Advantage Actor-Critic Method (A2C)

Hongshan Li (hongshal@amazon.com)

September 10, 2021

A few problems with policy gradient algorithms (REINFORCE)

- The algorithm is not sample efficient, one trajectory is only used to update the policy once and it is then tossed away.
- the training can be unstable, because the gradient policy gradients are "offset reinforced" by the state action value. Recall

$$\nabla J(\pi_\theta) = \alpha \times q_\pi(s,a) \times \nabla \log \pi(a|s)$$

$$q_\pi(s,a) = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

  So actions with outlier rewards can change the policy gradient substantiatlly and make the learning unstable.

A2C is another policy gradient based method that tackles the above problems. Main ideas of A2C are:
Sample experiences from different randomizations of the environment in parallel, so that more experiences with more variety can be used to train the policy.
Use a value net to predict a state value $v(s)$, which is used as the baseline of offset reinforcement. This reduces the variance of training.
Beside training a policy net (actor), we also train a critic (value net), f the state $v(s)$. We can use it compute the "advantage" of the action

$$A(s,a) = q_\pi(s,a) - v(s)$$

and use $A(s,a)$ as the "offset reinforcement", so that the policy gradient becomes

$$\nabla(J(\theta)) = \alpha \times A(s,a) \times \nabla \log \pi(a|s)$$

By using $A(s,a)$ we will still maximize the likelihood of good actions, in fact, if you have good and mediocre actions with positive reward, using $A(s,a)$ as offset reinforcement would minimize the likelihood of the mediocre action, because it would have negative A.
If both actor and critic learn as expected, then higher $Q(s,a)$ should correspond to higher $V(s)$. Thus, variance of $Q(s,a) - V(s)$ is lower than that of $Q(s,a)$.

1 Training value net (critic) with Bellman equation

Value net predicts the value of the state if we were to follow the policy (actor) so more precisely, value net predicts $v_\pi(s_t)$ at state s_t. $v_\pi(s_t)$ is defined recursively in terms of next state

$$v_\pi(S_t) = \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}))$$

(S_t means the random variable of state at time step t, s_t means a specific state the environment assumes at time step t)
This gives us a loss objective for training the value net. At current time step, the objective is to make your prediction as close as you would be if you were in the next state. Intuition is, being in the next state always expose you with a bit more understanding of the env. This is what Richard Sutton would say "update your guess with a better guess".
So the loss objective of critic is

$$L(v_\pi(s_t, a_t)) = ||r_{t+1} + \gamma v_{s_{t+1}} - v_{s_t}||^2$$

The equation

$$v_\pi(S_t) = \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}))$$

is called Bellman equation (Richard Bellman, same guy coined the term dynamic programming). It is not exactly a definition, it is a consequence of two properties of the MDP

- (Memoryless) the transition probability only depends on the current state
- (Ergodicness) the action of the agent does not change the model (transition probablity) of the env

2 A2C algorithm

Algorithm 1: A2C

INPUT: a differentiable policy π_θ; a differentiable value net v_ϕ; learning rates α and β for π and v respectively; a predefined number of parallel environment n

repeat

  Generate n trajectories $\tau_1, \cdots, \tau_n$ in parallel using replicas of the π. Each trajectory τ_i is of the form $(s_{i,0}, a_{i,0}, r_{i,1}, \cdots, s_{i,T-1}, a_{i,T-1}, r_{i,T})$

  for $i = 1, \cdots, n$ **do**

    for $t = T-1, T-2, \cdots, 0$ **do**

      Compute

$$q_\pi(a_{i,t}, s_{i,t}) = r_{i,t+1} + \gamma r_{i,t+2} + \cdots + \gamma^{T-t-1} r_{i,T}$$

      Estimate state value through value net

$$v_{i,t} = v_\phi(s_{i,t})$$

      Compute advantage for $(s_{i,t}, a_{i,t}, r_{i,t+1})$

$$A(s_{i,t}, a_{i,t}) = q_\pi(a_{i,t}, s_{i,t}) - v_{i,t}$$

      Estimate policy gradient

$$\nabla \hat{J}(\theta) = \nabla \log \pi(a_{i,t}|s_{i,t}) A(s_{i,t}, a_{i,t})$$

      Update policy parameters

$$\theta \leftarrow \alpha \frac{1}{nT} \nabla \hat{J}(\theta)$$

      Compute value net loss

$$L(s_{i,t}, a_{i,t}) = ||v_\phi(s_{i,t}, a_{i,t}) - r_{i,t+1} - \gamma v_\phi(s_{i,t+1})||^2$$

      Compute the semi-gradient of L with respect to ϕ by treating $v_\phi(s_{i,t+1})$ as a constant

      Update ϕ by gradient descent with learning rate $\beta * \frac{1}{Tn}$

    end for

  end for

until The policy converges

References

[1] V. Minh et al, *Asynchronous Methods for Deep Reinforcement Learning*