Reinforcement Learning Manual

Deep Deterministic Policy Gradient

- Paper: Continuous control with deep reinforcement learning (2015)
- DDPG uses 4 neural networks
    - Non-target Critic
        - Q(s, a | theta^Q)
        - Weights: θ^{Q}
    - Non-target Actor
        - mu(s | theta^mu)
        - Weights: θ^{μ}
    - Target Critic
        - Q'
    - Target Actor
        - μ'
- Non-target Critic update
    - $y_i = r_i + \gamma Q'(s_{i+1},\ \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    - Target is a reward plus discounted action-value from target Critic at the next state and the action from target Actor using the next state. The target Critic uses the weights from the Critic network, and the target Actor uses the weights from the Actor network. Here, the target Critic and the Critic network are the separate network.
- Non-target Actor update
    - $\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,\ a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$
    - Policy gradient with non-target Actor weights is estimated by a mean of gradients of action-values Q(s, a) from the sampled states and the actions from the gradients of non-target Actor from the sampled states.
- Target Critic and target Actor update
    - Both are updated by tau weighted average as follows. In practice, we use a very small tau value, so it means that target network weights are not hugely impacted by non-target network in each update, so the update of target networks is slow.
    - Target Critic
        - $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
    - Target Actor
        - $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$

Policy Gradient

- Policy gradient is to update policy by taking gradient of policy and taking gradient ascent to maximize expectation of rewards
- In practice, the actor-critic algorithm is used. It means that actor is updated by policy gradient, and critic is updated by TD error.
    - Actor
        - theta = theta + learning_rate (alpha) * gradient of log policy * action-value
    - Critic
        - delta = reward + gamma * TD error
        - w = w + learning_rate (beta) * delta * feature(s, a)
- A(s, a) = Q(s, a) - V(s)
    - A(s, a) is advantage function. It tells us how much better than usual to take a particular action a.
    - Q(s, a) is action-value function
    - V(s) is state-value function
- Variety of policy gradient algorithms
    - REINFORCE
        -
    - Actor-Critic
    - Off-policy Policy Gradient
    - A3C
    - DPG
    - DDPG
- https://www.youtube.com/watch?v=KHZVXao4qXs
- https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d
- https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Generalized Advantage Estimation (GAE)

- The generalized advantage estimator, $GAE(\gamma, \lambda)$, is a discounted sum of Bellman residual terms as followed,
    - $\widehat{A}_t^{GAE(\gamma,\lambda)} := \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$
    - It's estimating the advantage function, not the value function.
- This GAE produces a biased estimator and the discounted policy gradient g^γ as follows,
    - $g^\gamma =$
- The value function is optimized by a trust region method.
- Goal is to reduce variance of policy gradients.

- Parameters
  - γ
    - A variance reduction parameter
    - $\gamma \in [0, 1]$
  - λ
    - A compromise between bias and variance.
    - $\lambda \in [0, 1]$
  - Empirically, the best value of lambda is much lower than the best value of gamma.
- Github
  - https://github.com/bsivanantham/GAE
  - https://github.com/Anjum48/rl-examples/tree/56aca982fcf4426c02aa7e5fb58a4f8affab8020
- Video
  - https://sites.google.com/site/gaepapersupp/

Actor Critic

- The idea is to learn the value function as well as the policy.
- Critic
  - Critic updates the value function parameters **w**. Depending on the algorithms, the value function could be action-value Q(a|s), or state-value function V(s). The update algorithm is the following.
  - First, compute TD error of action value by delta = reward + gamma * Q(next s, next a) - Q(s, a)
  - Then, w = w + learning rate for **w** * delta * gradient of Q(s, a)
  -
- Actor
  - Actor updates the policy parameters **theta** for pi(a, s). The update algorithm is the following
  - theta <- theta + learning rate for **theta** * action value Q * gradient of log pi(a, s).
  - So Actor's update depends on the output of Critic, action value Q
- https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f#:~:text=The%20%E2%80%9CCritic%E2%80%9D%20estimates%20the%20value,such%20as%20with%20policy%20gradients).

Trust Region Policy Optimization (TRPO)

- https://github.com/pat-coady/trpo
- https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

- https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-part-2-f51e3b2e373a

Videos

- Trust Region Policy Optimization
  - https://sites.google.com/site/trpopaper/
- Deep Deterministic Policy Gradient
  - https://www.youtube.com/watch?v=tJBIqkC1wWM&feature=youtu.be

Neural network

- Implementing a neural network from scratch
  - http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/
- Demystifying deep convolutional neural network
  - https://www.cs.ryerson.ca/~aharley/neural-networks/
- Convolutional neural network for visual recognition
  - https://cs231n.github.io/neural-networks-case-study/

Derivative of sigmoid function

- Things to remember is the following
  - $\frac{e^{-z}}{(1+e^{-z})^2} = \frac{1e^{-z}}{(1+e^{-z})(1+e^{-z})} = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} = \frac{1}{1+e^{-z}}\frac{e^{-z}+1-1}{1+e^{-z}} = \frac{1}{1+e^{-z}}(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}) = \frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}})$
  - $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Batch Normalization

- https://arxiv.org/abs/1502.03167

TensorFlow

- tf.convert_to_tensor
  - https://stackoverflow.com/questions/58897927/why-we-need-tf-convert-to-tensor
- tf.GradientTape
  - Compute gradient
  - https://medium.com/analytics-vidhya/tf-gradienttape-explained-for-keras-users-cc3f06276f22
  - https://www.tensorflow.org/guide/autodiff

    - https://www.tensorflow.org/api_docs/python/tf/GradientTape
- assign_sub
    - Use this when update parameters by previous - learning rate * gradient
    - https://www.tensorflow.org/api_docs/python/tf/compat/v1/assign_sub
- kernel_initializer
    - https://keras.io/api/layers/initializers/

Questions

- What is policy iteration methods? What is policy gradient methods? What is derivative-free optimization methods?