The University of Southern Mississippi

The Aquila Digital Community

Master's Theses

Summer 7-24-2022

Reinforcement Actor-Critic Learning As A Rehearsal In MicroRTS

Shiron Manandhar

Follow this and additional works at: https://aquila.usm.edu/masters_theses

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

Manandhar, Shiron, "Reinforcement Actor-Critic Learning As A Rehearsal In MicroRTS" (2022). *Master's Theses*. 914.
https://aquila.usm.edu/masters_theses/914

This Masters Thesis is brought to you for free and open access by The Aquila Digital Community. It has been accepted for inclusion in Master's Theses by an authorized administrator of The Aquila Digital Community. For more information, please contact aquilastaff@usm.edu.

The University of Southern Mississippi

The Aquila Digital Community

Master's Theses

Summer 7-24-2022

Reinforcement Actor-Critic Learning As A Rehearsal In MicroRTS

Shiron Manandhar

Follow this and additional works at: https://aquila.usm.edu/masters_theses

Part of the Artificial Intelligence and Robotics Commons

The University of Southern Mississippi

The Aquila Digital Community

Master's Theses

Summer 7-24-2022

Reinforcement Actor-Critic Learning As A Rehearsal In MicroRTS

Shiron Manandhar

Follow this and additional works at: https://aquila.usm.edu/masters_theses

Part of the Artificial Intelligence and Robotics Commons

REINFORCEMENT ACTOR-CRITIC LEARNING AS A REHEARSAL IN MIRCORTS

by

Shiron Manandhar

A Thesis
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Computing Sciences and Computer Engineering
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved by:

Dr. Bikramjit Banerjee, Committee Chair
Dr. Andrew Sung
Dr. Chaoyang Zhang

August 2022

COPYRIGHT BY

SHIRON MANANDHAR

2022

ABSTRACT

Real-time strategy (RTS) games have provided a fertile ground for AI research with notable recent successes based on deep reinforcement learning (RL). However, RL remains a data-hungry approach featuring a high sample complexity. In this thesis, we focus on a sample complexity reduction technique called reinforcement learning as a rehearsal (RLaR), and on the RTS game of MicroRTS to formulate and evaluate it. RLaR has been formulated in the context of action-value function based RL before. Here we formulate it for a different RL framework, called actor-critic RL. We show that on the one hand the actor-critic framework allows RLaR to be much simpler, but on the other hand it leaves room for a key component of RLaR–a prediction function that relates a learner's observations with that of its opponent. This function, when leveraged for exploration, accelerates RL as our experiments in MicroRTS show. Further experiments provide evidence that RLaR may reduce actor noise compared to a variant that does not utilize RLaR's exploration. This study provides the first evaluation of RLaR's efficacy in a domain with a large strategy space.

ACKNOWLEDGMENTS

This is to thank all of those who have assisted me in this effort. This work was supported in part by Air Force Research Lab grant FA8750-20-1-0105.

I would like to thank my advisor, Dr. Bikramjit Banerjee, for providing continuous support for my research work. I was able to complete this research through his vision and guidance.

I would like to thank Dr. Andrew Sung and Dr. Chaoyang Zhang for serving as committee members and for providing feedback on my research work.

Lastly, I would like to thank my family and friends for their support and encouragement.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGMENTS iii

LIST OF ILLUSTRATIONS v

LIST OF TABLES vi

LIST OF ABBREVIATIONS vii

1 INTRODUCTION **1**

2 RELATED WORK **4**

3 BACKGROUND **6**

3.1 Reinforcement Learning 6

3.2 Reinforcement Learning as a Rehearsal (RLaR) 7

3.3 MicroRTS 8

4 METHODOLOGY **10**

4.1 Actor Network 11

4.2 Critic Network 12

4.3 Prediction Network for RLaR 13

5 EXPERIMENTAL RESULTS **16**

6 CONCLUSION **21**

BIBLIOGRAPHY **22**

LIST OF ILLUSTRATIONS

Figure

1.1 A state of MicroRTS game 2

4.1 The actor network used for all three agent types. Neural network layers are shaded in blue. Inputs (ω) are shaded in yellow, and outputs (samplings from softmax layers) are shaded in pink. Here N_E is the number of entities owned by any player, N_A is the number of actions allowed, N_T is the number of entity-types that can be produced, and N_L is the number of locations that can be attacked. Masks are computed from inputs to suppress, and thus reduce, the support of softmax distributions. For instance, only the visible locations that contain opponent entities are allowed to be activated for sampling "Attack Location Index". 10

4.2 The critic network used for RLAlpha and RLaR agents. Neural network layers are shaded in blue. Inputs are shaded in yellow. ω_t and ω_t^- contain the same components (from the perspectives of the player and its opponent, respectively) as shown in Fig. 4.1's input. 12

4.3 The prediction network used for the RLaR agent. Neural network layers are shaded in blue. Inputs are shaded in yellow. One of the inputs is from the output layer of the policy/actor network (π_{t-1}) shown in Fig. 4.1. This network minimizes 2 standard loss components: *latent* loss (KL divergence between the captured distribution and standard Gaussians ($\mathcal{N}(0,I)$), shown in red double-headed arrow), and a *reconstruction* loss measured by the cross entropy between the input ω_t^- and predicted $\hat{\omega}_t^-$. A third loss function (entropy of the captured conditional distribution) is added to the actor network's loss, and does not participate in training this network. 13

5.1 The 4 maps used in our experiments. White cells are unobserved, purple cells are observed by both blue and red teams. The learning agents always assume the role of the blue team, but there is no advantage to either role due to initial symmetry. The red team is MicroPhantom. 17

5.2 Learning curves of RLAlpha (RLAlpha+A2C+SIL) and RLaR (RLaR+A2C+SIL) against MicroPhantom in 4 maps. Baseline RL (A2C+SIL) is excluded due to poor performance. The terminal reward for win/loss/draw are +1000/-1000/+50. The initial policy/actor was trained by supervised learning from games between MentalSeal and MicroPhantom on large set of maps, but performs poorly in (a). 18

5.3 Plots showing the number of steps that the learner needs before at least one of its units gets within a distance threshold of 4.0 of the opponent's base, thereby bringing it within the radius of the learner's visibility. 19

LIST OF TABLES

Table

5.1 **Performance of trained policies for 3 variants in 4 maps.** 19

LIST OF ABBREVIATIONS

RL	-	Reinforcement Learning
RLaR	-	Reinforcement Learning as a Rehearsal
RTS	-	Real Time Strategy (Games)
A2C	-	Advantage Actor Critic
LSTM	-	Long short-term memory
SIL	-	Self-imitation learning

Chapter 1

INTRODUCTION

Real-time strategy (RTS) games belong to the genre of 2-player strategy games where a player's goal is to build sufficient economic and military might to destroy the opponent. A wide array of actions are available to a player, ranging from gathering resources, to building bases that train and churn out soldiers, to attacking opponent's units and bases to ultimately destroy them. For over two decades, RTS games have provided a rich substrate for AI research as they feature many of its key challenges, viz., complex dynamic environments with incomplete information and partial observability (fog-of-war), simultaneous and durative actions with potentially non-deterministic effects, real-time response, and unfathomably large strategy spaces. Consequently, research focused on RTS games can have significant potential impact in many real-world domains (e.g., business, finance, governance, etc.) as they share many of the same challenges.

Reinforcement learning (RL) has been a popular technique for training AI agents for computer games, including RTS games. Decades of research in this field boosted by the deep learning revolution have culminated in spectacular successes recently, where trained agents have matched and surpassed human expertise in domains where humans were once considered invulnerable to AI [12, 25]. However, RL remains a data-hungry approach that requires the agent to conduct a large number of simulations in order to comparatively evaluate a vast space of strategic alternatives. This is often measured as sample complexity. Despite decades worth of significant effort devoted toward reducing sample complexity, it still takes hundreds of millions of samples/simulations to train an RL agent in complex domains such as RTS games.

In this thesis, we focus on a sample complexity reduction technique called reinforcement learning as a rehearsal (RLaR), and on the RTS game of MicroRTS to formulate and evaluate it. RLaR has been formulated in the context of action-value function based RL before [10]. Here we formulate it for a different RL framework, called actor-critic RL. We show that on the one hand the actor-critic framework allows RLaR to be much simpler, but on the other hand it leaves room for a key component of RLaR–a prediction function that relates a learner's observations with that of its opponent. This function, when leveraged for exploration, accelerates RL as our experiments in MicroRTS show. Further experiments provide evidence that RLaR may reduce actor noise compared to a variant that does not utilize RLaR's exploration.

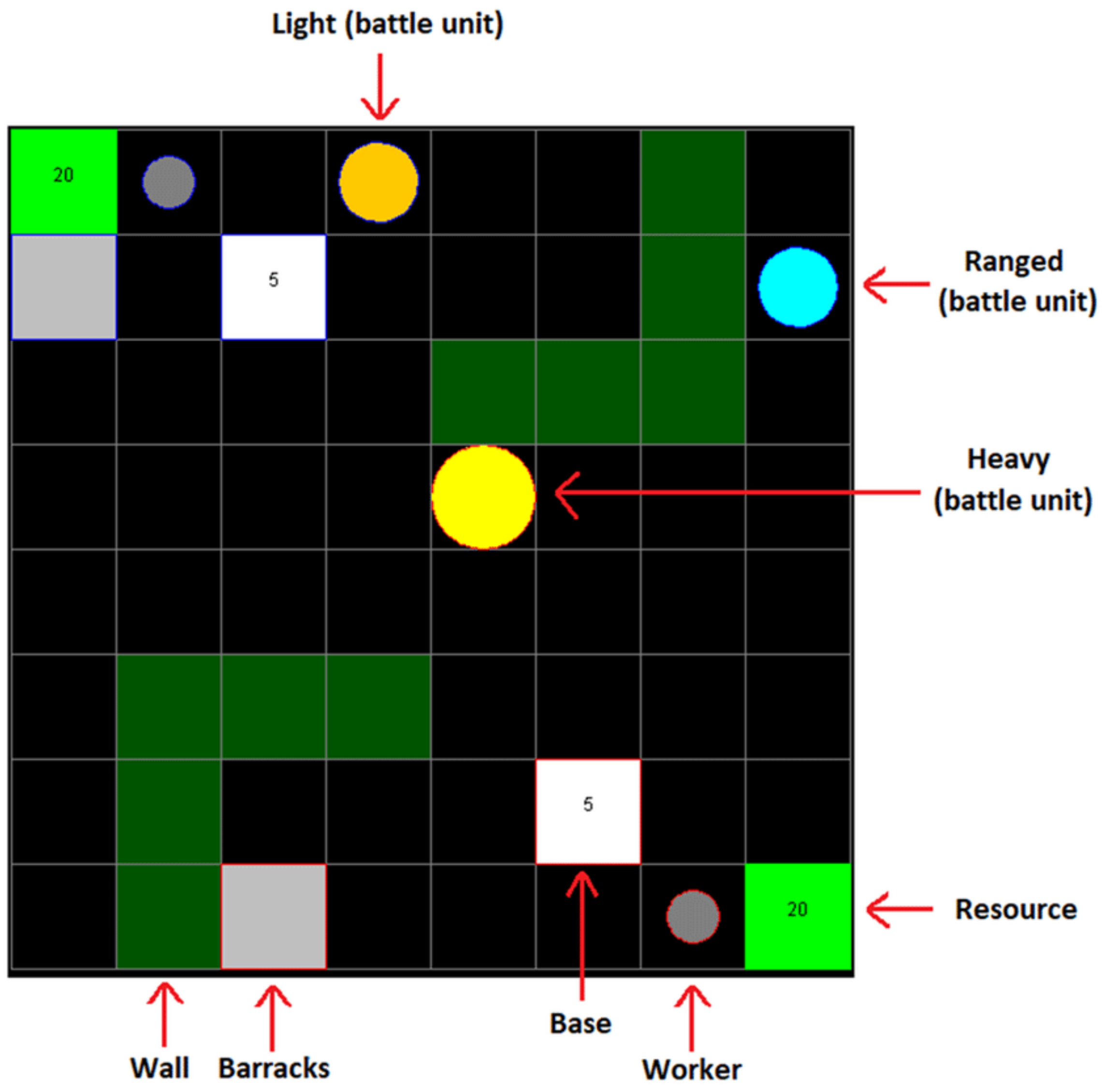


Figure 1.1: A state of MicroRTS game

Figure 1.1 represents a state of a game in MicroRTS. There are two players in the game, red and blue, and their objective is to destroy all their opponent units. There are seven units in the game, which are highlighted in Figure 1.1. Units owned by a player are outlined in that player's color (red or blue). The light green square boxes that contain a number is the mineable resources available to both players. These units are not owned by any single player. The light grey box that contains a number is called a base that produces workers. The number in a base represents the amount of resources available to it for worker production.

Workers harvest resources from green box and either return them to bases, or use them to build barracks (dark grey box). Barracks can only be built by a worker and it produces soldiers. Soldiers are attack units that destroys opponent units. The three types of soldier units are: light, ranged and heavy. Light units have low attack power but move fast, ranged units attack from long ranged and heavy units move slow but have high attack power.

Chapter 2

RELATED WORK

One of the earliest work that called for attention to RTS games as a challenging domain for AI research was by M. Buro [2]. In that work, Buro identified a broad mix of AI challenges, including resource management, opponent modeling and learning, real-time adversarial planning, spatial and temporal reasoning, and decision making under uncertainty. Another definitive account of task decomposition for RTS-playing AIs can be found in [28]. Due to the sheer size of the strategy space as well as the availability of human play data, some of the earliest AI approaches considered case based reasoning (CBR) and planning, e.g., [20, 16] among many others. These approaches match a current situation with situations stored in a knowledgbase of cases (from past/human play) to trigger the corresponding response, leading to fast and real-time response despite the vastness of the strategy space. Sharma et al. [20] combined CBR with reinforcement learning in a transfer learning context to facilitate the reuse of tactical plan components. Ontañón et al. [16] demonstrated the utility of case based planning for real-time decision making in the game of Wargus–a Warcraft II clone.

Standard AI techniques for search, planning, state estimation, etc. have also long been adapted to RTS and strategy games. For instance, Forbus et al. [6] applied qualitative spatial information acquired from geometric and path-finding analyses to wargames. Weber et al. [27] used a particle model with state estimation to track opponent units under fog-of-war. Perkins [18] applied Voronoi tesselation followed by search space pruning to identify regions and choke points in RTS maps. Churchill and Buro [4] used AI planning to optimize build orders in StarCraft, taking into account timing and scheduling constraints. Churchill et al. [5] also adapted the α-β pruning technique for durative actions for fast heuristic search in RTS combat. Chung et al. [3] applied Monte Carlo planning to a version of Open RTS (ORTS). Balla and Fern [1] applied the well-known Monte Carlo Tree Search algorithm based on upper confidence bounds, called UCT, to tactical assault planning in Wargus. To reduce the search space to a manageable size, most of these techniques rely on abstraction in state and action spaces.

Learning techniques, specifically supervised learning and reinforcement learning (RL), have also been applied to RTS games. Synnaeve and Bessiere [24] presented a Bayesian learning framework to predict the opponent's build tree based on replays, applied to StarCraft. Wender and Watson [29] evaluated a range of major RL algorithms for decentralized micromanagement in Broodwar (StarCraft). Marthi et al. [11] view an RTS player's control of a set of units as a robot with multiple effectors, and applied concurrent hierarchical Q-learning to efficiently control units. However, all units are afforded Q-functions at the bottom level. By contrast, Jaidee and Muñoz-Avila [9] use a single Q-function for each unit *type*, thus significantly reducing learning complexity. Since the spectacular success of (deep-) RL reported in Mnih et al.'s 2015 Nature article [12] where AI matched and even surpassed human level play in a range of Atari games, deep-RL has become a staple for computer games, including RTS games. Recently, (multi-agent) RL has achieved grandmaster-level sophistication in StarCraft II [25].

In this thesis, we focus on MicroRTS (a.k.a μRTS) [17]–a girdworld RTS game developed by Santiago Ontañón for AI research and competition. It contains much of the same components of an RTS game, but with less complex graphics. It has been the substrate of RTS competitions held in conjunction with the IEEE Conference on Games (COG) since 2017. Other relevant competitions include the Open RTS (ORTS) game AI competition (held from 2006-2009), AIIDE StarCraft AI competition and CIG/COG StarCraft RTS AI competition, both held annually since 2010. The annual MicroRTS competitions feature two tracks: the fully observable "Classic Track", and the "Partial Observability" track that simulates fog-of-war. We focus on the latter. The top two entries in the latest 2021 competition were MentalSealPO and MicroPhantom. Since MicroPhantom follows a published methodology based on constraint programming and decision theory [19], we have chosen this bot as the opponent against which we train our learning agents.

Chapter 3

BACKGROUND

3.1 Reinforcement Learning

Reinforcement learning problems are modeled as *Markov Decision Processes* or MDPs [22]. An MDP is given by the tuple $\langle S, A, R, P \rangle$, where S is the set of environmental states that an agent can occupy at any given time, A is the set of actions from which it can select one at a given state, $R : S \times A \mapsto \mathbb{R}$ is the reward function, i.e., $R(s,a)$ specifies the reward from the environment that the agent gets for executing action $a \in A$ in state $s \in S$; $P : S \times A \times S \mapsto [0,1]$ is the state transition probability function, i.e., $P(s,a,s')$ specifies the probability of the next state in the Markov chain being s' following the agent's selection of action a in state s. The agent's goal is to learn a policy $\pi : S \mapsto A$ that maximizes the sum of current and future rewards from any state s, given by,

$$V^{\pi}(s^0) = \mathbb{E}_P[R(s^0, \pi(s^0)) + \gamma R(s^1, \pi(s^1)) + \gamma^2 R(s^2, \pi(s^2)) + \gamma^3 \ldots] \tag{3.1}$$

where $s^0, s^1, s^2, \ldots$ are successive samplings from the distribution P following the Markov chain with policy π, and $\gamma \in (0,1)$ is a discount factor.

In this thesis we consider policy search methods [23] explicitly maintain a policy $\pi_\theta(a|s)$ denoting the probability of taking action a in state s, with the distribution being parametrized by θ. In this thesis we use a policy gradient method—belonging to the class of policy search methods—where $\pi_\theta(a|s)$ is differentiable w.r.t θ.

One popular policy gradient technique, called Advantage Actor-Critic (A2C), uses two function approximations. One function approximation represents the *actor*, viz. $\pi_\theta(a|s)$ responsible for selecting an action given a state, as stated above. The other function approximation represents the *critic*, viz., $V^{\pi}_{\phi}(s)$ which gives the value of the state s under the actor policy π (in essence it critiques the actor's performance), and is parametrized by ϕ. Normally θ is improved by policy gradient, optimizing

$$J(\theta) = \mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta} A(s,a) \tag{3.2}$$

where $d^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s | s_o, \pi_\theta)$ is the discounted state distribution that results from following policy π_θ, and $A(s,a)$ is called the advantage function that represents how much better (or worse) the value of taking action a in state s is compared to the average value from state s. A simple yet good estimate of the advantage function is the temporal difference (TD) error [23] given by

$$A_{TD}(s,a) = r_{sa} + \gamma V^{\pi}_{\phi}(s') - V^{\pi}_{\phi}(s) \tag{3.3}$$

where $r_{sa} \sim R(s,a)$ and $s' \sim P(s,a,.)$. This estimate only depends on the reward and states from the actual trajectories and the critic itself. While the mean squared TD errors (from equation 3.3) is used as the loss function for updating the parameters ϕ of the critic network, the actor network's parameters θ are updated using the gradient [30]

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta}}, a \sim \pi_{\theta}} \nabla_{\theta} \log \pi_{\theta}(a|s) A_{TD}(s,a). \tag{3.4}$$

In order to encourage exploration, an exploration bonus is added to the objective $J(\theta)$ whereby the *entropy* of the policy π_{θ} is also maximized, precluding the policy from settling into deterministic actions that could foreclose exploration. This gives a more complete expression for θ update:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta}}} \left[\mathbb{E}_{a \sim \pi_{\theta}} \nabla_{\theta} \log \pi_{\theta}(a|s) A_{TD}(s,a) - \beta \nabla_{\theta} \sum_{a} \pi_{\theta}(a|s) \log(\pi_{\theta}(a|s)) \right] \tag{3.5}$$

where β is the entropy bonus weight.

When the MDP is partially observable (POMDP), the state is not directly observed. Instead, the agent receives an observation, ω, that is (perhaps noisily) correlated with the hidden state. A common technique is to simply replace the states in the above equations with observations, or a history of past observations, as a sufficient statistic for the hidden state. In training neural networks π_{θ} and V^{π}_{ϕ}, history is accommodated via recurrence, e.g., using LSTM [7].

In this thesis, we use a variation of A2C, called A2C with self-imitation learning (A2C+SIL) [15], where apart from the A2C loss functions a SIL loss function is added where advantages corresponding only to positive experiences are used. In other words, states where advantages are negative are zeroed out, thus simulating a learner's desire to recreate positive experiences from its past. This approach has been shown to be effective for hard exploration tasks.

3.2 Reinforcement Learning as a Rehearsal (RLaR)

RLaR [10] was designed for partially observable settings where a training stage could be distinguished from an execution stage where the learned policy is applied/evaluated.

Furthermore, it was formulated in context of Q-learning [26], where an action-value function called Q-function is learned. It is related to the value function as follows:

$$V^{\pi}(\omega) = \max_{a} Q^{\pi}(\omega, a).$$

$Q^{\pi}(\omega, a)$ represents the long term value from following action a upon receiving observation ω, and the policy π thereafter. A Q-learning agent learns the optimal Q-values, $Q^{*}(\omega, a)\ \forall \omega, a$, and then constructs the optimal policy $\pi^{*}(\omega) = \arg\max_{a} Q^{*}(\omega, a)$. RLaR allows a learner to observe the hidden state (s that includes system state as well as opponent's observations and actions) in addition to its observation (ω), but *only* during the training stage as if to practice/rehearse. A RLaR agent learns an augmented Q-function, $Q^{*}(s, \omega, a)$, as well as an auxiliary predictor function (essentially a conditional probability distribution) $P(s|\omega)$, during the training/rehearsal stage. During the execution stage, the agent can construct a policy that no longer relies on hidden features, as

$$\pi^{*}(\omega) = \arg\max_{a} \sum_{s} Q^{*}(s, \omega, a) P(s|\omega).$$

This approach has been shown to expedite RL in simple 2-agent tasks [10], as well as in a larger swarm foraging task [14] more recently. In this thesis, we formulate RLaR within the actor-critic framework instead of Q-learning, and evaluate its effectiveness in a game with a large strategy space viz., MicroRTS.

3.3 MicroRTS

The components present in Microrts are bases, resources, barracks, worker units, and soldier units. A game is played between two players (learning agent controls the blue team), and the winner is determined when a player destroys all its opponent's units, including base, barracks and soldiers/units. If neither of the players is able to destroy it opponent's units within a given number of steps (3000 for this thesis), then it is a draw. Both the players are given a worker unit, a base and 5 number of resources initially. Their locations, as well as the locations of unowned mineable resources, are symmetric to prevent either player from having an initial advantage. Worker units can harvest resources and build bases and barracks. Barracks produce soldier units of three types: light, heavy and ranged. Light units have less hitpoints whereas heavy units have high hitpoints, but both can only attack immediately neighboring cells. By contrast, ranged units can attack from 3 grid cells away.

In this thesis, the learning agent is allowed to create up to $N_E = 70$ units–a number determined from game traces between MicroPhantom and MentalSealPO. We also limit the map sizes to 16×16 in order to restrict training time. Actions available to a unit include "noop", "attack", and 4 directions each of "move", "harvest", "return", and "produce", leading to $N_A = 18$ action types. Actions "attack" and "produce" are further qualified by which location to attack and what type of unit to produce. Considering $N_T = 7$ types of units and up to 10 hitpoints, these choices lead to a state space of maximum size $(7 * 10)^{70+70} * \binom{256}{70+70} \approx 10^{333}$, assuming both players are allowed up to 70 units. The learner's observation space is of maximum size $(7 * 10)^{70} * \binom{128}{70} \approx 10^{166}$, assuming about half of the grid space is available to locate its units. Its action space is of maximum size $18^{70} \approx 10^{87}$, conservatively assuming only one attack location and one produce type per unit. This leads to a strategy (mapping from observations to actions) space that is truly unfathomable.

Chapter 4

METHODOLOGY

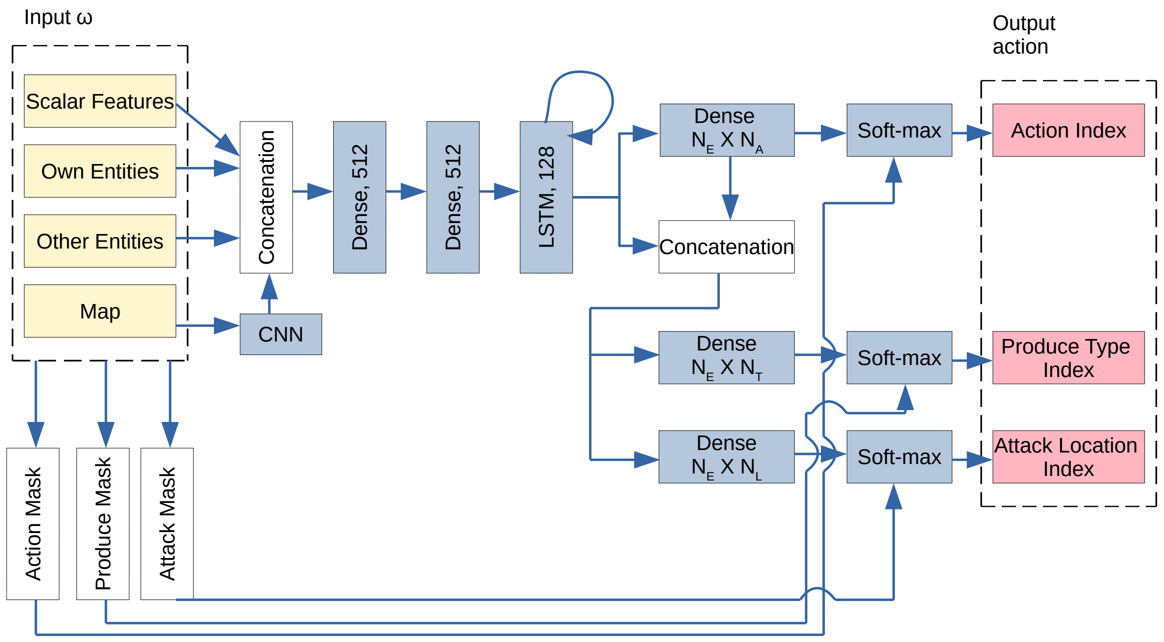


Figure 4.1: The actor network used for all three agent types. Neural network layers are shaded in blue. Inputs (ω) are shaded in yellow, and outputs (samplings from softmax layers) are shaded in pink. Here N_E is the number of entities owned by any player, N_A is the number of actions allowed, N_T is the number of entity-types that can be produced, and N_L is the number of locations that can be attacked. Masks are computed from inputs to suppress, and thus reduce, the support of softmax distributions. For instance, only the visible locations that contain opponent entities are allowed to be activated for sampling "Attack Location Index".

We apply actor-critic learning to MicroRTS using deep neural networks. The architectures of these networks are described next. Despite the existence of an OpenAI Gym framework [8] for RL in MicroRTS, we develop our own framework to gain the ability to (a) pass the hidden state to the RLaR agent, and (b) select an opponent of our choice (MicroPhantom for this thesis).

4.1 Actor Network

The architecture of the actor network, π_θ, is shown in Fig. 4.1, and is used for all versions of RL studied here. Its input is the learner's observation at step t, ω_t, consisting of the following components

Scalar Features: Binary encoding of scalar features, e.g., time, score, resources;

Own Entities: Sparse encoding of its own units (their types, locations, health and resource);

Other Entities: Similar sparse encoding of other visible units either owned by the opponent, or unowned (e.g., harvestable resources);

Map: A grid encoding of all visible units with their types.

The actor's output specifies the learner's action at step t, a_t. This is sampled from 3 soft-max probability distributions to yield the following:

Action Index: For each of up to N_E (=70 in our experiments) units that the learner owns, one of (N_A=) 18 indices that encode noop, attack, and 4 directions each of move, harvest, return, and produce;

Produce Type Index: If the produce action is selected for any of up to N_E units, the type index (from a set of $N_T = 7$ possible types) of what that unit will produce;

Attack Location Index: If the attack action is selected for any of up to N_E units, the target location of the attack from a set of N_L $(= 256)$ possible locations.

The soft-max layers are also provided with masks that reduce the support of the distributions, by deactivating elements that are invalid. Examples include movement directions that are blocked, harvest directions that do not contain resources, return directions that do not contain any self-base, produce types that are disallowed or require more resources than the agent/unit possesses, attack locations that are invisible or do not contain opponent units, etc. These masks allow the distributions to be learned rapidly despite the large strategy space, and are computable from ω_t and the information available from the unit_type_table provided at the beginning of the game. Similar invalid action masks are also used in [8].

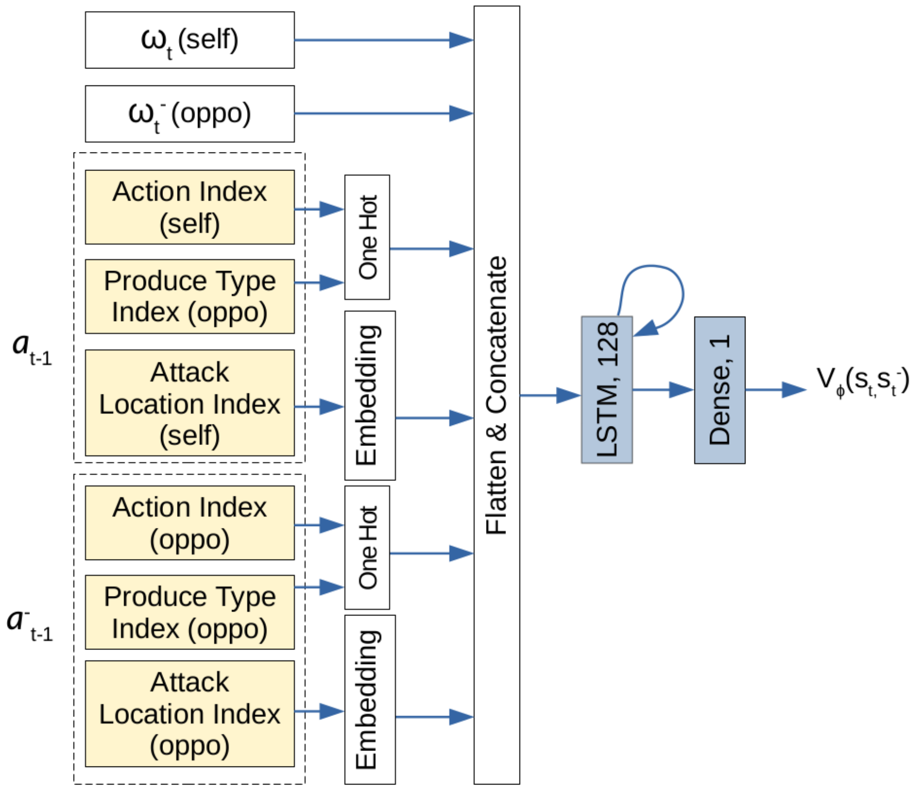


Figure 4.2: The critic network used for RLAlpha and RLaR agents. Neural network layers are shaded in blue. Inputs are shaded in yellow. ω_t and ω_t^- contain the same components (from the perspectives of the player and its opponent, respectively) as shown in Fig. 4.1's input.

4.2 Critic Network

Let $s_t = (\omega_{1:t}, a_{1:t-1})$ be the observation-action history of the learner, and $s_t^- = (\omega_{1:t}^-, a_{1:t-1}^-)$ be that of the opponent. Normally the opponent's observations are not available to a learner, hence for baseline RL the critic network learns the function $V_\phi(s_t)$ as described in Section 3.1. A distinct feature of RLaR is that both the learner and opponent's observations are available to the learner during the training stage, and accommodated in its critic, $V_\phi^\pi(s_t, s_t^-)$. Following [10], s_t^- can be marginalized out to compute a policy as

$$\pi^* = \arg\max_\pi \sum_{s_t^-} P(s_t^- | s_t, \pi) V^\pi(s_t, s_t^-),$$

using the learned auxiliary distribution $P(s_t^- | s_t, \pi)$. However, the actor-critic framework's clean separation of the policy from the value function makes this unnecessary. Since only the actor is needed after the training stage, and the critic is discarded, the accommodation of s_t^- in V is immaterial as long as the actor network is independent of s_t^-. Thus, for actor-critic training a simpler strategy is to exclude s_t^- altogether from the actor network, i.e., $\pi_\theta(a|s_t)$ instead of $\pi_\theta(a|s_t, s_t^-)$. This obviates the need for marginalization in the actor, and allows us to use the actor network from Sec. 4.1 for all methods. Notice that s_t^- still impacts the actor updates since V is needed in equation 3.4 via equation 3.3. This strategy is followed in AlphaStar [25], hence we call this approach RLAlpha and include it as a baseline in our experimental study. Both RLAlpha and RLaR use the critic network architecture shown in Fig. 4.2. While N_A, N_T are small and are converted to one-hot representation, N_L is large and is therefore embedded. The critic for baseline RL simply omits ω_t^- and a_t^- in its input, and is not shown separately. In contrast with the standard practice of combining the actor and critic networks to enable shared layers, we separate these networks such that the critics of the RL variants studied can be built incrementally without touching the actor.

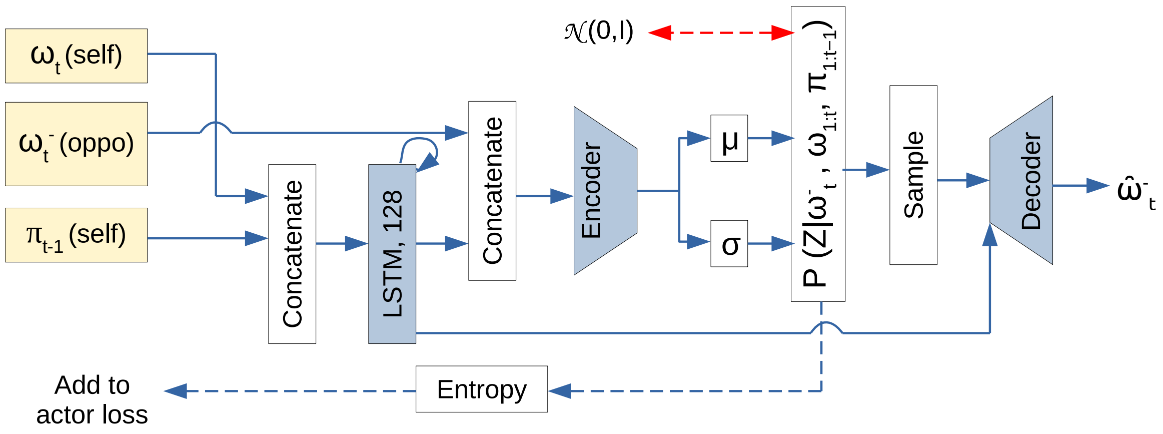


Figure 4.3: The prediction network used for the RLaR agent. Neural network layers are shaded in blue. Inputs are shaded in yellow. One of the inputs is from the output layer of the policy/actor network (π_{t-1}) shown in Fig. 4.1. This network minimizes 2 standard loss components: *latent* loss (KL divergence between the captured distribution and standard Gaussians ($\mathcal{N}(0, I)$), shown in red double-headed arrow), and a *reconstruction* loss measured by the cross entropy between the input ω_t^- and predicted $\hat{\omega}_t^-$. A third loss function (entropy of the captured conditional distribution) is added to the actor network's loss, and does not participate in training this network.

4.3 Prediction Network for RLaR

Although the auxiliary distribution $P(s_t^-|s_t, \pi)$ was shown to be unnecessary for actor-critic in Sec. 4.2, there are still good reasons to learn it. An important feature of RLaR (as explained in [10]) is a principled incentive for exploration,

$$\pi_{explore} = \arg\min_{\pi} - \sum_{s_t^-} P(s_t^-|s_t, \pi) \log P(s_t^-|s_t, \pi), \tag{4.1}$$

that seeks to reduce the entropy of the prediction $P(s_t^-|s_t,\pi)$. Ideally, if $P(s_t^-|s_t,\pi)$ is 1 then s_t is perfectly predictive of s_t^- under the current policy π, and the RLaR agent is truly independent of s_t^-. While RLAlpha does not have any incentive for this exploration, we can still endow RLaR with this capability for the following potential benefits:

- $\pi_{explore}$ may reduce noise in actor updates. Consider two situations where the learner observes s_t in both, but the opponent observes $s_{t,1}^-$ in one, and $s_{t,2}^-$ in another. While the critic can distinguish these situations being privy to $s_{t,1}^-$ and $s_{t,2}^-$, the actor cannot. If $V_\phi^\pi(s_t,s_{t,1}^-) \neq V_\phi^\pi(s_t,s_{t,2}^-)$, then the resulting updates will appear as noise to the actor. However, if $P(s_t^-|s_t,\pi)=1$ then $(s_t,s_t^-)\equiv s_t$ under π, and the above situation will not materialize. Thus $\pi_{explore}$ may push the actor toward generating situations where the updates are more stable.

- In the context of MicroRTS (and RTS games in general), $\pi_{explore}$ may encourage spying. In the partially observable setting of MicroRTS, a player can observe the set union of what its units can observe depending on their locations. Therefore, with strategically located units (a.k.a spies), a learner could make $\omega_t^- \subset \omega_t$, which would also minimize the entropy of $P(\omega_t^-|\omega_t,\pi)$. While spying may not be a worthwhile goal in and of itself, choosing actions with the knowledge of the opponent's configuration may be more desirable than without. Specifically, the success of the learned policy may be less dependent on the opponent's strategy, and more robust against other strategies.

Consequently, we seek to minimize the entropy of the distribution $P(\omega_t^-|\omega_{1:t},\pi_{1:t-1})$, which reflects the objective of equation 4.1 more closely than $P(s_t^-|s_t,\pi)$ in the context of MicroRTS. In particular, the condition $(\omega_{1:t},\pi_{1:t-1})$ subsumes (s_t,π) as the action history embedded in s_t is sampled from the policy history $\pi_{1:t-1}$. Although MicroRTS allows the opponent's actions a_{t-1}^- to be observed partly/wholly as a part of ω_t with sufficient proximity, we focus on the prediction of ω_t^- alone, rather than s_t^- in order to restrict the size of the prediction network.

To capture the conditional distribution $P(\omega_t^-|\omega_{1:t},\pi_{1:t-1})$, we use a probabilistic auto-encoder (shown in Fig. 4.3) similar to [21], albeit with an additional objective. In particular, an encoder network learns a latent representation of ω_t^- notated by latent variable Z, thus capturing the distribution $P(Z|\omega_t^-,\omega_{1:t},\pi_{1:t-1})$. A decoder network is then tasked with reconstructing ω_t^- given inputs Z and $\omega_{1:t},\pi_{1:t-1}$, thus inferring the distribution $P(\omega_t^-|Z,\omega_{1:t},\pi_{1:t-1})$. Unlike [21], we do not use this auto-encoder as a generative model; yet we perform standard optimization of the variational evidence lower bound (ELBO) by minimizing the latent and reconstruction losses to update the predictor network, since it allows the latent variables to be distributed as $P(Z|\omega_{1:t}, \pi_{1:t-1})$. Our objective, in addition to the ELBO, is to minimize the entropy of this distribution. In order to serve as the exploration component (equation 4.1), the gradients resulting from this entropy loss is only used to update the actor network, not the predictor network itself. The predictor update is solely based on the ELBO.

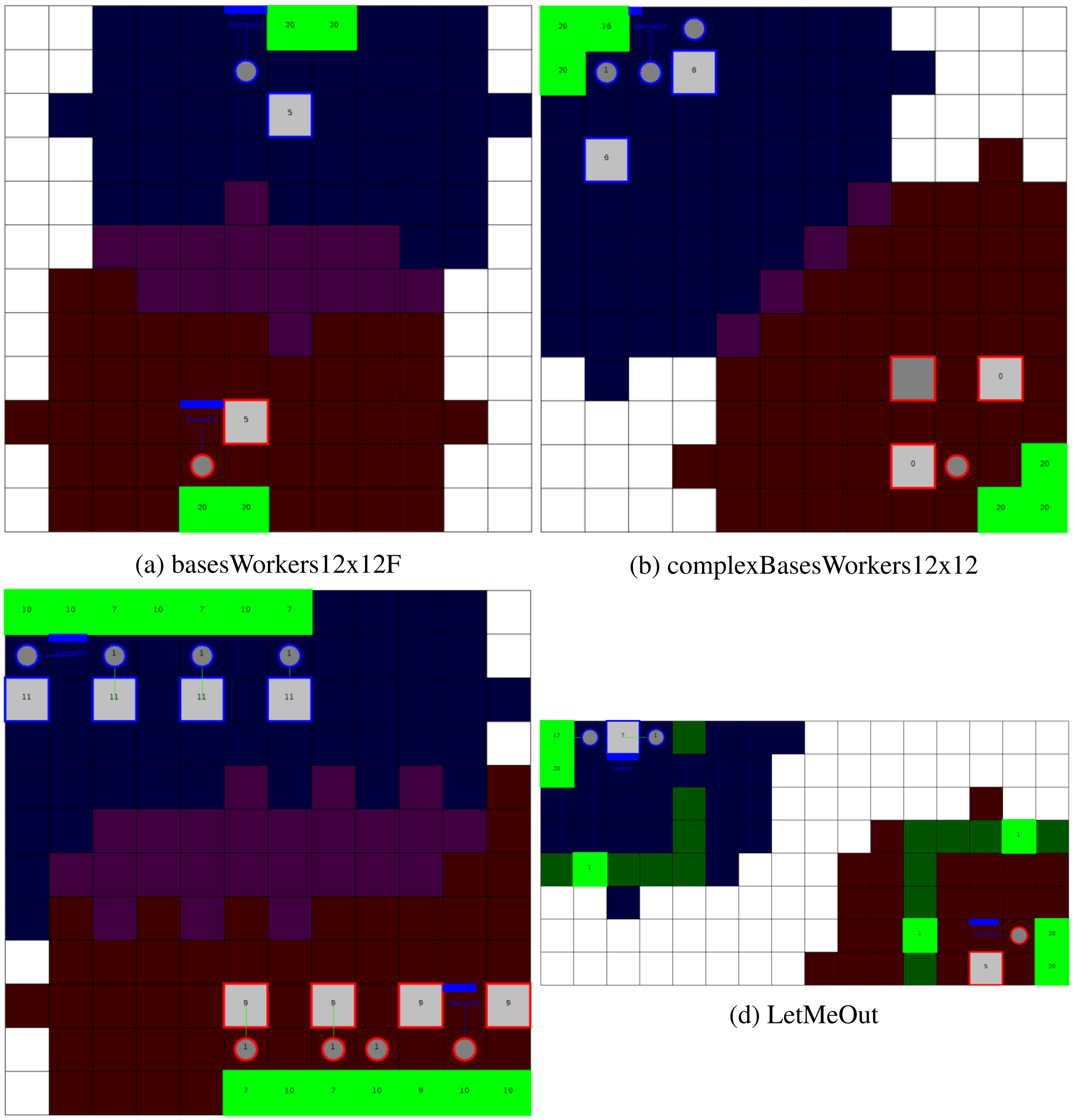
Chapter 5

EXPERIMENTAL RESULTS

We experiment with the three methods discussed in Section 4, viz., baseline RL, RLAlpha, and RLaR. For baseline RL, we use the advantage actor-critic (A2C) algorithm described in Section 3.1, modified with self-imitation learning [15], A2C+SIL. Both RLAlpha and RLaR are built on top of A2C+SIL, thus sharing this common baseline. We train each variant in four different maps, shown in Fig. 5.1. We selected these maps to incorporate variety of difficulty. For instance, the map "basesWorkers12x12F" (Fig. 5.1(a)) has the resources (bright green cells) in (relatively) opposite and non-corner locations, compared to other maps. The map "FourBasesWorkers12x12" (Fig. 5.1(c)) contains more initial bases and resources than other maps. Finally, the map "LetMeOut" (Fig. 5.1(d)) has a very different layout than other maps, where the players are walled (dark green cells) off, with doorways initially blocked by resources (although the blue agent had cleared one doorway by the time the screenshot was taken). Games are capped at a maximum of 3000 steps. We use a sparse reward scheme, with 0 reward for any intermediate step, and non-zero rewards only for terminal steps: $+1000$ for a win, -1000 for a loss, $50+\mathit{score}$ for a draw (i.e., when a game does not complete within 3000 steps), where *score* is the learner's MicroRTS assigned terminal score that reflects the strength/weakness of its final position in the absence of a clear winner. 50 bonus points are added for drawn games in order avoid 0 returns for the entire trajectory when $\mathit{score} = 0$. The rest of the parameters are set as follows:

- $\gamma = 0.999$
- $\beta = 0.005$
- Actor learning rate = 5×10^{-5}
- Critic learning rate = 5×10^{-4}

The learning curves corresponding to the 4 maps are shown in Fig. 5.2, over a series of 5500 games. Each curve is averaged over 6 independent trials, with half standard deviation bands shown in corresponding colors. The initial policy/actor for all versions were trained by supervised learning from a set of games played between MicroPhantom and MentalSeal. This results in positive initial performance of all variants, as seen in



(a) basesWorkers12x12F

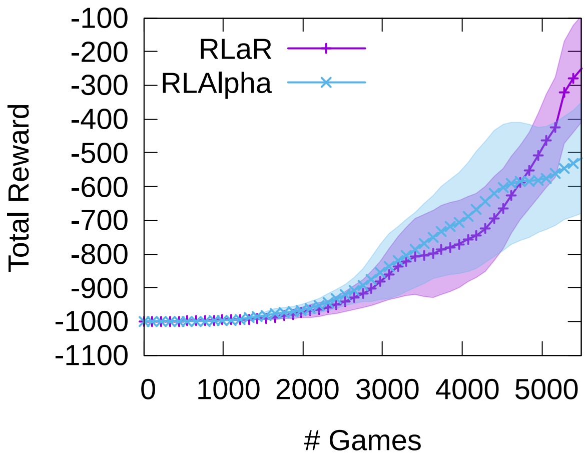
(b) complexBasesWorkers12x12

(c) FourBasesWorkers12x12

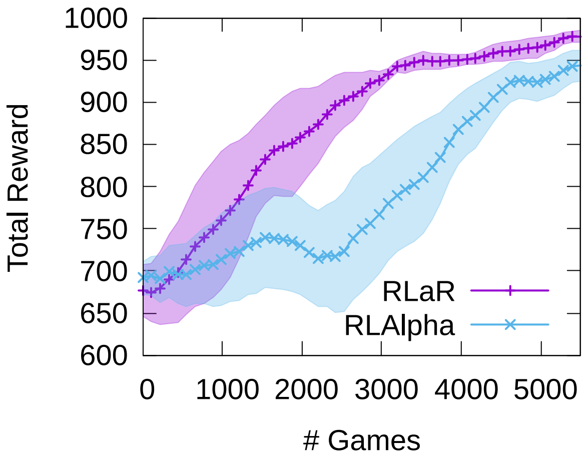
(d) LetMeOut

Figure 5.1: The 4 maps used in our experiments. White cells are unobserved, purple cells are observed by both blue and red teams. The learning agents always assume the role of the blue team, but there is no advantage to either role due to initial symmetry. The red team is MicroPhantom.

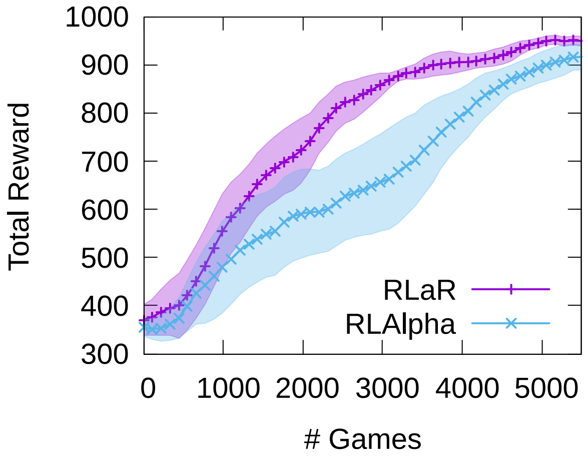
Figs. 5.2(b-d), although the trained initial policy was practically useless in (a). The learning curves demonstrate a superior learning rate for RLaR, and also serve as an ablation for the predictor network as that is the only difference between RLAlpha and RLaR. Also note that a total reward approaching +1000 indicates that the agent has learned to almost always



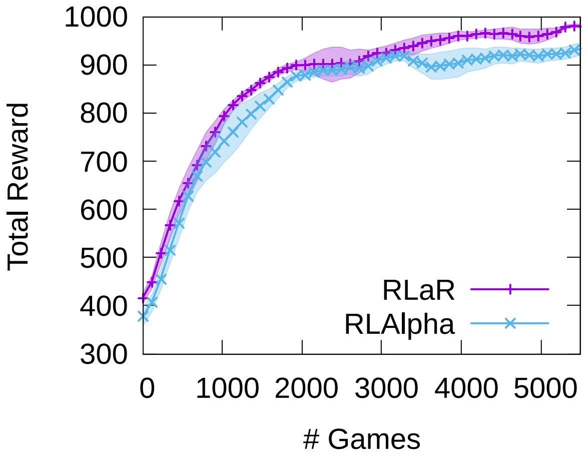
(a) basesWorkers12x12F



(b) complexBasesWorkers12x12



(c) FourBasesWorkers12x12



(d) LetMeOut

Figure 5.2: Learning curves of RLAlpha (RLAlpha+A2C+SIL) and RLaR (RLaR+A2C+SIL) against MicroPhantom in 4 maps. Baseline RL (A2C+SIL) is excluded due to poor performance. The terminal reward for win/loss/draw are +1000/-1000/+50. The initial policy/actor was trained by supervised learning from games between MentalSeal and MicroPhantom on large set of maps, but performs poorly in (a).

defeat MicroPhantom. Videos of trained RLaR policy against MicroPhantom are posted at `https://tinyurl.com/y3xhb9nt`

Baseline RL is not shown in Fig. 5.2 as its performance is poor in comparison with RLAlpha and RLaR. In particular, starting with the trained initial policy, baseline RL essentially *unlearns* it, dropping the total reward to -1000 (even in maps (b-d)) before improving it again. Essentially, baseline RL is unable to leverage the initial policy at all, requiring more time to learn. We show the performance of the learned policy at the end of 5500 games for all three variants in Table 5.1. Table 5.1 clearly demonstrates the futility of single agent (baseline A2C+SIL) RL in the face of a large strategy space.

Table 5.1: **Performance of trained policies for 3 variants in 4 maps.**

Maps	RL(A2C+SIL)	RLAlpha	RLaR
Fig. 5.1(a)	-998.7 ± 2.0	-515.9 ± 164.9	**-249.7** ± 161.0
Fig. 5.1(b)	591.9 ± 73.1	943.6 ± 18.5	**977.8** ± 8.9
Fig. 5.1(c)	254.8 ± 118.9	916.5 ± 26.4	**949.4** ± 11.5
Fig. 5.1(d)	125.5 ± 290.7	933.5 ± 12.5	**979.6** ± 4.1

Although the centralized (i.e., joint) critic of RLAlpha brings it closer to RLaR, Table 5.1 also demonstrates the scope for further improvement in terms of a principled exploration component that is unique to RLaR.

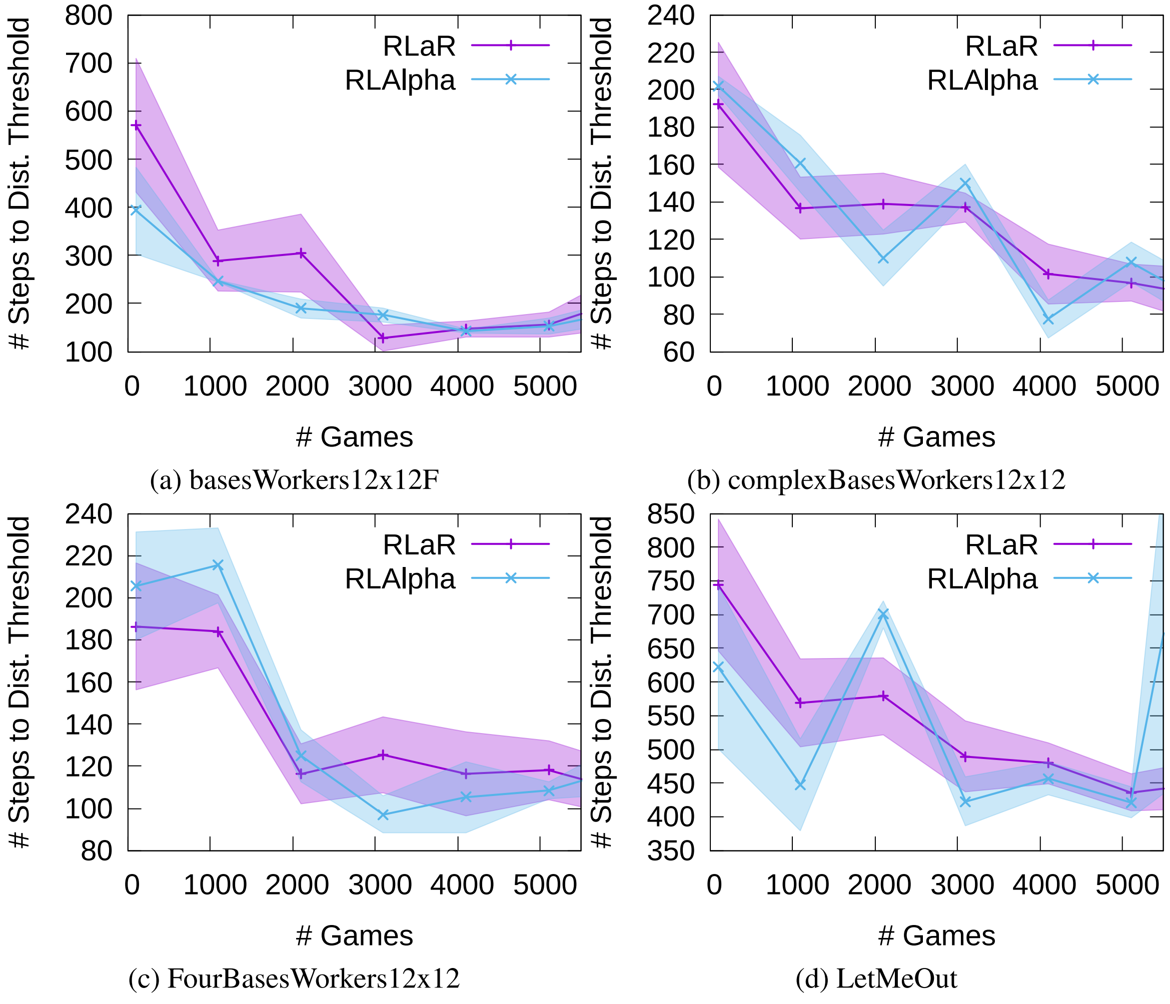


Figure 5.3: Plots showing the number of steps that the learner needs before at least one of its units gets within a distance threshold of 4.0 of the opponent's base, thereby bringing it within the radius of the learner's visibility.

In order to further evaluate the impact of RLaR's characteristic exploration, we conduct a second experiment. In this experiment, we note the number of steps in a game that it takes the learner to get close enough to the opponent's base, i.e., for any of its units to get within a distance threshold of the opponent's base. When there are multiple opponent bases, we take the centroid of their locations. This can be viewed as a rough measure of how quickly the learner deploys spies. The results are shown in Fig. 5.3 for a distance threshold of 4.0–sufficient to bring it within the observable radius. The first observation is that this measure does not correlate accurately with learning performance (Fig. 5.2), as early spying can end in failure while late spying can still end in victory. Neither is it a measure of the effectiveness of spying, as observing the opponent's base does not mean all of the opponent's units are also visible. However, another observation from Fig. 5.3 is that while the trend is expected to be decreasing with continued learning, this does not occur reliably with RLAlpha. Particularly in Fig. 5.3 (b) and (d), we notice spikes where the learner appears to be regressing in terms of this measure. RLaR, by contrast, achieves a steadier acceleration toward proximity. As proximity is a reliable predictor of the opponent's observation in MicroRTS, we speculate that this is a direct result of RLaR's use of predictor based exploration.

Chapter 6

CONCLUSION

We have presented a principled formulation of reinforcement learning as a rehearsal (RLaR) for the first time within the actor-critic framework. We have shown how a key component of RLaR, a prediction function that correlates the opponent's observations to the learner's own observations, can be constructed within a deep learning pipeline. Although the formulation is in the context of MicroRTS, it can be easily extended to other RTS games, e.g., StarCraft. We have experimentally validated two of the benefits of RLaR compared to a variant that has all the same features as RLaR except the prediction function. Consistent with previous findings on RLaR in smaller strategy spaces, we have shown that RLaR improves learning speed even in a domain with a large strategy space such as MicroRTS. A second experiment has shown that RLaR achieves visibility of the opponent's base more predictably as learning progresses. We speculate that this might be indirect evidence of noise reduction in actor updates–a second benefit of our approach–and at least partly responsible for improved learning rate of RLaR.

Reward shaping [13] is a well-established technique in RL where domain/prior knowledge is often used to supplement the reward function, in order to shape and accelerate learning. It is conceivable that a shaping function that rewards a learner for observing more of the opponent's units and penalizes it for observing less, could achieve similar learning speedup as RLaR in this thesis, because that is a known effect of reward shaping. Additionally, it might also achieve similar noise reduction, since the effect of such shaping on the actor in terms of the generated trajectories is likely to be similar. Further experiments can be conducted in the future to evaluate these intuitions. In contrast with this potentially alternative approach, we have relied on a simple (sparse) reward scheme in this thesis, and avoided explicit domain-specific reward engineering. More importantly, our approach is more general than reward shaping, as shaping functions can vary from domain to domain, but entropy minimization of the prediction function is a general principle that does not need domain-specific engineering, and can benefit domains well beyond RTS games.

BIBLIOGRAPHY

[1] Radha-Krishna Balla and Alan Fern. UCT for Tactical Assault Planning in Real-Time Strategy Games. In *International Joint Conference of Artificial Intelligence (IJCAI). San Francisco, CA, USA*, pages 40–45, 2009.

[2] Michael Buro. Real-Time Strategy Games: A New AI Research Challenge. pages 1534–1535. International Joint Conferences on Artificial Intelligence, 2003.

[3] Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte Carlo Planning in RTS Games. In *in IEEE Symposium on Computational Intelligence and Games (CIG)*. Citeseer, 2005.

[4] David Churchill and Michael Buro. Build Order Optimization in StarCraft. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 7(1):14–19, Oct. 2011.

[5] David Churchill, Abdallah Saffidine, and Michael Buro. Fast Heuristic Search for RTS Game Combat Scenarios. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 8(1):112–117, Jun. 2012.

[6] K.D. Forbus, J.V. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *IEEE Intelligent Systems*, 17(4):25–30, 2002.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[8] Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning. In *2021 IEEE Conference on Games (CoG)*, 2021.

[9] Ulit Jaidee and Héctor Muñoz-Avila. ClassQ-l: A Q-learning algorithm for adversarial real-time strategy games. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[10] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[11] Bhaskara Marthi, Stuart J Russell, David Latham, and Carlos Guestrin. Concurrent hierarchical reinforcement learning. In *International Joint Conference of Artificial Intelligence (IJCAI)*, pages 779–785, 2005.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[13] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[14] Trung Nguyen and Bikramjit Banerjee. Reinforcement Learning as a Rehearsal for Swarm Foraging. *Swarm Intelligence*, 16(1):29–58, 2021.

[15] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *ICML*, 2018.

[16] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Learning from demonstration and case-based planning for real-time strategy games. In *Soft Computing Applications in Industry*, pages 293–310. Springer, 2008.

[17] Santiago Ontañón. The Combinatorial Multi-Armed Bandit Problem and its Application to Real-Time Strategy Games. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 58–64, Boston, MA, 2013. AAAI.

[18] Luke Perkins. Terrain analysis in real-time strategy games: An integrated approach to choke point detection and region decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 6, pages 168–173, 2010.

[19] Florian Richoux. MicroPhantom: Playing MicroRTS under uncertainty and chaos. In *2020 IEEE Conference on Games (CoG)*, pages 670–677, 2020.

[20] Manu Sharma, Michael Holmes, Juan Santamaría, Arya Irani, Charles Jr, and Ashwin Ram. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. pages 1041–1046, 01 2007.

[21] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[22] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[23] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[24] Gabriel Synnaeve and Pierre Bessiere. A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011*, pages 79–84, 11 2011.

[25] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[26] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 3:279 – 292, 1992.

[27] Ben Weber, Michael Mateas, and Arnav Jhala. A particle model for state estimation in real-time strategy games. pages 103–108, 01 2011.

[28] Ben George Weber, Michael Mateas, and Arnav Jhala. Building human-level AI for real-time strategy games. In *AAAI Fall Symposium Series*, 2011.

[29] Stefan Wender and Ian Watson. Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar. In *2012 IEEE conference on computational Intelligence and Games (CIG)*, pages 402–408. IEEE, 2012.

[30] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.