Deep Hierarchical Reinforcement Learning

by

Karush Suri

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Electrical & Computer Engineering
University of Toronto

Deep Hierarchical Reinforcement Learning

Karush Suri
Master of Applied Science
Graduate Department of Electrical & Computer Engineering
University of Toronto
2021

Abstract

The biological paradigm of learning by trial and error has motivated tremendous success in the field of Machine Learning. Reinforcement Learning (RL) has emerged as one of the essential learning mechanisms for complex tasks involving sequential decision-making. While RL rests behind a myriad of breakthroughs, its practical application to real-world scenarios remains an open question. Alternative learning mechanisms such as supervised learning readily extend to practical settings. RL, on the other hand, remains dormant from the application perspective. Primary reasons behind this limitation are threefold, (1) restricted scalability of agents to high-dimensional action spaces, (2) reduced robustness towards abrupt dynamics and (3) limited extension of RL algorithms to large-scale scenarios in light of practical considerations.

This thesis addresses the aforesaid three challenges through the lens of hierarchies which serve as suitable abstractions of composit behavior. Novel evolutionary RL methods present a continuously evolving hierarchy which provisions dominant skill transfer among members of its population. Evolutionary agents demonstrate improved scalability to high-dimensional action spaces and efficacious use of hardware resources. Novel energy-based RL schemes, on the other hand, minimize surprise utilizing low energy configurations among members of the multi-agent hierarchy. This presents improved robustness to abrupt transition dynamics and suitable collaboration among agents. The

framework of energy-based surprise minimization steers practical application of hierarchical RL to the setting of trade execution. A novel bi-level hierarchy abstracts order and bid operations while minimizing surprise on large-scale trading scenarios. The end result of this study is a hierarchical scheme demonstrating trade patterns analogous to human execution with this thesis serving as a motivation for application of RL to practical problems.

Acknowledgments

Behind every new idea is directed passion. I would like to thank my advisors Yuri Lawryshyn and Konstantinos Plataniotis for directing my passion towards novel ideas and practical insights. In addition to their knowledge, I have greatly benefited from their encouragement and higher research standards. This thesis would not have been possible without the support of these amazing mentors. I am extremely grateful to Yuri for providing me the opportunity to work and collaborate with the RBC Capital Markets team.

I would also like to extend my gratitude to Xiao Qi (David) Shi for providing his support and insights throughout my research. David's cheerful yet practical thinking allowed me to become a better researcher and realize the importance of Reinforcement Learning as an emerging field. My appreciation goes out to the entire RBC Capital Markets team and Gavin Ding from Borealis AI for providing valuable feedback on my work and helping me in my research journey. Along similar lines, I appreciate Kory Fong and Gabriel Iszlai from RBC Innovation Lab for equipping me with the tools and resources required for undertaking industrial research.

My time at the University of Toronto was filled with enriching experiences as part of the Center for Management of Technology & Entrepreneurship (CMTE) group. My gratitude goes out to the entire CMTE staff for providing me the support and care in completing my graduate studies. I would like to acknowledge David Kivlichan for being an accompanying colleague and providing valuable insights and feedback during our group tutorial sessions. My gratitude goes out to the Department of Computer Science (DCS) for providing computational resources in order to carry out large-scale experiments.

I owe a debt of gratitude to my peers Aravind Varier and Shashank Saurav who accompanied me in my graduate studies and made Toronto exciting for me. I am also grateful to Divyam Sharma, Shreya Kohli, Mohit Mohta, Aayush Sharma, Devashish Garg and Ashi Agarwal for their constant support and appreciation for my work.

Lastly, I would like to thank my parents for their unwavering suport and compassion while providing me the opportunity to move to Toronto.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Humans learn from past experiences and demonstrate the ability to retain gained knowledge over time. This biological paradigm of learning by trial and error has motivated tremendous success in the field of Machine Learning. Reinforcement Learning (RL) has emerged as one of the essential learning mechanisms for complex tasks involving sequential decision-making. This emergence has given rise to numerous breakthroughs surpassing human-level intelligence [68, 94], consequently enhancing the frontiers of science and technology.

While RL has presented success in a myriad of paradigms, its practical application to real-world tasks remains an open question. For instance, extension of RL to cost-sensitive and safety-critical domains such as autonomous driving and healthcare present an unforesable future. While alternative learning mechanisms such as supervised learning and probabilistic reasoning are suitable candidates for majority of our daily needs, RL remains a quite corner from the application perspective.

Primary reasons behind this limitation of trial-and-error learning framework are lack of data efficiency, scalability and robustness to quick changes in the world of humans. Having a closer look at the desidarata of practical learning schemes narrows down the problem to three key components, (1) Scalability to higher dimensions, (2) Robustness to fast-paced changes and (3) Extension of algorithms to novel and large-scale scenarios.

Scalability in the face of uncertainty: Most machine learning systems fall prey to the curse of dimensionality. RL is no different than these systems. Increasing degrees of freedom as controllable parameters of the learning agent prohibit efficient learning. RL agents face the growing challenge of an ever-increasing stream of data whose samples, unlike supervised learning, do not contain labelled optimal information. This presents RL agents with a twofold problem, to efficiently interpret incoming data, and simultaneously utilize past knowledge for optimal behavior. This hinders scalability of an

agent to higher dimensions. Additionally, the problem may incorporate a third challenge wherein the number of agents are more than one.

Robustness to fast-paced changes: Various real-world scenarios move at different timescales and require variable levels of knowledge from humans. Similarly, an agent's environment may change drastically over the course of a few timesteps which require faster learning of dynamics. This non-stationary nature of the environment presents the agent with the necessity to become robust towards abrupt transitions in dynamics. Various existing methods aim to address this challenge using a predefined model which is not readily available to the agent in case of novel scenarios.

Practical extensions to novel scenarios: Perhaps the most critical conundrum faced by RL research is its extension towards new domains. Learning algorithms demonstrate ample success on video games [68, 115] and board games [94, 90] but fall short of optimality on practical tasks [15]. This arises as a direct of consequence of lack of RL applicability to large-scale problems presenting scalability and robustness challenges. Suitable solutions would require addressing the above two challenges in light of a large-scale novel setting.

This thesis takes a closer look at the aforesaid three challenges in light of practical applications of RL. The framework of learning is modelled through the lens of hierarchies which are a collection of stages of operation. Analogous to human behavior, operation of RL algorithms may be broken down into smaller skills or subset of abilities. For instance, a *child* first learns to *stand* and gradually *walk* in order to achieve his final goal to *run*. Akin to this example, this thesis aims to explore low-level behaviors in RL agents which provide them with the ability to complete nontrivial tasks by utilizing higher levels of the hierarchy. Following is a brief outline of the methods presented here-

- Chapter 2 explores *Evolutionary Reinforcement Learning* which has recently presented suitable alternatives for the scalability of RL to high-dimensional control problems. A population of RL agents is realized as a continuously evolving hierarchy over generations. This evolutionary perspective of RL as a hierarchy yields intriguing results across members of the population. The end result of Chapter 2 is a scalable RL algorithm which efficaciously extends to large action spaces.

- Chapter 3 explores *Energy-based Reinforcement Learning* which aims to address a hierarchy's robustness towards fast-paced dynamics. The presented algorithm utilizes an energy-based objective which aids in minimization of *surprise* arising from abrupt transitions across multiple agents. This leads to suitable convergence of agents towards surprise-robust behavior.

- Chapter 4 solidifies ideas presented in previous formulations by adopting them in a *Practical Deep Hierarchical Reinforcement Learning* setting. A hierarchical framework introduced in this chapter aims to tackle abrupt dynamics utilizing energy-based surprise minimization for the novel task of executing trades in the stock market. The end result is a scheme which provides proof-of-concept of application of RL to practical scenarios.

1.2 Evolutionary Reinforcement Learning

Concepts and applications of Reinforcement Learning (RL) have seen a tremendous growth over the past decade [68]. These consist of applications in arcade games [68], board games [94] and lately, robot control tasks [57]. A primary reason for this growth is the usage of computationally efficient function approximators such as neural networks [48]. Modern-day RL algorithms make use of parallelization to reduce training times [67] and boost agent's performance through effective exploration giving rise to scalable methods [42, 31, 119]. However, a number of open problems such as approximation bias, lack of scalability in the case of long time horizons and lack of diverse exploration restrict the application of scalability to complex control tasks.

State-of-the-art RL algorithms such as Soft Actor-Critic (SAC) [30] maximize entropy which is indicative of continued exploration. However, using a computationally expensive framework limits scalability as it increases the number of gradient-based [84] updates of the overall algorithm. Moreover, tasks consisting of long time horizons have higher computational overhead as a result of long trajectory lengths. For instance, obtaining accurate position estimates [42] over longer horizons require additional computation times which varies *linearly* with the hardware requirement. Such a variation calls for increased scalability in the RL domain.

Diverse exploration strategies are essential for the agent to navigate its way in the environment and comprehend intricate aspects of less visited states[64]. Various modern-day RL methods lack significant exploration [67, 68] which is addressed by making use of meta-controller[49] and curiosity-driven [11] strategies at the cost of sample efficiency and scalability.

Recent advances in RL have leveraged evolutionary computing for effective exploration and scalability [86, 38, 43, 97, 65]. These methods often fall short of optimal performance and depict sensitivity towards their hyperparameters. A common alternative for improving performance is to combine gradient-based objectives with evolutionary methods [43]. These algorithms allow a population of learners to gain dominant skills [83] from modern-day RL methods and depict robust control while demonstrating scalability. However, their applications do not extend to high-dimensional tasks as a result of sensitivity to mutational hyperparameters which still remains an open problem.

Chapter 2 introduces the Evolution-based Soft Actor Critic (ESAC), an algo-

rithm combining ES with SAC for state-of-the-performance equivalent to SAC and scalability comparable to ES. The contributions of Chapter 2 are threefold; (1) ESAC abstracts exploration from exploitation by exploring policies in *weight space* using evolutions and exploiting gradient-based knowledge using the SAC framework. (2) ESAC makes use of soft winner selection function which, unlike prior selection criteria [43], does not shield winners from mutation. ESAC carries out genetic crossovers in hindsight resulting in dominant skill transfer between members of the population. Lastly, (3) ESAC introduces the novel Automatic Mutation Tuning (AMT) which maximizes the mutation rate of ES in a small clipped region and provides significant hyperparameter robustness without making use of backpropagation updates.

1.3 Energy-based Reinforcement Learning

Besides arcade games [68], board games [94, 90] and robot control tasks [57, 92] RL has lately seen strides in real-time games [115]. The rise of RL has led to an increasing interest in the study of multi-agent systems [58, 114], commonly known as Multi-Agent Reinforcement Learning (MARL). In the case of partially observable settings, MARL enables the learning of policies with centralised training and decentralised control [47]. This has proven to be useful for exploiting value-based methods which are otherwise deamed to be sample-inefficient [104, 20].

Value Factorization [99, 80] is a common technique which enables the joint value function to be represented as a combination of individual value functions conditioned on states and actions. In the case of Value Decomposition Network (VDN) [99], a linear additive factorization is carried out whereas QMIX [80] generalizes the factorization to a non-linear combination, hence improving the expressive power of centralised action-value functions. Furthermore, monotonicity constraints in QMIX enable scalability in the number of agents. On the other hand, factorization across multiple value functions leads to the aggregation of approximation bias [34, 36] originating from overoptimistic estimations in action values [23, 50]. Moreover, value factorization methods yield state-based estimates and do not account for spurious changes in partially-observed observations, commonly referred to as surprise [1].

Surprise minimization [8] is a recent phenomenon observed in the case of single-agent RL methods which deals with environments consisting of spurious states. In the case of model-based RL [41], surprise minimization is used as an effective planning tool in the agent's model [8] whereas in the case of model-free RL, surprise minimization is witnessed as an intrinsic motivation [1, 61] or generalization problem [16]. On the other hand, utilization of surprise minimization towards MARL has been limited as a result of which agents remain unaware of drastic changes in the environment [60]. Thus, surprise minimization in the case of multi-agent settings requires attention from a

critical standpoint.

Chapter 3 introduces the Energy-based MIXer (EMIX), an algorithm based on QMIX which minimizes surprise utilizing the energy across agents. The contributions of Chapter 3 are threefold; (1) EMIX introduces a novel surprise minimization technique across multiple agents in the case of multi-agent partially-observable settings. (2) EMIX highlights a practical use of energy functions in MARL with theoretical guarantees and experiment validations of the energy operator. Lastly, (3) EMIX extends Maxmin Q-learning [50] addressing overestimation bias across agents in MARL. When evaluated on a range of challenging StarCraft II scenarios [89], EMIX demonstrates consistently stable performance for multi-agent surprise minimization by improving the QMIX framework. Moroever, the ablation study highlights a necessity of energy-based scheme and the need for elimination of overestimation bias in MARL.

1.4 Hierarchical Reinforcement Learning for Trade Execution

RL has demonstrated profound success in applications such as video games [68], board games [94] and robot control [92, 57]. Primary reason for increasing developments in RL is the provision of deep neural networks as efficient function approximators. Combination of non-linear approximations of action distributions in the agent-environment paradigm allows for perception of large state spaces which otherwise hinder scalability of the control framework. While applications of RL span a wide variety of tasks which serve as test benchmarks for real world problems, RL is seldomly used for practical applications [7].

Consider the problem of autonomous driving wherein a system autonomously operates a vehicle in real-time. Formulating this as a reinforcement learning problem may be carried out by treating the system as *agent* and a simulator as *environment*. At each step, the system observes pixel inputs as *states* from the simulator and outputs steering, acceleration and brake commands as *actions*. The simulator consists of a reward function which depends on the distance covered by the vehicle and its speed under different traffic conditions. Additionally, the reward consists of a penalty if the system collides with any obstacles in the simulator, for instance another vehicle or pedestrians. The system learns to optimize the reward function by maintaining constant speed and covering more distance in different traffic conditions. Whenever the vehicle approaches an obstacle, the system acts to avoid it by utilizing the brake and directional steering commands. Such behaviors acquired by the system during training denote promise for practical adaptation.

However, the framework cannot be adapted in real-world traffic for two main reasons. Firstly, the system would learn from trial and error leading to frequent mistakes causing damage to the vehicle or obstacles. In general, the *agent*

cannot carry out *catastrophy minimization* quickly and efficiently. Secondly, irrespective of obstacle penalty in the reward function, the system may abruptly encounter an obstacle (a child running to cross the street) resulting in collision. Even though the *agent* tries to act optimally (pushing the brakes), it fails to evade a collision as a result of the fast-paced dynamics. As a result, the *agent* cannot carry out *surprise minimization* betweeen consecutive states.

While various open problems such as data efficiency [55] and estimation bias [32, 112] hinder an agent's performance, *catastrophy* and *surprise minimization* hinder agent's scalability to real world scenarios. Catastrophic and surprising states scale linearly with the state space of the environment making it increasingly challenging for the agent to quickly switch between actions. Furthermore, the occurence of such abrupt transitions is rarely encountered during the training phase resulting in a lack of informative data collection. This highlights the necessity to approach RL from a realistic viewpoint.

In order to address the aforementioned challenges from a practical standpoint, we turn our attention to the more challenging scenario of trading. The *agent* (trader) interacts in the *environment* (market) to maximize its *reward* (profit). Chapter 4 presents Trade Execution using Reinforcement Learning (TradeR), a hierarchical framework for executing trade bids on abrupt real market experiences during the COVID19 stock market crash. TradeR fulfills *catastrophy minimization* by providing the agent with order and bid policies capable of estimating order quantities and executing bid actions respectively. The order policy determines the quantity for the bid. The high-level bid policy, utilizes the estimated quantity in conjunction with current states to execute actions as trade bids. TradeR further tackles *surprise minimization* by generalizing an energy-based scheme [101] comprising of a surprise value function. The value function estimates surprise correponding to a given state which is inferred by the agent as intrinsic motivation. In a high-frequency and large-scale study of 35 stock symbols from the S&P500 index for 2019 fiscal year, TradeR demonstrates improved robustness to abrupt changes with consistent profitable outcomes.

1.5 Background

Reinforcement Learning: This sub-section provides the reader with the background and notation for Chapter 2 and Chapter 4. We review the RL setup wherein an agent interacts with the environment in order to transition to new states and observe rewards by following a sequence of actions. The problem is modeled as a finite-horizon Markov Decision Process(MDP) [103] defined by the tuple $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$ where the state space is denoted by $\mathcal{S}$ and action space by $\mathcal{A}$, r presents the reward observed by agent such that $r : \mathcal{S} \times \mathcal{A} \to [r_{min}, r_{max}]$, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ presents the unknown transition model consisting of the transition probability to the next state $s_{t+1} \in \mathcal{S}$ given the current state

$s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ at time step t and γ is the discount factor. A policy $\pi_\theta(a_t|s_t)$ is a function of model parameters θ. Standard RL defines the agent's objective to maximize the expected discounted reward $\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$ as a function of the parameters θ. SAC [30] defines an entropy-based[124] objective expressed in Equation 1.1.

$$J(\pi_\theta) = \sum_{t=0}^{T} \gamma^t [r(s_t, a_t) + \lambda \mathcal{H}(\pi_\theta(\cdot|s_t))] \tag{1.1}$$

wherein λ is the temperature coefficient and $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy exhibited by the policy $\pi(\cdot|s_t)$ in s_t. For a fixed policy, the soft Q-value function can be computed iteratively, starting from any function $Q : \mathcal{S} \times \mathcal{A}$ and repeatedly applying a modified Bellman backup operator $\mathcal{T}^\pi$ given by Equation 1.2

$$\mathcal{T}^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P}[V(s_{t+1})] \tag{1.2}$$

where $V(s_t)$ is the soft state value function expressed in Equation 1.3.

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log(\pi(a_t|s_t))] \tag{1.3}$$

The setup considers a parameterized state value function $V_\psi(s_t)$, a soft Q-function $Q_\phi(s_t, a_t)$ and a policy $\pi_\theta(a_t|s_t)$ which can be represented with nonlinear function approximators such as neural networks with ψ, ϕ and θ being the parameters of these networks.

Evolution Strategies: This sub-section continues to build on the prior knowledge for Chapter 2. We review the Evolution Strategies [86] framework which is motivated by natural evolution. ES is a heuristic search procedure in which a population of offsprings is mutated using random perturbations. Upon mutation, the fitness objective corresponding to each member of the population is evaluated and offsprings with greater scores are recombined to form the population for the next generation. Let n be the number of offsprings in the population. The parameter vectors of the model can then be represented as $\theta_{es,(i)}$ such that $i = 1, 2, ..n$. A total of n random perturbations $\epsilon_{(i)}$, $i = 1, 2, ..n$ are sampled from a Gaussian distribution $\mathcal{N}(0, 1)$ in order to mutate $\theta_{es,(i)}$ and evaluate the fitness objective $\mathbb{E}_{\epsilon_{(i)} \sim \mathcal{N}(0,1)}[O(\theta_{es,(i)} + \sigma\epsilon_{(i)})] = \frac{1}{\sigma}\mathbb{E}_{\epsilon_{(i)} \sim \mathcal{N}(0,1)}[O(\theta_{es,(i)} + \sigma\epsilon_{(i)})\epsilon_{(i)}]$. Here, σ is the mutation rate which controls the extent of mutation. In the case of RL, the fitness objective $O(\theta_{es,(i)})$ is the episodic reward observed by members of the population.

Multi-Agent Learning: This subs-section emphasizes on the background and notation for Chapter 3. We review the cooperative MARL setup which is in departure from the notation of conventional RL formulation used in Chapter 2 and Chapter 4. The problem is modeled as a Partially Observable Markov

Decision Process (POMDP) [103] defined by the tuple $(\mathcal{S}, \mathcal{A}, r, N, P, Z, O, \gamma)$ where the state space $\mathcal{S}$ and action space $\mathcal{A}$ are discrete, $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$ presents the reward observed by agents $a \in N$ where N is the set of all agents, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ presents the unknown transition model consisting of the transition probability to the next state $s' \in \mathcal{S}$ given the current state $s \in \mathcal{S}$ and joint action $u \in \mathcal{A}$ at time step t and γ is the discount factor. The setup considers a partially observable setting in which each agent n draws individual observations $z \in Z$ according to the observation function $O(s, u) : \mathcal{S} \times \mathcal{A} \rightarrow Z$. A joint policy $\pi_\theta(u|s)$ is a function of model parameters θ. Standard RL defines the agent's objective to maximize the expected discounted reward $\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t, u_t)]$ as a function of the parameters θ. The action-value function for an agent is represented as $Q(u, s; \theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \gamma^t r(s, u)|s = s_t, u = u_t]$ which is the expected sum of payoffs obtained in state s upon performing action u by following the policy π_θ. The optimal policy is denoted by π_θ^* such that $Q(u, s; \theta^*) \geq Q(u, s; \theta) \forall s \in S, u \in A$. In the case of multiple agents, the joint optimal policy can be expressed as the Nash Equilibrium [74] of the Stochastic Markov Game as $\pi^* = (\pi^{1,*}, \pi^{2,*}, ...\pi^{N,*})$ such that $Q(u^a, s; \theta^*) \geq Q(u^a, s; \theta) \forall s \in S, u \in A, a \in N$.

Q-Learning is an off-policy, model-free algorithm suitable for continuous and episodic tasks. The algorithm uses semi-gradient descent to minimize the Temporal Difference (TD) error: $\mathbb{L}(\theta) = \mathop{\mathbb{E}}_{b \sim R}[(y - Q(u, s; \theta))^2]$ where $y = r + \gamma \max_{u' \in A} Q(u', s'; \theta^-)$ is the TD target consisting of θ^- as the target parameters and b is the batch sampled from memory R.

Chapter 2

Evolutionary Reinforcement Learning

2.1 Overview

Advances in Reinforcement Learning (RL) have demonstrated data efficiency and optimal control over large state spaces at the cost of scalable performance. Genetic methods, on the other hand, provide scalability but depict hyperparameter sensitivity towards evolutionary operations. However, a combination of the two methods has recently demonstrated success in scaling RL agents to high-dimensional action spaces. Parallel to recent developments, this chapter presents the Evolution-based Soft Actor-Critic (ESAC), a scalable RL algorithm. ESAC abstracts exploration from exploitation by combining Evolution Strategies (ES) with Soft Actor-Critic (SAC). Through this lens, the method enables dominant skill transfer between offsprings by making use of soft winner selections and genetic crossovers in hindsight and simultaneously improve hyperparameter sensitivity in evolutions using the novel Automatic Mutation Tuning (AMT). AMT gradually replaces the entropy framework of SAC allowing the population to succeed at the task while *acting as randomly as possible*, without making use of backpropagation updates. In a study of challenging locomotion tasks consisting of high-dimensional action spaces and sparse rewards, ESAC demonstrates improved performance and sample efficiency in comparison to the Maximum Entropy framework. Additionally, ESAC presents efficacious use of hardware resources and algorithm overhead. A complete implementation of ESAC can be found at this link.

2.2 Related Work

Scalable Reinforcement Learning: Recent advances in RL have been successful in tackling sample-efficiency [30] and approximation bias (also known as overestimation bias) which stems from value of estimates approximated by

the function approximator. Overestimation bias is a common phenomenon occurring in value-based methods [35, 33, 50] and can be addressed by making use of multiple critics in [23] in the actor-critic framework [67]. This in turn limits scalability of algorithms [42] by increasing the number of gradient-based updates. Moreover, memory complexity of efficient RL methods increases linearly with the expressive power of approximators [44], which in turn hinders scalability of RL to complex control tasks.

Evolutionary Reinforcement Learning: Intersection of RL and Evolutionary methods has for long been studied in literature [38, 97, 65, 70, 77, 78]. [86] presents the large-scale parallelizable nature of Evolution Strategies (ES). Performance of ES on continuous robot control tasks in comparison to various gradient-based frameworks such as Trust Region Policy Optimization (TRPO) [93] and Proximal Policy Optimization (PPO) [92] have been found comparable. On the other hand, ES falls short of competitive performance resulting in local convergence and is extremely sensitive to mutation hyperparameters.

An alternative to a pure evolution-based approach is a suitable combination of an evolutionary algorithm with a gradient-based method [40], commonly referred to as Evolutionary Reinforcement Learning (ERL) [43]. ERL makes use of selective mutations and genetic crossovers which allow weak learners of the population to inherit skills from strong learners while exploring. ERL methods are scalable to high-dimensional control problems including multi-agent settings [83]. Such an approach is a suitable trade-off between sample efficiency and scalability but does not necessarily introduce mutation robustness. Other methods in literature [38] follow a similar approach but are often limited to directional control tasks which require little mutation. Thus, addressing scalability and exploration while preserving higher returns and mutation robustness requires attention from a critical standpoint. The work presented in this chapter is parallel to prior efforts made towards this direction.

2.3 A Motivating Example: The Discrete Cyclic MDP

The setup considers a long-horizon discrete cyclic MDP as its motivation for the work. The MDP has a state space $\mathcal{S}^3$ consisting of 3 states- $S0, S1$ and $S2$ and a discrete action space $\mathcal{A}^3$ consisting of 3 actions- clockwise, anticlockwise and stay. The agent starts in state $S0$. The reward function $r : \mathcal{S}^3 \times \mathcal{A}^3$ assigns a reward of $+1$ for moving clockwise and -1 otherwise. Each episode lasts 2000 timesteps and terminates if the agent reaches the end of horizon or incurs a negative reward. Figure 4.1 presents the long-horizon discrete cyclic

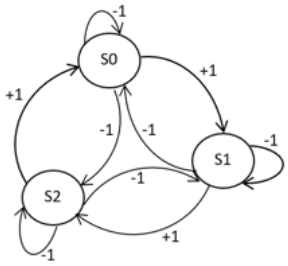

Figure 2.1: The long-horizon discrete Cyclic MDP.

MDP.

The cyclic MDP, being a long-horizon problem, serves as a suitable benchmark for agent's behavior consisting of minimum computational overhead and is a small-scale replication of policy-search for scalable policy-based agents. The environment consists of a global objective which the agents can achieve if they solve the environment by obtaining the maximum reward of +2000. In order to assess evolution-based behavior, the setup compares the performance of a population of 50 offsprings utilizing ES with PPO [92] and Deep Deterministic Policy Gradient (DDPG) [57], an efficient off-policy RL method. Although DDPG is primarily a continuous control algorithm, it is employed as a result of the minimal nature of the problem. Figure 2.2 (left) presents the performance of ES in comparison to gradient-based agents in the cyclic MDP averaged over 3 runs. The ES population presents sample efficiency by solving the task within the first 100 episodes. DDPG, on the other hand, starts solving the task much latter during training. The use of a deterministic policy allows DDPG to continuously move left whereas in the case of ES, the population carries out exploration in the weight space and moves along the direction of the strong learners. Lastly, PPO finds a local solution and does not succeed at solving the task. Driven by clipped updates, PPO restricts the search horizon in policy space leading to a sub-optimal policy.

ES has proven to be scalable to large-scale and high dimensional control tasks [86]. The setup assesses this property of ES in the cyclic MDP by varying the operational hardware (number of CPUs) [19] and algorithm overhead (population size). The setting measures the average wall-clock time per episode [5]. As shown in Figure 2.2 (center), ES is parallelizable in nature and can be scaled up to larger population sizes by reducing the computation time. The large-scale readily parallelizable nature of ES is a convincing characteristic for utilizing CPU-based hardware. However, ES relies on excessively sensitive hyperparameters such as mutation rate. Figure 2.2 (right) presents the sensitivity of ES to mutation rate within a small range with a constant population size of 50. Varying population size does not present a trend in sensitivity indicating that mutation rate is the dominant hyperparameter governing policy behavior among offsprings. Hyperparameter sensitivity requires attention in the case of RL applications such as for real-world continuous control [63]. These include excessive tuning of parameters and detailed ablation studies. The cyclic MDP highlights this sensitive nature of ES and serves as a motivating example for tackling sensitivity while preserving optimal performance in a scalable manner.

2.4 Evolution-based Continuous Control

The motivation behind ESAC stems from translating the scalability and tackling the mutation sensitivity of ES observed in discrete cyclic MDP to continuous

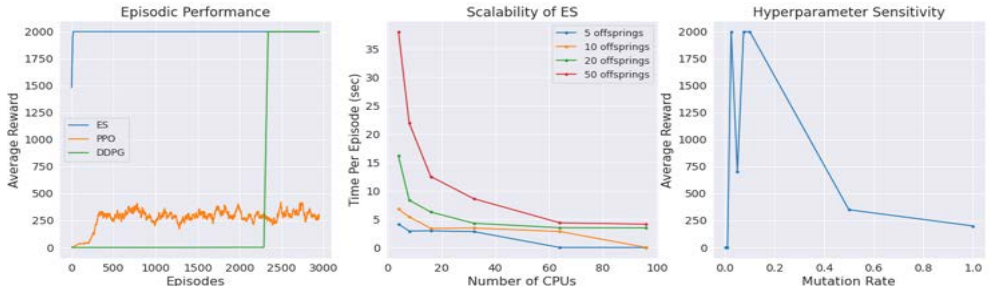

Figure 2.2: Comparison of ES, DDPG and PPO in the discrete cyclic MDP.

control tasks. ESAC combines the scalable nature of ES with the limited approximation bias of SAC to yield a CPU-friendly state-of-the-art equivalent algorithm.

2.4.1 Evolution-based Soft Actor Critic

Overview: Figure 2.3 provides a high-level schematic of the ESAC algorithm and its components. The population is evaluated in the environment with the fitness metric as episodic rewards obtained by each offspring. Top w winners are then segregated for mutation consisting of ES update followed by crossovers between perturbed offsprings and winners. The new population is formed using crossed-over offsprings and SAC agent. The SAC agent executes its own episodes at fixed timesteps and stores these experiences in a dedicated replay-buffer following policy update. During the SAC update timesteps, ESAC utilizes AMT which maximizes the mutation rate in a clipped region. SAC update timesteps are exponentially annealed to reduce entropy noise and abstract exploration in weight space.

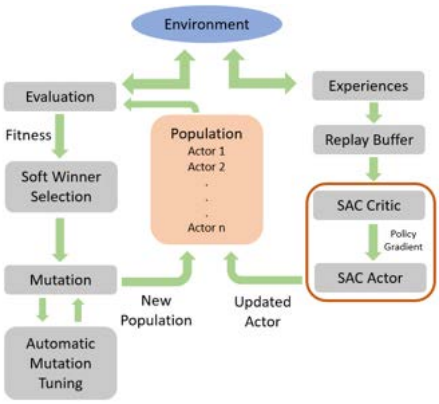

Figure 2.3: Workflow of ESAC combining ES with SAC.

Algorithm: Algorithm 3 presents the ESAC algorithm. The procedure begins by initializing ψ, θ, θ_{es}, ϕ being the parameters of state-value function, SAC policy, ES policy and Q-function respectively. We then initialize learning rate for SAC agent α, learning rate of ES population α_{es}, mutation rate σ, p_{sac} which is the probability of SAC updates, $\bar{\psi}$ is the parameter vector of the target value function, ζ is the clip parameter, τ is the target smoothing coefficient, e is the fraction of winners and g is the gradient interval. A population of n actors pop_n is initialized along with an empty replay buffer R. Following the main loop,

for each offspring i in the population, the method draws a noise vector ϵ_i from $\mathcal{N}(0,1)$ and perturbs the ES policy vector θ_{es} to yield the perturbed parameter vector $\theta_{es,(i)}$ as per the expression $\theta_{es,(i)} + \sigma\epsilon_{(i)}$. $\theta_{es,(i)}$ is then evaluated to yield the fitness $F_{(i)}$ as episodic rewards. These are collected in a normalized and ranked set F. The procedure now executes soft winner selection wherein the first $w = (n * e)$ offsprings from F are selected for crossovers by forming the set W. The soft winner selection allows dominant skill transfer between winners and next generation offsprings. Mutation is carried out using the ES update [86]. SAC gradient updates are executed at selective gradient intervals g during the training process. p_{sac} is exponentially annealed to reduce entropy noise and direct exploration in the weight space. During each g, the agent executes its own episodes by sampling $a_t \sim \pi_\theta(a_t|s_t)$, observing $r(s_t|a_t)$ and s_{t+1} and storing these experiences in R as a tuple $(s_t, a_t, r(s_t, a_t), s_{t+1})$. Following the collection of experiences, the method updates the parameter vectors ψ, ϕ and θ by computing $\nabla_\psi J_V(\psi)$, $\nabla_\phi J_Q(\phi_{(i)})$ and $\nabla_\theta J_\pi(\theta)$ where $J_V(\psi)$, $J_Q(\phi_{(i)})$ and $J_\pi(\theta)$ are the objectives of the state-value function, each of the two Q-functions $i \in \{1, 2\}$ and policy as presented in [30] respectively. Gradient updates are followed by AMT update (subsection 2.4.2) which leads to hindsight crossovers between winners in W and ES policy parameter vector θ_{es}. Crossovers are carried out as random replacements between elements of weight vectors. In the case of hindsight crossovers, replacements between weight vector elements of current & immediate previous generations is carried out. This allows the generation to preserve traits of dominant offsprings in hindsight. Finally, the new population is formed using θ, θ_{es} and W.

2.4.2 Automatic Mutation Tuning (AMT)

Maximization of randomness in the policy space is akin to maximization in the weight space as both formulations are a multi-step replica of generalized policy improvement algorithm. This allows one to leverage the more suitable weight space for parallel computations. Policy updates during execution of offsprings require tuning the exploration scheme. To this end, we automatically tune σ with the intial value $\sigma_{(1)}$. σ is updated at fixed timesteps in a gradient-ascent manner without making use of backpropagation updates. AMT motivates guided exploration towards the objective as a result of the expansion of the search horizon of population which in turn enables the agent to maximize rewards *as randomly as possible*. AMT makes use of the SmoothL1 (Huber) [39] loss function provided in Equation 2.1 and the update rule is mathematically expressed in Equation 2.2.

$$SmoothL1(x_i, y_i) = \begin{cases} 0.5(x_i - y_i)^2, \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, \text{otherwise} \end{cases} \tag{2.1}$$

Algorithm 1 Evolution-based Soft Actor-Critic (ESAC)

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \theta_{es}, \phi$
Initialize α, α_{es}, σ, ζ, τ, e, g, p_{sac}
Initialize a population of n actors pop_n and an empty replay buffer R
for generation=1,∞ **do**
    for $i \in pop_n$ **do**
        sample $\epsilon_{(i)} \sim \mathcal{N}(0, 1)$
        $F_{(i)} \leftarrow$ evaluate $(\theta_{es,(i)} + \sigma\epsilon_{(i)})$ in the environment
    end for
    normalize and rank $F_{(i)} \in F$
    select the first $w = (n * e)$ offsprings from F to form the set of winners W
    set $\theta_{es} \leftarrow \theta_{es} + \frac{\alpha_{es}}{n\sigma} \sum_{i=1}^{n} F_{(i)}\epsilon_{(i)}$
    if $generation\%g == 0$ & $\mu \sim \mathcal{N}(0, 1) < p_{sac}$ **then**
        for each environment step **do**
            $a_t \sim \pi_\theta(a_t|s_t)$
            observe $r(s_t|a_t)$ and $s_{t+1} \sim P$
            $R \leftarrow R \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$
        end for
        for each gradient step **do**
            $\psi \leftarrow \psi - \alpha\nabla_\psi J_V(\psi)$
            $\phi \leftarrow \phi - \alpha\nabla_\phi J_Q(\phi_{(i)})$ for $i \in \{1, 2\}$
            $\theta \leftarrow \theta - \alpha\nabla_\theta J_\pi(\theta)$
            $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\psi$
        end for
        Update σ using Equation 2.3
    end if
    crossover between $\theta_{es,(i)}$ and θ_{es} for $i = 1, 2, ..w$
    Form new population pop_n using θ, θ_{es}, W
end for

$$\sigma_{(t+1)} \leftarrow \sigma_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t)}} SmoothL1(R_{max}, R_{avg}) \tag{2.2}$$

Here, R_{max} is the reward observed by winner offspring and R_{avg} is the mean reward of the population with $\sigma_{(t)}$ and $\sigma_{(t+1)}$ the mutation rates at timesteps t and $t + 1$ respectively. While exploring in weight space, the SmoothL1 loss tends to take up large values. This is indicative of the fact that the deviation between the winner and other learners of the population is significantly high. In order to reduce excessive noise from weight perturbations ϵ_i, the update is clipped in a small region parameterized by the new clip parameter ζ. Suitable values for ζ range between 10^{-6} to 10^{-2}. The clipped update is mathematically expressed in Equation 2.3.

$$\sigma_{(t+1)} \leftarrow \sigma_{(t)} + clip(\frac{\alpha_{es}}{n\sigma_{(t)}} SmoothL1(R_{max}, R_{avg}), 0, \zeta) \tag{2.3}$$

The update can be expanded recursively and written in terms of the initial mutation rate $\sigma_{(1)}$ as expressed in Equation 2.4 (derived in section A.1).

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\hat{\Lambda}} SmoothL1(R_{max,(t)}, R_{avg,(t)}) \tag{2.4}$$

Here, $\hat{\Lambda}$ is defined as the Tuning Multiplier and can be mathematically expressed as in Equation 2.5. The chapter directs the curious reader to section A.1 for a full derivation.

$$\hat{\Lambda} = \prod_{t'=1}^{t-1} (1 + \frac{\alpha_{es}}{n\sigma^2_{(t')}} SmoothL1(R_{max,(t')}, R_{avg,(t')})) \tag{2.5}$$

It can be additionally shown that AMT, when combined with soft winner selection, leads to policy improvement with high probability among the set of winners. The proof is deferred to section A.2.

2.5 Experiments

Domain	Tasks	ESAC	SAC	TD3	PPO	ES
MuJoCo	HalfCheetah-v2	10277.16±403.63	**10985.90±319.56**	7887.32±532.60	1148.54±1455.64	3721.85±371.36
	Humanoid-v2	5426.82±229.24	**5888.55±44.66**	5392.89±363.11	455.09±213.88	751.65±95.64
	Ant-v2	3465.57±337.81	3693.08±708.56	**3951.76±370.00**	822.34±15.76	1197.69±132.01
	Walker2d-v2	**3862.82±49.80**	3642.27±512.59	3714.89±90.35	402.33±27.38	1275.93±243.78
	Swimmer-v2	**345.44±17.89**	31.68±0.41	110.85±23.02	116.96±0.74	254.42±109.91
	Hopper-v2	**3461.63±118.61**	3048.69±467.21	3255.27±184.18	1296.17±1011.95	1205.73±185.25
	LunarLanderContinuous-v2	**285.79±9.60**	66.52±26.75	273.75±4.51	124.47±11.58	74.41±109.69
	Reacher-v2	-2.01±0.07	-0.50±0.05	-5.12±0.17	**-0.21±0.07**	-4.43±2.06
	InvertedDoublePendulum-v2	**9359.35±0.60**	9257.96±86.54	5603.72±3213.51	88.52±4.73	259.39±36.75
DeepMind Control Suite	HumanoidStand	**805.08±135.67**	759.08±125.67	745.15±291.377	8.41±3.33	10.57±0.30
	HumanoidWalk	**883.00±21.97**	843.00±7.97	686.33±56.23	2.20±0.18	10.59±0.34
	HumanoidRun	**358.82±101.12**	341.45±18.14	291.82±2101.12	2.29±0.16	10.55±0.30
	CheetahRun	**773.14±3.00**	227.66±13.07	765.22±27.93	371.70±19.82	368.62±32.87
	WalkerWalk	**971.02±2.87**	175.75±15.51	941.45±27.01	316.54±79.54	308.94±44.36
	FishUpright	914.96±2.04	285.69±21.03	838.32±34.86	561.39±111.59	**997.58±0.26**

Table 2.1: Average returns on 15 locomotion tasks from MuJoCo & DeepMind Control Suite.

Figure 2.4: Robust behavior of ESAC observed on the WalkerWalk task.

Evaluation experiments aim to evaluate performance, sample efficiency, scalability and mutation sensitivity of ESAC. Specifically, the setup aims to answer the following questions-

- How does the algorithm compare to modern-day RL methods for complex tasks?
- How do evolutionary operations impact scalability in the presence of gradients?

- Which components of the method contribute to sensitivity and scalability?

2.5.1 Performance

The experiments assess performance and sample efficiency of ESAC with state-of-the-art RL techniques including on-policy and off-policy algorithms. We compare the method to ES [86]; SAC [30]; Twin-Delayed Deep Deterministic Policy Gradient (TD3)[23] and PPO [92] on a total of 9 MuJoCo [110] and 6 DeepMind Control Suite [105] tasks. The chapter directs the reader to section A.3 for complete results. The tasks considered consist of sparse rewards and high-dimensional action spaces including 4 different versions of Humanoid. Additionally, the evaluation considers the LunarLander continuous task as a result of its narrow basin of learning. All methods were implemented using author-provided implementations except for ES in which Virtual Batch Normalization [87] was omitted as it did not provide significant performance boosts and hindered scalability.

Agents were trained in OpenAI's Gym environments [10] framework for a total of 5 random seeds. Training steps were interleaved with validation over 10 episodes. For all agents the setup uses nonlinear function approximators as neural networks in the form of a multilayer architecture consisting of 2 hidden layers of 512 hidden units each activated with ReLU [73] nonlinearity and an output layer with tanh activation. Experiments use this architecture as a result of its consistency in baseline implementations. Agents use Adam [45] as the optimizer (refer to section A.4 for hyperparameters). For ESAC and SAC, agents utilize a Diagonal Gaussian (DG) policy [95] without automatic entropy tuning for a fair comparison. Training of gradient-based methods was conducted on 4 NVIDIA RTX2070 GPUs whereas for ES and ESAC, a total of 64 AMD Ryzen 2990WX CPUs were used.

Table 3.1 presents total average returns of agents on all 15 tasks considered for experiments. ESAC demonstrates improved returns on 10 out of 15 tasks. ESAC makes use of evolution-based weight-space exploration to converge to robust policies in tasks where SAC often learns a sub-optimal policy. Moreover, utilization of evolutionary operations demonstrates consistency across different seeds for high-dimensional Humanoid tasks indicating large-scale suitability of the method to complex control.

2.5.2 Behaviors

Combination of RL and evolutionary methods provides suitable performance on control benchmarks. It is essential to assess the behaviors learned by agents as a result of weight-space exploration. We turn our attention to observe meaningful patterns in agent's behavior during its execution in the environment. More specifically, the experiment aims to evaluate the robustness of ESAC scheme which promises an efficacious policy as a result of effective exploration. The

setup initializes a learned ESAC policy on the WalkerWalk task and places it in a challenging starting position. The Walker agent stands at an angle and must prevent a fall in order to complete the task of walking suitably as per the learned policy. Figure 2.4 demonstrates the behavior of the Walker agent during its first 100 steps of initialization. The agent, on its brink of experiencing a fall, is able to gain back its balance and retain the correct posture for completing the walking task. More importantly, the agent carries out this manoeuvre within 50 timesteps and quickly gets back on its feet to start walking. Figure 2.4 is an apt demonstration of robust policies learned by the ESAC agent. Dominant skill transfer arising from hindsight crossovers between winners and offsprings provisions effective exploration in weight space.

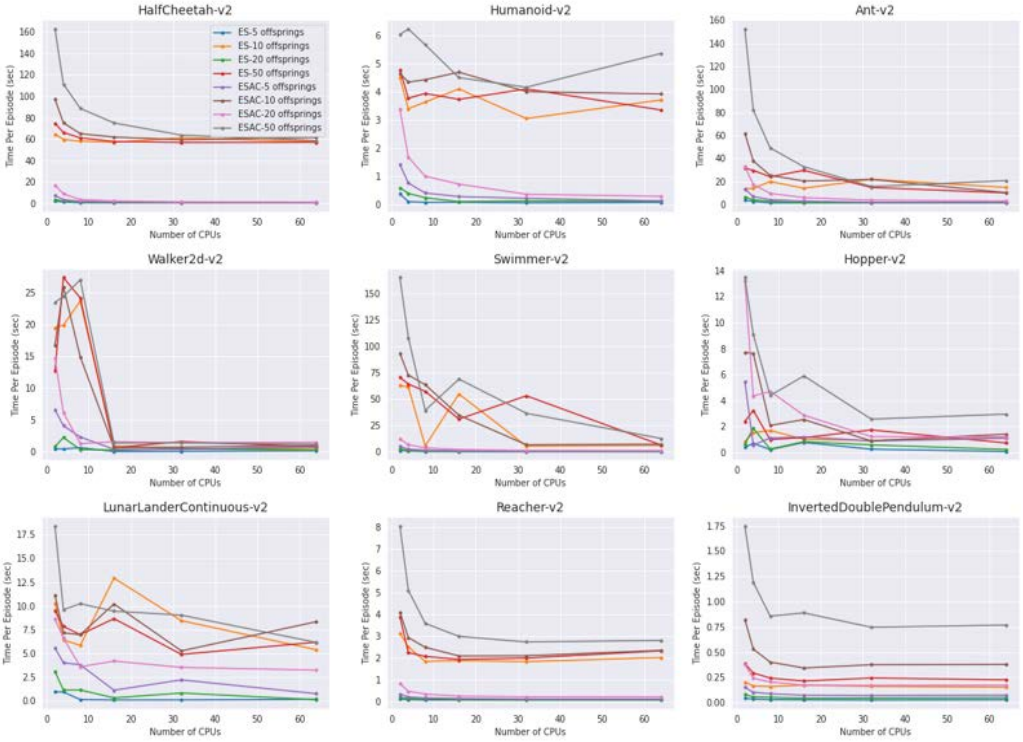

Figure 2.5: Variation of average time per episode (in seconds) with the number of operational CPUs and population size (in legend).

2.5.3 Scalability

Experiments assess scalability of the method with ES on the basis of hardware resources and algorithm overhead. The setup varies the number of CPUs by keeping other training parameters constant. Parallelization on multiple CPU-based resources is readily available and cost-efficient in comparison to a single efficient GPU resource. Experiments also vary number of offsprings in the population by fixing CPU resources. Out of mutation rate σ and population

size n, n governs the computational complexity of ES with σ being a scalar value. Thus, assessing variation w.r.t n provides a better understanding of resource utility and duration. For both experiments, the setup trains the population for 10^6 steps and averages the wall-clock time per episode. Another effective way to demonstrate scalability is by monitoring overall time taken to complete the training tasks [86]. However, this often tends to vary as initial learning periods have smaller episode lengths which does not compensate for fixed horizons of 1000 steps in MuJoCo.

Figure 2.5 presents the scalable nature of ESAC equivalent to ES on the MuJoCo and LunarLanderContinuous tasks. Average wall-clock time per episode is reduced utilizing CPU resources which is found to be favourable for evolution-based methods. Moreover, the variation depicts consistency with the increasing number of members in the population indicating large-scale utility of the proposed method. A notable finding here is that although ESAC incorporates gradient-based backpropagation updates, it is able to preserve its scalable nature by making use of evolutions as dominant operations during the learning process. This is in direct contradiction to prior methods [43]which demonstrate reduced sample-efficiency and the need for significant tuning when combining RL with scalable evolutionary methods. Reduction in the number of SAC updates by exponentially annealing the gradient interval allows ESAC to reduce computation times and simultaneously explore using AMT.

2.5.4 Ablation Study

Mutation Sensitivity: ES presents sensitivity to σ which is addressed by making use of AMT in ESAC. The AMT update gradually increases mutation rate σ using clip parameter ζ as learning progresses. Figure 2.6 (top) presents the variation of average normalized rewards with different values of the new hyperparameter ζ for HalfCheetah-v2 and Ant-v2 tasks. Each experiment was run for 1 million steps. The population presents robustness and performance improvement for small values with the optimal range being 10^{-4} to 10^{-2}. On the other hand, sensitivity is observed in the 10^{-1} to 1 region which accounts for larger updates with high variance. Hyperparameter variation is limited to a smaller region, in contrast to a wider spread of σ in the ES update. Offsprings remain robust to significantly large values of ζ due to early convergence of the population at the cost of poor performance among weak learners of the population. However, this is addressed by making use of hindsight crossovers which allow simultaneous

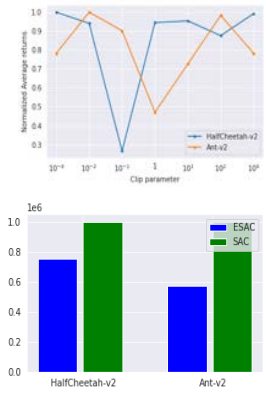

Figure 2.6: Top: Mutation sensitivity of ESAC, Bottom: Number of backprop updates in ESAC compared to SAC

transfer of dominant traits.

Number of Updates: The main computation bottleneck in SAC arises from the number of backprop updates. This is tackled by exponentially annealing these updates and increasing winner-based evolutions and crossovers for transferring skills between SAC agent and ES offspings. Figure 2.6 (bottom) presents a comparison between the number of backprop updates carried out using SAC and ESAC during the training phase of HalfCheetah-v2 and Ant-v2 tasks. Results for the updates are averaged over 3 random seeds. ESAC executes lesser number of updates highlighting its computationally efficient nature and low dependency on a gradient-based scheme for monotonic improvement. Complete results can be found in section A.3.

2.6 Summary

This chapter introduced ESAC which combines the scalable nature of ES with low approximation bias and state-of-the-art performance of SAC. ESAC addresses the problem of mutation-sensitive evolutions by introducing AMT which maximizes the mutation rate of evolutions in a small clipped region as the SAC updates are exponentially decayed. ESAC demonstrates improved performance on 10 out of 15 MuJoCo and DeepMind Control Suite tasks including different versions of Humanoid. Additionally, ESAC presents scalability comparable to ES, reducing the average wall-clock time per episode by approximately 60%, hence depicting its suitability for large-scale RL tasks involving continuous control.

Chapter 3

Energy-based Reinforcement Learning

3.1 Overview

Multi-Agent Reinforcement Learning (MARL) has demonstrated significant success in training decentralised policies in a centralised manner by making use of value factorization methods. However, addressing surprise across spurious states and approximation bias remain open problems for multi-agent settings. Towards this goal, this chapter introduces the *Energy-based MIXer (EMIX)*, an algorithm which minimizes surprise utilizing the energy across agents. Contributions of this chapter are threefold; (1) EMIX introduces a novel surprise minimization technique across multiple agents in the case of multi-agent partially-observable settings. (2) EMIX highlights a practical use of energy functions in MARL with theoretical guarantees and experiment validations of the energy operator. Lastly, (3) EMIX extends Maxmin Q-learning for addressing overestimation bias across agents in MARL. In a study of challenging StarCraft II micromanagement scenarios, EMIX demonstrates consistent stable performance for multi-agent surprise minimization. Moreover, the ablation study highlights a necessity of the energy-based scheme and the need for elimination of overestimation bias in MARL. Implementation of EMIX can be found at this link.

3.2 The Value Factorization Problem

3.2.1 Surprise Minimization

Despite the recent success of value-based methods [67, 37] RL agents suffer from spurious state spaces and encounter sudden changes in trajectories. These anomalous transitions between consecutive states are termed as *surprise* [1]. Quantitatively, surprise can be inferred as a measure of deviation [8, 16] among states encountered by the agent during its interaction with the environment.

While exploring [12, 109] the environment, agents tend to have higher deviation among states which is gradually reduced by gaining a significant understanding of state-action transitions. Agents can then start selecting optimal actions which is essential for maximizing reward. These actions often lead the agent to spurious experiences which the agent may not have encountered. In the case of model-based RL, agents leverage spurious experiences [8] and plan effectively for future steps. In the case of model-free RL, surprise results in sample-inefficient learning [1]. This can be tackled by making use of rigorous exploration strategies [96, 53]. However, such techniques do not necessarily scale to high-dimensional tasks and require extrinsic feature engineering [49] in conjunction with meta models [26]. A suitable way to tackle high-dimensional dynamics is by utilizing surprise as a penalty on the reward [16]. This leads to improved generalization. However, such solutions do not show evidence of improvement in multiple agents [82].

3.2.2 Overestimation Bias

Recent advances [23] in value-based methods have addressed overestimation bias (also known as approximation error) which stems from the value estimates approximated by the function approximator. Such methods make use of dual target functions [116] which improve stability in the Bellman updates. This has led to a significant improvement in single-agent off-policy RL methods [30]. However, MARL value-based methods continue to suffer from overestimation bias [2, 59].

Figure 3.1 highlights the overestimation bias originating from the overoptimistic estimations of the target value estimator. Plots present the variation of absolute TD error during learning for state-of-the-art MARL methods, namely Independent Q-Learning [104], Counterfactual Multi-Agent Policy Gradients (COMA) [20], VDN [99] and QMIX [80]. Significant rise in error values of value factorization methods such as QMIX and VDN presents the aggregation of errors from individual Q-value functions.

Various MARL methods [22] make use of a dual architecture approach which increases the stability in value factorization. Another suitable approach observed in literature is the usage of weighted bellman updates in double Q-learning [123]. The Weighted Double Deep Q-Network (WDDQN) provides stability and sample efficiency for fully-observable MDPs. In the case of cooperative POMDPS, Weighted-QMIX (WQMIX) [81] yields a

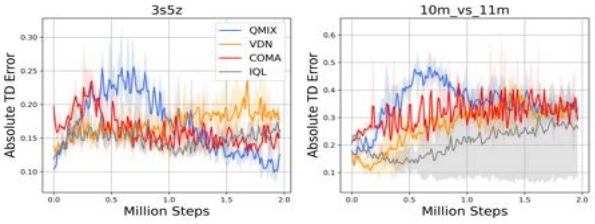

Figure 3.1: Absolute TD error for state-of-the-art MARL methods in StarCraft II micromanagement scenarios.

more sophisticated weighting scheme which aids in the retrieval of optimal policy [75]. Although suitable for value factorization in challenging micromanagement tasks, the method needs to be carefully hand-engineered and, in the case of multiple weighting schemes, does not include a basis for selection. A more practical approach in the case of single-agent methods is the use of a family of Q-functions [50] wherein each estimator is optimized individually. Such a framework provides a generalized method for training agents with greedy policies and minimum approximation error. Although successful in single-agent settings, generalized Q-function methods do not demonstrate evidence for scalability in the number of agents [75].

3.2.3 Energy-based Models

Energy-Based Models (EBMs) [51, 52] have been successfully applied in the field of machine learning [106] and probabilistic inference [62]. A typical EBM $\mathcal{E}$ formulates the equilibrium probabilities [88] $P(v,h) = \frac{\exp\left(-\mathcal{E}(v,h)\right)}{\sum_{\hat{v},\hat{h}}[\exp\left(-\mathcal{E}(\hat{v},\hat{h})\right)]}$ via a Boltzmann distribution [54] where v and h are the values of the visible and hidden variables and $\hat{v}$ and $\hat{h}$ are all the possible configurations of the visible and hidden variables respectively. The probability distribution over all the visible variables can be obtained by summing over all possible configurations of the hidden variables. This is mathematically expressed in Equation 3.1.

$$P(v) = \frac{\sum_h[\exp\left(-\mathcal{E}(v,h)\right)]}{\sum_{\hat{v},\hat{h}}[\exp\left(-\mathcal{E}(\hat{v},\hat{h})\right)]} \tag{3.1}$$

Here, $\mathcal{E}(v,h)$ is called the equilibrium free energy which is the minimum of the variational free energy and $\sum_{\hat{v},\hat{h}}[\exp\left(-\mathcal{E}(\hat{v},\hat{h})\right)]$ is the partition function.

EBMs have been successfully implemented in single-agent RL methods [76, 29]. These typically make use of Boltzmann distributions to approximate policies [54]. Such a formulation results in the minimization of free energy within the agent. While policy approximation depicts promise in the case of unknown dynamics, inference methods [111] play a key role in optimizing goal-oriented behavior.

A second type of usage of EBMs follows the maximization of entropy [126]. The maximum entropy framework [30] highlighted in Soft Q-Learning (SQL) [29] allows the agent to obey a policy which maximizes its reward and entropy concurrently. Maximization of agent's entropy results in diverse and adaptive behaviors [125]. The maximum entropy framework is equivalent to approximate inference in the case of policy gradient methods [91]. Such a connection between likelihood ratio gradient techniques and energy-based formulations leads to diverse and robust policies [27] and their hierarchical extensions [28] which preserve the lower levels of hierarchies.

In the case of MARL, EBMs have witnessed limited applicability as a result

of the increasing number of agents and complexity within each agent [14]. While the probabilistic framework is readily transferable to opponent-aware multi-agent systems [118], cooperative settings require a firm formulation of energy which is scalable in the number of agents [25] and accounts for environments consisting of spurious states [117].

3.3 Energy-based Surprise Minimization

This section introduces the novel surprise minimizing EMIX agent. The motivation behind EMIX stems from spurious states and overestimation bias among agents in the case of partially-observed settings. EMIX aims to address these challenges by making use of an energy-based surprise value function in conjunction with dual target function approximators.

3.3.1 The Surprise Minimization Objective

Firstly, the setup formulates the energy-based objective consisting of surprise as a function of states s, joint actions u and standard deviations σ within states for each agent a. One can call this function as the surprise value function $V^a_{surp}(s, u, \sigma)$ which serves as a mapping from agent and environment dynamics to surprise. This leads to the defination of an energy operator presented in Equation 3.2 which sums the free energy across all agents.

$$\mathcal{T}V^a_{surp}(s, u, \sigma) = \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s, u, \sigma)\right) \tag{3.2}$$

The objective makes use of the Mellowmax operator [3] as the energy operator. The energy operator is similar to the SQL energy formulation [29] where the energy across different actions is evaluated. In the case of EMIX, inference is carried out across all agents with actions as prior variables. However, in the special case of using an EBM as a Q-function, the EMIX objective reduces to the SQL objective. The chapter directs the curious reader to section B.2 for details on connection between SQL and EMIX energy formulation.

The choice of the energy operator is based on its unique mathematical properties which result in better convergence. Of these properties, the most useful result is that the energy operator forms a contraction on the surprise value function indicating a guaranteed minimization of surprise within agents. This is formally stated in Theorem 1. Proof of Theorem 1 is deferred to section B.1.

Theorem 1. *Given a surprise value function* $V^a_{surp}(s, u, \sigma) \forall a \in N$*, the energy operator* $\mathcal{T}V^a_{surp}(s, u, \sigma) = \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s, u, \sigma)\right)$ *forms a contraction on* $V^a_{surp}(s, u, \sigma)$*.*

The energy-based surprise minimization objective can then be formulated by simply adding the approximated energy-based surprise to the initial Bellman objective as expressed below.

$$L(\theta) = \mathop{\mathbb{E}}_{b\sim R}[\frac{1}{2}(y - (Q(u,s;\theta) + \beta \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s,u,\sigma)\right)))^2]$$

$$= \mathop{\mathbb{E}}_{b\sim R}[\frac{1}{2}(r + \gamma \max_{u^{'}} Q(u^{'},s^{'};\theta^{-}) + \beta \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s^{'},u^{'},\sigma^{'})\right)$$
$$- (Q(u,s;\theta) + \beta \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s,u,\sigma)\right)))^2]$$

$$= \mathop{\mathbb{E}}_{b\sim R}[\frac{1}{2}(r + \gamma \max_{u^{'}} Q(u^{'},s^{'};\theta^{-}) + \beta \log \frac{\sum_{a=1}^{N} \exp\left(V^a_{surp}(s^{'},u^{'},\sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V^a_{surp}(s,u,\sigma)\right)} - Q(u,s;\theta))^2]$$

$$L(\theta) = \mathop{\mathbb{E}}_{b\sim R}[\frac{1}{2}(r + \gamma \max_{u^{'}} Q(u^{'},s^{'};\theta^{-}) + \beta E - Q(u,s;\theta))^2]$$

Here, E is defined as the surprise ratio with β as a temperature parameter and $\sigma^{'}$ as the deviation among next states in the batch. The surprise value function is approximated by a universal function approximator (in this study a neural network) with its parameters as ϕ. $V_a(s^{'},u^{'},\sigma^{'})$ is expressed as the negative free energy and $\sum_{a=1}^{N} \exp\left(V_a(s,u,\sigma)\right)$ the partition function. Alternatively, $V_a(s,u,\sigma)$ can be formulated as the negative free energy with $\sum_{a=1}^{N} \exp\left(V_a(s^{'},u^{'},\sigma^{'})\right)$ as the partition function. The objective incorporates the minimization of surprise across all agents as minimizing the energy in *surprising* states. Such a formulation of surprise acts as intrinsic motivation and at the same time provides robustness to multi-agent behavior. Furthermore, the energy formulation in the form of energy ratio E is a suitable one as it guarantees convergence to minimum surprise at optimal policy π^*. This is formally expressed in Theorem 2 with its corresponding proof in section B.1.

Theorem 2. *Upon agent's convergence to an optimal policy π^*, total energy of π^*, expressed by E^* will reach a thermal equilibrium consisting of minimum surprise among consecutive states s and $s^{'}$.*

The objective can be modified to address approximation error in the target Q-values using Maxmin Q-learning [50]. This leads to the introduction of m target approximators making $\{Q_1(u^{'},s^{'};\theta^{-}), Q_2(u^{'},s^{'};\theta^{-})..., Q_m(u^{'},s^{'};\theta^{-})\}$ as the set of target approximators. This allows the objective to address overestimation bias in a scalable manner without using multiple Q-functions at the same time.

The complete EMIX objective is mathematically expressed in Equation 3.3.

$$L(\theta) = \mathop{\mathbb{E}}_{b\sim R}[\frac{1}{2}(r + \gamma \max_{u'} \min_{i} Q_i(u', s'; \theta^-) + \beta E - Q(u, s; \theta))^2] \tag{3.3}$$

Here, i depicts each of the m target estimators with $\min_i Q_i(u', s'; \theta^-)$ indicating the estimate with minimum error.

3.3.2 Energy-based MIXer (EMIX)

Algorithm 2 Energy-based MIXer (EMIX)

Initialize ϕ, θ, $\theta_1^-, ..., \theta_m^-$, agent and hypernetwork parameters.
Initialize learning rate α, temperature β and replay buffer $\mathcal{R}$.
for environment step **do**
  $u \leftarrow (u_1, u_2..., u_N)$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{(s, u, r, s')\}$
  if $|\mathcal{R}| >$ batch-size **then**
    for random batch **do**
      $Q_{tot}^{\theta} \leftarrow$ *Mixer-Network*$(Q_1, Q_2..., Q_N, s)$
      $Q_i^{\theta^-} \leftarrow$ *Target-Mixer*$_i(Q_1, Q_2..., Q_N, s'), \forall i = 1, 2.., m$
      Calculate σ and σ' using s and s'
      $V_{surp}^a(s, u, \sigma) \leftarrow$ *Surprise-Mixer*(s, u, σ)
      $V_{surp}^a(s', u', \sigma') \leftarrow$ *Target-Surprise-Mixer*(s', u', σ')
      $E \leftarrow \log \frac{\sum_{a=1}^{N} \exp(V_{surp}^a(s', u', \sigma'))}{\sum_{a=1}^{N} \exp(V_{surp}^a(s, u, \sigma))}$
      Calculate $L(\theta)$ using E in Equation 3.3
      $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
    end for
  end if
  if update-interval steps have passed **then**
    $\theta_i^- \leftarrow \theta, \forall i = 1, 2.., m$
  end if
end for

Algorithm 3 presents the EMIX algorithm. The method initializes surprise value function parameters ϕ, mixer parameters θ, target parameters θ_i^- for $i = 1, 2..., m$ and lastly the agent and hypernetwork parameters of QMIX. A learning rate α, temperature β and replay buffer $\mathcal{R}$ are instantiated. During environment interactions, agents in state s perform joint action u, observe reward r and transition to next-states s'. These experiences are stored in replay buffer $\mathcal{R}$ as (s, u, r, s') tuples. In order to make agents explore the environment, an ϵ-greedy schedule is used similar to the original QMIX [80] implementation. During the update steps, a random batch of $batch - size$ is sampled from $\mathcal{R}$. The total Q-value Q_{tot}^{θ} is computed by the mixer network with its inputs as the Q-values of all the agents conditioned on s via the hypernetworks. Similarly, the target mixers approximate $Q_i^{\theta^-}$ conditioned on s'. In order to evaluate

surprise within agents, the method computes the standard deviations σ and $\sigma^{'}$ across all observations z and $z^{'}$ for each agent using s and $s^{'}$ respectively. The surprise value function called the *Surprise-Mixer* estimates surprise $V^{a}_{surp}(s, u, \sigma)$ conditioned on s, u and σ. The same computation is repeated using the Target-Surprise-Mixer for estimating surprise $V^{a}_{surp}(s^{'}, u^{'}, \sigma^{'})$ within next-states in the batch. Application of the energy operator for $V^{a}_{surp}(s, u, \sigma)$ and $V^{a}_{surp}(s^{'}, u^{'}, \sigma^{'})$ yields the energy ratio E which is used in Equation 3.3 to evaluate $L(\theta)$. Finally, batch gradient descent is used to update parameters of the mixer θ. Target parameters θ_{i}^{-} are updated every $update - interval$ steps.

We now take a closer look at the surprise-mixer approximating the surprise value function. In order to condition surprise on states, joint actions and the deviation among states, the setup constructs an expressive architecture motivated by provable exploration in RL [66]. Such models have proven to be efficient in the case of provable exploration [66] as it allows the agent to learn an exploration policy for every value of abstract state related to the latent space. The method borrows from this technique and extends it to the surprise minimization setting. Figure 3.2 presents the expressive architecture of surprise-mixer network utilized for surprise value function approximation and minimization. The surprise-mixer maps transitions consisting of states s, joint actions u and deviations σ to a surprise value $V^{a}_{surp}(s, u, \sigma)$ for all agents a. The architecture allows the agent to learn a robust and surprise-agnostic policy for every value of abstract state related to the latent space. Moreover, the latent space accommodates every value of surprise across agents as a result of state standard deviations induced in the intermediate representations. Surprise value estimates $V^{a}_{surp}(s, u, \sigma)$ are evaluated by the energy operator with the resulting expression becoming a part of the Bellman objective in Equation 3.3 comprising of the total Q-values Q_{tot} estimated by the mixer network.

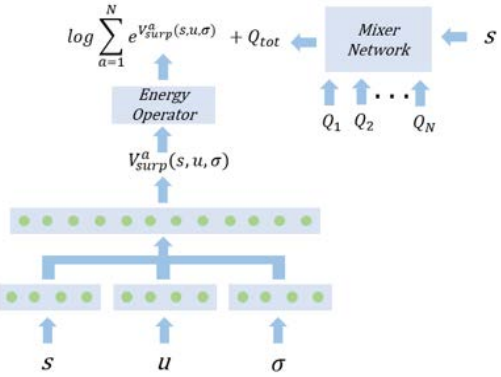

Figure 3.2: Surprise-Mixer architecture for estimation of the surprise value function.

3.4 Experiments

Experiments aim to evaluate the performance, consistency, sample-efficiency and effectiveness of the various components of the method. Specifically, the evaluation aims to answer the following questions- (1) How does the method compare to current state-of-the-art MARL methods in terms of performance, consistency and sample efficiency?, (2) How much does each component of the method contribute to its performance? and (3) Does the algorithm validate

Scenarios	EMIX	SMiRL-QMIX	QMIX	VDN	COMA	IQL
2s_vs_1sc	90.33 ± 0.72	88.41 ± 1.31	89.19 ± 3.23	91.42 ± 1.23	**96.90 ± 0.54**	86.07 ± 0.98
2s3z	**95.40±0.45**	94.93±0.32	95.30±1.28	92.03±2.08	43.33±2.70	55.74±6.84
3m	**94.90±0.39**	93.94±0.22	93.43±0.20	94.58±0.58	84.75±7.93	94.79±0.50
3s_vs_3z	**99.58±0.07**	97.63±1.08	99.43±0.20	97.90±0.58	0.21±0.54	92.32±2.83
3s_vs_4z	**97.22±0.73**	0.24±0.11	96.01±3.93	94.29±2.13	0.00±0.00	59.75±12.22
3s_vs_5z	52.91±11.80	0.00±0.00	43.44±7.09	**68.51±5.60**	0.00±0.00	18.14±2.34
3s5z	**88.88±1.07**	88.53±1.03	88.49±2.32	63.58±3.99	0.25±0.11	7.05±3.52
8m	**94.47±1.38**	89.96±1.42	94.30±2.90	90.26±1.12	92.82±0.53	83.53±1.62
8m_vs_9m	**71.03±2.69**	69.90±1.94	68.28±2.30	58.81±4.68	4.17±0.58	28.48±22.38
10m_vs_11m	75.35±2.30	**77.85±2.02**	70.36±2.87	71.81±6.50	4.55±0.73	32.27±25.68
so_many_baneling	**95.87±0.16**	93.61±0.94	93.35±0.78	92.26±1.06	91.65±2.26	74.97±6.52
5m_vs_6m	**37.07±2.42**	33.27±2.79	34.42±2.63	35.63±3.32	0.52±0.13	14.78±2.72

Table 3.1: Comparison of success rate percentages between EMIX and state-of-the-art MARL methods for StarCraft II micromanagement scenarios.

the theoretical claims corresponding to its components?

3.4.1 Energy-based Surprise Minimization

The setup assesses performance and sample-efficiency of EMIX on multi-agent StarCraft II micromanagement scenarios [89]. StarCraft II scenarios are selected particularly for three reasons. Firstly, micromanagement scenarios consist of a larger number of agents with different action spaces. This requires a greater deal of coordination in comparison to other benchmarks [98] which attend to other aspects of MARL performance such as opponent-awareness [13]. Secondly, micromanagement scenarios consist of partial observability wherein agents are restricted from responding to enemy fire and attacking enemies when they are in range [80]. This allows agents to explore the environment effectively and find an optimal strategy purely based on collaboration rather than built-in game utilities. Lastly, micromanagement scenarios in StarCraft II consist of multiple opponents which introduce a greater degree of surprise within consecutive states. Irrespective of the time evolution of an episode, environment dynamics of each scenario change rapidly as the agents need to respond to enemy's behavior.

Evaluation compares EMIX to current state-of-the-art methods, namely QMIX [80], VDN [99], COMA [20] and IQL [104]. In order to compare the surprise-based scheme against pre-existing surprise minimization mechanisms, the setup compares EMIX additionally to a model-free implementation of SMiRL [8] in QMIX. All methods were implemented using the PyMARL framework [89]. The SMiRL component was additionally incorporated as per the update rule provided in [16]. This implementation is called SMiRL-QMIX for comparisons. Agents were trained for a total of 5 random seeds consisting of 2 million steps in each environment. Experiments use an ϵ-greedy exploration scheme wherein ϵ is annealed from 1 to 0.01 during the initial stages of training. Details related to the implementation of EMIX are presented in section B.4.

In order to assess the performance and sample-efficiency of agents the

setup evaluates the success rate percentages of each multi-agent system in completing each scenario. A completion of a scenario indicates the victory of the team over its enemies. Scenarios consist of varying difficulties in terms of the number of agents, map locations, distance from enemies, number of enemies and the level of difficulty. Table 3.1 presents the comparison of success rate percentages between EMIX and state-of-the-art MARL algorithms on StarCraft II micromanagement scenarios. Out of the 12 scenarios considered, EMIX presents higher success rates on 9 of these scenarios depicting the suitability of the proposed approach. In scenarios such as *3m*, *3s5z* and *8m* performance gains between EMIX and other methods such as QMIX and VDN are incremental as a result of the small number of agents and simplicity of tasks. On the other hand, EMIX presents significant performance gains in cases of *so_many_baneling* and *5m_vs _6m* which consist of a large number of opponents and a greater difficulty level respectively. Complete results for all scenarios including plots presenting agents' learning performance can be viewed in section B.3.

When compared to QMIX, EMIX depicts improved success rates on all of the 12 scenarios. For instance, in scenarios such as *3s_vs_5z*, *8m_vs_9m* and *5m_vs_6m* QMIX presents sub-optimal performance. On the other hand, EMIX utilizes a comparatively improved joint policy and yields better convergence in a sample-efficient manner. Moreover, on comparing EMIX with SMiRL-QMIX, the reader may note that EMIX demonstrates a higher average success rate. This highlights the suitability of the energy-based scheme in the case of a larger number of agents and complex environment dynamics for surprise minimization.

In addition to improved performance and sample-efficiency, EMIX also presents consistency in its learning across different random seeds. Deviation in success rates for EMIX is comparable to pre-existing value factorization methods such as QMIX and VDN. This indicates that the energy-based formulation of surprise minimization is compatible with value factorization methods and enables all the agents to exhibit the same optimal behavior across different runs.

3.4.2 Ablation Study

The chapter now presents the ablation study for the various components of EMIX. Experiments aim to determine the effectiveness of the energy-based surprise minimization method and the multiple target Q-function scheme. Additionally, the setup also aims to determine the extent up to which the presented framework is viable in the standard QMIX objective.

Energy-based Surprise Minimization and Overestimation Bias: To weigh the effectiveness of the multiple target Q-function scheme the implementation omits the energy-based surprise minimization from EMIX and replace it with the prior QMIX objective. For simplicity, the implementation makes use

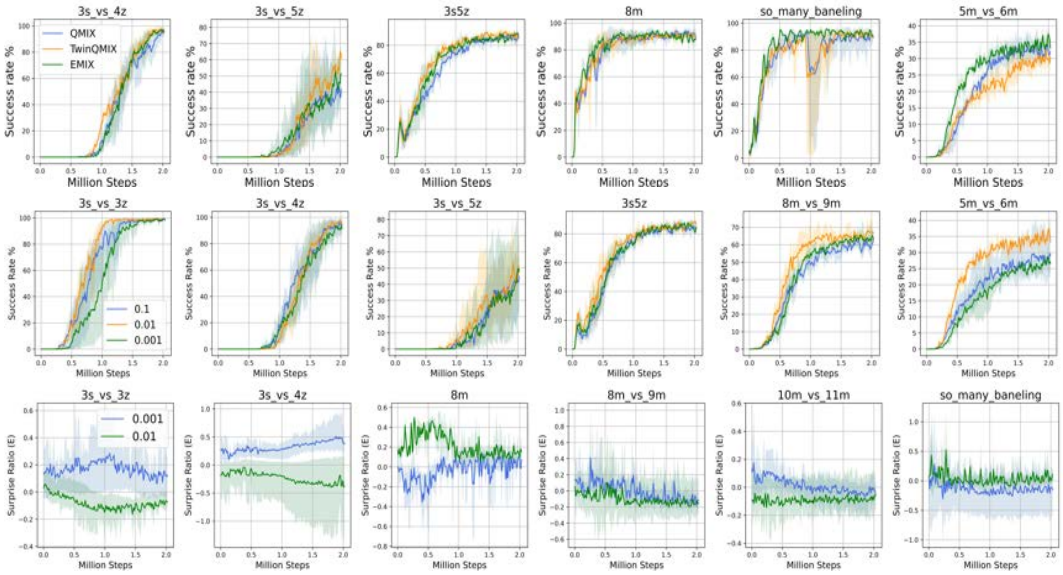

Figure 3.3: Ablations on six different scenarios for each of EMIX's components (top), its variation in performance (middle) and surprise minimization (bottom) with temperature β.

of two target Q-functions. This implementation of QMIX combined with the dual target function scheme is called *TwinQMIX*. Thus, experiments compare between QMIX, TwinQMIX and EMIX to assess the contributions of each of the proposed methods. Figure 4.5 (top) presents the comparison of average success rates for QMIX, TwinQMIX and EMIX on 6 different scenarios with lines indicating average success rates and the shaded area as standard deviation across 5 random seeds.

In comparison to QMIX, TwinQMIX adds stability to the original objective and yields performance gains in the form of improved success rates and sample-efficient convergence. For instance, in the *3s_vs_5z* scenario, TwinQMIX significantly improves the performance of QMIX by reducing the overoptimistic estimates in the initial QMIX objective. However, in the *5m_vs_6m* scenario, TwinQMIX falls short of optimal sample efficiency as a result of underoptimistic estimates yielded by the $\min_i Q_i^-(s, u, \sigma)$ operation.

On comparing TwinQMIX to EMIX the reader may note that the energy-based surprise minimization scheme provides significant performance improvement in the modified QMIX objective. The EMIX objective demonstrates sample-efficiency and improved success rate values when compared to the TwinQMIX implementation. Additionally, the surprise minimization term βE adds to the stability of the TwinQMIX objective. This is demonstrated in the *5m_vs_6m* scenario wherein the EMIX implementation improves the performance of TwinQMIX in comparison to QMIX by compensating for the underoptimistic estimations in the bellman updates. In the case of *so_many_baneling* scenario, EMIX tackles surprise effectively by preventing a significant

drop in performance which is observed in cases of QMIX and TwinQMIX. *so_many _baneling* scenario consists of a large number of opponents (27 banelings) which force the agents to act quickly. This inherently induces a large amount of surprise in the form of state-to-state deviations. EMIX successfully tackles this hindrance and prevents the drop in success rates.

Temperature Parameter: The study now evaluates the extent of effectiveness of surprise minimization objective in accordance with the temperature parameter β. Figure 4.5 (middle) presents the variation of success rates of the EMIX objective with β during learning. EMIX was evaluated for three different values (as presented in the legend) of β. While the objective is robust to significant changes in the value of β, it presents sub-optimal performance in the case of high ($\beta = 0.1$) and low ($\beta = 0.001$) temperature values. In the case of high β values, the objective suffers from overestimation error in the bellman updates introduced by the energy term. The error compensates for the bias removed by the dual Q-function scheme. On the other hand, low β values do not include surprise minimization and EMIX agents face surprising states.

The importance of β can be validated by assessing its usage in surprise minimization. However, it is difficult to evaluate surprise minimization directly as surprise value function estimates $V^a_{surp}(s, u, \sigma)$ vary from state-to-state across different agents and present high variance during agent's learning. The study instead observes the variation of energy ratio E as it is a collection of surprise-based sample estimates across the batch. Additionally, E consists of prior samples $V^a_{surp}(s, u, \sigma)$ for $V^a_{surp}(s^{'}, u^{'}, \sigma^{'})$ which makes inference across different agents tractable. Figure 4.5 (bottom) presents the variation of energy ratio E with the temperature parameter β during learning. The evaluation compares two stable variations of E at $\beta = 0.001$ and $\beta = 0.01$. The objective minimizes E over the course of learning and attains thermal equilibrium with minimum energy. Intuitively, equilibrium corresponds to convergence to optimal policy π^* which validates the claim in Theorem 2. With $\beta = 0.01$, EMIX presents improved convergence and surprise minimization for 5 out of the 6 considered scenarios, hence validating the suitable choice of β. On the other hand, a lower value of $\beta = 0.001$ does little to minimize surprise across agents.

3.5 Summary

This chapter introduced the Energy-based MIXer (EMIX), a multi-agent value factorization algorithm based on QMIX which minimizes surprise utilizing the energy across agents. The method proposes a novel energy-based surprise minimization objective consisting of an energy operator in conjunction with the surprise value function. The EMIX objective satisfies theoretical guarantees of total energy and surprise minimization with experimental results validating these claims. Additionally, EMIX extends Maxmin Q-learning for

addressing overestimation bias across agents in MARL. On a total 9 out of 12 challenging StarCraft II micromanagement scenarios, EMIX demonstrates improved consistent and stable performance for multi-agent surprise minimization. Ablations carried out on the proposed energy-based scheme, multiple target approximators and temperature parameter highlight the suitability of EMIX components.

Chapter 4

Hierarchical Reinforcement Learning for Trade Execution

4.1 Overview

Advances in Reinforcement Learning (RL) span a wide variety of applications which motivate development in this area. While application tasks serve as suitable benchmarks for real world problems, RL is seldomly used in practical scenarios consisting of abrupt dynamics. This allows one to rethink the problem setup in light of practical challenges. This chapter presents Trade Exeuction using Reinforcement Learning (TradeR) which aims to address two such practical challenges of *catastrophy* and *surprise minimization* by formulating trading as a real-world hierarchical RL problem. Through this lens, TradeR makes use of hierarchical RL to execute trade bids on high-frequency real market experiences comprising of abrupt price variations during the 2019 fiscal year COVID19 stock market crash. The framework utilizes an energy-based scheme in conjunction with surprise value function for estimating and minimizing surprise. In a large-scale study of 35 stock symbols from the S&P500 index, TradeR demonstrates robustness to abrupt price changes and catastrophic losses while maintaining profitable outcomes. I hope that my work serves as a motivating example for application of RL to practical problems.

4.2 Related Work

4.2.1 Trade Execution using RL

Various prior works in literature have adopted trading as a suitable task for developing practical RL algorithms. [69] presents the first RL trading framework capable of training Q-learning and recurrent RL agents based on financial metrics such as Sharpe Ratio. The work of [120] provides a framework for Deterministic Policy Gradients in optimizing daily trades using additional financial metrics and cumulative returns. [9] extends the RL trading strategy

towards intraday execution comprising of asynchronous distributed updates in conjunction with backtesting methods. The resulting method is found to be sample efficient. [24] and [107] reduce the need for financial metrics during evaluation as a result of optimization of cumulative portfolio returns. The framework of portfolio optimization is further extended to cryptocurrency based on memory vectors [108] and event-based transitions [85]. [15] generalizes these methods by retrieving the optimal strategy using deterministic portfolio optimization under varying dynamics.

While most methods focus on the application of RL to the trading problem, [121] improves the practicality of RL framework utilizing an ensemble of policy-based agents from the prior work of [120]. The framework for improving practical RL trading presented in this chapter is parallel to [121] and [120].

4.2.2 Hierarchical RL

Various RL pipelines [49, 72, 71, 4, 79, 18, 56, 113, 102] leverage hierarchical structure in policies allowing the agent to address different sequential problems. [79] presents the primary framework of temporal abstraction for training the policy at varying time scales consisting of sequential MDPs. [49] extends the temporal abstraction algorithm by incorporating intrinsic motivation by virtue of fulfilled sub-goals by the agent. Another suitable technique for learning different policy levels is the MAXQ algorithm [18] which provisions the usage of different Q-values at each stage of the hierarchy. However, MAXQ demonstrates exponential computational requirements in the case of increasingly complex action spaces. A more scalable alternative are Feudal Networks [113] which provision a manager and worker network for policy abstraction.

Various methods leverage hierarchies in order to solve sparse rewards and long-horizon tasks. [72] demonstrates the efficacy of learning hierarchies for locomotion tasks requiring efficient exploration. [71] highlights the need for sample efficient hierarchies in robot control using off-policy correction. [56] extend hierarchical RL for learning relevant skills which support continuous adaptation. This chapter's practical extension of hierarchical RL aligns well with these prior methods.

4.3 Trade Execution using Reinforcement Learning

This section introduces the TradeR framework. TradeR leverages hierarchies for long-horizon trade execution and an energy-based intrinsic motivation scheme for surprise minimization.

4.3.1 Hierarchical Trade Execution

Maximization of long-term payoffs requires an agent to predict accurately towards the future. Accurate predictions steer agent's beliefs toward an optimal

policy $\pi^*(a_t|s_t)$. However, incorporating accurate prediction mechanisms has proven to be a non-trivial challenge in the long-horizon. While bootstrapping value estimates in TD targets extends the forward-view of the agent, this small fix in itself is insufficient as timesteps tend to infinity. Another suitable alternative is to make use of n-step returns which trade-off high variance at the cost of biased estimates. Tuning the discount factor γ also does little to align agent's beliefs with counterfactual states stemming from expected value estimates.

Hierarchical abstraction of policy into its components is a suitable mechanism which allows temporal coherency in state and value predictions. A policy $\pi(a_t|s_t)$ may be decoupled into its M components $(\pi_1(a_t|s_t), \pi_2(a_t|s_t), ...\pi_M(a_t|s_t))$ where each sub-policy $\pi_m(a_t|s_t)$ is responsible for solving a sub-task in the form of an MDP. Consequently, the joint MDP is decoupled into M sub-MDPs in which the agent can interact sequentially or simulatenously. In the case of sequential interaction, the sub-MDPs together form a Markov Chain representing the joint MDP. In the case of simulatenous interactions, the joint MDP constitutes a multi-node head-to-tail graph wherein the sub-MDPs are temporally independent. This insight is presented in Figure 4.1.

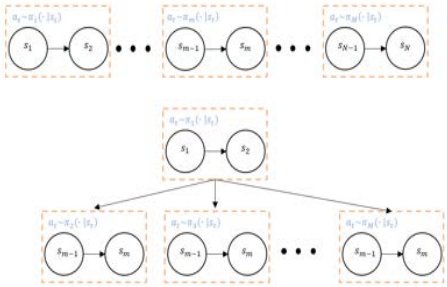

Figure 4.1: Contrast between Sequential and Simultaneous sub-MDPs.

The TradeR framework utilizes sequential interactions of policies in its sub-MDPs. The joint trading problem is abstracted into two sub-problems of (1) order determination and (2) bid execution. The agent with parameters θ, jointly optimizes the objective $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$ since both tasks share a common reward function. Each task is executed using a nonlinear function approximator (deep neural networks) denoting the sub-policy $\pi_{ord}(\cdot|s_t)$ for order network and $\pi_{bid}(a_t|s_t)$ for bid network respectively. The chapter directs the reader to section C.1 for details on algorithm.

Order Determination: At a given time-step t, the order policy observes state s_t from the trading environment. Upon observing state s_t, the order network deterministically follows the sub-policy $\pi_{ord}(y_t|s_t)$ for estimating the order quantity y_t to be bought, sold or held by the agent. The quantity y_t denotes the amount agent is willing to expense in the case of a loss or gain in the case of a profit.

Bid Execution: The high-level bid policy observes s_t and the quantity estimated by order policy y_t, in order to yield an action $a_t \sim \pi_{bid}(a_t|s_t)$ denoting the bid to be executed (buy, sell or hold the quantity). Action a_t is executed

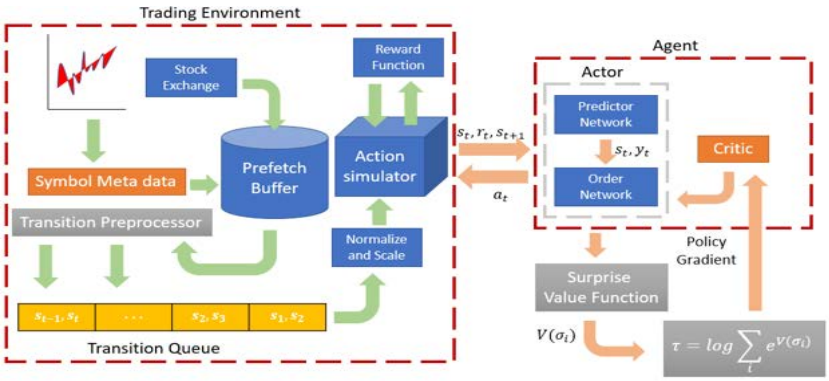

Figure 4.2: The hierarchical TradeR framework for minimizing catastrophy using surprise value function as part of the energy-based scheme.

in the environment by the dual-policy framework which results in a transition to the next state s_{t+1} by means of the transition probability $p(s_{t+1}|s_t, a_t) \in P$.

Order and bid policies can be updated jointly or asynchronously using either the on-policy or off-policy setting. In the on-policy case, order determination can be carried out on price samples being observed by the current policy. Bids are then executed on price samples and the corresponding quantities being estimated. In the off-policy setting, a replay buffer may be used to store experiences of quantity estimation and bid executions which are later used to update the policy. Compatibility of the framework with both on-policy and off-policy settings highlights that TradeR can be combined with any RL agent. Another notable observation is that while the action space for order determination is continuous, the bid execution task consists of discrete actions. Such a framework consisting of both continuous and discrete action spaces is representative of the practical nature of trading wherein the order quantity is prefixed in a given range and human experts make the bid once a suitable price quote is encountered.

4.3.2 Energy-based Intrinsic Motivation

Despite the provision of hierarchical order and bid policies, the agent's policy may cripple to sub-optimal performance as a result of abrupt changes in prices. This is in agreement with modern RL applications of robot control [72, 71] wherein a sudden change in the position and angle of robot's placement hurts the robustness of the policy and steers it off the path towards convergence. One can leverage contraction theory [17, 6] to highlight mathematical properties which would aid in extracting a surprise-robust policy. Such a mapping guarantees convergence in asymptotically-stable nonlinear systems [6] and provides a dynamical framework for assessing stability of control in RL [29]. A contraction mapping between any two functions $f_1(w)$ and $f_2(w)$ implies that the norm

distance between $f_1(w)$ and $f_2(w)$ decays at a constant (in some cases geometric [17]) rate. Given a contraction operator τ when iteratively applied on $f_1(w)$ and $f_2(w)$, the mapping $\tau f_1(w) - \tau f_2(w)$ is a contraction if the inequality in Equation 4.1 is satisfied.

$$||\tau f_1(w) - \tau f_2(w)|| \leq \eta ||f_1(w) - f_2(w)||, \ \forall w \text{ s.t. } \eta < 1 \tag{4.1}$$

Equation 4.1 is a generalization of the fixed-point theory in Banach metric spaces [6] which provides suitable conditions for assessing stability of nonlinear systems. The key component of evaluating a nonlinear system is motivated by its convergence towards a fixed point in the Banach metric space. Convergence towards a fixed point indicates stability of the overall mapping.

The method borrows from this insight in order to form a contraction on standard deviations in state transitions as a continuum in a nonlinear space. Upon realizing input samples as standard deviations of state transitions in this continuous nonlinear space, a simple yet elegant formulation of a contraction mapping can be achieved. To utilize a contraction mapping on σ_i with i being the state dimension, we seek a contraction operator $\mathcal{T}$ which is tractable.

A suitable choice is the alternate Boltzmann (mellowmax) operator introduced in [3] and presented in Equation 4.2.

$$\mathcal{T} f(w) = \log \sum_{w} \exp f(w) \tag{4.2}$$

The Mellowmax operator can be interpreted as an energy-based function. Retaining properties of the Boltzmann distribution, Equation 4.2 is an asymptotically stable formulation of the Gibbs distribution. Mellowmax has been suitably adopted in control and learning settings [29, 101] wherein the probability distribution forms a continuum over the input space. Additionally, the exponent in $\mathcal{T}$ is tractable as it is followed by the log which prevents the arguments from exploding.

$$\hat{r}(s_t, a_t, \sigma_i) = r(s_t, a_t) + \log \sum_{i} \exp \left(V_{surp}(\sigma_i) \right) \tag{4.3}$$

Equation 4.3 presents the modified reward signal utilized for surprise minimization as an intrinsic motivation objective. The method first encodes the stanard deviations σ_i corresponding to each dimension of state space i using a surprise value function $V_{surp}(\sigma_i)$. The surprise value function $V_{surp}(\sigma_i)$ quantifies the degree of surprise encountered by the agent in past states of the batch based on transition dynamics of the MDP. Note that $V_{surp}(\sigma_i)$ does not build a model of the world and it does not have access to state space dynamics. The surprise value function serves as *surprise critic* guiding the agent away from surprising states. Figure 4.2 presents the complete TradeR framework. Application of surprise value function is followed by the mellowmax operator which estimates nonlinear surprising configurations corresponding to actions of the agent. This

energy-based estimate of surprise is coupled with reward $r(s_t, a_t)$ as intrinsic motivation in order to yield the surprise-agnostic reward signal $\hat{r}(s_t, a_t)$.

4.3.3 The TradeR Algorithm

Algorithm 3 TradeR

Initialize policy $\pi_\theta = (\pi_{ord}, \pi_{bid})$ using θ and surprise value function V_{surp} using ϕ.
Initialize learning rate α, temperature β and replay buffer $\mathcal{R}$.
Initialize environment with initial state s_1
for environment step **do**
    Sample quantity $y_t \sim \pi_{ord}(\cdot|s_t)$
    Sample bid $a_t \sim \pi_{bid}(\cdot|s_t, y_t)$
    Execute a_t and observe s_{t+1} and r_t
    Update buffer $\mathcal{R} = \mathcal{R} \cup (s_t, a_t, r_t, s_{t+1})$
end for
for random batch **do**
    Compute deviations σ_i in states
    Set reward $\hat{r}(s_t, a_t, \sigma_t) = r(s_t, a_t) + \beta \log \sum_i \exp(V_{surp}(\sigma_i))$
    Calculate policy loss $L(\theta)$ using $\hat{r}(s_t, a_t, \sigma_i)$
    Update policy $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
end for

Algorithm 3 presents the TradeR algorithm corresponding to an RL agent. The hierarchical policy of the RL agent π_θ is composed of order and bid policies, π_{ord} and π_{bid}, respectively and parameterized using θ. The notation uses θ to jointly denote the parameters of the order and bid policies. Corresponding to each step in the environment, the agent observes state s_t as a vector of prices and volumes of symbol corresponding to the past 5 timesteps. The agent then samples an order quantity y_t from its order policy $\pi_{ord}(\cdot|s_t)$. Based on the order quantity, the agent samples the final bid a_t from its high-level bid policy $\pi_{bid}(\cdot|s_t, y_t)$ which it executes in the environment to transition to next state s_{t+1} and observe reward r_t. The replay buffer is updated with the corresponding (s_t, a_t, r_t, s_{t+1}) tuple.

During the learning phase, the agent samples a random batch of transitions from the replay buffer and computes standard deviations σ_i between consecutive states of the batch. The surprise value function V_{surp} estimates surprise based on standard deviation values σ_i which is further encoded using the energy-based Mellowmax operator. The energy-based estimate is weighed using a temperature parameter β with the transition reward $r(s_t, a_t)$ to yield the complete reward signal $\hat{r}(s_t, a_t, \sigma_i)$. Finally, the agent's policy is updated using a standard RL loss $L(\theta)$ computed from reward $\hat{r}(s_t, a_t, \sigma_i)$ and target estimates $Q(s_{t+1}, a_{t+1})$.

4.4 Experiments

Experiments aim to assess practical application of TradeR and its components. More specifically, the evaluation addresses the following three questions-

- How can TradeR be applied to practical trading pipelines consisting of real market data?
- What trends (if any) does TradeR present which would aid in understanding its practical utility?
- What contributions do TradeR's components hold?

4.4.1 Learning Trade Execution

Experiments begin by analyzing the practical utility of TradeR from real market data. The study collected price transitions from the Global US Stock Market for 2019 fiscal year comprising of COVID19 market crash. In order to tackle catastrophy and surprise minimization, the setup aims to challenge the algorithm with regard to the best and worst case scenarios. To this end, the study shortlisted 20 best and 15 worst stock symbols from the S&P500 index consisting of price transitions at 1 minute intervals. TradeR is combined with Proximal Policy Optimization (PPO) [92] and compared with strong baselines such as Deep Deterministic Policy Gradient (DDPG) [57] and Twin-Delayed DDPG (TD3) [23]. In the interest of evaluating data-efficiency and performance over longer temporal spans, agents were trained for only 500 episodes consisting of 10^4 steps each. Note that this is a significantly challenging training setup when compared to conventional RL benchmarks [110, 105, 21, 122] which train locomotion agents upto 10^3 steps. In order to evaluate practical utility of agents, the evaluation normalizes all returns with respect to maximum profits earned by a professional trading algorithm. The chapter directs the curious reader to section C.1 for details on experimental setup.

Figure 4.3 (left) presents the combined normalized performance of TradeR on all symbols in comparison to baseline methods and PPO. TradeR learns quickly as a result of bi-level abstract hierarchies in the framework. Estimation of meaningful order quantities by the order policy provisions the high-level bid policy to learn suitable bidding strategies which lead to profitable episodes in the long-horizon. While deterministic execution algorithms such as DDPG and TD3 demonstrate high variance across random seeds, TradeR presents consistent trends in its returns.

While improved returns highlight the suitability of an RL algorithm, they do not necessarily throw light on its practical viability. Various RL methods demonstrate superhuman level performance [68, 112] but only a handful can adapt towards real-world experience [7]. Motivated by this insight, experiments aim to answer the central insight of TradeR's practical utility.

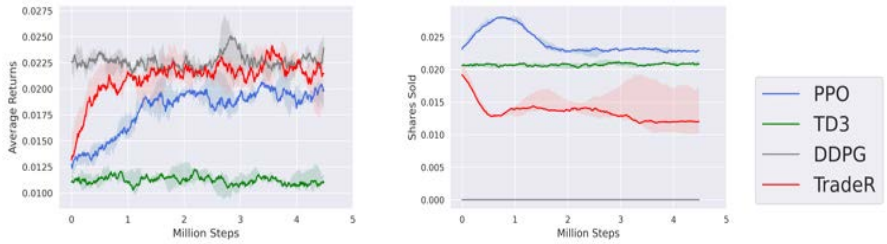

Figure 4.3: Left: Average normalized returns for all agents on the 35 symbol S&P500 trading benchmark.

Towards this evaluation, one can study the trading strategies learned by agents during training. Trading strategies are observed as per the manner in which an agent manages its share inventory. Figure 4.3 (right) presents shares sold by agents across all symbols during training. TD3 and DDPG agents maintain deterministic policies and do not improve their trading strategies in accordance with policy improvement steps. PPO, on the other hand, presents a different trading strategy. The PPO agent begins by selling a large number of shares which is akin to huge losses for the 2019 fiscal year. Following the volatile trend observed in the market, policy improvement steps gradually improve the strategy of agent towards a conservative trading behavior. Upon approaching convergence, PPO reduces its share selling strategy and holds on to valuable assets with the motivation for long-term payoffs.

Trading strategies for baseline methods fall prey to market crash and are only able recover after large number of updates (as in PPO). TradeR, on the other hand, exhibits a robust trading strategy which learns to minimize the number of shares sold efficiently. The order policy prefixes a low order quantity based on volatile price variations which the bid policy infers as a *sell* bid. The TradeR agent thus continues to *buy more* and *sell less* leading to an optimal policy yielding high long-term returns. This key property of *data-adaptive execution* is akin to *catastophy* and *surprise minimization*.

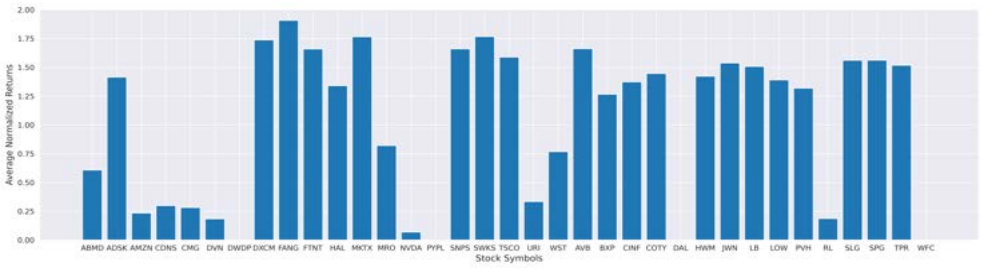

Figure 4.4: Performance of TradeR observed on all 35 S&P500 symbols consisting of best (ABMD-WST) and worst (AVB-WFC) performing stocks for 2019 fiscal year.

To understand *data-adaptive execution* we turn our attention back towards returns. As a consequence of best and worst symbols in the setup, TradeR

demonstrates visible patterns which distinguish its performance across different trajectories. Figure 4.4 presents comparison of returns obtained by TradeR upon convergence on all 35 symbols in the setup. The arrangment (from left to right) depicts best (ABMD-WST) to worst (AVB-WFC) symbols during the pandemic. A notable finding one may observe is that TradeR performs consistently well on worst symbols with its performance variable for best ones. This stark distinction in TradeR's performance is reflective of its practical applicability highlighting greedy exploitation at lower prices. The agent familiarizes itself with the trend of price variations and utilizes the lower prices of worst symbols to maximize its payoffs. In the case of best symbols, the agent is aware of rising prices but at the same time conservative with regard to the volatility of the market. This results in a safe strategy reflecting the practical tradeoff between risk and profit in trading.

4.4.2 Ablation Study

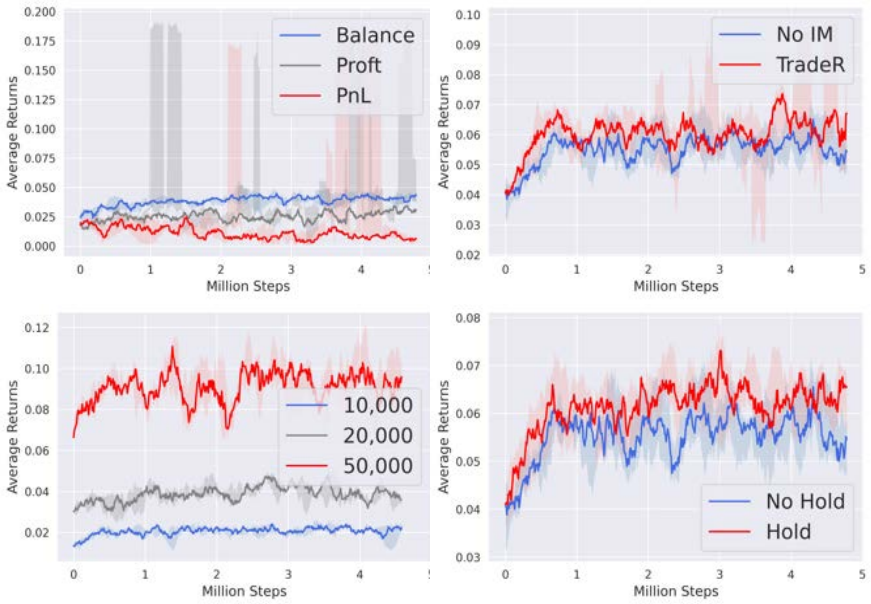

Figure 4.5: Reward ablations for TradeR corresponding to all 35 symbols from the S&P500 benchmark.

The study now evaluates efficacy of various components of TradeR framework. To this end, the ablation study also throws light on practical considerations such as the choice of reward function and the starting balance amount provided to the agent.

Reward Functions: Figure 4.5 (top-left) presents reward ablations carried out for different reward functions on the 35 symbol S&P500 benchmark. The study considers three reward choices, (1) *balance amount* presents the current

remaining balance of the agent, (2) *Profits* indicate the net gains obtained from the start state of the episode until termination and (3) *Profit and Loss (PnL)* which denotes the per-step gains/loss between consecutive states. PnL, being a per-step varying reward signal, presents high variance due to instantaneous profits and losses observed by the agent as a result of frequent price fluctuations. Since PnL is a noisy and transition-dependent reward signal, it hinders the agent from learning meaningful trading behaviors due to its strong dependence on price quotes. Profit, on the othert hand, does not provision stable learning as a result of invarying rewards observed from the 'hold' action. The agent often makes an early profit and does not act in subsequent timesteps leading to sub-optimal convergence.

Balance demonstrates an apt learning performance as the agent is incited to act out of the risk of going bust. From the perspective of practical trading scenarios, balance has another key advantage. In the case of most symbols, the balance reward metric presents the least deviation due to its reduced dependence on price fluctuations. This allows the agent to holistically make decisions based on volume and prices in contrast to solely weighing price quotes at the current timestep.

Energy-based Intrinsic Motivation: Figure 4.5 (top-right) presents reward ablations for the energy based scheme. In the absence of energy-based Intrinsic Motivation (IM), the agent learns slowly and demonstrates sub-optimal trading strategies which fail to retrieve profitable outcomes. Intrinsic motivation aids in efficient surprise minimization by evading catastrophic states as a result of safety-directed behavior. The agent, by virtue of estimated deviations in state-transition dynamic, is able to seek low energy configurations. This in turn induces sample efficient and stable learning over random seeds.

Starting Balance: Figure 4.5 (bottom-right) presents reward ablations for different values of starting balance amounts. The TradeR agent is provided with $\$10K$, $\$20K$ and $\$50K$ as seed balance amount during training. The objective of this ablation experiment is to evaluate the stability of agent with growing amounts of trading burden. Performance of the framework is found consistently increasing with the increasing balance amount indicating its suitability to understand trade patterns and wisely leverage optimal policies for placing trade bids. Note that the agent does undergo catastrophic losses which is often the case with risk-taking human brokers who favour large bid amounts at larger balance values.

Hold Action: Contrary to pre-existing methods in literature [121, 15], the implementation provides the agent with an additional hold action which is equivalent to carrying out no transaction at the given timestep. Figure 4.5 (bottom-left) presents reward ablations for hold action. The study compares TradeR to the setting where the hold action is turned off and the agent is forced

to trade by either buying or selling shares. TradeR demonstrated improved stable performance in the case of hold action indicating the necessity for a broader action scope which is often restricted in practical RL applications due to the curse of dimensionality. The hold action provides agent with the ability to simply save on previously bought earnings and channel a transaction only when the suitable price is encountered. This is in accordance with human patterns of trading wherein traders buy shares in excess and allow their savings to grow until a suitable quote is encountered.

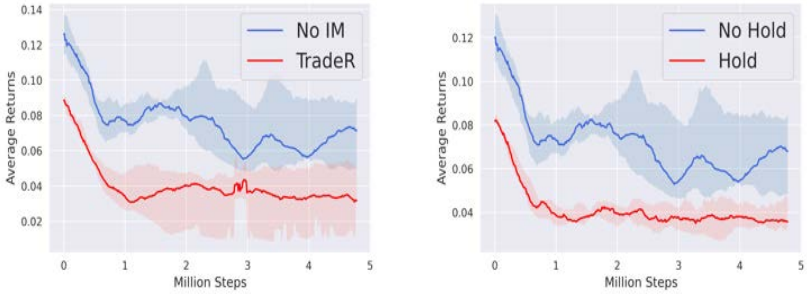

Figure 4.6: Ablations for total shares sold by TradeR agent over all 35 symbols from the S&P500 benchmark.

One can gain intuition into which components contribute the most towards TradeR's risk-averse and long-term behavior of *buy more and sell less*. Figure 4.6 (left) presents ablations for shares sold by TradeR agent with and without the presence of energy-based intrinsic motivation scheme. TradeR demonstrates a lower number of total shares sold in the presence of intrinsic motivation validating its suitability for surprise and catastrophy minimization. Moreover, the agent presents a lower variance across its random seeds indicating consisting risk-averse behavior in conjunction with rewarding outcomes.

Along similar lines, Figure 4.6 (right) presents ablations for shares sold by the TradeR agent with and without the provision of hold action. The agent favors holding values over longer time horizons and selling them once the suitable price quote is encountered. This allows the agent to sell more shares at lower prices rather than simply buying shares and accumulating assets in hopes for encountering a rewarding quote. Similar to Figure 4.5 (bottom-left), this behavior of agent is parallel to human brokers who prefer to wait for suitable quote and sell shares in large quantities.

4.5 Summary

Application of RL is hindered by practical challenges such as *catastrophy* and *surprise minimization*. In order to address these challenges, the study focussed on the application of trading. This chapter introduced the TradeR

framework which is a hierarchical RL algorithm for practical high-frequency trading consisting of abrupt price variations. The lower level policy estimates order quantities on the basis of which the higher level policy executes trading bids in a data-driven market environment. TradeR utilizes an energy-based intrinsic motivation scheme in conjunction with surprise value function for estimating and minimizing surprise. A large-scale study of 35 stock symbols obtained at 1 minute intervals from the S&P500 index during 2019 fiscal year COVID19 market crash presents suitability of the hierarchical trading scheme. Furthermore, the ablation study conducted on payoffs and shares highlights the need for energy-based scheme in facilitating practical human-like trading patterns.

Chapter 5

Conclusions

5.1 Discussion

Practical applications of RL are faced with many challenges. This thesis explored 3 such hindrances in light of hierarchical RL.

Firstly, we looked at evolutionary RL through the lens of scalable control wherein a population of actor agents is utilized to learn behaviors. Each offspring of the population is considered a part of a staged hierarchy which grows linearly over time. Hierarchical formulation of evolutionary RL provides a computationally-efficient outlook to the learning paradigm. ESAC addresses the problem of scalability arising from mutation-sensitive evolutions. The scheme introduces AMT which maximizes the mutation rate of evolutions in a small clipped region as the SAC updates are exponentially decayed. ESAC demonstrates improved performance on a variety of locomotion tasks consisting of high-dimensional action spaces and sparse rewards. Additionally, ESAC presents scalability comparable to ES, reducing the average wall-clock time per episode by approximately 60%. While evolutionary RL demonstrates scalability in terms of control dimensions, it does not account for the growing number of agents in population. This forces one to rethink hierarchies in light of multi-agent learning.

The work continued to explore hierarchies consisting of multiple agents acting simulatenously. This leads to a uniform hierarchical framework wherein all agents are placed at the same level and must collaborate to achieve a common objective. Application of MARL is hindered by surprise-based state spaces and fast-paced dynamics which cripple the operation of various agents in the hierarchy. EMIX presents a novel energy-based surprise minimization objective consisting of an energy operator in conjunction with the surprise value function. The EMIX objective satisfies theoretical guarantees of total energy and surprise minimization with experimental results validating these claims. Additionally, EMIX suitably tackles overestimation bias across agents in MARL. Suitability of the energy-based scheme is validated by its ability to drive the multi-agent hierarchy towards consistent and stable performance.

Utilization of EMIX as a hierarchical RL scheme is limited to online learning and standard evaluation benchmarks [89]. Tackling abrupt dynamics requires attention from a practical standpoint.

The quest for practical extensions of RL in the face of its challenges led to the pivotal contribution of this thesis. Much like humans, RL agents succumb to the degree of catastophy and surprise arising from abrupt transitions in the world. The problem requires attention from a proof-of-concept mechanism. Towards this goal, the thesis focussed on the application of trading. The TradeR framework presents a hierarchical RL algorithm for high-frequency trading. The lower level policy estimates order quantities on the basis of which the higher level policy executes trading bids in a data-driven market environment. TradeR utilizes the energy-based intrinsic motivation scheme introduced by EMIX for minimizing surprise. The large-scale success of TradeR on S&P500 index during 2019 fiscal year COVID19 market crash presents the suitability of hierarchical RL to practical problems. Analogous to humans, TradeR adopts safe bidding strategies which lead the agent to profitable outcomes in long-horizon.

RL is a fast-moving and dynamically adapting area of research. Introduction of sophisticated agents and data-efficient frameworks has motivated extensions of RL towards hierarchical learning. Adoption of hierarchies to practical scenarios has witnessed limited yet continual progress. This thesis is an attempt to steer research towards this direction and motivate the application of RL to practical problems.

5.2 Future Work

Extending Evolutionary Methods: Evolutionary RL presents a suitable method for scaling control-based schemes to high dimensional action spaces. However, the ESAC algorithm has a few limitations in light of its practical execution. High variance of the SAC agent under sparse rewards requiring consistent optimal behavior hurts the performance of ESAC. One could address this by combining the framework with a meta-controller or using a more sophisticated architecture such as a master-slave framework [100]. This would provide the agent with additional ability to learn and control its learning simultaneously. Meta-controllers are an active area of research and pave the way for potential future directions in light of Meta Learning.

Energy-based Hierarchies: EMIX serves as a practical example of energy-based models in cooperative MARL. Utilization of an explicit surprise value function allows the agent to esimate and tackle surprise from a long-horizon perspective. Surprise minimization can be further extended towards opponent-aware and hierarchical MARL wherein agents deal with a greater degree of stochasticity under fast-paced dynamics. This extension of energy-based models would aid in gaining an understanding of MARL in practical settings such

as safety control and sensitivity analysis. Other potential directions include multi-level MARL and extension of hierarchies towards increasingly complex action spaces.

Practical Hierarchical Reinforcement Learning: The trade execution setup serves as a practical motivation of application of RL to real-world scenarios. Provision of a bi-level hierarchy in conjunction with previously presented surprise minimization highlights the suitability of TradeR in sequential decision-making. Although TradeR presents a suitable mechanism for trading, the method suffers from two shortcomings. Firstly, the TradeR framework requires a large amount of data for training and evaluation on price-based experiences. While humans learn to trade intuitively with minimal (5-10 symbols) data requirements, TradeR requires large number of price transitions to understand evasion of surprising states. Secondly, transition data from the stock exchange is held static during TradeR's learning phase. This is not entirely reflective of an ever-moving market which consists of multiple brokers (*agents*) continuously competing in the trading market (*environment*). A more realistic setting would include trading as a multi-agent problem and tackle non-stationarity of dynamics. These challenges present various alternative directions for future work.

Appendix A

Evolution-based Soft Actor-Critic (ESAC)

A.1 Derivation

The AMT and ES update rules are given as

$$\sigma_{(t)} \leftarrow \sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)})$$

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t)}} \sum_{i=1}^{N} R_i \epsilon_i$$

Using the expression for $\sigma_{(t)}$ in the ES update yields the following

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n(\sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}\Lambda_{(t-1)}} \sum_{i=1}^{n} R_i \epsilon_i$$

where $\Lambda_{(t-1)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)})$. Expanding $\sigma_{(t-1)}$ using the AMT update rule yields

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}(\sigma_{(t-2)} + \frac{\alpha_{es}}{n\sigma_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}\sigma_{(t-2)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}\sigma_{(t-2)}\Lambda_{(t-2)}} \sum_{i=1}^{n} R_i \epsilon_i$$

where $\Lambda_{(t-2)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)})$. Expanding this recursively yields the following

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\Lambda_{(t-1)}\Lambda_{(t-2)}...\Lambda_{(1)}} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)} \prod_{t'=1}^{t-1} \Lambda_{(t')}} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\hat{\Lambda}} \sum_{i=1}^{n} R_i \epsilon_i$$

Hence, yielding the AMT update in the form of initial mutation rate $\sigma_{(1)}$.

A.2 Policy Improvement

This section shows that AMT improves the policy of winners in the population. Consider two successive gradient intervals g indexed by (l) and $(l-1)$. Let $p_{(l)}$ and $p_{(l-1)}$ be the probabilities of convergence to the optimal policy $\pi^*_{\theta_{es}}(a_t|s_t)$ in the weight space at (l) and $(l-1)$ respectively.

The formulation starts by evaluating the mutation rates at (l) and $(l-1)$ which are given as $\sigma_{(l)} > \sigma_{(l-1)}$. We can now evaluate the probabilities of convergence to $\pi^*_{\theta_{es}}(a_t|s_t)$ as

$$p_{(l)} \geq p_{(l-1)}$$

Using this fact, one can evaluate the winners (indexed by q) in the sorted

reward population F.

$$\sum_{q=1}^{w} p_{(l)}^{(q)} F_{(l)}^{(q)} \geq \sum_{q=1}^{w} p_{(l-1)}^{(q)} F_{(l-1)}^{(q)}$$
$$= \mathbf{E}_{F_{(l)}^{(q)} \sim F_{(l)}}[F_{(l)}^{(q)}] \geq \mathbf{E}_{F_{(l-1)}^{(q)} \sim F_{(l-1)}}[F_{(l-1)}^{(q)}]$$
$$= \mathbf{E}[W_{(l)}] \geq \mathbf{E}[W_{(l-1)}]$$

Here, $p_{(l)}^{(q)}$ is the probability of convergence of actor q (having observed reward $F_{(l)}^{(q)}$) to its optimal policy $\pi_{\theta_{es}}^{(q),*}(a_t|s_t)$ at interval (l). $W_{(l)}$ represents the set of winners at (l). The mathematical expression obtained represents that the set of winners $W_{(l)}$ formed at the next gradient interval (l) is at least as good as the previous set of winners $W_{(l-1)}$, i.e.- $\pi_{\theta_{es},(l)}^{(q)}(a_t|s_t) \geq \pi_{\theta_{es},(l-1)}^{(q)}(a_t|s_t)$. This guarantees policy improvement among winners of the population.

A.3 Additional Results

The study evaluates the performance and sample-efficiency of ESAC on 9 MuJoCo and 6 DeepMind Control Suite [105] tasks. Figure Figure A.1 presents learning behavior of ESAC in comparison to SAC, TD3, PPO and ES on all 15 tasks. Training setup for all agents was kept same with different values of hyperparameters (presented in section A.4). ESAC demonstrates improved returns on 10 out of 15 tasks as presented in Table 3.1. Results are averaged over 5 random seeds with Humanoid experiments evaluated for 10 million steps.

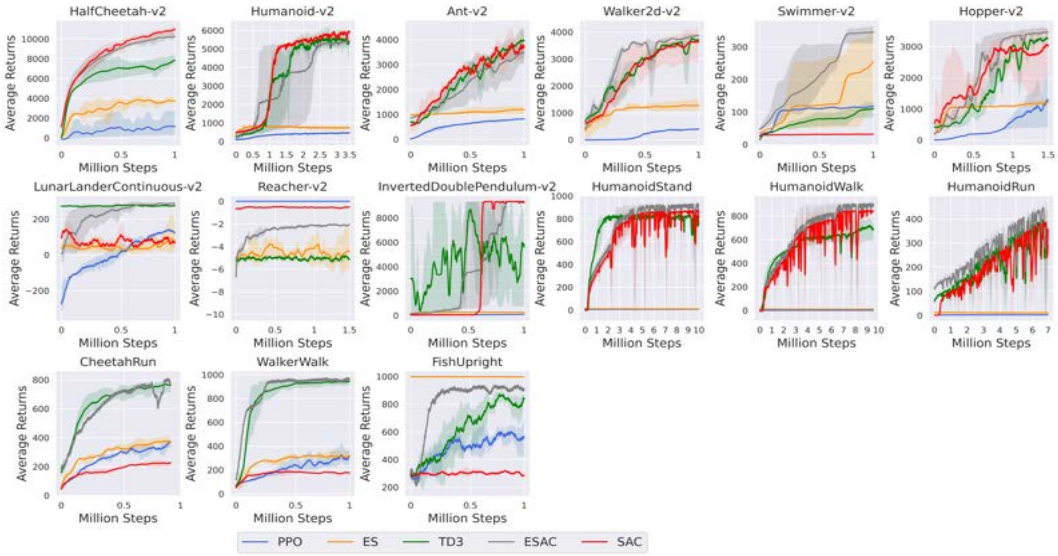

Figure A.1: Average Returns on MuJoCo and DeepMind Control Suite tasks.

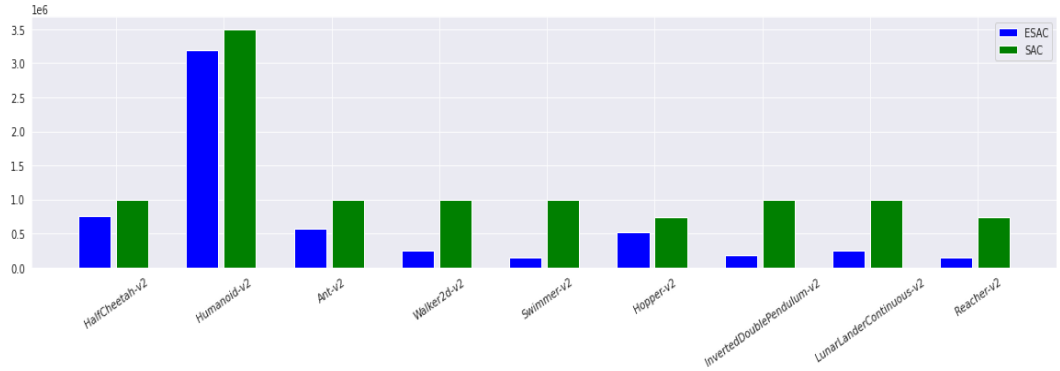

Figure A.2: Complete results on the number of backprop updates.

A.4 Hyperparameters

MuJoCo: Hyperparameter values for experiments are adjusted on the basis of complexity and reward functions of tasks. In the case of MuJoCo control tasks, training-based hyperparameters are kept mostly the same with the number of SAC episodes varying as per the complexity of task. Tuning was carried out on HalfCheetah-v2, Ant-v2, Hopper-v2 and Walker2d-v2 tasks. Out of these, Ant-v2 presented high variance indicating the requirement of a lower learning rate. Number of SAC updates in SAC and ESAC implementations were kept 1 for a fair comparison with TD3. All tasks have a common discount factor $\gamma = 0.99$, SAC learning rate $\alpha = 3 \times 10^{-4}$, population size $n = 50$, mutation rate $\sigma = 5 \times 10^{-3}$ and winner fraction $e = 0.4$. ES learning rate α_{es} was kept fixed at 5×10^{-3} for all tasks except Ant-v2 having $\alpha_{es} = 1 \times 10^{-4}$.

The only variable hyperparameter in experiments is number of SAC episodes executed by the SAC agent. Although the ESAC population in general is robust to its hyperparameters, the SAC agent is sensitive to the number of episodes. During the tuning process, the number of episodes were kept constant to a value of 10 for all the tasks. However, this led to inconsistent results on some of the environments when compared to SAC baseline. As a result, tuning was carried out around this value to obtain optimal results corresponding to each task. The SAC agent executed a total of 10 episodes for each gradient interval g for Ant-v2, Walker2d-v2, Hopper-v2 and Humanoid-v2 tasks; 5 episodes for HalfCheetah-v2, LunarLanderContinuous-v2, Reacher-v2 and InvertedPendulum-v2 tasks; and 1 episode for Swimmer-v2 task.

DeepMind Control Suite: The DeepMind control suite presents a range of tasks with sparse rewards and varying complexity for the same domain. Hyperparameter values for these tasks are different from that of MuJoCo control tasks. Tuning was carried on the Cheetah, Quadruped and Walker tasks in order to obtain sample-efficient convergence. In the case of SAC,

granularity of hyperparameter search was refined in order to observe consistent behavior. However, different values of temperature parameter produced varying performance. As in the MuJoCo case, implementations kept the number of updates fixed to 1 in order to yield a fair comparison with TD3.

All tasks have a common discount factor $\gamma = 0.99$, SAC learning rate $\alpha = 3 \times 10^{-4}$, population size $n = 50$, mutation rate $\sigma = 1 \times 10^{-2}$, winner fraction $e = 0.4$ and ES learning rate 1×10^{-2}. As in the case of MuJoCo tasks, the SAC is found to be sensitive to the number of episodes during the gradient interval g. These were kept constant at 5 and then tuned around this value for optimal performance. Final values of the SAC episodes were 5 for CartpoleSwingup, WalkerWalk and WalkerRun tasks and 1 for the remaining tasks.

Appendix B

Energy-based MIXer (EMIX)

B.1 Proofs

Theorem 1. *Given a surprise value function* $V^a_{surp}(s,u,\sigma)\forall a \in N$*, the energy operator* $\mathcal{T}V^a_{surp}(s,u,\sigma) = \log \sum_{a=1}^{N} \exp\left(V^a_{surp}(s,u,\sigma)\right)$ *forms a contraction on* $V^a_{surp}(s,u,\sigma)$*.*

Proof. The formulation first defines a norm on surprise values $||V_1 - V_2|| \equiv \max_{s,u,\sigma} |V_1(s,u,\sigma) - V_2(s,u,\sigma)|$. Suppose $\epsilon = ||V_1 - V_2||$,

$$\log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \leq \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma) + \epsilon\right)$$

$$= \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \leq \log \exp\left(\epsilon\right) \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right)$$

$$= \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \leq \epsilon + \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right)$$

$$= \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) - \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right) \leq ||V_1 - V_2|| \tag{B.1}$$

$$= \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) - \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right) < \gamma ||V_1 - V_2|| \quad (\text{as } \gamma < 1) \tag{B.2}$$

Similarly, using ϵ with $\log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right)$,

$$\log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)+\epsilon\right) \geq \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right)$$
$$= \log \exp\left(\epsilon\right) \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \geq \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right)$$
$$= \epsilon + \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \geq \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right)$$
$$= ||V_1 - V_2|| \geq \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right) - \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \tag{B.3}$$
$$\gamma||V_1 - V_2|| > \log \sum_{a=1}^{N} \exp\left(V_2(s,u,\sigma)\right) - \log \sum_{a=1}^{N} \exp\left(V_1(s,u,\sigma)\right) \quad (\text{as } \gamma < 1) \tag{B.4}$$

Results in Equation B.2 and Equation B.4 prove that the energy operation is a contraction. □

Theorem 2. *Upon agent's convergence to an optimal policy π^*, total energy of π^*, expressed by E^* will reach a thermal equilibrium consisting of minimum surprise among consecutive states s and $s^{'}$.*

Proof. The proof begins by initializing a set of M policies $\{\pi_1, \pi_2..., \pi_M\}$ having energy ratios $\{E_1, E_2..., E_M\}$. Consider a policy π_1 with surprise value function V_1. E_1 can then be expressed as

$$E_1 = \log[\frac{\sum_{a=1}^{N} \exp\left(V_1^a(s^{'},u^{'},\sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V_1^a(s,u,\sigma)\right)}]$$

Assuming a constant surprise between s and $s^{'}$, One can express $V_1^a(s^{'},u^{'},\sigma^{'}) = V_1^a(s,u,\sigma) + \zeta_1$ where ζ_1 is a constant. Using this expression in E_1 we get,

$$E_1 = \log[\frac{\sum_{a=1}^{N} \exp\left(V_1^a(s,u,\sigma) + \zeta_1\right)}{\sum_{a=1}^{N} \exp\left(V_1^a(s,u,\sigma)\right)}]$$
$$E_1 = \log[\frac{\exp\left(\zeta_1\right) \sum_{a=1}^{N} \exp\left(V_1^a(s,u,\sigma)\right)}{\sum_{a=1}^{N} \exp\left(V_1^a(s,u,\sigma)\right)}]$$
$$E_1 = \zeta_1$$

Similarly, $E_2 = \zeta_2$,$E_3 = \zeta_3$...,$E_M = \zeta_M$. Thus, the energy residing in policy π is proportional to the surprise between consecutive states s and $s^{'}$. Clearly, an

optimal policy π^* is the one with minimum surprise. Mathematically,

$$\pi^* \geq \pi_1, \pi_2 ..., \pi_M \implies \zeta^* \leq \zeta_1, \zeta_2 ..., \zeta_M$$
$$= \pi^* \geq \pi_1, \pi_2 ..., \pi_M \implies E^* \leq E_1, E_2 ..., E_M$$

Thus, proving that the optimal policy consists of minimum surprise at thermal equilibrium. □

B.2 Connection between EMIX and Soft Q-Learning

The Soft Q-Learning objective with $V^{\theta^-}_{soft}(s^{'})$ and $Q_{soft}(u, s; \theta)$ as state and action value functions respectively is given by-

$$J_Q(\theta) = \mathbb{E}_{s,u\sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s'\sim R}[V^{\theta^-}_{soft}(s^{'})] - Q_{soft}(u, s; \theta))^2]$$
$$= J_Q(\theta) = \mathbb{E}_{s,u\sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s'\sim R}[\log \sum_{u\in A} \exp Q(u^{'}, s^{'}; \theta^-)] - Q_{soft}(u, s; \theta))^2]$$

The gradient of this objective can be expressed as-

$$\nabla_\theta J_Q(\theta) = \mathbb{E}_{s,u\sim R}[(r + \gamma \mathbb{E}_{s'\sim R}[\log \sum_{u\in A} \exp Q(u^{'}, s^{'}; \theta^-)] - Q_{soft}(u, s; \theta))]\nabla_\theta Q_{soft}(u, s; \theta) \quad \text{(B.5)}$$

And the gradient of the EMIX objective is obtained as-

$$L(\theta) = \mathbb{E}_{s,u,s'\sim R}[\frac{1}{2}(r + \gamma \max_{u'} \min_i Q_i(u^{'}, s^{'}; \theta^-) + \beta \log(\frac{\sum_{a=1}^{N} \exp(V^a_{surp}(s^{'}, u^{'}, \sigma^{'}))}{\sum_{a=1}^{N} \exp(V^a_{surp}(s, u, \sigma))}) - Q(u, s; \theta)^2)]$$

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,u,s'\sim R}[(r + \gamma \max_{u'} \min_i Q_i(u^{'}, s^{'}; \theta^-) + \beta \log(\frac{\sum_{a=1}^{N} \exp(V^a_{surp}(s^{'}, u^{'}, \sigma^{'}))}{\sum_{a=1}^{N} \exp(V^a_{surp}(s, u, \sigma))}) - Q(u, s; \theta))]\nabla_\theta Q(u, s; \theta) \quad \text{(B.6)}$$

Comparing Equation B.5 to Equation B.6 the reader may notice that Soft Q-Learning and EMIX are related to each other as they utilize energy-based models. Soft Q-Learning makes use of a discounted energy function which downweights the energy values over longer horizons. Actions consisting of lower energy configurations are given preference by making use of $Q_{soft}(u, s; \theta)$ as the negative energy. On the other hand, EMIX makes use of a constant energy function weighed by β which minimizes surprise-based energy between

consecutive states. Both the objectives can be thought of as energy minimizing models which search for an optimal energy configuration. Soft Q-Learning searches for an optimal configuration in the action space whereas EMIX favours optimal behavior on spurious states. In fact, EMIX can be realized as a special case of Soft Q-Learning if the mixer agent utilizes an energy-based policy and attains thermal equilibrium. This leads us to express Theorem 3.

Theorem 3. *Given an energy-based policy π_{en} with its target function $V(s^{'}) = \log \sum_{u \in A} \exp Q(u^{'}, s^{'}; \theta^{-})$, the surprise minimization objective $L(\theta)$ reduces to the Soft Q-Learning objective $L(\theta_{soft})$ in the special case when the variational free energy function $\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)$ is equal to the partition function $\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s, u, \sigma)\right)$.*

Proof. The EMIX objective is given by-

$$L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r + \gamma \max_{u^{'}} \min_{i} Q_i(u^{'}; s^{'}, \theta^{-}) + \beta \log(\frac{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s, u, \sigma)\right)}) - Q(u, s; \theta)^2)]$$

Replacing the greedy policy term $\max_{u^{'}} \min_{i} Q_i(u^{'}, s^{'}; \theta^{-})$ with the energy-based value function $V(s^{'}) = \log \sum_{u' \in A} \exp Q(u^{'}, s^{'}; \theta^{-})$, we get,

$$L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s' \sim R}[V(s^{'})] + \beta \log(\frac{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s, u, \sigma)\right)}) - Q(u, s; \theta)^2)]$$

$$= L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s' \sim R}[\log \sum_{u' \in A} \exp Q(u^{'}, s^{'}; \theta^{-})] + \beta \log(\frac{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s, u, \sigma)\right)}) - Q(u, s; \theta)^2)]$$

At thermal equilibrium, $\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s, u, \sigma)\right) = \sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)$,

$$= L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s' \sim R}[\log \sum_{u' \in A} \exp Q(u^{'}, s^{'}; \theta^{-})] + \beta \log(\frac{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)}{\sum_{a=1}^{N} \exp\left(V_{surp}^{a}(s^{'}, u^{'}, \sigma^{'})\right)}) - Q(u, s; \theta)^2)]$$

$$= L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r+\gamma \mathbb{E}_{s' \sim R}[\log \sum_{u' \in A} \exp Q(u', s'; \theta^-)] + \beta \log(1) - Q(u, s; \theta)^2)]$$

$$= L(\theta) = \mathbb{E}_{s,u,s' \sim R}[\frac{1}{2}(r + \gamma \mathbb{E}_{s' \sim R}[\log \sum_{u' \in A} \exp Q(u', s'; \theta^-)] - Q(u, s; \theta)^2)] \tag{B.7}$$

Equation B.7 represents the Soft Q-Learning objective, hence proving the result. □

B.3 Complete Results

This section presents the complete results of EMIX agents for all the 12 scenarios considered in StarCraft II micromanagement. While some scenarios depict significant performance improvements, other scenarios present incremental gains as a result of early surprise minimization during the exploration phase.

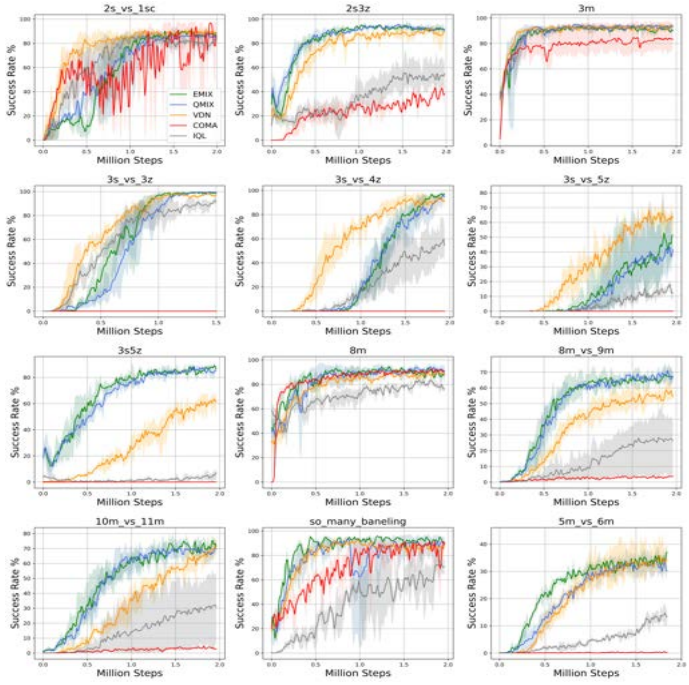

Figure B.1: Learning comparison of success rate percentages between EMIX and state-of-the-art MARL methods for all StarCraft II micromanagement scenarios.

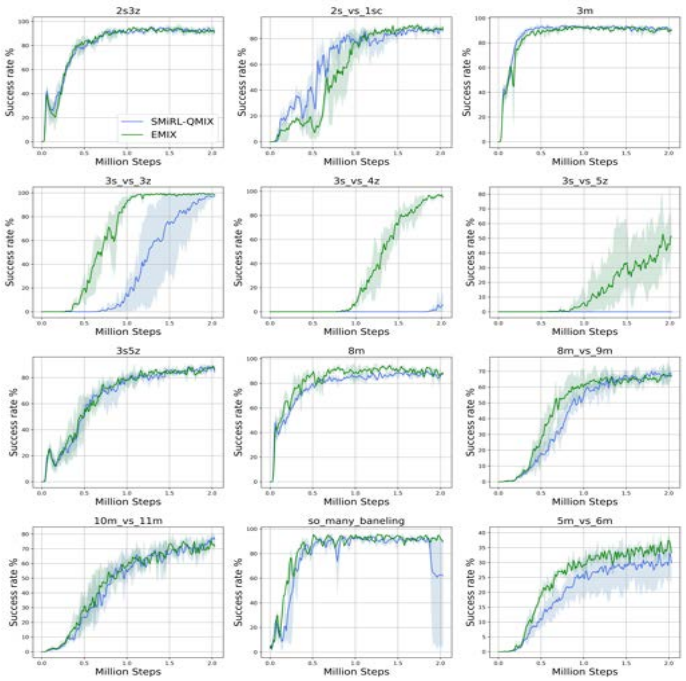

Figure B.2: Learning comparison of success rate percentages between EMIX and SMiRL-QMIX for all StarCraft II micromanagement scenarios.

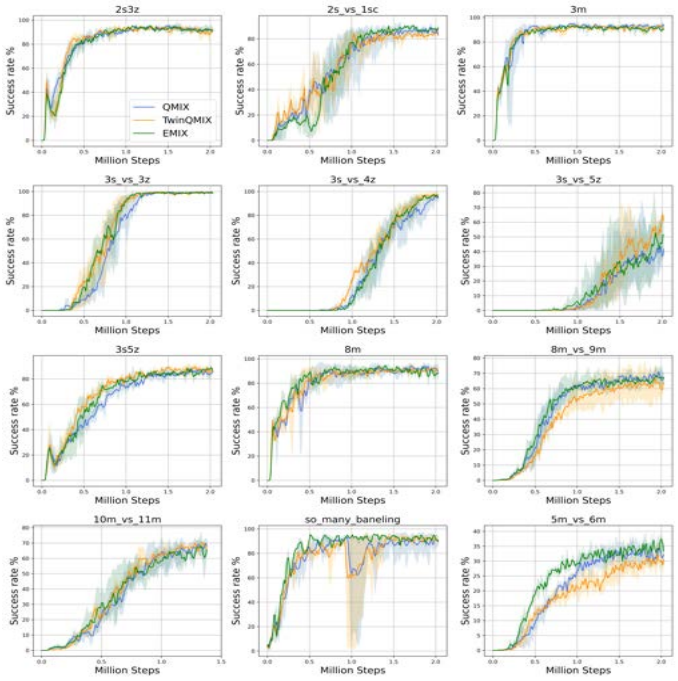

Figure B.3: Comparison of success rate percentages between EMIX, TwinQMIX (EMIX without surprise minimization) and QMIX for all 12 StarCraft II micromanagement scenarios.

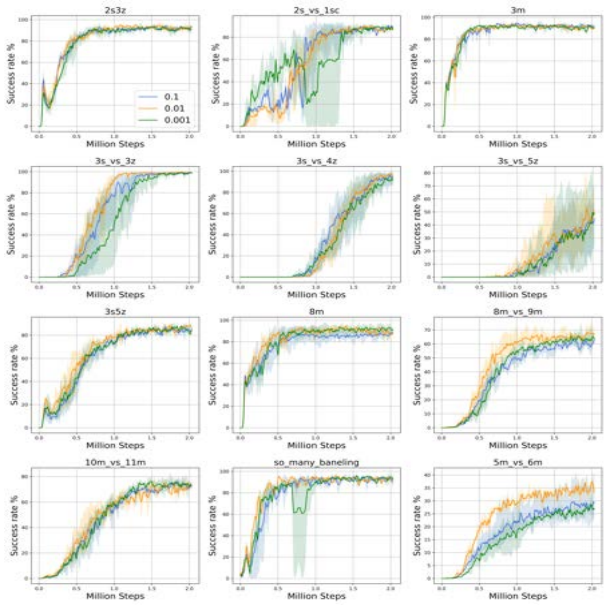

Figure B.4: Comparison of success rate percentages with different β values for EMIX on all 12 StarCraft II micromanagement scenarios.

B.4 Implementation Details

Model Specifications: This section highlights model architecture for the surprise value function. At the lower level, the architecture consists of 3 independent networks called *state_net*, *q_net* and *surp_net*. Each of these networks consist of a single layer of 256 units with ReLU non-linearity as activations. Outputs from each of the networks are concatenated and are provided as input to the *main_net* consisting of 256 units with ReLU activations. The *main_net* yields a single output as the surprise value $V^a_{surp}(s, u, \sigma)$ which is reduced along the agent dimension by the energy operator.
Hyperparameters: Table B.1 presents hyperparameter values for EMIX. Value of β was tuned between 0.001 and 1 in intervals of 0.01 with best performance observed at $\beta = 0.01$.

Hyperparameters	Values
batch size	$b = 32$
learning rate	$\alpha = 0.0005$
discount factor	$\gamma = 0.99$
target update interval	200 episodes
mixer embedding size	32
temperature	$\beta = 0.01$
target Q-functions	2

Table B.1: Hyperparameter values for EMIX agents

Appendix C

TradeR Details

C.1 Implementation Details

This section highlights the setup for training and evaluating the TradeR framework. The implementation combines TradeR with Proximal Policy Optimization (PPO) consisting of an Actor-Critic framework [46, 67]. The Actor consists of order and bid networks as two seperate policies. Order quantity y_t estimated by the order network using policy $\pi_{ord}(y_t|s_t)$ is concatenated with the environment state s_t and provided as input state to the bid network. The bid network utilizes the high-level bid policy $\pi_{bid}(a_t|s_t, y_t)$ to estimate the final bid a_t as environment action. The agent, consisting of order and bid policies, is trained jointly using policy gradient framework provisioned by the critic. During policy gradient updates, the surprise value function is utilized to estimate surprise using standard deviations across state dimensions which are then operated upon by the Mellowmax operator. An alternate method for applying the surprise value function would be to estimate surprise corresponding to each state. However, it was empirically found this technique demonstrates high variance in its estimates and often steers the agent towards surprising states. In order to further stabilize training, the method adds a small penalty to agent's loss function in the form of energy-based surprise estimates. This ensures that information about surprising states is propagated directly into actor's policies which may not be the case for some symbols in the training benchmark.

S&P 500 Data: Market data utilized for learning and evaluation of TradeR was collected from the S&P500 market index. This stock market index is similar to other global market indices such as Dow and Jones 100 index. The study specifically selects S&P500 as a benchmark as it cosists of a wider variety of stock symbols and suitably demonstrates global market trends. The data collected consists of price and volume transitions which are provided as state to the agent. More specifically, the following transitions were collected and presented to the agent at each timestep-
Open Price: The opening price of stock symbol in the market for current day.

Close Price: The closing price of stock symbol in the market for past day.
Low Price: The lowest price observed for the stock symbol in the market during current day.
High Price: The highest price observed for the stock symbol in the market during current day.
Volume: The quantity of shares which changed hands between the current and past timesteps.

All prices were observed in US Dollars ($). Transitions were observed at 1 minute intervals from 35 stock symbols during the 2019 fiscal year ranging from 1^{st} April 2019 to 31^{st} March 2020. This includes the COVID19 market crash which was observed in February 2020. The study selects 35 stock symbols (20 best and 15 worst) from the market as these were reflective of majority of symbols in the market during the later half of 2019.

Computational Requirements: The agent was trained for a total of 5 million steps over 5 random runs for each symbol. This results in 6 GPU hours of training for a single run of one symbol. The total computational cost of training TradeR was 30 GPU hours for one symbol and 1050 GPU hours for all 35 symbols. In the interest of time, the setup parallelized computation on 4 NVIDIA RTX2080 Titan GPUs which reduced the time to 262.5 GPU hours resulting in 11 days of training. However, the major computational bottleneck resulted from the ablation study of TradeR. A total of 5 ablation experiments in addition to TradeR experiments were performed which increased the computation sixfold. Thus, the total computational time required for training and evaluating TradeR was $(6 \times 262.5 = 1575)$ GPU hours resulting in 66 days of experiments. Each training run consisted of 500 episodes of 10^4 timesteps with policy gradient updates carried out at every 100 steps. The implementation tried longer timesteps of $12,000$ and $15,000$ but these did not demonstrate significant robustness to variance in agent's performance. In the case of $15,000$ timesteps, the agent often steered away from its consistent behavior and incurred lossy outcomes which were otherwise found absent in the 10^4 timesteps setting.

Training Details: The order network consists of an input layer of 128 units with tanh nonlinearity, a hidden layer of 128 units with tanh nonlinearity and an output layer of 1 unit with sigmoid nonlinearity. The bid network has a similar architecture except that the output layer has 3 units each corresponding to its 3 bid actions of buy, sell or hold. Action outputs from both networks are concatenated to yield the final 2-dimensional vector comprising of an order quantity in the range $[0, 1]$ and a discrete action from the set $\{0, 1, 2\}$ denoting the type of bid. The architecture for critic network is same as that of order network. In the case of surprise value function, the implementation adopts a different architecture. The two-layered network consists of an input layer of 128 units with ReLU [73] nonlinearity followed by an output layer having

number of units equivalent to the number of state dimensions.

Algorithm 4 Trading Environment

Input: S, a_t
Output: r_t, s_{t+1}
Initialize account_balance, max_account_balance, max_num_shares, max_share_price, max_open_positions
Initialize net_worth, shares_held, shares_sold, cost_basis, sales_value
/ / / / / / / / / / / / FETCH NEXT OBSERVATION FROM SEQUENCE
function OBSERVATION(S)
    $obs \leftarrow \frac{S[-5:]}{max\_share\_price}$
    $obs \leftarrow (obs|[balance, net\_worth, shares\_held, shares\_sold, cost\_basis, sales\_value])$
    return obs
end function
/ / / / / / / / / / / / EXECUTE ACTION BY BUYING/SELLING SHARES
function STEP(S, a_t)
    if $a_t == 0$ **then**:
        $shares\_bought \leftarrow \frac{balance}{S[-1]} \times amount$
        $prev\_cost \leftarrow cost\_basis \times shares\_held$
        $add\_cost \leftarrow shares\_bought \times S[-1]$
        $balance \leftarrow balance - add\_cost$
        $cost\_basis \leftarrow \frac{prev\_cost + add\_cost}{shares\_held + shares\_bought}$
        $shares\_held \leftarrow shares\_held + shares\_bought$
    else if $a_t == 1$ **then**:
        $shares\_sold \leftarrow shares\_held \times amount$
        $balance \leftarrow balance + shares\_sold \times current\_price$
        $shares\_held \leftarrow shares\_held - shares\_sold$
        $sales\_value \leftarrow shares\_sold \times current\_price$
    end if
    $net\_worth \leftarrow balance + shares\_held \times current\_price$
    $reward \leftarrow balance$
    $obs \leftarrow OBSERVATION(S)$
    return reward, obs
end function
/ / / / / / / / / / / / RESET ENVIRONMENT TO A RANDOM STATE
function RESET(S)
    $balance \leftarrow account\_balance$
    $net\_worth \leftarrow account\_balance$
    $shares\_held \leftarrow 0$
    $shares\_sold \leftarrow 0$
    $sales\_value \leftarrow 0$
    $cost\_basis \leftarrow 0$
    $obs \leftarrow observation(S)$
    return obs
end function

Execution Setup: The agent interacts in a custom-designed trading environment which replicates the buying and selling of shares in real global markets. The motivation behind the environment stems from Capital Markets framework

which consists of various different order bids being processed at the same time. Figure 4.2 presents the dynamic trading environment used to simulate real-world market data. The price sequence corresponding to a symbol is obtained from the stock exchange which is stored in a prefetch buffer. Similar to price quotes, the symbol metadata (date-time intervals and volume transitions) are appended to the sequence variable in the prefetch buffer. A batch of transitions is sampled from the prefetch buffer and preprocessed using the transition preprocessor. Preprocessing steps on a batch include filtering and sorting of price quantities and volume vectors according to their date-time interval. The transition preprocessor feeds the processed transitions into a transition queue which preserves the temporal structure of prices as in the real stock market. Each entry in queue is sampled for normalization and scaling as per the maximum quantity in price vectors. This is an additional step in comparison to the real market environment in order to facilitate accurate approximations and lower variance in order quantities of the agent. Normalized and scaled transitions are passed to the action simulator which simulates the transitions using internal local variables based on actions selected by the agent. At each timestep, the action simulator queries the reward function and yields the next state s_{t+1} and reward r_t to the agent. An additional technique to speed up learning would be to make use of multiple action simulators in an asynchronous fashion which obeys the real market trading constraints. However, this requires multiple actions to be queried by the agent while updating its policy resulting in large computational requirements.

Algorithm 2 presents the trading setup used for simulating a real-world market scenario. Stock sequence S corresponding to a symbol is traversed on the basis of shares bought and sold during agent's execution. Following are the three key functions-
RESET- This method resets the environment to an initial random state with all data variables initialized to 0.
OBSERVATION- This method processes the observation variables *obs* by combining micro-level observations such as price quotes with macro-level observations such as the current balance, net worth, total shares held and sold by the agent. The combined observation is 36-dimensional vector which represents the state of the agent.
STEP- The STEP method transitions the agent from current to next timestep. Based on the action a_t selected by the agent, the function updates macro-level observations which are stored in a queue-like fashion. Following macro-level update of dynamics, the method provides a scalar reward value proportional to the change in prices which leads to the next-state obtained using OBSERVATION method.

Note on Comparison with Soft Actor Critic (SAC): In orde to assess the efficacy of TradeR with state-of-the-art RL methods, experiments tried to implement SAC on the real-market trading environment. However, SAC

presented significantly high variance in its returns with no meaningful trading strategies learned over the course of training. To address this issue, the entropy parameter of SAC was tuned for 5 different values between 0 and 0.5 in steps of 0.05. While lower values of entropy parameter demonstrated learning of algorithm, the returns obtained by SAC agent were not comparable to any of the baseline agents and not consistent across its random seeds. The study conjectures that SAC does not perform well under abrupt dynamics as significant deviations between consecutive states hurt the entropy of action distribution. Comparison of TradeR with SAC is left as a potential direction for future work.

C.2 Complete Results

Symbol	TradeR	PPO	TD3	DDPG
ABMD	**0.94±0.04**	0.76±0.08	0.44±0.03	0.87±0.06
ADSK	**0.04±0.01**	0.03±0.03	0.02±0.03	0.04±0.04
AMZN	0.01±0.00	0.01±0.01	0.0±0.00	**0.1±0.00**
CDNS	**0.12±0.01**	0.12±0.03	0.07±0.01	0.11±0.01
CMG	**1.14±0.1**	0.69±0.19	0.23±0.02	0.44±0.02
DVN	0.14±0.07	0.12±0.02	0.08±0.01	0.**16±0.01**
DWDP	**0.01±0.0**	0.0±0.0	0.0±0.0	0.0±0.0
DXCM	**1.33±0.17**	1.09±0.14	0.68±0.07	1.26±0.13
FANG	**1.33±0.82**	1.07±0.18	0.67±0.09	1.32±0.17
FTNT	0.07±0.01	0.06±0.0	0.04±0.0	0.**07±0.01**
HAL	0.06±0.0	0.06±0.01	0.04±0.0	0.**07±0.01**
MKTX	**0.3±0.01**	0.19±0.01	0.12±0.0	0.18±0.01
MRO	**0.01±0.01**	0.01±0.04	0.01±0.03	0.01±0.05
NVDA	**0.04±0.0**	0.03±0.0	0.02±0.0	0.03±0.0
PYPL	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
SNPS	**1.28±0.15**	1.08±0.06	0.63±0.09	1.27±0.11
SWKS	1.2±0.17	1.15±0.03	0.65±0.07	1.**34±0.16**
TSCO	1.13±0.08	1.03±0.11	0.62±0.07	1.**19±0.15**
URI	**0.29±0.04**	0.25±0.02	0.15±0.02	0.27±0.04
WST	**0.96±0.08**	0.82±0.07	0.49±0.05	0.94±0.15
AVB	**1.39±0.07**	1.09±0.09	0.65±0.05	1.27±0.67
BXP	**0.3±0.02**	0.25±0.02	0.15±0.01	0.15±0.16
CINF	**1.27±0.75**	1.14±0.1	0.69±0.07	1.23±0.72
COTY	1.14±0.07	1.16±0.18	0.71±0.07	**1.28±0.74**
DAL	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
HWM	0.57±1.43	0.58±0.01	0.34±0.01	0.**67±0.33**
JWN	1.2±0.24	1.19±0.08	0.77±0.11	**1.39±0.78**
LB	1.16±0.16	**1.23±0.15**	0.71±0.11	1.21±0.74
LOW	**1.36±0.19**	1.2±0.15	0.74±0.06	1.34±0.7
PVH	**1.32±0.12**	1.25±0.23	0.78±0.06	1.02±0.68
RL	**0.04±0.01**	0.04±0.04	0.02±0.0	0.03±0.02
SLG	0.07±0.01	0.07±0.01	0.04±0.0	0.**08±0.04**
SPG	**1.29±0.21**	1.17±0.16	0.67±0.14	1.28±0.64
TPR	**1.19±0.11**	1.14±0.12	0.69±0.1	1.17±0.73
WFC	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0

Table C.1: Normalized average rewards for all 35 symbols from the S&P500 benchmark observed at 1 minute intervals for 2019 fiscal year.

Figure C.1: Normalized average rewards learned over 5 million steps for all 35 symbols.

Symbol	TradeR	PPO	TD3	DDPG
ABMD	**0.61±0.05**	0.87±0.04	0.63±0.02	0.0±0.0
ADSK	**1.41±0.09**	1.96±0.04	1.49±0.04	0.0±0.0
AMZN	0.23±0.03	0.4±0.21	**0.03±0.0**	0.0±0.0
CDNS	**0.3±0.03**	0.41±0.01	0.32±0.0	0.0±0.0
CMG	0.28±0.07	0.41±0.09	*0.15±0.0*	0.0±0.0
DVN	**0.18±0.01**	0.28±0.04	0.23±0.02	0.0±0.0
DWDP	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
DXCM	**1.74±0.11**	2.51±0.08	1.94±0.16	0.0±0.0
FANG	**1.91±0.42**	2.63±0.45	2.18±0.24	0.0±0.0
FTNT	**1.66±0.17**	2.39±0.06	1.86±0.04	0.0±0.0
HAL	**1.34±0.33**	1.88±0.24	1.58±0.23	0.0±0.0
MKTX	1.77±0.12	2.49±0.08	**1.57±0.02**	0.0±0.0
MRO	**0.82±0.2**	1.28±0.32	1.03±0.17	0.0±0.0
NVDA	**0.07±0.0**	0.1±0.0	0.07±0.01	0.0±0.0
PYPL	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
SNPS	**1.66±0.13**	2.37±0.1	1.8±0.03	0.0±0.0
SWKS	**1.77±0.13**	2.44±0.07	1.91±0.08	0.0±0.0
TSCO	**1.59±0.16**	2.32±0.04	1.8±0.04	0.0±0.0
URI	**0.33±0.03**	0.45±0.04	0.36±0.02	0.0±0.0
WST	**0.77±0.08**	1.12±0.01	0.85±0.03	0.0±0.0
AVB	1.66±0.33	2.23±0.07	**1.64±0.04**	0.0±1.66
BXP	**1.27±0.11**	1.79±0.08	1.36±0.01	1.33±1.39
CINF	**1.37±0.09**	2.02±0.05	1.61±0.07	1.59±1.6
COTY	**1.45±0.21**	2.07±0.26	1.76±0.14	0.0±1.64
DAL	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
HWM	**1.42±0.05**	2.09±0.05	1.65±0.03	1.66±1.66
JWN	**1.54±0.03**	2.16±0.23	1.74±0.12	0.0±1.75
LB	**1.51±0.11**	2.07±0.09	1.65±0.14	0.0±1.75
LOW	**1.39±0.28**	1.93±0.04	1.48±0.02	1.49±1.47
PVH	**1.32±0.3**	1.98±0.22	1.54±0.21	1.53±0.22
RL	**0.19±0.03**	0.26±0.01	0.2±0.0	0.2±0.2
SLG	**1.56±0.17**	2.14±0.27	1.71±0.06	0.0±1.74
SPG	**1.56±0.15**	2.21±0.14	1.7±0.25	0.0±1.82
TPR	**1.52±0.08**	2.0±0.06	1.65±0.15	0.0±1.65
WFC	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0

Table C.2: Normalized average shares sold for all 35 symbols from the S&P500 benchmark observed at 1 minute intervals for 2019 fiscal year.

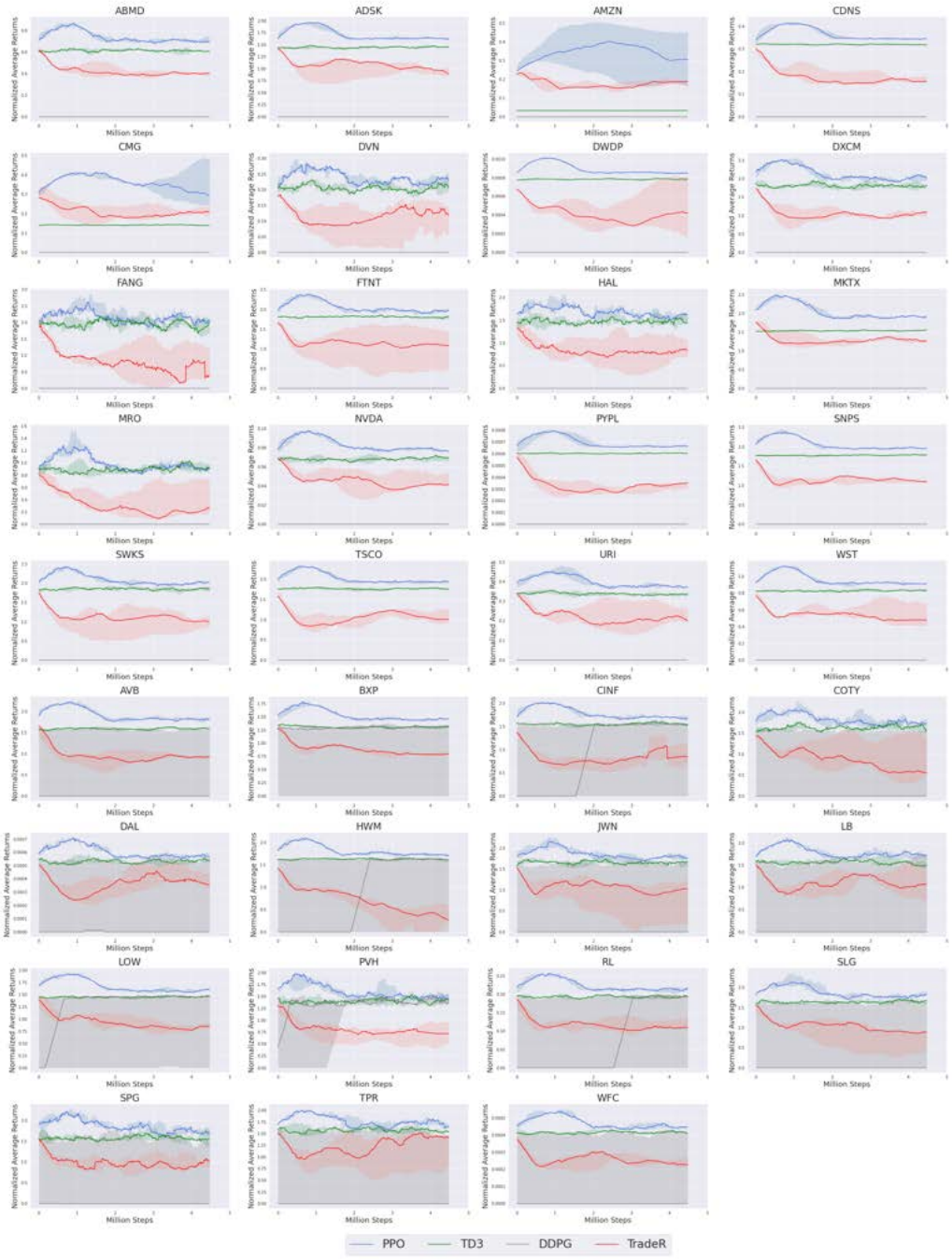

Figure C.2: Normalized average shares sold over 5 million steps for all 35 symbols.

Bibliography

[1] Joshua Achiam and Shankar Sastry. *Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning*. 2017.

[2] Johannes Ackermann et al. "Reducing Overestimation Bias in Multi-Agent Domains Using Double Centralized Critics". In: *arXiv* (2019).

[3] Kavosh Asadi and Michael L Littman. "An alternative softmax operator for reinforcement learning". In: *International Conference on Machine Learning*. 2017.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. *The Option-Critic Architecture*. 2016.

[5] Prasanna Balaprakash et al. "Scalable reinforcement-learning-based neural architecture search for cancer deep learning research". In: *Proceedings of the SC* (2019).

[6] Stefan Banach. "On operations in abstract sets and their application to integral equations". In: (1922).

[7] Marc G Bellemare et al. "Autonomous navigation of stratospheric balloons using reinforcement learning". In: *Nature* 588 (2020), pp. 77–82.

[8] Glen Berseth et al. "SMiRL: Surprise Minimizing RL in Entropic Environments". In: (2019).

[9] Ioannis Boukas et al. "A deep reinforcement learning framework for continuous intraday market bidding". In: *arXiv* (2020).

[10] Greg Brockman et al. *OpenAI Gym*. arXiv. 2016.

[11] Yuri Burda et al. "Exploration by random network distillation". In: *ICLR*. 2019.

[12] Yuri Burda et al. "Large-Scale Study of Curiosity-Driven Learning". In: *ICLR*. 2019.

[13] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "Multi-agent reinforcement learning: A survey". In: *2006 9th International Conference on Control, Automation, Robotics and Vision*. 2006.

[14] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. "Multi-agent reinforcement learning: An overview". In: *Innovations in multi-agent systems and applications-1*. 2010.

[15] Ayman Chaouki et al. *Deep Deterministic Portfolio Optimization*. 2020.

[16] Jerry Zikun Chen. *Reinforcement Learning Generalization with Surprise Minimization*. 2020.

[17] Krzysztof Ciesielski et al. "On Stefan Banach and some of his results". In: *Banach Journal of Mathematical Analysis* 1.1 (2007).

[18] Thomas G Dietterich. "Hierarchical reinforcement learning with the MAXQ value function decomposition". In: *Journal of artificial intelligence research* 13 (2000), pp. 227–303.

[19] Lasse Espeholt et al. “SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference”. In: *ICLR*. 2020.

[20] Jakob Foerster et al. *Counterfactual Multi-Agent Policy Gradients*. 2017.

[21] Justin Fu et al. “D4rl: Datasets for deep data-driven reinforcement learning”. In: *arXiv* (2020).

[22] Zipeng Fu, Qingqing Zhao, and Weinan Zhang. “Reducing Overestimation in Value Mixing for Cooperative Deep Multi-Agent Reinforcement Learning”. In: *ICAART* (2020).

[23] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *CoRR* abs/1802.09477 (2018).

[24] Ziming Gao et al. “Application of Deep Q-Network in Portfolio Management”. In: *2020 5th IEEE International Conference on Big Data Analytics (ICBDA)*. IEEE. 2020, pp. 268–275.

[25] Jordi Grau-Moya, Felix Leibfried, and Haitham Bou-Ammar. “Balancing two-player stochastic games with soft q-learning”. In: *arXiv* (2018).

[26] Abhishek Gupta et al. “Meta-Reinforcement Learning of Structured Exploration Strategies”. In: *Advances in Neural Information Processing Systems 31*. 2018.

[27] Tuomas Haarnoja. “Acquiring Diverse Robot Skills via Maximum Entropy Deep Reinforcement Learning”. PhD thesis. UC Berkeley, 2018.

[28] Tuomas Haarnoja et al. “Latent space policies for hierarchical reinforcement learning”. In: *arXiv* (2018).

[29] Tuomas Haarnoja et al. “Reinforcement learning with deep energy-based policies”. In: *arXiv* (2017).

[30] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *CoRR* abs/1801.01290 (2018).

[31] Danijar Hafner et al. *Dream to Control: Learning Behaviors by Latent Imagination*. 2019.

[32] Hado Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. Vol. 23. 2010, pp. 2613–2621.

[33] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: (2015). AAAI 2016.

[34] Hado V. Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems 23*. 2010.

[35] Hado V. Hasselt. “Double Q-learning”. In: *NIPS 23*. 2010.

[36] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[37] Matteo Hessel et al. “Rainbow: Combining improvements in deep reinforcement learning”. In: *arXiv* (2017).

[38] Rein Houthooft et al. “Evolved Policy Gradients”. In: *NIPS*. 2018, pp. 5400–5409.

[39] Peter J. Huber. “Robust estimation of a location parameter”. In: *Annals of Mathematical Statistics* 35.1 (Mar. 1964), pp. 73–101.

[40] Perttu Hämäläinen et al. “PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation”. In: *CoRR* abs/1810.02541 (2018).

[41] Lukasz Kaiser et al. *Model-Based Reinforcement Learning for Atari*. 2019.

[42] Dmitry Kalashnikov et al. "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *CoRL*. Vol. 87. PMLR. PMLR, 2018, pp. 651–673.

[43] Shauharda Khadka and Kagan Tumer. "Evolution-Guided Policy Gradient in Reinforcement Learning". In: *NIPS*. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, 1196–1208.

[44] Shauharda Khadka, Connor Yates, and Kagan Tumer. "A Memory-Based Multiagent Framework for Adaptive Decision Making". In: *17th ICAAMS*. 2018.

[45] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 3rd International Conference for Learning Representations, San Diego, 2015.

[46] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[47] Landon Kraemer and Bikramjit Banerjee. "Multi-agent reinforcement learning as a rehearsal for decentralized planning". In: *Neurocomputing* 190 (Feb. 2016).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*. Curran Associates, Inc., 2012, pp. 1097–1105.

[49] Tejas D. Kulkarni et al. "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *NIPS*. 2016, 3682–3690.

[50] Qingfeng Lan et al. "Maxmin Q-learning: Controlling the Estimation Bias of Q-learning". In: *International Conference on Learning Representations*. 2020.

[51] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1 (2006).

[52] Yann LeCun et al. "Energy-based models in document recognition and computer vision". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 1. 2007.

[53] Lisa Lee et al. "Efficient exploration via state marginal matching". In: *arXiv* (2019).

[54] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 2014.

[55] Sergey Levine et al. "Offline Reinforcement Learning: Tutorial, Review". In: *and Perspectives on Open Problems* (2020).

[56] Alexander C Li et al. "Sub-policy Adaptation for Hierarchical Reinforcement Learning". In: *arXiv* (2019).

[57] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[58] Ryan Lowe et al. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. 2017.

[59] Xueguang Lyu and Christopher Amato. "Likelihood Quantile Networks for Coordinating Multi-Agent Reinforcement Learning". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020.

[60] Luis Macedo and Amilcar Cardoso. "The role of surprise, curiosity and hunger on exploration of unknown environments populated with entities". In: *2005 portuguese conference on artificial intelligence*. 2005.

[61] Luis Macedo, Rainer Reisezein, and Amilcar Cardoso. "Modeling forms of surprise in artificial agents: empirical and theoretical study of surprise functions". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 26. 2004.

[62] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, 2002.

[63] A. Rupam Mahmood et al. "Benchmarking Reinforcement Learning Algorithms on Real-World Robots". In: *CoRR* abs/1809.07731 (2018).

[64] Parvin Malekzadeh et al. *MM-KTD: Multiple Model Kalman Temporal Differences for Reinforcement Learning*. 2020.

[65] Thomas Miconi et al. *Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity*. 2020.

[66] Dipendra Misra et al. "Kinematic State Abstraction and Provably Efficient Rich-Observation Reinforcement Learning". In: *arXiv* (2019).

[67] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016).

[68] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013).

[69] John Moody and Matthew Saffell. "Reinforcement Learning for Trading". In: *Proceedings of the 11th International Conference on Neural Information Processing Systems*. NIPS. 1998, 917–923.

[70] David E Moriarty and Risto Mikkulainen. "Efficient reinforcement learning through symbiotic evolution". In: *Machine Learning* 22.1-3 (1996), pp. 11–32.

[71] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems*. 2018, pp. 3303–3313.

[72] Ofir Nachum et al. "Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning?" In: *arXiv* (2019).

[73] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *ICML*. Ed. by Johannes Fürnkranz and Thorsten Joachims. 2010, pp. 807–814.

[74] John F. Nash. "Equilibrium points in n-person games". In: *Proceedings of the National Academy of Sciences* 36.1 (1950).

[75] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications". In: *IEEE transactions on cybernetics* (2020).

[76] Brendan O'Donoghue et al. "Combining policy gradient and Q-learning". In: *arXiv* (2016).

[77] Paolo Pagliuca, Nicola Milano, and Stefano Nolfi. *Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization*. 2019.

[78] Aloïs Pourchot and Olivier Sigaud. *CEM-RL: Combining evolutionary and gradient-based methods for policy search*. 2018.

[79] Doina Precup. "Temporal abstraction in reinforcement learning." In: (2001).

[80] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *ICML 2018: Proceedings of the Thirty-Fifth International Conference on Machine Learning*. 2018.

[81] Tabish Rashid et al. *Weighted QMIX: Expanding Monotonic Value Function Factorisation*. 2020.

[82] Wei Ren, Randal W Beard, and Ella M Atkins. "A survey of consensus problems in multi-agent coordination". In: *Proceedings of the 2005, American Control Conference, 2005.* 2005.

[83] Golden Rockefeller, Shauharda Khadka, and Kagan Tumer. "Multi-Level Fitness Critics for Cooperative Coevolution". In: *ICAAMS*. 2020.

[84] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[85] Jonathan Sadighian. "Extending Deep Reinforcement Learning Frameworks in Cryptocurrency Market Making". In: *arXiv* (2020).

[86] Tim Salimans et al. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. 2017.

[87] Tim Salimans et al. "Improved Techniques for Training GANs". In: *29th NIPS*. 2016.

[88] Brian Sallans and Geoffrey E Hinton. "Reinforcement learning with factored states and actions". In: *Journal of Machine Learning Research* 5 (2004).

[89] Mikayel Samvelyan et al. *The StarCraft Multi-Agent Challenge*. 2019.

[90] Julian Schrittwieser et al. *Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model*. 2019.

[91] John Schulman, Xi Chen, and Pieter Abbeel. "Equivalence between policy gradients and soft q-learning". In: *arXiv* (2017).

[92] John Schulman et al. "Proximal Policy Optimization Algorithms." In: *CoRR* abs/1707.06347 (2017).

[93] John Schulman et al. "Trust Region Policy Optimization". In: *ICML*. Ed. by Francis Bach and David Blei. Vol. 37. PMLR. Lille, France: PMLR, 2015, pp. 1889–1897.

[94] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489.

[95] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *arXiv* (2020).

[96] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models". In: *arXiv* (2015).

[97] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computing* 10.2 (June 2002), 99–127.

[98] Peter Stone and Manuela Veloso. "Multiagent systems: A survey from a machine learning perspective". In: *Autonomous Robots* 8 (2000).

[99] Peter Sunehag et al. "Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '18. 2018, 2085–2087.

[100] Karush Suri and Rinki Gupta. "Transfer Learning for sEMG-based Hand Gesture Classification using Deep Learning in a Master- Slave Architecture". In: *IC3I* (2018).

[101] Karush Suri et al. "Energy-based Surprise Minimization for Multi-Agent Value Factorization". In: *arXiv* (2020).

[102] Karush Suri et al. *Maximum Mutation Reinforcement Learning for Scalable Control*. 2020.

[103] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018.

[104] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *In Proceedings of the Tenth International Conference on Machine Learning*. 1993.

[105] Yuval Tassa et al. "DeepMind Control Suite". In: *CoRR* abs/1801.00690 (2018).

[106] Yee Whye Teh et al. "Energy-based models for sparse overcomplete representations". In: *Journal of Machine Learning Research* 4 (2003).

[107] Thibaut Théate and Damien Ernst. "An application of deep reinforcement learning to algorithmic trading". In: *arXiv* (2020).

[108] Thibaut Théate and Damien Ernst. "An application of deep reinforcement learning to algorithmic trading". In: *arXiv* (2020).

[109] Sebastian B Thrun. "Efficient exploration in reinforcement learning". In: (1992).

[110] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IROS, 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033.

[111] Marc Toussaint. "Robot trajectory optimization using approximate inference". In: *Proceedings of the 26th annual international conference on machine learning*. 2009.

[112] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[113] Alexander Sasha Vezhnevets et al. "Feudal networks for hierarchical reinforcement learning". In: *arXiv* (2017).

[114] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575 (Nov. 2019).

[115] Oriol Vinyals et al. *StarCraft II: A New Challenge for Reinforcement Learning*. 2017.

[116] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. 2016.

[117] Ermo Wei et al. "Multiagent soft q-learning". In: *arXiv* (2018).

[118] Ying Wen et al. "Probabilistic recursive reasoning for multi-agent reinforcement learning". In: *arXiv* (2019).

[119] Yuhuai Wu et al. *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*. 2017.

[120] Zhuoran Xiong et al. "Practical deep reinforcement learning approach for stock trading". In: *arXiv* (2018).

[121] Hongyang Yang et al. "Deep reinforcement learning for automated stock trading: An ensemble strategy". In: *SSRN* (2020).

[122] Tianhe Yu et al. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1094–1100.

[123] Yan Zheng et al. "Weighted double deep multiagent reinforcement learning in stochastic cooperative environments". In: *Pacific Rim international conference on artificial intelligence*. 2018.

[124] Brian D. Ziebart. "Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy". PhD thesis. USA, 2010. ISBN: 9781124414218.

[125] Brian D Ziebart. "Modeling purposeful adaptive behavior with the principle of maximum causal entropy". In: (2010).

[126] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *AAAI*. 2008.