Spring 5-2021

Teachability And Interpretability In Reinforcement Learning

Jeevan Rajagopal
University of Nebraska - Lincoln, jeevan.rajagopal@huskers.unl.edu

TEACHABILITY AND INTERPRETABILITY IN REINFORCEMENT LEARNING

by

Jeevan Rajagopal

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Stephen Donald Scott

Lincoln, Nebraska

May, 2021

TEACHABILITY AND INTERPRETABILITY IN REINFORCEMENT LEARNING

Jeevan Rajagopal, M.S.

University of Nebraska, 2021

Adviser: Stephen Scott

There have been many recent advancements in the field of reinforcement learning, starting from the Deep Q Network playing various Atari 2600 games all the way to Google Deempind's Alphastar playing competitively in the game StarCraft. However, as the field challenges more complex environments, the current methods of training models and understanding their decision making become less effective. Currently, the problem is partially dealt with by simply adding more resources, but the need for a better solution remains.

This thesis proposes a reinforcement learning framework where a teacher or entity with domain knowledge of the task to complete can assist the agent in finding a policy, while also providing human observers a more understandable format for how the agent formulates its overall policy to complete a given task. This framework is evaluated on the Seaquest environment within the OpenAI Gym framework.

ACKNOWLEDGMENTS

First, I would like to thank my advisor Dr. Stephen Scott for taking me on as his student and advising me throughout my graduate studies. The insights I received were quite valuable in better understanding my work and the field as a whole.

I would also like to thank Eleanor Quint, Zeynep Hakguder, Haluk Dogan, and Jacob Williams for providing me the key insights that formed the core of my thesis.

Finally, I would like to thank my committee for taking the time to participate in my defense and providing valuable feedback on my thesis work.

Table of Contents

List of Figures **viii**

List of Tables **x**

1 Introduction **1**

1.1 Thesis Contributions 2

1.2 Thesis Roadmap 3

2 Background Information **5**

2.1 Major Concepts 5

2.1.1 Reinforcement Learning 5

2.1.2 Q Learning 8

2.1.3 Epsilon-Greedy Approach 10

2.1.4 Seaquest 12

2.2 Techniques Used 14

2.2.1 Transfer Learning 14

2.2.2 Reward Shaping 15

2.2.3 Action Shaping 16

2.3 Atari Learning Environment (ALE) 16

2.4 Keras RL 17

3 Related Works **19**

3.1 Human-level Control Through Deep Reinforcement Learning 19

3.2 Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning 21

3.3 Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition 22

3.4 Classifying Options for Deep Reinforcement Learning 23

3.5 Hierarchical Reinforcement Learning for Playing a Dynamic Dungeon Crawler Game 24

3.6 Identifying Subgoals by Local Graph Partitioning 25

3.7 Strategic Attentive Writer for Learning Macro-Actions 25

3.8 Deep Reinforcement Learning Boosted by External Knowledge 26

3.9 Hybrid Reward Architecture for Reinforcement Learning 27

4 Teachable-Interpretable DQN (TIDQN) **29**

4.1 Teachability in TIDQN 29

4.2 Interpretability in TIDQN 30

4.3 Representation of Tasks 30

4.3.1 Explaining a Representation of Tasks 32

4.3.2 Limitations of the Representation of Tasks 33

4.4 Completion Vector 34

4.5 Auxiliary Feature Extractors 35

4.6 Proposed Modifications 36

4.6.1 Diver Rescue 37

4.6.2 Removal of State Augmentation 37
4.6.3 Removal of Environment Reward 38
4.6.4 Adding Feature Extractors 38
4.6.4.1 Heuristic Location Classifier 39
4.6.4.2 Single-Label Location Classifier 40
4.6.4.3 Multi-Label Location Classifier 41
4.6.5 Action Shaping using Auxiliary Feature Extractors 43
4.6.6 Transfer Learning from vanilla DQN 44
4.6.7 Modifying the DQN Architecture 45

5 Experimental Setup 47

5.1 Training Setup 47
5.2 Testing Setup 48
5.2.1 10×10 Testing 48
5.2.2 Combining Proposed Methods 48
5.3 Description of Metrics 49
5.3.1 Episode Reward 50
5.3.2 Max Divers Per Episode 50
5.3.3 Total Divers Per Episode 51
5.3.4 Rate of Diver Acquisition 51
5.4 Results and Discussion 52
5.4.1 Diver Rescue 52
5.4.2 Removal of Environment Reward 60
5.4.3 Use of Auxiliary Feature Extractor 61
5.4.4 Option Heads Architecture 62

6 Conclusions and Future Directions 64

6.1 Conclusion 64
6.2 Future Directions 65
6.2.1 Action Shaping via Completion Vector 65
6.2.2 Testing Alternative Representation of Tasks 65
6.2.3 State Augmentation with the Representation of Tasks 66
6.2.4 Using Configurable Environments 66
6.2.5 Utilizing Bayesian Networks as Supervisory Network 67
6.2.6 Improving Auxiliary Feature Extractor Performance 67
6.2.7 Inclusion of Constraints 67
6.2.8 Testing Alternatives to Including State Augmentation 68

A Investigation of the Performance of the Auxiliary Feature Extractor 69
A.1 Problem Outline 69
A.2 Description of Metrics 69
A.2.1 Subset Accuracy 69
A.2.2 Hamming Loss 70
A.2.3 F1 Score 70
A.3 Testing Method 71
A.4 Results and Discussion 71
A.5 Conclusion 73

B Seeds Used for Training and Testing 74

Bibliography 75

List of Figures

2.1 "The agent-environment interaction in reinforcement learning" [24]. Below is the order of operations that occur between an agent and environment. Figure courtesy of Mitchell [13]. 7
2.2 An image representing the Seaquest environment. 13

3.1 An implementation of the DQN architecture for the game Seaquest. . . . 20
3.2 A model diagram of the proposed approach's implementation of the Option Heads architecture. Note that each head is constructed exactly the same. 24

4.1 A Representation of Tasks for the game Seaquest. It shows the main objective of the environment, which is to rescue divers in groups of six. . 31
4.2 An example of the diver locator auxiliary feature extractor transforming an observation V into quadrants were divers are located. The bits correspond to the following quadrants: no divers, upper left, lower left, upper right, lower right. The orange line shows a representation of the quadrant used to determine labels and is not present in practice. 39
4.3 A model description of the multi-label diver locator 42
4.4 The single-head TIDQN model. 45

5.1 The aggregate max divers per episode over 10×10 averaging, comparing the inclusion and exclusion of state augmentation against the baseline. The x-axis represents the highest number of divers found in an episode, with the y-axis representing the number of episodes recorded for a given x-value. 53
5.2 The aggregate max divers per episode over 10×10 averaging, comparing combinations of state augmentation with other modifications against the baseline DQN and baseline TIDQN. The x-axis represents the highest number of divers found in an episode, with the y-axis representing the number of episodes recorded for a given x-value. 53

List of Tables

5.1 Episode reward values over a 10×10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation. 52
5.2 The highest number of divers rescued per episode over a 10×10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation. 54
5.3 Total divers rescued per episode over a 10×10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation. 54
5.4 Episode reward values over a 10×10 average comparing the various modifications to the state augmented TIDQN against the baseline DQN. All non-baseline approaches in this table used reward shaping in addition to the modification listed. 54
5.5 Maximum number of divers rescued per episode over a 10×10 average for each approach and modification. 54
5.6 Total divers rescued per episode over a 10×10 average for each approach and modification. 55

5.7 Summary Diver Acquisition Rates over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers for all approaches. Lower values indicate the agent acquiring the subsequent diver faster. Italicised values indicate the number of transitions that contributed to the value were not high enough to indicate the policy was able to reach said transition reliably. 55
5.8 Diver Acquisition Rates over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 55
5.9 Diver Acquisition Rates of the 'No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 55
5.10 Diver Acquisition Rates of the 'No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 56
5.11 Diver Acquisition Rates of the 'Option Heads No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 56
5.12 Diver Acquisition Rates of the 'Option Heads State Augmentation Only' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 56

5.13 Diver Acquisition Rates of the ‘State Augmentation plus No Environment Reward’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k + 1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 56

5.14 Diver Acquisition Rates of the ‘Diver Locator Reward’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k + 1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. 57

5.15 Diver Acquisition Rates of the ‘No State Augmentation plus Action Shaping’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k + 1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. Since there is only one transition between five to six divers, it is considered as a fluke rather than a consistent process. 57

5.16 Diver Acquisition Rates of the ‘Option Heads with No State Augmentation plus Action Shaping’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k + 1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. Since there are only two transition between two to three divers, it is considered as a fluke rather than a consistent process. 57

A.1 Table describing the micro-average metrics for each label used in the multi-label diver locator. Values in bold indicate the label which had the highest value for that particular metric. 71

A.2 Table describing the macro-average metrics for each label used in the multi-label diver locator. Values were generated from 10-fold cross validation of the model. 72

B.1 Train and Test values used in the 10×10 comparisons. 74

Chapter 1

Introduction

Deep learning has been integrated in many facets of modern living. Ranging from voice-activated digital assistants to self-driving vehicles, deep learning has found numerous use cases throughout modern society. However, reinforcement learning still struggles in the face of real-world complexity. This is in part due to the difficulty of incorporating long-term strategies and planning, along with which can be seen in various video games, such as Montezuma's Revenge [23], Defense of the Ancients [11] and Starcraft [7]. Unlike previous benchmarks for reinforcement learning, these games showcase several parallels to the real world, including partial observability, very long-term planning, and a myriad of ways to interact with the environment to achieve the goal. One of many directions taken to tackle this problem involves the use of domain knowledge, or knowledge derived from human experts, to try and guide the computer towards better performance.

This thesis proposes a reinforcement learning framework where a teacher or entity with domain knowledge of the task to complete can assist the agent in finding a policy, while also providing human observers a more understandable format for how the agent formulates its overall policy to complete a given task. This framework is evaluated on the Atari 2600 game Seaquest to determine its efficacy against the DQN network by Minh et al. [15].

Historically, there have been many techniques that attempted to address the issue of scalability present in reinforcement learning [10, 25, 9, 18]. Despite the promise these approaches showed, they seemingly fell out of favor until the field once again found itself approaching the wall of complexity, this time in environments like StarCraft [30] and Defense of the Ancients (DotA) [3].

Many modern approaches [16, 26, 28] have taken to revisiting the principles outlined by older papers [25, 10], but only to the extent of adapting and modifying the approaches to fit the current trend of deep learning. As such, the choices made by the agent remain inscrutable to human observers.

We propose a framework that seeks to include both interpretability and teachability to the reinforcement learning process with minimal modifications to the existing deep reinforcement learning process. By establishing a relationship between a "teacher", or some entity that has domain knowledge of the task, and the agent, the means by which information is transmitted between the two parties and the transparency of said information, we believe this framework will provide additional information on the policies created with minimal to no cost on the quality of policy produced.

1.1 Thesis Contributions

This thesis makes the following contributions. The repository is available online using the provided link.[1]

- Proposed a new framework to assist developers in creating a decomposition of tasks that can be easily understood in a given environment.

[1] `https://github.com/blackhole077/TIDQN-repository`

- Combined auxiliary feature extractors into the DQN architecture and tested its efficacy in policy creation.
- Attempted multiple variants of state augmentation and shaping as potential implementations of the proposed framework to determine efficacy against a baseline.
- Attempted to implement the Option Heads [1] approach and Bootstrapped DQN [17] approach in the Keras RL library.
- Tested all variants of TIDQN against the baseline DQN on the Seaquest [6] environment, using environment-specific metrics such as Total Divers per Episode, Diver Acquisition Rate and Max Divers per Episode to better illustrate the difference in performance.
- Determined that the current implementation of TIDQN formed a negative correlation between the level of external intervention and policy performance and provided possible causes.

1.2 Thesis Roadmap

The rest of this thesis is organized as follows. Chapter 2 will cover background information that is necessary to understand concepts presented in the proposed approach, along with explanations for how these concepts tie into the approach. Chapter 3 will cover works that are related to the proposed approach and the differences, if any, between said works and the proposed approach. Chapter 4 covers the framework itself, along with its components, how they interact with one another, and their intended purpose in improving the policies created. Chapter 5 covers experimental setup, starting with how training and testing were conducted, an explanation of all metrics used and the various modifications that were attempted over the course of experimentation, along with results and observations made from said modifications. Finally, Chapter 6 will cover the overall conclusions that can be made based on the observations made and the future directions that the proposed approach can be taken for further investigation.

Chapter 2

Background Information

This chapter covers core concepts, terms, and technologies used throughout the remainder of the thesis, along with explanations for what purpose certain technologies serve in the proposed approach.

2.1 Major Concepts

2.1.1 Reinforcement Learning

Reinforcement Learning, as defined by Sutton and Barto [24], is a sub-area of machine learning that focuses on the learning of a mapping from states to actions that maximizes some numerical reward. More informally, it is the paradigm of machine learning that focuses on the machine learning what to do by way of trial-and-error.

Unlike the other paradigms of machine learning, reinforcement learning makes use of an interaction between the machine, usually defined as an agent, and the environment it is placed in. The agent uses a policy to guide what actions to take inside of an environment and, in turn, receives some reward from the environment. As these actions cause changes to the environment, the agent is tasked with learning an optimal mapping between states and actions to maximize the reward it receives from the environment.

In the context of a game, the concepts present in reinforcement learning are much more intuitive. Much like a player learning to play a game to get a high score, an agent learns to act in an environment to maximize the total reward. While the proposed approach makes use of the Atari 2600 video game Seaquest, reinforcement learning has use cases that extend into the real world, commonly being paired with robotics in hopes of achieving autonomous entities.

Figure 2.1 shows the basic interaction between the agent and environment. At some time step t, the agent will receive state information, denoted s_t, from the environment. Using the state information, the agent chooses an action, denoted a_t, based on its current policy π. The environment then changes based on the action taken and returns both a new state s_{t+1} and a reward value r_{t+1}. Through this feedback loop, π is optimized to find the action that maximizes the expected discounted reward value given some state, where the expectation is with respect to the randomness in the functions δ, r, and π. This value is described in Equation 2.1 from Mitchell [13]. In Equation 2.1, γ is the **discount factor**, or some value $0 < \gamma < 1$ that reduces the impact of future rewards.

$$V^{\pi}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \tag{2.1}$$

A general formulation of reinforcement learning problems, provided by Mitchell [13], is the **Markov Decision Process (MDP)**. Within an MDP, an agent is tasked with learning some policy $\pi : S \rightarrow A$, that maps each state $s \in S$ to some action $a \in A$. At some time step t, the agent will perceive state information s_t and select an action a_t via the policy $\pi(s_t) = a_t$. Once an action is selected, the state-transition function $\delta(s_t, a_t)$ returns the subsequent state s_{t+1} and the reward function $r(s_t, a_t)$ returns a

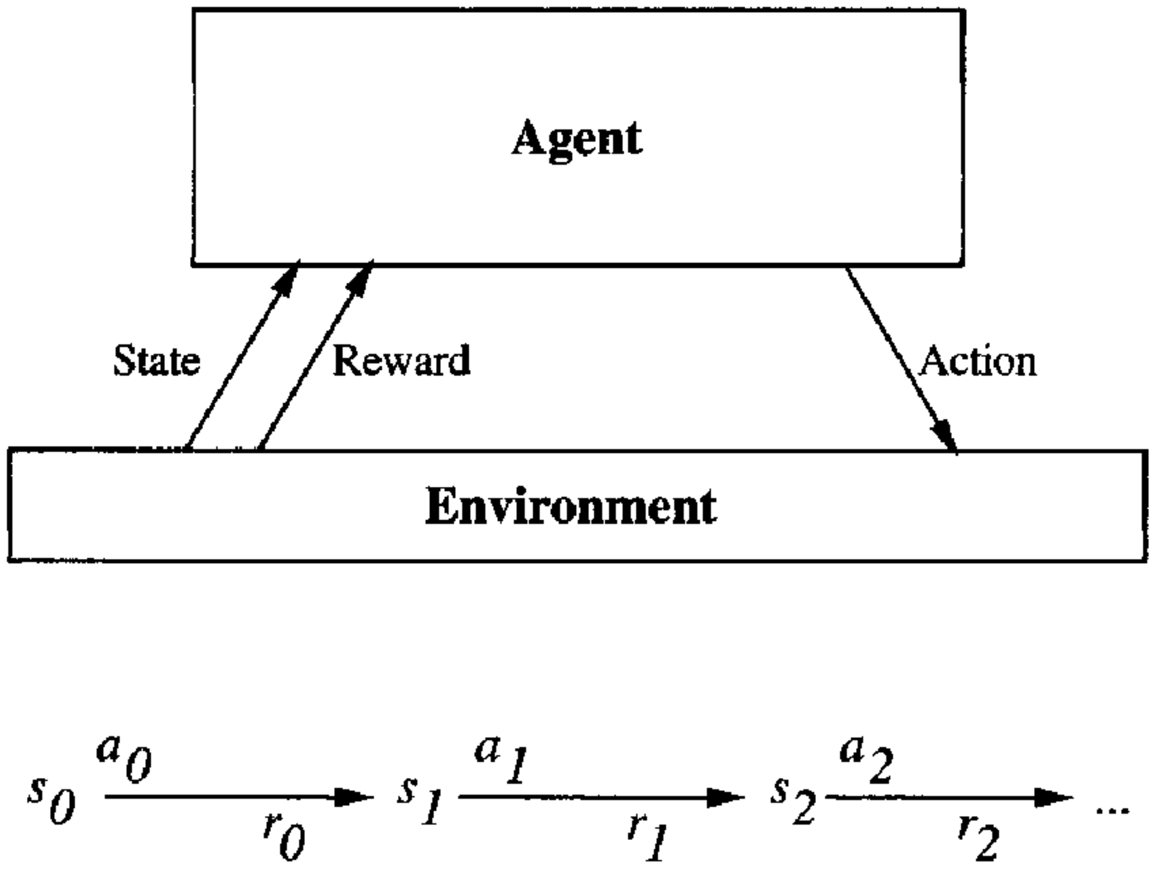


Figure 2.1: "The agent-environment interaction in reinforcement learning" [24]. Below is the order of operations that occur between an agent and environment. Figure courtesy of Mitchell [13].

reward r_t. Due to the Markov property, δ and r only rely on the information known in the current time-step, ignoring any prior information. Note that while δ and r only rely on what is known at the current time-step, the functions themselves can be either deterministic or non-deterministic.

Based on the descriptions provided, the basic agent-environment interaction is equivalent to the Markov Decision Process, as all components present in the feedback loop are directly tied to processes in a given MDP.

However, limitations exist within the MDP framework, a major one being the lack of temporal abstraction. In other words, due to how an MDP is formulated, movement through the MDP space is limited to one unit of time per decision. This limitation becomes apparent in more complex environments where a series of decisions could be grouped together. Sutton et al. [25] proposed the use of **Semi-Markov Decision Processes (SMDP)** via 'options' in response to this.

A major difference between an MDP and SMDP is the use of a random variable τ to determine how many units of time will elapse before the decision selected terminates. This is done to allow an MDP formulation to work over continuous time. Furthermore, while $\delta(s_t, a_t)$, $r(s_t, a_t)$ remain Markovian, the inclusion of τ alongside the current state and selected decision breaks the Markov property.

The ultimate goal within RL is to construct some optimal policy $\pi_* : S \rightarrow A$ that can output the optimal action $a_* \in A$ such that the expected discounted reward, shown in Equation 2.1 is maximized, for any state $s \in S$. To this end, the proposed approach utilizes Q Learning, discussed in Section 2.1.2.

2.1.2 Q Learning

There are many different methods that can be used to approach the reinforcement learning problem. A very commonly used method is Q Learning, which makes extensive use of the **action-value function** or **Q function**, denoted $Q_\pi(s, a)$. This function is defined by Sutton and Barto [24] as the expected return starting from s, taking the action a, and thereafter following policy π. Equation 2.2 from Mitchell [13] shows the non-deterministic version of the Q function. Q Learning is the method that attempts to directly learn an optimal policy π^* by learning the optimal Q function.

$$Q_\pi(s, a) \equiv E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a') \tag{2.2}$$

Once a Q function is learned, it can be used to realize a policy by selecting the action that satisfies $\max_a Q(s_t, a)$. Since the definition of a Q function is the expected return from state s after taking action a and following a policy π, if an optimal Q function is found then the policy that forms the expected return must also be optimal as well.

One of the major advantages the Q function and, by proxy, Q Learning has is that neither the transition function nor the reward function are required to be known, which makes it feasible to structure reinforcement learning as a dynamic programming problem and apply it to environments where such functions cannot be determined as noted by Mitchell [13].

Additionally, Q Learning is considered an **off-policy technique**. Off-policy techniques are a method where the learned action-value function Q is approximating the optimal action-value function Q_* independent from the current policy. This can be seen in Equation 2.3, where the update assumes that the subsequent action is chosen greedily, even if a different method is used by the current policy (e.g., using Epsilon-Greedy selection to play Seaquest). Equation 2.3 is the basis for the Q Learning algorithm provided by Sutton and Barto [24].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.3}$$

As mentioned earlier, a Q function estimate, or **Q value**, for a given state-action pair (denoted here as $Q(s_t, a_t)$) can be defined as the expected reward if, starting from state s_t and taking action a_t, the policy π is followed to termination.

In order to update it, the following operations occur. First, the discounted expected reward (denoted as $\gamma \arg\max_a Q(s_{t+1}, a)$) is subtracted by the current Q value estimate to produce the **error value**. The error value is then combined with the immediate reward for selecting a_t and multiplied by the **learning rate** α. Finally, this value is combined with the current Q value estimate and the update is performed. Minh et al. [15] combined the Q Learning technique with the function approximation capabilities of deep neural networks such that the technique was feasible in more complex environments.

However, as noted by Sutton and Barto [24], these changes lent themselves to increased instability, due to the use of function approximation, bootstrapping via Q Learning and off-policy training via Q Learning. These three components were termed “The Deadly Triad”, whereupon having all three present seems to incur unstable behavior. As such, techniques like experience replay were introduced alongside DQN [15] to try and stabilize the learning process.

Despite these issues, Q Learning remains a powerful tool for learning policies in environments that more closely mirror the real world. Therefore, the focus of the proposed approach is on its more modern counterpart, the Deep Q Network (DQN), created by Minh et al. [15] which is a deep learning implementation that utilizes Q Learning at its core, discussed in Section 3.1. The proposed approach builds off of this network architecture due to the advantages presented by using Q Learning, as well as utilizing the simplicity of the underlying method to determine if the proposed approach can further expand upon it.

2.1.3 Epsilon-Greedy Approach

In reinforcement learning, the trade-off between exploration and exploitation plays a large role in the overall quality of policies learned. Low exploration in the early stages of policy construction can lead to stifled policy growth, whereas if exploitation is not emphasized in the later stages of construction, the agent may not learn any optimal strategy, thus resulting in a half-baked policy either way. Though there are multiple methods to balance out the two, the proposed approach utilizes the **Epsilon-Greedy approach**.

The Epsilon-Greedy approach is defined by Sutton and Barto [24] as an **action-value method**, or a method for selecting actions based on the estimated values. This is different from Q Learning in that the method does not calculate any values. Rather, it is the means by which a policy may choose to select actions based on the values provided via functions like Q Learning.

The 'Greedy' portion of the method uses Equation 2.4.

$$a_t = \arg\max_a Q_t(a) \tag{2.4}$$

Where a_t refers to the action at a given time step t, a refers to an action $a \in A$, and $Q_t(a)$ refers to the estimated value of action a at time step t. The second half of the method, 'Epsilon', refers to some probability ϵ_t that the action a_t is selected via uniform distribution across the set of available actions A. This is done to encourage exploration in the environment and to curtail the appearance of sub-optimal policies. As $t \to \infty$, $\epsilon_t \to 0$, meaning that over time, then the probability of taking a "random" action decreases to zero. As such, this technique is only used during the training phase. During testing, or any "real-world" utilization of the policy learned, only the greedy choice is used (i.e., $\epsilon = 0$).

In the context of both Q-Learning and DQN [15], the Epsilon-Greedy method is used for action selection inside of the Q Learning algorithm, though in the latter the ϵ value is linearly annealed over the course of 1M time-steps.

2.1.4 Seaquest

Seaquest [6] is a video game created in 1983 for the Atari 2600 system. The main objective of the game is to, as a yellow submarine, pick up divers for rescue while also fending off enemies, such as sharks and other submarines. Figure 2.2 illustrates what the game looks like.

To clarify some terminology, the player "picks up" divers by simply moving their submarine over the location of the diver, such that the two entities are touching one another somehow. Conversely, if the player touches an enemy, or a projectile spawned by an enemy, they will lose a life. Additionally, the player "rescues" divers by moving up to the top of the playable area, whereupon at least one diver will be removed from the player's current diver count, shown at the bottom of the screen.

Moving on, there are confounding factors that add difficulty to the game. Namely, the player has a limited oxygen capacity, represented by a bar at the bottom of the screen. Furthermore, if the player attempts to resurface without any divers on board, they will instead lose a life. In addition to this, the player can only have up to six divers on board at a given time, though if a player surfaces with the maximum number on board, they will receive a large point total of fifty points per diver. On top of this, an additional fifty points is awarded per diver for subsequent six-diver runs, up to a maximum of 1000 points per diver. Additionally, enemies will gain ten points, up to a maximum of 90 points. However, if the player resurfaces with less than this amount, no point total is awarded. Finally, the more the player resurfaces, the more difficult the game becomes, as enemies become faster.

The reason this environment was chosen was two-fold. The simplicity of the environment, as described above, allows for an equally simple representation to be constructed from it, a suitable quality for exploratory work. Furthermore, polices

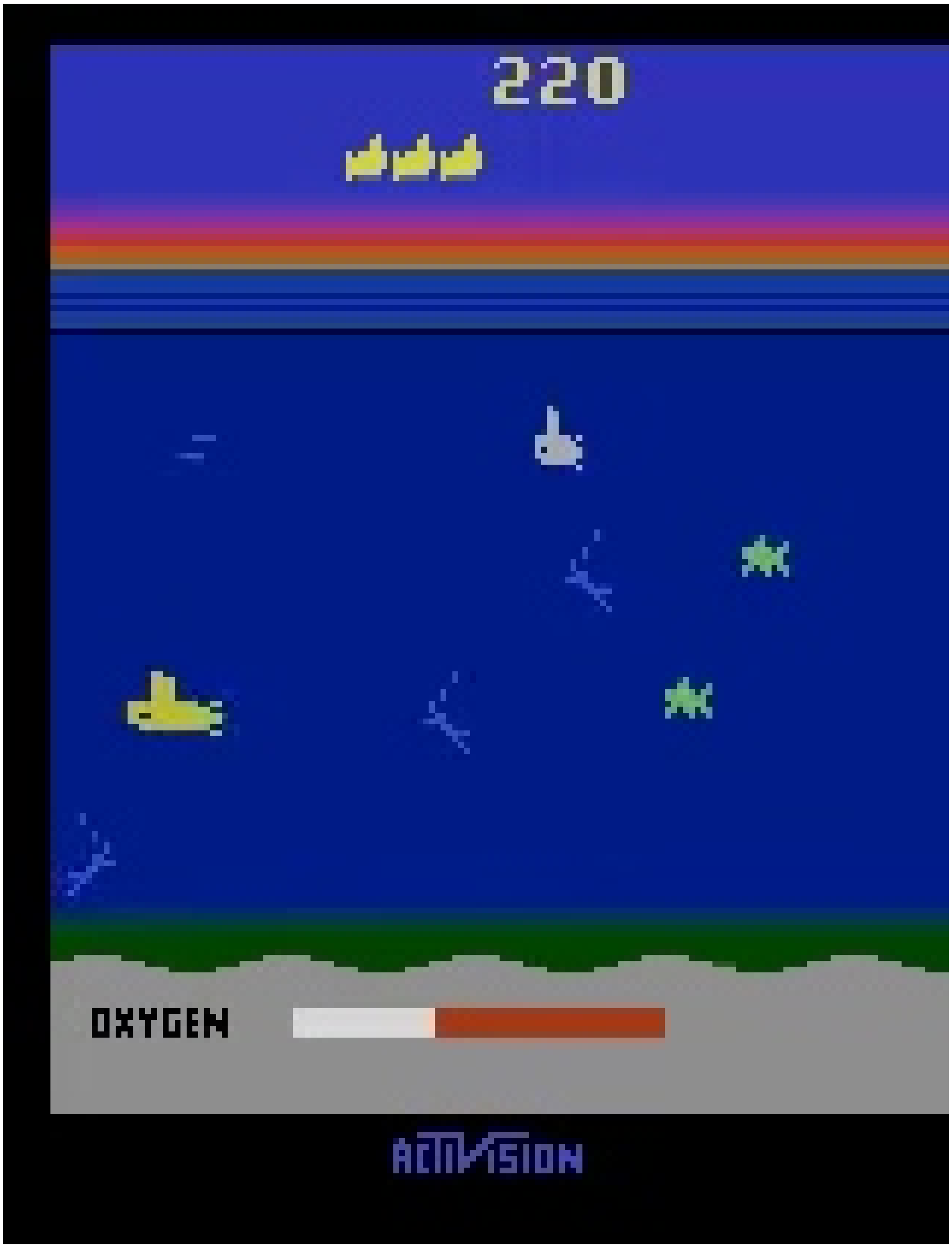


Figure 2.2: An image representing the Seaquest environment.

constructed by the DQN approach [15] seemingly do not understand focus on acquiring six divers and surfacing, choosing instead to defeat as many enemies as possible while surfacing only when absolutely necessary, sometimes failing to do even that. Based on the how the point values scale between divers and enemies, it is obvious to human players that divers are indeed the primary source of points and as such should be treated as the main objective of the game.

Since the current DQN-based approach fails to construct a policy that mirrors this knowledge, it can be inferred that the baseline policy is currently hindered by the long-term nature of such a goal, making it easy to determine if the proposed approach has any meaningful impact on the resulting policies.

2.2 Techniques Used

2.2.1 Transfer Learning

Transfer learning can be defined as a technique for improving the training efficiency of a deep learning model. This technique is performed by extracting the knowledge of a source model that was trained on a related task, and using it as a base for a related task. In the proposed approach, this is done by directly copying the parameters of the source model and applying them to the target model. A potential pitfall of this type of transfer learning is the restrictions placed on model architecture, which may prevent certain changes to the target architecture. In other words, in order for the transfer learning to work as intended, the layers and ordering of the source model must be preserved and present somewhere within the target model.

The proposed approach makes use of this in multiple ways. First, the base weights, or weights of a baseline DQN model trained on the Seaquest environment for 20M steps, was used as a base for both the proposed approach and another baseline.

Afterwards, these two approaches were trained for an additional 5M steps before testing. This was done to determine the efficacy of the proposed approach in ideal conditions, while allowing time for multiple modifications described in Chapter 4.

Another use of transfer learning was in creating auxiliary feature extractors for the environment. Similar to the previous case, a baseline DQN model was used, though only the weights in its convolutional layers were transferred over. However, due to dissimilarity in tasks, this approach was scrapped in favor of training a model from scratch.

2.2.2 Reward Shaping

Reward shaping is a technique used to introduce external influences into the development of a reinforcement learning agent's policy. In other words, reward shaping is done by either modifying the existing environment reward, or adding an additional source of reward separate from the environment. This can include positive rewards for performing certain actions, or negative rewards for entering certain states. Equation 2.5 represents the combination of the immediate reward signal from the environment r_t and the external shaping signal $r_{shaping}$.

$$r_t + r_{shaping} \tag{2.5}$$

The proposed approach makes extensive use of reward shaping to try and incentivize behaviors outlined in the Representation of Tasks described in Chapter 4. Furthermore, additional modifications described in Chapter 5 cover other attempts to use reward shaping, along with general observations made by doing so.

2.2.3 Action Shaping

Similar to reward shaping, action shaping is a technique used for introducing external influences to the development of a reinforcement learning agent's policy. However, unlike reward shaping, action shaping works by modifying the actions available to the agent A_s. Action shaping can be a viable alternative in situations where reward shaping may not be a good fit, such as preventing the agent from entering undesirable states.

In the case of the proposed approach, this is done by introducing a mask M that is applied onto the Q values $Q(s)$ which are produced at a given state $s \in S$. The mask is generated by different methods described in Chapter 4, but the result is such that $Q(s) \circ M = Q'(s)$, where certain Q values are set to zero before an action is selected using the Epsilon-Greedy approach described in Section 2.1.3.

The proposed approach attempts to use action shaping in conjunction with an auxiliary feature extractor to determine the effects of heavier external influences on the resulting policy.

2.3 Atari Learning Environment (ALE)

The Atari Learning Environment (ALE) is a wrapper around Atari emulator Stella constructed by Yavar Naddaf [2, 12]. This environment was used as a major baseline in the OpenAI Gym environment [5] and was used by Minh et al. for their DQN approach [15]. Due to the general simplicity of Atari 2600 games, it was chosen as the baseline environment for the proposed approach.

2.4 Keras RL

Keras RL [19] is a third-party library initially developed by Matthias Plappert that uses Keras to utilize aspects of the OpenAI gym library and environment. The library was chosen due to it leveraging the accessible nature of Keras. It contains various agents, including DDPG and SARSA, but for these experiments only DQN is utilized. Keras RL makes use of Processors, which serve to execute auxiliary code at certain steps in the feedback loop between the agent and the environment. This ranges from logging certain values to reward shaping based on actions or custom metrics. For experimentation, two separate processors are created to represent the baseline and experimental processor.

For comparative purposes, two different models and processors were created. The former, named Atari, was built to represent the basic DQN that was used by Minh et al. and as such contains no modifications that influence the policy. The latter, named TIDQN, was built to demonstrate the performance of policies that included various amounts of information from a teacher against the baseline. As such, it contains auxiliary feature extractors, as well as a conditional matrix to represent said information. Furthermore, to allow for better comparison, the number of divers rescued within an episode, along with a cumulative sum of divers was recorded for both. The reason for this is that the experimental goal is to determine if the inclusion of outside information will result in a higher quality policy which, in the context of Seaquest, means a policy that actively rescues six divers at a time.

Over the course of experimentation, various adjustments to the knowledge provided by a teacher have been made, whether it was due to unintended consequences of providing shaping signals based on the provided knowledge, or due to negligible performance increase when compared to the baseline. Chapter 5 covers the results of these experiments.

Chapter 3

Related Works

This chapter covers similar approaches taken by other authors and describes the overall points to each approach, along with either how this proposed approach is different, or how this approach attempts to directly extend the prior work.

3.1 Human-level Control Through Deep Reinforcement Learning

This approach by Minh et al. [15] is a fundamental paper for modern reinforcement learning, as it describes an approach that combines Q Learning with deep learning. This approach, termed Deep Q Network (DQN), refers to the use of a function approximator when using the Q learning function described in Section 2.1.2. This is done to allow for scalability in the face of increasingly complex environments.

The modern architecture makes use of a convolutional stack to process images that represent the state at some time-step, s_t, which is then passed to a simple feed-forward network. Overall, the architecture effectively processes images from the environment and converts them into Q values for all actions available to the agent. From there, the action, a_t is selected by either using the action with the highest Q value, is selected at random, per the Epsilon-Greedy method described in Section 2.1.3. From there,

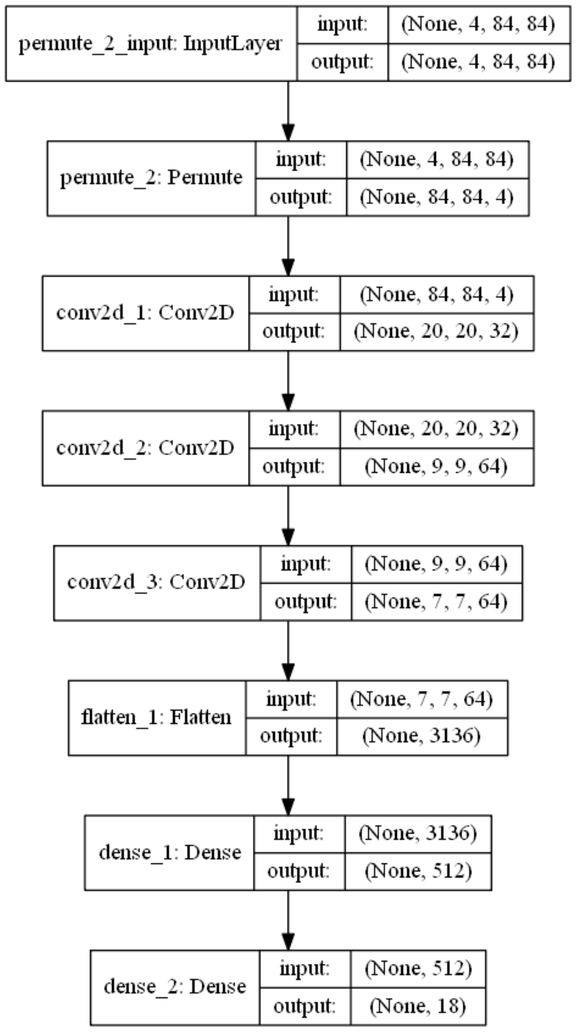


Figure 3.1: An implementation of the DQN architecture for the game Seaquest.

the agent receives a reward, r_t, and the environment moves to the next state, s_{t+1} creating the agent-environment interaction loop.

Furthermore, the authors implement a technique called experience replay to alleviate correlations between samples taken from the environment. Essentially, as the agent interacts with the environment, the resulting tuples of information $((s_t, a_t, r_t, s_{t+1})$ are aggregated into a large data structure. From there, batches are randomly sampled and used to train the network, resulting in more stability during training.

As this work forms the foundation to many modern approaches [1, 17, 4, 29], the proposed approach naturally will be extending this work as well, by proxy.

3.2 Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning

As mentioned in Section 2.1.1, Sutton et al. [25] created a framework to unify multiple approaches to temporal abstraction in reinforcement learning. This framework, termed **Options**, describes the general concept of a temporally extended sequence of actions that would facilitate more complex actions in a given environment.

The approach goes on to describe what constitutes an option $o \in \mathcal{O}$ is, where $\mathcal{O}$ referred to the set of all options. A given option o contains 3 components:

- An initiation set $\mathcal{I} \subseteq S$ where for some state $s_t \in S$, an option o is available if and only if $s_t \in \mathcal{I}$.
- A policy $\pi : S \times A \to [0, 1]$ that is followed for the duration of the option.
- A termination condition $\beta : S^+ \to [0, 1]$ that determines the probability that a state $s' \in S$ is in the set of terminal states $s' \in S^+$ for the option.

A note to make about π is that it may be limited to a given option $o \in \mathcal{O}$. However, as Sutton et al. attempted to merge the options framework with the base reinforcement learning framework, the authors instead chose to denote a "policy over options" $\mu : S \times O \to [0, 1]$, along with the assertion that primitive actions $a \in A$ can be seen as a special case of options. In other words, for a state $s \in S$, $\mathcal{O}_s$ represented all possible actions primitive and temporally extended.

This paper is fundamental in other approaches described in this chapter [1, 16] due to the notion that the principles of options can be combined with current approaches to reinforcement learning to create more robust policies. Therefore, the proposed approach will be extending this work by proxy.

3.3 Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition

Another fundamental approach to handling the complexity of the reinforcement learning problem is the MAXQ approach described by Dietterich [10], which involves an efficient approach to hierarchical reinforcement learning by way of decomposing the value function. It not only describes a framework that forms the basis of hierarchical reinforcement learning approaches, it provides several mathematical guarantees for using the MAXQ approach.

There are some conceptual similarities between the proposed approach and the MAXQ approach, namely in the idea of utilizing an external teacher or programmer to construct the graphical representation of the environment, along with the notion of reducing the complexity of the state space by way of state abstraction. Finally, the proposed approach utilizes the concept of termination predicates to control when a sub-task is finished in place of a supervisory network when utilizing the 'Option Heads' approach described in Section 3.4.

However, unlike the MAXQ approach the proposed framework uses the notion of 'task decomposition' to create hypothetical policies that could be learned or utilized via the Options framework [25]. Though Dietterich stated that a given option assumes its policy is already learned, more recent applications of the Options framework have reduced or removed the need for a learned policy.

3.4 Classifying Options for Deep Reinforcement Learning

The approach described in this paper by Arukulmaran et al. [1] is an attempt to incorporate options [25] into the structure of DQN [15] without sacrificing model performance. The architecture this paper extends is the Bootstrapped DQN by Osband et al. [17]. However, since the goals of the Bootstrapped DQN approach are largely unrelated to the proposed approach, this paper will be used instead.

The inclusion of options within the DQN architecture is achieved through the implementation of **'option heads'**. The term is likely derived from the idea that instead of a **single-head** approach, where one dense layer outputs a number of Q values equal to the number of actions in the environment, it takaes a **multi-head** approach where multiple of these layers are created and work in parallel. These option heads modify the output of the DQN such that multiple heads form the overall policy created in a given environment. In other words, a policy trained on this architecture would become $\pi(s) = \arg\max_a Q_{\mathcal{O}(s)}(s, a)$, where $Q_{\mathcal{O}(s)}$ represents a supervisory network $O(s) : \mathcal{S} \rightarrow \mathcal{O}$ which determines what 'option head' is selected at some state $s \in S$.

Training a supervisory network, however, requires manual control of which option head is trained, along with separate buffers for experience replay. The authors noted the use of an oracle during training, which also trained the proposed supervisory network.

Due to the paper successfully including sub-policies within a singular DQN model, there are efforts made in the proposed approach to utilize this, as options heads are a suitable implementation of the Representation of Tasks (RoT) discussed in Chapter 4.

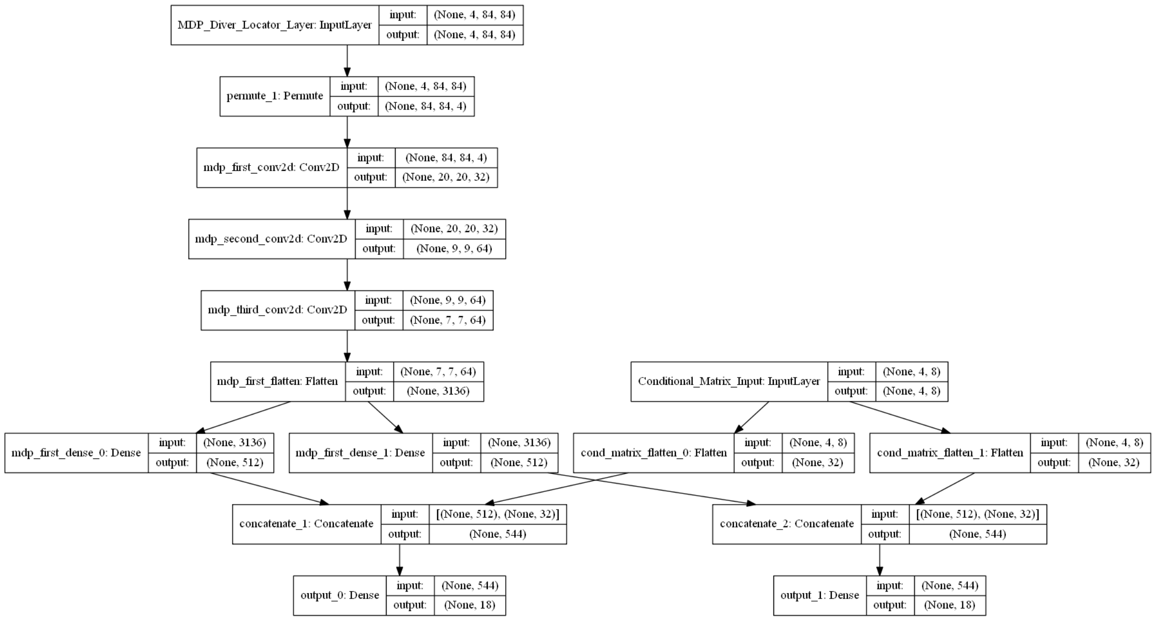

MDP_Diver_Locator_Layer: InputLayer | input: (None, 4, 84, 84) | output: (None, 4, 84, 84)
permute_1: Permute | input: (None, 4, 84, 84) | output: (None, 84, 84, 4)
mdp_first_conv2d: Conv2D | input: (None, 84, 84, 4) | output: (None, 20, 20, 32)
mdp_second_conv2d: Conv2D | input: (None, 20, 20, 32) | output: (None, 9, 9, 64)
mdp_third_conv2d: Conv2D | input: (None, 9, 9, 64) | output: (None, 7, 7, 64)
mdp_first_flatten: Flatten | input: (None, 7, 7, 64) | output: (None, 3136)
Conditional_Matrix_Input: InputLayer | input: (None, 4, 8) | output: (None, 4, 8)
mdp_first_dense_0: Dense | input: (None, 3136) | output: (None, 512)
mdp_first_dense_1: Dense | input: (None, 3136) | output: (None, 512)
cond_matrix_flatten_0: Flatten | input: (None, 4, 8) | output: (None, 32)
cond_matrix_flatten_1: Flatten | input: (None, 4, 8) | output: (None, 32)
concatenate_1: Concatenate | input: [(None, 512), (None, 32)] | output: (None, 544)
concatenate_2: Concatenate | input: [(None, 512), (None, 32)] | output: (None, 544)
output_0: Dense | input: (None, 544) | output: (None, 18)
output_1: Dense | input: (None, 544) | output: (None, 18)

Figure 3.2: A model diagram of the proposed approach's implementation of the Option Heads architecture. Note that each head is constructed exactly the same.

3.5 Hierarchical Reinforcement Learning for Playing a Dynamic Dungeon Crawler Game

This paper by Niel et al. [16] outlines an approach the authors termed Dynamic HRL (dHRL), which is based on the MAXQ framework proposed by Diettreich [10]. However, the major difference for their paper is that their framework has a "root behavior", or some root policy, which determines what sub-policy to choose from, along with how different sub-policies can be trained simultaneously.

However, it seems that while the ideas underpinning the author's approach are similar to the proposed approach, the execution is quite different, with the largest of these differences being that their approach is simply not a deep learning approach at all, using only a shallow feed-forward network to simulate Q-learning, as opposed to the DQN [15] approach the proposed approach utilizes.

3.6 Identifying Subgoals by Local Graph Partitioning

This paper by Simsek et al. [22] describes an approach for identifying sub-goals by seeing if sub-goals and, by proxy, their respective sub-policies can be identified automatically by the model during execution. The core concept revolves around the use of a given local transition graph, which is a state transition graph that restricts to only the states that the agent has recently gone through. Through this, identification of both sub-goals and potential options becomes feasible without the intervention of an external force (i.e., a teacher).

However, there are major differences between the proposed approach and this paper. One of these differences is that the paper is founded under the assumption that tabular reinforcement learning, a system in which the value function is stored using a table, is in place. However, such a system is known for not scaling to more complex problems, unlike more modern solutions [15]. Another difference is that the approaches themselves are different with respect to including external knowledge. While the proposed approach is attempting to leverage domain knowledge to identify potential sub-policies, this approach attempts to define a method such that the model itself could perform the same task.

3.7 Strategic Attentive Writer for Learning Macro-Actions

The Strategic Attentive Writer for Learning Macro-Actions (STRAW), by Vezhnevets et al. [27], approaches Deep Reinforcement Learning and HRL with the notion of implicit plans, which are plans that are defined by the network itself and updated periodically depending on the environment. Furthermore, the model includes a method for learning how long a given plan, or series of actions, should be committed to which allows more robust action in the face of unexpected circumstances or situations the agent has not seen before.

Despite potential applications in future extensions of the proposed framework, there remain some major differences between the two. One difference is the assumption of external knowledge. In the proposed approach, the framework establishes this as a component that is assumed to be present, unless otherwise stated. The focus of the authors' work, however, resides in the fact that the agent should be able to come up with new strategies and actions without external interference, whether it be from a teacher or not, similar to older approaches [22]. Additionally, the idea that the agent must perform all portions of planning is effectively antithetical to what the proposed approach, though it is not mutually exclusive.

3.8 Deep Reinforcement Learning Boosted by External Knowledge

Deep Reinforcement Learning Boosted by External Knowledge (DRL-EK) [4] describes a more modular approach to scalable reinforcement learning. Furthermore, the approach taken attempts to utilize knowledge outside of what may be readily available to the agent to boost both the training efficiency and quality of the resulting policy.

The authors describe their architecture as a collection of four modules, each with a specific task to perform. The Object Recognition Module, handled by the YOLO [21] library, takes in an image and extracts features, passing them to the next two modules.

The Reinforcement Learning module, using A3C [14] at its core, handles calculation of the value function and policy, using the input image and features from the first module. The Knowledge Base module, comprised of Dueling DQN (DDQN) [31], takes in only the features provided by the first module and outputs an action. The last module, the Action Selector module, takes in the actions suggested by both the RL module and KB module, and uses Q-learning to select which action to take (if either).

While there are quite a few similarities, including the idea of augmenting the input state with external information, along with the modular approach, key differences remain. One of these differences is the overall ‘meta-structure’ of the approaches, as the proposed approach does not make use of any Bayesian networks and only utilizes the base DQN architecture. Furthermore, the authors tested their work in a 3D partially observable environment, whereas the proposed approach is only tested in the Seaquest environment. Finally, the model does not account for human feedback, and it is noted as a future work by the authors.

3.9 Hybrid Reward Architecture for Reinforcement Learning

Hybrid Reward Architecture for Reinforcement Learning (HRA), by van Seijen et al. [26] describes an approach that attempts to address difficulties in finding an optimal value function due to complexity in the environment’s reward function.

The approach HRA uses to address this is decomposing the reward function into simpler functions and divvying them between multiple agents. However, the authors note that a multi-head model would be equivalent to having multiple agents. The authors then claim that, through this method of training, policies of similar quality can be found with greater efficiency due to each reward function being less complex.

Finally, the authors describe methods by which to further improve the performance of HRA by way of domain knowledge: By removing irrelevant features, identifying terminal states and using pseudo-reward functions, which likely includes the use of reward shaping.

The approach has some similarities to the proposed approach. The notion of using a multi-head model, with each head representing an option [25] has been proposed in other papers mentioned [1, 17], which serve as the works the proposed approach is directly extending. Furthermore, the concept of leveraging domain knowledge to boost the performance of a given agent is similar to the proposed approach. However, the largest difference is the underlying goals. While HRA focuses on the increasing efficiency by lowering the complexity of a given reward function, the proposed approach seeks to provide a human-interpretable method for injecting domain knowledge into a given environment.

Although the application described in the paper is limited to tabular methods, the underlying principles can be seen in more recent works [26, 16]. Similar to more current approaches, the proposed approach attempts a different decomposition from the one described by Dietterich, opting instead to view an environment as a potential collection of options. While options and the MAXQ approach are not inherently mutually exclusive, the proposed approach bears more resemblance to the former than the latter.

Chapter 4

Teachable-Interpretable DQN (TIDQN)

This chapter covers the framework, termed TIDQN, created for the proposed approach along with a description of what each portion does and its intended purpose. Furthermore, it describes the contribution it provides to reinforcement learning as a whole.

4.1 Teachability in TIDQN

The teachability aspect of TIDQN is present in the use of a Representation of Tasks and Completion Vector. While these tools are more conceptual additions, their implementations can be added and removed freely from the base reinforcement learning problem depending on how much external help the agent should receive. Ranging from having no state augmentation to having the set of available actions directly modified by a set of feature extractors trained by external sources, there are multiple configurations that affect the amount of teaching an agent receives and, by proxy, the policy that comes from it.

4.2 Interpretability in TIDQN

The interpretability of TIDQN is more roundabout, though it still makes use of the aforementioned Representation of Tasks and Completion Vector. Just as these tools are used to introduce external influence into the reinforcement learning problem, they can assist a human in understanding the desired outcomes an optimal policy should reflect by presenting information about the environment in an accessible format. Naturally, this strongly depends on the chosen format for the Representation of Tasks, but graphical methods will likely retain this property.

Furthermore, it can be used to determine where a sub-optimal policy may begin to deviate, though it may not provide context for why. An example of this would be, in the Seaquest environment, if an agent gains the necessary six divers to surface for extra points, but fails to move towards the surface. While the reason behind it not doing so is not immediately apparent, the fact that it is not abiding to the outcomes described by the Representation of Tasks remains clear. In other words, the interpretability for humans can also be seen as a debugging tool for determining what changes should be made to an existing approach.

4.3 Representation of Tasks

The **Representation of Tasks (RoT)** is an integral component to the framework presented. It is used to describe a relationship between tasks in the form of requirements, such that performing the tasks in the prescribed order would result in the overall goal of an environment being achieved. As such, a Representation of Tasks is some abstraction with embedded domain knowledge about what the agent should do, given the environment it is placed in.

A Representation of Tasks should additionally serve as a means by which humans

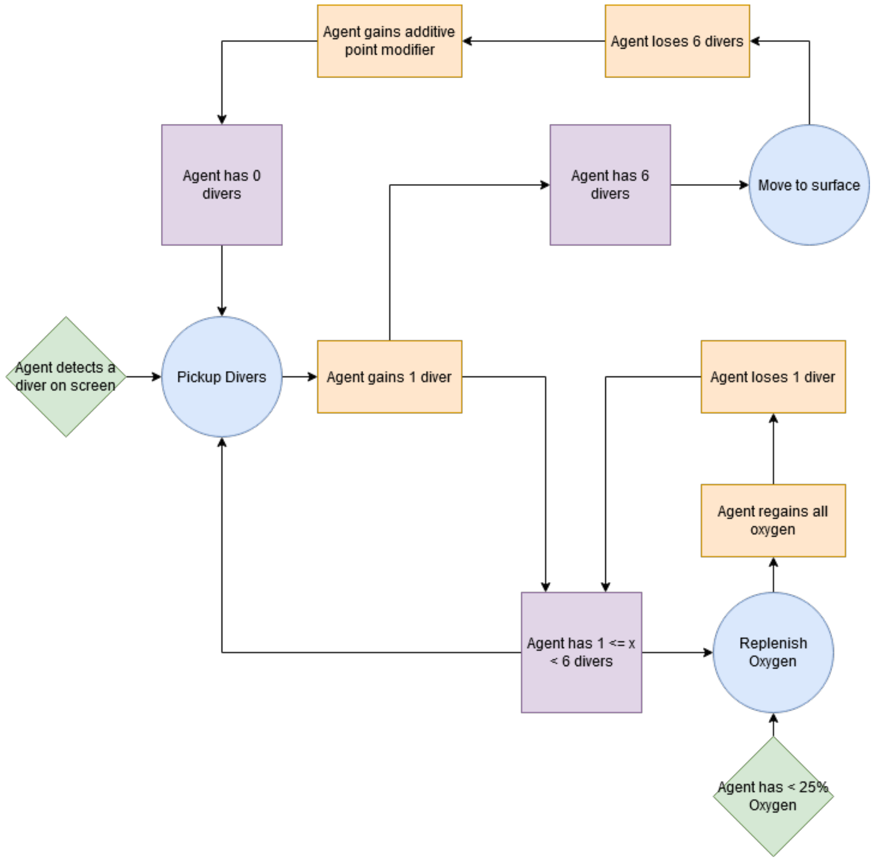


Figure 4.1: A Representation of Tasks for the game Seaquest. It shows the main objective of the environment, which is to rescue divers in groups of six.

can understand the agent's tasks and goals within a given environment. Note that, while the Representation of Tasks illustrates the overall goals an agent should have in the environment, by initializing certain portions discussed in Section 4.3, it can also provide an idea of what can be done in a particular state within the environment. This is to help create an intuitive understanding of what factors could be influencing the agent's behavior as it attempts to create a policy based on the RoT. To this end, the proposed approach makes use of a directed graph to illustrate the RoT of Seaquest, shown in Figure 4.1.

4.3.1 Explaining a Representation of Tasks

The circular nodes in Figure 4.1 are Task Nodes, or nodes that represent a potential policy that the agent could learn. As these nodes represent potential policies, it is possible to create a new Representation of Tasks for a given Task Node for a more granular description of the main RoT.

The square nodes in Figure 4.1 are Conditional Nodes, or nodes that represent hard constraints or immutable mechanics present in the environment. In the case of Seaquest, this can include something like the need to have at least one diver before being able to resurface and replenish the oxygen supply. If the agent does not meet this condition, it will lose a life and end the episode. As such, this is treated as a hard constraint. Similarly, for the agent to receive bonus points on subsequent rescues of six divers, the agent must surface with exactly six divers, or else receive no bonus and lose only one diver similar to resurfacing for oxygen. While failing this Condition Node does not explicitly end the episode, it remains a hard constraint with respect to achieving a given result.

The diamond nodes in Figure 4.1 are Auxiliary Nodes, or nodes that are akin to Conditional Nodes, except that they can be added and removed without impacting the flow of the RoT. In other words, these are additional rules that a teacher may place on the agent in order to improve the resulting policy.

Finally, the rectangular nodes, though not necessary for the RoT, show the effects that occur within the environment after a given Task Node is completed. Multiple effects that occur are chained together for clarity.

For all nodes present, any directed edge that connects some node A to B indicates a precedence relationship, meaning that for B to be eligible for completion, A must be completed. In the case of Task Nodes, all Conditional and Auxiliary Nodes must be true in order for it to be eligible for selection. For Seaquest, an example of this can be seen in the Task Node “Replenish Oxygen” where, in order for the task to be eligible for selection, its conditions “Agent has $1 \leq x < 6$ divers” and “Agent has $\leq 25\%$ O_2” must evaluate to True.

A more formal definition of the Representation of Tasks $\mathcal{T}$ is as follows. $\mathcal{T}$ is defined as an abstraction of the root task on a given environment $\mathcal{E}$, that is constructed by some teacher that has domain knowledge about $\mathcal{E}$.

For example, using the RoT derived in Figure 4.1, a teacher constructs $\mathcal{T}$ as a vector of boolean values of size $n = 5$, representing all Conditional and Auxiliary Nodes present. An observation $\mathcal{E}$ from the environment Seaquest comes as a grayscale image, which is flattened into a vector of pixel values. The agent then receives a concatenation of these two vectors to determine which action it will take next.

For the Seaquest environment, the Representation of Tasks will be constructed as a directed graph. The reason for this is that, due to the simplicity of the environment, there is no need to have a more sophisticated representation to model it. Furthermore, it is conceptually simpler to understand where certain inclusions are located and how these inclusions may interact with the resulting behavior the agent exhibits. However, for more complex environments, a multitude of these representations may be necessary to fully encapsulate both the mechanical aspects and meta-knowledge present.

4.3.2 Limitations of the Representation of Tasks

While the RoT is meant more as a tool for understanding the environment and policies created within it, there are a few limitations that arise. Some of these limitations are dependant on the environment, but there are others that are inherent to the chosen means of illustrating the RoT.

In an ideal scenario, all Task Nodes would have a policy trained specifically for handling said task. However, this is not necessarily feasible in certain environments. For example, in Figure 4.1 there is a Task Node for returning to the surface with six divers in tow. Though the contents of the Task Node may be simple enough for a policy to be created, the Condition Node complicates matters as acquiring six divers is notably quite difficult for an agent. Furthermore, since Seaquest, like several other environments, lacks the ability to manipulate the environment to create more specialized scenarios, creating a policy for the Task Node becomes quite arduous.

The proposed approach implements the Representation of Tasks as a directed graph, due to the simplicity of its corresponding environment and accessibility to those unfamiliar with said environment. However, this particular representation does not remain as accessible with more complex environments, even if it can be represented in this format. Furthermore, it does not capture any non-deterministic relationships between Condition Nodes and Task Nodes which, while not necessary in the case of Seaquest, may be necessary for representing environments where randomness is more pervasive. This can be addressed by simply changing what is used for the Representation of Tasks.

4.4 Completion Vector

Another component of the framework is the Completion Vector $\mathcal{C}$. The Completion Vector contains values corresponding to Conditional Nodes and Auxiliary Nodes in the Representation of Tasks. The purpose of the Completion Vector is to augment the observation V (e.g., the pixel information of the four most recent game frames of Seaquest) with the domain knowledge that is encoded in the Representation of Tasks. By including this information, similar states can be differentiated more easily by the agent.

However, the Completion Vector may have more elements present than the total number of Conditional and Auxiliary Nodes present in the Representation of Tasks. In the case of Seaquest, the Completion Vector utilized by the proposed approach is a vector of eight boolean values. The first six values correspond to the number of divers in unary. The remaining two values correspond to the Auxiliary Nodes "Diver detected on screen" and "Oxygen below 25 percent".

In practice, the Completion Vector can be defined as $\mathcal{C} = f(V)$, where f refers to some vector-valued function that takes an observation from $\mathcal{E}$ as a vector V and outputs information about it. In Seaquest, f takes in the pixel values of the current frame and outputs the information described earlier. Generally speaking, $\mathcal{C}$ will include information defined in a Representation of Tasks $\mathcal{T}$, but is not restricted by what is present in $\mathcal{T}$. For state augmentation, including $\mathcal{C}$ in the observation would result in a new augmented observation $V' = V \odot \mathcal{C}$, where $\odot$ refers to the concatenation of the two vectors.

In the proposed approach, the Completion Vector is predominately used for reward shaping, discussed in Section 2.5, though discussion of its use in action shaping is discussed in Chapter 6.

4.5 Auxiliary Feature Extractors

In an ideal scenario, all information that is present in an environment would be readily available, even if it is not immediately utilized by the agent. However, it is more likely that the teacher will need a piece of information that is simply unaccounted for in the environment. In the case of Seaquest, the location of divers serves as a version of this issue. In both cases, this information could be valuable in informing the agent's behavior and improving the policy, thus driving a need for some means of extracting this information.

To address this, an Auxiliary Feature Extractor (AFE) can be used to pull the information from the environment. Furthermore, the Representation of Tasks can be used as a guiding tool for determining which pieces of information should be pulled from the environment.

A question that arises is how to incorporate Auxiliary Feature Extractors into an existing Representation of Tasks. The proposed approach handles this by assuming information gained through an AFE to correspond to an Auxiliary Node to simplify the changes made. However, if the AFE is extracting information that is used to directly evaluate an integral aspect of the game, it is reasonable to treat its output as a Conditional Node instead. In the case of Seaquest, this distinction is made between the 'diver locator' AFE and the 'number of divers held' AFE, respectively.

While the proposed approach utilizes multiple AFEs, the only AFE that is considered in Chapter 5 is the 'diver locator' AFE, which attempts to classify the positions of divers in dynamically-sized quadrants relative to the agent's position. However, said AFE is used for both reward shaping and action shaping, discussed in Sections 2.2.2 and 2.2.3 respectively, and are evaluated separately to determine the impact of an AFE on the resulting policy.

4.6 Proposed Modifications

As the goal for the game Seaquest is to rescue divers in batches of six, many modifications are centered around the acquisition and detection of divers. However, there were other more indirect attempts to influence the agent's policy. For all modifications, it is compared against both the baseline DQN and a baseline version of the proposed approach that only contains the reward shaping for picking up divers when applicable.

4.6.1 Diver Rescue

The core issue present in the original approach is the disconnect between the agent's policy and the long-term goal of the environment, which is to rescue divers in batches of six. This goal, however, is not immediately apparent as the environment does not award any points for picking up divers nor for surfacing with any number of divers less than six. In other words, the only reward signal the agent is likely to receive from the environment is for defeating enemies, which do award an immediate point value. As such, policies trained in this fashion end up skewed towards defeating enemies efficiently, despite this not being the goal of the environment.

The proposed approach attempts to rectify this issue by using reward shaping, described in Section 2.2.2 to shift the policy towards this goal. Hyperparameter searching was conducted alongside building a heuristic system to find the best magnitude of reward signals to ensure the resulting policy was able to perform similarly to the original, whilst making active attempts to achieve the original goal of the environment.

4.6.2 Removal of State Augmentation

An initial modification to the TIDQN framework is the removal of state augmentation. By removing state augmentation, the original DQN [15] architecture can be used, though modifications such as reward shaping and the presence of other feature extractors are still feasible. This is done to determine if state augmentation, or potentially its implementation, causes detrimental effects independent of other modifications. While it felt necessary to determine the overall impact of having state augmentation in the first place, the 'basic' form of TIDQN does assume that state augmentation is used. Therefore, other modifications are assumed to be done on a state-augmented TIDQN unless otherwise specified.

To differentiate this modification of TIDQN from the others, it is termed ‘No State Augmentation’ in Chapter 5, with ‘State Augmentation Only’ referring to the ‘basic’ TIDQN counterpart.

4.6.3 Removal of Environment Reward

Due to the presence of an easily gained short-term reward within the environment of Seaquest, it is hypothesized that the agent may not be given many opportunities to engage with the long-term objective. In other words, the policy ends up learning to only attack enemies and rescue divers as needed due to having significantly more opportunities to learn the former objective.

This approach sought to remove that issue by zeroing out all rewards from the environment such that all sources of positive and negative reward came directly from reward shaping. By doing so, the teacher has much more granular control over what behaviors the agent should be rewarded for and the agent has little to no ‘distraction’ from the environment. This is also done to determine the impact of using the TIDQN to remove readily available reward sources on the resulting policy.

4.6.4 Adding Feature Extractors

To gain more granular control over the reward shaping process for an agent, it seemed necessary to employ additional feature extractors. These Auxiliary Feature Extractors (AFE), however, would not be directly tied to the DQN model and therefore were not part of the state augmentation. Instead, these feature extractors were part of the Processor, directly influencing the policy instead.

The main feature extractor employed was one to locate divers based on their current quadrant, illustrated in Figure 4.2. Assuming the information was correct, then additional rewards could be provided in the form of positive reinforcement being given when the agent moved towards a quadrant containing a diver. Below are the different approaches taken to try and create a feature extractor that could be used reliably for this purpose.

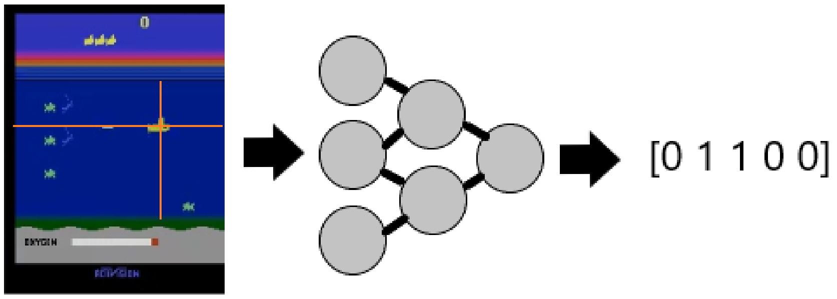


Figure 4.2: An example of the diver locator auxiliary feature extractor transforming an observation V into quadrants were divers are located. The bits correspond to the following quadrants: no divers, upper left, lower left, upper right, lower right. The orange line shows a representation of the quadrant used to determine labels and is not present in practice.

4.6.4.1 Heuristic Location Classifier

In order to quickly determine the location of divers, a heuristic method was initially employed. The method used a 2×2 kernel, and thresholds on both the mean value and standard deviation to determine if the kernel was centered on a diver. While the heuristic worked quite well in most scenarios, it became inconsistent when divers were partially obscured by enemies. Furthermore, it was clear that the approach taken increased training time significantly and would likely not scale well, nor generalize to other environments.

To address this issue, a computational graph was constructed such that the GPU would handle the bulk of the operations. Additionally, in place of threshold values a small stack of dense layers were implemented. Despite the drop in computational cost its performance degraded, likely due to the use of a neural network as opposed to a purely heuristic approach. Therefore, the approach was shifted towards a deep learning approach to see if similar performance could be gained.

4.6.4.2 Single-Label Location Classifier

In order to move away from hard-coded heuristics, it became clear that extracting the necessary information would require the need for some sort of feature extraction from the environment. However, Seaquest and other Atari environments do not have such auxiliary information readily available. As such, convolutional neural networks were utilized in the task of extracting features from the environment for use in the conditional matrix.

The model was a direct copy of the original DQN, with additional dense and dropout layers between the first dense layer and output layer. A more detailed description of the model used for these experiments can be found in Appendix A. Initially, transfer learning was attempted by loading in weights from a well-trained vanilla DQN, but this was abandoned due to poor classification performance.

To train the feature extractor, the dataset was comprised of images of different policies during testing. Videos taken during testing were decomposed into images at a sample rate of 1/4 FPS (1 image per 4 seconds) and labeled by hand. The criterion for labeling was the presence of divers with respect to the agent, with each label corresponding to a quadrant (e.g., up-left, down-right, etc.) with an additional 'no divers' label. The diver orientation (e.g., facing left vs. facing right) was disregarded to maintain consistency in labeling. Furthermore, in the event of multiple labels being possible, the diver that was closest to the submarine was used for determining the label.

This initial attempt did not have desirable results, with a potential cause being a mismatch between the dataset and model's construction. Upon further investigation of examples the model consistently labeled incorrectly, it became clear that many of these instances contained multiple divers where both the true and predicted labels could apply. While human labeling on these instances used a heuristic of labeling the quadrant with the diver closest to the player, the model was not able to learn this information. Therefore, modifications to this approach were made.

4.6.4.3 Multi-Label Location Classifier

In order to better utilize the dataset, the labeling process and model were both modified to match a multi-label problem. These changes included making use of intermediate data structures to house applicable labels and changing the activation function of the model to a sigmoid function. Furthermore, changes to the expected input were modified to better match the expected input once placed inside of the Processor described in the Keras-RL section to streamline the prediction process.

Ultimately, this approach was chosen for use in auxiliary feature extraction. While a heuristic method was seemingly more accurate compared to the multi-label classifier, its implementation was cost-prohibitive and could not be incorporated easily into the overall architecture. Further investigation into the performance of the Auxiliary Feature Extractor can be found in Appendix A.

The multi-label AFE was used for both reward shaping and action shaping, as well as the 'supervisory network' for changing between heads in the Option Heads approach described in Section 3.4.

In both cases, the set of appropriate actions was calculated by masking actions based on the output of the AFE. In the case of reward shaping, a small positive

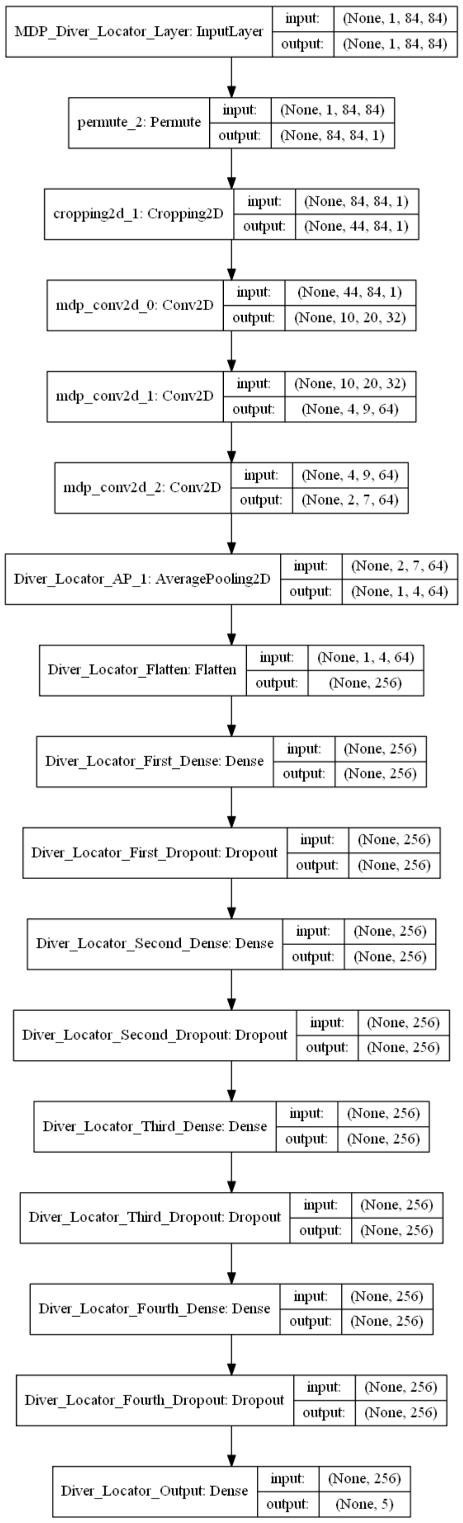


Figure 4.3: A model description of the multi-label diver locator

reward of .0001 was awarded to the agent for taking an action that was within the set of actions described, with no additional reward otherwise. However, in the case of action shaping, the set of appropriate actions was used as a mask over the set of all actions, restricting the agent directly.

4.6.5 Action Shaping using Auxiliary Feature Extractors

As discussed earlier, the output of the Auxiliary Feature Extractor was used in multiple ways. While initially the AFE was used only for reward shaping, it became clear that reward shaping may not be a good fit for the task the AFE was trained on. Therefore, rather than using the output of the auxiliary feature extractors for reward shaping, they would instead be converted into a Q Value mask that would restrict the available actions.

In the case of Seaquest the diver locator, which serves as the auxiliary feature extractor, will generate a prediction of where it thinks divers are located with respect to the agent submarine. Assuming divers are indeed present, a boolean mask that filters out actions that would not move the agent closer to a diver is generated. This mask is then applied to the Q Values generated by the policy, before any action selection can occur. Equation 4.1 shows a mathematical representation, where $Q'(s)$ is the Hadamard product of the original Q Values generated at state $s \in S$ and the mapping function M which takes the prediction of feature extractor $E(s)$ to generate the boolean mask.

$$Q'(s) = Q(s) \circ M(E(s)) \tag{4.1}$$

Though this modification introduces much more external influence, the goal is to understand if such methods will result in more desirable policies. Furthermore, in the range of methods of incorporating teachability into TIDQN, this modification has the highest level of external intervention of the ones proposed, allowing an indirect look into the effect different levels of teaching have on the agent.

4.6.6 Transfer Learning from vanilla DQN

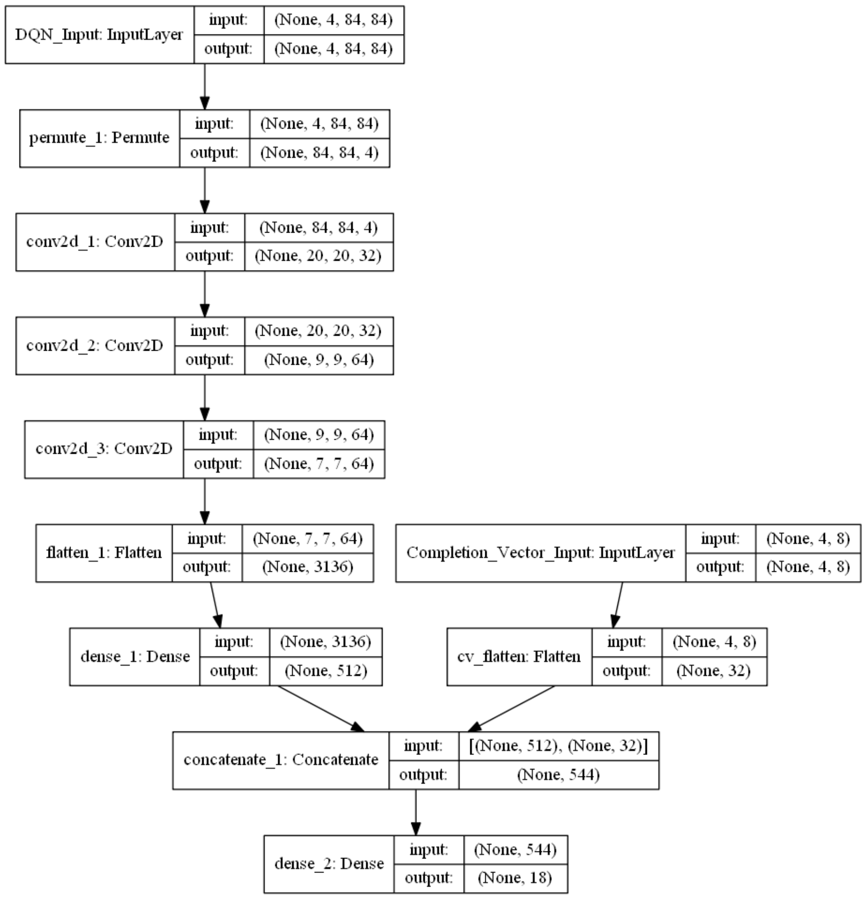
Originally, both approaches were trained completely from scratch for approximately 5M steps before evaluation. However, it became clear that the vanilla agent did have the overall capacity to achieve the intended goal without any additional modifications, with a handful of runs reaching six divers. However, it was those runs that also made it clear the agent had little understanding of the importance rescuing divers had, as these moments all ended with the episode ending before the agent could surface.

As such, a different approach was proposed. By loading in the weights of a well-performing vanilla DQN before training the resulting policy should, intuitively speaking, perform much better for both approaches. This in turn would remove a potential confounding factor of 'insufficient training' when comparing the resulting modifications and their impact on overall policy.

To determine the efficacy of introducing external knowledge via the Representation of Tasks and conditional matrix, we compared the performance using the DQN network proposed by Minh et al. [15]. The reason DQN was chosen over more recent and/or sophisticated architectures is the underlying simplicity of DQN discussed in Section 2.1.2, making it a prime target for introducing new adjustments to the framework.

4.6.7 Modifying the DQN Architecture

A major limitation of using DQN as the basis for TIDQN, shown in Figure 4.4 is the inability to have separate policies for a given environment. As such, all Task Nodes in the Representation of Tasks ultimately compete against one another for relevance, leading to a more generalized, but potentially underperforming policy. Therefore, to



DQN_Input: InputLayer | input: (None, 4, 84, 84) | output: (None, 4, 84, 84)
permute_1: Permute | input: (None, 4, 84, 84) | output: (None, 84, 84, 4)
conv2d_1: Conv2D | input: (None, 84, 84, 4) | output: (None, 20, 20, 32)
conv2d_2: Conv2D | input: (None, 20, 20, 32) | output: (None, 9, 9, 64)
conv2d_3: Conv2D | input: (None, 9, 9, 64) | output: (None, 7, 7, 64)
flatten_1: Flatten | input: (None, 7, 7, 64) | output: (None, 3136)
Completion_Vector_Input: InputLayer | input: (None, 4, 8) | output: (None, 4, 8)
dense_1: Dense | input: (None, 3136) | output: (None, 512)
cv_flatten: Flatten | input: (None, 4, 8) | output: (None, 32)
concatenate_1: Concatenate | input: [(None, 512), (None, 32)] | output: (None, 544)
dense_2: Dense | input: (None, 544) | output: (None, 18)

Figure 4.4: The single-head TIDQN model.

address the issue surrounding implementing options within a DQN, modifications to the original architecture were necessary. These modifications were initially proposed and implemented in Deep Exploration with Bootstrapped DQN by Osband et al. [17], where the original DQN structure was augmented with additional, identical heads that were trained independently of one another. For comparison, please refer to Figures 4.4 and 3.2 for the single-head TIDQN and multi-head TIDQN respectively.

However, this implementation treated each head as differing choices meaning that, in the context of TIDQN, it was no different than having a single network. A subsequent paper by Arukulmaran et al. [1] re-framed Bootstrapped DQN to better match the options framework by having each head specialize in a task specified by the developers, noted in Section 3.4. This particular modification proves invaluable in fully realizing the Representation of Tasks, as it allows for Task Nodes to be trained independently while sharing common features between them. Due to a lack of openly available implementations of either architectures in Keras, it was necessary to re-implement within the Keras-RL library.

Additional modifications from the Option Heads framework were made for use in the proposed approach. Namely, the inclusion of state augmentations $\mathcal{C}$, along with the use of heuristic methods instead of a supervisory network $O(s)$ and the implementation of action shaping for the diver-related option head. These modifications are tested against the 'vanilla' implementation and the baseline DQN approach to determine efficacy.

Chapter 5

Experimental Setup

This chapter will cover the details regarding how experimentation was carried out, including a description of metrics used, different modifications to the proposed approach that were taken, as well as what steps were taken to implement certain modifications.

5.1 Training Setup

In order to ensure that a fair comparison was made between the proposed approach and original DQN, a baseline was necessary. Initially, the policies were trained from scratch for 2M steps for early exploratory work. For assessing model performance, however, transfer learning was used to emulate the differences in policy when both models were trained sufficiently well.

To this end, transfer learning, described in Section 2.2.1, was used to get the base weights of a baseline DQN model trained for 20M steps, after which the models were trained an additional 5M steps to gauge the efficacy of each alteration made to the proposed approach. Additionally, the base weights were tested separately without any additional training to verify that the policy learned was of good quality.

Finally, both approaches were trained on 10 randomly selected 5-digit training seeds, the values of which can be found in Appendix B. This was done to account for randomness in end performance due to choice of training seed.

5.2 Testing Setup

All approaches were tested for 100 episodes, with footage of performance taken per episode to help analyze unusual performance in certain seeds. Similar to training, testing occurred over 10 randomly selected 5-digit seeds, for a total of 100 different test runs. These results are then combined and analysed to show the average performance of each approach over various metrics discussed below.

5.2.1 10×10 Testing

As mentioned in previous sections, 10×10 testing, in reference to the use of multiple training seeds and testing seeds, is used to accommodate comparative analysis of both approaches and any additional modifications. Furthermore, since 10×10 testing aggregates and averages any metrics tracked so as to get the overall performance, all metrics below are described separate from the notion of 10×10 testing.

In addition, for comparative analysis of all metrics described below the mean, standard deviation, minimum, and maximum values of each metric are tracked to provide a comprehensive view of how a policy using each approach would perform overall. Furthermore, a histogram is used to graphically display the Max Divers per Episode metric, as it was better visualized via frequency.

5.2.2 Combining Proposed Methods

Although testing individual modifications provides more concrete evidence of their impact on policy performance, it seems unlikely that only one modification is used in practice. Therefore, for all non-baseline approaches, the following combinations of modifications were tested:

- No State Augmentation
- No State Augmentation plus Action Shaping via Auxiliary Feature Extractor
- State Augmentation (i.e., baseline TIDQN)
- State Augmentation plus Reward Shaping via Auxiliary Feature Extractor

Even with different modifications, the TIDQN framework is still applicable to all combinations listed. This is due to the fact that state augmentation is tied directly to whether the Completion Vector is visible to the agent, the use of an AFE can be treated as including Auxiliary Nodes to an existing RoT, and both reward shaping and action shaping as a whole are only guided by the use of an RoT and otherwise operate independently of an RoT. In other words, for any given modification to DQN, as long as it does not change the set tasks that comprise the core of an environment, then the TIDQN framework should still apply to it.

5.3 Description of Metrics

Since the overall goal of the proposed approach is not easily quantified by only the episode reward over timestep, alternative metrics were necessary to determine the efficacy of modifications made by a teacher. This section covers the various metrics created, along with how these metrics are tracked.

The following metrics were chosen alongside standard reward-based metrics due to the environment itself not providing a base reward for rescuing divers, despite it being a core objective in this environment. Furthermore, it allows for a more granular comparison of policies trained to determine if a given approach or modification successfully outperforms the baseline.

5.3.1 Episode Reward

While not a custom metric, episode reward remains an important statistic for experiments. This metric corresponds to the reward gained by an agent throughout a single episode. In the case of the environment Seaquest, an episode ends when the agent loses all lives available. Furthermore, points are only awarded in the following circumstances:

- The agent shoots an enemy
- The agent rescues a batch of six divers

In both instances, the reward values are clipped to the range of $[-1, 1]$. Due to how Keras RL implements the train and test portions of DQN, this clipping occurs at both train and test times. Though this makes determining if the agent successfully learns the long-term goal difficult, when combined with the other metrics, episode reward helps demonstrate the changes the policy undergoes when using the proposed approach.

5.3.2 Max Divers Per Episode

As its name suggests, this metric is meant to track the highest number of divers an agent acquires before the episode ends. This metric is reset to its initial state (0) at the end of each episode, when the agent loses all lives. Due to its episodic nature, this metric highlights the average performance of a given policy at rescuing six divers within an episode.

5.3.3 Total Divers Per Episode

This metric refers to the total number of divers picked up in an episode. Since this total is not reset due to its per-timestep nature, this metric can also be viewed as a cumulative sum of divers picked up during a particular test run. It is used to determine whether the inclusion of diver-related reward shaping has any meaningful impact on the agent's policy.

5.3.4 Rate of Diver Acquisition

The Rate of Diver Acquisition describes the number of steps it takes an agent to move from k to $k+1$ divers in tow (e.g., from four to five divers). This metric is slightly more complex than the others described, due to the fact that the transition $k \to k+1$ can occur multiple times within a single episode. For example, if an agent moves from two to three divers, then loses a life, then the agent is set back to two divers. This, in turn, means there will be at least two recordings of the number of steps taken to move from two to three divers for that particular episode. Conversely, if an agent never reaches some k value of divers (e.g., the agent never gets more than three divers, therefore $k = 4, 5, 6$ are never reached) then there would be no step count for those values.

To try and extract meaningful relationships between the approaches through this metric, a few additional steps were necessary. First, to account for both possibilities described earlier, each episode contained six bins, one for each transition possible. Within a given bin $b \in B$, a list structure, empty by default, was used to contain the number of steps the agent took to acquire the subsequent diver. Multiple instances were appended to their respective lists. The equation below describes this structure.

Table 5.1: Episode reward values over a 10×10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation.

Episode Reward	Reward Mean	Reward STD	Reward Minimum	Reward Maximum
Baseline DQN	**152.839**	5.621	**140.180**	**166.560**
No State Augmentation	102.686	3.119	95.370	109.930
State Augmentation Only	33.789	1.238	30.550	36.790
Option Heads No State Augmentation	5.565	**0.568**	4.737	6.879
Option Heads State Augmentation Only	4.459	0.931	2.928	6.144

$$B = [b_1, b_2, b_3, b_4, b_5, b_6]$$

$$b_i = [ts_1, ts_2, \ldots, ts_n]; b_i \in B; n, ts \in \mathbf{Z}$$

Once the per-episode values were aggregated, they were summed together with an auxiliary structure that tracked how many instances of each transition occurred. This also indirectly provided corroborating evidence for the Max Divers per Episode metric, though that metric does not account for multiple instances occurring in the span of one episode. From this point, the summed values were divided by the auxiliary data structure to average the values, which is done to ensure that the number of times a transition occurs within each episode is preserved in calculation.

5.4 Results and Discussion

5.4.1 Diver Rescue

As the core task presented in Seaquest, nearly all tables presented in Chapter 5 are centered around this particular task. As discussed earlier, all tables will include the mean, standard deviation, minimum and maximum for each metric tracked.

For Diver Rescue, the most important metrics can be seen in Figure 5.1, alongside

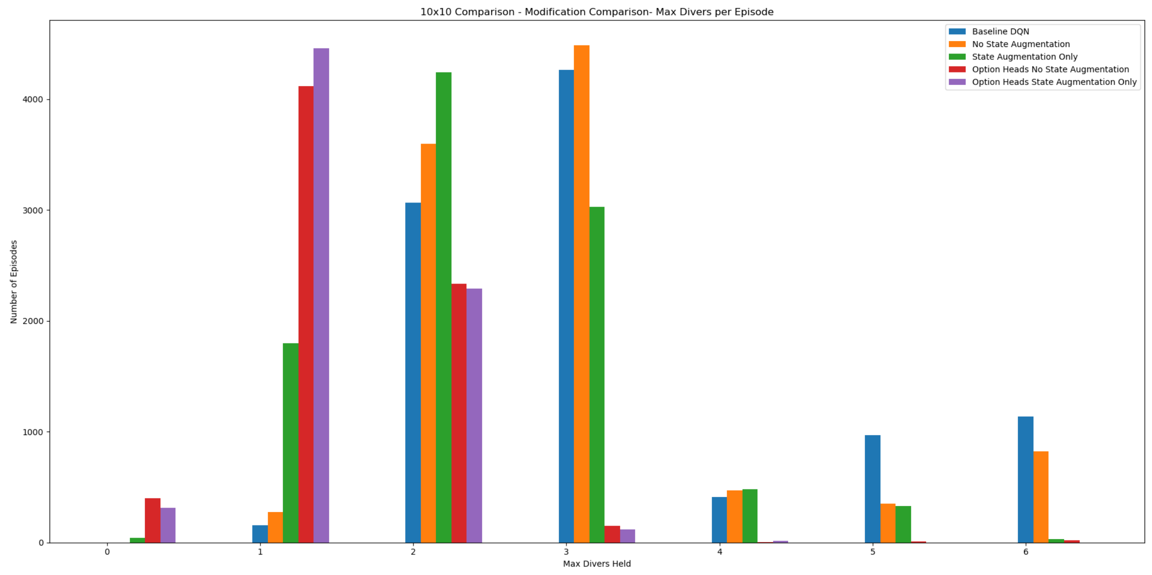


Figure 5.1: The aggregate max divers per episode over 10×10 averaging, comparing the inclusion and exclusion of state augmentation against the baseline. The x-axis represents the highest number of divers found in an episode, with the y-axis representing the number of episodes recorded for a given x-value.

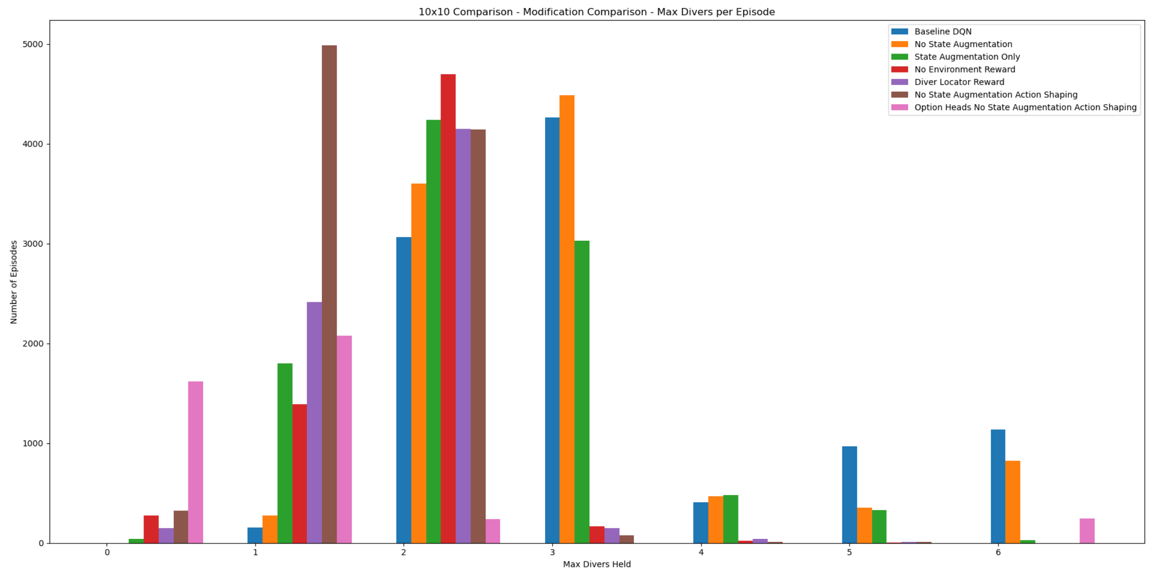


Figure 5.2: The aggregate max divers per episode over 10×10 averaging, comparing combinations of state augmentation with other modifications against the baseline DQN and baseline TIDQN. The x-axis represents the highest number of divers found in an episode, with the y-axis representing the number of episodes recorded for a given x-value.

Table 5.2: The highest number of divers rescued per episode over a 10 × 10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation.

Maximum Divers per Episode	Mean	STD	Minimum	Maximum
Baseline DQN	**3.238**	0.109	**3.020**	**3.530**
No State Augmentation	2.949	0.105	2.670	3.290
State Augmentation Only	2.310	**0.100**	2.080	2.550
Option Heads No State Augmentation	0.953	0.172	0.707	1.364
Option Heads State Augmentation Only	0.975	0.236	0.598	1.320

Table 5.3: Total divers rescued per episode over a 10×10 average comparing the inclusion and exclusion of state augmentation against the baseline DQN. All non-baseline approaches in this table only used reward shaping alongside state augmentation.

Total Divers per Episode	Mean	STD	Minimum	Maximum
Baseline DQN	**224.726**	129.532	**4.430**	**445.750**
No State Augmentation	188.981	108.740	3.770	374.500
State Augmentation Only	182.594	104.295	3.710	361.030
Option Heads No State Augmentation	62.064	31.406	1.808	117.202
Option Heads State Augmentation Only	70.271	**29.455**	2.072	105.577

Table 5.4: Episode reward values over a 10 × 10 average comparing the various modifications to the state augmented TIDQN against the baseline DQN. All non-baseline approaches in this table used reward shaping in addition to the modification listed.

Episode Reward	Reward Mean	Reward STD	Reward Minimum	Reward Maximum
Baseline DQN	**152.839**	5.621	**140.180**	**166.560**
State Augmentation Only	33.789	1.238	30.550	36.790
No Environment Reward	2.552	0.640	1.670	3.920
Diver Locator Reward Shaping	3.270	0.490	2.320	4.330
No State Augmentation Action Shaping	7.424	0.523	6.347	8.816
Option Heads No State Augmentation Action Shaping	1.078	**0.193**	0.781	1.656

Table 5.5: Maximum number of divers rescued per episode over a 10 × 10 average for each approach and modification.

Maximum Divers per Episode	Diver Mean	Diver STD	Diver Minimum	Diver Maximum
Baseline DQN	**3.238**	0.109	**3.020**	**3.530**
State Augmentation Only	2.310	0.100	2.080	2.550
No Environment Reward	1.138	0.418	0.630	1.910
Diver Locator Reward Shaping	1.135	0.303	0.760	1.760
No State Augmentation Action Shaping	1.389	**0.080**	1.173	1.561
Option Heads No State Augmentation Action Shaping	0.420	0.266	0.177	1.260

Table 5.6: Total divers rescued per episode over a 10×10 average for each approach and modification.

Total Divers per Episode	Diver Mean	Diver STD	Diver Minimum	Diver Maximum
Baseline DQN	**224.726**	129.532	**4.430**	**445.750**
State Augmentation Only	182.590	104.290	3.710	364.680
No Environment Reward	68.670	26.750	2.380	110.540
Diver Locator Reward Shaping	81.200	37.720	2.600	142.610
No State Augmentation Action Shaping	98.289	53.073	1.684	174.918
Option Heads No State Augmentation Action Shaping	24.590	**12.448**	0.802	45.750

Table 5.7: Summary Diver Acquisition Rates over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers for all approaches. Lower values indicate the agent acquiring the subsequent diver faster. Italicised values indicate the number of transitions that contributed to the value were not high enough to indicate the policy was able to reach said transition reliably.

Average Diver Acquisition Rate	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Baseline DQN	202.649	113.626	137.163	119.895	**106.364**	92.524
No State Augmentation	**98.301**	90.315	162.562	116.435	115.712	**92.497**
State Augmentation Only	163.266	123.022	149.498	99.963	158.747	94.324
Option Heads No State Augmentation	356.808	97.009	137.046	133.333	*1679.000*	NaN
Option Heads State Augmentation Only	377.228	108.953	162.671	4063.667	NaN	NaN
State Augmentation No Environment Reward	183.311	57.261	597.807	966.038	167.000	NaN
State Augmentation Diver Locator Reward	148.891	63.767	655.607	200.500	*6.111*	NaN
No State Augmentation Action Shaping	325.159	36.898	**122.515**	**78.363**	118.850	*6.000*
Option Heads No State Augmentation Action Shaping	1434.828	**17.873**	*53.000*	NaN	NaN	NaN

Table 5.8: Diver Acquisition Rates over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (Baseline DQN)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	202.649	113.626	137.163	119.895	106.364	**92.524**
Step STD	23.692	**7.580**	11.807	12.815	14.156	23.432
Step Minimum	146.440	96.969	108.151	87.868	82.174	**38.300**
Step Maximum	253.755	**131.450**	165.640	146.088	162.000	148.429
Number of Transitions Recorded	13523	14107	9304	4069	2737	1298

Table 5.9: Diver Acquisition Rates of the 'No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (No State Augmentation)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	98.301	90.315	162.562	116.435	115.712	**92.497**
Step STD	12.991	**9.432**	12.598	18.615	32.038	35.467
Step Minimum	72.059	65.746	130.065	79.750	23.000	**11.667**
Step Maximum	124.372	**113.786**	194.642	159.967	178.421	178.750
Number of Transitions Recorded	12478	13121	7437	2376	1574	978

Table 5.10: Diver Acquisition Rates of the 'No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (State Augmentation Only)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	163.266	123.022	149.498	99.963	158.747	**94.324**
Step STD	**10.112**	38.701	15.618	31.627	142.184	71.767
Step Minimum	138.290	87.883	108.404	46.200	24.000	**1.000**
Step Maximum	**190.397**	324.109	202.887	205.571	1487.625	245.000
Number of Transitions Recorded	16841	12225	5234	1187	556	43

Table 5.11: Diver Acquisition Rates of the 'Option Heads No State Augmentation' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (Option Heads No State Augmentation)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	356.808	**97.009**	137.046	133.333	1679.000	NaN
Step STD	181.899	**15.104**	79.874	50.476	4743.272	NaN
Step Minimum	161.935	63.800	4.000	75.000	**2.000**	NaN
Step Maximum	1097.600	**136.294**	541.666	276.000	13418.000	NaN
Number of Transitions Recorded	9194	3754	178	30	8	0

Table 5.12: Diver Acquisition Rates of the 'Option Heads State Augmentation Only' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (Option Heads State Augmentation Only)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	377.228	**108.953**	162.671	4063.667	NaN	NaN
Step STD	92.752	**81.896**	278.663	12959.726	NaN	NaN
Step Minimum	217.012	20.500	**4.000**	**4.000**	NaN	NaN
Step Maximum	670.148	**407.520**	1897.000	45167.500	NaN	NaN
Number of Transitions Recorded	11583	3481	156	17	0	0

Table 5.13: Diver Acquisition Rates of the 'State Augmentation plus No Environment Reward' modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (No Environment Reward)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	183.311	**57.261**	597.807	966.038	167.000	NaN
Step STD	106.941	44.308	1405.747	2260.976	**0.000**	NaN
Step Minimum	88.607	25.483	31.000	**9.000**	167.000	NaN
Step Maximum	629.118	403.425	10782.000	8378.000	**167.000**	NaN
Number of Transitions Recorded	9280	6751	241	33	3	0

Table 5.14: Diver Acquisition Rates of the ‘Diver Locator Reward’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster.

Diver Acquisition Rate (Diver Locator Reward)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	148.891	**63.767**	655.607	200.500	6.111	NaN
Step STD	**21.230**	37.489	3756.409	285.107	0.333	NaN
Step Minimum	110.075	18.595	**1.000**	4.000	6.000	NaN
Step Maximum	259.340	**255.844**	31415.500	761.000	7.000	NaN
Number of Times Reached	12529	5134	205	47	9	0

Table 5.15: Diver Acquisition Rates of the ‘No State Augmentation plus Action Shaping’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. Since there is only one transition between five to six divers, it is considered as a fluke rather than a consistent process.

Diver Acquisition Rate (No State Augmentation Action Shaping)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	325.159	**36.898**	122.515	78.363	118.850	6.000
Step STD	124.623	**16.375**	91.914	62.465	40.777	NaN
Step Minimum	138.574	17.302	**1.000**	3.000	38.000	6.0
Step Maximum	870.736	**123.840**	387.000	205.500	170.000	6.0
Number of Times Reached	14439	4745	117	34	15	1

Table 5.16: Diver Acquisition Rates of the ‘Option Heads with No State Augmentation plus Action Shaping’ modification over a 10×10 average comparing the number of time-steps taken to move from k to $k+1$ divers. Lower values indicate the agent acquiring the subsequent diver faster. Since there are only two transition between two to three divers, it is considered as a fluke rather than a consistent process.

Diver Acquisition Rate (Option Heads NSA + AS)	0 to 1	1 to 2	2 to 3	3 to 4	4 to 5	5 to 6
Step Mean	1434.828	**17.873**	53.000	NaN	NaN	NaN
Step STD	1124.080	**29.533**	9.899	NaN	NaN	NaN
Step Minimum	175.354	**0.500**	46.000	NaN	NaN	NaN
Step Maximum	5926.075	**171.500**	60.000	NaN	NaN	NaN
Number of Times Reached	4734	265	2	0	0	0

Tables 5.2 and 5.5, which cover the ‘Maximum Divers per Episode’ metric. This metric directly captures the number of episodes that the agent managed to reach the six divers needed to achieve the first half of the long-term objective, with Tables 5.1 and 5.4 a close second due to the positive correlation between diver rescue and episode reward.

Based on Tables 5.1, 5.2, and 5.3 it is clear that TIDQN does not show any significant gains compared to the baseline DQN approach, as the baseline DQN approach dominates the mean, minimum and maximum values for all metrics described in Chapter 4. To highlight this disparity, in Table 5.1 the minimum reward (140.180) still exceeds the maximum reward gained by all other approaches (109.930, 36.790, etc.). Furthermore, using Tables 5.4, 5.5, and 5.6 a more granular degradation of performance can be seen even as the additional modifications are used in conjunction with state augmentation.

There is also an interesting negative trend that appears in all metrics, which involves the level of ‘teaching’ that is enforced on the agent via TIDQN. With minimal involvement (i.e., ‘No State Augmentation’) the results, while still worse than the baseline, suffer only minimal degradation. However, as state augmentation and other modifications are added to increase the external influences present, the performance loss increases rapidly. While specific modifications and additions are discussed in their own section, the general negative trend seems symptomatic of a larger underlying issue with the current state of the TIDQN framework.

Looking at the Diver Acquisition Rates on Tables 5.8–5.12, a few observations can be made about the impact of TIDQN on the average number of steps taken to acquire divers. Namely, the implementation of the single-head TIDQN does seem to lead to slightly faster diver acquisition, though this trend is both weak and effectively disappears with stronger enforcement of teaching and use of state augmentation.

However, when combined with the number of transitions recorded, it is clear that the inclusion of any TIDQN elements drastically reduces the number of later transitions (e.g., four to five divers, five to six divers), with some never reaching a single transition. Interestingly, the mean number of steps at later transitions on Tables 5.8, 5.9, and 5.10, along with any modifications combined with these approaches seems to be rather similar to one another. When combined with the drastic drop in transition count, the hypothesis that the baseline DQN performs better due to a stronger balance between defeating enemies and rescuing divers, whereas TIDQN may be overvaluing the latter. This in turn may lead to the scenario observed where TIDQN agents do not have nearly as many higher diver acquisitions.

Alongside the hypothesis of imbalanced priorities, there are several ideas regarding the mixed performance of the TIDQN compared to the baseline. The first major issue is how information was gathered from the environment. The major extractor of information was based on checking the mean value of several 2×2 kernels against a threshold to determine the number of divers an agent has at any given time.

While the process was very simple, its simplicity meant it was vulnerable to unexpected changes in the environment. For example, when an agent receives six divers the divers begin to 'flicker' every two frames. This causes the counter to output values similar to the following: $[5, 0, 6, 6, 0, 0, 6, 6, 6, 0, 0, \dots]$. For TIDQN, this caused many issues in how reward shaping is handled and likely harmed the policy as a result.

Another potential issue is how state augmentation was implemented. The method of augmentation, shown in Figures 4.4 and 3.2, could be ill-suited for the DQN architecture. By augmenting in this fashion, it is possible that the information is not handled properly or simply causes more harm than good, which can be seen by comparing the performance of the 'No State Augmentation' against the 'State Augmentation Only' rows in Tables 5.1, 5.2, and 5.3.

5.4.2 Removal of Environment Reward

In the case of removing the environment reward entirely, it is clear that this simply harms the policy with seemingly little payoff. This can be clearly seen in Figure 5.2, where the modification's poor performance is immediately apparent. The corresponding Tables 5.4 and 5.5, along with Table 5.6, further corroborate this, as the modification ranked worst in nearly all metrics listed.

Additionally, Table 5.13 shows that the diver acquisition rate seems to perform the worst of any single-head approach with egregious variance and maximum values for the transitions it actually reached consistently, though it remains on similar footing to any option head approach.

The most likely explanation for this performance is due to the fact that Seaquest, at its core, only has effectively two means of gaining points. While reward shaping is in place for attaining one of these goals, it is neither a sufficient replacement nor a necessary replacement.

While rescuing divers remains the main objective of Seaquest, it must be balanced against the agent actively keeping their submarine alive. As such, it is likely better to leave the reward values from the environment untouched to help the agent understand this portion of gameplay. However, in environments where there are a plethora of means to gain reward, or where the desired objective is not as long-term, this approach may have a positive effect.

Furthermore, this type of modification could instead be converted into training the agent in subsections of the environment, an approach that is discussed further in Section 6.2.4 as a future direction.

5.4.3 Use of Auxiliary Feature Extractor

The use of an AFE was demonstrated across multiple experiment setups to be a failed modification to TIDQN. This can be seen in Tables 5.4, 5.5, and 5.6, where similar to the ‘No Environment Reward’ modification, its performance paled in comparison to the baseline TIDQN. However, it performed very slightly better than ‘No Environment Reward’, reinforcing the idea that having access to the short-term reward plays some role in reaching long-term goals.

For diver acquisition, Table 5.14 shows evidence that reward shaping was indeed not a good fit for the data generated by the AFE, which is made more apparent when compared to the use of the AFE for action shaping, shown in Table 5.15. While the reward shaping approach does perform better on reaching the first transition, its performance drops quickly thereafter, with the action shaping approach showing generally lower mean and standard deviation.

Interestingly, both approaches reach the upper transitions very infrequently, yet have somewhat similar performance to the baselines. Evidence of this can be seen in the mean steps taken in Table 5.15 are slightly better than the ‘No State Augmentation’ TIDQN shown on Table 5.9 and baseline DQN shown on Table 5.8. While the difference in sample size is still overwhelming, there is still some promise in looking further into this particular approach.

The inclusion of auxiliary feature extractors, namely the diver locator, poses an interesting issue. While its performance was not as poor as the ‘no environment reward’ modification, it paled in comparison to both the state augmented and non-state augmented baselines. While this is unexpected, it is possible this occurred due to the sub-par performance of the feature extractor. In other words, the performance of the feature extractor, described further in Appendix A, may have extended into the performance of the policy that relied upon it.

Ultimately, it may be the case that an auxiliary feature extractor may need to be even higher quality than the resulting policy to avoid doing more harm than good for creating a better policy. While this situation is somewhat avoidable with a single-head approach, it is more difficult to manage with the option heads approach, which makes direct use of the feature extractor to determine which option head to follow.

Although training high-quality models for a given environment is not necessarily a difficult task, it remains somewhat brittle as, even with transfer learning, these extractors likely cannot be easily swapped between environments. Furthermore, it could be cost-prohibitive depending on the dataset necessary to train these extractors. While environments like Seaquest are somewhat simple to acquire data from, more complex applications may render this modification more difficult than its worth.

5.4.4 Option Heads Architecture

The Option Heads approach, which uses the approach described by Arukulmaran et al. [1] ultimately did not perform as well as anticipated. Sitting at the bottom of Tables 5.1, 5.2, and 5.3, the option heads approaches were well below both the baseline DQN and single-head TIDQN in nearly all aspects. To cement the overall performance, Tables 5.16, 5.12, and 5.16 all show similar trends of high variance and maximum steps alongside failure to consistently transition past as few as three divers in many cases. Compared to the single-head TIDQN, which had similar, though inferior performance to the baseline DQN, it is clear that the policy derived via the multi-head approach simply failed at learning the desired policy.

While these results are abysmal compared to the other approaches, these results are not necessarily representative of the work done by Arukulmaran et al. [1]. Since Keras RL currently lacks support for multi-head approaches and the source code for the original implementation of Option Heads was unobtainable, it remains entirely possible that the current implementation of this approach remains flawed. Further investigation of the results provided showed unusual behavior within the option heads, though the source remains unclear.

Therefore, the results are treated more as an ‘initial attempt’ at re-implementing the architecture, compared to the other implementations of TIDQN. Whether these flaws are in spite of or due to modifications to bridge TIDQN and Option Heads is an investigation for future works.

Chapter 6

Conclusions and Future Directions

6.1 Conclusion

Many learning problems, from image recognition to text generation, have found residence within modern society through the power of deep learning. Despite this, reinforcement learning struggles even with the power of deep learning.

This thesis proposed a framework to try and introduce teachability in a way that was still easily understood by a human observer, in order to improve the overall performance of the policy that came forth. However, based on the results outlined in Chapter 5, TIDQN returns mixed results when applied to the Seaquest environment. Essentially, while some TIDQN-based methods did perform better than the baseline DQN at acquiring divers quickly, it seemed to come at the cost of overall performance in the environment.

Despite TIDQN demonstrating a negative correlation between the amount of external information passed to the agent through the channels outlined in Chapter 4 and performance on standard metrics, the need for an approach that facilitates teachability and interpretability remains.

6.2 Future Directions

The proposed approach is ultimately an exploratory look into ways to introduce both teachability and interpretability into the problem of deep reinforcement learning. While the initial results were mixed, there are many modifications and new approaches that can be taken in light of these results.

6.2.1 Action Shaping via Completion Vector

The Completion Vector was primarily used for reward shaping. For certain Task Nodes, like diver rescue, this was adequate. However, for other Task Nodes, such as oxygen replenishment, reward shaping did not perform well for encouraging the agent towards completing the task. This is likely because while certain Condition Nodes are active, the agent should not be allowed to perform other actions outside of what is necessary to finish a given task. Therefore, trying to use the Completion Vector more for action shaping and figuring out if the Completion Vector needs to be adjusted to handle this change would be a good future step.

6.2.2 Testing Alternative Representation of Tasks

The Representation of Tasks that was implemented in this thesis was a directed graph, due to the simple nature of the environment it is tied to. However, it likely would not remain as interpretable given environments of higher complexity, even if an RoT can be constructed.

Therefore, an avenue of interest for furthering this work would be understanding the use cases of other graphical representations of a given environment and investigating whether those representations remain as accessible in terms of understanding an agent's behavior.

To this end, Bayesian networks [8] would be a first step in this direction to determine if more sophisticated representations improve the resulting policy, as it still maintains similar properties to the RoT used in TIDQN.

6.2.3 State Augmentation with the Representation of Tasks

The TIDQN used in this work only augmented the state with the Completion Vector. However, some information cannot be represented with a simple bit vector, such as the dependency relation between different Task Nodes. Thus, investigating the impact that incorporating the full Representation of Tasks would have on model performance would would be worthwhile. While this could be done with the RoT used for Seaquest via an adjacency matrix, more sophisticated models, such as the Bayesian Network approach described earlier, could be integrated directly as a supplementary model to the DQN.

6.2.4 Using Configurable Environments

While many environments, like the Atari Learning Environment, don't actually contain sub-environments that focus on only one part of a task, it would be very interesting to see if training an option heads structure on this type of environment would have more pronounced results as this guarantees that certain states can be reached with reasonable probability.

Furthermore, there is currently an inability to capture more nuanced states without introducing more auxiliary feature extractors. An example of this in Seaquest is attempting to capture the states where an agent successfully resurfaces with six divers. Since the only indications of successfully completing this task involve otherwise negative indicators (e.g., oxygen meter going to zero, divers being decremented one at a time), it becomes difficult to reconcile with the Representation of Tasks. By finding a more flexible environment it may be feasible to fully utilize the RoT without issue.

6.2.5 Utilizing Bayesian Networks as Supervisory Network

Currently, the option heads architecture does not use any supervisory network, but uses the completion vector directly to determine which head should be used. However, this is somewhat limiting and may hinder performance in certain environments. However, using a Bayesian network [8] that can incorporate not only the completion vector but other state information and feature extractors would not only allow for more flexibility in selecting which option head to use, but retain some degree of interpretability by humans compared to a standard deep neural network.

6.2.6 Improving Auxiliary Feature Extractor Performance

The current hypothesis is that the performance of TIDQN is impacted by the quality of any auxiliary feature extractors such that poor performance propagates to the policy learned by the agent. Therefore, it is important to verify whether this hypothesis is correct by constructing a feature extractor that performs significantly better than the one used by the proposed approach.

6.2.7 Inclusion of Constraints

The field of RL Safety prioritizes policies that adhere to certain safety requirements placed upon them over raw performance. Techniques that fall into this area include the use of hard constraints, or rules that terminate an episode if broken, such as the work by Quint et al. [20].

As a future direction, it would be interesting to see if including the use of hard and soft constraints as new additions to the Representation of Tasks would result in better policies that also have properties described in Chapter 4.

6.2.8 Testing Alternatives to Including State Augmentation

As discussed in Chapter 4, state augmentation is a basic component of the TIDQN framework. Furthermore, the seemingly detrimental impact of state augmentation that was performed by TIDQN was discussed in Chapter 6, where it was hypothesized that concatenation of the state information was ill-suited for the DQN architecture. Therefore, testing alternative methods, such as directly feeding the raw state augmentation to the output layer of the DQN (i.e., removing concatenation) or performing additional processing by feeding through a sub-network tied to the DQN architecture would be interesting future directions.

Appendix A

Investigation of the Performance of the Auxiliary Feature Extractor

A.1 Problem Outline

As mentioned in Chapter 4, there were multiple approaches to the Auxiliary Feature Extractor (AFE), with each approach having its own challenges. Ultimately, a multiclass, multilabel approach was selected for use, despite having less than ideal test performance. This appendix seeks to investigate the overall performance of the network selected and understand more concretely where the potential performance issues are. Furthermore, this appendix seeks to give more evidence regarding the impact a poor-quality AFE has on a policy that utilizes it.

A.2 Description of Metrics

A.2.1 Subset Accuracy

For multi-label classification problems, subset accuracy is used to determine the overall performance of a given model. However, it is also the strictest measure, as predictions are only treated as correct if and only if said prediction matches the true label exactly, described in Equation A.1. To this end, accuracy could be seen as a lower bound of overall model performance, as it does not capture instances where the model correctly predicted at least one of the labels. In the context of the AFE, it is treated as the base ‘accuracy’ of the model and compared against the Hamming Loss. Note that the equation calculates the accuracy across the set of predicted values and corresponding true labels.

$$A_{subset} = \Sigma_{i=0}^{n} y_{i,true} \wedge y_{i,pred} \tag{A.1}$$

A.2.2 Hamming Loss

Since subset accuracy is calculated as whether the prediction exactly matches the ground truth, it may not fully capture the performance of the model. Hamming Loss, calculated using Equation A.2 where n refers to the number of classes present in the multi-label problem and $\oplus$ is the logical XOR operation, is different as the loss is based on individual differences or, in other words, it has lenience with respect to partially correct predictions. Note that this equation only accounts for one prediction versus one true label.

$$L_{hamming} = \frac{1}{n} \Sigma_{i=0}^{n} y_{i,true} \oplus y_{i,pred} \tag{A.2}$$

A.2.3 F1 Score

The F1 score, or F-measure, is a metric that combines the precision () and recall (i.e., the ability to detect positive instances) of a given binary classifier. In the case of multi-label classification, the classifier can be treated instead as n binary classifiers, where n refers to the number of total labels.

A.3 Testing Method

To determine the average values described, 10-fold cross validation was used. For each fold, a new model was trained and a multi-label confusion matrix, or a list of $2x2$ confusion matrices for each label possible, for each model was generated on the test set for that fold. These confusion matrices were then aggregated before the metrics were calculated, also known as the micro-average strategy.

However, for the Hamming Loss metric, which is calculated for each fold, the macro-average method, which averages the metrics themselves, was used. Therefore, for Hamming Loss the mean, standard deviation, minimum and maximum values are recorded instead.

A.4 Results and Discussion

Tables A.1 and A.2 contains the aforementioned metrics generated by running the 10-fold cross validation.

Table A.1: Table describing the micro-average metrics for each label used in the multi-label diver locator. Values in bold indicate the label which had the highest value for that particular metric.

	True Positive Rate	True Negative Rate	F1 Score
No Diver	**0.9283**	**0.7890**	**0.9473**
Up Left	0.8939	0.7681	0.8671
Down Left	0.8603	0.7175	0.8634
Up Right	0.8965	0.7809	0.8741
Down Right	0.8658	0.7354	0.8711

According to the results presented, it can be determined that, while the accuracy of each fold is somewhat low, the Hamming Loss is also low, both in value and variance. This indicates that there are several instances where the model has a partially correct prediction which, as mentioned earlier, is not reflected in the accuracy metric. For

Table A.2: Table describing the macro-average metrics for each label used in the multi-label diver locator. Values were generated from 10-fold cross validation of the model.

	Mean	STD	Minimum	Maximum
Hamming Loss	0.1570	0.0126	0.1432	0.1816
Subset Accuracy	0.5687	0.0284	0.5240	0.6160

the policy, this is likely adequate as partial correctness is still sufficient for guiding the agent towards desired goals, even if action shaping that is contingent on the AFE is used.

Table A.1 shows that each label has a high True Positive Rate (TPR), indicating that the model is capable of correctly identifying samples that contain divers in the expected quadrant. However, this conclusion comes under question upon looking at the True Negative Rate (TNR) of all labels. Given the large gap between the TPR and TNR, it can be inferred that the model may have simply learned to predict positive for any label more often than not, which would point towards a potential dataset imbalance.

To support this hypothesis of dataset imbalance, the ‘No Diver’ label seems to perform the best according to Table A.1. This could be due to the fact that it is the only label that is mutually exclusive to all other labels present, meaning it would have plenty of instances that the label is not true. This is more apparent in the scenario where action shaping is utilized, as a more conservative feature extractor might prevent the agent from trying to move towards divers that do not actually exist.

A.5 Conclusion

Based on these results, there is reason to believe that there are insufficient negative instances for a given quadrant, a problem that could normally be solved by training separate binary classifiers in a 'One v. Rest' method. However, since the problem is also multi-label, such an approach would not work without modifications. Therefore, methods that operate on a per-label basis or apply label-based weighting might help resolve this possible imbalance.

As for determining if the quality of the AFE used by the model is impacting the quality of the policy, it remains in the air. While there are potential dataset imbalances that are causing the model to behave more optimistically than it should, the fact that it still performs well enough at detecting the presence of divers leaves the conclusion unclear. It could be that, due to the implementation of action shaping and fail-safe of assuming 'no divers' if the corresponding bit is flipped regardless of the other bits may mitigate any issues caused by false positives being raised.

Appendix B

Seeds Used for Training and Testing

Table B.1: Train and Test values used in the 10×10 comparisons.

Train Seeds	Test Seeds
94202	94106
30978	93247
34731	87396
57962	85452
61252	66531
65417	54820
76019	46725
86705	19825
87903	16844
9989	15069

Bibliography

[1] K. Arulkumaran, N. Dilokthanakul, M. Shanahan, and A. A. Bharath. Classifying options for deep reinforcement learning, 2017.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents, Jun 2013.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[4] N. Bougie and R. Ichise. Deep reinforcement learning boosted by external knowledge. In H. M. Haddad, R. L. Wainwright, and R. Chbeir, editors, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 331–338. ACM, 2018.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *CoRR*, abs/1606.01540, 2016.

[6] S. Cartwright. Seaquest. Atari 2600, 1983.

[7] J. P. Chris Metzen. StarCraft. PC, 1998.

[8] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[9] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 271–278. Morgan Kaufmann, 1992.

[10] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13:227–303, 2000.

[11] IceFrog. Defense of the ancients. Steam, 2013.

[12] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[13] T. M. Mitchell. *Machine learning, International Edition.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[16] R. Niel and M. A. Wiering. Hierarchical reinforcement learning for playing a dynamic dungeon crawler game. In *IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India, November 18-21, 2018*, pages 1159–1166, 2018.

[17] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep exploration via bootstrapped DQN. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4026–4034, 2016.

[18] R. Parr and S. J. Russell. Reinforcement learning with hierarchies of machines. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 1043–1049. The MIT Press, 1997.

[19] M. Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016.

[20] E. Quint, D. Xu, H. Dogan, Z. Hakguder, S. Scott, and M. B. Dwyer. Formal language constraints for Markov decision processes. In *NeurIPS 2019 Workshop on Safety and Robustness in Decision Making*, 2019.

[21] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[22] Ö. Simsek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In L. D. Raedt and S. Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 816–823. ACM, 2005.

[23] U. Software. Montezuma's revenge. Atari 2600, 1984.

[24] R. S. Sutton and A. G. Barto. *Reinforcement learning—an introduction.* Adaptive computation and machine learning. MIT Press, 1998.

[25] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[26] H. van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang. Hybrid reward architecture for reinforcement learning. *arXiv:1706.04208 [cs]*, Nov 2017. arXiv: 1706.04208.

[27] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, and K. Kavukcuoglu. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3486–3494, 2016.

[28] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3540–3549. PMLR, 2017.

[29] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3540–3549, 2017.

[30] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

[31] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.