BSMRL: Estimating with Differential Equations (Draft ver.)

Anonymous authors
Paper under double-blind review

Abstract

In this paper, we propose BSMRL, a novel reinforcement learning framework that leverages the Black-Scholes-Merton (BSM) model for reward shaping. By integrating financial mathematics into the RL paradigm, BSMRL aims to enhance learning efficiency and stability in environments with sparse and delayed rewards. We further extend our approach by incorporating the Merton Jump-Diffusion Model to account for sudden changes in the environment, providing a more robust framework for real-world applications. Experimental results demonstrate that BSMRL outperforms traditional RL methods in various benchmark tasks, showcasing its potential for broader applications in software testing, quantitative finance, embodied AI, and beyond.

Contents

1 Introduction

2 Related Works

3 BSMRL

3.1 Black-Scholes-Merton Model

In the field of financial mathematics, the Black-Scholes-Merton (BSM) model**??** is a foundational framework for (European) option pricing. It assumes that the price of the underlying asset follows a geometric Brownian motion with constant volatility and drift. The BSM model provides a closed-form solution for European-style options.

They derived a partial differential equation (PDE) that the option price must satisfy, known as the Black-Scholes equation.

Theorem 3.1 (Black-Scholes Equation)**.** *The price of a European call option $C(S,t)$ on a non-dividend-paying stock satisfies the following PDE:*

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0 \tag{1}$$

where $C(S,t)$ is the price of the option at time t when the underlying asset price is S, σ is the volatility of the underlying asset, and r is the risk-free interest rate.

By applying Ito's Lemma and constructing a riskless portfolio, they eliminated the stochastic component and derived the pricing formula for European call options.

Theorem 3.2 (Black-Scholes-Merton Formula)**.** *The price of a European call option $C(S_t,t)$ on a non-dividend-paying stock is given by:*

$$C(S_t,t) = S_t\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2) \tag{2}$$

where:

$$d_1 = \frac{\ln(S_t/K) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t} \tag{3}$$

Here, $C(S_t,t)$ is the price of a European call option at time t, S_t is the current price of the underlying asset, K is the strike price, r is the risk-free interest rate, σ is the volatility of the underlying asset, T is the time to maturity, and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.

Rationale In Reinforcement Learning, the agent interacts with an environment to maximize cumulative rewards. However, in real-world scenarios, rewards can be sparse and delayed, making it challenging for agents to learn effective policies, leading to high variance in value estimates and slow convergence. To address this, we propose using the BSM model to shape rewards based on the agent's current state and time-to-go, providing more informative feedback and guiding the agent's learning process.

In a RL task, we have to estimate the value function $V(s)$ or action-value function $Q(s,a)$, usually within a certain period of time. For instance, in Monte-Carlo methods, we estimate the expected return over an episode, while in Temporal-Difference (TD) learning, we estimate the value function based on one-step transitions. This is analogous to option pricing, where the option's value depends on the underlying asset price and time to maturity. These method often suffer from slow convergence rate due to sparse rewards, e.g., in website bug mining, the agent only receives a reward when a bug is found, which may take a long time.

We assume that the noise in the environment follows a log-normal distribution, similar to asset price movements in financial markets. By mapping the agent's state to a proxy asset price and time-to-go to time-to-maturity, we can compute a potential function using the BSM formula. Thus, though the environment may not provide frequent rewards, the agent can still receive continuous feedback through the potential-based shaping rewards derived from the BSM model, which helps reduce variance and accelerates learning.

3.2 Merton Jump Model

In financial markets, asset prices often exhibit sudden and significant changes, known as jumps, which cannot be captured by the standard Black-Scholes model. To address this limitation, Robert C. Merton extended the Black-Scholes framework by incorporating jump processes into the asset price dynamics, leading to the Merton Jump-Diffusion Model. The Merton Jump-Diffusion Model assumes that the underlying asset price follows a stochastic process that combines both continuous diffusion and discrete jumps. The asset price dynamics under the Merton model can be described by the following stochastic differential equation (SDE):

$$dS_t = \mu S_t dt + \sigma S_t dW_t + J_t S_t dN_t \tag{4}$$

where:

- S_t is the asset price at time t.
- μ is the drift rate of the asset price.
- σ is the volatility of the continuous component.
- W_t is a standard Brownian motion.
- N_t is a Poisson process with intensity λ, representing the number of jumps up to time t.
- J_t is the jump size, typically modeled as a log-normal random variable.

Remark 3.1. *Such model can be also viewed as a Levy process, which generalizes Brownian motion by allowing for jumps.*

The Merton Jump-Diffusion Model leads to a modified option pricing formula that accounts for the possibility of jumps in the underlying asset price.

Theorem 3.3 (Merton Jump-Diffusion Option Pricing Formula)**.** *The price of a European call option $C(S_t, t)$ under the Merton Jump-Diffusion Model is given by:*

$$C(S_t, t) = \sum_{n=0}^{\infty} \frac{e^{-\lambda(T-t)}(\lambda(T-t))^n}{n!} C_{BS}(S_t, t; \sigma_n) \tag{5}$$

where:

- $C_{BS}(S_t, t; \sigma_n)$ *is the Black-Scholes price of the option with adjusted volatility* $\sigma_n = \sqrt{\sigma^2 + \frac{n\delta^2}{T-t}}$*, where* δ *is the standard deviation of the jump size.*
- λ *is the jump intensity.*
- T *is the time to maturity.*
- n *is the number of jumps.*

Rationale While the Black-Scholes model assumes continuous price movements, real-world environments often exhibit sudden changes or jumps, leading to fat-tailed reward distributions.

To better capture these dynamics, we propose using the Merton Jump-Diffusion Model for reward shaping in Reinforcement Learning. By incorporating jump processes, the Merton model provides a more accurate representation of environments with abrupt changes.

3.3 Adapted Decaying for Poisson Jump Intensity

We have assume that the jump intensity λ is constant in the Merton model. However, such assumption may lead to the overestimation of jump effects, resulting in not convergent value estimates. To address this, we introduce an adapted decaying mechanism for the jump intensity λ.

We use the random counting process $N(t)$ to model the number of jumps occurring up to time t. When a significant jump is detected in the reward signal, we increase the jump intensity λ to reflect the higher likelihood of future jumps. As the $N(t) \sim P(\lambda)$, then $t \sim \Gamma(n, \lambda)$, where n is the number of the observed jumps. After time T, where the option is bound to expire, we may count there are how many jumps with intensity λ. If there are too less jumps, then it means that the jump intensity is overestimated, thus we decay the λ accordingly. Specifically, we update λ as follows:

$$\lambda \leftarrow \lambda \cdot \exp(-\beta \cdot (N(T) - \lambda T)) \tag{6}$$

And we sample the observed time period T from $\Gamma(1, \lambda)$.

4 Algorithm Framework

The overall algorithm framework for BSMRL using Merton Potential Shaping is outlined in Algorithm 1. We introduce a method from **?**, whic allows us to decompose the reward into a predictable component and a chaotic component, thus estimate the volatility rate of the chaotic component, which will be used to compute the Merton potential shaping reward.

Algorithm 1 BSMRL

Hyperparameters: Learning rate α, Discount factor γ, Moving average rate η (for statistics), Maturity T, Risk-free rate r, Jump mean μ_J, Jump std δ_J, Expansion terms K.

Initialize:

$Q(s, a)$ arbitrarily.

$\bar{R}(s, a) \leftarrow 0$ ▷ Predictable Reward Component

$\Sigma^2_{chaos}(s, a) \leftarrow 0$ ▷ Chaotic Variance (Doob Volatility) [cite: 139]

$\lambda_{jump}(s, a) \leftarrow \lambda_{init}$ ▷ Estimated Jump Intensity for Levy Process

Function BlackScholes($S, K_{strike}, T, r, \sigma$):

$d_1 \leftarrow \frac{\ln(S/K_{strike}) + (r+0.5\sigma^2)T}{\sigma\sqrt{T}}$

$d_2 \leftarrow d_1 - \sigma\sqrt{T}$

return $S \cdot \Phi(d_1) - K_{strike}e^{-rT} \cdot \Phi(d_2)$

End Function

Function MertonPrice($S_0, T, r, \sigma_{chaos}, \lambda, \mu_J, \delta_J$):

$Price \leftarrow 0$

$k \leftarrow e^{\mu_J + 0.5\delta_J^2} - 1$ ▷ Expected jump size

$\lambda' \leftarrow \lambda(1 + k)$

for $n = 0$ to K **do**

    $w_n \leftarrow \frac{e^{-\lambda' T}(\lambda' T)^n}{n!}$ ▷ Poisson weight for n jumps

    $\sigma_n \leftarrow \sqrt{\sigma^2_{chaos} + \frac{n\delta_J^2}{T}}$ ▷ Effective volatility blending Doob noise & Jumps

    $r_n \leftarrow r - \lambda k + \frac{n\ln(1+k)}{T}$

    $Val_n \leftarrow$ BlackScholes($S_0, S_0, T, r_n, \sigma_n$) ▷ ATM Call Option as Enhanced Value

    $Price \leftarrow Price + w_n \cdot Val_n$

end for

return $Price$

End Function

loop

    Initialize state s

    while s is not terminal **do**

        Choose action a (e.g., ϵ-greedy based on Q)

        Take action a, observe reward R_{obs} and next state s'

        $\delta_{pred} \leftarrow R_{obs} - \bar{R}(s, a)$ ▷ Chaotic innovation

        $\bar{R}(s, a) \leftarrow \bar{R}(s, a) + \eta \cdot \delta_{pred}$ ▷ Update Predictable Component

        $\Sigma^2_{chaos}(s, a) \leftarrow (1-\eta)\Sigma^2_{chaos}(s, a) + \eta \cdot (\delta_{pred})^2$ ▷ Update Chaotic Variance

        $\lambda_{jump}(s) \leftarrow \lambda_{jump}(s)\exp(-\beta(\frac{(V_t - V_{t-T})}{\lambda_{jump}(s)} - \lambda_{jump}(s)T))$

        $T \leftarrow \Gamma(1, \lambda_{jump}(s))$ ▷ Adapted Decaying for Jump Intensity

        $a'_{best} \leftarrow \arg\max_{a'} Q(s', a')$

        $S_{underlying} \leftarrow Q(s', a'_{best})$ ▷ Underlying asset is the naive value

        $\sigma_{input} \leftarrow \sqrt{\Sigma^2_{chaos}(s', a'_{best})}$ ▷ Use Doob Volatility as input

        $V_{enhanced}(s') \leftarrow$ MertonPrice($S_{underlying}, T, r, \sigma_{input}, \lambda_{jump}(s'), \mu_J, \delta_J$)

        $Y_{target} \leftarrow R_{obs} + \gamma \cdot V_{enhanced}(s')$ ▷ Maximize Option-Adjusted Return

        $Q(s, a) \leftarrow Q(s, a) + \alpha\,(Y_{target} - Q(s, a))$

        $s \leftarrow s'$

    end while

end loop

5 Theoretical Analysis

5.1 MDP Formulation for OptionRL

5.2 Convergence Analysis

5.3 Variance Analysis

6 Discussion and Future Work

7 Conclusion

A Proofs

B Experiment Details