Some effective tricks are used to improve Soft Actor Critic

Chenjie Qin¹, Lijun Zhang², Dawei Yin³, Dezhong Peng⁴ and Yongzhong Zhuang*  
¹,⁴College of Computer Science, Sichuan University, Chengdu, 610065, China  
²Chengdu Ruibe Information Technology Limited Company, chengdu, China  
³Sichuan Zhiqian Technology Co., Ltd, chengdu, China  
*Chengdu Dinganhu Smart IOT Holding Co., Ltd, Chengdu, China  
*Corresponding author’s e-mail: zhunagy@dahiot.com

Abstract. The development of artificial intelligence is becoming more intelligent, and reinforcement learning (RL) plays an important role. RL has been widely used in recommendation systems, smart cars, game AI, and investment transactions. Since the emergence of DQN, DRL improvement methods have been proposed continuously. Soft Actor-Critic (SAC) is the current state-of-the-art RL algorithm. In this paper, some tricks are used to improve SAC. Firstly, the baseline is added to the policy gradient algorithm to reduce variance and speed up training. This paper adopts a new method to implement it in SAC. Because the temperature coefficient is a super parameter, which is difficult to adjust, so I propose a novel method to set it. These tricks proved to be effective on the MUJOCO.

1. Introduction
The development of artificial intelligence is becoming more intelligent and reinforcement learning (RL) plays an important role in recommendation systems, smart cars, game AI, and investment transactions. The concept of RL is that the Agent interacts with the environment through itself, which is different from supervised learning and unsupervised learning. The Agent tries to maximize the expectations of rewards, and get the best policy \( \pi^* \). Before the advent of DQN, people use a tabulation to store the values of all state-action pairs \( Q(s,a) \). Then the Agent selects an optimal action \( a^* \) for each state \( s \) from the tabulation. However, the tabulation consumes a lot of computational resource with an enormous state space and action space, and it is difficult to get reasonable actions for unknown states. In order to solve the shortcoming of RL, the DeepMind proposed an algorithm called DQN [11]. A convolutional neural network (CNN) is used to extract image features which are used to train the Agent, so that the Agent could solve tasks in continuous state space. The Agent trained by DQN has surpassed human master's level in many Atari games, and the emergence of DQN has also injected strong vitality into the RL community.

In the RL community, there are three main research directions: value-based model, policy-based model, and the Actor-Critic model. DQN is a typical value-based model, to get an approximate network \( Q(s,a; \theta) \), \( \theta \) is the parameter of the network. Although the DQN is effective, it also has shortcomings. The main disadvantage is that it cannot solve continuous action tasks. Therefore, the policy-based model is proposed, which can solve the above problem with faster convergence speed. Unfortunately, the variance of the policy-based model is large and accompanied by fluctuations. The Actor-Critic model
(Abbreviated as AC) which combines the advantages of the two models is proposed, which is better than the two methods mentioned above in most cases. SAC is an algorithm based on the AC model.

There are three tricks to improve the performance of the DQN. Firstly, ordinary DQN is updated by its maximum, which leads to overestimation. The Double DQN is proposed to avoid overestimation. A new Q network is introduced, and its update method is hard update[2]. Secondly, the prioritized replay is proposed to improve learning efficiency and solve the problem of sparse rewards[3]. Thirdly, the Dueling DQN is proposed to highlight the advantage value of the action, which reduces the variance of the algorithm. The idea of Dueling DQN is introduced into the algorithm of this paper, and a new method of setting temperature coefficient is proposed.

2. Related Work / Background

RL interacts with the environment to obtain the feedback (reward) of the environment which used to guide Agent update, and the algorithm flow is demonstrated in figure 1. The standard RL follows the Markov decision process (MDP) \((S, A, R, P)\), where \(S\) and \(A\) respectively denote the state space and action space, \(R\) denotes the reward function (state-action-dependent), \(P(\cdot | s, a)\) denotes the transition kernel. And a discount factor \(\gamma\) (between 0-1) is adopted to balance the importance of future rewards.

Based on the MDP, the state-action sequence is \(S, a_1, s_2, a_2, ..., s_t, a_t\). The goal of the RL is that the Agent does not mechanically get the current best reward, but has a foresight to maximize the cumulative reward, including future rewards. The reward is defined as \(r_t\) (a random return) at time-step \(t\), so the cumulative reward \(\text{return}\) at time-step \(t\) is \(R_t = \sum_t^T \gamma^t r_t\), where \(T\) is termination state (if the task is a continuous task, \(T \to \infty\)). The action-value approximate function is defined as a neural network \(Q(s, a)\), and the optimal action-value approximate function \(Q^*(s, a)\) is the maximum expectation of the cumulative reward. In other words, \(Q^*(s, a) = \max\ E[R_t | s_t = s, a_t = a, \pi]\), where \(\pi\) is the policy, which is used to select the action in each state \(s\). From the above, the Bellman equation could be easily get:

\[
Q^* = E[r + \gamma \max_a Q'(s', a')]
\]

This equation is the most important equation for the RL. A neural network with parameters \(\theta\) is used to approximate the optimal action-value function \(Q^*\) by the DQN. The loss function \(L_i(\theta)\) of the network changes at each iteration \(i\):

\[
L_i(\theta) = E_{s,a \sim \rho(s,a)} [(y_i - Q(s,a;\theta))^2]
\]

where \(y_i = \max_{a'}[r + \gamma Q(s', a';\theta)]\) is the target which is similar to the label of supervised learning, and \(\rho(s,a)\) represents the state-action distribution, the above formula is called TD error. By reducing the TD error, Q can be converted to a fixed point.

![Figure 1](image)

Figure 1 The interaction process between the agent and the environment.

2.1. Dueling network

In many cases, the size of the action value cannot express the difference in the action value very vividly. For example, In state \(s_1\), there are two actions \(a_1\) whose value is 1 and \(a_2\) whose value is 2; then The advantage value of \(a_1\) is -0.5, and the advantage value of \(a_2\) is 0.5, which is more vivid than the
previous one, and can reduce the variance of training. There is a new network architecture in Dueling network, which decomposes action value into the sum of state value and action advantage value. It is divided into two network outputs in the penultimate layer, $V(s)$ and $Adv(s, a)$, and their sum is $Q(s, a)$.

$$Q(s, a) = V(s) + Adv(s, a)$$  \hspace{1cm} (3)

As can be seen from equation (3), advantage-value $Adv(s, a) = Q(s, a) - V(s)$. The action value decomposition could well reflect the real advantages and disadvantages of each action in state $s$. In other words, we set the Agent to be able to select actions with high advantage value in different states. This method makes the distinction between the pros and cons of the actions clearer, especially in the excellent and the poor state. It not only improves the training speed, but also reduces the variance of the network, which is proved by the paper Dueling Network [4].

2.2. SAC

According to whether the behavior policy and the target policy are consistent, the RL algorithm can be divided into on-policy and off-policy learning. On-policy learning is the one cause for poor sample efficiency, in other words, it takes a lot of time to collect samples, such as SARSA, A3C[5], TRPO[6] and PPO[7]. This kind of sample waste will bring a lot of consumption. With the increase of feature complexity, the number of samples required will also increase rapidly, which means that once the samples are used, they will be discarded. Off-policy has a high sample utilization rate and can break sample correlation at the same time. SAC is this type of algorithm, and it is also the current state-of-the-art algorithm. SAC can well avoid suboptimality. Compared with other off-policy methods, SAC[8] is a random policy based on the maximum entropy framework. Entropy regularization is added to SAC, so action distribution will be more even that adds more exploration. In complex tasks, better policy can be learned through the maximum entropy framework. The calculation method of entropy is:

$$H(P) = E_{x \sim P}[- \log P(x)]$$ \hspace{1cm} (4)

where $P$ denote the action distribution, $x$ is variable. The SAC is almost the state of the art of single Agent model. In this paper, I use Dueling-network architecture to the SAC and adjust the temperature parameter $\alpha$ in my own way. I will show the performance on the MUJOCO.

3. Methodology

3.1. SAC based on advantage value

The goal of SAC is to get a maximum entropy model. In fact, other algorithms also have the concept of entropy, but they are all used in the exploration stage to increase the desire for exploration. However, SAC not only adds the entropy to the action selection, but also adds information entropy when calculating the Q value, which is used to ensure the maximum entropy:

$$\pi^* = \arg \max_{\pi} \sum_{t=0}^{T} E_{(s_t, a_t) \sim \rho} [r(s_t, a_t) + \alpha H(\pi(\cdot | s_t))]$$ \hspace{1cm} (5)

where $\alpha$ denotes the temperature coefficient that indicates the importance of entropy regularization, and thus controls the exploration of optimal policy. But how to adjust this coefficient is a tricky problem. In section 3.2, I propose a simple and effective temperature coefficient setting method that does not require parameter training. The Q function of SAC is soft-Q:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \rho} [V(s_{t+1})]$$ \hspace{1cm} (6)

where:

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t) - \alpha \log \pi(a_t | s_t)]$$ \hspace{1cm} (7)

There are two Q networks in SAC $Q_\theta$ and $Q_\phi$, which makes the network training stable and avoid overestimation. Actor-network is updated with a small Q value every time. The loss of policy:
\[ L(\phi) = \mathbb{E}_{s_t \sim D}[\alpha \log \pi_\theta(a_t|s_t) - \min_{i=1,2} Q_\theta_i(s_t, a_t(\mu, s))] \]  
where \( a_t(\mu, s) \) is resampled with \( \pi_\theta(s_t) \), \( \phi \) is the weights of Actor-network. In this paper, I changed \( Q(\ast) \) to \( Q(\ast) - V(\ast) \):

\[ L(\phi) = \mathbb{E}_{s_t \sim D}[\alpha \log \pi_\theta(a_t|s_t) - (\min_{i=1,2} Q_\theta_i(s_t, a_t(\mu, s)) - V(s_t))] \]

This paper adds a new network \( V(\omega) \) to evaluate state value \( V(s) \), and it's updated in a way similar to the action value function \( Q \):

\[ V(s_t) = \mathbb{E}_{s_t \sim D}[r + \gamma V(s_{t+1})] \]

This network is updated after the action value network \( Q \) is trained for a certain number of times, because we need to ensure that the baseline does not change within a certain number of training times, otherwise the network will be extremely unstable. Experiments have proved that the method is effective, but the effect of deterministic strategies such as DDPG is mediocre. The reason should be that SAC has good action randomness, and the V network can reflect the advantages of different actions. However, deterministic strategy actions have poor randomness and do not reflect the difference in actions, in Figure 2.

### 3.2. Simple variable temperature coefficient

SAC temperature coefficient \( \alpha \) controls the stochasticity of the optimal policy, which gets higher exploration and avoids suboptimality. However, how to set this coefficient is an intractable problem. The method given by the SAC of author is to train \( \alpha \) as a parameter, and it been proved to be effective. I propose a simpler method on this basis, which can get good results and make training easier. I assume that there is an inverse relationship between rewards and exploration. This is easy to understand that insufficient samples in the early stage lead to low reward expectations, therefore, the action distribution is very even so that the Agent is in the exploring state. With more and more samples, the expectation of rewards is getting bigger and bigger, and the Agent is in the state of utilization. Then the coefficient should be reduced at this time. Therefore, I define the temperature coefficient \( \alpha \) as the inverse function of the reward expectation:

\[ \alpha = f(Q_\theta(s, a)) \]

where \( f(\ast) \) is an inverse function, whose function value decreases as the independent variable increases.

Usually, it is necessary to limit \( \alpha \) within a scope, so functions, reciprocals of Sigmoid and Tanh functions can be used, and the tanh function works well.

In other words, the reward value becomes larger and the temperature coefficient becomes smaller, so that the early stage is conducive to exploration, and the later stage is conducive to obtaining higher rewards. In this way, higher rewards can be obtained, and because the method of obtaining is simple, it will not add a lot of calculation to training. This method is similar to the closed-loop control in the automatic control system, which can make the system finally stable. Two tricks are used in this paper, whose performances are shown in section 4.

### 4. Results & Discussion

In this paper, I do some experiments on MUJOCO to compare the method in this article with TD3 and SAC. Because I use the temperature coefficient that changes with the system, the Agent can well balance exploration and utilization.

At the same time, the idea of Advance-value is added, which reduces the variance of the system and increases the efficiency. Therefore, the two tricks enable the Agent to obtain higher returns compared to other algorithms, and reduce the variance of the algorithm, which is proved in the table 1, table 2 and figure 3.
Table 1  The mean rewards of the three algorithms after training 3 million times in the two games environments of MUJOCO (Hopper-v2, HalfCheetah-v2). The training batch size is 128, and the optimizer is Adam.

| Environment     | TD3      | SAC       | My method |
|-----------------|----------|-----------|------------|
| Hopper-v2       | 5108.90  | 5662.40   | 5968.49    |
| HalfCheetah-v2  | 1405.17  | 1662.76   | 1742.11    |

Table 2  The mean rewards of the three algorithms after training 2 million times in the two games environments of MUJOCO (Ant-v2, Walk2d-v2). The training batch size is 128, and the optimizer is Adam.

| Environment     | TD3      | SAC       | My method |
|-----------------|----------|-----------|------------|
| Ant-v2          | 1702.33  | 2232.93   | 2901.56    |
| Walk2d-v2       | 1007.69  | 1759.26   | 2170.63    |

Figure 2 (a) and (b) show improvements of the advantage(Q-V) in SAC. Figure 2 (c) and (d) shows that there is no improvement in DDPG.

Figure 2  Comparison chart of advantage(Q-V) value between DDPG and SAC. (a) and (c) are on Ant-v2, (b) and (d) are on HalfCheetah-v2.

Figure 3 shows the reward curves of the three algorithms in the four game environments. The algorithm in this article adds two tricks. It can be clearly seen that the improved algorithm proposed in this article is indeed effective, all the reward curves are more stable and more rewards are obtained.

Figure 3  Reward curves of three algorithms in four game.
5. Conclusions
This paper proposes a method to add two tricks, which significantly improves the reward value of the algorithm and reduces the variance. The algorithm is more stable by adding Advantage-value, and the variable temperature coefficient increases the average reward of the algorithm. Today's RL algorithms still have a lot of room for improvement, such as priority playback is an effective trick. However, tricks are not used randomly, which may bring a negative boost. We need to make appropriate adjustments based on the characteristics of the algorithm. Next I will extend the method to distributed systems.

References
[1] Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with Deep Reinforcement Learning[J].
[2] Van Hasselt, Hado, Guez, Arthur, and Silver, David. (2015). Web.
[3] Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. (2015). Web.
[4] Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling Network Architectures for Deep Reinforcement Learning." (2015). Web.
[5] Mnih, Volodymyr, Badia, Adrià Puigdomènech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. (2016). Web.
[6] Schulman, John, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust Region Policy Optimization." ArXiv.org (2017): ArXiv.org, Apr 20, 2017. Web.
[7] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms," ArXiv.org (2017): ArXiv.org, Aug 28, 2017. Web.
[8] Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic Algorithms and Applications." (2018). Web.