:
  - TD: biased, but low variance
  - MC: unbiased, but ***high variance***
- How to lower the variance of vanilla policy gradient algorithm?
  - To reduce the magnitude of $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$
    - Eg., use a smaller γ or subtract a ***baseline*** from $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$
  - To approximate $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ by a DNN and take advantage of its generalizability
  - To collect more samples

Baseline I

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi) (\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b)$$

- b reduces variance without adding bias ***as long as it's independent with actions***:

$$\begin{aligned} & \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)}; \Phi) b \\ & = \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \nabla_\Phi \log \mathrm{P}(\tau;\Phi) b \\ & = \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \frac{\nabla_\Phi \mathrm{P}(\tau;\Phi)}{\mathrm{P}(\tau;\Phi)} b \\ & = \nabla_\Phi \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) b = \nabla_\Phi b = 0 \end{aligned}$$

Baseline I

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi) (\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b)$$

- b reduces variance without adding bias ***as long as it's independent with actions***:

$$\begin{aligned} &\textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \sum_t \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)}; \Phi) b \\ &= \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \nabla_\Phi \log \mathrm{P}(\tau;\Phi) b \\ &= \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) \frac{\nabla_\Phi \mathrm{P}(\tau;\Phi)}{\mathrm{P}(\tau;\Phi)} b \\ &= \nabla_\Phi \textstyle\sum_\tau \mathrm{P}(\tau;\Phi) b = \nabla_\Phi b = 0 \end{aligned}$$

- Of what value?

Baseline II

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi)(\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b)$$

- The larger the b better, but $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b$ still needs to guide g_π to output good τ
- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ an estimate of $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)})$

Baseline II

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi) (\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} {\color{red}-b})$$

- The larger the b better, but $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b$ still needs to guide g_π to output good τ
- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ an estimate of $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)})$
- b an estimate of $V_\pi(\boldsymbol{s}^{(i,t)}) = \mathrm{E}_{\{\mathbf{s}^{(t')}, \mathbf{a}^{(t')}\}_{t'}} \left[\sum_{t'} \gamma^{t'} R^{(t')} | \mathbf{s}^{(0)} = \boldsymbol{s}^{(i,t)} \right]$ [3]
  - $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b$ estimates $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)}) - V_\pi(\boldsymbol{s}^{(i,t)})$, the ***advantage*** of π at state $\boldsymbol{s}^{(i,t)}$
- In REINFORCE: $b = \frac{1}{|\{j,t'' : \boldsymbol{s}^{(j,t'')} = \boldsymbol{s}^{(i,t)}\}|} \sum_{j,t'' : \boldsymbol{s}^{(j,t'')} = \boldsymbol{s}^{(i,t)}} \sum_{t'=t''}^{H^{(j)}} \gamma^{t'} R^{(j,t')}$

Function Approximations

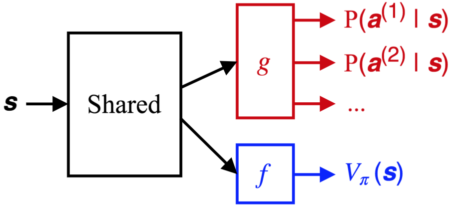
$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi) \left(\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b \right)$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ estimates $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)})$ using rolled-out from single episode
- ***Actor-critic***: why not use a DNN $f_{Q_\pi}(\boldsymbol{s}, \boldsymbol{a}; \Theta)$ to approximate $Q_\pi(\boldsymbol{s}, \boldsymbol{a}), \forall \boldsymbol{s}, \boldsymbol{a}$?

Function Approximations

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)} | \boldsymbol{s}^{(i,t)}; \Phi)\left(\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b\right)$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')}$ estimates $Q_\pi(\boldsymbol{s}^{(i,t)}, \boldsymbol{a}^{(i,t)})$ using rolled-out from single episode
- ***Actor-critic***: why not use a DNN $f_{Q_\pi}(\boldsymbol{s}, \boldsymbol{a}; \Theta)$ to approximate $Q_\pi(\boldsymbol{s}, \boldsymbol{a}), \forall \boldsymbol{s}, \boldsymbol{a}$?
- Baseline $b = \sum_{j:\boldsymbol{s}^{(j,t)}=\boldsymbol{s}^{(i,t)}} \sum_{t'=t}^{H^{(j)}} \gamma^{t'} R^{(j,t')}$ estimates $V_\pi(\boldsymbol{s}^{(i,t)})$
- ***Advantage actor-critic***: approximates $V_\pi(\boldsymbol{s})$ with $f_{V_\pi}(\boldsymbol{s}; \Theta)$



Advantage Actor-Critic ($b = f_{V_\pi}(\boldsymbol{s};\Theta)$)

$$\nabla_\Phi J(\Phi) \propto \sum_{i,t} \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(i,t)}|\boldsymbol{s}^{(i,t)};\Phi)(\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} - b)$$

- $\sum_{t'=t}^{H^{(i)}} \gamma^{t'} R^{(i,t')} \approx Q_\pi(\boldsymbol{s}^{(i,t)},\boldsymbol{a}^{(i,t)})$ can be approximated by $R^{(i,t)} + \gamma f_{V_\pi}(\boldsymbol{s}^{(i,t+1)};\Theta)$
  - No need for f_{Q_π}
- Bellman expectation equation for stochastic π:

$$V_\pi(\boldsymbol{s}) = \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_\pi(\boldsymbol{s}')], \forall \boldsymbol{s}$$

- Algorithm (TD): initialize Θ and Φ arbitrarily, iterate until converge:
  1. Take an action $\boldsymbol{a}$ from $\boldsymbol{s}$ using $g(\boldsymbol{s};\Phi)$
  2. Observe $\boldsymbol{s}'$ and reward R, compute $\hat{Q}_\pi \leftarrow R + \gamma f_{V_\pi}(\boldsymbol{s}';\Theta)$
  3. Update f_{V_π}:
  $$\Theta \leftarrow \Theta - \eta \nabla_\Theta \left[\hat{Q}_\pi - f_{V_\pi}(\boldsymbol{s};\Theta)\right]^2$$
  4. Update g_π:
  $$\Phi \leftarrow \Phi + \lambda \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi)(\hat{Q}_\pi - f_{V_\pi}(\boldsymbol{s};\Theta))$$

Pitfall: Exploration

- To learn f_{V_π} based on value iteration,the agent has to ***explore enough***
- But g_π is optimized for exploitation only:

$$\Phi \leftarrow \Phi + \lambda \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi)(\hat{Q}_\pi - f_{V_\pi}(\boldsymbol{s};\Theta))$$

- Solution?

Pitfall: Exploration

- To learn f_{V_π} based on value iteration,the agent has to ***explore enough***
- But g_π is optimized for exploitation only:

$$\Phi \leftarrow \Phi + \lambda \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi)(\hat{Q}_\pi - f_{V_\pi}(\boldsymbol{s};\Theta))$$

- Solution? To maximize the entropy of $g_\pi(\boldsymbol{s};\Phi)$ as well

$$\begin{aligned}\Phi \quad & \leftarrow \Phi + \lambda \nabla_\Phi \left[\log \mathrm{P}(\boldsymbol{a}|\boldsymbol{s};\Phi)(\hat{Q}_\pi - f_{V_\pi}(\boldsymbol{s};\Theta))\right. \\ & \qquad\qquad\qquad\qquad \left. + \mu \mathrm{H}(\mathbf{a} \sim g_\pi(\boldsymbol{s};\Phi))\right]\end{aligned}$$

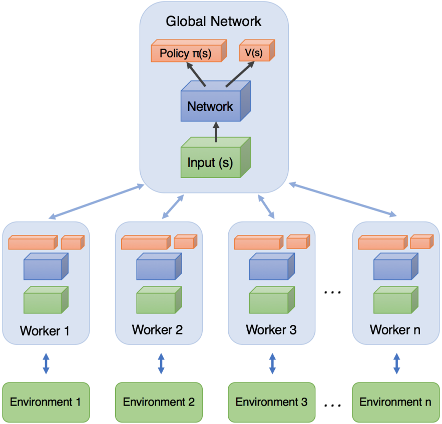
  - The larger μ, the more exploration

Asynchronous Advantage Actor-Critic (A3C)

- TD estimate reduces variance at the cost of bias/divergence

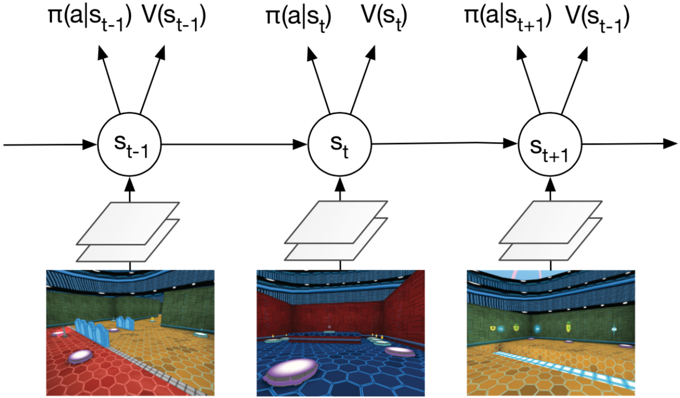
Asynchronous Advantage Actor-Critic (A3C)

- TD estimate reduces variance at the cost of bias/divergence
- A3C: use asynchronous workers to stabilizes f_{V_π} training
  - An alternative to experience reply



A3C on Labyrinth

- Task: to collect apples (+1 reward) and escape (+10 reward)
- End-to-end learning from pixels to policy
- State $\boldsymbol{s}^{(t)}$ modeled as a recurrent neural network (LSTM)
  - To have long-term memory



Variance Reduction by Having More Samples

Variance Reduction by Having More Samples

- Update rule for g_π in A3C:

$$\Phi \leftarrow \Phi + \lambda \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)}; \Phi) \hat{A}_\pi^{(t)}$$

where $\hat{A}_\pi^{(t)} = \hat{Q}_\pi^{(t)} - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta) = (R^{(t)} + \gamma f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)) - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)$

- Bellman expectation equation holds for multiple time differences:

$$\begin{aligned} Q_\pi(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) &= \mathrm{E}[R^{(t)} + \gamma V_\pi(\mathbf{s}^{(t+1)}) | \mathbf{s}^{(t)} = \boldsymbol{s}^{(t)}, \mathbf{a}^{(t)} = \boldsymbol{a}^{(t)}] \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 V_\pi(\mathbf{s}^{(t+2)})] \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 V_\pi(\mathbf{s}^{(t+3)})] \\ &\cdots \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 R^{(t+3)} + \cdots] \end{aligned}$$

- A3C replaces $\hat{A}_\pi^{(t)}$ with a K-step lookahead:

$$\hat{A}_K^{(t)} \leftarrow R^{(t)} + \gamma R^{(t+1)} + \cdots + \gamma^{K-1} R^{(t+K-1)} + \gamma^K f_{V_\pi}(\boldsymbol{s}^{(t+K)}; \Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)$$

Variance Reduction by Having More Samples

- Update rule for g_π in A3C:

$$\Phi \leftarrow \Phi + \lambda \nabla_\Phi \log \mathrm{P}(\boldsymbol{a}^{(t)} | \boldsymbol{s}^{(t)}; \Phi) \hat{A}_\pi^{(t)}$$

where $\hat{A}_\pi^{(t)} = \hat{Q}_\pi^{(t)} - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta) = (R^{(t)} + \gamma f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)) - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)$

- Bellman expectation equation holds for multiple time differences:

$$\begin{aligned} Q_\pi(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) &= \mathrm{E}[R^{(t)} + \gamma V_\pi(\mathbf{s}^{(t+1)}) | \mathbf{s}^{(t)} = \boldsymbol{s}^{(t)}, \mathbf{a}^{(t)} = \boldsymbol{a}^{(t)}] \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 V_\pi(\mathbf{s}^{(t+2)})] \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 V_\pi(\mathbf{s}^{(t+3)})] \\ &\cdots \\ &= \mathrm{E}[R^{(t)} + \gamma R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 R^{(t+3)} + \cdots] \end{aligned}$$

- A3C replaces $\hat{A}_\pi^{(t)}$ with a K-step lookahead:

$$\hat{A}_K^{(t)} \leftarrow R^{(t)} + \gamma R^{(t+1)} + \cdots + \gamma^{K-1} R^{(t+K-1)} + \gamma^K f_{V_\pi}(\boldsymbol{s}^{(t+K)}; \Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)}; \Theta)$$

- Update of Φ lags K time steps behind the current action

Generalized Advantage Estimation (GAE)

$$\hat{A}_K^{(t)} = R^{(t)} + \gamma R^{(t+1)} + \cdots + \gamma^{K-1} R^{(t+K-1)} + \gamma^K f_{V_\pi}(\boldsymbol{s}^{(t+K)};\Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)};\Theta)$$

- Define TD error at time t: $\delta^{(t)} = R^{(t)} + \gamma f_{V_\pi}(\boldsymbol{s}^{(t+1)};\Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)};\Theta)$
- We have $\hat{A}_K^{(t)} = \delta^{(t)} + \gamma\delta^{(t+1)} + \cdots + \gamma^{K-1}\delta^{(t+K-1)}$
- GAE [8]: let $\hat{A}_\pi^{(t)}$ be the exponential moving average of $\hat{A}_1^{(t)}, \hat{A}_2^{(t)}, \cdots$:

$$\begin{aligned}
\hat{A}_\pi^{(t)} & \leftarrow \hat{A}_1^{(t)} + \lambda \hat{A}_2^{(t)} + \lambda^2 \hat{A}_3^{(t)} + \cdots \\
& = \delta^{(t)} + \lambda(\delta^{(t)} + \gamma\delta^{(t+1)}) + \lambda^2(\delta^{(t)} + \gamma\delta^{(t+1)} + \gamma^2\delta^{(t+2)}) + \cdots \\
& = \frac{1}{1-\lambda}\delta^{(t)} + \frac{\lambda\gamma}{1-\lambda}\delta^{(t+1)} + \frac{\lambda^2\gamma^2}{1-\lambda}\delta^{(t+2)} + \cdots \\
& \propto \delta^{(t)} + \lambda\gamma\delta^{(t+1)} + (\lambda\gamma)^2\delta^{(t+2)} + \cdots
\end{aligned}$$

Generalized Advantage Estimation (GAE)

$$\hat{A}_K^{(t)} = R^{(t)} + \gamma R^{(t+1)} + \cdots + \gamma^{K-1} R^{(t+K-1)} + \gamma^K f_{V_\pi}(\boldsymbol{s}^{(t+K)};\Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)};\Theta)$$

- Define TD error at time t: $\delta^{(t)} = R^{(t)} + \gamma f_{V_\pi}(\boldsymbol{s}^{(t+1)};\Theta) - f_{V_\pi}(\boldsymbol{s}^{(t)};\Theta)$
- We have $\hat{A}_K^{(t)} = \delta^{(t)} + \gamma\delta^{(t+1)} + \cdots + \gamma^{K-1}\delta^{(t+K-1)}$
- GAE [8]: let $\hat{A}_\pi^{(t)}$ be the exponential moving average of $\hat{A}_1^{(t)}, \hat{A}_2^{(t)}, \cdots$:

$$\begin{aligned}
\hat{A}_\pi^{(t)} \quad & \leftarrow \hat{A}_1^{(t)} + \lambda \hat{A}_2^{(t)} + \lambda^2 \hat{A}_3^{(t)} + \cdots \\
& = \delta^{(t)} + \lambda(\delta^{(t)} + \gamma\delta^{(t+1)}) + \lambda^2(\delta^{(t)} + \gamma\delta^{(t+1)} + \gamma^2\delta^{(t+2)}) + \cdots \\
& = \frac{1}{1-\lambda}\delta^{(t)} + \frac{\lambda\gamma}{1-\lambda}\delta^{(t+1)} + \frac{\lambda^2\gamma^2}{1-\lambda}\delta^{(t+2)} + \cdots \\
& \propto \delta^{(t)} + \lambda\gamma\delta^{(t+1)} + (\lambda\gamma)^2\delta^{(t+2)} + \cdots
\end{aligned}$$

- Biased, but with much lower variance
  - $\delta^{(t)}$'s can have lower magnitudes when f_{V_π} is good enough
- In TD: $\hat{A}_\pi^{(t)} \leftarrow \delta^{(t)} + \lambda\gamma\delta^{(t+1)} + \cdots + (\lambda\gamma)^K\delta^{(t+K)}$

Policy Optimization vs. Policy/Value Iteration

- Policy optimization:
- Policy/value-iteration-based methods (e.g., DQN, DDPG):

Policy Optimization vs. Policy/Value Iteration

- Policy optimization:
  - Optimize policy "directly"
- Policy/value-iteration-based methods (e.g., DQN, DDPG):
  - Optimize policy "indirectly" (via Q/V exploiting Bellman equations)

Policy Optimization vs. Policy/Value Iteration

- Policy optimization:
  - Optimize policy "directly"
  - More compatible with auxiliary objectives & rich NN architectures (e.g., RNN)
- Policy/value-iteration-based methods (e.g., DQN, DDPG):
  - Optimize policy "indirectly" (via Q/V exploiting Bellman equations)
  - More compatible with different exploration strategies

Policy Optimization vs. Policy/Value Iteration

- Policy optimization:
  - Optimize policy "directly"
  - More compatible with auxiliary objectives & rich NN architectures (e.g., RNN)
  - More likely to work with different tasks/settings
- Policy/value-iteration-based methods (e.g., DQN, DDPG):
  - Optimize policy "indirectly" (via Q/V exploiting Bellman equations)
  - More compatible with different exploration strategies
  - Sensitive to task/settings; but more sample-efficient when working

Reference I

[1] Vivek F Farias and Benjamin Van Roy.
Tetris: A study of randomized constraint sampling.
In *Probabilistic and Randomized Methods for Design Under Uncertainty*, pages 189–201. Springer, 2006.

[2] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al.
Noisy networks for exploration.
arXiv preprint arXiv:1706.10295, 2017.

[3] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter.
Variance reduction techniques for gradient estimates in reinforcement learning.
Journal of Machine Learning Research, 5(Nov):1471–1530, 2004.

Reference II

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra.
Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971, 2015.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al.
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533, 2015.

[6] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al.
Massively parallel methods for deep reinforcement learning.
arXiv preprint arXiv:1507.04296, 2015.

Reference III

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver.
Prioritized experience replay.
arXiv preprint arXiv:1511.05952, 2015.

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel.
High-dimensional continuous control using generalized advantage estimation.
arXiv preprint arXiv:1506.02438, 2015.

[9] Hado Van Hasselt, Arthur Guez, and David Silver.
Deep reinforcement learning with double q-learning.
In *AAAI*, pages 2094–2100, 2016.

[10] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas.
Dueling network architectures for deep reinforcement learning.
arXiv preprint arXiv:1511.06581, 2015.

Reference IV