A Review of Reinforcement Learning Fundamentals and Recent Algorithms

Linden Black, *School of Physics and Astronomy, University of Nottingham, Nottingham, NG7 2RD, UK*

***Abstract*—This review paper begins by covering the reinforcement learning fundamentals and concepts. More recent advances in paramterisation of functions and gradient descent methods are then discussed. Finally, state of the art deep reinforcement learning methods, including AC3, PPO2 and DQNs are described. This review serves as a supplementary source to the DIA LunarLander-V2 main report.**

I. Reinforcement Learning Introduction

A. State, Action, Reward and Policy

REINFORCEMENT learning (RL) is a popular branch of machine learning that has applications in autonomous driving [1], algorithmic trading and finance [2], gaming [3] and more. The two main components of RL are an agent and the environment it interacts with. For example, the environment for a trading agent would be the market it traded in. This interaction can be split into two terms, state s, and action a. The state, s_t, describes the environment at timestep t, given information of the state the agent can take an action, a_t, which will lead to a new state, s_{t+1}. Ideal actions in particular states are reinforced through the use of rewards, r_t, with better actions receiving a greater reward. These rewards can now be used to optimise the performance of the agent through the search for an optimal policy, where the policy, $\pi(s|a)$, defines what action the agent will take given the current state. Optimising the reward of the next state is trivial, the difficultly comes in complex environments when the reward from a transition to state many timesteps in the future is also important.

B. Bellman Optimality Equations

It is necessary to quantify what states are better than others, this comes in the form of the value function, $V_\pi(s)$,

$$v_\pi(s) = \mathbb{E}_\pi[R_t|S_t = s] = \mathbb{E}_\pi\,\Big[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|S_t = s\Big] \quad (1)$$

The discount factor γ determines how much future experience is weighted in relation to the immediate reward. R_t is the long term discounted reward. S_t represents the set of$s$, which are the states possible at the current time step. At this point it is worth noting that the state and action system described in this paper and in the wider reinforcement learning field is a Markovnian process. The aim of this paper is to discuss algorithms and machine learning concepts as such Markov theory will not be included, further information can be found in the literature [4, 5]. Similarly we define the state-action value function (Q-function),

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t|S_t = s, A_t = a] \quad (2)$$

Finally, we arrive at the Bellman expectation equation [6] for the value function,

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \quad (3)$$

and for the state-action function,

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \quad (4)$$

C. Q-Learing

Q-learning [7] is a value based, model free, iterative algorithm that attempts to learn the optimal Q-function. It is an off-policy method meaning it learns the value of the optimal policy independent of the agents actions. It is a greedy-algorithm that defines the policy to be the action that maximises $Q(s,a)$.

$$Q(s,a) = r + maxQ(s', a') \quad (5)$$

The procedure creates and maintains a table of $Q[S,A]$. To introduce exploration into the system an epsilon greedy-policy can be defined. A small fraction,ϵ, of the time the agent will take a random action in place of exploiting the best known action from maximising $Q(s,a)$, this allows the agents to explore a wider range of trajectories increasing the generalisation. The exploration factor is set as a low number, commonly, $\epsilon = 0.05$. The method calculates the $Q(S,A)$ for each trajectory in the state space, as the space becomes more complex the method becomes computationally inefficient. As such Q-learning is limited to relatively non-complex, discrete state spaces.

D. Policy Gradients

An alternative approach to optimising the value function as seen in Q-learning, is to optimise the policy. Policy gradient methods work by parameterising the policy $\pi_\theta(a|s)$ and computing an estimator of the policy gradient which is then used in stochastic gradient ascent.

1) REINFORCE:

The REINFORCE [8] algorithm designed by Williams, uses the vanilla policy gradientas its objective function. This suffers from high variability in log probabilities from the policy distribution, leading to noisy gradients and unstable training.

$$L^{REINFORCE} = \mathbb{E}_{\pi\theta}\, log\pi_\theta(s,a)R_t \quad (6)$$

2) Advantage: The use of a baseline function can help to stabilise the training by reducing the variance in the policy gradient. A learned estimate of the value function is commonly used. One approach to this is using the advantage function [9],

$$A_\pi = Q_\pi(s,a) - V_\pi(s) \tag{7}$$

where the V-function serves as the baseline. It has the same expected value as the Q function so it reduces the variance. The advantage function quantifies how much better it is to take a specific action in compared to the average action at a given state, it rewards any action better than average for the current state. Calculating the Q-value is expensive and unnecessary. The temporal difference (TD) error δ_{TD} for the true value function $V_\pi(s)$ is an unbiased estimate of the advantage function $A_\pi(s,a)$ and is calculated instead. Considering a system of one timestep,

$$\delta_{TD}(s,s',r) = r + \gamma V_\pi(s') - V_\pi(s) \tag{8}$$

Where $V_\pi(s')$ is the value of the current action and resulting state. $V_\pi(s)$ is the prior value.

$$\begin{aligned}\mathbb{E}_\pi[\delta_{TD}|s,a] &= \mathbb{E}[r + \gamma V_\pi(s')|(s,a)] - V_\pi(s) \\ &= \hat{A}_\pi(s,a)\end{aligned} \tag{9}$$

Now the unbiased estimator of advantage has been defined, we arrive at the final objective function for the advantage policy gradient method. For a single time step the objective function is,

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{10}$$

The reward is maximised through stochastic gradient ascent on the derivative of $L^{PG}(\theta)$ which moves the parameters in the direction of the greatest reward. Policy based methods are useful in a continuous space where the number of states or actions to required estimate the value is infinite. In contrast, in the iterative Q-learning method to improve the policy, iteration over the entire action space is required, rendering it unsuitable for the continuous space.

II. Deep Q-Networks (DQNs)

In 2013 Deepmind proposed a solution to limitations of Q-learning in the form of a Deep Q Network (DQN)[3]. The paper describes using the popular machine learning method, deep learning [10] to learn successful policies from high-dimensional inputs using end-to-end reinforcement learning. The agents were trained to play the Atari 2600 games. The state space in these games is large and traditional methods would not have performed well due to their iterative nature. The human interpretation of the state space in an Atari game would be the image processed through vision. Using convolutional neural networks (CNNs) [11] to receive an image input, Deepmind were mimicking the human state processing method.

To model the Q-learning algorithm a fully connected network was added to the end of the convolutional layers. The standard Q-learning algorithm had to be adjusted to work with a non-linear function approximator. The first adjustment was using experience replay [12] to store the agent experience at each time step, pooling data over many episodes. In RL an episode is defined as the start to end of a game. During model training mini-batches were randomly uniformly sampled from this buffer. Learning from consecutive samples is inefficient as the data is highly correlated, which in neural networks leads to unstable training and divergent behaviour. The main advantage of experience replay this is that it circumvents this issue by removing the correlation, in short the network inputs are no longer from successive timesteps. Secondly, learning on the current policy means the current parameters determine the next data sample, this can lead to the network getting stuck in a local minima - the experience replay circumvents this by using random sampling. The learning is now off-policy, the current parameters were not used to generate the received sample. The second adjustment introduced the idea of a target and online Q-network. The target network is used to generate the data for the replay buffer and the online network is trained using standard back propagation. If the target network is updated every step the algorithm will be chasing a moving-target, by updating the target network after a number of steps this is avoided. For training, the features were the state, s_t and the labels were the Q-values predicted by the target Q-network, Q_t taken in time step t.

III. Actor-Critic Algorithms

A. Actor-Critic

Actor-Critic is a policy gradient method that was designed to reduce the variance of gradients and unstable training that strongly impacted the vanilla method. The critic estimates the value function, this could be the state-action value $Q(s,a)$ or the state-value $V(s)$ and the actor updates the parameterised policy in the direction suggested by the critic (such as with policy gradients).

B. Advantage Actor-Critic (A2C)

The advantage policy gradient method can be viewed as an actor critic method, with the actor being the policy, π and the predicted value function the critic. Both functions are parameterised by neural networks. With two networks an objective function that includes the parameters and outputs of the functions is required. The objective function shown in equation (10) satisfies this requirement and hence is the objective function for A2C.

C. Asynchronous Actor-Critic Algorithms (AC3)

Policy gradients are obtained online, meaning that we need to use data obtained by the current policy. In order to keep the data i.i.d. a large buffer of transitions is needed. To avoid this, an asynchronous version of the A2C algorithm was developed, named A3C[14]. It runs multiple environments in parallel then collects the data and performs gradient ascent the aggregated data. The actor and critic networks share the top part of the network and are then outputted individually with different activation functions, a softmax is used for the probabilistic policy function and linear activation is used for the value function.

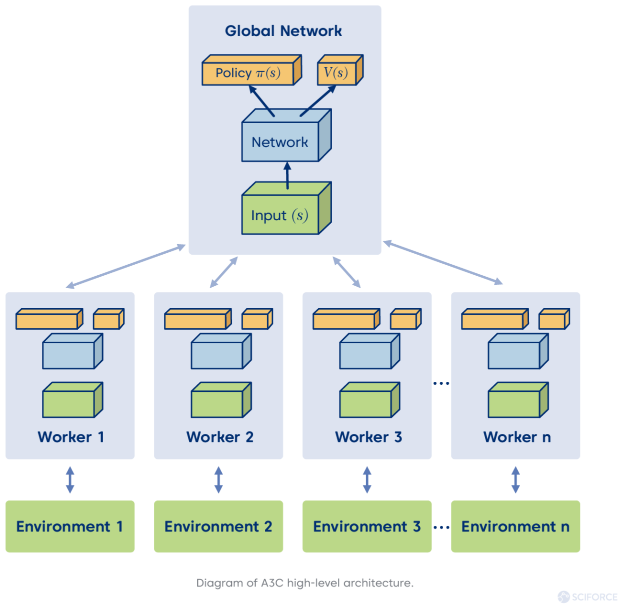

Fig. 1: A high-level representation of the AC3 architecture[13].

IV. Proximal Policy Optimisation (PPO)

Actor critic methods and DQNs are sensitive to hyper parameter adjustments and are complex. PPO was designed to overcome these problems. Using L^{PG} to perform multiple steps of optimisation on the same trajectory leads to policy updates far out of the desired region and can destroy the policy, hence a replay buffer needs to be used. PPO learns on-policy, meaning that it samples actions from the current policy. It uses a clipped surrogate objective function, $L^{CLIP}(\theta)$ to reduce the instabilities found in policy gradient methods.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1{-}\epsilon, 1{+}\epsilon)\hat{A}_t \quad (11)$$

$r_t(\theta)$ is the probability ratio [15], if the new action is has an higher probability under the new policy than the old it will be larger than 1, if less likely it will be between 0 and 1.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (12)$$

The clip function in the second term, modifies the surrogate objective function by clipping the probability ratio r_t, aiming to keep it within the range of $[1-\epsilon, 1+\epsilon]$ where ϵ is a hyper parameter, in the paper it takes a value of $\epsilon = 0.2$. In the left image the as the probability ratio increases to $1{+}\epsilon$ it is clipped at that value, note that the advantage from the action taken is positive so we limit the effect of the gradient update. On the right the action reduces the advantage. Where the probability ratio is low it is clipped at $1 - \epsilon$ to prevent the probabilities of these actions being reduced to 0, the temporal difference error that is used as the unbiased estimator for the advantage function is noisy, so it is not desirable to significantly change the gradient based on one estimate. Where $r_t(\theta)$ is high and the advantage decreases, it is not clipped. The minimum is then taken of the clipped and un-clipped value, with the final objective being a lower bound on the unclipped objective.

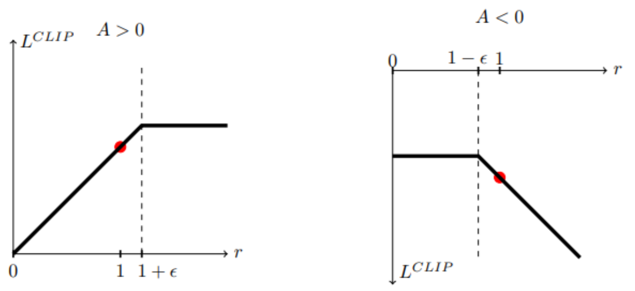

Fig. 2: Two graphs demonstrating the gradient clipping. The red dot indicates the starting point for the optimisation, $r = 1$. The right plot shows an advantage gain scenario and the left a reduction scenario, note that the function is clipped at a different value in each scenario.

With this scheme, the change in probability ratio is ignored when it would make the objective improve, and included when it makes it worse. In the final architecture the parameters are shared between networks predicting the policy and value function. As such a loss function has to be created that includes both the policy surrogate and the value function error term. The objective function is further improved with the addition of an entropy term to ensure exploration is undertaken [8]. The following function is approximately maximised

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (13)$$

Where L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{targ})^2$, $c1$ and $c2$ are coefficients and S denotes an entropy bonus.

V. Conclusion

To conclude, three advanced solutions to deep RL have been presented, each with unique adaptions of the simple principles that underpin RL. DQNs use a deep neural network to parameterise the Q function and use a replay buffer to learn off-policy. A2C uses the actor critic framework with two networks running in simultaneously, one predicting the value function and one the policy. AC3 uses multiprocessing to generate the input data in parallel removing the need for a replay buffer. PPO uses a clipped surrogate objective function to reduce the instability in training. AC3 and PPO are suitable for continuous environments, DQNs are limited to discrete environments only.

References

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” 2021.

[2] Z. Zhang, S. Zohren, and S. Roberts, “Deep reinforcement learning for trading,” 2019.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 2013.

[4] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes.* Berlin, Heidelberg:

Springer Berlin Heidelberg, 2012, pp. 3–42. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3$_1$

[5] R. Sutton, A. Barto, and R. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 19–22, 1992.

[6] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503 – 515, 1954. [Online]. Available: https://doi.org/

[7] W. Christopher, "Technical note: Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279 – 292, 1992.

[8] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, p. 229–256, 1992.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, p. 1057–1063.

[10] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and A. Peters, "A review of deep learning with special emphasis on architectures, applications and recent trends," *Knowledge-Based Systems*, vol. 194, p. 105596, 2020.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[13] Sciforce, "Reinforcement learning and asynchronous actor-critic agent (a3c) algorithm, explained," Mar 2021. [Online]. Available: https://medium.com/sciforce/reinforcement-learning-and-asynchronous-actor-critic-agent-a3c-algorithm-explained-f0f3146a14ab

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.