Soft Actor-Critic Models for The Application of Hedging Path-Dependent Derivatives

ELENE 6885
Christopher Reekie
Columbia University

Abstract

Within this project, a Soft Actor-Critic Deep Reinforcement Learning model was trained to hedge path dependent, exotic derivatives, common in the insurance retirement market, with a vanilla replicating call and underlying asset. With task specific reward function tuning, the algorithm performs well compared to benchmark with and without the presence of transaction costs but performed generally poorly when hedging the same task / liability with multiple hedge instruments. The resultant trained policies fall short of optimal hedge behavior due to the stochastic nature of the learned policies, but the strong performance illustrates potential for the application of Deep Reinforcement Learning in this field.

1. Introduction

A popular investment product, offered by insurance companies, to those seeking to save for retirement is the Fixed Indexed Annuity (FIA). In a FIA contract, the insurer guarantees that that the purchasing policyholder's account will grow at the minimum of an underlying index growth rate (e.g. the SP500), or 0% over a specified time horizon in exchange for a fee. The FIA product provides the purchaser the ability to participate in market returns while limiting the risk of loss in a market downturn.

The insurance companies writing FIA products aim to generate profits through fees rather than through market speculation. To this end, the insurance company will hedge the market exposure in the written FIA contract using market traded instruments (typically derivatives, swaps and futures).

The key challenge in hedging market risk within FIA products is that because the purchasing policyholder is often granted additional options within the contract, such as the right to unilaterally terminate the contract prior to expiration, there is no available market traded instrument that perfectly hedges the market risk in the FIA.

Hedging insurers often take an approach grounded in financial engineering theory to create hedge portfolios that closely offset the risk of the sold liability using market traded derivatives. These approaches rely on theoretical market models and risk-neutral pricing theory that often does not correlate with actual market behavior.

The goal of this project is to explore whether the model-free deep reinforcement model, Soft Actor-Critic (SAC), can learn from experience to hedge the risks inherent in the FIA contract. If successful, such an algorithm could be utilized to aid in establishing new hedge strategies to manage risk in exotic options.

The SAC model learns through simulated episodes where the actor chooses asset positions to hedge the simulated liability through a single year trading period. The model is rewarded for matching both the day to day value of the liability and in matching and offsetting the ultimate payoff of the liability.

The SAC algorithm was evaluated in three hedging scenarios with increasing complexity. Adjustments to the training routine and reward function were explored to achieve reasonable performance and training stability.

2. Background & Related Work

While there is no directly related research into the application of reinforcement learning for the purpose of hedging FIA-specific market embedded risk, the use of Deep Deterministic Policy Gradient models (DDPG) has been researched for the application of 'deep hedging' [8] of generic portfolios. The original authors demonstrate that DDPG can be used to derive optimal portfolios in the presence of transaction costs.

The use of a maximum entropy deep reinforcement models (SAC) for this specific task could not be found in the published literature. Resources to an in depth review of hedging and pricing theory can be found in the appendices (see [6] and [7]).

2.2 Formulation of Fixed Indexed Annuity Liability & Hedging Problem

In this section, a technical formulation of the problem task is provided. Including a review of the agent's objective and the hedge objective that forms the basis for the reinforcement learning reward function.

There are a range of varying FIA product types in the market, many with unique embedded derivatives which differentiate the risk of the contract from that of standard European type derivatives. For this project, it is assumed that the liability has the following features:

- A 1 year guaranteed term period with uncapped growth rate and 100% participation in the underlying market.
- The option to lapse the policy before term and receiving no market growth (i.e. return of premium).
- The 80% of positive index growth upon death (if death occurs before term).

The time t per dollar notional payoff structure of the FIA $F(t)$ is defined below:

$$F(t) = \begin{cases} \max\left(0, \frac{S_T}{S_0}\right), & if\ t = T \\ 80\% \cdot \max\left(0, \frac{S_t}{S_0}\right), & if\ death\ occurs\ at\ t < T \\ 0, & if\ lapse\ occurs\ \ t < T \end{cases}$$

The vanilla (publicly traded) option that most closely offsets the FIA liability is that of a European call option $C(t)$. The per dollar notional payoff of a call contract $C(t)$ is:

$$C(t) = \begin{cases} \max(0, \frac{S_T}{K}), & if\ t = T \\ 0, & t < T \end{cases}$$

Where:
$S_t = Index\ of\ Underlying\ at\ time\ t\ (e.g. SP500)$
$K = Chosen\ strike\ price\ of\ call\ (e.g. S_0)$
$t: 0 \leq t \leq T\ and$
$t = 0\ at\ inception\ of\ contract\ and$
$t = T\ at\ expiration\ of\ contract\ (i.e. T = 1\ year)$

In the experiments, the hedge option available for the SAC algorithm was a call option with strike $K = S_0$, the option that most closely replicates that of the hedged liability.

Both the liability $F(t)$ and Call option $C(t)$ have intrinsic option value ($OV_{FIA(t)}$ and $OV_{C(t)}$) prior to expiration/term ($t < T$).

2.3 Hedge Strategy Objective

The objective of the insurer is to neutralize exposure to market risk by entering positions which match the value and payoff of the sold FIA contract. In essence, the insurer seeks to ensure that it does not either gain or lose money in the contract due to market movements.

In aim of this goal, the hedging insurer seeks to establish a hedge strategy which determines the composition and management of a portfolio $\theta_i(t-1)$ of assets for each contract i (e.g. calls, stocks & futures positions), such that the expected value of the hedge portfolio at time t $V_i(t|\ \theta_i(t-1))$ matches the expected value and payoff $F_i(t)$ for contract $i\ \forall t$.

Therefore to satisfy this requirement, the hedge strategy $\theta_i(t)$ should minimize the expected gain or loss $|HT_i(t)|$. Where $HT_i(t)$ is the **Hedge Target** for contract i at time t which is determined by:

$$HT_i(t) = \begin{cases} V_i(t|\ \theta_i(t-1)) - F_i(t), & 0 \leq t \leq T \\ V_i(t|\ \theta_i(t-1)) - \left(0.8 \times \max\left(0, \frac{S_t}{S_0}\right)\right), & on\ death \\ V_i(t|\ \theta_i(t-1)) - 0, & on\ lapse \end{cases}$$

Since policyholder lapse, death and market movements are random variables, clearly there is no hedge strategy that can guarantee $HT_i(t) = 0\ \forall i$. Therefore the agent seeks to establish a policy to compose $\theta_i(t)$ such that:

$$E\left[\sum_{i,t}(HT_i(t)|\theta_i(t-1))\right] = 0$$

Which represents the following optimization problem:

$$\arg\max_{\theta_i(t-1)} -|HT_i(t)|\theta_i(t-1)|$$

The task of the RL agent in this project is to learn a policy to determine $\theta_i(t-1)$ at each time step, for a given liability.

3 Environment & State Space Simulation

3.1 Episode Specification

For the purpose of this project, the environment is defined as episodic in nature. A single episode starts with the initiation of the FIA contract liability, and the task of the agent is to choose asset positions at each time step to best hedge the liability over the next time step to achieve an expected hedge target of 0.

The episodes have a maximum of 252 timesteps and are stochastic in nature, in that the return on the underlying index is determined by a stochastic process (assumed to follow a 'real world' trajectory). The episode will terminate early in the event of death or lapse with some random probability.

Four states are defined for the policy holder; **active** where in the policyholder has not lapsed or died and continues to hold the contract, **lapsed** (terminal) where the policyholder has chosen to exit the contract, **dead** (terminal) where the policyholder has died and must be paid a partial benefit or **term** (terminal) where the policyholder has held the contract to the end of the episode and must be paid the credit earned by the index over the episode.

3.2 Episode Simulation

The underlying index price through the episode is simulated using the Heston stock price model [4], that assumes that the price of the underlying index/stock (S_t) follows a stochastic process along the 'real world' path as defined below:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$

Where the instantaneous volatility is itself a stochastic process defined by:

$$dv_t = \kappa(\theta - v_t)dt + \varepsilon\sqrt{v_t} dW_t^v$$

Where W_t^S and W_t^v are correlated wiener processes: $dW_t^S dW_t^v = \rho dt$.

A risk neutral parameterization of the Heston model was used to calculate the 'risk-neutral' option value of both the FIA liability ($OV_{FIA(t)}$) and the available hedge assets (e.g. $OV_{Call(t)}$) via Monte-Carlo simulation and numerical integration in a 'stochastic-in-stochastic' approach.

The annual 'risk free' growth rate of 1.2% was assumed, for the real world episode paths a random drift rate was applied by making the following adjustment to the risk-free rate for each timestep, illustrated in Figure 1 below:

$$\mu_{real-world} = \mu_{risk-free} \cdot e^z, \qquad z \sim N(0.05, 0.75)$$

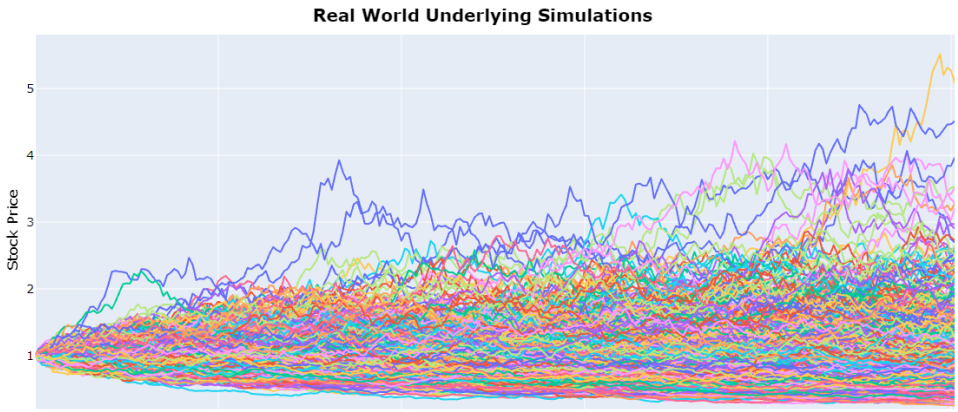

Figure 1

3.3 State Space Observation

At each time step, the agent can observe the following features:

$S_t = Current\ Stock\ Price$
$v_t = Current\ volatility\ of\ the\ stock$
$OV_{FIA(t)} = Risk\ neutral\ option\ value\ of\ liability\ at\ t$
$OV_{C(t)} = Risk\ neutral\ option\ value\ of\ a\ call\ contract$
$\Delta OV_{FIA(t)} = Delta\ of\ liability\ option\ value$
$\Delta OV_{Assets(t)} = Delta\ of\ call\ option\ value$

The delta (ΔOV) of an option value is its price sensitivity to a change in price of the underlying (S_t). An alternative to static hedging, is dynamic hedging which hedges via matching asset and liability Greeks (e.g. delta matching).

3.4 Action Space

The agent must choose how much (a numerical amount) of each available hedge asset to hold at each time step $\theta(t)$. Where $\theta(t)$ represents a vector of positions to hold in each of the available hedge assets (akin to notional). If a call and underlying stock are available hedge assets, the policy has an action space of dimension 2. For example, $\theta_i(t) = [0.25, 0.4]$ where 0.25 is the notional to hold in the call option and 0.4 is the notional to hold in the underlying asset for liability / episode i.

3.5 Reward Function & Transition Probabilities

The base reward function is derived from the hedge target $HT(t)$. At time $t-1$, the agent chooses $\theta(t-1)$ after observing the environment. The environment stepped by one timestep (representing a day), where the underlying stock price changes, the FIA liability transitions to the next state, and the agent receives a reward based on the difference between the new liability value/payoff and the value of its current portfolio value. The reward function granted after each state transition is:

$$R(S_t = Active) = -|\theta(t-1) \cdot \Delta_{t+1t} OV_{Assets} - \Delta_{t,t-1} OV_{FIA}|$$
$$R(S_t = Dead) = -\left|\theta(t-1) \cdot OV_{Assets}(t) - 0.8 \cdot \max\left(0, \frac{S_t}{S_0}\right)\right|$$
$$R(S_t = Lapse) = -|\theta(t-1) \cdot OV_{Assets}\ (t)|$$

The reward function encourages the actor to match the option value of the liability OV_{FIA}, but penalizes the actor strongly in the event of early lapse where the hedge asset must be sold without an offsetting liability, or on death where a partial amount of the underlying index growth must be credited.

The probability of death over the episode period was assumed as 1%, and the probability of lapse 4%; both are independent random variables. The per time step probability transitions (given 252 time steps) were:

$$P(S_{t+1} = Active\ |S_t) = 1 - \Pr\ (S_{t+1} = Dead \cup Lapse|S_t)$$
$$Pr(S_{t+1} = Dead\ |S_t) = 1 - (1 - 1\%)^{(\frac{1}{252})}$$
$$Pr(S_{t+1} = Lapse|S_t) = 1 - (1 - 4\%)^{(\frac{1}{252})}$$

3.6 Evaluation

Since each episode represents the simulation of hedging of a single liability and the goal of the agent is to find a policy that maximizes average reward, each policy was evaluated by averaging the returns over 200 episodes.

Finally, the best model retained from training was evaluated against a benchmark hedge strategy comprised of a static position of 95% in the underlying call option. This position closely matches the expected payoff but requires knowledge of the environment transition probabilities (death and lapse).

4 Reinforcement Learning Algorithm

4.1 Task & Model Choice

In this project, the reinforcement learning algorithm is tasked with learning a policy to choose $\theta_i(t-1)$, the optimal composition of hedge assets that optimize the hedge target goal. In effect, the learned policy π, must learn to choose $\theta_i(t-1)$ given the previous state S_{t-1} such that:

$$\pi(State_{t-1}) = \theta_i(t-1)\ \ s.t.\ \ E\left[\sum_{i,t} HT_i(t)\,|\theta_i(t-1)\right] = 0$$

A key challenge in the task for reinforcement learning algorithms besides the stochastic and complex nature of the problem is that the action space is continuous (the policy chooses a numerical amount to hold in each asset). Even with a bounded action space, the number of possible actions is theoretically infinite.

With these constraints in mind, the candidate model chosen for application in this project was the Maximum Entropy Soft Actor Critic model (SAC) proposed by Tuomas Haarnoja et. al. [2]. SAC is an off-policy deep reinforcement learning algorithm that produces a stochastic actor/policy.

This model was chosen for the following reasons:

- The actor/policy must choose from a continuous action space, which is an area where Actor-Critic algorithms retain state of the art performance.
- The actor is stochastic and the learned policy is a 'squashed' Gaussian distribution. The stochastic nature of the actor is appealing in that it may be more likely to explore the state space to find novel hedging methods when presented with a variety of hedge instruments.
- The SAC algorithm is sample efficient and can make use of a replay buffer of previous transitions. This is an advantage in a hedge simulation task given the computationally intensive nature of simulating episodes and calculating option values.

4.2 Soft Actor Critic Model Overview & Related Work

The soft actor critic algorithm makes use of 3 core components:

1. An Actor-Critic architecture with separate Policy and Q-Value and Target Q-Value Neural Networks.
2. An off-policy learning algorithm that allows for the use of a 'Replay Buffer' for sample efficiency.
3. An adjusted reward function embeds an entropy term to reward policy stochasticity.

In the Actor-Critic (AC) algorithms, the Actor is the policy and the Critic is a Q-Value approximator which evaluates the actions chosen by the Actor.

In Actor-Critic DRL, both the actor and the critics learn concurrently. Prior to SAC, a popular DRL A-C algorithm was the Deep Deterministic Policy Gradient algorithm which allowed for off-policy learning by utilizing Q-function estimators. These algorithms are empirically unstable, require hyperparameter

tuning and do not extend well to high dimensional & complex tasks[3].

The proposed Soft Actor Critic algorithm addresses the weaknesses of DDPG (training instability & poor complexity scaling) while retaining the key advantages of sample efficiency and state of the art performance in continuous state space tasks.

Unlike the deterministic actor in DDPG, Soft Actor Critic achieves the above by combining a stochastic actor with the Actor Critic approach and a unique reward adjustment that rewards maximizing policy entropy.

4.3 SAC Model Architecture & Policy Objective Function

The 'Actor' learned from the SAC algorithm is a Gaussian distributed 'stochastic' policy which is parameterized by a learned Neural Network ($f(State_t)$), the output of the sampled distribution is 'squashed' by the tanh function:

$$\pi_{\emptyset}(a_t|State_t) \sim \tanh\left(N(\mu_t, \sigma_t)\right)$$

Where a differentiable Neural Network ($f(\cdot)$) parameterizes the policy given the state:

$$f(State_t) = \mu_t, \sigma_t.$$

Tanh is used to 'squash'/bound the output of the Gaussian action space, since the output of the Gaussian is theoretically unbounded. By apply Tanh, the range of possible actions is bounded to (-1,1).

The 'Critic' learned in SAC are 2 pairs of Q-Value function approximators $Q_{\theta_i}\ i \in \{1,2\}$ used as current state Q-Value approximators and $Q_{\psi_j}\ j \in \{1,2\}$ used as target Q-Value approximators.

All Q-Value models are Neural Networks. The current state Q-Value networks learn from observational experience and the Target Q-Value networks learn via soft update from current state value networks via Polyak-averaging of their parameters with very low learning rate (tau).

The algorithm makes use of 'Clipped Double-Q Learning' trick. The Q-Value networks are trained to a single target which is the minimum of value of the two Q-Value target networks (proposed in TD3 to improve learning stability[10]) :

$$Q^*_{target} = \min_j Q_{\psi_j}\ j \in \{1,2\}$$

The final key component SAC is the objective function used to train the policy:

$$J_{\pi}(\emptyset) = \sum_{t=0}^{T} \gamma^t E_{\pi}[R^a_t + \alpha \mathcal{H}(\pi(\cdot|State_t)]$$

Where $\mathcal{H}\big(\pi(\cdot|S_t)\big) = -\log\left(\pi(\cdot|S_t)\right)$ is the entropy of the policy, and α is the temperature parameter. The addition of the entropy term encourages the policy to act randomly as while balancing maximizing the expected reward.

4.3 Soft Actor Critic Objective Functions & Training

The loss functions and update rules used to train each component of the SAC model are outlined in the following section. These functions are provided in Harnooja et. Al [3] and the pseudocode from the original paper (implemented in this project) is provided in the appendices.

It should be noted that the implementation in this project makes use of clipped Double-Q Learning. This empirically helps reduce positive bias in the Q-values.

4.3.1 Soft Q-Value Network Updates

The soft Q-Value network parameters (θ_i) are updated via stochastic gradient descent to minimize the following objective function:

$$J_Q(\theta_i) = E\left[\frac{1}{2}\left(Q_{\theta_i}(State_t, a_t) - \left(R^a_t + \gamma E\left[\min_j Q_{\psi_i}(State_{t+1}, a_{t+1}) - \alpha \log\left(\pi(\cdot|State_{t+1})\right)\right]\right)\right)^2\right]$$

Where $E\left[\min_j Q_{\psi_j}(State_{t+1}, a_{t+1}) - \alpha \log\left(\pi(\cdot|State_{t+1})\right)\right]$ is the next state value function and the Q_{ψ_j}-Value target network parameters are updated from Q_{θ_i} via the soft update Polyak-averaging function:

$$\psi_i = (1-\tau)\psi_i + \tau\theta_i$$

A target network is used to improve training stability by reducing correlation between the current policy and the target policy. In this implementation, the target networks are updated at the same cadence as the base Q-networks, but the τ value is very small ($\tau = 0.005$).

4.3.2 Policy Network Updates

The policy Network is trained to minimize the following loss function, via updating $\emptyset$ with stochastic gradient descent:

$$J_{\pi}(\emptyset) = E_{\pi}\left[\alpha \log\big(\pi(\cdot|State_t)\big) - \min_i Q_{\theta_i}(State_t)\right]$$

The updates to the policy Network serve to reduce the expected KL-divergence between $\pi_{\emptyset}(a_t|State_t)$ and $\exp\left(\frac{1}{\alpha}Q_{\theta_i}(State_t, a_t)\right)$ (refer to original paper for proof). Due to the stochastic nature of the actor, the 'reparameterization trick' (Kingma et al. [5]) is used to update the network parameters of the random layer via gradient descent.

Using a 'squashed Gaussian' distribution $\pi \sim \tanh\left(N(\mu_t, \sigma_t)\right)$, the log-likelihood of the policy action given the state is given by:

$$\log\big(\pi(\cdot|State_{t+1})\big) = \log(\boldsymbol{z}) - \sum_{i=1}^{D} \log\left(1 - tanh^2(z_i)\right)$$

Where $\boldsymbol{z}$ is a D dimensional sampling from $N(\mu_t, \sigma_t)$.

4.3.3 α Tuning Updates

Finally, empirical results demonstrate that the temperature parameter α should be tuned during learning to achieve a balance between exploration and exploitation through the learning process. The practical algorithm proposed in Haarnoja et al. uses the following objective function with which to compute gradients for α updates:

$$J(\alpha) = E_{\pi}\left[-\alpha \log\big(\pi(\cdot|State_t)\big) - \alpha\widehat{\mathcal{H}}\right]$$

Where $\hat{\mathcal{H}}$ is the target entropy with default recommended value equal to the dimension of the action space (e.g. 2 for 2 available action variables).

With α tuning, the α parameter typically starts very high, as the model seeks to explore the unexplored region, then as the model becomes more confident in the action reward space the α parameter reduces over training.

4.3.4 Experience Replay

SAC makes use of an experience replay buffer, which stores action value transition tuples during experience then randomly samples batches from the buffer to train the network with. This serves to decorrelate the training data to some extent and has been found to empirically improve convergence when using neural network function approximators.

4.3.5 Technical Implementation Details

The pseudocode for the training routine implemented in this project (from the original paper) is provided in the appendix (8.2). The algorithm was implemented in Python using Pytorch for the model's architecture & training routine.

The paths and environment were simulated using Python, and the risk neutral option values of the FIA liability and the hedge options were calculated using cuda (via the Numba python package).

In all experiments, the first 500,000 actions were chosen at random to facilitate initial exploration and learning. From then on, the agents own actions were used. Each model was trained using roughly 30,000 episodes (~8 million timesteps) and took 26 hours to train.

The calculation of log probabilities was numerically unstable, so to ensure stability in training the log values were 'clamped' between -20 and 2.

5. Experiments & Results

This project focuses on three unique setups of the reinforcement learning task, each with increasing complexity. Under each setup, various configurations of the SAC model were trained and evaluated. To the extent that the models underperform expectations, reward function scaling and parameter tuning methods were explored to improve performance and training stability.

The hedging task actor must balance taking a position in the call asset (which closely replicates the value of the hedged liability) while also taking into account the probability of early termination of the episode due to death or lapse. In all experiments, automatic tuning of the temperature parameter α was used.

To evaluate the performance of each of the models, the best model during training was evaluated for 3000 episodes against a 'benchmark' static strategy which consists of holding 95% of the replicating call option (determined via knowledge of the MDP transition probabilities ($Pr(S_{t+1} = Lapse|S_t)$, $Pr(S_{t+1} = Death|S_t)$ etc.).

5.1 SAC Hedging with a Single Call Option

5.1.1 Experiment Setup

In the first experiment, the SAC algorithm was tasked with learning the optimal hedging strategy by adjusting its position in a single replicating call option (action space of size 1). The output of the SAC model policy is the chosen notional position in the call to match that of the liability.

The actor can adjust its position (notional) at each timestep throughout the episode without penalty (transaction costs). Since the call option closely resembles the FIA liability, the key challenge for the agent in this task is balancing matching liability and asset option values with the probability of death or lapse of the liability.

5.1.2 Algorithm Adjustments

To achieve optimal performance in this task. The following parameter changes were made from the 'default' parameters suggested by the original authors:

- Batch sizes of 1024 were used instead of 256.
- The learning for Actor, Critic & Alpha tuning was adjusted from 3×10^{-4} to 1×10^{-4}.
- Scaling of output action by a factor of 2.0

The authors recommend a batch size of 256 and learning rate of 3×10^{-4}; however, given the episodic nature of the task and the relatively low probability of state transition to death or lapse, high batch sizes (1024+) and low learning rates (1×10^{-4}) resulted in optimal performance in this task by increasing the likelihood that the training batches would include one of these transitions.

An additional adjustment that improved performance was the scaling of the agent's actions. Because the actions of the agent are a 'squashed' Normal where the range of output is (-1,1) and the optimal proportion in the asset to hold is in the range of 0.85-0.95. Without scaling of actions, the actor would tend toward high mean Normal values to target the upper range of the tanh function leading to large gradients and instability. Therefore, the output of the agent was multiplied by a factor of 2.0 for a resultant action range of (-2,2).

This adjustment results in faster training and less training instability by preventing exploding gradients where the agent would parameterize the mean and covariance of the distribution to extremes.

5.1.3 Model Performance Comparison

Since lapse events incur large negative reward when matching the liability option value and the episodes are 252 time steps or shorter, with a small batch size (256) and high learning rate, the algorithm would encounter these events infrequently and would tend to overreact in learning.

Increasing the batch size to 1024 transitions and decreasing the learning rate served two purposes, the algorithm was better able to determine the probability of lapse event and tune its average position accordingly with the larger batch size, and the lower learning rate prevented an overcorrection in either direction.

The in-episode behavior of both trained policies are illustrated in the representative episodes below (Figure 2). The agent should closely match the liability option value, leaving some room for lapse and death probabilities.

The graph illustrates that the default parameterization (256 batch, 0.0003 LR, without action scaling) of the SAC algorithm fails to follow the target as closely as the model with larger batch size, lower learning rate and action scaling of 2.0 and consistently falls short of the liability option value (orange line).

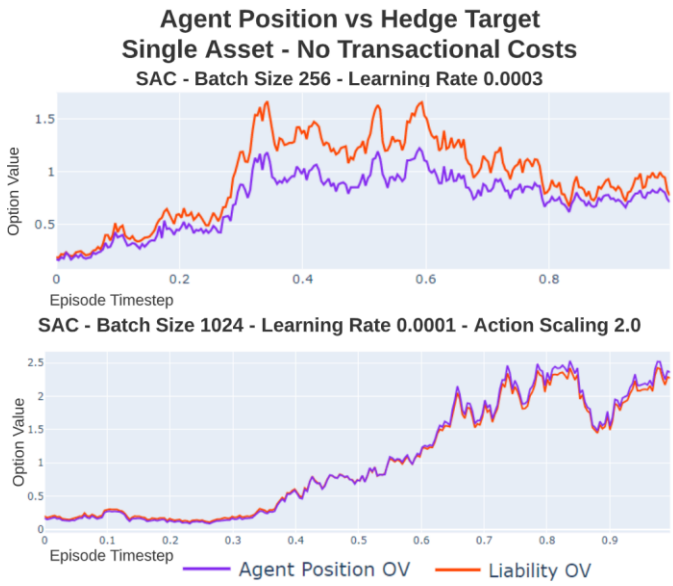

Figure 2

Figure 3 demonstrates the average episode reward over 3000 evaluation episodes with the default parameterization (green line), adjusted parameterization (red line) and the static benchmark strategy (green line). The default parameterization consistently under performs the agent with incorporated adjustments and the benchmark. The adjusted agent falls just short of the benchmark performance of the static hedge strategy, which is strong performance given that the benchmark is constructed with the 'optimal' notional using knowledge of the underlying envionment transition probabilities.

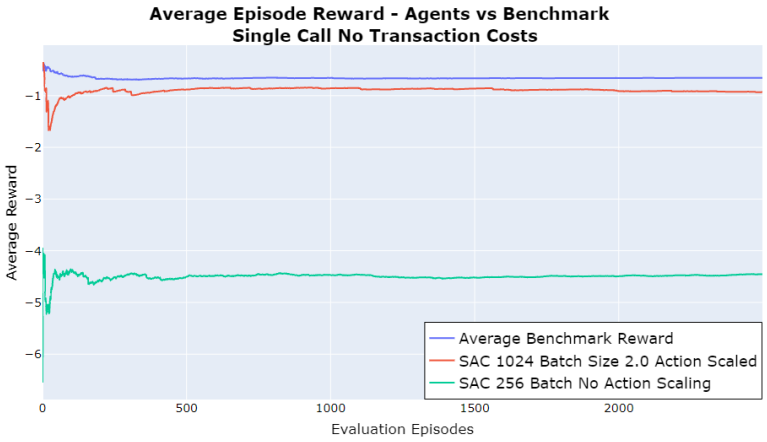

Figure 3

5.2 SAC Hedging with a Call Option & Transaction Costs

5.2.1 Experiment Setup

The SAC algorithm performs well in the environment where there are no transaction costs, but this setup does not reflect the realities of real world markets wherein adjusting a position in an asset incurs a cost. To reflect a more 'real world' scenario, a transaction cost equal to 5% of the current value of the call option is incurred for each change in position in the hedge asset (e.g. changing from 0.1 to 0.15 incurs a cost of 0.05 * 5% of current state option value). As such, the agent needs to balance adjusting its position to hit the hedge target with the negative reward incurred by doing so.

To maintain the Markov property of the task, the state space was extended to include the current asset holding to allow the agent to determine if it should increase or decrease its position. Since the negative reward incurred by changing asset position depends on the current position and new position, by including current position in the observation space the agent has the necessary information to adequately decide whether to adjust its existing position.

5.2.2 Algorithm Adjustments

Using the best performing training configuration from the previous experiment, without further adjustment, the SAC algorithm performed poorly when its actions incurred a cost.

Generally, the optimal approach with transaction costs is to take a static position in the hedge asset and adjust position only when the expected gain in doing so offsets the cost. SAC struggles in this regard in that the actor is stochastic and therefore compelled to take a non-zero action at each timestep and thus incur a negative reward.

SAC makes some progress toward learning early in the training process as the exploration reward from the entropy term offsets the negative reward from taking an action, but as the alpha parameter reduces, training becomes unstable and the actor falls into a suboptimal policy and is unable to learn (illustrated the representative episodes in Figure 4 below).

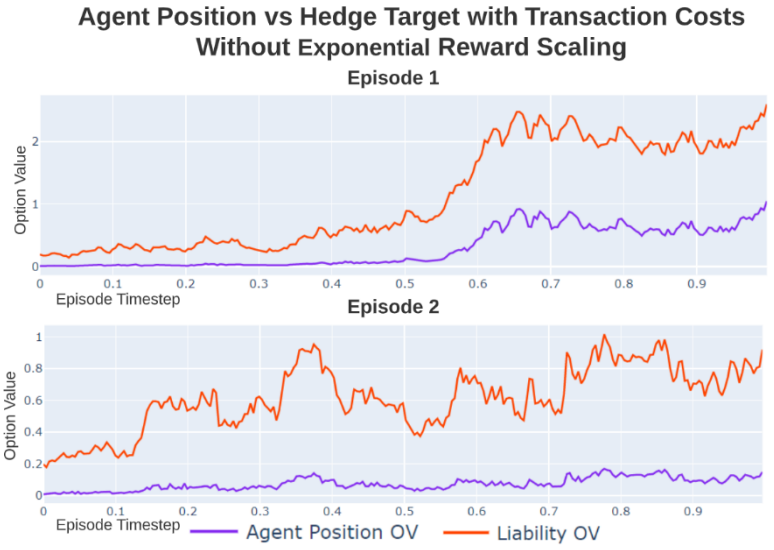

Figure 4

Based on the behavior of the agent, it appears the agent is preferring to not change its call position due to the negative reward incurred by doing so unless there is a significant difference in the agent's position vs the liability (see episode 1 above).

Increasing the alpha scaling parameter to encourage exploration did not improve performance, instead encouraging the actor to simply behave more randomly.

To remediate this weakness, an exponential adjustment to the reward function was explored.

Exponential Scaling of Hedge Mismatch Reward

The author comments in the original SAC paper that the SAC algorithm is very sensitive to reward signal. Consequently, an adjustment to the reward function, to place greater negative reward on missing the hedge target than incurring transaction costs was explored, the updated reward function took the following form:

$$R_t^a = -\left|e^{|\theta(t-1)\cdot\Delta_{t+1}{}^{OV}{}_{Assets} - \Delta_{t,t-1}{}^{OV}{}_{FIA}|} - 1\right| - Trans\ Costs$$

The negative reward signal from missing the hedge target is scaled exponentially. This reward scaling was **not** applied in the lapse/death transitions where the exponential scaling results in extreme negative rewards in these events.

This adjustment to the transition reward would allow exploration (via the entropy term) so long as that exploration did not significantly shift the agent's position too far from the liability target.

5.2.3 Model Performance

Implementing rewarding scaling was the most effective strategy in encouraging the algorithm to balance the trade off between matching the hedge target and incurring transactional costs. With exponential scaling, the algorithm achieved comparable performance to that of the trained model in the environment without transactional costs.

An illustrative example of the learned policy with reward scaling is shown in the figures below:

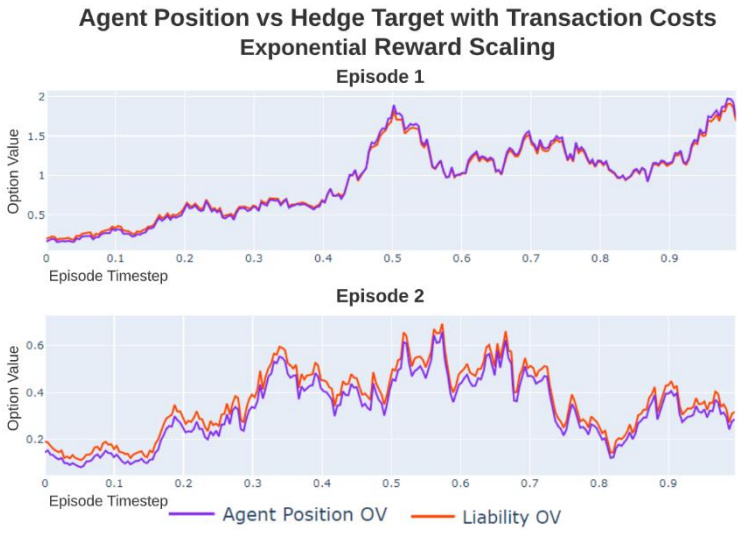

Figure 5

Interestingly, in training, the reward scaling produced volatility in critic loss due to the extreme negative reward signals from mismatch and encouraged a higher alpha parameter throughout training (see appendix 8.4.1).

A comparison of the best models trained with and without exponential reward scaling can be found in Figure 6. Without reward scaling the agent performs poorly, with reward scaling the agent closes the gap significantly to benchmark performance (blue line).

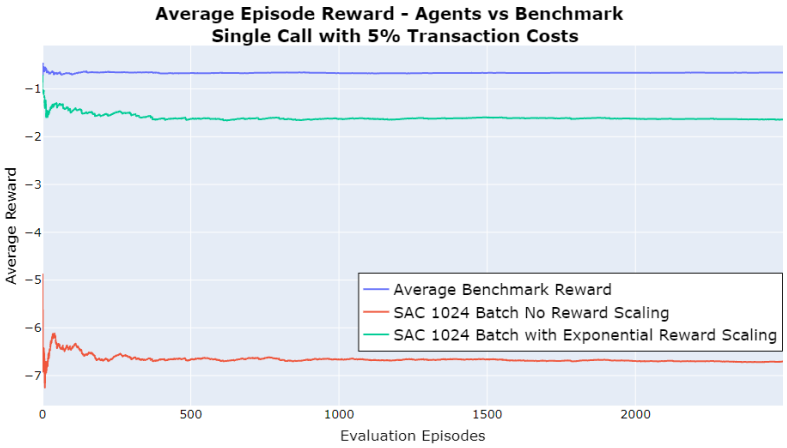

Figure 6

5.3 SAC Hedging with a Call Option & Stock with Transaction Costs

5.3.1 Experiment Setup

In the final experiment, the environment was extended to include an additional hedge instrument (the underlying index/stock price) in addition to the replicating call option. Changing position in the underlying stock incurs no transaction costs. This simulates a real world practitioner's approach to hedging where, to avoid transaction costs, the hedger will take a static hedge in the option and adjust the position over time with the underlying if experience deviates from expectation.

This grants the agent the ability to take a more advanced form of hedging, using the underlying index/stock to 'dynamically' hedge the liability.

5.3.2 Algorithm Adjustments

SAC performs relatively poorly in this problem setup compared to the task with a single call option with transaction costs. This is perhaps reflective of the increased complexity of the environment, in that the algorithm needs to learn how taking different positions in either asset affects the ultimate reward.

Similar adjustments were explored to those used in the previous experiments (exponential reward scaling, alpha tuning, action scaling and increased batch sizes). These adjustments did not improve performance to the same degree as with the experiment with a single asset and transaction costs.

In contrast, exponential reward scaling decreased rather than increased model performance, perhaps due to large negative rewards preventing model exploration to find a good policy.

Interestingly, the adjustment to the algorithm that improved performance the most in this task was removal of the current asset position from the state space observation. Since changing the asset position causes a negative reward via transactional costs, to maintain the Markov property, initial experiments included the current asset holding in the state space observations.

Removing the current asset position from the current state space observation increased model performance in the task significantly. This instability may possibly have been caused by the correlation between the state space observation and the policy action.

The in-episode performance of each of the three algorithms tested (with and without exponential reward scaling including current asset position and excluding asset position) is shown in Figure 7.

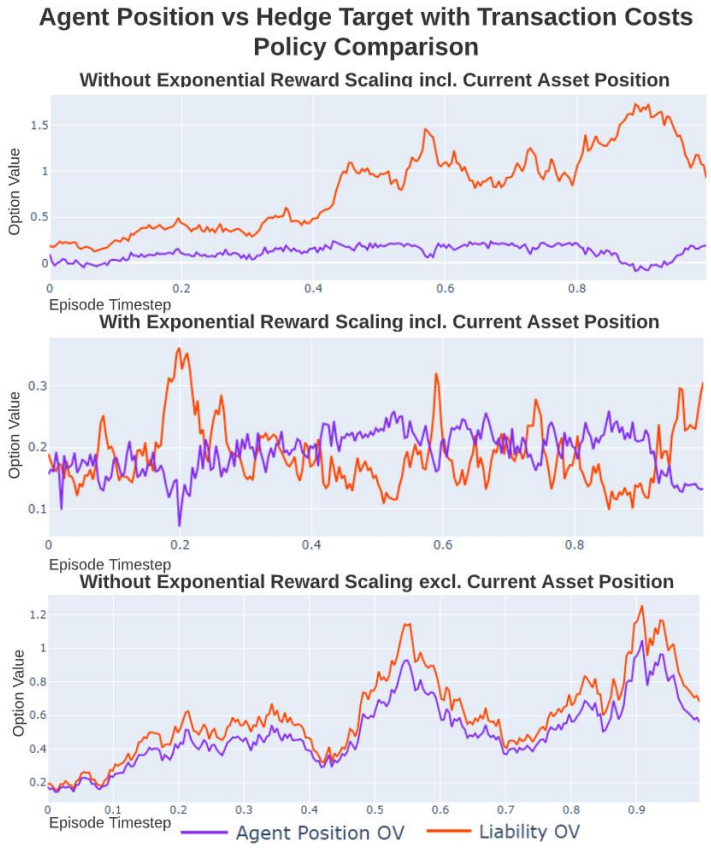

Figure 7

5.3.3 Model Performance

The average performance of the three training approaches, relative to the benchmark, is shown in Figure 8 below. Despite the adjustments to state space and algorithm, the best performing model still underperformed the model in the task involving only a single hedge asset.

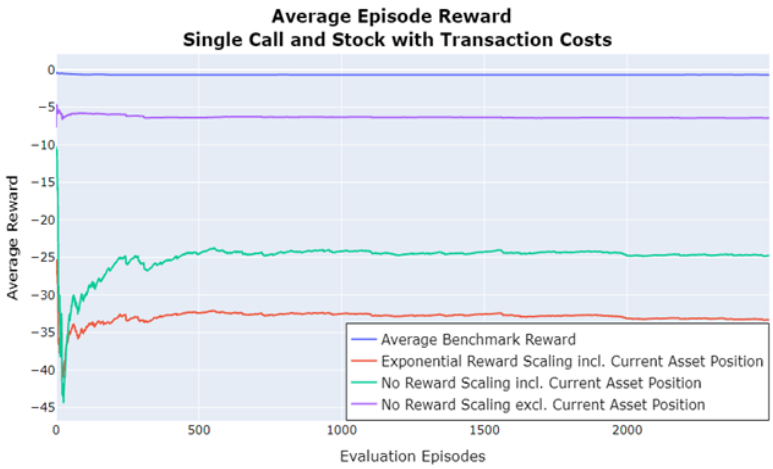

Figure 8

6. Conclusion

In the hedging application explored, the maximum entropy reward framework paired with a stochastic actor, unique to SAC, enabled fast, sample efficient learning of the task, at a cost of sacrificing optimal behavior in the resultant policy in the explored task.

The chosen RL task is particularly challenging for an algorithm to learn in that the environment and model is complex and state observations (option values) can be noisy estimates. While the action space is continuous, the optimal decision, where an action incurs a negative reward (transaction costs), is often to take infrequent actions (e.g. hold a static hedge position).

The stochastic nature of SAC enables the agent to explore and learn in a sample efficient manner within the environment; however, this stochasticity also serves as a key disadvantage in the task where the optimal choice may be to take infrequent actions rather than continuous actions at every timestep.

This stochasticity can be controlled to an extent during training by balancing the reward signal from the entropy term with the alpha parameter. However, due to the Gaussian nature of the policy, SAC is not capable of taking a single position and holding until a change in action is warranted, and instead will always attempt to make small changes at each timestep.

The SAC algorithm very quickly learned to efficiently hedge the liability when it had a constrained state space and its actions incurred no cost but struggled to balance exploration and optimization when presented with a scenario where its actions would generate a negative reward. The key to encouraging a desired outcome in this scenario was in taking advantage of SAC's sensitivity to reward signal. By exponentially scaling the cost of a hedge mismatch, SAC performance improved significantly in balancing the negative reward cost of adjusting its position every timestep.

The SAC algorithm performed poorly in the environment where it had access to multiple hedge instruments in the presence of transaction costs. Further task specific adjustments to the algorithm could be explored to potentially remediate the encountered issues e.g. more training epochs and incorporating exponential reward scaling toward the end of the training routine.

In light of SAC's weaknesses in the task, a Deep Deterministic Policy Gradient (DDPG) approach with a deterministic actor may be able to achieve higher performance. However, the training instability and the high level of hyper-parameter tuning commonly necessary with DDPG prevented exploration of this alternative approach within this project.

While a practitioner should likely avoid directly using SAC learned policies for hedging, the performance illustrated in this project implies that the provided SAC algorithm, or some variation, could be used as a tool to evaluate proposed hedging strategies. A potential area for further research in this vein may be the direct use of the learned Q-functions for evaluating existing hedge strategy actions or current positions and proposed actions.

Though the learned policies in this project did not outperform a hedging strategy developed via conventional means, the model free reinforcement learning algorithm was able to effectively hedge the complicated path-dependent exotic FIA liability. The results of the project highlight the potential for deep reinforcement learning applied to complex financial hedging tasks.

6. Acknowledgement

The following acknowledgements are necessary in support of this project; Sam Fieldman provided good feedback on the necessary level of hedging specific background required for the write up. Open AI's spinning up documentation [9] on the algorithm was useful in highlighting the min double Q trick. Tuomas Haarnoja's official implementation [2] & [3] were key in determining subtle requirements in the algorithm implementation necessary for stable learning (e.g. working with log std dev rather than std dev and clamping the output).

7. References

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", 2018

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, "Soft Actor-Critic Algorithms and Applications", 2018

[4] Heston, Steven L "A closed-form solution for options with stochastic volatility with applications to bond and currency options", 1993, (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.139.3204&rep=rep1&type=pdf)

[5] Diederik P Kingma, Max Welling 'Auto-Encoding Variational Bayes' 2013

[6] John C Hull, Options, Futures, and Other Derivatives, 10th Edition, 2018

[7] Paul Glasserman, 'Monte Carlo Methods in Financial Engineering'

[8] Jay Cao, Jacky Chen, John Hull, Zissis Poulos 'Deep Hedging of Derivatives Using Reinforcement Learning' 2019

[9] Open AI, Spinning Up Documentation https://spinningup.openai.com/en/latest/algorithms/sac.html

[10] Scott Fujimoto, Herke van Hoof, David Meger 'Addressing Function Approximation Error in Actor-Critic Methods'

8. Appendix

8.1 Individual Student Contributions in Fractions

This project was entirely my own work.

8.2 Training Routine Pseudocode
(Taken from [3] – the original paper):

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
  for each iteration **do**
    for each environment step **do**
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
    end for
    for each gradient step **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
      $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
      $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
    end for
  end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

8.3 Model Parameterization & Episode Simulation

Episodes of index returns and option values were pre-generated; the hedge environment stochastically generated an episode using a pre-generated underlying scenario and randomly generated transitions at training time. Each reinforcement model was evaluated using the same base 30K pre-generated scenarios and option values.

The scenarios were generated using Cuda compiled from the Numba framework using an Euler discretization method.

All scenario values were normalized using the initial stock price in the scenario (to constrain the input to the neural network to improve training stability).

Simulation parameters used for Heston Market Model (with Euler discretization):

Heston:
Timesteps ($dt = 3{,}000$)
$\rho = 0.8$
$v_0 = 0.25$
$S_0 = 100$
$\theta = 0.25$
$\kappa = 3$
$\xi = 0.3$
Simulations per OV = 35,000

SAC Parameters:
Optimizer – Adam (for Alpha, Actor & Q-Network optimization)
Learning rate – 0.0001
Alpha – 0.2~0.02, Target Alpha = dimension of the action space (per author's suggestion)
Batch Size - 1024
Architecture (identical to author's approach):
Actor Network – 3 dense layers, 256 nodes, ReLU Activation
Critic Networks: Comprised of 2 Q-Networks each (double Q-learning) with the following architectures:
Q-Networks – 3 dense layers, 256 nodes, ReLU activation

8.4 Training Results Graphs

8.4.1 Single Asset with Transactions Experiment Results Graphs

Illustrating reward impact / training (critic loss impact) when using exponential reward scaling.