Pareto Option Pricing with Deep Q Learning

CME 241

Eugene Francisco

March, 2025

1 Introduction

In this project, we investigate the problem of fair option pricing through the lens of a Markov Decision Process. The fair pricing of options has a closed form solution when the underlying securities' returns follow a Brownian Motion, as famously demonstrated by Fischer Black and Myron Scholes in 1973. However, closed form solutions become challenging once the underlying security dynamics are no longer Brownian.

To this end, we began by assuming that the distribution of returns of the underlying security was Pareto distributed with infinite variance. We picked the Pareto partly for its heavy tailed nature but also for its skewness.

Our project's ultimate goal was then to provide a function approximation for an option's optimal price by treating option valuation as an MDP problem and applying deep Q learning. Specifically, we first fix a strike price K and perform deep Q-learning using experience tracing and a neural network Q function approximation. We then repeat this process across multiple K, essentially learning how the value of the option shifts with the strike price across different time horizons and pricing horizons.

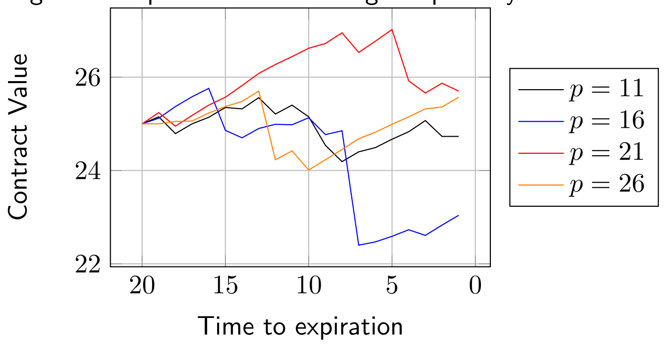
Ultimately, all the models we trained showed signs of being strong estimators for the fair option value; however, computational limitations in experience generation and the inherently sparse reward structure made training the models a challenging and time consuming task that required much fine tuning.

Still, the Q-learning technique allowed for easy generalization across the continuous time horizon, even when parameters for the return dynamics (the Pareto in our case) are only available for fixed time intervals.

2 Pricing Dynamics

The Pareto distribution is a heavy tailed distribution perhaps most well known for being the same distribution that governs the so-called *80-20* rule, where 80% of properties are exhibited by only 20% of the people. The Pareto distribution has two parameters: a scale parameter β and a shape parameter $\alpha > 1$, also known as the tail index. The PDF of the Pareto is

Figure 1: Experience traces using the price dynamics.



$$f(x) = \frac{\alpha\beta^\alpha}{x^{\alpha+1}} \tag{1}$$

If X is distributed Pareto, then

$$\mathbb{E}(X) = \frac{\alpha\beta}{\alpha - 1} \quad \mathrm{var}(X) = \begin{cases} \infty & : \alpha \in (1, 2], \\ \left(\frac{\beta}{\alpha-1}\right)^2 \frac{\alpha}{\alpha-2} & : \alpha > 2. \end{cases}$$

For our pricing dynamics, we fixed a small interval of time $\Delta t = 1$ and then assumed that

$$\frac{S_{t+\Delta t} - S_t}{S_t} \sim (\gamma + (X - \mu))\Delta t$$

where X is Pareto distributed with $\alpha = 2$ and $\beta = 0.01$, and $\mu := \mathbb{E}(X)$. We chose $\gamma = 0.005$ as the drift parameter. The choice of α and β here depend on the original choice of Δt. We note that the choice of α means that the Pareto distribution we're working with has infinite variance. Four experience traces over a 20 day horizon with a starting price of \$25 are plotted in Figure 1.

3 MDP Setup

The agent in the MDP is assumed to be the owner of a call option with fixed strike price K and time till expiration T. The state space for the MDP is a pair (p, t), where $p \in \mathbb{R}$ is the price of the underlying security and

$t = T - n\Delta t$ for some $n \in \mathbb{N}$ is the time till expiration, discretized according to the pricing dynamics. Terminal states are those where $t = 0$.

Given a non-terminal state (p, t), the agent can choose to either hold the option or execute the option. If the option is held, the immediate reward is 0. If the option is executed, the immediate reward is $p - K$.

If the agent chooses to hold, then the next state is always $(p', t + \Delta t)$, where p' is sampled according to the pricing dynamics from Section 2. If the agent chooses to execute, the the MDP ends after they receive their immediate reward.

4 Deep Q Learning Method

Due to the continuous nature of the state space, we used a neural network function approximation of the Q function to learn Q values. We then used deep Q-learning to train this function approximation of the Q function.

4.1 Function Approximation

We used a simple fully connected neural network to approximate the Q function. The layers of the network are as follows:

1. Linear layer 64 neurons
2. Linear layer 128 neurons
3. Linear layer 128 neurons
4. Linear layer 64 neurons
5. Linear layer 2 neurons

where all but the last layer were followed by a ReLU and then a batch normalization. We used the RMSprop optimizer and an ℓ_2-regularization with $\lambda = 0.0001$. We call this function approximation Q_θ our Q-learner. The two outputted neurons we interpreted as the Q_θ function for each action, with the first entry being $Q_\theta(s, \text{hold})$ and the second entry being $Q_\theta(s, \text{execute})$. Note that we overload the definition of Q_θ so that $Q_\theta(s)$ refers to the 1 by 2 output vector of Q_θ while $Q_\theta(s, a)$ refers to the entry associated with a of $Q_\theta(s)$.

4.2 Deep Q Learning

We used deep Q learning to train the Q-learner. At a high level, this involves alternating between two phases which we will refer to as the *experiencing* phase and the *learning* phase. We call each alternation between an experiencing phase and a learning phase a *learning cycle*. We detail both phases below.

4.2.1 Experiencing phase:

The experiencing phase is meant for the Q learner to gather data about the environment before learning.
Step 1: We fix in advance an experience threshold T, a price window $P \subset \mathbb{R}$ and a time horizon $0 < H \leq T$.[1] We also initialize an empty experience buffer $\mathcal{D}$ and an empty list of targets $\mathcal{T}$.
Step 2: Sample a random non-terminal state $s_t = (p, t)$ by sampling p and t within the pricing window and time horizon respectively. Collect an *experience* s_t, a_t, r_t by sampling an ε-greedy a_t using our current Q_θ function approximation on s_t, and recording the reward r_t. If we've chosen to hold, then we sample a next state $s_{t+\Delta t}$ according to the MDP. Now repeat with $s_{t+\Delta t}$. We stop this experience trace either when $t + j\Delta t = 0$ for the jth step, or when the agent chooses to execute.
Step 3: For each experience in the trace, add s_t to the experience buffer $\mathcal{D}$. Calculate a *target value* for the experience as follows. If a_t was to hold, then the target value is

$$\text{target} \leftarrow \frac{1}{1+\gamma} Q_\theta(s_{t-\Delta t}, a') \tag{1}$$

where a' is chosen greedily using the current Q_θ approximation on s_t.[2] If a_t was to execute, then the target value is

$$\text{target} \leftarrow p - K \tag{2}$$

Store the target value and the action a_t in $\mathcal{T}$.
Step 5: If the size of $\mathcal{D} \geq T$, then we are done. Otherwise, repeat from Step 2.

4.2.2 Learning Phase:

In the learning phase, we use the experience buffer $\mathcal{D}$ and the targets $\mathcal{T}$ as training examples to update Q_θ. To do this, we sample a minibatch of experiences X and their associated target Y from $\mathcal{D}$ and $\mathcal{T}$. Suppose that for each example $s \in X$, $r(s)$ is s's associated target value and $a(s)$ is its associated action from $\mathcal{T}$. Then we update the weights θ in Q_θ using gradient descent with respect to the loss function

$$J_\theta(X) = \frac{1}{|X|} \sum_{s \in X} (Q_\theta(s, a(s)) - r(s))^2. \tag{3}$$

Note that this is the MSE applied only to the entry of $Q_\theta(s)$ that corresponds to the action $s(a)$. The update

[1] P should be a connected interval of $\mathbb{R}$ that is reasonable for the choice of strike price.

[2] We note that if $t - \Delta t = 0$, then we force $Q_\theta(s_{t-\Delta t}, \text{hold})$ to be $\max\{0, p - K\}$; this is to enforce the end of lifetime hockey-stick value of the option.

rule for θ is standard for gradient descent, namely

$$\theta \leftarrow \theta - \alpha \nabla_\theta J_\theta(X)$$

for each minibatch X. Note that the target $r(s)$ values used are bootstrapped reward values taken from experience traces, and do not reflect true Q values. The goal of the learning phase is to make a slight improvement on Q_θ.

4.3 Training Strategy:

We picked a time horizon of 20 days with a pricing window of $[1, 30]$. For each Q-learner, we trained the Q learner for 1600 alternations of experiencing and learning. We scheduled the experiencing phases so that Q-learner would only see experiences starting with at most 20 days left for the first 800 iterations of the training loop; for the latter 800 cycles, we had the Q learner exposed to traces starting from between 1 day to 40 days till expiry; we did this so that the Q-learner had an opportunity to learn the end-of-contract pricing dynamics before moving to a broader time horizon.

Each experience phase collected 4,096 experiences. Each learning phase performed a gradient descent update on Q_θ in mini-batches of 1,024 experiences.

We also used an ε-scheduler to decay the probability of choosing a non-greedy action within each experience trace. Specifically, before each experience phase, we update ε by $\varepsilon \leftarrow \varepsilon \cdot 0.995$.

4.4 Target Network

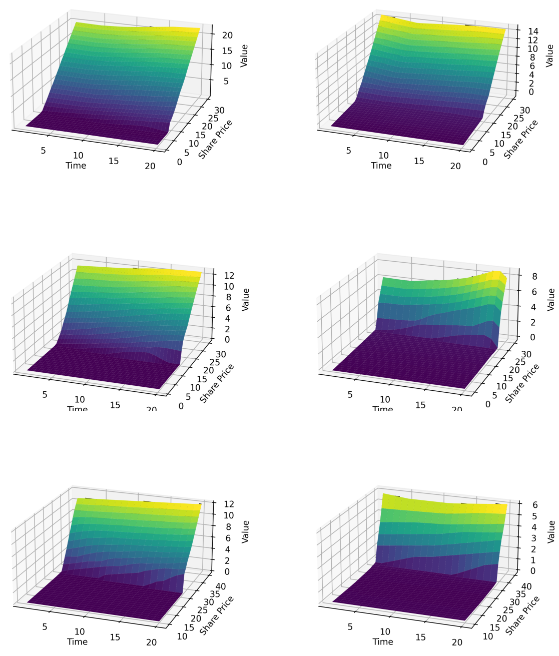
Instead of using the same Q_θ for the target calculation in (1), we actually use a target network $T_{\hat{\theta}}$, which is a frozen version of Q_θ that is periodically updated to match Q_θ. In our implementation, we update $\hat{\theta} \leftarrow \theta$ every 50 learning cycles. We implemented the target network to stabilize training as early experimentation without a target network revealed that the fat-tailed nature of the price dynamics led to unstable convergence of Q_θ.

5 Results

We trained across six different choices of K, namely $\{10, 15, 20, 25, 30, 35\}$. As expected, we saw linear behavior near expiration date. We were happy to see that the value of the contracts predicted by Q_θ grew as the time till expiry grew back, just as would be expected by the true contract value. The learned contract values across the different Q-learners is plotted in Figure 2.

In the Black-Scholes pricing model, an options price can be derived directly from the ratio p/K. However,

Figure 2: Left to Right, Top to Bottom, Expected Value Strike Price 10, 15, 20, 25, 30, 35. Note the different scales for value and time.



the models we trained suggest that this is not true in our regime. Figure 3 plots the contract value at 10 days till expiration for contracts where $p/K = 1.2$. We posit that this is probably because of the asymmetric nature of the return distribution, despite mean centering the distribution.

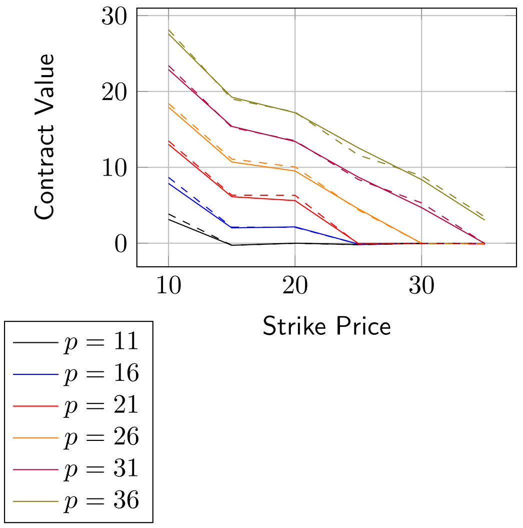
Figure 3: Fixed $p/K = 1.2$ ratio, contract value.

Strike Price	Value
10	2.71
15	2.82
20	7.01
25	6.6
30	7.98
35	7.11

5.1 Generalizing to more K:

Given any current share price p, time till expiration t, we'd like to know how our different Q-learners estimate the contract value as a function of strike price K. In Figure 4 are plotted the estimated contract values for different share prices p, plotted 16 days till expiration. The trend is roughly linear, though it is possible that this trend fails for longer time horizons.

Figure 4: Value as a function of strike K for (solid) $t = 16$ and (dashed) $t = 20$.



6 Analysis and Conclusion

When we first had the idea for this project, we were eager to apply the fair-pricing problem to heavy-tailed regimes, especially because preliminary sample tracing for security random walks revealed just how volatile price movements tended to be.

We jumped on the idea of using deep Q-learning to model the value of the contracts because it would allow us estimate option value along the continuous space of share prices. Beyond this, in the practical setting where values of α and β can only be empirically estimated for a fixed time scale, the Q network allows for a natural extension to expiry times outside of the discretized time scale.

Because of the asymmetric nature of returns under the Pareto assumption, we guessed that contract values would be explicit functions of both (independently) the current share price and strike price, as confirmed in our results.[3] We originally trained a more complex Q-learner that learned on experience traces with varying strike prices (with Q_θ taking in K as an extra input) but results were poor; rewards were already sparse in the Q-learner with fixed K, we posit that the problem may have been exacerbated when the state space was widened. This led us to consider using individual Q-learners for fixed K, and then combining the smaller models to make larger estimates.

In each learning cycle, the experiencing phase was far more computationally expensive than the learning phase. Around 90% of the Q-learning time for each of these models was spent on experience generation. Generating different experience traces has the benefit of being highly parallelizable, but each trace still must undergo its full expansion, and each experiencing phase requires the ε-greedy policy implied by $Q_{\hat{\theta}}$, limiting the amount of work that can be done in each learning cycle.[4] As mentioned previously, the sparse nature of the rewards made training slow. In Figure 3, we've placed visuals of the training progress of the Q-learner.

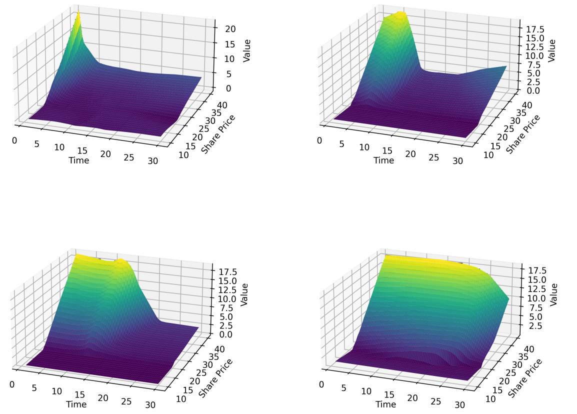


Figure 5: Left to Right, Top to Bottom, Num Learning Cycles 51, 351, 651, 1401.

We had originally intended to view the problem for a broader time horizon, and the values of α and β in the pricing dynamics were chosen with this broader time horizon of 40 to 80 days in mind. However, limitations in time and training meant we ultimately only explored a half of the time horizon that we would have liked. We anticipated before training the Q learners that training would be the largest computational bottleneck; in the end, it was experience generation. Should we have more time next time, we would allow our models to explore a broader time horizon, which would allow more for more information on how contract values change as a function of time-till-expiry, and also allow for a more nuanced generalization than Figure 4.

[3]Again, this is as opposed to the Black-Scholes model where contract value is a function of p/K.

[4]It might interest the reader to know that, after multithreading and transferring training to local GPU, experience generation consumed 80% of all 10 CPU cores on Eugene's M1 Pro (training two models concurrently).