Hedging an Investment With Deep Deterministic Policy Gradient (DDPG)

Julius L.H. de Clercq
Business Data Science
Erasmus University Rotterdam
Burgemeester Oudlaan 50, 3062 PA Rotterdam
`j.l.h.de.clercq@businessdatascience.nl`

Abstract

A Deep Deterministic Policy Gradient (DDPG) algorithm is used to find an optimal policy for weekly hedging of an investment in an exchange traded fund (ETF) by shorting a portfolio of other ETFs. The DDPG algorithm's policy function is not found to converge, due to convergence of its predicted Q-values to the unconditional mean of rewards of the initial policy. As a result, the DDPG algorithm does not outperform a vector autoregressive (VAR) model used as benchmark. Modifications to the DDPG algorithm are proposed to overcome this faulty convergence and to make it computationally more efficient for perfectly myopic optimization.

1 Introduction

Asset pricing theory asserts that a rational investor can have two objectives: maximize return, or minimize risk. Risk is widely accepted to be represented by the variance of returns. Hedging risk therefore implies minimizing the variance of returns. An investor can hedge the risk of an investment by short-selling a portfolio F, that replicates the vector of returns of the initial investment, denoted by S. The better this portfolio replicates the returns, the better is its ability to minimize the variance of the original investment[1].

A portfolio is a linear combination of investments, which means that the replicating portfolio's return vector F can be represented as the sum of the candidate hedging asset return vectors and their respective investment weights. The return vector of the hedged portfolio H therefore has the form:

$$H = S - \sum_{i=1}^{N} \omega_i F_i = S - \omega' \mathbf{F} \ , \qquad (1)$$

where S is return vector of the the hedged investment, N is the number of candidate hedging investments, ω_i denotes the weight of investment i and F_i is its return vector. ω denotes the vector of weigths and $\mathbf{F}$ the matrix of hedging investment returns. The objective is therefore to minimize the

[1]The code for this project can be found at: `https://github.com/juliusdeclercq/ETF_DDPG`.

Preprint. Under review.

variance of H by choosing the optimal portfolio weights[2]:

$$\min_{\omega} \mathbb{V}\text{ar}\left(H\right) = \min_{\omega} \mathbb{V}\text{ar}\left(S - \omega' F\right) . \tag{2}$$

One can find this replicating portfolio retrospectively through linear regression of the historical returns of the original investment S on the candidate hedging investments. The found coefficients can be used directly as the (negative) investment weights in the hedging portfolio. While this task appears simple enough, the real challenge is not to solve this retrospectively, but to solve this prospectively: the goal is to minimize $\mathbb{V}\text{ar}\left(H\right)$ for the next period, not for the past. Also, rebalancing the portfolio may be costly or infeasible to do in high frequency, so a realistic challenge would be to hedge on a weekly frequency. Prospectively, the task is significantly more challenging, and though many forecasting models have been developed, no definitive method exists. As the problem encompasses the transition from one state to the next, an action to be played (being the hedging portfolio weights), and a reward that can be obtained, this problem can be modelled as a Markov decision process (Bellman, 1957), which makes it a suitable environment for a reinforcement learning algorithm. In this paper we investigate whether reinforcement learning can be employed for tackling the hedging problem, where an exchange traded fund (ETF) is taken as initial investment, and a hedging portfolio can be constructed from a set of six other ETFs. Particularly, a Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2015) is employed, and tested against a vector auto-regressive (VAR) model.

The analysis finds that the DDPG algorithm is unable to learn a better policy than its initial configuration, and therefore is not able to outperform the VAR-model. Closer inspection finds that this is because the output of the critic network, which would ideally learn to forecast the reward, does not converge to the forecast of the reward, but rather to the unconditional mean reward, which is stationary as long as the actor is not updated, which does not happen when its loss function is constant: this is a self-reinforcing feedback loop. Learning the reward function proves to be too complex, and therefore too improbable to pre-empt convergence to the local minimum of the loss function at the unconditional mean. Adjustments to the algorithm are proposed to both overcome this issue, and to make the algorithm more efficient. The model is nevertheless tested against a benchmark, which is able to yield satisfactory rewards.

2 Method and data

2.1 Environment formulation

We consider the hedging problem for a set of seven exchange traded funds (ETFs), for which the returns are observed on a daily frequency, and of which one is taken to be the intitial investment and the rest is used as candidate investments for the hedging portfolio, implying $N = 6$. ETFs are chosen for the hedging problem, because such funds are portfolios of individual assets, and therefore better diversified and less sensitive to idiosyncratic variance. Also, some correlation with the initial investment is necessary for the assets to be of any value for the hedge, but the assets should also be sufficiently uncorrelated for the hedging problem to be a challenge. As each ETF is constructed around a specific sector or aspect of the economy, their degree of correlation is easily interpretable. The *Consumer Staples Select Sector SPDR Fund*, with ticker "XLP", is chosen as the initial investment. Table 1 describes the ETFs that are used for the hedging portfolio.

[2]Note that the monetary value that these investments represent are relative to the size of the initial investment, which has a weight of 1, meaning that if the initial investment constitutes $\$100M$, and $\omega_i = 0.1$, then this implies a short-sale of $\$10M$ in asset i.

Ticker	Description
TAN	Invesco Solar ETF
TFI	SPDR Nuveen Bloomberg Municipal Bond ETF
UGA	United States Gasoline Fund
UNG	United States Natural Gas Fund
XSD	SPDR S&P Semiconductor ETF
IHE	iShares U.S. Pharmaceuticals ETF

Table 1

Ticker-identifiers and descriptions of the exchange traded funds (ETFs) used for the hedging portfolio.

Rebalancing the portfolio may be costly, so continuous or high-frequency rebalancing is deemed unrealistic. Instead, the problem is formulated such that the portfolio may be rebalanced on a weekly frequency. The ETF returns are not available for all days of the year, because the markets are closed during the weekends and on holidays. Hence, rebalancing is said to be possible over the weekend and the variance of the investment is considered for the trading days of each week.

As stated in the introduction, the hedging problem is modelled as a Markov decision process (MDP), with state space $\mathcal{S}$, action space $\mathcal{A} = \mathbb{R}^N$, reward function $r(s_t, a_t)$, and transition dynamics $p(s_{t+1}|s_t)$. Each action $a \in \mathcal{A}$ is a $N \times 1$ vector of portfolio weights. Every state $s \in \mathcal{S}$ is the historical returns of the ETFs in the portfolio: $s = (S, \mathbf{F})'$. The reward function is defined as $r(s_{t+1}, a_t) = -\mathbb{V}\text{ar}\,(S_w - a_w' \mathbf{F}_w)$, where the index w denotes observations of the most recent week at time $t+1$. The reward is negative variance to indicate that maximizing reward implies minimizing the variance.

For computational purposes, the states need to have a fixed shape, which means that a window size must be chosen, determining how many lagged observations can be observed in each state. For daily return data, a lag window of 100 to 200 observations is generally accepted as an agreeable compromise between keeping sufficiently many observations for forecasting purposes, and disregarding old, uninformative observations. Therefore, in this analysis a lag window of 150 observations is taken, meaning that every state has shape 150×7, where the columns are ordered $(S, F_1, F_2, \dots, F_6)$. Note that this formulation regards all trading days to be consecutive: the distance between Fridays and Mondays are assumed to be equal to the distance of Mondays to Tuesdays. This can be considered unrealistic, as it may be reasonable to expect more information to become available over the weekend than overnight. But as markets are by definition closed outside of trading days, this is not deemed to be a threatening oversimplification, while loosening the assumption would pose a large modelling challenge in itself.

The data set comprises daily observations for the seven ETFs, excluding non-trading days, from $2008/04/21$ until $2023/12/29$. The hedging problem-environment is set up such that the first time step of the MDP occurs in the first week of 2009, because this is one of the first weeks for which 150 lagged observations are available, as are needed in each state. The data of the year 2023 is omitted from the training data, such that it can serve as an out-of-sample prediction set. In total this gives 729 weeks in the training data and 51 in the test data that are viable time steps for the MDP.

2.2 Deep Deterministic Policy Gradient

To find a suitable reinforcement learning algorithm, we first observe that we have a continuous, $N = 6$-dimensional action space. This already makes a range of algorithm classes unsuitable, as most reinforcement learning algorithms are only applicable to discrete action spaces of fairly limited size. Deep Q-learning, also known as Deep Q-Networks (DQN), for instance, was developed for action spaces comprising at most sixteen discrete actions (Mnih et al., 2015). A class of reinforcement learning algorithms that is adaptable to continuous action spaces, are policy gradients methods (Sutton and Barto, 2018). Policy gradient algorithms work not by choosing an action by maximizing a value function, but from a policy function $\pi(s_t)$ that maps the state to the probability of choosing an action: $\pi_\theta(a|s) = \mathbb{P}[a|s, \theta]$, where θ denotes the set of policy function parameters. Value functions may still be used to learn the optimal policy, but are not directly needed to choose an action. Methods that learn approximations to both policy and value functions are known as *actor-critic* methods. Policy gradient methods are traditionally stochastic: the policy function outputs a probability distribution over the possible actions, and selects an action by sampling from this probability distribution. But for stochastic policy gradients, the action space must be finite, and therefore discrete. However, policy gradients are adaptable to the continuous action space by using the policy function to choose an action directly: $a = \mu_\theta(s)$, which is known as the deterministic policy gradient (DPG) function, developed by Silver et al. (2014). μ is deterministic because action selection is no longer based on probability, so given a fixed state s and parameter set θ, the policy function deterministically selects action a.

Though the DPG method permits selection of actions in a (high-dimensional) continuous action space, Mnih et al. (2015) have shown that deep reinforcement learning can be very effective at optimal action selection. Lillicrap et al. (2015) developed an algorithm that combines DQN and DPG: Deep Deterministic Policy Gradient (DDPG), described by Algorithm 1. In a nutshell, the goal of DDPG is to learn the optimal (deterministic) policy function $\mu_\theta(s)$, which is defined as some neural network (the architecture used in this paper is described in Section 3). The policy function is trained by optimizing a predicted value function $Q(s, a)$, which also has a deep learning architecture. This combination of policy and value functions make this an actor-critic method, where the actor is the policy function μ and the critic is synonymous with the value function Q. The critic network is trained by minimizing the squared Euclidean distance between the predicted value $q = Q(s_t, a_t)$ and target value $q' = r_t + \gamma Q'(s_{t+1}, (\mu'(s_{t+1}|\theta^{\mu'}))$, where r is the observed reward and $\gamma \in [0, 1]$ is the discount factor inherited from the Bellman equation-form of the value function. The target values are not computed using the same actor and critic networks, because as these networks are continuously updated, the target values would be unstable, leading to divergent behaviour in the updating of the critic network. Instead, the target values are computed using target-actor μ' and target-critic Q' networks. These target networks are initialized as copies of the actor and critic networks, and are subsequently soft-updated towards the current actor and critic conform an updating rate $\tau \in [0, 1]$. For the sake of stability of the target values, τ needs to be set to a value close to zero. We therefore follow Lillicrap et al. (2015) by setting $\tau = 0.001$. Also following their implementation, the actor and critic networks are initialized with random parameter vectors θ^μ and θ^Q, but initialization of the final layer of the networks is done by taking draws from a uniform distribution $[-10^{-3}, 10^{-3}]$, to make sure that the initial output of the networks is near zero, helping prevent exploding gradient issues (Glorot and Bengio, 2010).

Optimization of the actor and critic network is performed by taking batches of experiences from the algorithm's memory. This memory is called the experience replay buffer $\mathbf{R}$, where each experience is a tuple (s_i, a_i, r_i, s_{i+1}) , $i \in \{1, \ldots, T\}$. The replay buffer has a limited size, in this analysis set to 10^6. If the replay buffer is full, the oldest experiences are erased from memory. At each training iteration, which occurs every time step once the replay memory contains as many experiences as the pre-specified batch size B ($|\mathbf{R}| \geq B$), a random batch of experience are selected from the replay buffer without replacement, and the loss functions of the actor and critic networks are computed for this batch of experiences. This method of experience replay batching was proposed by Mnih et al. (2015), and serves the purpose of breaking the temporal correlation of consecutive experiences, which allows for more stable and effective learning. The batch size B is set to 32. The algorithm runs for multiple cycles through the training data, where each cycle is known as an episode. After each episode the policy network is tested on the out-of-sample observations.

Algorithm 1 DDPG algorithm (Lillicrap et al., 2015)

```
Randomly initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s|\theta^Q)$ with weights $\theta^\mu$ and $\theta^Q$
Initialize target networks $\mu'$ and $Q'$ as copies of the actor and critic networks
for $i \leftarrow 1,\ M$ do
    Initialize exploration noise $\mathbf{N}$
    Receive initial observation state $s_1$
    for $t \leftarrow 1,\ T$ do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathbf{N}_t$
        Execute action, observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathbf{R}$
        if $|\mathbf{R}| \geq B$ then
            Sample random minibatch of $B$ transitions $(s_i, a_i, r_i, s_{i+1})$
            Set target q-value $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
            Update critic by minimizing the target loss: $L = \sum_{i=1}^{B}(y_i - Q(s_i, a_i|\theta^Q))^2$
            Update the actor using the sampled policy gradient:
                $\nabla_{\theta^\mu} J \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i)\nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$
            Update the target networks:
                $\theta^{Q'} \leftarrow \tau \cdot \theta + (1-\tau) \cdot \theta^{Q'}$
                $\theta^{\mu'} \leftarrow \tau \cdot \theta + (1-\tau) \cdot \theta^{\mu'}$
        end if
    end for
end for
```

2.2.1 Exploration noise

The core of reinforcement learning is to strike a balance between exploration and exploitation. The agent needs to explore the action space to be able to gain experience with its environment, and to consequently learn to play better actions that yield greater rewards. Due to the use of a deterministic policy function μ, no exploration of the action space is committed other than the actions that are played in the path to convergence to the loss-function minimizing policy, which implies that little of the action space would be explored. To facilitate exploration of the continuous action space, noise is added to the policy's action. This noise must be a stochastic process $\mathbf{N}$, which needs to be zero-centered, as it would otherwise introduce bias to the actions, causing the actions to drift up or down in its respective dimensions. Following Lillicrap et al. (2015) and originally proposed by (Wawrzynski, 2015), the exploration noise is defined as an Ornstein-Uhlenbeck process (Ornstein and Uhlenbeck, 1930), which is a physical model of Brownian motion with drag, defined by the following differential equation:

$$dx_t = -\delta x_t dt + \sigma dW_t\,, \tag{3}$$

where δ is a drag coefficient, σ is a variance coefficient and W_t is a Wiener process with unit variance. This type of noise is chosen here simply to keep in line with the original DDPG methodology, but its original application was in robotics simulation, where physical processes are modelled, and inertia needs to be accounted. For that reason, an autocorrelated noise process was introduced. Arguably, that is not likewise applicable to this setting of hedging, as no physical aspect of inertia exists within this context. However, it can be considered a realistic feature, as this type of autocorrelated noise implies that exploration is dampened after it has been erratic; modelling a certain degree of conservatism. Exploration noise parameter values are taken from Lillicrap et al. (2015): $\sigma = 0.2$ and $\delta = 0.15$. An initial value x_0 is also needed for computing the noise, which their paper does not specify, but this can only be zero as otherwise the noise would not be zero-centered. When testing the model on the out-of-sample observations, no exploration noise is added to get a clear assessment of the policy.

2.2.2 Zero-valued discount factor

The foremost hyperparameter to discuss is the discount factor γ, which represents the degree of orientation on future rewards of the algorithm: a γ close to 1 means that the algorithm is highly forward-looking, assigning great value to rewards many time steps ahead, and, conversely, a γ close to zero implies that the algorithm is myopic; meaning that it is short-sighted and prioritizes immediate rewards. In the hedging problem, the transition to the next state is independent from the action, and therefore future rewards are also independent of the current action. Because of this independence, it does not make sense for the algorithm to consider how a current action may yield future rewards, meaning that the only sensible value to set for γ is zero.

This causes a profound simplification of the algorithm, as described in Algorithm 2. Because $\gamma = 0$, the target value used for optimizing the critic is now equal to the reward, because of this, the target networks are obsolete, so any operation involving the target networks can be omitted; also making the τ parameter redundant. This implies that for reinforcement learning settings with a zero-valued discount factor, DDPG can be simplified to Algorithm 2 to avoid redundant computation.

Algorithm 2 DDPG algorithm with $\gamma = 0$

```
Randomly initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s|\theta^Q)$ with weights $\theta^\mu$ and $\theta^Q$
for $i \leftarrow 1,\ M$ do
    Initialize exploration noise $\mathbf{N}$
    Receive initial observation state $s_1$
    for $t \leftarrow 1,\ T$ do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathbf{N}_t$
        Execute action, observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathbf{R}$
        if $|\mathbf{R}| \geq B$ then
            Sample random minibatch of $B$ transitions $(s_i, a_i, r_i, s_{i+1})$
            Set target q-value $y_i = r_i$
            Update critic by minimizing the target loss: $L = \sum_{i=1}^{B}(y_i - Q(s_i, a_i|\theta^Q))^2$
            Update the actor using the sampled policy gradient:
                $\nabla_{\theta^\mu} J \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i)\nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$
        end if
    end for
end for
```

2.3 Benchmark algorithm

To put the performance of the DDPG algorithm in perspective, a benchmark model needs to be employed. The algorithm used as benchmark is a composite Vector Autoregressive (VAR) - Ordinary Least Squares (OLS) model. This algorithm fits a VAR(1) model on the state, and uses it to compute a five-step ahead forecast. This forecast is then appended to the last 25 observations from the state. On this combined data, an OLS model is fitted, taking S as dependent variable and the return vectors **F** as independent variables. The coefficients from this OLS regression are directly taken as ω vector, and is the action taken by the benchmark.

3 Results

Optimization of the actor and critic networks is conducted using the Adam optimization method (Kingma and Ba, 2014), with distinct learning rates for the actor and critic networks: $\eta_\mu = 10^{-4}$

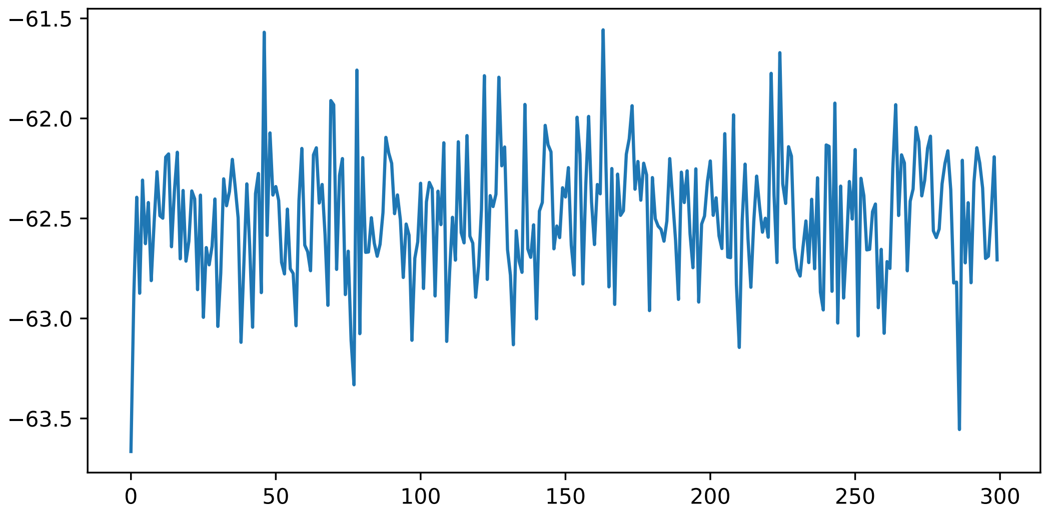

Figure 1. Mean prediction rewards per episode for the long experiment.

and $\eta_Q = 10^{-3}$, respectively. The actor both the actor and critic networks are defined as densely connected feed-forward neural networks with two hidden layers. The actor network has 64 hidden units in the first layer and 32 in the second layer, and the critic network has 128 units in the first layer and 64 in the second layer. The initial input of both networks is the 150×7 state matrix, which is first vectorized before being passed to the neural network. The critic network also takes the action as input, which is not included until the second hidden layer, to mitigate the vanishing gradient problem w.r.t. the actions. The rectified linear unit (`ReLU`) function (Glorot et al., 2011) is used as activation function in all hidden layers. The activation function for the output layer of the actor network is the hyperbolic tangent (`tanh`). The `tanh` function is used for the actor network to make sure the predicted weights lie in the interval $(-1, 1)$. Theoretically, any real valued weight would be allowed, but in practice weights are only sensible within a range of $(-2, 2)$, and the sum of weights should be around one. To allow the actor the freedom to play actions within this range, the action vector is subsequently scaled by two.

3.1 Stalled learning

Preliminary experiments indicated that no learning was taking place, as the mean prediction reward per episode was not increasing. To investigate this issue, grid search was conducted on the critic learning rate $\eta_Q \in \{10^{-2}, 10^{-3}, 10^{-4}\}$ and the exploration noise variance scalar $\sigma \in \{0.1, 0.2\}$, with fifteen episodes of runtime per hyperparameter-set. None of the combinations of these hyperparameter values yielded an instance of increasing rewards. One long experiment was run with 300 episodes, to investigate whether learning would happen in some later stage[3]. The mean rewards per episode are shown in Figure 1, from which can be observed that these never exceed the range $[-64, -61]$, and no trend can be seen in the data, indicating that the rewards do not increase and therefore the algorithm also does not appear to learn over the long run.

[3]Although fifteen episodes with $T = 729$ already constitutes 10,904 training iterations, so taking 300 episodes is probably overkill.

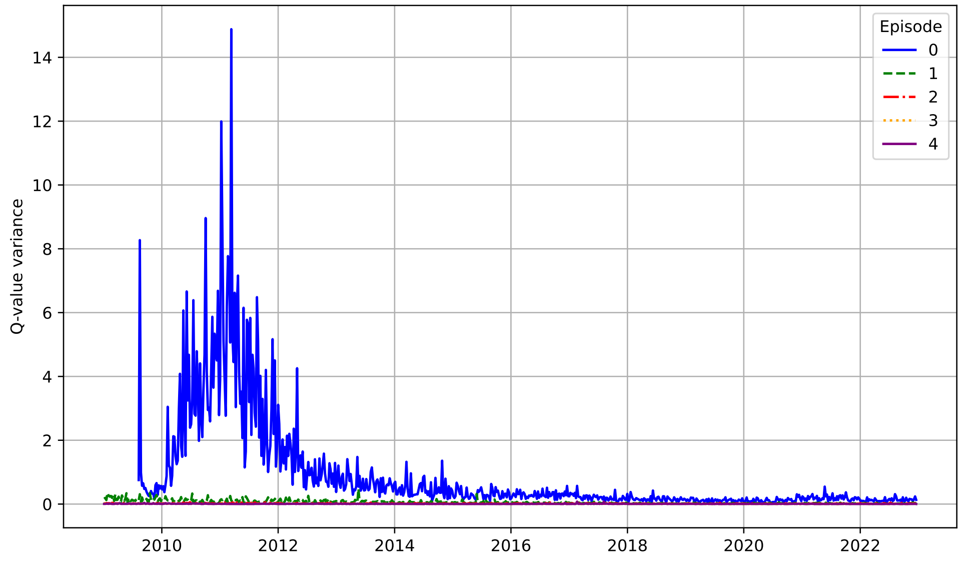

Figure 2. Variance of the Q-values computed from the sampled batch of experiences from the replay buffer per time step.

Close investigation of the trained model showed that the critic network always outputs a constant for any state-action pair. After five episodes, the Q-values of a typical batch of states and actions from replay memory was evaluated at a mean of -47.2327 and variance of 0.1373. The low variance of the Q-values indicates that they are evaluated to be approximately constant. To get a better idea of the speed of convergence to this constant, the variance of the Q-values of the batched experience from replay memory are plotted in Figure 2. The figure shows that the only sizeable variance of the Q-values occurs during the quarter of the first episode. Before the end of the first episode, the variance of the Q-values appears to have converged to zero.

Proof that the policy network is not meaningfully updated is provided by figure 3. The figure shows the mean and range of the prediction rewards achieved over the first five episodes by three DDPG models, that differ in value of the critic learning rate parameter, and a VAR model The ranges are so narrow that they are not always visible. As the range shows how the predictions differ over all episodes, the narrow ranges indicate that the predictions in later episodes do not differ from the initial predictions, illustrating that the policy network is not updated after the first episode. Due to the absence of learning, the differences in predicted actions, and therefore rewards, are only the consequence of the random initialization of actor and critic parameters θ^{μ} and θ^{Q}.

Figure 3 also shows the rewards achieved by the benchmark VAR-OLS model. The benchmark did well, achieving a mean reward of -0.384881. This shows that it is possible to consistently achieve a near perfect reward in the hedging problem.

4 Discussion

Ideally, the q-values would converge to accurate predictions of the reward, given by $r(s_{t+1}, a_t) = -\mathbb{V}\text{ar}\left(S_w - a_w' \mathbf{F}_w\right)$. Instead, the constant value at which they converge appears to be the unconditional

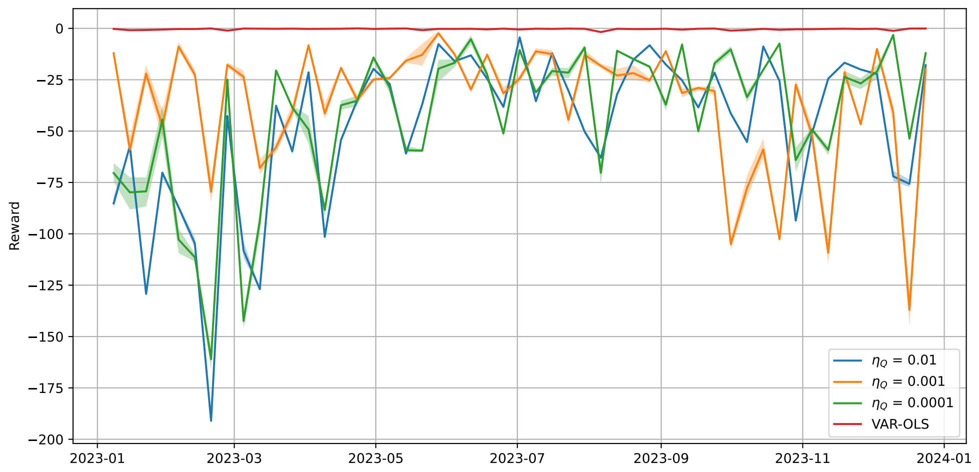

Figure 3. Mean and range of the rewards achieved at each time step in the test data over the first five episodes, by three distinct DDPG models, each differing only in value of the critic network learning rate η_Q, a single run of predictions from the VAR-OLS benchmark model.

mean reward achieved by the actions selected by the policy subject to exploration noise. If the actor - or policy- network would be learning, the rewards would not be stationary, so there would be no stationary mean reward to learn. But the actor learns by maximizing the expected reward that is computed by the critic network, so it cannot learn before the critic has learned to return an informative signal. But as the critic learns to (locally) minimize its loss function by computing the unconditional mean reward, which is a constant, the actor's learning stalls, as no change in its model parameters can yield a greater expected reward: the derivative of a constant is zero, so the policy gradients tend to zero. This is a self-reinforcing feedback loop, because as the policy is not updated, the realizations of rewards become stationary, implying that the unconditional mean reward has become a stable target for the critic to learn. The reason why the critic is unable to converge to the forecasted reward, is because the reward function is too complex to learn: just from being given the past returns and a set of weights, it needs to figure out that it is supposed to compute a forecast for the next period and from it compute equation 2. As it is too complex, convergence to this forecast is too improbable to pre-empt convergence to the local minimum of the loss function at the unconditional mean.

Not only is learning the reward function apparently infeasible due to the lure of the local minimum at the unconditional mean, it is also terribly inefficient: why endlessly train a computationally heavy algorithm to learn a complex function that we already know? The solution to both problems, is to embed the reward function in the Q function. Instead of using the network to directly compute a Q-value from a state-action pair, using only the state as input, the network can be used to output a 35×1 vector, with only the state as input, which can be reshaped to a 5×7 matrix that may represent a five-step ahead forecast of the return vectors. This matrix can then be passed with the action into the reward function, as given in equation 2. The proposed adjustment can be broken down into two parts, a forecast function:

$$\phi(s) = \hat{s}_{t+1} = (\hat{S}', \hat{\mathbf{F}}') \; , \tag{4}$$

where $\hat{S}$ and $\hat{\mathbf{F}}$ are forecasts of S and F, and which still follows a feedforward neural network architecture; and a forecasted reward function, where the forecasted Q-value Q^* is computed through the following equation:

$$Q^* = r(\phi(s), a) = -\mathbb{V}\text{ar}\,(\hat{S} - a'\hat{\mathbf{F}}) \tag{5}$$

4.1 Conclusion

The DDPG algorithm was unable to learn an effective policy due to quick convergence of the predicted Q-values to the unconditional mean reward. This problem may be overcome by implementing an adjusted critic function, given by equation 5, which would also avoid redundant learning of the known reward function. Due to the perfectly myopic setting, the DDPG algorithm for this problem can be simplified to Algorithm 2, again avoiding redundant computation.

References

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ornstein, L. S. and Uhlenbeck, G. E. (1930). On the theory of the brownian motion. *Physical review*, 36:823–841.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wawrzynski, P. (2015). Control policy with autocorrelated noise in reinforcement learning for robotics. *International Journal of Machine Learning and Computing*, 5(2):91.