Neuro-Hedge: Deep Reinforcement Learning for Optimal Derivative Hedging under Transaction Costs

Technical Report

February 9, 2026

Abstract

This report documents the implementation and evaluation of *Neuro-Hedge*, a Deep Reinforcement Learning (DRL) framework designed to optimize the hedging of European Call Options. While the Black-Scholes model provides an optimal hedging strategy in frictionless markets, it incurs prohibitive costs in the presence of transaction fees due to continuous rebalancing. This project utilizes the Deep Deterministic Policy Gradient (DDPG) algorithm to train an agent that balances the minimization of hedging error against transaction costs.

1 Introduction

Delta hedging is a fundamental risk management technique for options traders. In an ideal market, continuously adjusting the portfolio to match the option's delta (Δ) eliminates directional risk. However, real-world markets impose transaction costs (bid-ask spreads, commissions). Continuous rebalancing leads to excessive transaction costs, necessitating a discrete hedging strategy that trades off variance reduction against cost.

This project formulates the hedging problem as a Markov Decision Process (MDP) and solves it using DDPG, an off-policy actor-critic algorithm suitable for continuous action spaces.

2 Mathematical Formulation

2.1 Market Dynamics

The underlying asset price S_t is modeled using a Geometric Brownian Motion (GBM) as defined in `market_simulator.py`:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{1}$$

where μ is the drift, σ is the volatility, and W_t is a Wiener process.

2.2 Black-Scholes Baseline

The project utilizes the analytical Black-Scholes delta as a baseline and a training guide. For a European Call option with strike K, time to maturity $T - t$, and risk-free rate r, the theoretical price and delta are calculated via standard closed-form solutions.

3 Reinforcement Learning Environment

The hedging environment is implemented as a custom Gymnasium environment in `hedging_env.py`.

3.1 State Space

The observation space is a 3-dimensional vector, normalized to facilitate neural network convergence:

1. **Time to Expiry:** $T - t$
2. **Normalized Price:** S_t/K (Moneyness)
3. **Current Position:** n_{shares} (The number of shares currently held)

3.2 Action Space

The action space is continuous, representing the target hedge ratio:

$$a_t \in [0, 1] \tag{2}$$

This represents the desired fraction of the underlying asset to hold in the hedging portfolio.

3.3 Reward Function

The reward function is engineered to encourage low hedging error while penalizing transaction costs and large deviations from the theoretical Black-Scholes delta (used as a guide). The reward at step t is defined as:

$$R_t = -\lambda_{err}|\Pi_t - V_{BS}(S_t, t)| - \lambda_{cost}(Cost_t) - \lambda_{guide}|a_t - \Delta_{BS}| \tag{3}$$

Where:

- Π_t is the value of the replicating portfolio.
- V_{BS} is the theoretical option price.
- $Cost_t$ is the transaction cost incurred by rebalancing: $Cost_t = |a_t - a_{t-1}| \times S_t \times c_{rate}$.
- Δ_{BS} is the Black-Scholes delta.

4 DDPG Agent Architecture

The agent is implemented in `ddpg_agent.py` using the Actor-Critic architecture.

4.1 Network Structure

Both the Actor and Critic networks utilize Batch Normalization to handle covariate shift and improve training stability.

- **Actor:** Maps State $\rightarrow$ Action.
  - Input: State Dimension (3)
  - Hidden 1: 400 units + ReLU + BatchNorm
  - Hidden 2: 300 units + ReLU + BatchNorm
  - Output: 1 unit + Sigmoid (constrains action to $[0, 1]$)

- **Critic:** Maps (State, Action) → Q-Value.
  - Input: State Dimension (3)
  - Hidden 1: 400 units + ReLU + BatchNorm
  - Hidden 2: 300 units (Combined State+Action) + ReLU + BatchNorm
  - Output: 1 unit (Linear)

4.2 Exploration

Exploration is achieved using an Ornstein-Uhlenbeck (OU) process, which generates temporally correlated noise suitable for physical control problems. The noise is added to the actor's output during training.

5 Training Methodology

The training loop is defined in `train.py`.

5.1 Hyperparameters

Parameter	Value
Initial Asset Price (S_0)	100.0
Strike Price (K)	100.0
Maturity (T)	1.0 year
Volatility (σ)	0.2
Risk-free Rate (r)	0.05
Transaction Cost Rate	0.002 (0.2%)
Replay Buffer Size	1,000,000
Batch Size	256
Learning Rate (Actor)	5×10^{-5}
Learning Rate (Critic)	5×10^{-4}

Table 1: Simulation and Training Hyperparameters

5.2 Curriculum Learning (Teacher Forcing)

To prevent the agent from converging to suboptimal local minima (such as not hedging at all), the training implements a "Teacher Forcing" mechanism.

At the beginning of training, the agent executes the analytical Black-Scholes delta with high probability. As training progresses, a decay factor reduces this probability, forcing the agent to rely more on its neural network policy.

$$P(\text{Teacher Action}) = \max(0, \text{InitialRatio} \times \text{Decay}^{episode}) \tag{4}$$

6 Evaluation and Metrics

The evaluation script `evaluate.py` compares the trained DDPG agent against a naive Black-Scholes strategy that rebalances at every time step.

Key metrics recorded include:

1. **Terminal P&L:** The final profit or loss of the hedging strategy. Ideally, this should be close to zero (perfect hedge) minus transaction costs.
2. **Transaction Costs:** The total accumulated fees.
3. **Hedging Error:** The variance of the portfolio value relative to the option price.

7 Conclusion

The Neuro-Hedge project demonstrates the viability of DDPG for derivative hedging. By incorporating transaction costs into the reward function and using Black-Scholes theory as a training guide, the agent learns a strategy that approximates the theoretical delta while suppressing excessive trading to conserve capital.