Reinforcement Learning

Deep Deterministic Policy Gradient

U Kang
Seoul National University

In This Lecture

- Deterministic Policy Gradient
- Deep Deterministic Policy Gradient

Outline

- **Deterministic Policy Gradient**
- Deep Q-Network
- Deep Deterministic Policy Gradient
- Conclusion

Policy Approximation and its Advantages

- In policy gradient (PG) methods, the policy can be parameterized in any way, as long as $\pi(a|s,\theta)$ is differentiable with respect to its parameters θ; i.e., $\nabla_\theta \pi(a|s,\theta)$ exists and is finite

Policy Approximation and its Advantages

- If the action space is discrete and not too large, then a natural and common kind of parameterization is to form parameterized numerical preferences $h(s, a, \theta) \in R$ for each state–action pair
- The actions with the highest preferences in each state are given the highest probabilities of being selected, for example, according to an exponential soft-max distribution:

$$\pi(a|s, \theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$

- The action preferences $h(s, a, \theta)$ can be parameterized arbitrarily
  - E.g., ANN (as in AlphaGo)
  - E.g., linear: $h(s, a, \theta) = \theta^T x(s, a)$, where $x(s, a) \in R^{d'}$ is a feature vector

Stochastic Policy Gradient

- Performance objective

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi_\theta(s,a) r(s,a) \mathrm{d}a \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_\theta}\left[r(s,a)\right]$$

r(s,a): return

Stochastic Policy Gradient

- Stochastic policy gradient theorem

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]$$

Deterministic Policy Gradient

- Goal: learn deterministic policy
  - $a = \mu_\theta(s)$
- Advantage of deterministic policy
  - In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions

Deterministic Policy Gradient

- Performance objective

$$J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) r(s, \mu_\theta(s)) \mathrm{d}s$$
$$= \mathbb{E}_{s \sim \rho^\mu} \left[r(s, \mu_\theta(s)) \right]$$

$\rho^\mu(s)$: state distribution
$\mu_\theta(s)$: deterministic policy (S -> A)
$r(s, a)$: return

Deterministic Policy Gradient

- Deterministic policy gradient theorem

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \mathrm{d}s$$

$$= \mathbb{E}_{s\sim\rho^\mu} \left[\nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \right]$$

n: # of policy parameters (=$|\theta|$)
m: # of actions
$\mu_\theta(s)$: function from state -> R^m
$Q^\mu(s,a)$: function from state, action -> R
$\nabla_\theta \mu_\theta(s)$: matrix of size $R^{n\times m}$
$\nabla_a Q^\mu(s,a)$: vector of size R^m

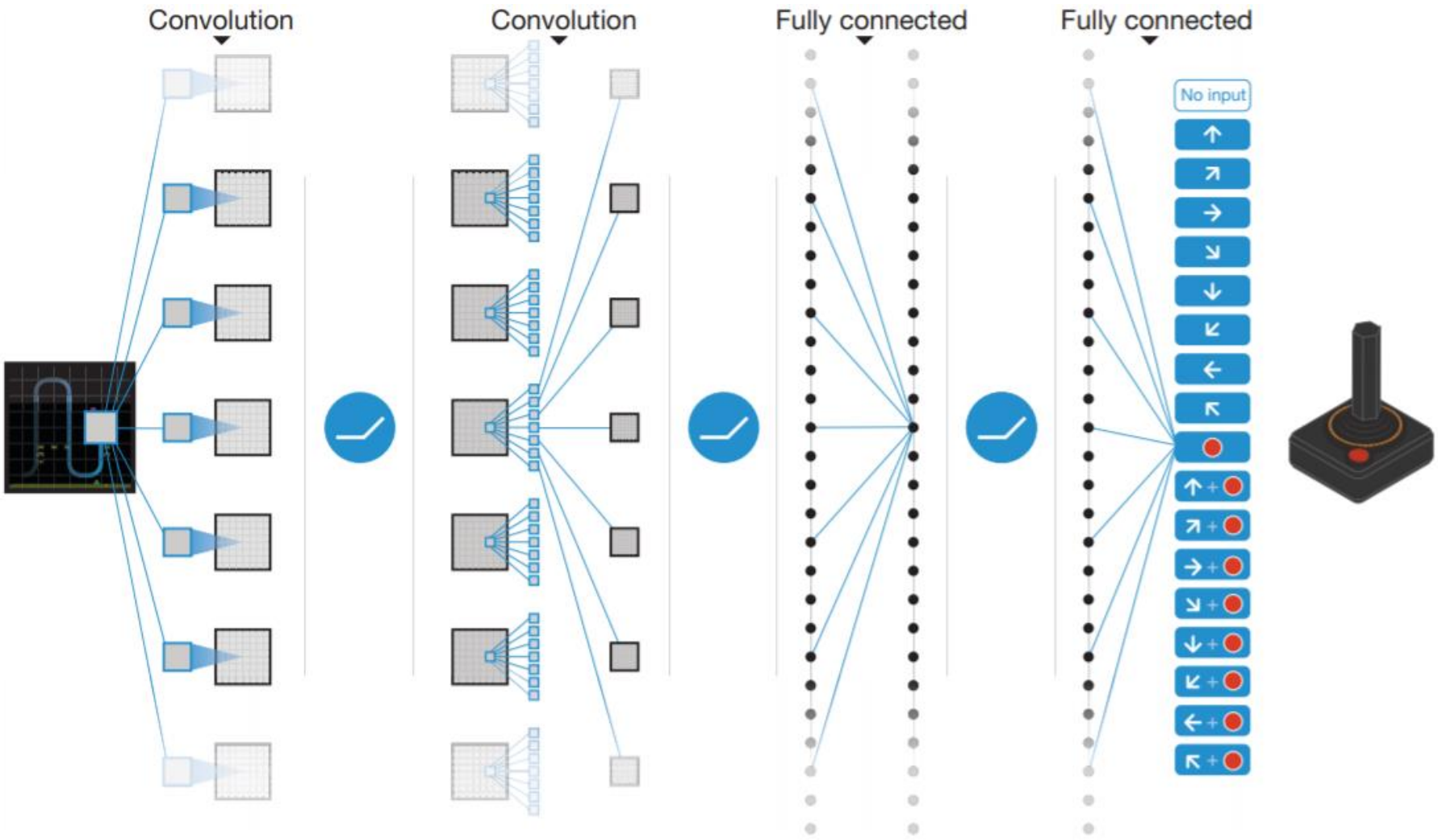
Outline

- ☑ Deterministic Policy Gradient
- ➡ ☐ **Deep Q-Network**
- ☐ Deep Deterministic Policy Gradient
- ☐ Conclusion

Human-level Video Game Play

- The basic architecture of DQN is similar to the deep convolutional ANN
- DQN has three hidden convolutional layers, followed by one fully connected hidden layer, followed by the output layer
- The three successive hidden convolutional layers of DQN produce 32 20 x 20 feature maps, 64 9 x 9 feature maps, and 64 7 x 7 feature maps
- The activation function of the units of each feature map is a rectifier nonlinearity (max(0, x))
- The 3,136 (64 x 7 x 7) units in this third convolutional layer all connect to each of 512 units in the fully connected hidden layer, which then each connect to all 18 units in the output layer, one for each possible action in an Atari game

Human-level Video Game Play



https://www.nature.com/articles/nature14236

Human-level Video Game Play

- The activation levels of DQN's output units were the estimated optimal action values of the corresponding state–action pairs, for the state represented by the network's input
- The assignment of output units to a game's actions varied from game to game, and because the number of valid actions varied between 4 and 18 for the games, not all output units had functional roles in all of the games
- It helps to think of the network as if it were 18 separate networks, one for estimating the optimal action value of each possible action
- In reality, these networks shared their initial layers, but the output units learned to use the features extracted by these layers in different ways

Human-level Video Game Play

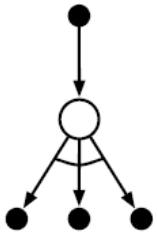
- DQN's reward signal indicated how a game's score changed from one time step to the next: +1 whenever it increased, −1 whenever it decreased, and 0 otherwise
- This standardized the reward signal across the games and made a single step-size parameter work well for all the games
- DQN used an ϵ-greedy policy, with ϵ decreasing linearly over the first million frames and remaining at a low value for the rest of the learning session

Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal



Q-learning

Sutton and Barto, Reinforcement Learning, 2018

Human-level Video Game Play

- After DQN selected an action, the action was executed by the game emulator, which returned a reward and the next video frame
- The frame was preprocessed and added to the four-frame stack that became the next input to the network
- DQN used the semi-gradient form of Q-learning to update the weights:

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{q}(S_t, A_t, w_t)$$

- The gradient was computed by backpropagation
- Mnih et al. used a *mini-batch* method that updated weights only after accumulating gradient information over a small batch of images (here after 32 images)
- This yielded smoother sample gradients compared to the usual procedure that updates weights after each action

Human-level Video Game Play

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

https://arxiv.org/pdf/1312.5602.pdf

Human-level Video Game Play

- Mnih et al. modified the basic Q-learning procedure in three ways
- 1) Experience replay
  - Store the agent's experience at each time step in a replay memory that is accessed to perform the weight updates
  - After the game emulator executed action A_t in a state represented by the image stack S_t, and returned reward R_{t+1} and image stack S_{t+1}, it added the tuple (S_t, A_t, R_{t+1}, S_{t+1}) to the replay memory
  - This memory accumulated experiences over many plays of the same game
  - At each time step multiple Q-learning updates (a mini-batch) were performed based on experiences sampled uniformly at random from the replay memory

Human-level Video Game Play

- 1) Experience replay
  - Q-learning with experience replay provided several advantages over the usual form of Q-learning
  - The ability to use each stored experience for many updates allowed DQN to learn more efficiently from its experiences
  - Experience replay reduced the variance of the updates because successive updates were not correlated with one another as they would be with standard Q-learning

Human-level Video Game Play

- 2) Fixed target (for stable learning)
  - As in other methods that bootstrap, the target for a Q-learning update depends on the current action-value function estimate
  - Its dependence on w_t complicates the process compared to the simpler supervised-learning situation in which the targets do not depend on the parameters being updated
  - Solution: whenever a certain number, C, of updates had been done to the weights w of the action-value network, they inserted the network's current weights into another network and held these duplicate weights fixed for the next C updates of w
  - The outputs of this duplicate network over the next C updates of w were used as the Q-learning targets
  - Letting $\tilde{q}$ denote the output of this duplicate network, the update rule was:

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{\mathrm{q}}(S_t, A_t, w_t)$$

Human-level Video Game Play

- 3) Error clipping
  - Goal: to improve stability
  - Clipped the error term

$$R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t)$$

  - so that it remained in the interval [−1, 1]
  - Reminder: the parameter is updated by

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{\mathrm{q}}(S_t, A_t, w_t)$$

Outline

- ☑ Deterministic Policy Gradient
- ☑ Deep Q-Network
- ➡ ☐ **Deep Deterministic Policy Gradient**
- ☐ Conclusion

Deep Deterministic Policy Gradient

- DDPG
  - Combines DPG and DQN
  - DDPG is based on DPG, thus, it uses the deterministic policy gradient to update the policy parameter
  - In addition, DDPG updates the action-value function Q(s,a) with DQN
  - Since DDPG updates both policy and value functions, it is an actor-critic method

Deep Deterministic Policy Gradient

- Actor
  - Updates the policy using the deterministic policy gradient

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\mu} \left[\nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \right]$$

  - Instead of the expectation, DDPG uses samples from the replay memory (as in DQN)

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

Deep Deterministic Policy Gradient

- Critic
  - Updates the value function as in DQN
  - However, unlike DQN, the update target slowly tracks the learned networks

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 target networks:

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

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

Deep Deterministic Policy Gradient

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

Outline

- ☑ Deterministic Policy Gradient
- ☑ Deep Q-Network
- ☑ Deep Deterministic Policy Gradient
- ☐ **Conclusion**

Conclusion

- Deterministic policy gradient is useful for high-dimensional action space
- Deep deterministic policy gradient combines DPG with DQN, and enables rich neural networks to learn value and policy functions

Questions?