2020-10-04
Yuki Kitayama

Reinforcement Learning Manual

Deep Deterministic Policy Gradient 3

Twin Delayed Deep Deterministic policy gradient (TD3) 4

Policy Gradient 4

Generalized Advantage Estimation (GAE) 5

Proximal Policy Optimization (PPO) 6

OpenAI Gym Super Mario Bros. 8

Google Research Football 8

Actor Critic with Experience Replay (ACER) 8

Actor Critic 8

Trust Region Policy Optimization (TRPO) 9

Distributional Reinforcement Learning 9

Quantile Regression DQN (QR-DQN) 9

Implicit Quantile Network (IQN) 10

OpenAI Gym 10

Videos 10

Neural network 10

Initialization of neural network parameters 11

Regularization 11

Inverted dropout 11

Derivative of sigmoid function 11

Exponential linear unit (ELU) 12

Batch Normalization 12

Adam 12

TensorFlow 13

PyTorch 13

Kullback Leibler divergence 13

Wasserstein metric **14**

Contraction mapping **14**

Questions **14**

2020-09-12 **14**

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
    - $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
- Blog
  - https://towardsdatascience.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197
- GitHub

    - Lunar lander continuous
        - https://github.com/shivaverma/OpenAIGym/tree/master/lunar-lander/continuous

Twin Delayed Deep Deterministic policy gradient (TD3)

- Official GitHub
    - https://github.com/sfujim/TD3

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,
  - $\widehat{A}_t^{GAE(\gamma,\lambda)} := \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$
    - 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.
- 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.

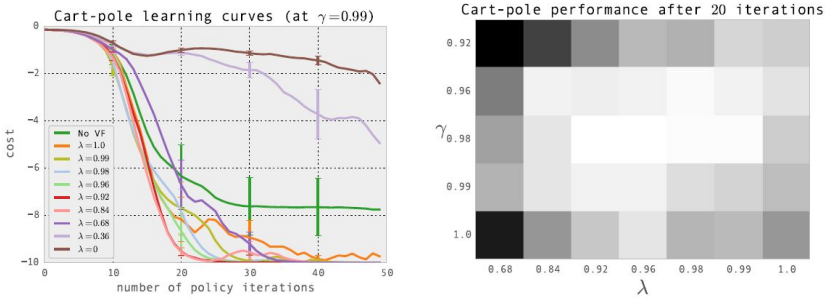

Figure 2: Left: learning curves for cart-pole task, using generalized advantage estimation with varying values of λ at $\gamma = 0.99$. The fastest policy improvement is obtain by intermediate values of λ in the range $[0.92, 0.98]$. Right: performance after 20 iterations of policy optimization, as γ 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,
        - $L^{CLPI}(\theta) = \widehat{E}_t[min(r_t(\theta)\widehat{A}_t,\ clip(r_t(\theta),\ 1-\varepsilon,\ 1+\varepsilon)\widehat{A}_t]$.

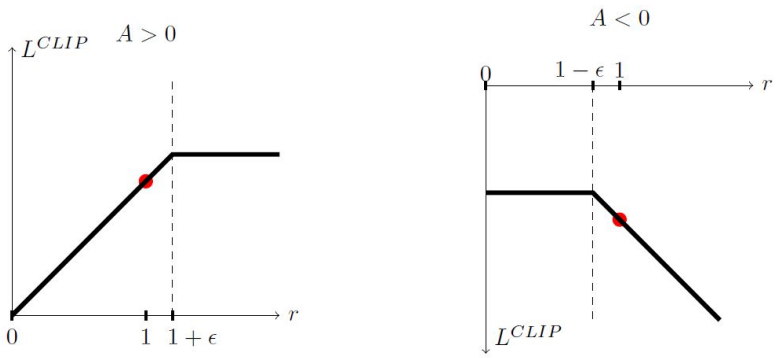

Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} 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
- Example
    - https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic
    - https://keras.io/examples/rl/actor_critic_cartpole/
    - https://medium.com/@asteinbach/actor-critic-using-deep-rl-continuous-mountain-car-in-tensorflow-4c1fb2110f7c
    -
- Blog

    - 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

Distributional Reinforcement Learning

- Concept
    - https://en.wikipedia.org/wiki/Contraction_mapping
- Blog
    - https://medium.com/analytics-vidhya/distributional-reinforcement-learning-part-1-c51-and-qr-dqn-a04c96a258dc
    - https://mtomassoli.github.io/2017/12/08/distributional_rl/
- GitHub
    - https://github.com/Silvicek/distributional-dqn
    - https://github.com/marload/DistRL-TensorFlow2
    - https://github.com/BY571/DQN-Atari-Agents

Quantile Regression DQN (QR-DQN)

- It is something like adding an additional hyperparameter to DQN.
- The number of quantiles N. This controls with what resolution we approximate the value distribution.
- The more N, the more able to estimate the upper and lower quantiles of the value distribution.

Implicit Quantile Network (IQN)

- GitHub
    - TensorFlow
        - https://github.com/chagmgang/tf2.0_reinforcement_learning/blob/master/rainbow/iqn.py
        - https://github.com/Kyushik/DRL/blob/master/11_IQN.py
    - PyTorch

        - https://github.com/dannysdeng/dqn-pytorch
        - https://github.com/BY571/IQN-and-Extensions/blob/master/IQN-DQN.ipynb

Deep Q Network (DQN)

- Blog
    - https://towardsdatascience.com/deep-reinforcement-learning-build-a-deep-q-network-dqn-to-play-cartpole-with-tensorflow-2-and-gym-8e105744b998

OpenAI Gym

- Useful wiki page in OpenAI Gym
    - https://github.com/openai/gym/wiki/Table-of-environments
-
- 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/

Initialization of neural network parameters

- Randomly initialized large weights does not work because the cost function produces the large values initially. It's because the last activation function produces the extreme value.
- Poor initialization results in exploding or vanishing gradients and slows down optimization.

- He initialization works well for networks with relu activations
- https://arxiv.org/abs/1502.01852
- https://medium.com/@prateekvishnu/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528

Regularization

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right) \tag{1}$$

To:

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{kj}^{[l]2}}_{\text{L2 regularization cost}} \tag{2}$$

- If lambda is too large, it would make a model which is over-smooth, resulting in a model with high bias like linear regression.
- Because we have an extra term in cost function, the resulted computed weights will be smaller. L2-regularization has assumption that a model with smaller weights is simpler than a model with large weights. It's called weight decay because weights are smaller by regularization.

Inverted dropout

- During training time, divide each dropout layer by keep_prob to keep the same expected value for the activations. For example, if keep_prob is 0.5, then we will on average shut down half the nodes, so the output will be scaled by 0.5 since only the remaining half are contributing to the solution. Dividing by 0.5 is equivalent to multiplying by 2. Hence, the output now has the same expected value.

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

Adam

- Adam means adaptive momentum estimation. It's a way to update weights in neural network. It's like you are averaging gradients, by using two hyperparameters beta1 and beta2. It calculates the exponentially weighted average of past gradients and the squares of past gradients, and use them in the gradient descent instead of just using gradients.
- $v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1 - \beta_1)\frac{\partial J}{\partial W^{[l]}}$
  - Averaging gradients, by calculating exponentially weighted average. If beta1 is large, averaging effect is large, and smoothes gradients.
- $v^{corrected}_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t}$
  - At first in calculating averages, the data is not enough and the average is biased to 0, so this correction reflects data more realistically.
- $s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1 - \beta_2)(\frac{\partial J}{\partial W^{[l]}})^2$
  - Get the similar average from the square of gradients.
- $s^{corrected}_{dW^{[l]}} = \frac{s_{dW^{[l]}}}{1-(\beta_2)^t}$
  - Same removing bias purpose.
- $W^{[l]} = W^{[l]} - \alpha\frac{v^{corrected}_{dW^{[l]}}}{\sqrt{s^{corrected}_{dW^{[l]}}}+\varepsilon}$
  - This is Adam way of gradient descent update of weights in neural network.
- In RMSprop, if $\beta = 0$, the adaptive momentum becomes the standard gradient descent without momentum. If the beta is large, we have a smoother update. Usually use 0.9

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/

PyTorch

- Official tutorials
    - https://pytorch.org/tutorials/index.html
-

Kullback Leibler divergence

- It's a distance measure between two distributions. When you get expected values from two distributions and try to get a difference of two expectations, the calculation eventually becomes below. Think P as the true probability distribution, and Q as the predicted probability distribution.
- $KL(P(x) - Q(x)) = \sum_x P(x) log_2(\frac{P(x)}{Q(x)})$
- The following video is intuitive.
    - https://www.youtube.com/watch?v=LJwtEaP2xKA
- https://www.youtube.com/watch?v=xmvxXXZUXdk
- Cross-entropy = entropy + KL divergence
- $D_{KL}(p||q) = H(p,q) - H(p)$
- https://www.youtube.com/watch?v=ErfnhcEV1O8

Wasserstein metric

- Wasserstein metric plays a key role in distributional RL. the Wasserstein metric trades off approximate solutions with likelihoods unlike KL divergence.
- This is something like a way to compute a distance between two distributions. The distribution is smooth and continuous things, but we discretize and make it histogram to approximate. Then we divide each bar in a histogram into small pieces in two distributions. Then we compute the distances between two small pieces, by distance of height and width.
- https://www.youtube.com/watch?v=ymWDGzpQdls&t=913s

Contraction mapping

- https://www.youtube.com/watch?v=_DynXugXksU
-

Questions

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

2020-09-12

-
- https://gym.openai.com/envs/MountainCarContinuous-v0/
- https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69
- https://blog.tensorflow.org/2018/07/deep-reinforcement-learning-keras-eager-execution.html