Deterministic Policy Gradients

Reading

- Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms." In *International conference on machine learning*, pp. 387-395. PMLR, 2014.
- Lillicrap, Timothy P., 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).
- Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In International conference on machine learning, pp. 1587-1596. PMLR, 2018.

Overview



- Q-learning works well for systems with finitely many actions
- However, most real-world systems have infinitely many control actions
 - One could discretize the action space and still use Qlearning
 - Unlikely to scale if we have more than a couple of dimensions to discretize over
- Need to extend Q-learning methods to infinite-state MDPs

Extending Q-learning to infinite action spaces

- First, suppose we know the optimal Q function

 in an infinite action space
- Can compute the policy π by maximizing Q over all actions: $\pi(s) = \max_{a} Q(s, a)$
 - May be hard if Q is a complex function (e.g., a neural net)
 - But would probably find a good local optimum eventually
- Of course, we don't know the optimal ${\cal Q}$

– Need to iteratively update Q and π

- Could apply the Q-learning iteration followed by the maximization above
 - Research in the past showed that this approach is quite unstable



• Standard policy gradient theorem is very inefficient (why?)

$$\nabla v_{\pi}(s_0) = \sum_{s} d_{\pi}(s) \sum_{a} \nabla \pi(a|s) q_{\pi}(s,a)$$

- Need to know/approximate q_{π} for all actions
- Hard to do in an infinite space
- One can avoid the expectation over all actions by introducing a deterministic policy gradient
- For a deterministic policy, there is only one action per state — Most policies are deterministic anyway, e.g., neural nets

¹David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms." In *International conference on machine learning*, pp. 387-395. PMLR, 2014.



• Standard policy gradient theorem is very inefficient

$$\nabla v_{\pi}(s_0) = \sum_{s} d_{\pi}(s) \sum_{a} \nabla \pi(a|s) q_{\pi}(s,a)$$

- However, stochastic policies have a crucial benefit — Exploration!
- If target policy is deterministic, need a stochastic behavior policy — And an off-policy algorithm!

¹David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms." In *International conference on machine learning*, pp. 387-395. PMLR, 2014.



• What would be different from the stochastic version:

$$\nabla_{\boldsymbol{\theta}} v_{\pi}(s_0) = \sum_{s} d_{\pi}(s) \sum_{a} \nabla_{\boldsymbol{\theta}} \pi(a|s) q_{\pi}(s,a)$$

- No need to average over all actions any more
- Can evaluate gradient at the specific action taken by the policy
- Deterministic policy gradient theorem:

$$\nabla_{\boldsymbol{\theta}} v_{\pi}(s_0) = \int_{S} d_{\pi}(s) \nabla_{\boldsymbol{\theta}} \pi(s; \boldsymbol{\theta}) \nabla_{a} q_{\pi}(s, a) \Big|_{a=\pi(s)} ds$$

- Policy gradients for deterministic policies only make sense over continuous spaces (why?)
- The q-value function is not differentiable w.r.t. a otherwise

Continuous-space Markov Decision Processes



- Before we look at the deterministic policy gradient proof, we need to talk about continuous-space MDPs
- How do we define an infinite-state MDP?
 - It is a 5-tuple (S, A, P, R, η) as before
 - Now S and A are infinite
 - Infinite spaces can be tricky but a standard choice is to use a probability density function (pdf)
 - The transition function is described with a pdf p(s'|s, a) $\mathbb{E}[S_{t+1}|S_t = s, A_t = a] = \int s' p(s'|s, a) ds'$
 - Recall that for any pdf $\int p(s'|s, a)ds' = 1$
 - The reward function R is defined similarly $R_e(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \int R(s', s, a)p(s'|s, a)ds'$
 - The initial condition η is also a pdf



• Turns out the Bellman equation is the same for continuousstate MDPs as it is for finite-state ones

$$\begin{aligned} q_{\pi}(s,a) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma \nu(S_{t+1})|S_t = s, A_t = a] \\ &= \int \big(R(s',a,s) + \gamma \nu(s')\big) p(s'|s,a) ds' \end{aligned}$$

- Don't have time to prove
- If π is deterministic, what is a?

$$-\operatorname{It's just} \pi(s)$$
$$v_{\pi}(s) = q_{\pi}(s, \pi(s)) = \int (R(s', \pi(s), s) + \gamma v(s')) p(s'|s, \pi(s)) ds'$$

Deterministic Policy Gradient Theorem Proof



Fairly similar to the stochastic case, with some differences

$$\nabla_{\theta} v_{\pi}(s) = \nabla_{\theta} [q_{\pi}(s, \pi(s))] = \nabla_{\theta} \int_{S} p(s'|s, \pi(s)) [R(s', \pi(s), s) + \gamma v_{\pi}(s')] ds' \quad \text{chain rule}$$

$$= \int_{S} \nabla_{\theta} \pi(s) \nabla_{a} [p(s'|s, a) R(s', a, s)] \Big|_{a=\pi(s)} ds' + \nabla_{\theta} \int_{S} p(s'|s, \pi(s)) \gamma v_{\pi}(s') ds'$$

$$= \nabla_{\theta} \pi(s) \nabla_{a} R_{e}(s, a) \Big|_{a=\pi(s)} + \nabla_{\theta} \int_{S} p(s'|s, \pi(s)) \gamma v_{\pi}(s') ds'$$

$$= \nabla_{\theta} \pi(s) \nabla_{a} R_{e}(s, a) \Big|_{a=\pi(s)} + \sum_{\substack{\text{Leibniz rule } + \\ \text{chain rule}}} \sum_{\substack{n=\pi(s) \\ n=\pi(s)}} p(s'|s, \pi(s)) \nabla_{\theta} v_{\pi}(s') + \nabla_{\theta} \pi(s) \nabla_{a} p(s'|s, a) \Big|_{a=\pi(s)} v_{\pi}(s') ds'$$

$$= \nabla_{\theta} \pi(s) \nabla_{a} \left(R_{e}(s, a) + \gamma \int_{S} p(s'|s, a) v_{\pi}(s') ds' \right) \Big|_{a=\pi(s)} + \sum_{\substack{n=\pi(s) \\ \gamma = \pi(s) \\ \gamma = \pi(s)$$

Deterministic Policy Gradient Theorem Proof, cont'd



- Does this look familiar?
 - Same sequence as stochastic policy gradient theorem

Deterministic Policy Gradient Theorem Proof, cont'd



$$\begin{aligned} \nabla_{\theta} v_{\pi}(s) &= \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} + \\ &+ \gamma \int_{S} \mathbb{P}_{\pi} [s \to s', 1] \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} ds' \\ &+ \gamma^{2} \int_{S} \mathbb{P}_{\pi} [s \to s', 2] \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} ds' \\ &+ \cdots \\ &= \sum_{t=0}^{\infty} \int_{S} \gamma^{t} \mathbb{P}_{\pi} [s \to s', t] \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} ds' \\ &= \int_{S} \sum_{t=0}^{\infty} \gamma^{t} \mathbb{P}_{\pi} [s \to s', t] \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} ds' \end{aligned}$$

Deterministic Policy Gradient Theorem Proof, cont'd



$$\nabla_{\theta} v_{\pi}(s) = \int_{S} \sum_{t=0}^{\infty} \gamma^{t} \mathbb{P}_{\pi}[s \to s', t] \nabla_{\theta} \pi(s) \nabla_{a} q(s, a) \Big|_{a=\pi(s)} ds'$$

- Similar to the stochastic case, we have a discounted aggregate visitation function $d_\pi(s)$
 - Now it is a pdf, not a probability function, since we have a continuous space

$$d_{\pi}(s') = \sum_{t=0}^{\infty} \gamma^{t} \mathbb{P}_{\pi}[s \to s', t]$$

-where

 $\mathbb{P}_{\pi}[s \to s', t] = \int p(s_1 | s, \pi(s)) \int p(s_2 | s_1, \pi(s_1)) \dots \int p(s' | s_{t-1}, \pi(s_{t-1})) ds_{t-1} \dots ds_2 ds_1$

• Plugging d_{π} back in gives the final result

$$\nabla_{\boldsymbol{\theta}} v_{\pi}(s_0) = \int_{S} d_{\pi}(s) \nabla_{\boldsymbol{\theta}} \pi(s; \boldsymbol{\theta}) \nabla_{a} q_{\pi}(s, a) \Big|_{a=\pi(s)} ds$$

Off-Policy Deterministic Actor-Critic



- Computing the deterministic gradient is easier
 So far, so good
- But learning using a deterministic policy is harder
 Not much exploration with a deterministic policy
- Need a behavior policy *b*
 - And an off-policy learning approach
 - What is the challenge with such an approach?
 - Data is generated from b, need to adapt the policy gradient theorem



 Since the data is generated from b we need to consider the quantity

$$J(\boldsymbol{\theta}) = \mathbb{E}_b \big[v_{\pi_{\boldsymbol{\theta}}}(S_t) \big]$$

• In the finite case the expectation is expanded as

$$J(\boldsymbol{\theta}) = \sum_{s} d_{b}(s) v_{\pi_{\boldsymbol{\theta}}}(s)$$
$$= \sum_{s} d_{b}(s) \sum_{a} \pi(a|s) q_{\pi_{\boldsymbol{\theta}}}(s, a)$$

- where d_b is the discounted aggregate visitation probability

- We want to pick $\boldsymbol{\theta}$ that maximizes $J(\boldsymbol{\theta})$
 - As usual, we'll use gradient ascent

Degris, Thomas, Martha White, and Richard S. Sutton. "Off-policy actor-critic." arXiv preprint arXiv:1205.4839 (2012).

Stochastic Off-Policy Actor-Critic, cont'd

• In the finite case the expectation is expanded as

$$J(\boldsymbol{\theta}) = \sum_{s} d_{b}(s) \sum_{a} \pi(a|s) q_{\pi_{\boldsymbol{\theta}}}(s,a)$$

• Look at the gradient (using chain rule)

$$\nabla_{\theta} J(\theta) = \sum_{s} d_{b}(s) \sum_{a} \nabla_{\theta} \pi(a|s) q_{\pi_{\theta}}(s,a) + \sum_{s} d_{b}(s) \sum_{a} \pi(a|s) \nabla_{\theta} q_{\pi_{\theta}}(s,a)$$

- Paper argues that minimizing first term is good enough
 - Shows proof in case of finite-state case
 - Infinite-case proof/claim has some issues

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \sum_{s} d_{b}(s) \sum_{a} \nabla_{\boldsymbol{\theta}} \pi(a|s) q_{\pi_{\boldsymbol{\theta}}}(s,a)$$

Degris, Thomas, Martha White, and Richard S. Sutton. "Off-policy actor-critic." arXiv preprint arXiv:1205.4839 (2012).



Off-Policy Deterministic Actor-Critic, cont'd



• Deterministic paper uses the same off-policy gradient approximation as in Sutton paper

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \int_{S} d_{b}(s) \nabla_{\boldsymbol{\theta}} \pi(s; \boldsymbol{\theta}) \nabla_{a} q_{\pi}(s, a) \Big|_{a=\pi(s)} ds$$

 Approximation quality is not clear but ultimately with modern RL algorithms the proof is in the pudding

- Crucially, this approximation means that one can apply the onpolicy algorithm in the off-policy setting
 - This seems suspicious, but if the approximation is good enough locally, then algorithm may work well
 - Given small enough learning rate and other stabilization techniques used in modern RL

🕲 Rensselaer

 Since trajectories are generated by behavior policy b, we can't train the critic on-policy

$$\delta_t = R_t + \gamma Q^{w}(S_{t+1}, A_{t+1}) - Q^{w}(S_t, A_t)$$

- The A_{t+1} would have been generated by b

- How did we modify TD-learning to make it off-policy?
 - Q-learning!
 - -Can't compute max anymore. What is the alternative? $\delta_t = R_t + \gamma Q^w (S_{t+1}, \pi(S_{t+1})) - Q^w (S_t, A_t)$
- The rest of the updates are the same as in the on-policy case
 $$\begin{split} \mathbf{w}' &= \mathbf{w} + \alpha_{\mathbf{w}} \delta_t \nabla_{\mathbf{w}} Q^{\mathbf{w}}(S_t, A_t) \\ \mathbf{\theta}' &= \mathbf{\theta} + \alpha_{\mathbf{\theta}} \nabla_{\mathbf{\theta}} \pi_{\mathbf{\theta}}(s) \nabla_a Q^{\mathbf{w}}(s, a) \Big|_{a = \pi(s)} \end{split}$$



 Lillicrap, Timothy P., 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).

Overview

- Combine classical actor-critic ideas with new insights from the DQN paper
- Use neural networks for both the actor and critic and update them using (deterministic) policy gradients
- Stabilize learning with recent insights such as experience replay and target networks (explained later)
- Also add batch normalization
- Method called deep deterministic policy gradient (DDPG)
 One of the early breakthroughs in continuous-action deep RL



- Environment is an MDP, as usual
 - No finite-state assumption necessary
 - Different environments considered, ranging from standard control tasks to RL benchmarks such as the pendulum
- Action space is continuous (possibly multidimensional)
- Observe a reward R_t at each time t
- The goal is to learn a (deterministic) policy π that maps the current state to a control action, maximizing the expected (discounted) reward

Training the Critic



- Critic is a neural network, trained using supervised learning
- Recall the iteration
 - After observing a tuple $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$, compute the difference $\delta_t = R_t + \gamma Q^w (S_{t+1}, \pi(S_{t+1})) Q^w (S_t, A_t)$

-Let
$$y = R_t + \gamma Q^w (S_{t+1}, \pi(S_{t+1}))$$
. Then minimize loss:
 $L_w(Q^w, S_t, A_t, y) = (y - Q^w(S_t, A_t))^2$

-i.e., follow gradient
$$2\delta_t \nabla_w Q^w(S_t, A_t)$$
:
 $w' = w + \alpha_w \delta_t \nabla_w Q^w(S_t, A_t)$

- Can train directly using gradient ascent
- Note that this is off-policy since data comes from past behavior policies in the replay buffer

Training the Actor



- The actor is a neural network that takes in the observation/state and outputs the control action
- Unlike DQN, this network is not a classifier, but a regressor
- Output layer has one neuron (for each action dimension)
- Trained using the deterministic policy gradient $\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \pi(S_t) \nabla_a Q^{\boldsymbol{w}}(S_t, a) \Big|_{a = \pi(S_t)}$

- Recall the idea in double Q-learning
 - Use two Q estimators, one to pick the maximal action and one to estimate the bootstrapped return
 - Reduces maximization bias
- Normally, Q^w and π_{θ} networks used to bootstrap the labels y $y = R_t + \gamma Q^w (S_{t+1}, \pi_{\theta}(S_{t+1}))$
 - This adds a lot of variance and slows down training ultimately
- Instead, we could have separate "target" networks $Q^{w'}$ and $\pi_{\theta'}$ to bootstrap the labels
 - We update these more slowly than the ones we train

Target Networks, cont'd

• Target $\boldsymbol{\theta}_t$ updated as a weighted average of previous target parameters and trained parameters $\boldsymbol{\theta}$

$$\boldsymbol{\theta}_t' = \tau \boldsymbol{\theta} + (1 - \tau) \boldsymbol{\theta}_t$$

- where au is close to 0
- Same for \boldsymbol{w} and \boldsymbol{w}'
- This process greatly stabilizes training
 - It may make it slower in some cases but the benefits in stability outweigh the cost in noise without targets
 - It is a low-pass filter of sorts



Exploration

- A common and simple way to enforce exploration is to add noise to the actions
- The choice of noise is essential because sometimes we need to add similar noise over several steps (e.g., to force a turn)
 - If we add random noise at every step, may not explore enough
- One popular choice is the Ornstein-Uhlenbeck (OU) process $\dot{x}(t) = -\theta x(t) + \sigma \eta(t)$
 - where θ and σ are parameters and η is Gaussian noise
 - this is effectively a random walk, where θ and σ determine the mean and variance, respectively
 - correlation over time ensures that non-trivial actions can be explored over time

Full DDPG 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 Rfor 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 RSample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from RSet $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ 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:

$$\begin{split} \theta^{Q'} &\leftarrow \tau \theta^Q + (1-\tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1-\tau) \theta^{\mu'} \end{split}$$

end for end for

Experimental environments



- Authors looked at a number of control tasks
- Classical benchmarks such as cartpole, pendulum
- MuJoCo environment tasks
 cheetah, monoped, locomotion tasks
- Torcs (driving simulator)



Results

Rensselaer

- DDPG much more stable than pure DPG on almost all tasks
 Target networks and batch norm seem to be a good combo
- Can also learn fairly good policies directly from pixels (blue)



Light grey: original DPG Dark grey: DPG + batch norm Green: DPG + batch norm + target networks (DDPG) Blue: raw pixel data

Accuracy of estimated Q-values



- Estimates reasonably accurate for simpler tasks
- Overestimation issues for complex tasks (e.g., cheetah)
- This is a known problem in all Q-learning-like methods
 - The overapproximation bias has returned!





 Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In International conference on machine learning, pp. 1587-1596.
 PMLR, 2018.

- Value function overestimation is a common problem in Qlearning
 - Already saw maximization bias in standard Q-learning
- It can degrade the quality of learning in several ways
 - Reduce exploration when a suboptimal action has a greatly overestimated value
 - Increase variance if several suboptimal actions have overestimated values
- Overall, learning is significantly slowed down
 - May not even converge in some cases

Correcting Overestimation Bias



- Target networks in the DDPG paper are an analogue of double Q-learning
 - But not quite double Q-learning since the networks change slowly – target and behavior nets are often similar
 - But true double Q-learning can't be applied to the deep RL setting exactly (why?)
 - Critics are trained on the same buffer (not independent)
- Instead train two separate critics and use the critic that provides the lower value

$$y = r + \gamma \min_{i=1,2} Q_{w_i}(s, \pi(s))$$

- Critics are trained in the same way as before (just initialized with different weights)
- Alleviates the overestimation error

Delayed Policy Updates



- Do not perform the policy gradient every time but rather every *T* iterations (where *T* is typically several hundred)
 - Same for target network updates
- What is the benefit of this approach?
 - Policy gradients are really expected values
 - Delaying allows us to collect more data from the current policy
 - Obtain a better estimate of the expectation
 - Also, effectively makes training more on-policy
 - Since we're effectively using the on-policy gradient, this ought to help as well

Results

- Training is much more stable and able to find better policies in most cases
- TD3 is my algorithm of choice for continuous-action RL

