MATHEMATICS OF REINFORCEMENT LEARNING WITH APPLICATIONS TO QUANTITATIVE FINANCE

A PREPRINT

Yuanhao JIANG*
School of Mathematics
The University of Edinburgh
Edinburgh, UK
s2132254@ed.ac.uk

August 30, 2022

ABSTRACT

This paper provides a new way of pricing in the car insurance industry, which is, using reinforcement learning algorithms to train pricing agents to promote proper price, such that our customer is most likely to purchase our product, and we also get the highest profit and a better portfolio. We first introduce general reinforcement learning knowledge, including how we model the interaction between agent and environment using Markov decision process, and the mathematics behind those methods to solve the reinforcement learning problem, especially the Actor-Critic algorithm. Then we start building the environment, and use artificial neural networks as our pricing models, and writing algorithms that are specific to our financial problem. In the end, we will compare the Actor-Critic algorithms with other popular algorithms including REINFORCE and PPO in terms of their rate of convergence to a stable reward.

Keywords Reinforcement Learning · Artificial Neural Networks · Quantitative Finance

Contents

*Current student at the University of Edinburgh.

1 Introduction

Modern way of calculating car insurance premium is by estimating how much risk the customer poses to the insurance provider, and more risk results in higher premium. The estimation of the risk for a customer is usually done by comparing various risk factors with the statistics of the database collected so far. Those factors might include customer's age, address, occupation, car's mileage, brand and model, and so on. For example, younger drivers pose higher risks since they are more likely to have an accident due to the lack of driving experience compare to elder drivers.

In this paper, instead of using the above method to promote car insurance, we use RL methods to train agents that will interact with the market and response with price. More specifically, we consider the insurance promoting scenario as markov decision process, with the current state being the current customer features and current portfolio. At each time step, the agent promote a price according to the current customer features and the current portfolio, after that the customer resopnse to the promotion. The reward is then given besed on the customer response and the portfolio. Then the portfolio updates and a new customer arrives. This set up will help us to solve the poblem in the context of reinforcement learning, further details are discussed in the remaining sections.

2 Reinforcement Learning

The reinforcement learning method we want to use is based on the Markov decision process (MDP) [1,2], as illustrated in Figure 1.

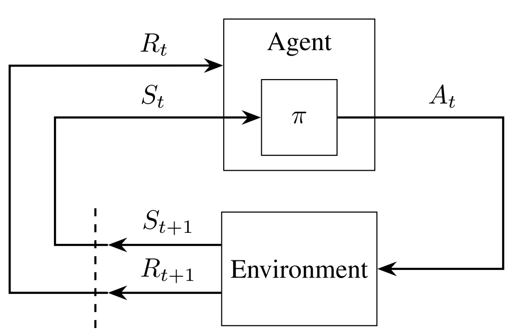

Figure 1: MDP Illustration [1]

However, in RL, the agent is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Our goal at each time step is to find a policy π to maximize the expected reward alonge the process afterwards, which is given by the value function

$$V(s) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \,\middle|\, S_t = s\right]$$

To achieve this, we need to evaluate the current policy, and improve it accordingly.

2.1 Policy Evaluation

We want to evaluate the value function under a given policy π. According to Bellman equation,

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

We can use it to iteratively compute v_π. The iteratively update is

$$v_{k+1}(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right]$$

which is proven that it will converge to the true value given any initial value function.

2.2 Policy Improvement

Now we have the evaluated value function for the policy, we want to improve it accordingly. To do this, we introduce the q-function under policy π [1]

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right]$$

The policy improvement theorem says that, let π and π' be any pair of policy, for all s we have

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \;\Rightarrow\; v_{\pi'}(s) \geq v_\pi(s)$$

Now we can improve the policy by

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s,a)$$

for all s.

2.3 Policy Iteration

By iteratively applying policy evaluation and policy improvement, we can obtain a sequence of monotonically improving policies and value functions [1] :

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement. This way of finding an optimal policy is called policy iteration.

2.4 Value Iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which itself is a protracted iterative computation requiring multiple sweeps through the state set.

We now truncate the policy evaluation such that it is stopped after just one sweep [1] (one update of each state):

$$\begin{aligned} v_{k+1}(s) &= \max_a q(s,a) \\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right] \end{aligned}$$

We keep iterate until v converges (to v_*), then we record the last policy $\pi(s) = \operatorname*{argmax}_a q(s,a)$.

2.5 Generalized Policy Iteration

As long as both processes, policy evaluation and policy improvement, continue to update all states, the ultimate result is typically the same — converge to the optimal value function and an optimal policy [1].

The general idea of alternating policy evaluation and policy improvement, independent of the frequency and convergence of the two processes [1] (instead of letting each complete before the other begins) is called Generalized Policy Iteration (GPI).

2.6 Temporal-Difference Learning

When we don't have the complete model of the environment, we need to estimate the value function from only experience. There are two methods to do so: Monte Carlo Methods and Temporal-Difference Method [1–3]. We will use Temporal-Difference method (more specifically, the TD(0) method, which converges to the maximum likelihood estimate of the function we estimate [3]) in this project, since the Monte Carlo methods might have higher variance than Temporal-Difference method [4–6].

To estimate $v_\pi(S_t)$ using only the experience, we take the average on all experienced trajectory' values. Temporal-Difference method uses $G_i = [R_{t+1}]_i + \gamma[v_\pi(S_{t+1})]_i$ to estimate the value of the ith trajectory starting at S_t, with v_π being the estimated value function at the ith time S_t is visitied, which can be initialized to anything at start. Now we have the incremental formula to update our value function:

$$\begin{aligned}
[v_\pi(S_t)]_{k+1} =& \frac{1}{k}\sum_{i=1}^{k} G_i \\
=& \frac{1}{k}\left(G_k + \sum_{i=1}^{k-1} G_i\right) \\
=& \frac{1}{k}\left(G_k + (k-1)[v_\pi(S_t)]_k\right) \\
=& [v_\pi(S_t)]_k + \frac{1}{k}(G_k - [v_\pi(S_t)]_k) \\
=& [v_\pi(S_t)]_k + \frac{1}{k}([R_{t+1}]_k + \gamma[v_\pi(S_{t+1})]_k - [v_\pi(S_t)]_k)
\end{aligned}$$

To simplify:

$$v_\pi(S_t) \longleftarrow v_\pi(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)]$$

with α being the stepsize. And this also applies to the q function:

$$q_\pi(S_t, A_t) \longleftarrow q_\pi(S_t, A_t) + \alpha[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)]$$

2.7 Policy Gradient Methods

When the state space is arbitrarily large, we cannot find an optimal policy or the optimal value function given the limited resources and time. Instead of keeping track of the value and the action(s) to select for each state (tabular methods), we can use parameterized functions, for example, artificial neural networks (ANNs), to approximate the value function and the policy function.

Previously methods learned the values of actions and then select actions based on their estimated action values (q function). Now we consider methods that instead learn a parameterized policy that select actions without consulting a value function. A value function may still be used to learn the policy parameter, but is not required for action selection [1].

We use $\boldsymbol{\theta}$ and $\boldsymbol{w}$ to denotes the policy's parameter vector and value function's weight vector, respectively. Then we have our policy $\pi(a \mid s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ and value function $v_\pi(s, \boldsymbol{w})$. We also define the scalar performance measure (objective) to be

$$J(\boldsymbol{\theta}) = \begin{cases} v_{\pi_{\boldsymbol{\theta}}}(s_0); & \text{episodic case} \\ \lim_{t\to\infty} \mathbb{E}\left[R_t \mid S_0, A_{0:t-1} \sim \pi\right]; & \text{continuing case} \end{cases}$$

Policy gradient methods seeks to maximize this performance measure, so their updates approximate gradient ascent in J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}$$

where $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is a stochastic estimate whose expection approximates the gradient of the performance measure w.r.t $\boldsymbol{\theta}_t$.

2.7.1 The Policy Gradient Theorem

The policy gradient theorem [7] says

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla\pi(a|s, \boldsymbol{\theta})$$

where μ is the stationary distribution of the state. Thus we can derive

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi\left(a|s,\boldsymbol{\theta}\right) \\
&= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\nabla\pi\left(a|S_t,\boldsymbol{\theta}\right)\right] \\
&= \mathbb{E}_\pi\left[\sum_a \pi\left(a|S_t,\boldsymbol{\theta}\right) q_\pi(S_t,a)\frac{\nabla\pi\left(a|S_t,\boldsymbol{\theta}\right)}{\pi\left(a|S_t,\boldsymbol{\theta}\right)}\right] \\
&= \mathbb{E}_\pi\left[q_\pi(S_t,A_t)\frac{\nabla\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}\right]
\end{aligned}$$

The policy gradient theorem can be generalized to include a comparison of the action value to an arbitrary baseline $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla \pi\left(a|s,\boldsymbol{\theta}\right)$$

with the same derivation we have

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi\left[\Big(q_\pi(S_t,A_t) - b(S_t)\Big)\frac{\nabla\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}\right]$$

And the discounted version of above is:

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi\left[\gamma^t\Big(q_\pi(S_t,A_t) - b(S_t)\Big)\frac{\nabla\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}\right]$$

One natural choice for baseline is an estimate of the state value, $\hat{v}(S_t,\boldsymbol{w})$, where $\boldsymbol{w}$ is the weight vector to be learned.

2.7.2 Actor-Critic Methods

Recall the TD method, we make use of $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right]$, by taking one sample estimate, and choosing $\hat{v}(S_t,\boldsymbol{w})$ as baseline [1], we have

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi\left[\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w})\Big)\frac{\nabla\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)}\right]$$

Again, by taking one sample estimate we have the update formula for $\boldsymbol{\theta}$:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}}\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w})\Big)\frac{\nabla\pi\left(A_t|S_t,\boldsymbol{\theta}_t\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}_t\right)} \\
&= \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}}\delta_t\nabla\ln\pi\left(A_t|S_t,\boldsymbol{\theta}_t\right)
\end{aligned}$$

As for the weight vector $\boldsymbol{w}$ for the estimated value function, at each time step t, we want to update it by minimizing the squared error, $\left[v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w}_t)\right]^2$:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha'_{\boldsymbol{w}}\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w}_t)\Big]^2 \\
&= \boldsymbol{w}_t - \alpha_{\boldsymbol{w}}\Big(v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w}_t)\Big)\nabla\hat{v}(S_t,\boldsymbol{w}_t)
\end{aligned}$$

where v_π is the true value function. Again, since $v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$, we take one sample estimate, which gives us the update formula for $\boldsymbol{w}$:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha_{\boldsymbol{w}}\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w}_t)\Big)\nabla\hat{v}(S_t,\boldsymbol{w}_t) \\
&= \boldsymbol{w}_t - \alpha_{\boldsymbol{w}}\delta_t\nabla\hat{v}(S_t,\boldsymbol{w}_t)
\end{aligned}$$

The actor-critic method based on the above updates formulas, updating policy parameter $\boldsymbol{\theta}$ and value function weight $\boldsymbol{w}$ simultaneously at each time step, is called two time-scale (natural) actor-critic algorithms [4, 8], details of the algorithm itself are shown in Algorithm 1 [1].

Algorithm 1 Actor-Critic

```
Input: a differentiable policy parameterization π(a|s, θ)
Input: a differentiable state-value function parameterization v̂(s, w)
Step sizes α_θ > 0, α_w > 0
θ ← 0, w ← 0
loop (for each episode)
    Initialize S (first state of the episode)
    loop (for each time step i)
        A ~ π(·|S, θ)
        Take action A, observe S′, R
        δ ← R + γ v̂(S′, w) − v̂(S, w)
        w ← w + α_w δ ∇v̂(S, w)
        θ ← θ + α_θ γ^i δ ∇ ln π(A|S, θ)
        S ← S′
    end loop
end loop
```

Table 1: Customer Feature Vector Description

gender	age	car cost	miles	brand	random feature 0	random feature 1	...
1.0	42.723	94032.096	30086.311	29.264	8.0	68.740	...

3 Build the Environment

The environment does the following things:

1. record the current state that can be queried anytime
2. when an action, i.e. a price, is given according to the current state, it outputs the reward, and goes to next state

At each time step, the state consists of a randomly generated customer's features and the current portfolio. After the agent performs an action, i.e. promotes a price to the customer, the reward is then computed based on the profit on this customer and the portfolio up to now.

3.1 Customer Features

A customer's features is represented by an array (or vector), denoted by x with 16 entries with each generated randomly from a distribution, including normal distribution, binomial distribution, and so on. For example, a feature vector x is described as Table 1

After a price, denoted by c, is promoted to a customer, the customer will give a resopnse, denoted by r, which is 0 if he does not buy the product, and 1 otherwise. And the profit we have for that customer, denoted by pf, can be computed by

$$\text{pf} = r * (c - g(x))$$

where $g(x)$ is the cost on that customer.

3.2 Resopnse

The resopnse is generated by a generalized linear model, whose input is a vector of a customer features vector x appended with a price c, i.e. $r = glm(x[0], \ldots, x[15], c)$.

The GLM used in the project is fitted using a randomly generated data set with each observation is a customer's features followed by a price, then followed by a resopnse, as illustrated by Table 2.

And the link function for the GLM is chosen to be the logistic function.

Table 2: Customer Resopnse Examples

	customer feature 0	...	customer feature 15	price	resopnse
1	1.0	...	2.0	1109.504	1.0
2	0.0	...	1.0	665.8	0.0
...	...	...	...	...	...

3.3 State

The state at start of each time step is the feature vector of the new customer to promote (might be modified a bit), appended by the portfolio, which is also a data vector consists of the following entries:

- average profit over all customers so far, expect for the new customer at current time step, denoted by avg_pf (initialized to 0), which is computed by
$$\text{avg\_pf}_t = \frac{1}{t}\sum_{i=0}^{t-1} \text{pf}_i,\ t > 0$$
- the portion of the buyers over all customers, expect for the new one, denoted by p (initialized to 0), which is computed by
$$\text{p}_t = \frac{1}{t}\sum_{i=0}^{t-1} r_i,\ t > 0$$
- the variance of the portion of buyers in their categories, denoted by var: we divide customers (expect for the new one at current time step) into 4 categories, according to their features, and for each category, we compute the portion of the customers who buy our product, denoted by pt_i with $i = 1, 2, 3, 4$, then we compute the variance of the 4 portions:
$$\text{var} = \text{Var(pt)} = \frac{\sum_{i=1}^{4}(\text{pt}_i - \bar{\text{pt}})^2}{3}$$
Later we use this to compute reward to make sure the agent will promote to all types of customers instead of only focusing on one type. Note that var ≤ 1.

Thus, the state s, or s_t, end up with the form
$$s_t = (x'_t[0],\ \dots,\ x'_t[-1],\ \text{avg\_pf}_t,\ \text{p}_t,\ \text{var}_t)$$
where x'_t is the modified x_t, including preprocess of converting some categorical data to meaningful data to the model.

3.4 Reward

The reward at each time step t, after the price c_t is promoted to the customer x_t, denoted by R_t, is calculated by
$$R_t = \text{pf}_t\,(1 - h(\text{var}_t))$$
where h is a function that indicates how much we care about the variance, i.e. how important it is to try to promote to customers of all categories. In this project I choose h to be such that $h(\text{var}) = \sqrt{\text{var}}/2$.

4 Build the model

With the environment well built, we now build the model for our policy and the value function. In this project we use artificial neuro networks (ANNs) to parameterize our policy and value function as it suits well on algorithms like Actor-Critic, REINFORCE, PPO and so on (for details see [9,10]). Each of our ANNs has three linear layers, with the ReLU function as the activation function. And the input of both function is the state, as illustrated in Figure 2. where $n(S)$ denotes the vector length of the state and $n(A)$ denotes the number of actions.

We generate the policy distribution by scoring each action. E.g. denoted the policy function output by y, then the probability of choosing action i ($i = 0, 1, ..., n(A) - 1$) is computed by
$$P(i) = \frac{y_i}{\sum_{j=0}^{n(A)-1} y_j}$$

$$S_t \to \boxed{\underset{n(S)\to 128}{\text{Linear}} \to \text{ReLU} \to \underset{128\to 128}{\text{Linear}} \to \text{ReLU} \to \underset{128\to 1}{\text{Linear}}} \to v_t$$

ANN for value function

$$S_t \to \boxed{\underset{n(S)\to 128}{\text{Linear}} \to \text{ReLU} \to \underset{128\to 128}{\text{Linear}} \to \text{ReLU} \to \underset{128\to n(A)}{\text{Linear}} \to \underset{(0.1,\,0.1)}{\text{Threshold}}} \to \pi_t \to c_t$$

ANN for policy function

Figure 2: ANN model for policy and value function

Note that we add a Threshold layer at the end of policy function, which is defined by

$$\text{Threshold}(0.1, 0.1) = \max(0.1, x)$$

which makes sure that

1. the probabilities are valid (no negative values, greater score results in greater probability);
2. all actions have the probability to be chosen at early stage, which is beneficial for training since it increase the exploration for different actions.

5 RL with Actor-Critic

In this project, we consider the episodic learning procedure, where each episode has finite time steps, T, to be specified as a constant, and we will also reset the environment at the start of each episode.

5.1 The Algorithm

Algorithm 2 uses (two time-scale) actor-critic method [1, 4, 8], and with the help of what we have built in previous sections (especially subsection 2.7.2), to solve the problem. And Figure 3 illustrates the algorithm.

Algorithm 2 Actor-Critic for financial problem

```
Initialize ANN policy parameterization π(c|s, θ), with any θ
Initialize ANN state-value function parameterization v̂(s, w) with any w.
Step sizes α_θ > 0, α_w > 0
loop (for each episode)
    Reset the environment
    Generate an x (first customer of the episode)
    Initialize S (first state of the episode)
    for t = 0, 1, ..., T do
        c ~ π(·|S, θ)
        Take action c, observe S', R from environment
        δ ← R + γ v̂(S', w) − v̂(S, w)
        w ← w + α_w δ ∇v̂(S, w)
        θ ← θ + α_θ γ^t δ ∇ ln π(A|S, θ)
        S ← S'
    end for
end loop
```

5.2 Training Result

Figure 4 shows the training result of Algorithm 1 (300 moving average reward vs iteration), with 600 iterations (5 episodes per iteration), $T = 300$, $\gamma = 0.99$, $\alpha_\theta = \alpha_{\boldsymbol{w}} = 3 \times 10^{-4}$. The graph shows that the moving average reward converges within only 50 iterations, that is, 250 episode, 75000 parameter updates for each network, and stays quite stable afterwards.

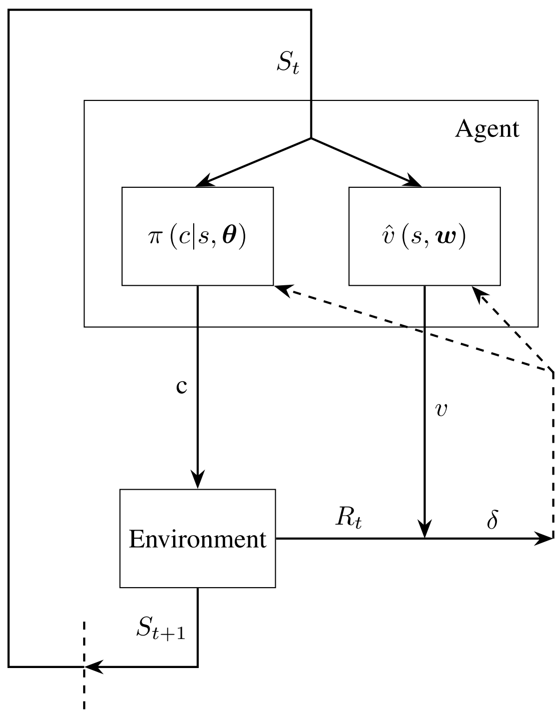

Figure 3: Actor-Critic for Finance Algorithm Illustration

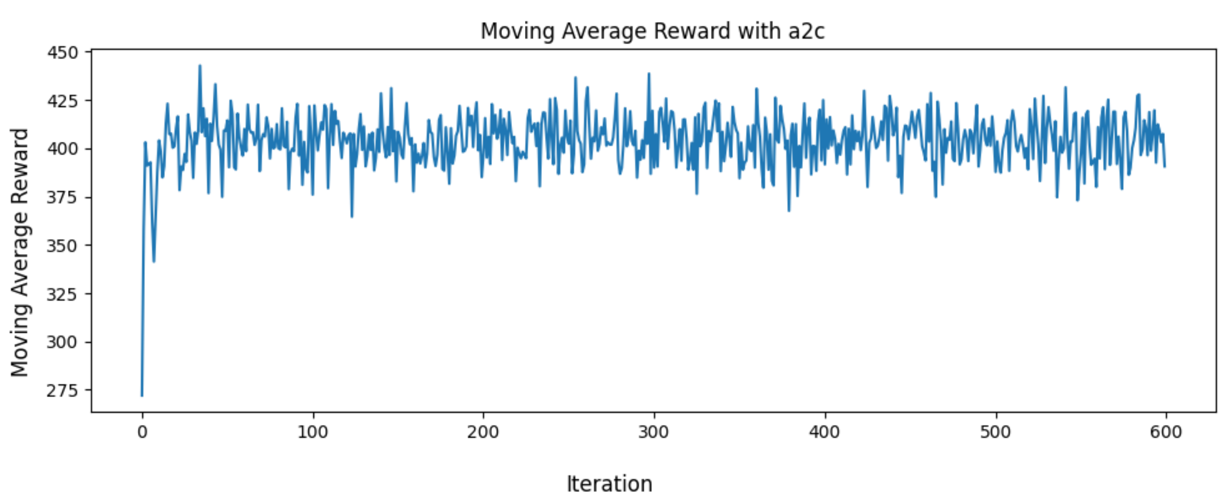

Figure 4: Training Result for Algorithm a2c

6 Actor-Critic, Reinforce, and PPO

6.1 Algorithms

To further show how Actor-Critic algorithm performs, we compare it to another two popular algorithms: REINFORCE [11] (Algorithm 3) and PPO [12] (Algorithm 4). The policy function of the two algorithms is implemented the same way as we did in Actor-Critic algorithm, and the same is true for the value function used in PPO method.

Algorithm 3 Reinforce for financial problem [1]

Initialize ANN policy parameterization $\pi\left(c|s,\boldsymbol{\theta}\right)$, with any $\boldsymbol{\theta}$
Step sizes $\alpha_{\boldsymbol{\theta}} > 0$
loop (for each episode)
  Reset the environment
  Generate an episode $S_0, C_0, R_1, ..., S_{T-1}, C_{T-1}, R_T$, following $\pi\left(\cdot|\cdot,\boldsymbol{\theta}\right)$
  for $t = 0, 1, ..., T$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^t G \nabla \ln \pi\left(C_t|S_t,\boldsymbol{\theta}\right)$
  end for
end loop

Algorithm 4 PPO for financial problem [13]

Initialize ANN policy parameterization $\pi\left(c|s,\boldsymbol{\theta}\right)$, with any $\boldsymbol{\theta}$
Initialize ANN state-value function parameterization $\hat{v}(s,\boldsymbol{w})$ with any $\boldsymbol{w}$.
Step sizes $\alpha_{\boldsymbol{\theta}} > 0$, $\alpha_{\boldsymbol{w}} > 0$
for iteration $k = 0, 1, 2, ...$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$, following $\pi\left(\cdot|\cdot,\boldsymbol{\theta}_k\right)$, where $\tau = S_0, C_0, R_1, ..., S_{T-1}, C_{T-1}, R_T$
  for $t = 0, 1, ..., T$ in each trajectory τ in $\mathcal{D}_k$ **do**
    $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $A_t \leftarrow G_t - \hat{v}(S_t)$
  end for
  loop (update multiple times)
    $\boldsymbol{\theta} \leftarrow \arg\max\limits_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi(c_t|S_t,\boldsymbol{\theta})}{\pi(c_t|S_t,\boldsymbol{\theta}_k)} A_t, \operatorname{clip}\left(\frac{\pi(c_t|S_t,\boldsymbol{\theta})}{\pi(c_t|S_t,\boldsymbol{\theta}_k)}, 1-\epsilon, 1+\epsilon\right) A_t\right)$
    $\boldsymbol{w} \leftarrow \arg\min\limits_{\boldsymbol{w}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \Big(\hat{v}(S_t,\boldsymbol{w}) - G_t\Big)^2$
  end loop
end for

6.2 Training result

Figure 5 shows the training result of these three algorithms (300 moving average reward vs iteration), with 600 iterations (5 episodes per iteration), $T = 300$, $\gamma = 0.99$, $\alpha_\theta = 3 \times 10^{-4}$, $\alpha_{\boldsymbol{w}} = 3 \times 10^{-10}$ (if applicable), $\epsilon = 0.5$ (if applicable). As we can see, the Actor-Critic algorithm converges the fastest, within 50 iterations, followed by REINFORCE algorithm, which converges within 250 iterations, despite a small drop of moving average around iteration 300, which is quickly corrected by the algorithm within 50 iterations. PPO algorithm converges the slowest, with about 300 iterations.

7 Conclusion

To summarize, the moving average rewards after convergence of these three algorithms are nearly the same (around 400), which is as expected. The Actor-Critic algorithm performs really well compared to anther two algorithms, it has highest converge rate, the reward is also very stable after convergence. In the mean time, the slow convergence for PPO algorithm is expected since it uses a clip function to prevent model from changing too fast in a single update [12, 13], although at current stage it seems to be unnecessary for our specific problem setup. Further investigation needs to be done for the unexpected small drop of the reward for the REINFORCE algorithm.

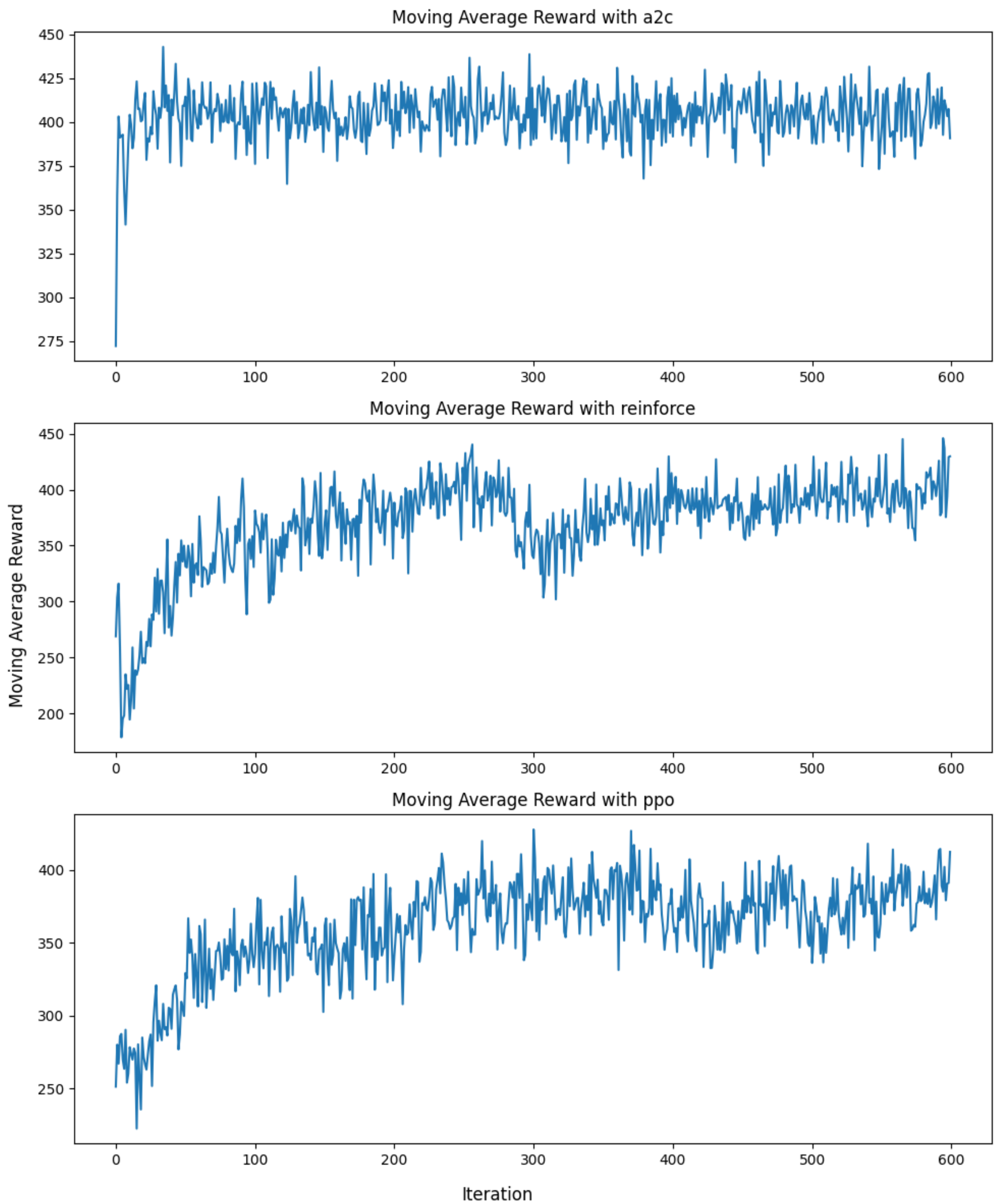

Figure 5: Training Result Comparison for Three Algorithm

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

[3] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[4] Ben Hambly, Renyuan Xu, and Huining Yang. Recent advances in reinforcement learning in finance. *arXiv preprint arXiv:2112.04553*, 2021.

[5] Vincent François-Lavet, Guillaume Rabusseau, Joelle Pineau, Damien Ernst, and Raphael Fonteneau. On overfitting and asymptotic bias in batch reinforcement learning with partial observability. *Journal of Artificial Intelligence Research*, 65:1–30, 2019.

[6] Ulrike Von Luxburg and Bernhard Schölkopf. Statistical learning theory: Models, concepts, and results. In *Handbook of the History of Logic*, volume 10, pages 651–706. Elsevier, 2011.

[7] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[8] Tengyu Xu, Zhe Wang, and Yingbin Liang. Non-asymptotic convergence analysis of two time-scale (natural) actor-critic algorithms. *arXiv preprint arXiv:2005.03557*, 2020.

[9] Lingxiao Wang, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural policy gradient methods: Global optimality and rates of convergence. *arXiv preprint arXiv:1909.01150*, 2019.

[10] Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. *Advances in neural information processing systems*, 32, 2019.

[11] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.