Deep Deterministic Policy Gradient

Christopher Mutschler

Deep Deterministic Policy Gradient

Question: can we also use ideas from value-based RL for continuous action spaces?

- Why can't we use Q-Learning and DQNs?

→ Not so easily!

But what was the original problem with Q-Learning?

- Q-Learning and variants (including DQNs) do not work with continuous actions
- Why is that? Remember:
  - We calculated the targets $r_t + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \theta_{i-1})$ by a single pass through the network
  - Our network was "static" and had $|\mathcal{A}|$ outputs

- Evaluating a continuous action space requires an exhaustive search over the available actions (and this becomes highly non-trivial!)[1]

[1] However, see approaches such as Lim et al.: Actor-Expert: A Framework for using Q-learning in Continuous Action Spaces.

Deep Deterministic Policy Gradient (DDPG)

- DDPG learns a Q-function and a policy
  - Off-policy data + Bellman equation to learn Q-function
  - Make use of the Q-function to learn the policy

- The intuition relies on Q-learning:
  - If you know $Q^*(s, a)$ then in each state the optimal action $a^*(s)$ can be *simply* found by

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

  - DDPG jointly learns approximations to $Q^*(s, a)$ and $a^*(s)$

    …and specifically adapts this for continuous action spaces!

Deep Deterministic Policy Gradient (DDPG)

Main idea:

- We assume the function $Q^*(s, a)$ to be differentiable with respect to a
- This allows for a gradient-based learning rule for a policy $\mu(s)$ that exploits this

- Instead of exhaustively looking for $\max_a Q(s, a)$ we approximate it:

$$\max_a Q(s, a) \approx Q(s, \mu(s))$$

- We look at two sides of DDPG:
  1. Its Q-Learning side
  2. Its policy gradient side

Deep Deterministic Policy Gradient (DDPG)

The Q-Learning side of DDPG

- Recap the Bellman optimality equation:

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

- Then we can optimize the mean-squared Bellman error (MSBE) (neural network parameters ϕ, set of transitions D, $d \in \{0;1\}$ indicates if s' is terminal):

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

- For optimization using SGD we apply the well-known tricks:
  - Replay buffers (off-policy!)
  - Use target networks and update it with delay by $\phi_{targ} \leftarrow p\phi_{targ} + (1-p)\phi, \;\; p \in [0;1]$

Deep Deterministic Policy Gradient (DDPG)

The Q-Learning side of DDPG

- Calculating the max over the actions in the target:
  - Target policy network μ_θ computes an action that approximately maximizes $Q_{\phi_{targ}}$
  - Target policy updates also computed using polyak averaging (see above)
  - Q-Learning in DDPG minimizes using SGD:

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

Deep Deterministic Policy Gradient (DDPG)

The Policy Learning side of DDPG: simple

- Policy $\mu_\theta(s)$ is deterministic
- $\mu_\theta(s)$ should return the action that maximizes $Q_\phi(s, a)$
- Action space is continuous, and we assume Q to be differentiable with respect to actions a

- Hence, we can use gradient ascent (with respect to the policy parameters θ only) and solve:

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

(Q-function parameters ϕ are treated as constants here)

Deep Deterministic Policy Gradient (DDPG)

Exploration-Exploitation:

- DDPG trains off-policy; policy is deterministic
- Hence an on-policy exploration is often not enough
- Solution: add noise to the actions at training time
  - Originally time-correlated Ornstein-Uhlenbeck (OU) process noise has been proposed
  - More recent work suggests to use zero-mean Gaussian noise as it is simpler and exhibits same performance
  - Over the course of training, we may reduce the scale of noise (but there is only a limited effect from this)
  - At test time we (of course) omit to add the noise

Deep Deterministic Policy Gradient (DDPG)

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

Use target networks (as in DQN) for both the actor and the critic

As we don't have a stochastic policy, we have to define a process for exploration

Execute action, store transition in the replay buffer and sample at random some transitions (exactly as in DQN)

Update critic using the target networks for both the Actor and the Critic

Update the Actor using the Deterministic Policy Gradient Theorem (Silver et al. 2014)

Progressively update the target networks

Deep Deterministic Policy Gradient (DDPG)

Summary:

- Special case in actor-critic-algorithms:
- Works only for continuous action spaces
- Is an off-policy algorithm utilizing the replay buffer trick from DQN
- Solves for deterministic policies instead of stochastic ones

+ impressive results both in simulation and in real world problems
+ one of the de-facto algorithms to use for continuous (or very large) action spaces

- very sensitive to the exploration process
- can be hard to tune – sensitive to hyperparameters
- slower (wall clock time) compared to other actor-critic-algorithms