Solving The Dynamic Volatility Fitting Problem: A Deep Reinforcement Learning Approach

Emmanuel Gnabeyeu [1] Omar Karkar [2] Imad Idboufous [2], presented by Emmanuel Gnabeyeu

[1]Sorbonne Université, Paris, France

Presented at the International Joint Conference on Neural Networks 2025, Pontifical Gregorian University, Rome, Italy

Abstract

The volatility fitting is one of the core problems in the equity derivatives business. Through a set of deterministic rules, the degrees of freedom in the implied volatility surface encoding (parametrization, density, diffusion) are defined. Whilst very effective, this approach widespread in the industry is not natively tailored to learn from shifts in market regimes and discover unsuspected optimal behaviors. In this paper, we change the classical paradigm and adapt the latest advances in Deep Reinforcement Learning(DRL) to solve the fitting problem. In particular, we show that variant of Deep Deterministic Policy Gradient (DDPG) can achieve at least as-good as standard fitting algorithms. Furthermore, we explain why the reinforcement learning framework is appropriate to handle complex objective functions and is natively adapted for online learning.

Basics of Reinforcement Learning

Reinforcement Learning (RL) is a goal-driven learning framework where an agent learns to act optimally in a stochastic environment through interaction without explicit supervision.

1. We model the environment as a **Markov Decision Process (MDP)** $(\mathcal{S}, \mathcal{A}, r, p)$ where at each time, the agent observes a current state $s_t \in \mathcal{S} \subset \mathbb{R}^d$, takes an action $a_t \in \mathcal{A} \subset \mathbb{R}^n$ and receives a scalar reward r_t denoting numerical feedback from the stochastic scalar-valued reward function $r = r(a_t, s_t)$ and $p(s_{t+1}|s_t, a_t)$ the transition dynamics.
2. **Goal:** Find a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ (i.e. a sequence of actions over time) that maximizes the expected **discounted return** (value functions) R_t for a discounting factor $\gamma \in [0, 1]$.
3. **State-action value function** (provide a prediction of how good each state/action pair is) :

$$Q^\pi(s, a) = \mathbb{E}^\pi[R_t|s_t = s, a_t = a] = \sum_{i=t}^{T} \mathbb{E}_{(s_i,a_i)\sim\rho_\pi}[\gamma^{(i-t)} r(s_i, a_i)]. \quad (1)$$

Problem Statement In Reinforcement Learning Framework

(a) **Market Observables and Parametrisation: Let us fix a maturity T**

- **Observables:** We observe the market at time t_i as a collection $\left\{\sigma^{Ask}_{Call/Put}(t_i, \kappa_j), \sigma^{Bid}_{Call/Put}(t_i, \kappa_j)\right\}_{j\in[1,...,n]}$ of markets quotes for different moneyness $(\kappa_1, ..., \kappa_n)$. reduced via a mapping to $\left\{\sigma^{\text{Ask}}(t_i, \kappa_j), \sigma^{\text{Bid}}(t_i, \kappa_j)\right\}_{j=1}^{n}$
- **Parametrization:** The volatility slice is modeled via a function $\Psi^{\text{vol}}_{\vec{\theta}}$, such as SVI, using a parameter vector: $\vec{\theta}_{t_i} = (\theta^1_{t_i}, \ldots, \theta^K_{t_i})$. The goal at t_{i+1} is to find optimal parameters $\vec{\theta}^*_{t_{i+1}}$ that improve the fit or maximize a chosen reward, rather than only minimizing distance from mid-volatilities: $\sigma^{\text{Mid}}(t_i, \kappa_j) = \frac{\sigma^{\text{Ask}}(t_i,\kappa_j)+\sigma^{\text{Bid}}(t_i,\kappa_j)}{2}$ as classically optimized.
- **Bump vector:** the problem is to find the shifted vector $\Delta\vec{\theta}_{t_i} = (\Delta\theta^1_{t_i}, ..., \Delta\theta^K_{t_i})$ to bump the old parameters $\vec{\theta}_{t_i}$ in order to maximize our selected reward function.

(b) **State and Action Spaces:**

- Our **State** is the set of tuple consisting of the observed markets quotes and the prior volatility parameters:

$$s_{t_i} = \left(\sigma^{\text{Bid}}(t_i, \kappa_j), \sigma^{\text{Ask}}(t_i, \kappa_j), \theta^1_{t_{i-1}}, \ldots, \theta^K_{t_{i-1}}\right)_{j=1}^{n} \in \mathbb{R}^N \quad where \quad N \geq 2n + K \quad (2)$$

- **Action:** A real-valued vector in $\mathbb{R}^K$, representing the parameter bump (continuous actions spaces):

$$a_{t_{i+1}} = \Delta\vec{\theta}_{t_{i+1}} = \vec{\theta}_{t_{i+1}} - \vec{\theta}_{t_i}$$

(c) **Reward Functions:** We measure the goodness of the fit with different objective functions:

- **Mean Squared Error (MSE)** or **Vega-Weighted MSE (BMSE)**

$$\xi(\vec{\theta}_{t_i}) = \sum_{j=1}^{n} \left(\sigma^{\text{Mid}}(t_i, \kappa_j) - \Psi^{\text{vol}}_{\vec{\theta}_{t_i}}(\kappa_j)\right)^2 \quad or \quad \xi(\vec{\theta}_{t_i}) = \sum_{j=1}^{n} \text{vega}_{BS}(\kappa_j) \left(\sigma^{\text{Mid}}(t_i, \kappa_j) - \Psi^{\text{vol}}_{\vec{\theta}_{t_i}}(\kappa_j)\right)^2 \quad (3)$$

Remark: These rewards may incorporate domain-specific constraints (e.g., liquidity, stability, macro factors). The RL reward is defined as the negative error: $r(s_{t_i}, a_{t_i}) = -\xi(\vec{\theta}_{t_i})$

Training the Reinforcement Learning agent

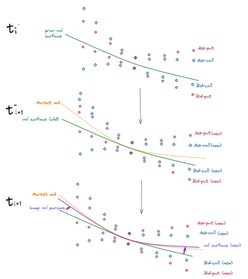


Figure 1. A snapshot of the agent acting in the market: The old volatility surface is bumped to a new one following the move of the market, conditional on prior volatility surface.

From the RL perspective, the fitting cycle looks as follows: At time t^-_{i+1},

1. The agent sees the state $s_{t_{i+1}}$ of our environment built on the market bid and ask volatilities (homogeneous variable) and the old volatility surface parameters $\vec{\theta}_{t_i}$ (endogenous variable),
2. It takes some action $a_{t_{i+1}} = \Delta\vec{\theta}_{t_{i+1}} = \theta_{t_{i+1}} - \theta_{t_i}$ on the adjustment of the old parameters $\vec{\theta}_{t_i}$, and receives some reward $r(s_{t_{i+1}}, a_{t_{i+1}} = \Delta\vec{\theta}_{t_{i+1}})$.
3. The resulting volatility slice maximizes the reward and it boils down a new state $s_{t_{i+2}}$ for the next move.

The error term between our volatility surface and the mid market is denoted $\xi(\vec{\theta}_{t_{i+1}})$.

A Neural Actor-Critic Methods for Volatility Fitting: Algorithmic Design

Our approach integrates value-based and policy-based RL methods and consider parametrized value function $Q(s, a; \theta^Q)$ and policies $\pi(s, a; \theta^\pi)$, then use Neural Networks as functional approximators: This lead to Neural Actor-Critic, a subclass of Deep Reinforcement Learning (DRL).

1. **Critic** learns the Q-function via the Bellman equation:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma Q^\pi\left(s_{t+1}, \pi^D(s_{t+1}, \theta^\pi)\right) \quad (4)$$

2. **Actor** is optimized to maximize Q-values by backpropagating through both actor and critic networks. It learns a deterministic policy $\pi^D(s_t, \theta^\pi)$, $\pi^* = \arg\max_{\pi\in\Pi} Q^\pi$.

It is a **Model-free, off-policy Actor-Critic algorithm**, like DDPG introduced by Lillicrap et al. The **exploration** is handled via noisy/stochastic actions from the policy $\pi^D(s_t, \theta^\pi)$, $a_t \sim \pi^D(s_t, \theta^\pi)+\mathcal{N}$.

- **Gaussian noise $\mathcal{N}$ with decaying variance** to reduce exploration as the agent is learning the optimal policy over time: $a_t \sim \pi^D(s_t; \theta^{(n)}) + \epsilon_t$, $\epsilon_t \sim \mathcal{N}(0, \sigma_n)$, $\sigma_n = \max(\sigma_0(1 - \frac{n}{N})^4, \sigma_{\min})$

In this setting, the state-action value becomes: $Q^\pi(s_{t_i}, a_{t_i}) = -\sum_{k=t_i}^{T} \mathbb{E}_{(s_k,a_k)\sim\rho_\pi}[\gamma^{(k-t_i)}\xi(\vec{\theta}_k)]$ (5)

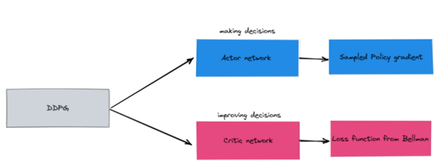


Figure 2. DDPG Framework.

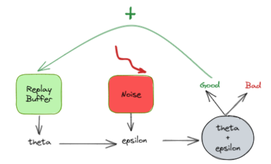


Figure 3. Replay Memory and Noise.

Results And Analysis from a static scenario

A Monte-Carlo on 5 random seeds is performed with a power decaying noise. The agent's best response (in *green*) and the mean volatility slice (in *blue*) over the last 1000 episodes are shown.

Type	Skew	High Smile	Inv. Smile
Bench	-0.011913	-0.000022	-0.000044
Seeds' Avg	-0.012145	-0.000163	-0.000315

Table 1. DDPG MSE Rewards

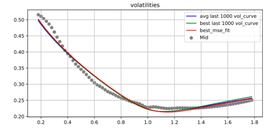


Figure 4. Implied volatility snapshot in a "Skewed" configuration:

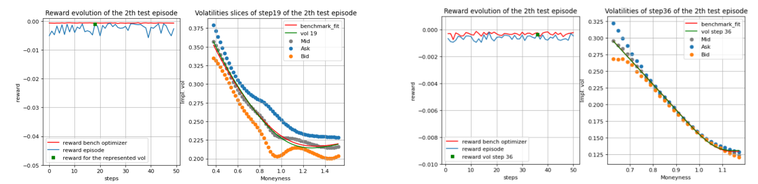
Results And Analysis from a Quasi-dynamic scenario

The agents are trained on **data generated for two stocks** (one with a wide spread and the other with a tight spread) across several episodes, each consisting of 50 steps.

1. **Training Phase:** Fine-tune hyperparameters (e.g: NeuralNet learning rates, volatility noise, replay buffer size, etc) to increase the average reward in the **evaluation episodes** (with deterministic policy evaluation and frozen network weights) and define a stopping threshold.
2. **Validation Phase:** We evaluate several **DDPG agents** under different random seeds and identify those reaching the reward threshold based on their mean episodic performance.

(a) Wide spread stock (b) Tight spread stock

3. **Testing Phase:** The best agent is tested over 50 steps on unseen data, demonstrating stable rewards and effective market tracking in both wide and tight spread scenarios.



(c) Wide spread market (d) Tight spread market

Conclusion

- **DRL is well-suited for volatility fitting** problem as they possess (a) native exploration (b) effective catalog of past experiences and (c) good predictive power in large spaces.
- **Reward shaping and environment design** are critical for guiding efficient learning.

emmanuel.gnabeyeu_mbiada@sorbonne-universite.fr