NATURAL GAS STORAGE VALUATION USING DEEP REINFORCEMENT LEARNING

by

Abdulaziz Aldoseri

B.Sc., Business Information Systems, University of Bahrain, 2008

M.A., Economics, Binghamton University, 2011

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2024

NATURAL GAS STORAGE VALUATION USING DEEP REINFORCEMENT LEARNING

APPROVED BY:

Prof. Refik Güllü (Thesis Supervisor)	.....
Assoc. Prof. Mustafa Gökçe Baydoğan	.....
Assist. Prof. Mehmet Yasin Ulukuş	.....

DATE OF APPROVAL: 10.09.2024

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Prof. Refik Güllü, whose exceptional support and encouragement have been invaluable throughout my thesis. His approachable nature and insightful guidance allowed me the freedom to explore my thesis topic independently.

I also extend my sincere thanks to the committee members, Assoc. Prof. Mustafa Gökçe Baydoğan and Assist. Prof. Mehmet Yasin Ulukuş, for their constructive feedback and thorough review, which significantly contributed to the enhancement of my thesis.

ABSTRACT

NATURAL GAS STORAGE VALUATION USING DEEP REINFORCEMENT LEARNING

Energy trading involves physically handling energy commodities such as oil, natural gas, and coal. These trading activities are enabled by a complex global network of energy conversion assets such as pipelines, refineries, and storage facilities. The profitability of energy trading relies on efficiently managing these assets' operational capacity constraints. The competitive management of these assets is called merchant operations. Energy merchant companies acquire conversion assets to support their trading activities. Therefore, the proper valuation of these assets is crucial. In practice, energy merchant companies rely on heuristic methods that produce deterministic operating policies. Deterministic policies are unable to capture the so-called embedded optionality in the energy conversion assets. In this thesis, we tackle the problem of natural gas storage valuation using a novel deep reinforcement learning (DRL) algorithm called soft actor-critic (SAC). SAC utilizes entropy regularization to improve policy exploration and stability. Our results show that SAC has learned an effective operating policy during training while other state-of-the-art DRL algorithms couldn't do so in our problem.

ÖZET

DERİN PEKİŞTİRMELİ ÖĞRENME KULLANARAK DOĞAL GAZ DEPOLAMA DEĞERLEMESİ

Enerji ticareti, petrol, doğal gaz ve kömür gibi enerji emtialarının fiziksel olarak işlenmesini içerir. Bu ticaret faaliyetleri, boru hatları, rafineriler ve depolama tesisleri gibi karmaşık bir küresel enerji dönüşüm varlıkları ağı tarafından desteklenmektedir. Enerji ticaretinin karlılığı, bu varlıkların operasyonel kapasite kısıtlamalarının verimli bir şekilde yönetilmesine dayanır. Bu varlıkların rekabetçi yönetimine tüccar operasyonları denir. Enerji tüccarı şirketleri, ticaret faaliyetlerini desteklemek için dönüşüm varlıkları edinirler. Bu nedenle, bu varlıkların doğru değerlemesi çok önemlidir. Uygulamada, enerji tüccarı şirketleri deterministik çalışma politikaları üreten sezgisel yöntemlere güvenirler. Deterministik politikalar, enerji dönüşüm varlıklarındaki sözde gömülü opsiyonelliği yakalamakta yetersiz kalır. Bu tezde, doğal gaz depolama değerlemesi problemini, yumuşak aktör-eleştirmen (SAC) adı verilen yeni bir derin pekiştirmeli öğrenme (DRL) algoritması kullanarak ele alıyoruz. SAC, politika keşfini ve istikrarını iyileştirmek için entropi düzenlemesini kullanır. Sonuçlarımız, SAC'nin eğitim sırasında etkili bir çalışma politikası öğrendiğini, diğer en gelişmiş DRL algoritmalarının ise problemimizde bunu yapamadığını göstermektedir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES vii
LIST OF TABLES ix
LIST OF SYMBOLS x
LIST OF ACRONYMS/ABBREVIATIONS xiii
1. INTRODUCTION 1
2. BACKGROUND AND LITERATURE REVIEW 4
2.1. Economics of Energy Commodities Storage 4
2.2. The Storage Problem 5
2.3. Reinforcement Learning 12
2.3.1. Markov Decision Process 12
2.3.2. Dynamic Programming 16
2.3.3. Temporal Difference 20
2.3.4. Approximation Methods 23
2.3.5. Policy Gradient Methods 28
2.3.6. RL Applications in Energy Operations 32
3. IMPLEMENTATION 35
3.1. Soft Actor-Critic 35
3.2. Implementation 38
3.3. Results 43
3.4. Validation 47
3.5. Major Observations 50
4. CONCLUSION 56
REFERENCES 57

LIST OF FIGURES

Figure 2.1. Agent environment interaction. 13

Figure 2.2. Policy evaluation. 17

Figure 2.3. Policy iteration. 19

Figure 2.4. Value iteration. 19

Figure 2.5. Temporal difference. 21

Figure 2.6. SARSA. 22

Figure 2.7. Q-learning. 23

Figure 2.8. Semi-gradient TD. 26

Figure 2.9. Semi-gradient SARSA. 27

Figure 2.10. Actor-critic. 31

Figure 3.1. SAC. 37

Figure 3.2. Sample of natural gas price information. 39

Figure 3.3. Performance of SAC. 44

Figure 3.4. Performance of SAC. 45

Figure 3.5. SAC policy type effect on performance. 46

Figure 3.6. Performance of PPO and TD3. 47

Figure 3.7. Validation performance of SAC. 48

Figure 3.8. Average policy update clipping in PPO. 52

Figure 3.9. Average PPO actor loss. 53

Figure 3.10. Average TD3 actor loss. 53

Figure 3.11. Average SAC actor loss. 54

Figure 3.12. Average SAC reward vs entropy. 54

LIST OF TABLES

Table 2.1. Example of tabular representation 24

Table 3.1. MDP specification 39

Table 3.2. Reward function specification 40

Table 3.3. Comparison of our model and ADP models 41

Table 3.4. Experiment design 42

Table 3.5. Validation setup 48

Table 3.6. Nonstationarity test results 49

Table 3.7. Comparison of algorithms 50

LIST OF SYMBOLS

a_t	Action taken at time t
$\overline{a}(x)$	Maximum buy and inject action at storage level x
$\underline{a}(x)$	Minim withdraw and sell action at storage level x
A_t	Action taken in a Markov decision process at time t
$\mathcal{A}$	Set of all actions in a Markov decision process
c^I	Marginal injection cost
c^W	Marginal withdrawal cost
C^I	Injection capacity
C^W	Withdrawal capacity
$\mathcal{D}$	Experience replay memory
$\boldsymbol{F}_t$	Futures curve at time t
$\boldsymbol{F}_{(t,t+1)}$	Prompt month $t+1$ futures price at time t
$\tilde{\boldsymbol{F}}_t$	Futures curve at time t excluding the spot price s_t and the prompt month futures price $\boldsymbol{F}(t,t+1)$
G_t	Return from time t
$H(\pi(\cdot\|s_t))$	Entropy function
$J(\boldsymbol{\theta})$	Performance measure for parameterized policy $\pi_{\boldsymbol{\theta}}$
$J(\pi)$	Maximum entropy reward function
$p(S_{t+1}, R_{t+1} \mid S_t, A_t)$	Probability of transition to state S_{t+1} with reward R_{t+1} from state S_t and action A_t
$\hat{q}(s,a,\mathbf{w})$	Approximate value of state action pair (s,a) given the parameter $\mathbf{w}$
Q	Largest common factor of flow capacities C^I and C^W and maximum inventory level $\overline{x}$
$Q(S_t, A_t)$	Estimate of the value of state action pair (S_t, A_t)
$Q(s,a,\boldsymbol{\phi})$	Soft Q-function
$r(a,s)$	Expected reward for action a given state s
R_t	Reward received from a Markov decision process at time t
$\mathbb{R}$	Set of real numbers

$\mathcal{R}$	Set of all rewards in a Markov decision process
$\tilde{s}_{t+1}$	Next period expected spot price
S_t	State of the Markov decision process at time t
$\mathcal{S}$	Set of all states in a Markov decision process
t	Timestep in a Markov decision process
U_t	Unbiased estimator for $v_\pi(S_t)$
$v(x_t, a_t, s_t)$	Value function given inventory x_t, action a_t, and spot price s_t
$v(x_t, \boldsymbol{F}t)$	Value function given inventory x_t and futures curve $\boldsymbol{F}t$
$v_\pi(s)$	Value of state s under policy π
$v_*(s)$	Value of state s under optimal policy
$\hat{v}(s, \mathbf{w})$	Approximate value of state s given the parameter $\mathbf{w}$
$V(s, \boldsymbol{\psi})$	Soft value function
$V(S_t)$	Estimate of the value of state S_t
$\overline{VE}(\mathbf{w})$	Mean squared value error given the parameter $\mathbf{w}$
$\mathbf{w}$	Weight parameter for approximating the value function
$\mathbf{x}(s)$	Feature vector for state s
$x_i(s)$	Element i of the feature vector for state s
$\overline{x}$	Maximum storage level
$\underline{x}$	Minimum storage level
$\mathcal{X}$	Set of feasible storage levels
α	Step-size parameter
γ	Discount factor
δ	TD error
ϵ	Probability of taking a random action
$\boldsymbol{\theta}$	Weight parameter for approximating the policy
$\mu(s)$	Probability distribution of state s
π	A policy
$\pi(a \mid s)$	Probability of taking action a while being in state s
$\pi(a \mid s, \boldsymbol{\theta})$	Probability of taking action a while being in state s given the parameter $\boldsymbol{\theta}$

$\pi(s)$	Action a taken in state s
$\boldsymbol{\phi}$	Weight parameter for approximating the soft Q-function
ϕ^I	In-kind injection fuel cost
ϕ^W	In-kind withdrawal fuel cost
$\boldsymbol{\psi}$	Weight parameter for approximating the soft value function

LIST OF ACRONYMS/ABBREVIATIONS

ADF	Augmented Dickey–Fuller
ADP	Approximate Dynamic Programming
ANN	Artificial Neural Networks
BI	Buy and Inject
DN	Do Nothing
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
KPSS	Kwiatkowski–Phillips–Schmidt–Shin
LP	Linear Program
LSMC	Least-Squares Monte Carlo
MDP	Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SGA	Stochastic Gradient Ascent
SGD	Stochastic Gradient Descent
TD	Temporal Difference
TD3	Twin-Delayed Deep Deterministic Policy Gradient
WS	Withdraw and Sell

1. INTRODUCTION

Energy trading involves physically handling energy commodities such as oil, natural gas, and coal. An interconnected global infrastructure of energy conversion assets enables physical energy trading, including production, storage, transport, and processing facilities. The term energy conversion asset refers to a facility that takes energy commodities as inputs and produces energy commodities as outputs through a conversion process. The conversion process can be physical, temporal, or spatial. Oil refining is an example of physical conversion, where crude oil is converted into petroleum products. Another example is oil tankers that resemble both temporal and spatial conversion. Collectively, conversion assets comprise what is called an energy trading network that matches global energy supply and demand. These assets are the backbone of the energy trading network. Hence, properly managing these assets' operational capacity constraints is crucial in energy trading.

From an asset operator's perspective, the problem is to decide on the optimal operating policy that maximizes profits. Given how commodity prices evolve, this is achieved by managing the assets' capacity limits. This perspective is referred to in the literature as merchant operations [1]. An energy merchant would actively adjust the conversion asset's operating policy to support its trading activity given the market prices. For example, an oil refinery continually optimizes its production portfolio of petroleum products according to the market's spot and futures prices. This ability to adjust the operating policy in response to changing market conditions is crucial in valuing commodity conversion assets. In the energy trading industry, it is not uncommon to fully or partially lease a conversion asset's capacity, such as a pipeline or storage facility capacity, for a given duration. An energy merchant company would need to properly assess the value of that conversion asset capacity to be leased. Practitioners in the industry use various heuristic-based methods for conversion asset valuations. These methods are primarily based on deterministic models that take as input the current futures price curve. Given the inherent volatility in commodity prices, practitioners re-

optimize their model as the commodity futures prices curve evolves. The deterministic policy derived from the heuristic methods is unable to capture the so-called embedded optionality of the conversion assets.

The task of conversion asset valuation is quite similar to the problem of sequential decision-making under uncertainty. In the stochastic optimization literature, such problems are formulated as a Markov Decision Process (MDP). Various solution approaches for this problem class were recently proposed, such as Approximate Dynamic Programming (ADP) and Reinforcement Learning (RL). The term reinforcement learning simultaneously refers to three interrelated concepts: a problem, a class of solution methods, and the field that studies both the former and the latter [2]. RL aims to maximize a reward signal from the MDP environment without explicit instructions on how to do this. RL achieves this goal by pursuing a deliberate strategy that balances exploring and exploiting actions that result in these rewards. Tabular RL methods can achieve optimality in a small MDP with finite discrete state and action spaces. However, many problems of interest, including this thesis, suffer from the so-called curse of dimensionality, which occurs when the MDP's state space is large and continuous, hence the need for approximations.

Approximation methods range from simple linear functions to large artificial neural networks (ANNs). The combination of ANN and RL resulted in deep RL (DRL). Recent advances in DRL have been able to solve many challenging problems. One of the most recent advances was deep Q-network (DQN), which achieved human-level performance on multiple Atari games [3, 4]. This was followed by many achievements, such as DRL surpassing top-ranked human players in the strategy game GO [5]. Problems based on games are considered small compared to real-world applications that are large, complex, and contain sophisticated decision-making. Such problems require additional approximation for the action spaces. These methods are referred to as policy gradient methods. The combination of state and action space approximations resulted in a class of DRL named actor-critic algorithms. In the actor-critic architecture, the actor behaves according to its decision policy while the critic provides feedback to the

agent. DRL's main advantage lies in learning a policy for a given task that can generalize over new data instead of reoptimizing a model repeatedly. The resulting policy is then used to derive the decisions in real-time by executing a forward pass on the actor's ANN, i.e. the policy. Arriving at such a policy would significantly improve the decision-making process in our problem of natural gas storage valuation. For example, an energy merchant company would use such a policy to evaluate multiple price forecasts produced by Monte Carlo simulations before acquiring natural gas storage space. Another example would be that after acquiring the storage space, a merchant operator might use this policy as a baseline to evaluate trading strategies.

This thesis aims to design an MDP environment that resembles a natural gas storage facility. Then, we will solve the MDP by utilizing a new and promising algorithm called soft actor-critic (SAC) based on the actor-critic architecture. We will show how SAC can solve the problem of natural gas storage valuation by learning an operating policy. Our computational results indicate that SAC successfully learned a policy during training, outperforming other state-of-the-art DRL algorithms. However, there is still much room for improvement regarding generalization.

The remainder of this thesis is organized as follows: Chapter 2 will discuss the storage problem and cover the necessary background for DRL, as well as a literature review on the theory and DRL applications in energy storage. Chapter 3 discusses our experiment design, implementation, and reported results. Finally, we have the conclusion in Chapter 4.

2. BACKGROUND AND LITERATURE REVIEW

2.1. Economics of Energy Commodities Storage

In the energy industry, storage facilities support trading activities [6]. Manufacturers can store input energy commodities to be readily available for consumption. Other industries, such as refining, require complex storage facilities, including input, output, and intermediate products. Energy commodity traders either own or lease the capacity of storage facilities. In the face of fluctuating energy commodity prices, competitively managing the storage capacity is referred to in the literature as merchant operations [1].

With the declining profitability of arbitrage-based trading policies [7, 8], energy merchant companies are more inclined to own and operate energy assets, especially storage facilities [6]. In recent years, many market observers have noted a noticeable trend where energy merchant companies heavily invest in storage assets [9–11]. Solid theoretical [12] and empirical [13] economic arguments support the trend of storage facility acquisition by energy merchant companies. By owning a storage facility, an energy merchant company reduces transaction costs and avoids the pitfalls of the hold-up problem [14].

The hold-up problem arises in situations [15] where (1) there is a contractual relationship between an upstream and a downstream party. (2) The downstream party invests in the upstream party's contractual commitments. The contract is incomplete, i.e., it does not cover all future expected situations. (3) The upstream party has an economic incentive to exploit ambiguities in the contract, hence the name "hold-up." In the case of natural gas storage, the upstream party is the storage facility owner, and the downstream party is the merchant company that stores natural gas in that facility. As demand for natural gas increases, the merchant company wants to seize this opportunity to sell natural gas by withdrawing it from storage. The facility

owner is incentivized to ask for additional operational fees not stipulated in the original contractual agreement. Given that the contract is incomplete, the merchant company is in a situation where it either accepts the owner's demands and incurs higher transaction costs or takes the case to court. The latter choice is not optimal. Therefore, the energy merchant is forced to accept the increase in transaction costs [6].

Economists have noted that a natural response by the downstream party facing the risk of hold-up is to pursue vertical integration [16,17]. This is consistent with what market observers have noted regarding energy merchant companies' trend of storage facility acquisition. However, vertical integration theories also encompass other energy conversion assets, such as production, transport, and processing facilities. By acquiring storage facilities, energy merchant companies would mitigate transaction cost risks and gain access to a wider range of optionalities, such as adjustments in operating levels in response to market prices [13]. This operational flexibility would, in turn, enable improved trading policy execution and higher profit margins.

In the context of merchant operations, the goal is to determine a policy that maximizes the value of the storage assets by trading an energy commodity, given its operational space and flow limits, as well as market prices. Storage assets' importance stems from their influence on the energy value chain by linking decisions across multiple periods. For example, many energy conversion assets utilize storage facilities to alleviate demand and supply disruptions. This inter-temporal linkage leads to intractable policies for complex operational decisions [1].

2.2. The Storage Problem

The storage problem, also known as the warehouse problem, was described by [18] as follows "given a warehouse with fixed capacity and an initial stock of a certain product, which is subject to known seasonal price and cost variations, what is the optimal pattern of purchasing (or production), storage and sales?". Our treatment of the storage problem differs from the definition of a warehouse problem in two major ways.

First, the capacity property mentioned in the warehouse problem description refers to the warehouse's inventory space, not the flow of products. The warehouse problem assumes no restrictions on the flow of products. In practice, however, storage assets have limited flow capacities, hence the term capacitated storage. Second, energy commodity merchants operate in wholesale markets where the purchase and sale prices are the same, i.e., spot prices. The warehouse problem assumes that the warehouse operator has a price for procuring (or producing) the product and another price for selling the product. Natural gas storage assets have two operational features: inventory space and flow capacity limits. The storage asset's space refers to the maximum inventory levels, while the flow capacity characterizes restrictions on natural gas injection and withdrawal amounts. We will initially examine a model based on the spot price and then expand it to include the futures price curve. These formulations are based primarily on the work of [1].

We begin with a simplified Markov Decision Process (MDP) formulation with two state variables: the inventory level and the spot price. The inventory level is affected by the actions taken by the energy merchant, while the spot price evolves exogenously. We assume that the energy merchant is a price taker. Hence, the energy merchant's actions do not influence the spot price. In addition to space constraints, the capacitated storage facility exhibits capacity constraints on inflows and outflows. We can relax the capacity constraints for our storage facility. This type of facility is called the fast asset, while the capacitated facility is called the slow asset.

There are three broad actions that can be taken in such MDP. In each stage, the merchant can either buy and inject (BI), withdraw and sell (WS), or do nothing (DN). An optimal operating policy for the fast asset would resemble a basestock-level policy. In such a policy, the optimal decision for an energy merchant at each stage of the MDP would be either BI to fill the storage, WS to empty the storage, or DN to do nothing. Note that in the fast asset case, the optimal decision is independent of the inventory level since there is no restriction on the storage flow capacity. On the other hand, in the slow asset case, the optimal policy is affected by the current inventory level at each

stage of the MDP. The optimal policy for the slow asset is called a basestock target policy. This policy aims to adjust the inventory level as closely as possible toward BI or WS target inventory levels. Refer to [1] for detailed proof of the optimality of the basestock level and target policies.

The basestock policies we described earlier are similar to service level basestock found in the inventory management literature [19]. The difference, however, is that the latter aims to maintain appropriate inventory given uncertain demand, while the former aims to maximize profit by optimizing trading decisions given stochastic price signals. Contrary to inventory management, our treatment of the storage problem does not include a demand component. Our interest is in optimizing the decisions of an energy merchant that operates in the wholesale market.

At each time t, the energy merchant makes a decision a_t to trade. A positive decision corresponds to BI, a negative decision corresponds to WS, and DN is zero. Each action a_t is executed during the period $[t, t+a)$. Hence, the change in inventory level is reflected at $t+1$. Each decision a_t results in a reward realized immediately at time t. The inventory space has limits on maximum and minimum storage levels denoted $\underline{x}$ and $\overline{x}$ respectively, where $\underline{x}$ and $\overline{x} \in \mathbb{R}^+$ and $\underline{x} < \overline{x}$. We then define a feasible inventory set $\mathcal{X}$ as

$$\mathcal{X} = [\underline{x}, \overline{x}]. \tag{2.1}$$

The flow capacity constraints on injections and withdrawals are the constants $C^I > 0$ and $C^W < 0$, respectively. We can now define injections and withdrawals capacity functions as

$$\overline{a}(x) = \min(C^I, \overline{x} - x) \tag{2.2}$$

$$\underline{a}(x) = \max(C^W, \underline{x} - x). \tag{2.3}$$

This function characterizes the maximum amount of natural gas a merchant can inject and withdraw at any arbitrary stage t given the capacity constraints C^I and C^W, the current inventory level x, and the feasible inventory set $\mathcal{X}$. The reward is given by a

function that depends on the quantity traded a and spot price s, both at time t. In practice, additional costs are associated with injection and withdrawal operations: an in-kind fuel cost for injections and withdrawals, ϕ^I and ϕ^W, respectively. Furthermore, there are constant marginal injection and withdrawal costs c^I and c^W. The adjusted injection and withdrawal spot prices become s^I, defined as $s^I = \phi^I s + c^I$, and s^W, defined as $s^W = \phi^W s - c^W$. We define the reward function in terms of the adjusted spot prices as follows

$$r(a,s) = \begin{cases} -s^W a, & \text{if } a \in \mathbb{R}^- \\ 0, & \text{if } a = 0 \\ -s^I a, & \text{if } a \in \mathbb{R}^+ \end{cases}, \tag{2.4}$$

the MDP formulation allows for a Bellman optimality structure

$$v(x_t, a_t, s_t) = r(a_t, s_t) + \delta \mathbb{E}\left[\max v(x_{t+1}, a_{t+1}, \tilde{s}_{t+1}) \mid s_t\right], \tag{2.5}$$

where the value function $v(x, a, s)$ is solved by applying the max operator recursively. The expectation is taken over the spot price evolution, $\tilde{s}_{t+1}$, conditioned on the current spot price s_t. We can model the evolution of spot prices using any of the option pricing methods in the finance literature. A popular method used to model natural gas spot price evolution was proposed by [20], where they used a one-factor model in which the current spot price s_t is a sufficient statistic to model the evolution of the spot price $\tilde{s}_{t+1}$. A computational analysis of this MDP formulation was done by [21]. Discretizing the MDP action space aided in solving the problem using a dynamic programming backward recursion. Discretization was necessary because the max operator is computationally intractable for large action spaces. A workaround is to reduce the feasible inventory set $\mathcal{X}$ into $1 + \overline{x}/Q$ target inventory levels, where Q is the largest common factor of flow capacities and maximum inventory level, C^I, C^W and $\overline{x}$. The results showed that in the slow asset case, in addition to the spot price, the inventory level significantly impacts the optimal policy, i.e., the policy has a basestock target structure in which, at each stage of the MDP, it specifies two critical inventory level targets for BI and WS that depend on the price and inventory level information.

The spot price one-factor model is seldom used in practice. Energy merchants prefer models that utilize the complete pricing information provided by the market [22–24]. This means that the market's daily spot and futures prices are incorporated into the model when valuing natural gas storage. Solving such an MDP for an exact solution is not feasible. Alternatively, energy merchants typically employ heuristic-based methods and settle for suboptimal approximate yet tractable policies. For brevity, we will restrict our discussion to heuristic methods. We extend our previous MDP formulation by including the futures price curve. We use $\boldsymbol{F}_t$ to denote the vector of the futures price curve observed at time t. Thus, we can replace the one-factor spot price s_t in the expectation term with $\boldsymbol{F}_t$ giving us

$$v(x_t, \boldsymbol{F}_t) = \max_a r(a, s_t) + \delta \mathbb{E}\left[v(a + x_t, \boldsymbol{F}_{t+1}) \mid \boldsymbol{F}_t\right], \tag{2.6}$$

where $\boldsymbol{F}_{t+1}$ is a vector of the expected next period futures price curve conditioned on the current futures price curve $\boldsymbol{F}_t$.

Many heuristic solutions are proposed in the literature [25]. Practitioners typically employ a heuristic based on a Linear Program (LP) [23]. One such method embeds a portfolio of spread options into an LP. A spread option enables the energy merchant to trade on the difference in a commodity's futures prices. The spread option LP solves the problem by deciding on injection and withdrawal quantities given the operational constraints and spread option prices of natural gas. Another LP-based heuristic uses the current market futures curve to solve the problem. This method differs from the spread option LP by solving the problem based on the seasonality of natural gas prices, i.e., the intrinsic value of storage. Both methods are implemented within a Monte Carlo simulation for natural futures prices.

The performance of these LP-based heuristic methods was analyzed by [26]. They first developed an Approximate Dynamic Programming (ADP) heuristic that extended the spot price model proposed by [21] to include the futures curve. To solve the problem using the same dynamic programming backward recursion method, they initially reduced the dimensionality of the MDP by approximating the futures price curve and the value function. The futures curve was redefined such that $\tilde{\boldsymbol{F}}_t$ is the observed futures

curve at time t, excluding the spot price s_t and prompt month futures price $\boldsymbol{F}_{(t,t+1)}$. This results in the following formulation

$$v(x_t, \boldsymbol{F}_t) = \max_a r(a, s_t) + \delta \mathbb{E}\left[v(a + x_t, s_{t+1}, \tilde{\boldsymbol{F}}_t) \mid \boldsymbol{F}_{(t,t+1)}\right], \tag{2.7}$$

where the next period spot price, s_{t+1}, is conditioned on the prompt month futures price $\boldsymbol{F}_{(t,t+1)}$. This simplified the MDP as the observed futures curve for the remaining months, $\tilde{\boldsymbol{F}}_t$, became redundant and eventually omitted as the prompt month futures price $\boldsymbol{F}_{(t,t+1)}$ is a sufficient statistic for the next period spot price s_{t+1}. Note that applying the max operator within the expectation term is computationally intractable. In addition to inventory level discretization, the expectation was estimated using a Monte Carlo simulation of the futures price curve to evaluate ADP policies. This model's policy has the same basestock target properties as in [21], which we discussed earlier.

The analysis done by [26] used this ADP method to benchmark the performance of the LP-based heuristics used in practice. Their results show that while the LP-based heuristic methods are computationally fast, their performance was inferior to the ADP method. Having said that, they noted that periodically reoptimizing the LP-based heuristics at each stage of the MDP improved its performance significantly, albeit at a higher computational cost. Another observation is that the ADP policy was sensitive to changes in injection and withdrawal capacities. Furthermore, the performance of the ADP policy was affected by increasing the resolution of MDP discretization for its state space variables x_t and $\boldsymbol{F}_t$. Another example of an ADP heuristic energy merchants employ for natural gas storage optimization is the Least-Squares Monte Carlo (LSMC) method [27–29]. LSMC approximates the futures curve using a linear basis function and then applies least-squares regression to calculate the ADP policy [30].

Even though ADP heuristic methods can incorporate the futures curve into the MDP, their use has many limitations compared to DRL on larger problems in practice. For example, the ADP approach to approximations, such as discretizing the MDP state space and simplifying the decisions, limits its application to a small set of problems. The ADP methods we discussed deliberately lower the resolution of the MDP's state

and action spaces to facilitate the computation of the policy using the max operator. The ADP methods discussed in this section solve MDPs with a single action variable. Hence, the max operator in the ADP methods we discussed is a greedy action selection rule for a specific MDP instance, i.e., a deterministic policy for the Monte Carlo simulation of the problem. In contrast, DRL provides the energy merchant with a generalized policy that can be directly used on any instance of a similar MDP by executing a forward pass on its ANN. Moreover, advanced DRL methods can handle high-dimensional action spaces and are capable of learning deterministic and stochastic policies. Hence, ADP approximation unable to generalize to new state observations, which requires a complete reoptimization of the model whenever new data is presented.

Moreover, state-of-the-art ADP methods such as LSMC require problem-domain expertise to design the basis function approximations. DRL's use of ANN when approximating the MDP avoids the issue of manually designing state features and enables it to generalize from a smaller training dataset into new states. Another limitation of LSMC is that it is more suitable for approximating large exogenous state variables while the endogenous state variables are kept simple. The basis function used by [29] approximates the exogenous state variable, i.e., the futures curve, while the endogenous state variable is a scalar indicating the inventory level. A complicated endogenous state variable would pose a challenge to LSMC. This differentiation between endogenous and exogenous state variables is not found in DRL, as all state variables are inputs to the ANN. Furthermore, correlations in state variables affect the performance of LSMC since the basis function approximations use linear regression. In DRL, there are many techniques that aid ANNs in overcoming the problem of input correlation. Another issue is that the ADP heuristic methods require a model of the MDP state space, i.e., a probabilistic state transition model for the spot and futures prices to be used by the Monte Carlo simulator. The issue of model error has detrimental effects on the ADP policy [31]. Model-free DRL avoids this pitfall by solving the MDP without assuming any probability distribution of the state variables.

These issues point to the limitations of using ADP to solve large MDP problems. Larger problems of practical interest involve complex MDPs and require advanced algorithms. For example, an energy merchant needs to coordinate the operating policies of multiple assets, which would entail many state and action variables. Such a complex policy with multiple decisions is not possible using ADP. This leads us to the next section, where we examine the theory of DRL and discuss recent attempts to solve the energy storage problem using state-of-the-art DRL algorithms.

2.3. Reinforcement Learning

RL attempts to solve the problem of an agent's sequential interactions with its environment. This interaction is defined using the MDP formulation. In an MDP, an agent observes the state of the environment and takes an action, S_t and A_t, respectively. In a later time-step, $t+1$, the agent receives a reward and new state observation from the environment, R_{t+1} and S_{t+1}. Given this simple depiction of the agent-environment interaction, the problem becomes how an agent learns to choose the action A for each observed state S to maximize the cumulative rewards R received at each time step. The following formulations are based on [2].

2.3.1. Markov Decision Process

The MDP formulation captures the essential features of the RL problem, namely the states of the environment, actions taken by the agent, and the rewards received by the agent (S, A, R). This abstract formulation focuses on the essential components of the repeated agent-environment interaction [32]. This interaction takes the following form, $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$ where the agent observes the state of the environment S_t, takes action A_t, then the environment returns the following state observation and a reward S_{t+1}, R_{t+1}. Another essential feature of the MDP formulation is the Markov property. This property states that the transition probability of the next reward and the state of the environment, (S_{t+1}, R_{t+1}), depends only on the previous state-action pair (S_t, A_t).

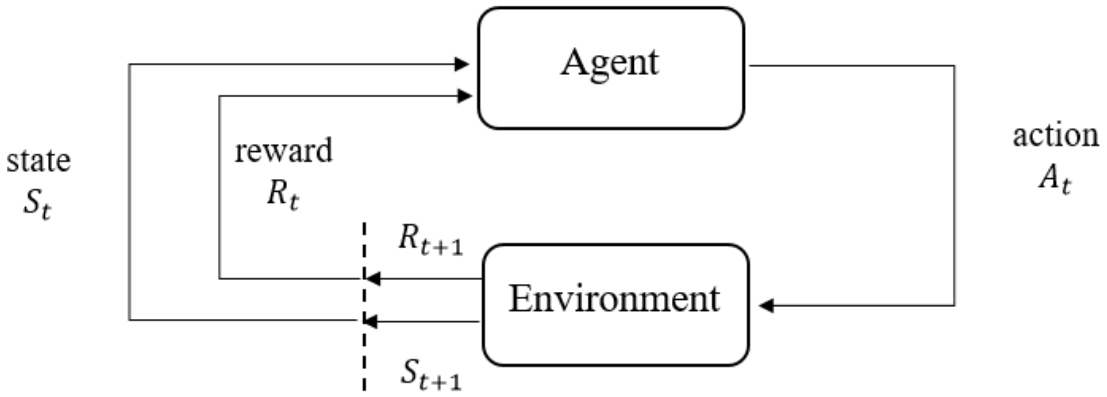

Figure 2.1. Agent environment interaction.

The Markov property is formally defined as

$$p(S_{t+1}, R_{t+1} \mid S_t, A_t) = p(S_{t+1}, R_{t+1} \mid S_t, A_t, \ldots, S_0, A_0). \tag{2.8}$$

Hence, knowing the current state and action must be sufficient to know the probability of the following state and reward for the Markov property. Then, the MDP transition probability is defined as follows

$$p(s', r \mid s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\} \tag{2.9}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.10}$$

Such that s', r, s, and a are instances of the random variables S_{t+1}, R_{t+1}, S_t, and A_t, taken from the sets $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{A}$, and the dynamics of the MDP are fully defined by the function p. The accumulation of rewards an agent receives from the environment is called the return

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{2.11}$$

where t is the current step, and T is the final step in the agent-environment interaction. Additionally, the return can be discounted using a constant factor γ, thus giving preference to immediate rewards

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.12}$$

RL solution methods primarily depend on what is known as the value functions. These functions are used to estimate the expected downstream cumulative rewards, i.e., the

return G_t, starting from the current state S_t and following a policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \quad \text{for all } s \in \mathcal{S}, \tag{2.13}$$

where $v_\pi(s)$ is subscripted with the policy π. The policy π maps an observed state s onto an action a, such that $\pi(s) = a$. A policy π can be deterministic, $\pi(s)$, or stochastic where $\pi(a \mid s)$ gives the probability of choosing an action given a state. Therefore, the policy π defines the agent's behavior while interacting with the environment. In addition to the value function, RL uses the action-value function,

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]. \tag{2.14}$$

The main difference between value functions and action-value functions is that the action a is taken at time t, and then afterward, the policy π is followed for subsequent actions. In its current form, it is unclear how we can estimate the value function in Equation 2.13. We can rewrite it in recursive form

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && (2.15)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && (2.16)\\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] && (2.17)\\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S}. && (2.18)
\end{aligned}$$

Similarly, we can rewrite equation (2.14) in recursive form too

$$q_\pi(s, a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')], \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{2.19}$$

These equations define the functions $v_\pi(s)$ and $q_\pi(s, a)$, as a recursive expectation over the MDP's state and action spaces. These recursive formulations are known as the Bellman equations [33]. For a given MDP and any arbitrary policy π, these equations solve the problem of calculating v_π and q_π. Given the structure of this formulation, the Bellman equations can be solved as a system of linear equations. However, the computational cost increases as the MDP's state and action spaces become large.

The most important goal in RL is to efficiently estimate the functions v_π and q_π. Furthermore, knowing the values of the functions v_π or q_π is insufficient to solve an MDP. Solving an MDP entails finding an optimal/near-optimal policy π that achieves the highest cumulative rewards. We say a policy π_* is optimal if and only if $v_{\pi_*}(s) \geq v_\pi(s)$ for all $s \in S$. An MDP must have at least one optimal policy π_* [33]. Hence, the optimal value function is defined as follows

$$v_*(s) = \max_\pi v_\pi(s), \quad \text{for all } s \in \mathcal{S}. \tag{2.20}$$

Similarly, the optimal action-value function is defined as follows

$$q_*(s,a) = \max_\pi q_\pi(s,a), \quad \text{for all } s \in \mathcal{S} \text{ and for all } a \in \mathcal{A}. \tag{2.21}$$

Note that the optimal action-value function can be defined in terms of the optimal value function

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \tag{2.22}$$

We may rewrite the optimal value function without referencing any specific policy as in Equations 2.20 and 2.21 as follows

$$\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) && (2.23)\\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] && (2.24)\\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && (2.25)\\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] && (2.26)\\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_*(s')\right]. && (2.27)
\end{aligned}$$

Similarly, we do the same for the optimal action-value function

$$\begin{aligned}
q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] && (2.28)\\
&= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]. && (2.29)
\end{aligned}$$

We refer to Equations 2.27 and 2.29 as the Bellman optimality equations [33]. We can solve the MDP by adopting a greedy policy if we know the values of the Bellman optimality equations v_* and q_*. The max operator in both v_* and q_* represents this

greedy policy. However, we must first evaluate v_* and q_*. Unfortunately, the max operator adds nonlinearity to our Bellman equations. Hence, we cannot solve them as a system of linear equations. This is true even for a small MDP, which takes us to the next section, where we examine iterative algorithms to solve the MDP by finding the optimal policy and value functions.

2.3.2. Dynamic Programming

Dynamic programming (DP) refers to algorithmic methods for solving an MDP, i.e., finding the optimal policy π_*. In DP, we assume that we have access to the transition probability function $p(S_{t+1}, R_{t+1} \mid S_t, A_t)$ that fully defines the dynamics of the MDP. We first discuss how DP can efficiently estimate the value function v_π of an MDP for a given arbitrary policy π. To achieve this, DP utilizes the Bellman equation by substituting the equality with an assignment operator. In other terms, the Bellman equation becomes an update rule. Recall the Bellman equation in Equation 2.18, where

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]. \tag{2.30}$$

We modify this equation by adding an index k and holding the policy π constant. We then read the equality ($=$) as an assignment operator

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')], \tag{2.31}$$

this is referred to as an iterative update. In this formulation, we use the notation v_{k+1} for our estimate of v_π at iteration $k + 1$. Just as in the Bellman equation, the estimate v_{k+1} is an expectation applied recursively over all v_k. In DP terminology, this is called a sweep. Another aspect is that in this formulation, we solve for each value function $v_{k+1}(s)$ in an iterative manner, by sweeping over the state space $\mathcal{S}$, as opposed to solving a system of linear equations simultaneously, $|\mathcal{S}|$ equations in $|\mathcal{S}|$ unknowns. The iterative policy evaluation algorithm for estimating v_π is in Figure 2.2. In this algorithm, our estimate of v_π is initialized arbitrarily. Then, iteratively, our estimates become more accurate at each sweep, $v_0, v_1, v_2, \ldots, \approx v_\pi$.

Input π, the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining the accuracy of estimation

Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Figure 2.2. Policy evaluation.

Now that we have estimated v_π, we proceed with an algorithm to improve π. Recall that the value function v_π is an expectation of the downstream cumulative rewards of starting in a state s and following the policy π. However, the policy π may not be optimal. Hence, in any state s, an action a might result in higher cumulative rewards rather than the action given by the policy $\pi(s)$. This is an instance of an action-value function, where at a state s, we might choose an action a not based on policy π, recall the action-value function in Equation 2.19 where

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]. \tag{2.32}$$

Furthermore, given how v_π and q_π account for the cumulative downstream rewards, choosing an action a in a greedy manner in any state s is an optimal action as follows

$$\begin{aligned} \pi'(s) &= \arg\max_a q_\pi(s, a) && (2.33)\\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] && (2.34)\\ &= \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]. && (2.35) \end{aligned}$$

Note that π' is assigned an action a using the arg max operator applied to the action-value function $q_\pi(s, a)$. Hence, π' is a greedy policy with respect to $q_\pi(s, a)$ for any given state s.

We refer to this method as policy improvement, where this inequality holds

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{2.36}$$

Then it follows that a greedy policy π', obtained by policy improvement, with respect to v_π yields the Bellman optimality equation

$$v_{\pi'}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\pi'}(s')]. \tag{2.37}$$

Therefore, from Equation 2.27, v_* must be equal to $v_{\pi'}$, also π_* and π' must be both optimal policies. These results give us the essential tools to solve an MDP. We now need an algorithm that formalizes the steps of estimating an optimal policy π_*. An algorithmic method used in DP to estimate π_* is referred to as policy iteration. In policy iteration, we alternate between policy evaluation and policy improvement until we arrive at our estimate of π_* given the following steps, $\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \ldots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*}$. Where $\xrightarrow{\text{E}}$ and $\xrightarrow{\text{I}}$ indicate policy evaluation and policy improvement, respectively. We begin with arbitrary initializations for v_π and π. After estimating the value function v_{π_0} using our initial policy π_0, we then update the policy π_1 greedily using our value function estimates v_{π_0}. Note that after updating the policy from π_0 to π_1, our value function estimates v_{π_0} need to be reevaluated into v_{π_1}. The latter step may result again in an update on the policy. Alternating between policy evaluation and improvement is called the policy iteration algorithm.

We can improve policy iteration by reducing the computational cost of sweeping the state space in the policy evaluation step. This is achieved by combining policy evaluation and policy improvement into a single step by updating v_{k+1} using the Bellman optimality equation to estimate directly v_π

$$v_{k+1}(s) = \max_a \left[\sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right] \right]. \tag{2.38}$$

The convergence of both the policy and value iteration algorithms is determined by monitoring the changes in the value function v_π. During each sweep, when the maximum change in v_π is small enough, specifically $\Delta < \theta$, this indicates that the algorithm is approaching convergence.

1. Initialization

   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$, and $V(terminal) = 0$

2. Policy Evaluation

   Loop:

   $\Delta \leftarrow 0$

   Loop for each $s \in \mathcal{S}$:

   $v \leftarrow V(s)$

   $$V(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$

3. Policy Improvement

   policy-stable $\leftarrow$ *true*

   For each $s \in \mathcal{S}$:

   old-action $\leftarrow \pi(s)$

   $$\pi(s) \leftarrow \arg\max_{a} \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$$

   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 2.3. Policy iteration.

Algorithm parameter: a small threshold $\theta > 0$ determining the accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}$ arbitrarily except that $V(terminal) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$$V(s) \leftarrow \max_{a} \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$$

Figure 2.4. Value iteration.

While the DP methods we presented are efficient in solving small MDPs, these methods are seldom used in practice. This is due to the so-called curse of dimensionality, as policy iteration and value iteration algorithms become impractical when an MDP's state and action spaces grow too large. Furthermore, DP methods require complete knowledge of the MDP's dynamics, more specifically, the probability transition function $p(s', r \mid s, a)$. In practical problems, however, this is rarely the case. This leads us to another class of MDP solution methods that are less susceptible to the curse of dimensionality and do not require knowledge about the MDP's dynamics.

2.3.3. Temporal Difference

Temporal difference (TD) is one of RL's most important and central ideas [2]. TD offers a solution method that does not require a perfect model of an MDP's dynamics. This is done by estimating the value function v_π by interacting with the environment and updating v_π estimates in an online and incremental fashion. At each time step, $t+1$, the agent performs an update on the previous time step estimated value function $V(S_t)$ using the difference between the observed reward R_{t+1} and estimated value function $V(S_{t+1})$ as follows

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{2.39}$$

where $R_{t+1} + \gamma V(S_{t+1})$ is the target, $V(S_t)$ is our old estimate, and α is our step-size parameter. Using our estimate $V(S_{t+1})$ in updating the value function $V(S_t)$ is called bootstrapping. The idea of bootstrapping was used in DP when the sweep was done over the state space as in Equation 2.31. Bootstrapping significantly improves the convergence rate and has been shown to converge with the probability of 1 [34, 35]. The difference between the target and the old estimate is called the TD error,

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{2.40}$$

In Figure 2.5, we show a TD algorithm to estimate v_π. Note how the TD algorithm estimates v_π without the need for the transition function $p(s', r|s, a)$ that defines the MDP dynamics.

The main difference between TD and DP updates is that in DP, the update was an expectation over the value functions of all states. While in TD, we do not take expectations since we cannot access the MDP dynamics model; we perform a sample update instead. This means that with each agent interaction with the environment, we immediately update our estimate $V(S_t)$ as shown in Equation 2.39. We further extend our TD method to utilize our value function estimates to improve the policy. The first algorithm we discuss is SARSA [36, 37],

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{2.41}$$

The name SARSA is derived from the abbreviation (State, Action, Reward, State, Action) given by the sequence $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. In SARSA, we modify the update rule of TD in Equation 2.39 by substituting the estimated value function $V(S_t)$ with the estimated action-value function $Q(S_t, A_t)$. The ϵ-greedy policy is a policy that gives preference to optimal actions in each state, but with a probability ϵ it behaves randomly. This randomness ensures that the agent sufficiently explores the MDP's action space.

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0,1]$
Initialize $V(s)$, for all $s \in \mathcal{S}$ arbitrarily except that $V(terminal) = 0$
    Loop:
        Initialize S
        Loop for each step:
            $A \leftarrow$ action given by π for S
            Take action A, observe R, S'
            $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
            $S \leftarrow S'$
        Until S is terminal

Figure 2.5. Temporal difference.

SARSA is an on-policy control algorithm. In an on-policy control algorithm, the agent's behavior and the target policies, $b(s)$ and $\pi(s)$ respectively, are identical. The

behavior policy, by its name, is the policy that specifies which action an agent takes. The target policy is the policy used for estimating the value and action value functions. Hence, in SARSA, ϵ-greedy is both a behavior and target policy.

Algorithm parameter: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ arbitrarily except that $Q(terminal,.) = 0$
Loop:
Initialize S
Choose A from S using policy derived from Q (e.g., ε-greedy)
Loop for each step:
Take action A, observe R, S'
Choose A' from S' using policy derived from Q (e.g., ε-greedy)
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
$S \leftarrow S'; A \leftarrow A';$
Until S is terminal

Figure 2.6. SARSA.

Alternatively, we can use an off-policy TD control method in which the behavior and target policies differ. We begin with Q-learning [38],

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{2.42}$$

Note the target in Q-learning, $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$, and the Bellman optimality equation for q_* in Equation 2.29. The Q-learning algorithm allows the agent to follow a behavior policy for action A_t, $b(S_t, A_t)$, thus driving the updating process of $Q(S_t, A_t)$. The operator $\max_a$ represents the greedy target policy, which simultaneously approximates q_* as shown in the target. Expected SARSA [39] is an improvement over Q-learning,

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{2.43}$$

In Expected SARSA, the target is an expectation of the next state-action pair. Since the target now is an expectation, this results in additional computational costs compared to SARSA and Q-learning.

Algorithm parameter: step size $\alpha \in (0,1]$, small $\varepsilon > 0$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ arbitrarily except that $Q(terminal,.) = 0$

Loop:

Initialize S

Loop for each step:

Choose A from S using policy derived from Q (e.g., ε-greedy)

Take action A, observe R, S'

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \max_a Q(S',a) - Q(S,A)\right]$$

$S \leftarrow S'; A \leftarrow A';$

Until S is terminal

Figure 2.7. Q-learning.

Note that the policy used in the target need not be the same as the behavior policy. For example, if the policy used by the target was greedy, $\max_a$, then Expected SARSA becomes Q-learning. Thus, Expected SARSA is a generalization of Q-learning.

2.3.4. Approximation Methods

In our previous discussion of MDP solution methods, we assumed that our state and action spaces were reasonably sized and discrete. In practice, however, the state and action spaces are large and not necessarily discrete. Furthermore, given a limited agent-environment interaction, we would like our MDP solution methods to generalize to new unobserved states. This leads us to approximation methods.

The central idea in approximation methods is to employ a parameterized function, either linear or non-linear, for our value function estimate. Hence, we approximate the value function $v_\pi(s)$ with the parameterized value function $\hat{v}(s, \mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^d$ is a parameter weight vector with dimension d, such that $d \ll |\mathcal{S}|$. The function $\hat{v}$ can be a linear function where $\mathbf{w}$ represents the weights of the features, or it can be a large neural network with $\mathbf{w}$ representing the weights of the network connections.

In previous value function estimations using DP and TD, an update to a value function estimate $v(s)$ doesn't affect other value function estimates. We made an implicit assumption of having the value and action-value functions stored in tabular format.

Table 2.1. Example of tabular representation.

State s	$\boldsymbol{v_\pi(s)}$
s_0	101.5
s_1	203.7
⋮	⋮

Replacing the tabular representation with an approximation would make our value function estimate less accurate. However, to arrive at an optimal policy, we do not need to estimate the value function $v(s)$ precisely [40, 41]. Another feature of value function approximation is that a change in the weight vector $\mathbf{w}$ to improve an estimate $\hat{v}(s, \mathbf{w})$ would affect other value function estimates as well. This is a beneficial aspect of value function approximation, as it achieves the goal of generalization. A drawback to the generalization feature is that improving some value function estimates would lower the accuracy of other value function estimates. This requires introducing a performance objective to our value function approximation.

We define the difference between the value function $v_\pi(s)$ and our approximation $\hat{v}(s, \mathbf{w})$ as the mean squared value error

$$\overline{VE}(\mathbf{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2 , \tag{2.44}$$

where $\mu(s)$ is the distribution of state s. For a given MDP, $\mu(s)$ is interpreted as the fraction of time spent at state s during an experience. This will give more preference when adjusting the weight vector $\mathbf{w}$ towards states that matter the most. Our objective is to find an optimal weight vector $\mathbf{w}^*$ such that $\overline{VE}(\mathbf{w}^*) \leq \overline{VE}(\mathbf{w})$ for all $\mathbf{w}$. Attaining this objective depends on our value function approximation method. A linear approximation could yield a global optimum, while we should settle for a local

optimum when using a non-linear approximation. Nevertheless, the appropriate choice of value function approximation method and its convergence guarantees is still an open research question [2].

Now that we have a performance objective, $\overline{VE}(\mathbf{w})$, we need to define the method by which we improve our value function approximation $\hat{v}(s, \mathbf{w})$ and minimize $\overline{VE}(\mathbf{w})$ as much as possible. A widely used method is Stochastic Gradient Descent (SGD). In SGD, we improve our value function approximation $\hat{v}(s, \mathbf{w})$ with each agent-environment interaction in an incremental on-line fashion. Assuming that we observe the MDP states according to the distribution μ, this simplifies our performance objective $\overline{VE}(\mathbf{w})$, since we will not need to explicitly calculate $\mu(s)$ for each state. Furthermore, the value function approximation has to be a differentiable function with respect to $\mathbf{w}$. Hence, with each observation generated by the agent-environment interaction, we incrementally adjust our weight vector $\mathbf{w}$ as follows

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t), \tag{2.45}$$

where $\nabla \hat{v}(S_t, \mathbf{w}_t)$ is the gradient, i.e., column vector of partial derivatives of $\hat{v}(S_t, \mathbf{w}_t)$, and a is the step size parameter. Note that $v_\pi(S_t)$ is not known; therefore, we substitute it with an unbiased estimate U_t

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t), \tag{2.46}$$

where we define U_t as

$$U_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t). \tag{2.47}$$

Note that the gradient of $[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)]$ is complicated. In SGD, we assume that the bootstrapped target U_t is independent of the weight vector $\mathbf{w}$. A simple workaround is to take the gradient of our estimate $\hat{v}(S_t, \mathbf{w}_t)$ and omitting it for the target $\hat{v}(S_{t+1}, \mathbf{w}_t)$. This is referred to as semi-gradient methods [42]. Below is the Pseudocode of semi-gradient TD [35, 43]. A simple value function approximation method is a linear function. We define a state feature vector as $\mathbf{x}(s) = (x_1(s), x_2(s), \ldots, x_d(s))$ with dimension d similar to our weight vector $\mathbf{w}$. The linear function approximation

is then the inner product of two vectors

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s), \tag{2.48}$$

taking the gradient with this formulation is straightforward

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s), \tag{2.49}$$

then our SGD update becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \mathbf{x}(S_t). \tag{2.50}$$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v}: \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Loop:
Initialize S
Loop for each step:
Choose $A \sim \pi(.|S)$
Take action A, observe R, S'
$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$
$S \leftarrow S'$;
Until S is terminal

Figure 2.8. Semi-gradient TD.

Artificial Neural Networks (ANNs) have gained much attention and popularity for non-linear function approximations. Many of the advances in supervised learning, a subfield of machine learning, are due to ANNs' ability to learn from labeled data and generalize to new observations. The generalization capacity of ANNs is essential for our purposes. ANNs update their connection weights $\mathbf{w}$ using the same SGD scheme discussed earlier. With each observation, the ANN weights $\mathbf{w}$ are adjusted to improve the performance objective $\overline{VE}(\mathbf{w})$. The most popular algorithm to adjust the ANN weights $\mathbf{w}$ is back-propagation [44].

In semi-gradient TD, we can improve the policy by replacing the value function approximation $\hat{v}(s, \mathbf{w})$ with action-value function approximation $\hat{q}(s, a, \mathbf{w})$. This gives us semi-gradient SARSA [36]:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t). \tag{2.51}$$

Similar to the value function linear approximation in Equation 2.48, the action-value linear approximation would take the following form

$$\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s, a) = \sum_{i=1}^{d} w_i x_i(s, a). \tag{2.52}$$

As in the tabular SARSA, semi-gradient SARSA is an on-policy control algorithm that utilizes an ϵ-greedy policy. Figure 2.9 shows the Pseudocode of semi-gradient SARSA.

Unlike the on-policy semi-gradient versions of TD and SARSA, combining function approximation with an off-policy algorithm, such as Q-learning, results in many problems. In TD, the policy is held static as we aim to estimate v_π, while in SARSA, both behavior and target policies are identical $b(s) = \pi(s)$. The off-policy aspect of the approximated Q-learning has been shown to diverge and cause instability [45–47].

Input: a differentiable action-value function $\hat{q}: \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize action-value function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g. $\mathbf{w} = \mathbf{0}$)
    Loop:
        $S, A \leftarrow$initial state and action (e.g. ε-greedy)
        Loop for each step:
            Take action A, observe R, S'
            If S' is terminal:
                $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$
                End
            Choose A' as a function of $\hat{q}(S', ., \mathbf{w})$ (e.g. ε-greedy)
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$
            $S \leftarrow S'$;
            $A \leftarrow A'$;

Figure 2.9. Semi-gradient SARSA.

There are three elements in an RL algorithm that, when combined, could potentially result in divergence and instability [48]. They are collectively called the deadly triad [42, 49]. First is function approximation, in which we aim to generalize over a large state space. The second is bootstrapping, where we update our TD or DP targets using previous estimates. Third is off-policy learning, in which the update targets use a policy $\pi(s)$ that differs from the agent's behavior policy $b(s)$. The literature has not yet settled on overcoming the deadly triad challenge [2]. However, recent attempts such as [3, 4] developed a deep Q-network (DQN) algorithm, which produced promising results. The DQN algorithm combined bootstrapping and off-policy learning of Q-learning and ANN for function approximation. In other words, DQN had all the elements of the deadly triad. Nonetheless, DQN achieved impressive results as shown in [3, 4].

The key to DQN's success relied on two of its novel features. First is experience replay, where DQN stored a tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ in a replay memory. Then, DQN performed SGD on a random mini-batch of samples from the replay memory. This reduced the variance and improved the sample efficiency of Q-learning. Second, to stabilize Q-learning, DQN uses two ANNs in SGD. The ANN used in the target update of Q-learning was kept constant, while the other ANN network was updated using SGD. After several SGD iterations, the updated ANN is copied into the target ANN, thus providing a relatively stable target for Q-learning.

2.3.5. Policy Gradient Methods

Previously discussed control methods, such as SARSA and Q-learning, depended on action-value function estimates, $Q(S, A)$, for optimal action selection. An alternative approach is learning a parameterized policy directly without applying value function estimation. Instead, the value function will aid in improving the parameterized policy performance. The parameterized policy takes the form $\pi(a|s, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the parameter vector with dimension d' such that $\boldsymbol{\theta} \in \mathbb{R}^{d'}$.

If the action space is discrete and not too large, we may utilize an exponential soft-max distribution for our parameterized policy

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}, \tag{2.53}$$

where the function $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$ gives a value representing the preference of choosing a state-action pair given the parameter vector $\boldsymbol{\theta}$. This formulation has two advantages over ϵ-greedy policy discussed earlier. First, the soft-max formulation can approach a deterministic policy, whereas the ϵ-greedy keeps exploring randomly with probability ϵ. Second, the soft-max policy parameterization allows action selection with arbitrary probabilities. This is useful with problems that require the parameterized policy to be stochastic, which is quite difficult to achieve using action-value methods. For example, in problems where multiple actions are optimal, a soft-max policy can attribute equal probability to each one of these actions. The function $h(\cdot)$ that determines our action preference can be an artificial neural network (ANN) or simply a linear function approximation

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a), \tag{2.54}$$

where the feature vector $\mathbf{x}(s,a) \in \mathbb{R}^{d'}$ is similar to Equation 2.48. Using a parameterized policy leads to a smoother adjustment of action probability. This smooth transition is in contrast with ϵ-greedy, which changes the action probability abruptly with a slight adjustment to the action-value estimate. This smooth transition is achieved by updating the parameter vector $\boldsymbol{\theta}$ by stochastic gradient ascent (SGA). Similarly to our treatment of SGD, we define SGA as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}. \tag{2.55}$$

We utilize the function $J(\boldsymbol{\theta})$ as our performance measure as we did in Equation 2.44, hence the gradient of the approximated performance measure $\nabla J(\boldsymbol{\theta})$ adjusts the parameter vector $\boldsymbol{\theta}$. The performance measure needs to be adequately defined in order to calculate its gradient $\nabla J(\boldsymbol{\theta})$; we initially define $J(\boldsymbol{\theta})$ as

$$J(\boldsymbol{\theta}) = v_\pi(s), \tag{2.56}$$

where $v_\pi(s)$ is the value function starting from state s and following the parameterized

policy π. It is natural to choose the value function $v_\pi(s)$ as our performance measure. We expand the right-hand side using the definitions of the Bellman equation

$$v_\pi(s) = \sum_a \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a), \tag{2.57}$$

we then substitute back the expanded term into Equation 2.56 and take the gradient, which yields the following

$$\nabla J(\boldsymbol{\theta}) = \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta}). \tag{2.58}$$

By replacing $q_\pi(s, a)$ with a parameterized approximation $\hat{q}(s, a, \boldsymbol{w})$, this renders our performance measure gradient $\nabla J(\boldsymbol{\theta})$ an approximate performance measure gradient $\widehat{\nabla J(\boldsymbol{\theta}_t)}$. We can now substitute the resulting gradient into Equation 2.55, which yields

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \mathbf{w}) \nabla \pi(a|S_t, \boldsymbol{\theta}_t). \tag{2.59}$$

Note that this formulation accounts for all actions $\sum_a \hat{q}(S_t, a, \boldsymbol{w}) \nabla \pi(a|S_t, \boldsymbol{\theta}_t)$. We need to restrict the performance measure to a specific action A_t. This leads us to the REINFORCE algorithm [50]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}, \tag{2.60}$$

this is called the REINFORCE update. For detailed derivation, refer to [2]. Recall that G_t is the cumulative downstream rewards, which is not known at time t. To overcome this, we need to resort to an approximation. This leads us to a special class of policy gradient algorithms called the actor-critic methods [51].

In the actor-critic methods, we utilize approximations for the policy and value functions. We refer to the parameterized policy $\pi(a|s, \boldsymbol{\theta})$ as the actor, while the parameterized value function $\hat{v}(s, \boldsymbol{w})$ as the critic. We modify Equation 2.60 by restricting the return G_t to a single time step. This is achieved by applying our previously discussed TD structure in Section 2.3.3 as follows

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}, \tag{2.61}$$

the expression is simplified using the definition of the TD error δ from Eq. (2.40)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \frac{\nabla\pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}. \tag{2.62}$$

Now, the intuition of our SGA formulation is straightforward. The policy parameter vector $\boldsymbol{\theta}$ is adjusted by the gradient of our performance measure $\widehat{\nabla J(\boldsymbol{\theta}_t)}$, given here by the TD error δ which is weighted by the fractional term $\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$. The fractional term accounts for the proportionality of the state-action pairs during each update of the parameter vector $\boldsymbol{\theta}$. When updating the vector $\boldsymbol{\theta}$ in SGA, the fractional term $\frac{\nabla\pi(A|S,\boldsymbol{\theta})}{\pi(A|S,\boldsymbol{\theta})}$ was replaced with $\nabla \ln \pi(A|S, \boldsymbol{\theta})$, which is an equivalent term. Additionally, the variable I, was added which acts as a compounding discount factor for the vector $\boldsymbol{\theta}$ update. The discounting factor can aid in convergence properties.

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\mathbf{w}} > 0$
Initialize policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g. to $\mathbf{0}$)
Loop:
Initialize S
$I \leftarrow 1$
Loop while S is not terminal :
$A \sim \pi(.|s, \boldsymbol{\theta})$
Take action A, observe R, S'
If S' is terminal: $\hat{v}(S', \mathbf{w}) \doteq 0$
$\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\, \hat{v}(S, \mathbf{w})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\, \nabla \ln \pi(A|S, \boldsymbol{\theta})$
$I \leftarrow \gamma I;$
$S \leftarrow S';$

Figure 2.10. Actor-critic.

2.3.6. RL Applications in Energy Operations

Recent advances in DRL enabled solving large and complex MDPs [2, 52]. The DRL models can observe the complete MDP state space, i.e., the entire futures price curve and many other operational variables, without the need to approximate or discretize the state space. Furthermore, feature extraction is handled by the deep ANN without the need for prior problem domain knowledge. However, the application of DRL in energy storage and trading is a recent phenomenon [53, 54]. Thus, we will broadly discuss DRL implementations for multiple control problems in the energy industry while emphasizing energy storage and trading applications.

The widespread adoption of renewable energy sources substantially increased the importance of energy storage [55]. The analysis of energy storage is quite complicated, given the variability and intermittency of power supply from renewable sources. Traditionally, energy power systems solution methods fall into two main categories [56]: (1) classical algorithms such as stochastic programming [57] and robust optimization [58], and (2) heuristic algorithms [59]. Both the classical and heuristic methods provide a deterministic policy. The increased penetration of renewable sources and uncertainty requires a policy solution that can adapt in real-time to the changing dynamics of the electric grid. Several papers have attempted to utilize DRL to solve this problem. For example, in economic scheduling problems, DRL was used to optimize battery capacity [60, 61].

Some energy experts believe natural gas and renewables are complementary energy sources [62, 63]. Synergies between natural gas and renewables exceed power supply stability. Promising technologies such as power-to-gas, in which excess renewable power is converted into hydrogen, highlight the increased importance of integrating natural gas and renewables. In a paper by [64], DRL was used to control a simulated power grid with natural gas and renewable energy. Their experiments have shown that DRL can support grid operators in real-time control of renewable and natural gas operations. DRL has also been shown to improve efficiency in energy consump-

tion. In their study [65], DRL was shown to reduce energy consumption in residential and small commercial buildings. This is achieved by incorporating DRL into the demand response systems that automatically adjust the energy consumption level as the market-driven electricity prices evolve. Renewable energy producers sell their power supply by participating in the electricity market. In this market, power suppliers place bids for future demand. The problem facing renewable energy producers is predicting their future power supply levels, given the weather conditions forecasts. Failing to supply power for a given bid results in penalties by regulatory agencies. DRL was used in a study by [66] to devise a profitable bidding strategy for wind power producers. Refer to [67] for a review of DRL applications in optimal bidding and dispatch strategies in the power market.

RL and agent-based simulation were used to model the oil value chain. In their paper, [68] built a simulation that resembled an integrated oil value chain, from production to refining and distribution. They showed that agents can learn and derive complex operational policies by allowing them to learn independently during the simulation. The goal of deriving operational policies was to enable economic feasibility studies regarding investments in the oil value chain. The simulation allows decision-makers to evaluate different configurations and investment decisions, such as adding storage capacity levels, increasing the number of ship tankers, or increasing the capacity of certain pipelines. Furthermore, DRL has been shown to improve the economics of energy production optimization. In their paper, [69] used DRL to improve the efficiency of oil-well production. Real-time data from the reservoir operations were fed back into their algorithm, which guided the search for the optimal production policy. Refer to [70] for a recent survey on DRL application in oil and gas production.

DRL has been used in the real-time optimization of time-critical processes. For example, the policy must be adjusted to meet operational requirements in an oil refinery. However, reconfiguring the refinery's sub-units is too expensive and thus requires careful planning. An actor-critic DRL model was used by [71] to optimize the refinery policy in real-time. The results have shown that the model produced policies with high

accuracy. Another RL application area in the oil and gas industry has been in production system maintenance scheduling. A multi-agent SARSA simulation was used by [72] to optimize maintenance scheduling in an oil refinery. The simulation experiments have shown that RL could solve the refinery scheduling problem using online data. Refer to [73] for additional applications of RL in maintenance optimization in the oil and gas industry.

RL was studied in managing pipeline capacity. Some pipelines are used to transport multiple petroleum products. This is known as batch scheduling. In batch scheduling, the problem of managing the pipeline capacity consists of three interrelated tasks: (1) batch planning, (2) tracking each product in the pipeline, and (3) monitoring the pipeline's operational conditions. This problem was solved using DRL by [74]. Another area of DRL research is in coal mine operations, where production policy is closely associated with mine security considerations. Risks in coal mine operations include gas outbursts [75], roof falls [76], and flooding [77]. Many methods were proposed to assess the risks of coal mines and address these complex issues. Recently, RL was used in this pursuit as well. Q-learning with a cost factor was used by [78] to account for the risk aversion preference in coal mine operations. The simulated results showed that RL can aid in scheduling such complex operational tasks.

3. IMPLEMENTATION

3.1. Soft Actor-Critic

While achieving successful results on small tasks, the performance of many deep RL algorithms in large, complex real-world problems still faces many challenges. First is very high sample complexity. As the computational cost increases substantially with large MDPs, this is especially true with continuous state and action spaces. Second is a high variance in performance due to hyperparameter configurations, i.e., instability in algorithm performance due to initial seeds, different learning rates, and other constants. Recently, many have proposed algorithms to tackle these challenges. For this thesis, we have chosen a new and promising algorithm called soft actor-critic (SAC). Based on the actor-critic architecture, SAC outperformed many state-of-the-art algorithms even in complex problems with continuous action spaces [79]. SAC is an off-policy actor-critic DRL algorithm that has the following main features: (1) separate ANNs for the actor and critic, (2) off-policy learning, and (3) maximum entropy reward function. The max entropy reward function aims to maximize both the expected reward and the entropy of the policy. By maximizing the entropy of a policy, the actor acquires diverse behavior that encourages exploring as many actions as possible aimed at performing the task, which is a desirable characteristic as opposed to directly learning a deterministic greedy policy which may not be optimal in many problems [80]. Moreover, maximum entropy formulation has been shown to improve exploration and robustness significantly [81]. The maximum entropy reward function is defined as follows

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \right], \tag{3.1}$$

where the expected cumulative reward is influenced by the entropy function $H(\pi(\cdot|s_t))$ scaled by a temperature variable α. The temperature variable α can recover the original reward function at the limit as $\alpha \to 0$. The expectation is taken over the trajectory distribution $\rho_\pi(s_t, a_t)$ which is induced from policy $\pi(a_t|s_t)$. The entropy maximization

reward function improves the exploration of the MDP while simultaneously excluding futile regions. Furthermore, the stochastic policy allows the actor to learn multiple optimal actions concurrently.

In SAC, three distinct ANNs are used to approximate the following: soft value function $V(s, \boldsymbol{\psi})$, soft Q-function $Q(s, a, \boldsymbol{\phi})$ and finally policy $\pi(a|s, \boldsymbol{\theta})$. Each has the following parameters $\boldsymbol{\psi}$, $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ respectively. We will define the performance objectives and update the rules for the approximated functions. The update rules follow the same procedures of SGD and SGA previously discussed. The performance objective of the approximated value function is defined as follows,

$$J_V(\boldsymbol{\psi}) = \mathbb{E}_{(s_t)\sim\mathcal{D}}\left[\frac{1}{2}(V(s_t, \boldsymbol{\psi}) - V(s_t))^2\right]. \tag{3.2}$$

Note the similarity with Equation 2.44. The distribution $\mathcal{D}$ is a distribution of previously encountered observations or it could be based on experience replay. Since $V(s_t)$ is not known at time t, we approximate it as

$$V(s_t) = \mathbb{E}_{a_t\sim\pi_{\boldsymbol{\pi}}}\left[Q(s_t, a_t, \boldsymbol{\phi}) - \log \pi(a_t|s_t, \boldsymbol{\theta})\right], \tag{3.3}$$

where now we call $V(s_t)$ as a soft state value function. Taking the gradient of Equation 3.2 with respect to $\boldsymbol{\psi}$ results in the value function update rule

$$\hat{\nabla}_{\boldsymbol{\psi}} J_V(\boldsymbol{\psi}) = \nabla_{\boldsymbol{\psi}} V(s_t, \boldsymbol{\psi})\left(V(s_t, \boldsymbol{\psi}) - Q(s_t, a_t, \boldsymbol{\phi}) + \log \pi(a_t|s_t, \boldsymbol{\theta})\right). \tag{3.4}$$

By defining $V(s_t)$ as a function of the parameterized Q-function $Q_{\boldsymbol{\phi}}$ and policy $\pi_{\boldsymbol{\theta}}$, the update rule of $V_{\boldsymbol{\psi}}$ becomes more stable. The soft Q-function performance objective is defined as

$$J_Q(\boldsymbol{\phi}) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q(s_t, a_t, \boldsymbol{\phi}) - \hat{Q}(s_t, a_t)\right)^2\right]. \tag{3.5}$$

We incorporate our soft state value into Equation 3.5 by defining $\hat{Q}(s_t, a_t)$ as follows

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_p\left[V(s_{t+1}, \bar{\boldsymbol{\psi}})\right], \tag{3.6}$$

hence the update rule for the parameterized Q-function $Q_{\boldsymbol{\phi}}(s, a)$ becomes

$$\hat{\nabla}_{\boldsymbol{\phi}} J_Q(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} Q(s_t, a_t, \boldsymbol{\phi})\left(Q(s_t, a_t, \boldsymbol{\phi}) - r(s_t, a_t) - \gamma V(s_{t+1}, \bar{\boldsymbol{\psi}})\right). \tag{3.7}$$

Note the use of a moving average for the soft state value function parameter $\bar{\psi}$. In practice, it has been shown that using either a moving average or a periodic update of the target network improves stability. Periodic target network updates have been discussed previously with DQN [3,4]. Finally, we define the policy performance objective as follows

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\log \pi(a_t|s_t,\boldsymbol{\theta}) - Q(s_t,a_t,\boldsymbol{\phi})\right]. \tag{3.8}$$

To differentiate $Q_{\boldsymbol{\phi}}$ using the policy parameters $\boldsymbol{\theta}$, we implicitly define the policy $\pi_{\boldsymbol{\theta}}$ by another differentiable function, i.e., ANN, as follows

$$a_t = f_{\boldsymbol{\theta}}(\epsilon_t; s_t). \tag{3.9}$$

Therefore, Q-function and policy are now $Q_{\boldsymbol{\phi}}(s, f_{\boldsymbol{\theta}}(\epsilon; s))$ and $\pi_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\epsilon; s)|s)$ respectively. The function $f_{\boldsymbol{\phi}}$ is an ANN with input noise parameter ϵ. The update rule for the policy is as follows,

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t,\boldsymbol{\theta}) + (\nabla_{a_t} \log \pi(a_t|s_t,\boldsymbol{\theta}) - \nabla_{a_t} Q(s_t,a_t,\boldsymbol{\phi}))\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\epsilon_t; s_t). \tag{3.10}$$

Initialize parameter vector $\psi, \bar{\psi}, \theta, \phi$

Loop:

For each environment step do:

$a_t \sim \pi_\theta(a_t|s_t)$

$s_{t+1} \sim p(s_{t+1}|s_t, a_t)$

$\mathcal{D} \sim \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$

end for

For each gradient step

$\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$

$\phi_i \leftarrow \phi_i - \lambda_Q \hat{\nabla}_{\phi_i} J_Q(\phi_i)\ for\ i \in \{1,2\}$

$\theta \leftarrow \theta - \lambda_\pi \hat{\nabla}_\theta J_\pi(\theta)$

$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$

end for

End loop

Figure 3.1. SAC.

Figure 3.1 shows the pseudocode for SAC adopted from [79]. The SAC algorithm is structured to alternate between environment interactions and gradient updates. During the environment steps, the SAC appends the tuple $(s_t, a_t, r(s_t, a_t), s_{t+1})$ to its experience replay memory $\mathcal{D}$. As for the gradient update steps, note that for the Q-function, the SAC algorithm utilized the double Q-learning (DQL) method [82]. In some MDPs, Q-learning tends to have a positive bias in estimating the Q-function; this issue is also present in actor-critic architectures [83]. SAC uses DQL to address this issue by updating two sets of the parameter $\boldsymbol{\phi}$. Then the $Q_{\boldsymbol{\phi}}$ with the lowest value is used in updating the value function and policy in Equations 3.4 and 3.10. Finally, SAC uses exponential smoothing for the moving average parameter $\bar{\boldsymbol{\psi}}$.

3.2. Implementation

In our implementation, we expanded the natural gas storage MDP formulation discussed earlier in Section 2.2 initially proposed by [21]. We added three variables to the MDP's state space. First, we included a vector containing the natural gas futures contract prices for the next four months. Second, we included an indicator for the storage utilization level. Finally, we added the average price of natural gas currently held in storage. The expanded MDP formulation aims to follow industry practice by incorporating futures prices into the model. We restricted our futures curve prices to the next four months. This restriction is due to an observed market phenomenon where energy futures contracts on extended maturity dates exhibit high volatility and low liquidity [84, 85]. Hence, futures prices for further maturities could cause undesirable noise for our model. We used actual data for spot and future contract prices downloaded from the Energy Information Administration (EIA). In Figure 3.2, we show a sample of the natural gas pricing information obtained from EIA. Note that the four front-month futures prices provide an indication of spot price movement.

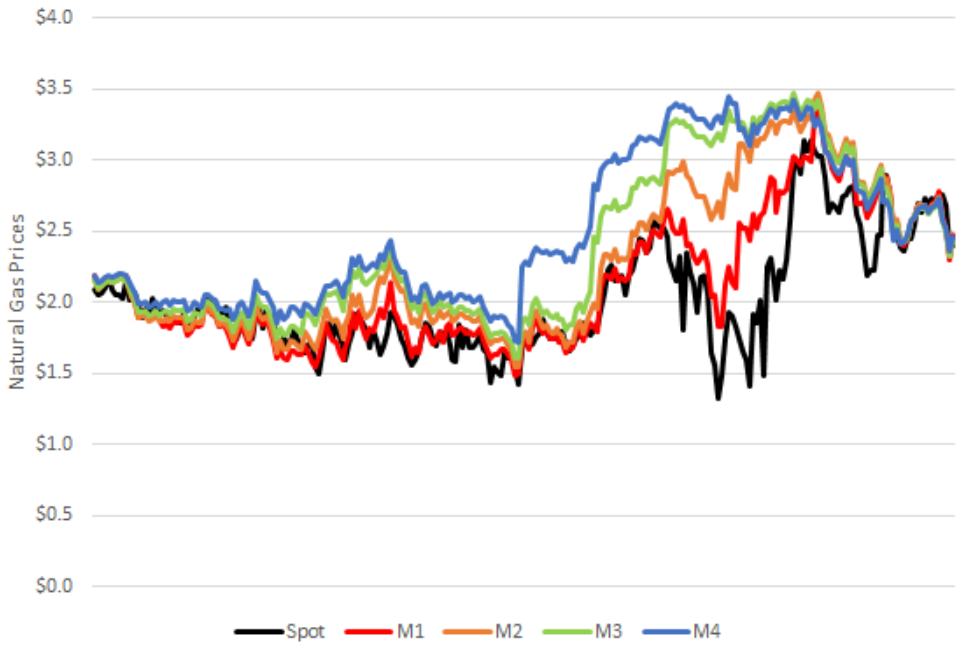

Figure 3.2. Sample of natural gas price information.

Table 3.1. MDP specification.

MDP Element	**Specifications**
Action Space	Scalar $[-1, 1]$
Natural Gas Price Information	Daily Spot and Four Prompt Months Futures Prices
Inventory Level	Scalar $[0, 1]$
Average Price of Inventory in Storage	Scalar
Storage Space	1000 Units
Max Flow Capacity	100 Units/Day
Symmetrical Flow	Yes

The other two variables we included in the MDP state space aim to improve our SAC model's training process. We expect the agent to take advantage of the information about the storage utilization level and the current average price of natural gas in storage. The storage utilization level is a $[0, 1]$ normalized scalar. Additionally, we designed the storage flow capacity to be symmetrical, i.e., inflow and outflow rates are identical. Furthermore, we control the flow of natural gas in our storage facility using our capacity function. At any given time, the maximum flow capacity is less than or equal to 10% of the total storage space. In each step, the agent observes the state of the MDP, which includes the current spot price, the next four months' futures contract prices, storage utilization level, and the average price of natural gas held in storage. Given the observed MDP state, the SAC algorithm uses the policy gradient ANN architecture to derive an action in each step. The action is a normalized scalar $[-1, 1]$. Where -1 indicates a strong signal to withdraw and sell, 1 indicates a strong signal to buy and inject, and 0 is a signal to do nothing. We interpolate the action signal into the set of feasible actions through the capacity function. Given the flow restrictions and current inventory level, the capacity function controls the amount of natural gas traded at each time step. Once a feasible action is obtained from the capacity function, it is then passed onto the reward function. The reward function calculates the reward signal by accounting for the costs associated with withdrawal and injection actions. In our reward function, we incorporated the in-kind and marginal costs discussed earlier in Section 2.2. We used the same in-kind and marginal cost values given in [21].

Table 3.2. Reward function specification.

Constants	**Value**
Injection Fuel Loss Factor	1%
Withdrawal Fuel Loss Factor	0%
Injection Marginal Cost	\$0.02/Unit
Withdrawal Marginal Cost	\$0.02/Unit

In our MDP formulation, we assume that in each time step t, the action a_t is processed during the period $[t, t+1)$. Hence, the capacity function ensures that the daily flow capacity is associated with the same period $[t, t+1)$. Additionally, we set the reward r_t to be realized immediately at time t. Therefore, the effects of action a_t with respect to the (1) storage utilization, (2) cumulative reward, and (3) average price of natural gas in storage, are all reflected at time $t+1$. The duration of each episode in our experiments is set to 1,000 steps. These steps are given by the fact that the spot and futures data used cover four years from 2019 to 2022, excluding weekends and other occasions where no natural gas trading occurs.

Table 3.3. Comparison of our model and ADP models.

Model Features	**ADP Models** [21, 26, 29]	**Our Model**
Decision Intervals	Monthly	Daily
Policy Actions	Low-Resolution	High-Resolution
Policy Approximation	No	ANN
Value Function Approximation	No [21, 26] Linear Basis [29]	ANN
Requires Price Model	Yes	No
Includes Futures Curves	No [21, 26] Yes [29]	Yes
High-Dimensional State Space	No [21, 26] Futures Curve [29]	Yes
High-Dimensional Action Space	No	Yes

In Table 3.3, we illustrate the difference between previous ADP models discussed earlier in Section 2.2 and our SAC model. Previous models share the same low-resolution action given its monthly decision interval. In practice, however, energy merchants operate daily on the spot market and utilize the monthly futures data to anticipate price evolution. The monthly decision policy given by the previous ADP models and the LP-based heuristics also doesn't reflect the reality of energy merchant operations. Additionally, our model doesn't assume any low-resolution basestock policy structure, as our model can handle high-resolution ANN-based policy. Moreover, the SAC model utilizes ANNs to approximate the policy and value function, the actor and critic, respectively. ADP methods are based on a deterministic greedy policy for a

specific MDP instance. The ADP methods generate MDP instances using Monte Carlo simulations based on a pricing model. Our SAC model learns a policy online using real data and doesn't require a price model. We included a scalar indicating the average price of natural gas in storage. Meanwhile, the ADP models use a mean-reverting price model. Hence, the ADP solution method implicitly assumes a mean price for natural gas when making trading decisions, irrespective of the cost of natural gas in storage. We believe that energy merchant decisions would be significantly influenced by the actual cost of natural gas in storage. Finally, using ANNs in our models enabled us to expand the MDP state and action spaces considerably. This allowed us to incorporate many state variables without discretization; more importantly, our model can learn a complex policy with many decision variables. ADP methods, on the other hand, are limited in this regard.

One of the main issues in stochastic optimization, in general, and DRL in particular, is the problem of benchmarking and the reproducibility of results. According to [86], this issue renders judging reported results of state-of-the-art DRL algorithms hard, if not meaningless. They argued that in DRL, many factors, other than the choice of algorithm, influence results significantly. These factors include hyperparameters, ANN architecture, reward scale, random seeds, MDP environments, and code bases.

Table 3.4. Experiment design.

Experiment Design Factor	**Choice**
Hyperparameter Selection	Formal Search Procedure
MDP Environment	OpenAI Gym
Code Base	Stable-Baselines 3
Training Duration	Five Million Steps
Total Data Points	1000 Observations
Model Evaluation Intervals	Every 100,000 Steps

In our experiments, we attempted to address these issues as much as possible. On the issue of hyperparameter selection. We selected our SAC hyperparameters using a hyperparameter optimization framework, Optuna [87]. We sampled 500 different hyperparameter configurations and selected a combination of hyperparamers that performed better on average for our task. On the issue of random seeds, we trained multiple instances of our SAC models concurrently to account for the random initialization of each agent's ANN initialization parameters. Additionally, we used OpenAI Gym [88,89] to formulate our MDP environment, and the choice of using OpenAI Gym was based on similar experiments in the literature. Regarding the issue of variations in code bases, i.e., implementations of DRL algorithms, it has been shown that differences in implementations produced significant variations in results. The experiments done by [90] have demonstrated that even small code-level optimization substantially impacts results more than algorithmic choice. These findings highlight the importance of standardizing DRL implementations to assess and compare results objectively. There are many benchmarked DRL implementations available to the DRL research community. For our experiments, we used the Stable-Baselines 3 DRL library [91].

3.3. Results

In our experiments, we trained multiple agents of the SAC model. Training multiple agents aims to control random seeding and ANN initialization parameters. The training is plotted in Figure 3.3. The average reward is shown in red, while shaded regions show the performance range of the agents. The results show that SAC learned a policy for our natural gas storage problem. While SAC succeeded in learning a policy for our training agents, the performance variance is quite noticeable. We believe that initial random seeds of the actor and critic ANNs have impacted the learning process. In Figure 3.4, we plot the highest and lowest-performing SAC models. While the developers of SAC claimed that it is robust with respect to hyperparameters [79], our results indicate that it is still quite challenging to escape a poorly initialized ANN. Having said that, we expect that increasing the training budget and using an adaptive entropy schedule might mitigate this issue.

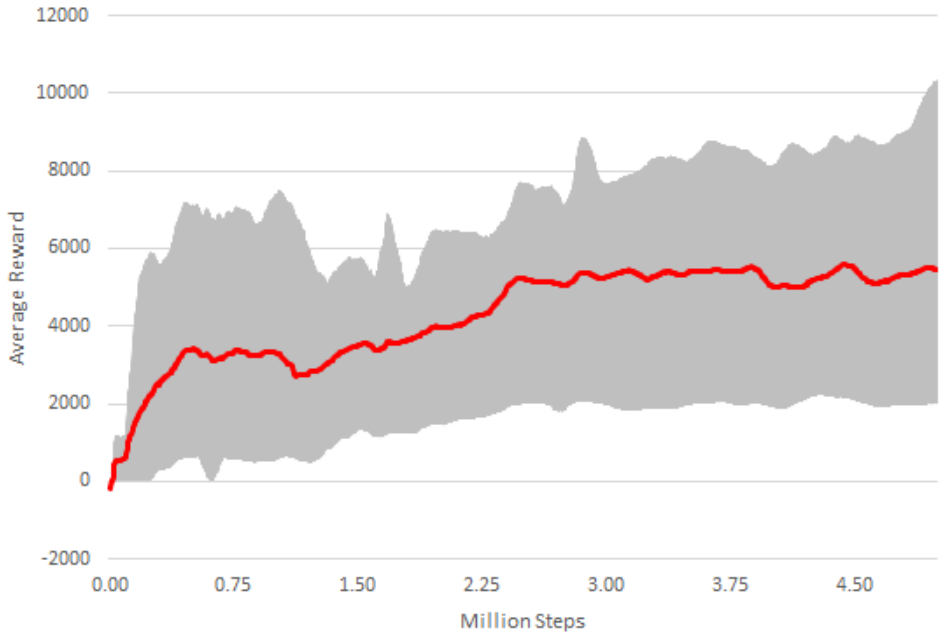

Figure 3.3. Performance of SAC.

Unlike the experiments done by the SAC developers, which focused on robotics control simulations, our experiment demonstrates that our capacitated storage and reward structure, governed by the capacity and reward functions, presented SAC with a more challenging task. While the multiple ANNs used in SAC enable the agent to learn a policy in training, our experiments have shown that training the SAC architecture is computationally expensive, requiring a large training budget. This is a known challenge in off-policy DRL algorithms, as we aim to train multiple ANNs with constantly changing targets. This is in contrast with machine learning, where a labeled data set is fixed during the training process. The source of changing ANN targets in DRL is the updating done to the policy and value functions, i.e., the actor and critic.

We trained the SAC model on a Core i7 10510U 1.8GHz CPU with 8GB of RAM. We noticed that the SAC algorithm requires less memory than we anticipated. Most of the processing demand was on the CPU. We did not use the GPU due to PyTorch's lack of support for AMD GPUs on Windows OS, on which Stable-Baselines 3 is based. Another interesting aspect is that SAC allows us to use its policy in either a stochastic or a deterministic manner. Given the current observation in the stochastic policy approach, the SAC model would sample an action. While in the deterministic

policy, the SAC model would pursue a greedy policy. Figure 3.5 compares each policy type's average and maximum rewards. In addition, we included the standard deviations average and maximum. We noticed that there is not much difference in the performance between the two types of policies.

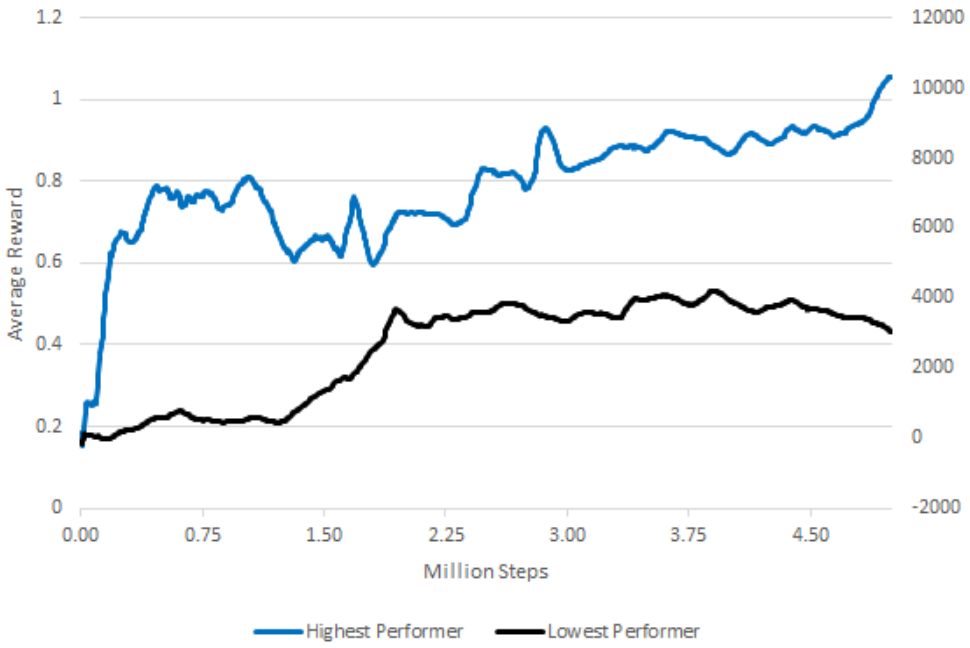

Figure 3.4. Performance of SAC.

The choice of policy type when deploying such a model in practice will depend on the task at hand. The problem of competitively managing capacitated storage might benefit from a stochastic policy since it would pursue a riskier trading strategy. A deterministic policy might be more suited for risk-averse tasks like robotics, as we would like a more predictable course of action. Depending on the situation, we might sometimes want to switch from a stochastic to a deterministic policy and vice-versa. An example of such a situation would be in navigation applications. At the beginning of a lengthy trip, we might want the navigation system to assess different routes using a stochastic policy. As we approach the destination, the directions given by the navigation system should follow a deterministic policy.

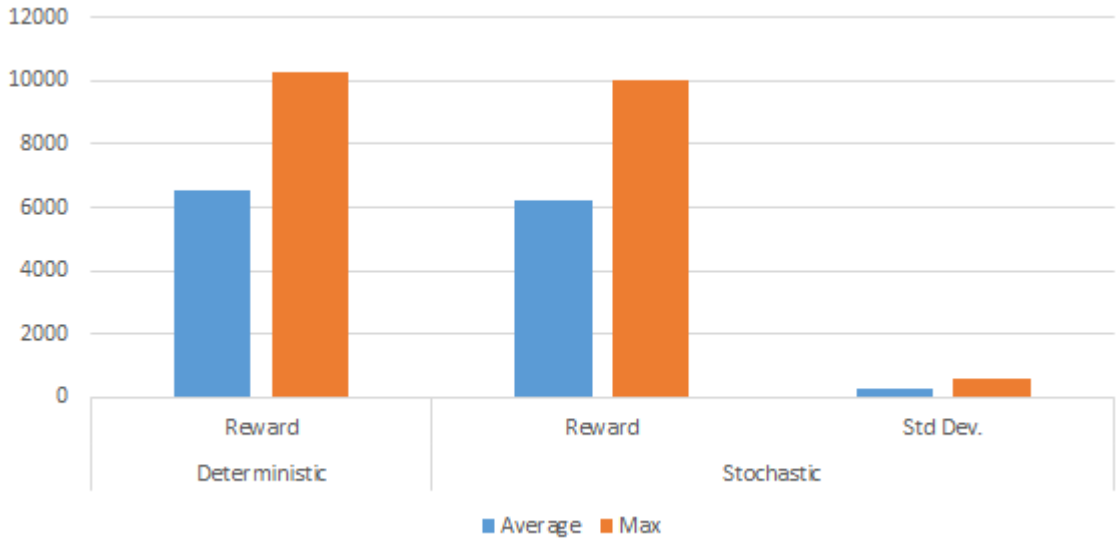

Figure 3.5. SAC policy type effect on performance.

We performed additional computational experiments to compare SAC's results with other state-of-the-art DRL algorithms. We used two algorithms that follow the actor-critic architecture similar to SAC. These algorithms are Proximal Policy Optimization (PPO) [92] and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [83]. Both PPO and TD3 have demonstrated outstanding results in complex control tasks. PPO was the first algorithm to beat a human champion in an esports game. In the strategic game Dota 2, PPO has beaten the human world champions on live stream [93]. On the other hand, TD3 has shown impressive performance improvements in robotics control, surpassing SAC [83]. Before running the PPO and TD3 models directly on our MDP environment, we performed the hyperparameter tuning routine similar to SAC's. The results of our computational experiments on PPO and TD3 are shown in Figure 3.6. While PPO and TD3 have succeeded in many challenging tasks, they weren't so in our problem of competitively managing the capacitated storage. Our assessment of these results is that the SAC has performed much better on the given tasks due to its modified maximum entropy reward function, which incentivizes exploration and reward maximization. On the other hand, PPO and TD3 incorporate clipping mechanisms, limiting changes to the actor's policy during each gradient update. In the following sections, we will provide a more detailed analysis of the performance of these algorithms.

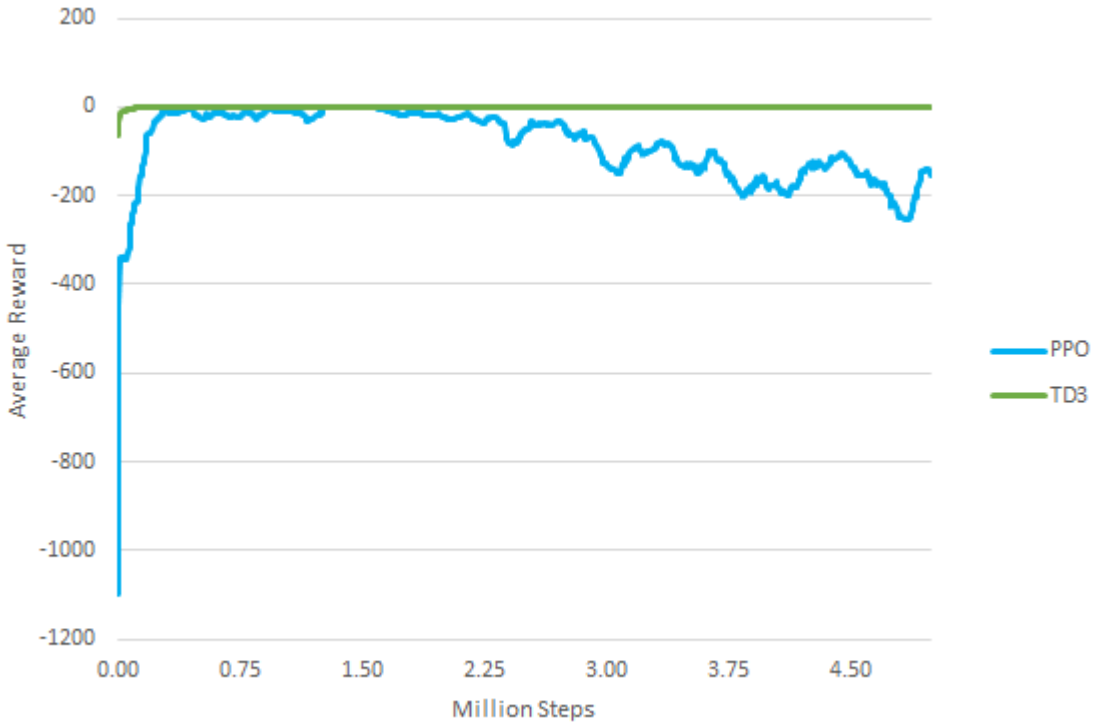

Figure 3.6. Performance of PPO and TD3.

3.4. Validation

Even though SAC managed to learn a policy for our problem. We still need to verify its generalization ability. In DRL, the use of ANN can introduce the risk of overfitting. Instead of learning from interacting with the MDP and then generalizing this learning into new observations, an agent might memorize the MDP, resulting in undesirable behavior when the environment is slightly modified. While in supervised learning, there are many formal standard techniques to verify overfitting, in DRL, there is a lack of consensus in the research community on this matter [94]. Typically, in DRL, stochasticity is introduced into the training environment to improve robustness. However, the effectiveness of such measures is not guaranteed and overfitting can still occur. We conducted a further analysis using a validation test dataset to verify if our SAC model is overfitting. Using a test dataset for validation is an essential and practical measure in verifying the models' generalization ability from training data. We used the same 1000 observations covering the natural gas price information for 2019-2022 for our training data. Our validation data has 249 observations for the year

2023. The observations are split into 80-20 for training and validations, respectively. In Table 3.5, we summarize our validation experiment parameters. We used the same training dataset to guide the policy update, we then validated the model every 10,000 steps using our validation dataset.

Table 3.5. Validation setup.

Parameter	**Choice**
Training Period	5 Million Steps
Early Stopping	No
Validation Interval	Every 10,000 Steps
Training Data Size	1,000 Observations
Validation Data Size	249 Observations

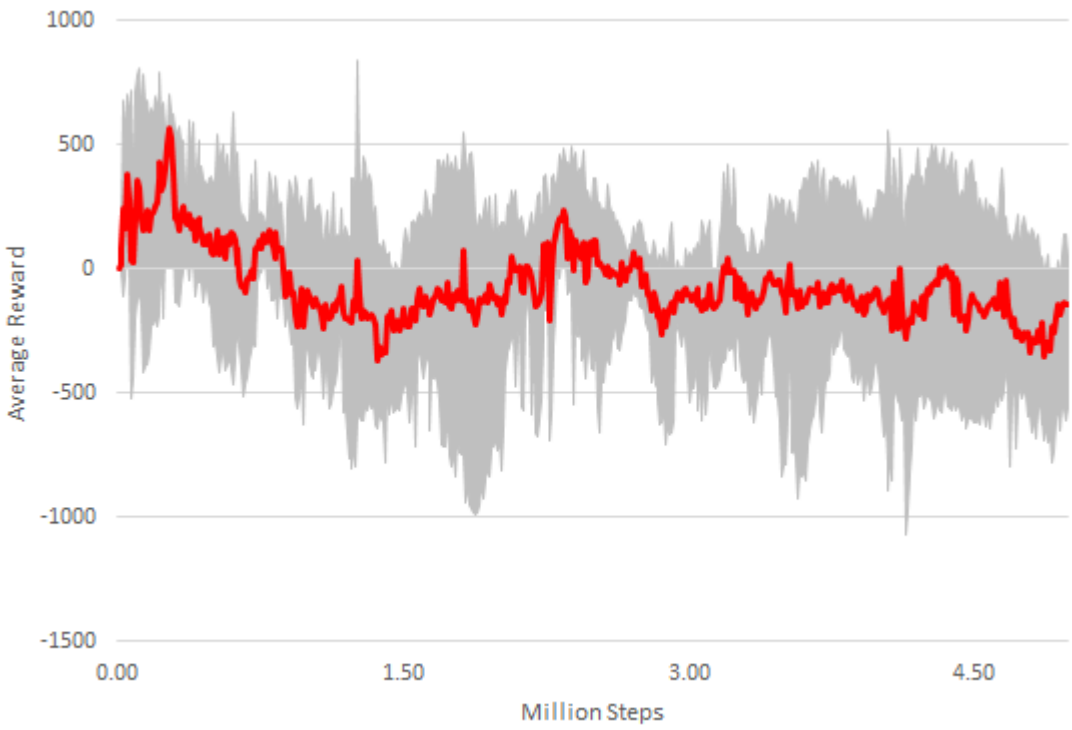

Figure 3.7. Validation performance of SAC.

In Figure 3.7, we have the performance of our SAC model on the validation dataset. We can see that the model could not learn a policy that generalizes from the training data into the validation data. Several factors might have contributed to these results. First, our MDP environment is deterministic due to the capacitated storage MDP, which does not have stochastic elements. This determinism lowers the sample complexity required by DRL algorithms. For our SAC model, which utilizes a large replay buffer to update the policy, a deterministic environment lowers the variety of samples the algorithm learns from, which in turn causes overfitting. Second, our datasets comprise a time series of natural gas prices. This temporal correlation of prices is at odds with the assumption that observations are independent. Correlation of recent observations would cause overfitting. Third is nonstationarity in the dataset. Our combined training and validation datasets span five years; hence, nonstationarity is highly likely. Nonstationarity need not be across training and validation datasets; the training dataset alone can contain nonstationarity, resulting in poor learning and generalization. We used the Augmented Dickey-Fuller (ADF) and Kwiatkowski–Phillips–Schmidt–Shin (KPSS) statistical tests to check for stationarity in our training data.

Table 3.6. Summary of Nonstationarity test results.

Variable	**ADF Test Result**	**KPSS Test Result**
Spot	Nonstationarity (p-value: 0.1718)	Nonstationarity (p-value: 0.01)
Front Month 1	Nonstationarity (p-value: 0.5031)	Nonstationarity (p-value: 0.01)
Front Month 2	Nonstationarity (p-value: 0.3910)	Nonstationarity (p-value: 0.01)
Front Month 3	Nonstationarity (p-value: 0.3975)	Nonstationarity (p-value: 0.01)
Front Month 4	Nonstationarity (p-value: 0.3945)	Nonstationarity (p-value: 0.01)

The results show that, indeed, our training data exhibits nonstationarity. The presence of nonstationarity in training data hinders learning and generalization. For example, value function approximations, i.e., the critic, can become unstable given how data distribution changes. As the critic's approximation degrades, so does the actor, which leads to suboptimal performance. Furthermore, exploring the action space becomes more challenging, given that rewards are directly influenced by the spot price information. Finally, prolonged training on a dataset with nonstationarity might show that performance is improving, while in reality, the model could be overfitting.

3.5. Major Observations

Besides the issue of generalization. We need to discuss why some of our models couldn't learn policy from our training dataset. Although SAC, PPO, and TD3 are all DRL algorithms based on the actor-critic architecture. We believe that the variance in their performance in training is mainly driven by the policy specification. In Table 3.7, we compare our DRL algorithms policy specifications.

Table 3.7. Comparison of algorithms.

Policy Specification	**PPO**	**TD3**	**SAC**
Type	Stochastic	Deterministic	Stochastic
Update Target	On-Policy	Off-Policy	Off-Policy
Exploration Method	Policy Sampling	Gaussian Noise	Policy Sampling and Entropy Regularization
Stabilization	Policy Clipping	Policy Clipping, DQL and Delayed Update	DQL and Entropy Regularization

PPO and TD3 policy specifications prioritize exploitation, as opposed to SAC, which prioritizes stability and exploration. Even though PPO has a stochastic policy,

it is restricted by an on-policy update, meaning that the agent's behavior and target policy are identical. This limits the PPO's ability to explore the action space and lowers its sample efficiency. PPO's exploration is based primarily on action sampling from the policy. As the training progresses, the policy becomes more deterministic, reducing the exploration of the action space. Resulting in the policy converging to a sub-optimal solution.

TD3 combines a deterministic policy with an off-policy target by adding Gaussian noise to the target action selection. The Gaussian noise added to the target action helps in stabilizing the policy. TD3 adds another Gaussian noise to the behavior policy action selection to encourage exploration. This is a necessary step as the action selection in TD3 is based on the deterministic policy, as there is no action sampling in TD3. PPO's stochastic policy and TD3's off-policy weren't enough to allow for the action space exploration required for our problem, as both algorithms converged to bad solutions, i.e., do nothing. On the other hand, SAC policy is stochastic and updates the target using an off-policy method. Additionally, SAC incorporates a maximum entropy reward function. Combining these three elements enabled SAC to perform our task and learn a policy successfully. Moreover, DRL algorithms that follow the actor-critic architecture typically employ policy stabilization measures. These measures aim to mitigate the risks of abruptly updating the policy and causing performance to deteriorate. We believe policy stabilization has played a significant role in the performance of PPO and TD3 algorithms. We previously discussed some of the policy stabilization measures. In Section 2.3.3, we discussed DQN's delayed network update. In TD3, the delayed update is applied to the policy instead. In Section 3.1, we discussed DQL. In addition to policy delayed updates and DQL, we have a new stabilization measure called policy clipping.

The policy clipping strategy used in PPO and TD3 aims to mitigate abrupt changes in the actor's policy and Q-function, respectively. While it is generally favorable to stabilize the training process, the clipping strategy causes the policy to get stuck in a local optimum solution. In our problem, the local optimum for PPO and

TD3 was to do nothing. In Figure 3.8, we can see the average clipping fraction for PPO's policy update. Note how the clipping strategy prevents PPO from updating the policy early in the training process. Then, as the training progresses, the PPO policy becomes more deterministic, and clipping becomes irrelevant.

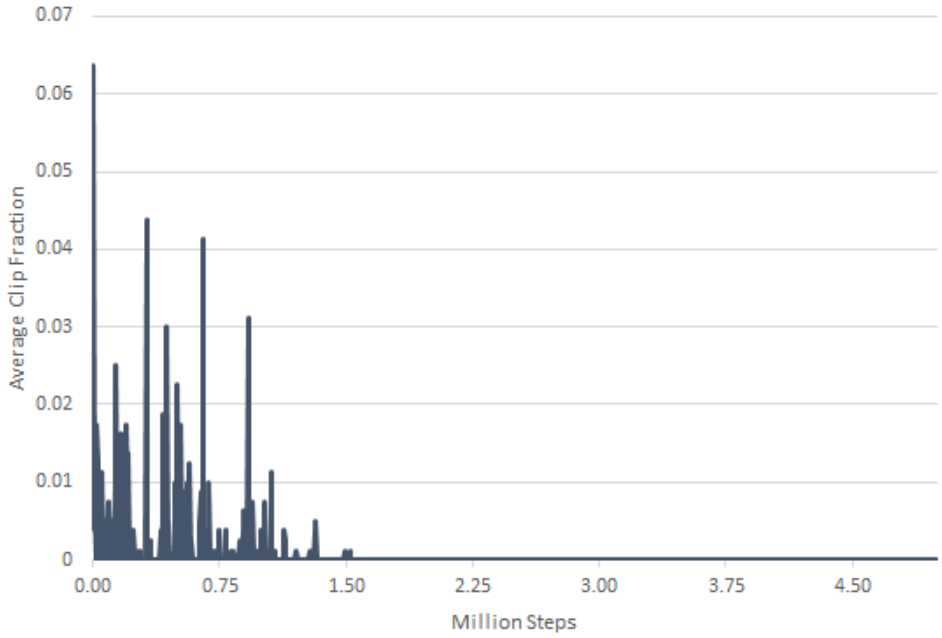

Figure 3.8. Average policy update clipping in PPO.

In Figures 3.9 and 3.10, we have the average actor loss, i.e., policy performance objective, of PPO and TD3, respectively. PPO and TD3 converge to a local optimum by minimizing the actors' loss. These results are inline Figure 3.6 where both algorithms could not learn a policy. Figure 3.11 shows the average SAC actor loss. Note that the actor loss in SAC was growing even as at the end of our training experiment. The increasing actor loss is due to higher levels of entropy. In Figure 3.12, we plot the SAC average performance against average entropy. Our SAC implementation utilizes an adaptive entropy schedule. Therefore, as the training stagnates near the end of our experiment, the entropy coefficient is increased to expand action space exploration. As the entropy increases, so does the actor loss. This dynamic adjustment of entropy resulted in better exploration and played a significant role in improving training performance. SAC's maximum entropy aids in guiding the policy away from low into high-reward regions [79].

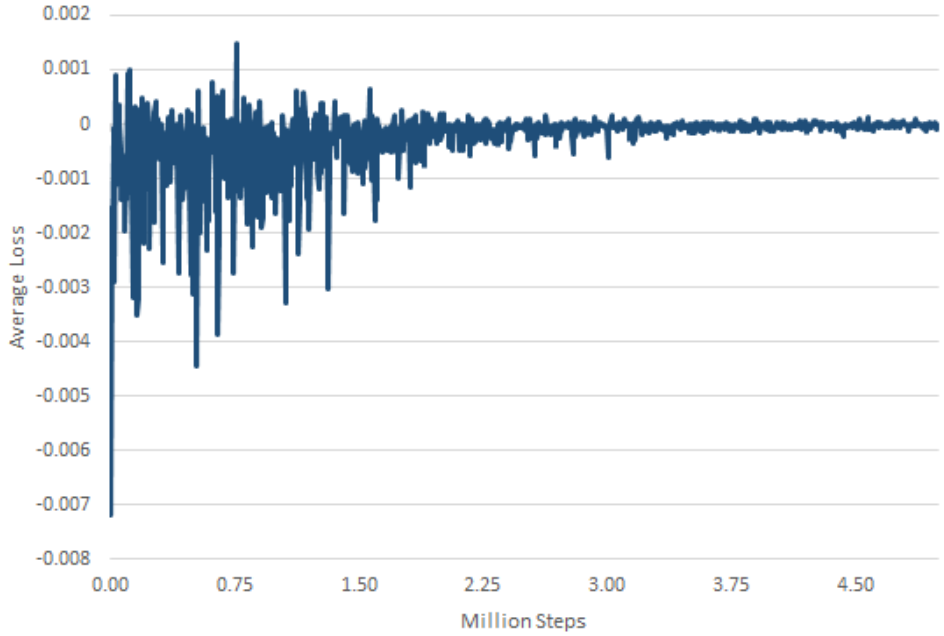

Figure 3.9. Average PPO actor loss.

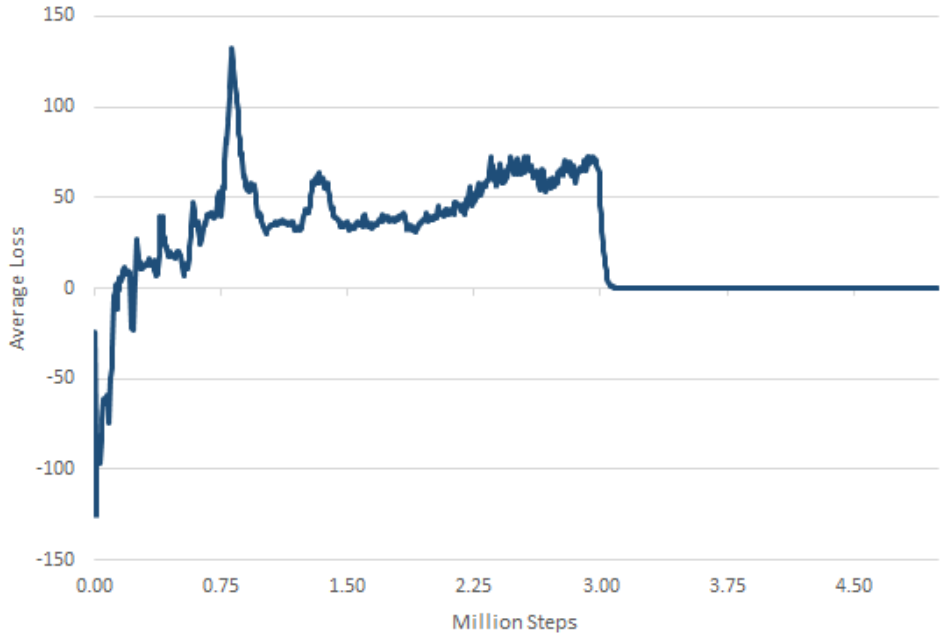

Figure 3.10. Average TD3 actor loss.

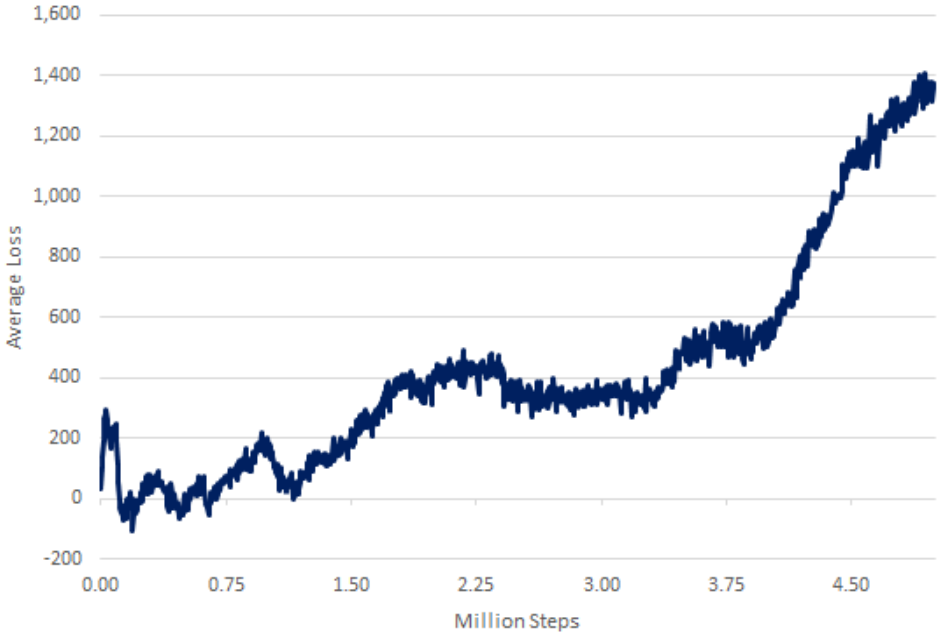

Figure 3.11. Average SAC actor loss.

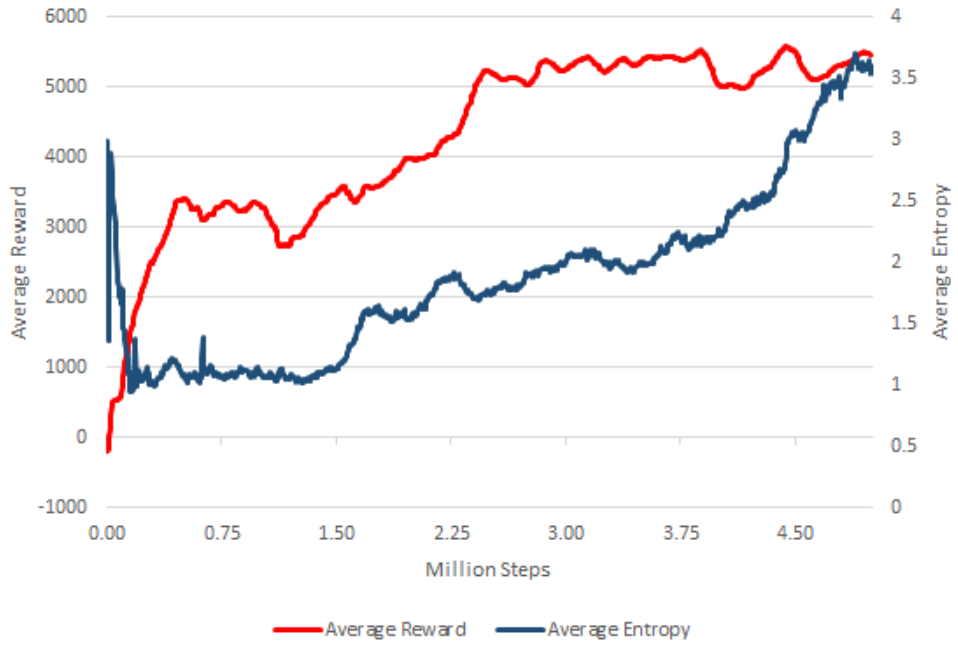

Figure 3.12. Average SAC reward vs entropy.

Our task requires the agent to have a long planning horizon for its actions. This extended planning is achieved by tolerating initial losses due to buy-and-inject actions and withdrawing-and-selling for profit later. The SAC's maximum entropy reward function allows the agent to accept initial losses and enables longer planning by the agent. Conversely, PPO and TD3 weren't successful at this task, which caused the agent to become passive and do nothing.

SAC's incorporation of the maximum entropy reward function has proven very useful. The policy is incentivized to explore the action space and learn complex actions. This is advantageous for our task, where an agent must conduct complex reasoning regarding the pricing information received and the physical constraints of storage space and flow capacities. Moreover, the stochastic policy can learn multiple optimal actions for the same state. This feature allows us to explore multiple solutions for the problem using the same stochastic policy. Finally, the maximum entropy reward function improves the algorithm's efficiency. This is achieved by exploring promising solution regions.

These findings point to a critical takeaway. While all of the algorithms used in our experiment follow the actor-critic architecture, different algorithmic implementations and policy specifications significantly impact task-specific performance. In other words, robotics and strategic games require a risk-averse action policy with small incremental changes. While other tasks, such as trading, would require the agent to take short-term risks and incur losses to learn a successful policy for these tasks.

Our experiments has shown that the SAC model can learn to train a policy for managing capacitated natural gas storage given spot and futures contract prices. However, our validation tests have shown that the generalization of our policy to a new dataset was not successful. Furthermore, we cannot comment on the optimality gap in our obtained policy. Nonetheless, the ability of SAC to learn such a difficult task is a promising indicator of DRL's potential in solving complex real-world problems.

4. CONCLUSION

The field of DRL is an exciting and promising research avenue. In recent years, it has achieved human-level performance in many tasks. One of the main strengths of DRL is the ability to learn how to perform a task without prior knowledge. DRL achieves this by updating a policy based on simulated or real-time experience.

However, DRL is not mature enough to be deployed to many real-world practical problems. While this might seem like a drawback, we would argue that it is the opposite, as this creates an opportunity for many advancements in DRL research. In this thesis, we used DRL to solve the problem of natural gas storage valuation. Given the evolution of energy prices, we experimented with DRL to value these assets. The main takeaway of our experiment is that DRL can learn a policy for valuing natural gas storage or any similar commodity. Our experiments have shown that different algorithmic implementations of DRL lead to varying learning for our agents. The SAC model, which uses a maximum entropy reward function, enabled the agent to learn an operating policy for our problem. Compared to other state-of-the-art DRL algorithms, namely PPO and TD3, SAC was more suitable for our problem. PPO and TD3's use of clipping policy updates to stabilize the learning process has made agents passive and unsuccessful in learning a policy for our task.

Our validation tests have shown that generalization is still a challenge for DRL. Even as our SAC model learns a policy for our training dataset, the results when we applied the policy to the validation dataset were poor. A major area for future work is to study DRL's generalization ability. We found very few papers that systematically address the issue of overfitting and generalization. Other areas of future work include studying the augmenting DRL algorithm features, such as the incorporation of SAC's maximum entropy into PPO, for example.

REFERENCES

1. Secomandi, N. and D. J. Seppi, "Real Options and Merchant Operations of Energy and Other Commodities", *Foundations and Trends in Technology, Information and Operations Management*, Vol. 6, No. 3-4, pp. 161–331, 2014.

2. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Massachusetts, 2020.

3. Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", ArXiv:1312.5602 [cs.LG], 2013.

4. Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-Level Control Through Deep Reinforcement Learning", *Nature*, Vol. 518, No. 7540, pp. 529–533, 2015.

5. Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, "Mastering the Game of Go Without Human Knowledge", *Nature*, Vol. 550, No. 7676, pp. 354–359, 2017.

6. Pirrong, C., *The Economics of Commodity Trading Firms*, Tech. rep., Trafigura, 2014.

7. Acharya, V. V., L. A. Lochstoer and T. Ramadorai, "Limits to Arbitrage and Hedging: Evidence from Commodity Markets", *Journal of Financial Economics*, Vol. 109, No. 2, pp. 441–465, 2013.

8. Birge, J. R., A. Hortaçsu, I. Mercadal and J. M. Pavlin, "Limits to Arbitrage

in Electricity Markets: A Case Study of MISO", *Energy Economics*, Vol. 75, pp. 518–533, 2018.

9. Ascher, J., P. Laszlo and G. Quiviger, *Commodity Trading at a Strategic Crossroad*, Tech. rep., McKinsey & Company, 2012.

10. Deloitte, *Trading Up, A Look at Some Current Issues Facing Energy and Commodities Traders*, Tech. rep., Deloitte, 2013.

11. Meersman, S., R. Rechtsteiner and G. Shrap, *The Dawn of a New Order in Commodity Trading*, Tech. rep., OliverWyman, 2012.

12. Williams, J. C. and B. D. Wright, *Storage and Commodity Markets*, Cambridge University Press, Cambridge, UK, 2005.

13. Pirrong, C., *Commodity Price Dynamics: A Structural Approach*, Cambridge University Press, Cambridge, UK, 2011.

14. Nuryyev, G., T. Korol and I. Tetin, "Hold-Up Problems in International Gas Trade: A Case Study", *Energies*, Vol. 14, No. 16, p. 4984, 2021.

15. Klein, B., "The Hold-Up Problem", P. Newman (Editor), *The New Palgrave Dictionary of Economics and the Law*, pp. 241–244, Macmillan Reference Limited, London, UK, 1998.

16. Joskow, P. L., "Vertical Integration and Long-Term Contracts: The Case of Coal-Burning Electric Generating Plants", *Journal of Law, Economics, & Organization*, Vol. 1, No. 1, pp. 33–80, 1985.

17. Joskow, P. L., "Vertical Integratio", *The Antitrust Bulletin*, Vol. 55, No. 3, pp. 545–586, 2010.

18. Cahn, A. S., "The Warehouse Problem", *Bulletin of the American Mathematical*

Society, Vol. 54, No. 1073, 1948.

19. Zipkin, P. H., *Foundations of Inventory Management*, McGraw-Hill, Irwin, New York, 2000.

20. Jaillet, P., E. I. Ronn and S. Tompaidis, "Valuation of Commodity-Based Swing Options", *Management Science*, Vol. 50, No. 7, pp. 909–921, 2004.

21. Secomandi, N., "Optimal Commodity Trading with a Capacitated Storage Asset", *Management Science*, Vol. 56, No. 3, pp. 449–467, 2010.

22. Maragos, S., "Valuation of the Operational Flexibility of Natural Gas Storage Reservoirs", E. Ronn (Editor), *Real Options and Energy Management Using Options Methodology to Enhance Capital Budgeting Decisions*, chap. 14, pp. 431–456, Risk Books, London, UK, 2002.

23. Gray, J. and P. Khandelwal, "Towards a Realistic Natural Gas Storage Model", *Commodities Now*, Vol. 6, pp. 1–4, 2004.

24. Gray, J. and P. Khandelwal, "Realistic Gas Storage Models II: Trading Strategies", *Commodities Now*, Vol. 9, pp. 1–5, 2004.

25. Nadarajah, S. and N. Secomandi, "A Review of the Operations Literature on Real Options in Energy", *European Journal of Operational Research*, Vol. 309, No. 2, pp. 469–487, 2023.

26. Lai, G., F. Margot and N. Secomandi, "An Approximate Dynamic Programming Approach to Benchmark Practice-Based Heuristics for Natural Gas Storage Valuation", *Operations Research*, Vol. 58, No. 3, pp. 564–582, 2010.

27. Jong, C. D., "Gas Storage Valuation and Optimization", *Journal of Natural Gas Science and Engineering*, Vol. 24, pp. 365–378, 2015.

28. Malyscheff, A. M. and T. B. Trafalis, "Natural Gas Storage Valuation via Least Squares Monte Carlo and Support Vector Regression", *Energy Systems*, Vol. 8, No. 4, pp. 815–855, 2017.

29. Nadarajah, S., F. Margot and N. Secomandi, "Comparison of Least Squares Monte Carlo Methods with Applications to Energy Real Options", *European Journal of Operational Research*, Vol. 256, No. 1, pp. 196–204, 2017.

30. Powell, W. B., *Approximate Dynamic Programming*, John Wiley & Sons, Hoboken, New Jersey, 2011.

31. Secomandi, N., G. Lai, F. Margot, A. Scheller-Wolf and D. J. Seppi, "Merchant Commodity Storage and Term-Structure Model Error", *Manufacturing & Service Operations Management*, Vol. 17, No. 3, pp. 302–320, 2015.

32. Puterman, M. L., *Markov Decision Processes*, John Wiley & Sons, Hoboken, New Jersey, 1994.

33. Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, 1957.

34. Dayan, P., "The Convergence of TD(λ) for General λ", *Machine Learning*, Vol. 8, No. 3-4, pp. 341–362, 1992.

35. Sutton, R. S., "Learning to Predict by the Methods of Temporal Differences", *Machine Learning*, Vol. 3, No. 1, pp. 9–44, 1988.

36. Rummery, G. A. and M. Niranjan, *On-line Q-learning Using Connectionist Systems*, Tech. rep., University of Cambridge, Department of Engineering, 1994.

37. Singh, S. P. and R. S. Sutton, "Reinforcement Learning with Replacing Eligibility Traces", *Machine Learning*, Vol. 22, No. 1-3, pp. 123–158, 1996.

38. Watkins, C. J. C. H., *Learning from Delayed Rewards*, Ph.D. Thesis, University of Cambridge, 1989.

39. John, G. H., "When the Best Move Isn't Optimal: Q-learning with Exploration", *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, p. 1464, Seattle, Washington, 1994.

40. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 210–229, 1959.

41. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress", *IBM Journal of Research and Development*, Vol. 11, No. 6, pp. 601–617, 1967.

42. Sutton, R. S., "Introduction to Reinforcement Learning with Function Approximation", *Advances in Neural Information Processing Systems*, Vol. 28, Montréal, Canada, 2015.

43. Sutton, R. S., *Temporal Credit Assignment in Reinforcement Learning*, Ph.D. Thesis, University of Massachusetts Amherst, 1984.

44. LeCun, Y., D. Touresky, G. Hinton and T. Sejnowski, "A Theoretical Framework for Back-Propagation", *Proceedings of the Connectionist Models Summer School*, Vol. 1, pp. 21–28, Pittsburgh, Pennsylvania, 1988.

45. Baird, L., "Residual Algorithms: Reinforcement Learning with Function Approximation", *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37, Tahoe City, California, 1995.

46. Bradtke, S., "Reinforcement Learning Applied to Linear Quadratic Regulation", *Advances in Neural Information Processing Systems*, Vol. 5, pp. 295–302, San Francisco, California, 1992.

47. Tsitsiklis, J. and B. V. Roy, "Analysis of Temporal-Difference Learning with Function Approximation", *Advances in Neural Information Processing Systems*, Vol. 9, p. 1075–1081, Denver, Colorado, 1996.

48. Maei, H., C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver and R. S. Sutton, "Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation", *Advances in Neural Information Processing Systems*, Vol. 22, p. 1204–1212, Vancouver, British Columbia, Canada, 2009.

49. Sutton, R. S., "On the Virtues of Linear Learning and Trajectory Distributions", *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference*, p. 85, Tahoe City, California, 1995.

50. Williams, R. J., "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", *Machine Learning*, Vol. 8, pp. 229–256, 1992.

51. Sutton, R. S., D. McAllester, S. Singh and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", *Advances in Neural Information Processing Systems*, Vol. 12, p. 1057–1063, Denver, Colorado, 1999.

52. Powell, W. B., *Reinforcement Learning and Stochastic Optimization*, John Wiley & Sons, Hoboken, New Jersey, 2021.

53. Curin, N., M. Kettler, X. Kleisinger-Yu, V. Komaric, T. Krabichler, J. Teichmann and H. Wutte, "A Deep Learning Model for Gas Storage Optimization", *Decisions in Economics and Finance*, Vol. 44, No. 2, pp. 1021–1037, 2021.

54. Hanga, K. M. and Y. Kovalchuk, "Machine Learning and Multi-Agent Systems in Oil and Gas Industry Applications: A Survey", *Computer Science Review*, Vol. 34, p. 100191, 2019.

55. Denholm, P., E. Ela, B. Kirby and M. Milligan, *Role of Energy Storage with Renewable Electricity Generation*, Tech. rep., National Renewable Energy Lab, 2010.

56. Cao, D., W. Hu, J. Zhao, G. Zhang, B. Zhang, Z. Liu, Z. Chen and F. Blaabjerg, "Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review", *Journal of Modern Power Systems and Clean Energy*, Vol. 8, No. 6, pp. 1029–1042, 2020.

57. Xu, Y., Z. Y. Dong, R. Zhang and D. J. Hill, "Multi-Timescale Coordinated Voltage/Var Control of High Renewable-Penetrated Distribution Systems", *IEEE Transactions on Power Systems*, Vol. 32, No. 6, pp. 4398–4408, 2017.

58. Li, P., C. Zhang, Z. Wu, Y. Xu, M. Hu and Z. Dong, "Distributed Adaptive Robust Voltage/Var Control with Network Partition in Active Distribution Networks", *IEEE Transactions on Smart Grid*, Vol. 11, No. 3, pp. 2245–2256, 2019.

59. Zhao, B., Z. Xu, C. Xu, C. Wang and F. Lin, "Network Partition-Based Zonal Voltage Control for Distribution Networks with Distributed PV Systems", *IEEE Transactions on Smart Grid*, Vol. 9, No. 5, pp. 4087–4098, 2017.

60. Bui, V.-H., A. Hussain and H.-M. Kim, "Double Deep Q-learning-Based Distributed Operation of Battery Energy Storage System Considering Uncertainties", *IEEE Transactions on Smart Grid*, Vol. 11, No. 1, pp. 457–469, 2019.

61. Yang, X., Y. Wang, H. He, C. Sun and Y. Zhang, "Deep Reinforcement Learning for Economic Energy Scheduling in Data Center Microgrids", *IEEE Power & Energy Society General Meeting (PESGM)*, pp. 1–5, Atlanta, Georgia, 2019.

62. Cochran, J., O. Zinaman, J. Logan and D. Arent, *Exploring the Potential Business Case for Synergies Between Natural Gas and Renewable Energy*, Tech. rep., National Renewable Energy Laboratory, 2014.

63. Safari, A., N. Das, O. Langhelle, J. Roy and M. Assadi, "Natural Gas: A Transition Fuel for Sustainable Energy System Transformation?", *Energy Science & Engineering*, Vol. 7, No. 4, pp. 1075–1094, 2019.

64. Zhang, B., W. Hu, J. Li, D. Cao, R. Huang, Q. Huang, Z. Chen and F. Blaabjerg, "Dynamic Energy Conversion and Management Strategy for an Integrated Electricity and Natural Gas System with Renewable Energy: Deep Reinforcement Learning Approach", *Energy Conversion and Management*, Vol. 220, p. 113063, 2020.

65. Wen, Z., D. O'Neill and H. Maei, "Optimal Demand Response Using Device-Based Reinforcement Learning", *IEEE Transactions on Smart Grid*, Vol. 6, No. 5, pp. 2312–2324, 2015.

66. Cao, D., W. Hu, X. Xu, T. Dragičević, Q. Huang, Z. Liu, Z. Chen and F. Blaabjerg, "Bidding Strategy for Trading Wind Energy and Purchasing Reserve of Wind Power Producer – A DRL Based Approach", *International Journal of Electrical Power & Energy Systems*, Vol. 117, p. 105648, 2020.

67. Zhu, Z., Z. Hu, K. W. Chan, S. Bu, B. Zhou and S. Xia, "Reinforcement Learning in Deregulated Energy Market: A Comprehensive Review", *Applied Energy*, Vol. 329, p. 120212, 2023.

68. Fuller, D. B., V. J. M. F. Filho and E. F. de Arruda, "Oil Industry Value Chain Simulation with Learning Agents", *Computers & Chemical Engineering*, Vol. 111, pp. 199–209, 2018.

69. Xin, G.-j., K. Zhang, Z.-z. Wang, Z.-f. Sun, L.-m. Zhang, P.-y. Liu, Y.-f. Yang, H. Sun and J. Yao, "Soft Actor-Critic Based Deep Reinforcement Learning Method for Production Optimization", *Proceedings of the International Field Exploration and Development Conference*, Vol. 10, pp. 353–366, Singapore, 2024.

70. Sircar, A., K. Yadav, K. Rayavarapu, N. Bist and H. Oza, "Application of Machine Learning and Artificial Intelligence in Oil and Gas Industry", *Petroleum Research*, Vol. 6, No. 4, pp. 379–391, 2021.

71. Oh, D. H., D. Adams, N. D. Vo, D. Q. Gbadago, C. H. Lee and M. Oh, "Actor-Critic Reinforcement Learning to Estimate the Optimal Operating Conditions of the Hydrocracking Process", *Computers & Chemical Engineering*, Vol. 149, p. 107280, 2021.

72. Aissani, N., B. Beldjilali and D. Trentesaux, "Dynamic Scheduling of Maintenance Tasks in the Petroleum Industry: A Reinforcement Approach", *Engineering Applications of Artificial Intelligence*, Vol. 22, No. 7, pp. 1089–1103, 2009.

73. Wari, E., W. Zhu and G. Lim, "Maintenance in the Downstream Petroleum Industry: A Review on Methodology and Implementation", *Computers & Chemical Engineering*, Vol. 172, p. 108177, 2023.

74. Tu, R., H. Zhang, B. Xu, X. Huang, Y. Che, J. Du, C. Wang, R. Qiu and Y. Liang, "Machine Learning Application in Batch Scheduling for Multi-Product Pipelines: A Review", *Journal of Pipeline Science and Engineering*, Vol. 4, No. 3, p. 100180, 2024.

75. Lama, R. and J. Bodziony, "Management of Outburst in Underground Coal Mines", *International Journal of Coal Geology*, Vol. 35, No. 1, pp. 83–115, 1998.

76. Duzgun, H. and H. Einstein, "Assessment and Management of Roof Fall Risks in Underground Coal Mines", *Safety Science*, Vol. 42, No. 1, pp. 23–41, 2004.

77. Qingjie, Q. I., S. U. N. Zuo, L. I. U. Wengang, W. Anhu, Y. Jinghu, L. I. U. Siyun, S. U. N. Lifeng and W. Wei, "Study on Risk Assessment Model of Coal Mine Water Accident Induced by Flood Disaster", *Coal Science and Technology*, Vol. 51, No. 1, pp. 395–402, 2023.

78. Xiao-hu, Z., Z. Ke-ke, W. Qing-qing and M. Fang-qing, "Research and Application of Reinforcement Learning Based on Constraint MDP in Coal Mine", *WRI World Congress on Computer Science and Information Engineering*, Vol. 4, pp. 687–691,

Los Angeles, California, 2009.

79. Haarnoja, T., A. Zhou, P. Abbeel and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 1861–1870, Stockholm, Sweden, 2018.

80. Haarnoja, T., H. Tang, P. Abbeel and S. Levine, "Reinforcement Learning with Deep Energy-Based Policies", *Proceedings of the 34th International Conference on Machine Learning*, pp. 1352–1361, Sydney, NSW, Australia, 2017.

81. Ziebart, B. D., *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*, Ph.D. Thesis, Carnegie Mellon University, 2010.

82. Hasselt, H., "Double Q-learning", *Advances in Neural Information Processing Systems*, Vol. 23, p. 2613–2621, Vancouver, British Columbia, Canada, 2010.

83. Fujimoto, S., H. van Hoof and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods", *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 1587–1596, Stockholm, Sweden, 2018.

84. Eyeland, A. and K. Wolyniec, *Energy and Power Risk Management*, John Wiley & Sons, Hoboken, New Jersey, 2003.

85. Mahoney, D., *Modeling and Valuation of Energy Structures: Analytics, Econometrics, and Numerics*, Palgrave Macmillan, London, UK, 2016.

86. Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger, "Deep Reinforcement Learning That Matters", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 1, pp. 3207–3214, New Orleans, Louisiana, 2018.

87. Akiba, T., S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-

Generation Hyperparameter Optimization Framework", *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, Anchorage, Alaska, 2019.

88. Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym", ArXiv:1606.01540 [cs.LG], 2016.

89. Towers, M., A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan and O. G. Younis, "Gymnasium: A Standard Interface for Reinforcement Learning Environments", ArXiv:2407.17032 [cs.LG], 2024.

90. Engstrom, L., A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph and A. Madry, "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO", ArXiv:2005.12729 [cs.LG], 2020.

91. Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations", *Journal of Machine Learning Research*, Vol. 22, No. 268, pp. 1–8, 2021.

92. Schulman, J., F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms", ArXiv:1707.06347 [cs.LG], 2017.

93. OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning", ArXiv:1912.06680 [cs.LG], 2019.

94. Zhang, C., O. Vinyals, R. Munos and S. Bengio, "A Study on Overfitting in Deep Reinforcement Learning", ArXiv:1804.06893 [cs.LG], 2018.