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 goald is to reduce teh variance of policy gradients, which exist in high sample complexity in difficult control problems. GAE provides a method to nicely estimate the advantage function.
- The generalized advantage estimator, $GAE(\gamma, \lambda)$, is a discounted sum of Bellman residual terms as followed,
    - $\hat{A}_t^{GAE(\gamma,\lambda)} := \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$
    - 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.
- Parameters
    - γ
        - A variance reduction parameter
        - $\gamma \in [0,\ 1]$
        - In cart-pole, the best results are obtained $\gamma \in [0.96,\ 0.99]$
    - λ
        - A compromise between bias and variance.
        - $\lambda \in [0,\ 1]$
        - In cart-pole, the best results are obtained $\lambda \in [0.92,\ 0.99]$
    - Empirically, the best value of lambda is much lower than the best value of gamma.

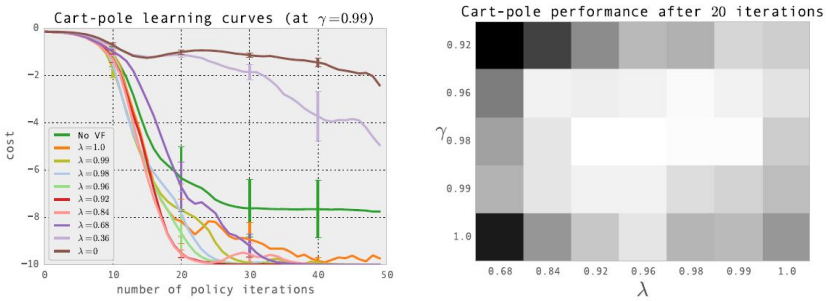

Figure 2: Left: learning curves for cart-pole task, using generalized advantage estimation with varying values of λ at $\gamma = 0.99$. The fastest policy improvement is obtain by intermediate values of λ in the range $[0.92, 0.98]$. Right: performance after 20 iterations of policy optimization, as γ and λ are varied. White means higher reward. The best results are obtained at intermediate values of both.

    -
- Example
    - Cartpole task
        - The value function uses a neural network with one 20-unit hidden layer, and uses a linear policy.
        - 20 trajectories per batch, a maximum length of 1,000 timesteps, the cost as negative reward and in minimized.
- Github
    - https://github.com/bsivanantham/GAE
    - https://github.com/Anjum48/rl-examples/tree/56aca982fcf4426c02aa7e5fb58a4f8affab8020
- Video
    - https://sites.google.com/site/gaepapersupp/

Proximal Policy Optimization (PPO)

- PPO uses clipping the probability ratio and taking minimum of clipped and non-clipped in order to prevent the policy from updating too much.

- Clipping is controlled by ε.
- Probability ratio is $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. If the left-hand-side is greater than 1, it means the action is more probable in the current policy. If the action takes more probable in our current policy than our former, this would lead to a large policy gradient step and result in an excessive policy update.
- Taking minimum is done by the following,
  - $L^{CLPI}(\theta) = \widehat{E}_t[min(r_t(\theta)\widehat{A}_t,\ clip(r_t(\theta),\ 1-\varepsilon,\ 1+\varepsilon)\widehat{A}_t]$.

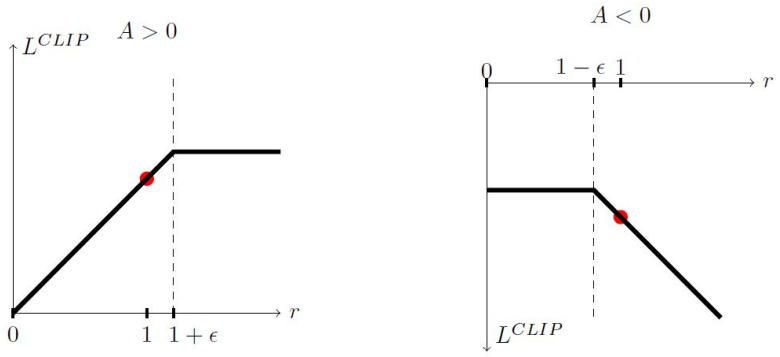

Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

-
- Suppose the following 4 cases, which all follow the above graphs.
  - A = 1, r = 1.2, e = 0.1 -> 1.1
    - Advantage is positive, so an action is better than the average action. So we should let the policy increase the probability of the action. But 1.2 is clipped, and the result is only 1.1. So we don't update policy too much.
  - A = 1, r = 0.8, e = 0.1 -> 0.8
    - Advantage is positive, so an action is better than the average. But r < 1, so the action is not probable in the current policy, so clipping is not applied.
  - A = -1, r = 1.2, e = 0.1 -> -1.2
    - Advantage is negative, so this action should be discouraged. And r > 1, so the action is more probable in the current policy, so we don't clip.
  - A = -1, r = 0.8, e = 0.1 -> -0.9
    - Advantage is negative, so not good action, and r < 1 so that the action is not probable in the current policy. But we don't wanna big change, so it's clipped to -0.9.
- Multiple epochs of minibatch updates
- Empirical results from robotic locomotion and Atari games.
- Clipped probability ratios
- $L^{CLPI}(\theta) = \widehat{E}_t[min(r_t(\theta)\widehat{A}_t,\ clip(r_t(\theta),\ 1-\varepsilon,\ 1+\varepsilon)\widehat{A}_t]$.

    - Ignores the change in probability ratio when it would make the objective function improve, and include it when it makes the objective worse. By taking the minimum of the clipped and unclipped objective, the final objective is a lower bound on the clipped objective.
- $L^{CPI}(\theta) = \widehat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\widehat{A}_t] = \widehat{E}_t[r_t(\theta)\widehat{A}_t]$.
    - CPI is conservative policy iteration
    - $\widehat{A}_t$ is an estimator of the advantage function at timestep t.
    - $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} = r_t(\theta)$ is the probability ratio.
    - We want to maximize this.
    - If r_t(theta) is greater than 1, it means the action is more probable in the current policy. If the action take more probable in our current policy than our former, this would lead to a large policy gradient step and result in an excessive policy update. That's why we use L^CLIP to not have too large policy updates.
- Program
    - https://github.com/jw1401/PPO-Tensorflow-2.0
    - https://github.com/clwainwright/proximal_policy_optimization
    - https://pylessons.com/PPO-reinforcement-learning/
    - https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6
    - https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-2-2-gae-and-ppo-loss-fe1b3c5549e8
    - https://github.com/takuseno/ppo
    - https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow/tree/master/contents/12_Proximal_Policy_Optimization

OpenAI Gym Super Mario Bros.

- https://pypi.org/project/gym-super-mario-bros/

Google Research Football

- https://github.com/google-research/football

Actor Critic with Experience Replay (ACER)

- Truncation with bias correction trick
- Average policy network
- In Atari, $c = 10$, importance weight truncation is used.
- In MuJoCo, $c = 5$, the truncated threshold for ACER.

- Applicable to both discrete action and continuous action.
- Retrace
- Stochastic Dueling Network (SDN)
- Video
    - https://www.youtube.com/watch?v=GkB395ZGvN0&list=PLkmHIkhlFjiTlvwxEnsJMs3v7seR5HSP-&index=6

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

OpenAI Gym

- https://gym.openai.com/docs/
-

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

Exponential linear unit (ELU)

- ELU can have negative values, so it has the mean of activation close to zero. It is argued that having mean activation closer to zero can let faster learning and convergence happen.
- https://medium.com/@krishnakalyan3/introduction-to-exponential-linear-unit-d3e2904b366c

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?

Install Ubuntu on Windows

- Get Rufus
    - https://rufus.ie/
- Get Ubuntu ISO file
    - https://ubuntu.com/download/desktop
    - ISO file is a disk image of an optical disk which means an archive file that contains everything that would be written to an optical disc, sector by sector, including the optical disc file system. ISO is International Organization for Standardization.
- Have Ubuntu on a USB stick, bootable.
- Try Ubuntu before installing it.
- Make free space on the computer hard disk
    - Command prompt as system administrator
    - $ diskmgmt.msc
    - Shrink volume with 20,000 MB shrink volume
    - Check there's a free space with about 20GB.
- Disable fast startup
    - But in my case, there's no checkbox of turn on fast startup, so did nothing.
- Disable secure boot in Windows to allow dual booting with Linux
    - https://www.appgeeker.com/recovery/disable-uefi-secure-boot-in-windows-10.html
- Install Ubuntu from USB
- Make swap partition

    - Open gparted
    - Right-click unallocated, and New
    - Recommended size is the same as RAM (In my case, 15GB) to twice as big as RAM.
    - But this time I set 4GB as kinda sqrt(15GB RAM). Name swap, choose File System as linux-swap
    - From Edit, apply
    - $ sudo blkid /dev/sda3
        - Get UUID of a newly created swap partition
    - $ gedit admin:///etc/fstab
        - Add a line UUID=[UUID you just got] none swap
    - From gparted, right-click swap partition, swapon.
- Resource
    - Ubuntu official
        - https://ubuntu.com/tutorials/create-a-usb-stick-on-windows#1-overview
        - https://ubuntu.com/tutorials/try-ubuntu-before-you-install#1-getting-started
        - https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
    - Windows dual booting
        - https://itsfoss.com/install-ubuntu-1404-dual-boot-mode-windows-8-81-uefi/
        - https://www.tecmint.com/install-ubuntu-alongside-with-windows-dual-boot/
        - https://help.ubuntu.com/community/WindowsDualBoot
    - Disable secure boot
        - https://itsfoss.com/disable-uefi-secure-boot-in-windows-8/
    - Remove Ubuntu from Windows dual booting
        - https://itsfoss.com/uninstall-ubuntu-linux-windows-dual-boot/
    - UEFI
        - https://www.howtogeek.com/56958/htg-explains-how-uefi-will-replace-the-bios/
    - Increase size of available shrink space of hard drive partition in Windows
        - https://medium.com/@terajournal/increasing-size-of-available-shrink-space-for-hard-drive-partition-in-windows-8fffa50535d3

Allow root user login in Ubuntu GUI

- https://linuxconfig.org/how-to-allow-gui-root-login-on-ubuntu-20-04-focal-fossa-linux

Install PyCharm in Ubuntu

- Download Linux PyCharm community edition
- $ sudo tar xzf pycharm-*.tar.gz -C /opt/
- Resource

    - https://www.jetbrains.com/help/pycharm/installation-guide.html#standalone
-
-