Learning Complex Behaviors from Simple Ones:

An analysis of Behavior-based Modular Design for RL Agents

by

Kevin Jatin Vora

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2021 by the
Graduate Supervisory Committee:

Yu Zhang, Chair
Yezhou Yang
Sarbeswar Praharaj

ARIZONA STATE UNIVERSITY

August 2021

ABSTRACT

Traditional Reinforcement Learning (RL) assumes to learn policies with respect to reward available from the environment but sometimes learning in a complex domain requires wisdom which comes from a wide range of experience. In behavior based robotics, it is observed that a complex behavior can be described by a combination of simpler behaviors. It is tempting to apply similar idea such that simpler behaviors can be combined in a meaningful way to tailor the complex combination. Such an approach would enable faster learning and modular design of behaviors. Complex behaviors can be combined with other behaviors to create even more advanced behaviors resulting in a rich set of possibilities. Similar to RL, combined behavior can keep evolving by interacting with the environment. The requirement of this method is to specify a reasonable set of simple behaviors.

In this research, I present an algorithm that aims at combining behavior such that the resulting behavior has characteristics of each individual behavior. This approach has been inspired by behavior based robotics, such as the subsumption architecture and motor schema-based design. The combination algorithm outputs n weights to combine behaviors linearly. The weights are state dependent and change dynamically at every step in an episode. This idea is tested on discrete and continuous environments like OpenAI's "Lunar Lander" and "Biped Walker". Results are compared with related domains like Multi-objective RL, Hierarchical RL, Transfer learning and basic RL. It is observed that combination of behaviors is novel way of learning which helps the agent achieve required characteristics. A combination is learned for a given state and so agent is able to learn faster in an efficient manner compared to other similar approaches. Agent beautifully demonstrates characteristics of multiple behaviors which helps the agent to learn and adapt to the environment. Future directions are also suggested as possible extensions to this research.

DEDICATION

This thesis is dedicated to my family, friends and teachers for their constant support, motivation, encouragement and helping me evolve by learning

ACKNOWLEDGEMENTS

First and foremost I am deeply grateful to Dr. Yu "Tony" Zhang, for his assistance at every stage of the research project. He helped me grow by learning about research and motivated me with the impact it could have. He pushed me to achieve more and I cannot describe his efforts throughout this journey in a few words. I would also like to thank my committee members Dr. Yezhou Yang and Dr. Srabeswar Praharaj for their invaluable feedback, time and support.

And most importantly, I would like to thank my girlfriend for her help and support during the times I felt lost. Finally, I would also like to thank my parents, grandparents, uncle, aunt and my brothers and sisters for their constant love, support, motivation and faith in me. Without them this journey wouldn't have been possible.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

Chapter 1

INTRODUCTION AND RELATED WORK

1.1 Introduction

Intelligent agents operating in the real-world are subject to various uncertainties that are difficult to fully specify at the design time. As a result, we often need these agents to learn from interacting with the environment. Significant contributions to Reinforcement Learning literature(RL) includes Q learning (Watkins (1989)), SARSA, etc. These have contributed much to the research in the domain of learning theory in Artificial Intelligence(AI) (Sutton and Barto (2018)).

Most of the RL algorithms are sample inefficient and fail to learn the complex task. In real-world tasks, a physical robot must learn to adapt to a complex model of its dynamics coupled with the environment. For example, for a biped humanoid robot to learn, it must not only learn to control various joints but also model complex interactions with the environment. As a result, even the best RL algorithms fail to efficiently address this problem (Yu (2018)). However, a behavior in complex domain may be considered as combination of multiple simple behaviors. A desired behavior can be defined as the way in which an agent acts in response to a particular situation or stimulus. For example, if the task is to travel from point A to point B, the agent would have to walk towards the goal and avoid obstacles based on the current environment setting. Even though it may be easy to specify individual rewards for each of these behaviors separately, one of the challenges is that it is difficult to combine these rewards in a meaningful way since RL takes a single reward. Furthermore, even when such a combination is provided, using a single reward may

be less informative and even misleading for learning (Watanabe and Sawa (2010)). For example, if an agent walks close by an obstacle and is unable to balance and falls down; it is unclear whether the negative reward should be attributed to walk or obstacle avoidance behavior. In addition, using a single reward doesn't facilitate the interpretability for the agent's behavior which is often needed for agents operating with humans in the loop (Graves and Czarnecki (2000), Zhang *et al.* (2017)). An architecture that learns and combines behaviors is desirable for interpretability since it naturally supports reward attributability to behaviors.

Our work is inspired by behavior based robot learning. Subsumption Architecture is a behavior based robotics architecture. It is proposed in contrast to traditional AI. Subsumption is a bottom-up architecture that takes sensory information as input to generate potential action in the environment. This is achieved by decomposing behavior into sub-behaviors organized in a hierarchical structure. Each layer implements a particular level of behavioral proficiency, and higher levels are able to subsume lower levels in order to create the desired behavior. An example of subsumption architecture is the Roomba vacuum cleaner robot. Here, the agent has 3 behaviors (High level to low level):

1. Clean
2. Avoid obstacles
3. Wander around

Let's say that the agent is wandering around. When the agent detects an obstacle, it activates the avoiding obstacle behavior which then inhibits the wandering around behavior. Similarly, when the agent detects dirt, the clean behavior is activated, which then inhibits both the lower level behaviors.

While subsumption architecture is interesting and inspirational, it is often activating a simple behavior at any point of time and it is difficult to achieve situations where desired behavior must share characteristics of multiple simple behaviors at the same time. We propose to adopt a schema based approach that combines behaviors. This is called the schema based robotics approach as described by Balch and Arkin (1998), Zhang and Parker (2013), Zhang and Parker (2010), Weitzenfeld *et al.* (1998) and Graves and Czarnecki (2000). The problem of combining behavior is difficult because at every step agent requires different characteristics borrowed from different behaviors. When an obstacle is only partially blocking the path, we would like the agent to figure out a way to dodge the obstacle while moving forward. In such cases, we would like the agent to demonstrate characteristics from multiple behaviors to be more efficient. Since the world around the agent is complex that requires interactions between individual behaviors, a learning strategy is required which can efficiently combine behaviors given the state of the agent. During training, agent needs to experiment with different combinations in different states and learn to generalize which combination is best suited in any state. We consider this as a RL problem where the agent's goal is to learn to combine in order to accomplish the task when simple trained behaviors are available.

We find that combination of behaviors is a modular and efficient approach. Compared to several baseline methods, our agent performs better. We have assigned weight parameters to all the behaviors the agent has to learn. Eventually, the agent learns to assign the appropriate values to the weights and acts such that desired behavior is achieved. We also observe the ratio of weights which changes depending on the situation and that helps us interpret how the agent thinks. In addition to this, behavior learned in one environment is modular since weights are frozen for individual behavior (Lu *et al.* (2021)) and can be easily transferred to another similar

environment. Moreover, We observe that combining behaviors learned in different environments demonstrate efficient learning of characteristics of multiple behaviors.

One of the limitations of our approach is the assumption of prior knowledge about which simple behavior may benefit the new task. When conflicting behaviors are included, the learning system performed less desired. For example, if the agent is able to learn walking forward behavior which is supposed to be combined with walking backward, the combination fails to achieve characteristics of both in a resultant behavior.

As we proceed further, Related work puts together closely related areas which is followed by a methodology section that describes the central idea of the thesis. Implementation details, results, and analysis are elaborated in the next chapter (Experiments, Results and Analysis) which is followed by Future work and Conclusion.

1.2 Related Work

Approaches that take advantage of prior knowledge for RL agents have been experimented in various AI settings. There has been a lot of contribution in the domain of Hierarchical Reinforcement Learning, Multi-objective Reinforcement learning, ensemble methods, multi-task learning, etc. Existing literature looks a lot like the core idea we present but there are significant areas where we differ. We borrow ideas from a wide range of literature but the end goal is to create a meaningful learning algorithm that combines multiple behaviors using Deep Reinforcement Learning.

1.2.1 Hierarchical Reinforcement Learning

It is necessary to look at modular approaches like Hierarchical RL (HRL) which include Barto and Mahadevan (2003), Xiaoqin *et al.* (2009), Cai *et al.* (2013), Doroodgar and Nejat (2010) and many others. HRL is a process of learning in a hierarchy such

that parents can invoke children as its action and the lowest level in the hierarchy is responsible for taking actions for the agent in the environment. Hierarchical structure makes it possible to divide the learning problem into smaller problems. These smaller problems traverse down the hierarchy such that the lowest level is responsible for taking actions that lead to successfully solving the child problem. Learning to call child problem and solving simpler problems one by one enables the agent to maximize its rewards and reach the goal. For example, the bug algorithm in path planning whose goal is to travel from point A to point B and avoid obstacles. This problem can be divided into reach the goal and follow the boundary of obstacles subtasks. Initially, the agent starts moving towards the goal and as soon as it encounters an obstacle, it revolves around it once and when it figures out the closest point to the goal, it again follows the boundary till it reaches the closest point identified and then starts moving towards the goal again. So the root can call moving towards the goal subtask and switch to boundary following subtask when an obstacle is encountered. In order to achieve each subtask, a different subtask is called which takes care of necessary actuation. As a result, using multiple subtasks agent is able to avoid obstacles and reach the goal. Using such an approach has the following advantages:

- Sample Efficiency: As the complexity of the task increases, it takes more and more time and data to learn. With HRL we can achieve better sample efficiency by using prior knowledge of behavior hierarchy.

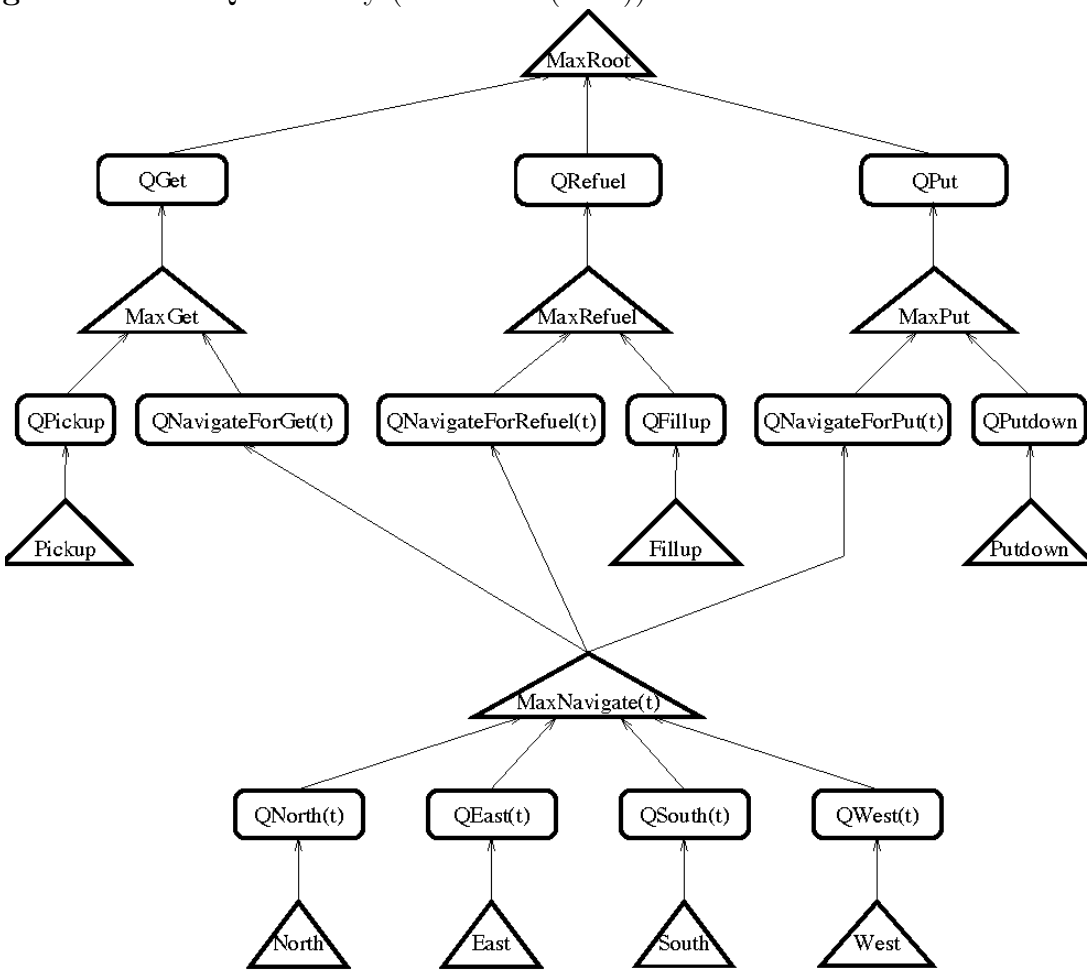
- Exploration: HRL presents an optimized way of exploration since the larger subspace has been reduced to a few sub-tasks in the hierarchy.

- Reusability: As lower-level policies can be reused in different situations.

Next, we discuss the most relevant HRL work to ours:

The MAXQ decomposition (Dietterich (1998)) is a combination of procedural semantics (subroutine hierarchy) and declarative semantics (a representation of the value function of a hierarchical policy). Authors motivate their Hierarchical Q learning algorithm by using a gridworld "Taxi" domain to showcase the results. As we see in the figure 1.1, the root node calls a simpler task at a time. Now, the goal of the subtree is to achieve the subtask and return the control back to the root note to select another subtask. In order to achieve the subtask, at the lowest level agent takes an action. As the agent acts, step by step subtasks are completed which help the agent to achieve the end goal.

Figure 1.1: MaxQ Hierarchy (Dietterich (1998))



For example, if the task is to pick up a passenger from location A and drop off at location B, the learning strategy would call QGet which makes the call to QNavigate in order to pick up and then control returns back to the root node where Qput is called in order to drop off the passenger after necessary navigation steps. As a result navigation sub-problem can be reused and Qget and Qput use it to formulate their policies which helps the agent in the successful completion of the goal. This technique improves exploration and it quickly learns from fewer trials.

In contrast to this method, we combine behaviors instead of just creating disjoint behaviors in a hierarchy. One behavior can overlap with another and at a given time step, we expect desired combination by learning weighted hierarchies. Furthermore, generalizing to continuous domain tasks is also difficult with this approach. In addition to this, FeUdal Networks is also an well known HRL technique.

FeUdal Networks (Vezhnevets *et al.* (2017)) design a manager-worker architecture, where the Manager operates at a lower temporal resolution and sets abstract goals which are assigned to and enacted by the Worker. In this way, the manager is at a higher level of hierarchy and the worker is at the lower level. This framework has the following 2 modules:

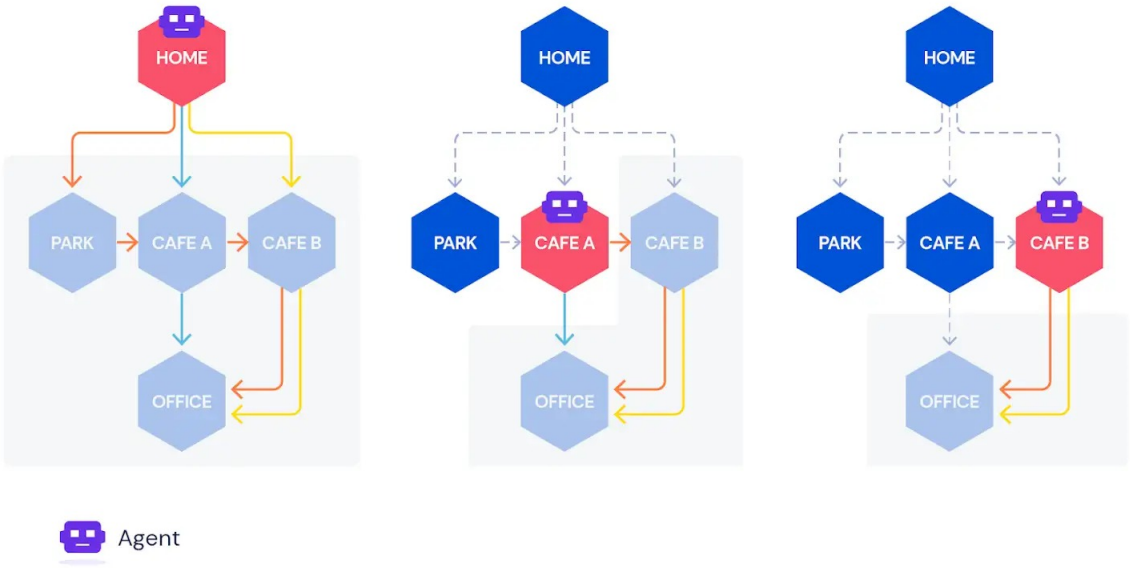
- Manager: For a given input state representation, the manager computes a latent state representation s_t and uses s_t to produce g_t (goal vector).
- Worker: Worker produces an output action based on state, goal vector, and external observation.

Whereas we know behaviors we generate and we have the ability to learn necessary combinations to earn rewards from the environment. The FeUdal network generates subgoals using a manager but doesn't exactly know what subgoals it may generate. Also, at a time worker aims to achieve a particular goal instead of a combination

of goals; for example, achieving rewards for multiple behavioral goals (walking and avoiding obstacles).

Fast reinforcement learning with generalized policy updates is another related technique that deals with combining policies based on preferences.

Figure 1.2: Fast Reinforcement Learning With Generalized Policy Updates (Barreto *et al.* (2020))



Fast reinforcement learning with generalized policy updates by Barreto *et al.* (2020) aims at reusing already learned policies. The authors describe 2 processes: Generalised Policy Evaluation (GPE) and Generalised policy improvement (GPI) in this paper. As shown in the figure 1.2, the blue, orange, and yellow arrows denote different policies learned by GPE. Once the agent has a number of policies, GPI stitches together a new policy using existing policies such that at every state agent's preference is taken into account in order to select which policy best works for the agent. For example, the agent is initially at home and it gives higher priority to the commute time to the office than coffee or food. Therefore, it leaves for cafe A first, and the policy in blue is selected. While having coffee at cafe A, the agent's meeting

is canceled and it can spend some time on getting good food before going to the office. As these preference changes, the agent follows policy in orange instead of blue. In this way policies are weighted by preferences and number of policies are reused such that it can benefit the agent. This work looks like a special case of HRL option framework (Randlov (1999)) as it uses preferences to tailor a new policy from existing policies.

On the other hand, we deal with combining behaviors. At a time instead of just following one policy, multiple behaviors are combined to achieve the required characteristics. During the process of combination, weights are learned to achieve the desired behavior.

In addition to techniques described earlier, there many different strategies to achieve HRL which includes option framework Randlov (1999), Option-Critic Architecture Bacon *et al.* (2017), etc. Since most HRL approaches deal with dividing a bigger problem into smaller problems arranged in a hierarchy and using it to achieve the task, the notion of combination is not observed. As a result, we also look at Multi-Objective Reinforcement Learning which has a combination component for incorporating multiple objectives.

1.2.2 Multi-Objective Reinforcement Learning

Multi-Objective Reinforcement Learning (MORL) is an interesting area of research as described in Liu *et al.* (2014), Sprague and Ballard (2003), Vamplew *et al.* (2011) and many others. MORL is a branch of RL which deals with learning trade-off between multiple objectives in order to achieve the goal. For example, if an agent has an objective to balance between speed and energy consumption, we have conflicting situations since higher speed needs more fuel. MORL is able to learn a Pareto frontier such that it optimizes the trade-off based on experience.

A Survey of Multi-Objective Sequential Decision-Making (Roijers *et al.* (2013))

summarizes approaches to multiobjective reinforcement learning (MORL). Since there can be conflicting objectives for an agent in decision-theoretic planning and learning, MORL seems to be a challenging problem. Some problems can be converted into a single objective and solved while others require specific methods in MORL. The paper presents a comparative study of different MORL algorithms and approaches.

Traditional ways to combine results from multiple objectives in RL revolves around the idea of Linear Scalarization (Van Moffaert *et al.* (2013)). In the simplest version of this technique, the domain expert decides the weight for each objective, and weights and objectives are formulated as a linear combination. There are some limitations (Vamplew *et al.* (2008)) of this technique and researchers try to overcome them using the Chebyshev function and a few other techniques. The key difference between this approach and our combination learning lies in the state information which enables us to change the weight with respect to the current state and adapt to the situation-oriented behavior. In addition to this, our work also relates to meta-learning.

1.2.3 Meta-Learning

Meta-Learning research as described in Hospedales *et al.* (2020), Vilalta and Drissi (2002), Vanschoren (2019), Schweighofer and Doya (2003), and many others are also a closely related domain which deals with the process of learning to learn an algorithm. Meta-learning can be used in single-task and multi-task learning (discussed later in this section). In multi-task scenarios, task agnostic knowledge is used for improving the learning of new task which belongs to the same family of tasks. While in single-task scenarios, only one task is solved repeatedly in order to improve performance over multiple episodes. Meta-learning can also be used to learn a set of hyperparameters like learning rate, regularization factor, etc which can contribute to making the network using these parameters stronger, leading to better performance.

While meta-learning is somewhat related to the combination strategy, what we propose also deals with learning about a combination of learning RL behaviors but this is done using deep RL, and behaviors are known beforehand so that it would combine into expected behavior. This can also be recognized as a special case of meta-learning.

Moreover, we would also like the combination to adapt to new environments even if the behaviors are trained under different environments. This relates to the domain of transfer learning.

1.2.4 Transfer Learning

Transfer Learning is the process of learning a target task by using external experience from other tasks. For example, a coffee maker agent can learn to make coffee and then reuse its skill to make a cappuccino. Transfer learning enables reusability. Still, it doesn't overcome the traditional way of learning. Usually, transfer learning deals with learning about a source domain and apply learned knowledge to a target domain which helps the algorithm reuse the knowledge and perform well in the target domain with some fine-tuning. Domain adaptation is a similar approach to transfer learning with a difference in the source distribution and target distribution. The goal of domain adaptation is to overcome this variation such that source knowledge becomes useful in the target domain.

Using Transfer learning in RL can be useful for training Game Playing agent. AlphaGo (Silver *et al.* (2016)) uses this technique to learn the chessboard game. This method is also applied in the Natural Language Processing domain. Expert demonstrations can be applied to model RL solutions for Spoken Dialogue Systems (Andreas *et al.* (2016)), building shaped rewards for Sequence Generation (Bahdanau *et al.* (2016)),transferring policies for Structured Prediction (Chang *et al.* (2015)).

The behaviors can be combined in such a way that they can become useful in a different domain as well. The key difference between transfer learning and combination approach is that transfer learning tries to gain experience and adapts to the requirement of the environment, while the combination is formulated to work for a new environment by learning to combine behaviors as required which can be achieved because learning is designed with prior knowledge of target domain. Most importantly, behaviors are known and combined in such a way that it helps to achieve adaptation. So combination based approach is a subset of transfer learning approach where we have prior knowledge about the target domain and so we know required behaviors and combine them in a way such that characteristics from multiple behaviors helps the agent in the target domain.

After transfer learning, it becomes necessary to explore technique which aims at solving related tasks simultaneously, like multi-task deep RL.

1.2.5 Multi-task Deep Reinforcement Learning

Multi-Task Deep Reinforcement Learning (Vithayathil Varghese and Mahmoud (2020)) deals with learning related tasks simultaneously with the help of deep RL. At regular intervals, individual agents learning a particular "related task" share (D'Eramo *et al.* (2019)) their weights with the global network and the global network makes it possible to share the learned parameters with other individual agents. The main goal of multi-task learning is to learn generalized tasks so that the knowledge can be used to perform closely related tasks. The set of related tasks is learned simultaneously by the agent during the training process. In the end, the overall performance of the agent is improved and the knowledge gained during the process can be used to perform other tasks as well. Authors claim that transfer learning can only succeed when the task is similar while different related tasks need multitask learning.

Multitask learning can also be achieved by learning shared representation of value function, Progressive Neural Networks (overcomes catastrophic forgetting), etc.

So, multitask learning deals with a combination of parameters to share knowledge and learn related tasks while we apply RL to achieve a combination of behaviors. For our combination approach, each behavior is trained from a unique reward and then combined to achieve the goal such that the agent learns the characteristics of each behavior. In our approach behaviors are frozen and the trained models are used for combination, so behaviors are always known while in multitask RL, related tasks are somewhat lost after learning.

Lastly, we consider ensemble methods in RL which deal with a different ideology of combination.

1.2.6 Ensemble methods in Reinforcement Learning

Ensemble Algorithms in Reinforcement Learning (Wiering and Van Hasselt (2008)) is an area inspired by supervised learning in machine learning where it is believed that instead of one model deciding the label if multiple models can give their input to decide the label it would be more informative. Ensemble methods in RL describe ensemble methods like majority voting (MV), rank voting, Boltzmann multiplication (BM), and Boltzmann addition. The goal of ensemble methods in RL is to learn value function or policy which are of different nature and ensemble of such techniques can help the agent learn experience from a wide variety of algorithms. The way ensemble methods learn includes the probability of selecting an action and rewarding every algorithm for the selected action since that's the only possible way for updating and learning.

These methods bring us a lot closer to our approach since there are multiple learning strategies and agent learns to collect better rewards. But the key difference

is that agent is not trying to learn a modified behavior. All learning algorithms in ensemble work towards the same reward and by means of ensemble they come together to still achieve that reward. Ensemble learning doesn't promote modular learning whereas we propose a technique to learn different behaviors and combine them so when the agent uses the combination in the environment it can leverage characteristics of multiple behaviors in an environment with unseen rewards.

Chapter 2

METHODOLOGY

2.1 Motivation

Combining the power of HRL and MORL, we propose a framework that has different behaviors in the hierarchy and a parent learning to merge these different behaviors into a meaningful combination. The combination can act as an update that adds features to the existing learning strategy. For example, if we have an obstacle avoiding agent, we would like to have additional features like cleaning (identify dirt and clean it) added to obstacle avoidance so that we get a vacuum cleaner agent. Most interesting idea is that there are many related behaviors but we do not find a good way of combining and reusing these learned experiences efficiently. A good example involves a self-driving car where we need a combination of simple learning experiences like driving in a lane, avoiding obstacles, following signs and signals, adapting to traffic, etc. It would be a lot easier to have an agent learn this individual behavior and have a combination of these for our driver agent than learning a complex end-to-end network where we do not reuse experience and the agent is not aware of which sub-behavior led to the error.

Our goal is to develop a combination learning algorithm that learns simpler behaviors using RL and combines them in a meaningful way to maximize the rewards from the environment. We demonstrate that this is an efficient approach where the agent learns to combine behaviors and converge faster than individually learning an end-to-end framework or transferring learned knowledge for learning additional features. We can also detect points of error from the combination function which would

enable us to re-train or correct the faulty behavior. We demonstrated successfully combined behaviors and efficiency in OpenAI gym's "Biped Walker" and "Lunar Lander" environments.

Prior work in HRL, multitask learning, transfer learning, etc have a goal which is not to combine behaviors but to maximize rewards in end-to-end learning by dividing into a smaller task or reusing previous experience. Multi-objective RL doesn't distinctly weigh an objective based on a state, it proves to be a good area to extend the technique in order to achieve combination. So in order to achieve characteristics of multiple behaviors and efficiently use behaviors as modules, C-DDPG architecture is designed.

2.2 Background

"Subsumption architecture" (Brooks (1986)) and "Motor Schema Based Navigation for a Mobile Robot" (Arkin (1987)) have been overshadowed by modern day RL, and deep learning techniques. These play a major role in behavior based learning strategies. So, we take inspiration from these techniques and look forward to learning behaviors by means of combination using deep learning frameworks that enable us to generalize well during the process of learning combinations.

Schema based navigation for mobile robot divide the behaviors for navigation like stay on path, avoid obstacles, find intersection and landmarks, etc. This methodology also presents a communication mechanism for interaction between schemas to carry out the necessary behavior. For example, if the robot detects an obstacle, it needs to showcase how certain the detection is and whether it is a moving obstacle or a static obstacle. Depending on this, lower level actions adapt to the situation to overcome the obstacle by controlling the velocity. Formulation of divide and conquer strategy and implementing mobile navigation with schema based approach has been

an inspirational approach. Extending such an idea with modern day RL and deep learning framework builds the foundation of our approach to combine behaviors.

Subsumption architecture is a milestone in behavior based architecture for robotics. It is a reactive robotic architecture which has been widely influential in Real-time AI which is guided by symbolic representations of the environment to take actions by learning a bottom up strategy. In this approach end to end task tasks are divided into sub-behaviors arranged in hierarchical structure. For example, a vacuum cleaner robot's end to end task of cleaning is divided into wander, avoid obstacles and clean. Each behavior is learnt in a hierarchical fashion, such that upper level behavior utilizes the lower level simpler behavior to accomplish the task. This strategy takes raw sensory information and computes output for each behavior such that higher level subsumes lower level behaviors. In terms of RL, subsumption architecture is model free as it only relies on sensory input to take the action and gradually builds its experience of the world. One of the advantages of this approach is that behaviors can function independently making this approach modular. Since only one behavior is executed at a time we think of an idea to combine behaviors independently using the Deep RL framework.

With these existing techniques for behavior learning and modern Deep RL we aim to develop a combination-DDPG framework that can combine related behaviors which share the same action space. We learn how schemas can be useful and how subsumption aims to combine behaviors. As a result, we aim at learning the desired behavior combination by linear combination such that it generates resultant actions from multiple behaviors which helps the agent achieve the required behavior. This architecture to combine behaviors is able to leverage the power of deep RL and behavior based RL.

As we go further, the environment, algorithms and deep learning framework is

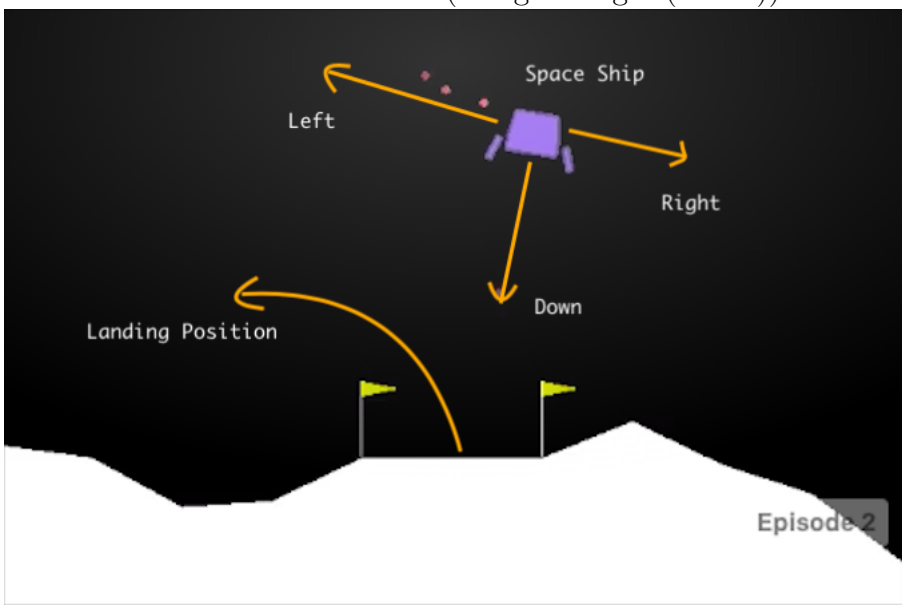
covered. These are put together as architecture to combine behaviors is introduced.

2.2.1 Environment

- Lunar Lander:

  This environment is available from OpenAI gym environment which is an open source library for experimentation in Artificial Intelligence. As Shown in the figure 2.1, the space ship is our agent with 3 engines that enable the agent to take 4 actions (left, right, main, no action) and channel its direction towards the landing position. States of an agent include horizontal and vertical position from center, horizontal and vertical speed, head angle, angular speed and 2 boolean indicating if left and right leg have touched the surface.

Figure 2.1: Lunar Lander Environment (Google images (2021b))

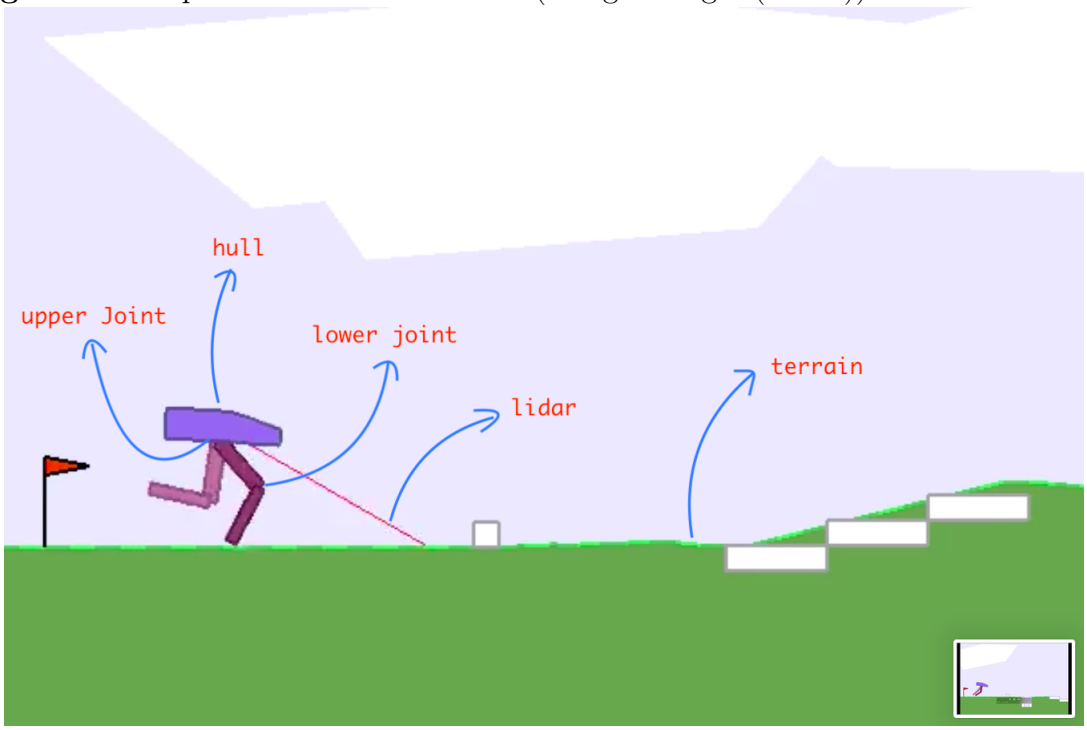


- Biped Walker:

  This environment is also available from the OpenAI gym. As shown in the figure 2.2, our agent is a Biped walker with upper and lower joint angles enabling the hull of the agent to walk. There are 2 variants of this environment: one with

obstacles on terrain as shown in the figure and another is a simple grass terrain with minor bumps. It is easier to learn in the later while very difficult to learn with obstacles. The action space consists of 4 values corresponding to joint angles and the state space consists of hull angle speed, angular velocity, horizontal speed, vertical speed, the position of joints and joints angular speed, legs contact with the ground, and lidar.

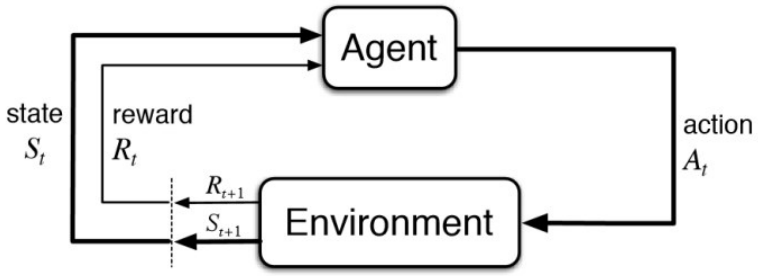
Figure 2.2: Biped Walker Environment (Google images (2021a))



2.2.2 Q learning

In Reinforcement learning, Q learning is a common algorithm that enables the agent to learn from experience. The basic structure of the simple RL scenario is shown in the figure 2.3. An agent exists in an environment like "Lunar Lander" or "Biped walker" where it observes and takes actions. For every action the agent takes in a state, it receives a penalty or a reward.

Figure 2.3: Q Learning



In Q learning agent learns a Q-function to improve it's actions and maximize rewards over timesteps. Preliminaries of Q learning (Q$< s, a, R, s' >$):

- Let s be the current state and s' be the next state.
- R is the reward for action a in state S
- α is the learning rate and γ is the discount factor

Q function is given by:

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

Agent explores and exploits the environment and keeps updating the Q function to develop it's experience and take better actions to maximize the reward. In order to make sure Q-learning is able to generalize well, the ϵ-greedy action selection strategy has been followed in this work. The value of ϵ starts high and decreases over time. As a result, the agent follows a strategy that helps to explore more initially and gain experience by experimenting with the environment. As the agent gains sufficient exposure to the environment, it it necessary to exploit right areas and take right actions to maximize the rewards. During exploration, agent takes a bunch of random

actions which lead to new experiences. On the other hand, the exploitation agent takes the best action from the q-values.

2.2.3 Deep learning framework

- Deep Q-Network (DQN) [Mnih *et al.* (2013)]:

  Traditional Q learning keeps a track of its experience for every state-action pair in a table and uses its experience to take better action which can maximize the rewards. When these state and action pairs grow exponentially, it becomes infeasible to keep track in a table. As a result, there is a need for a technique that can generalize a variety of state-action pairs and predict q values such that taking action corresponding to maximum q-value would result in the maximization of the agent's rewards.

  With Deep learning, we need data to train a neural network. In order to use the agent's experience during training, we keep a track of state, action, next state, and reward within a tuple. A list of such tuples constitutes an experience replay. Usually, in traditional Q-learning, past experience is not used and the agent learns at every timestep. Experience replay enables the use of batches of state transition for learning. As a result, training is more stable. Also, the model is not instantly adjusted to correct values but learns little by little depending on the learning rate.

  Double Q learning (van Hasselt *et al.* (2015)): It has been found that a deep Q learning model can overestimate expected rewards when target values for updating the same model is used for taking best action and calculating Q(s,a). In order to deal with this, we use 2 different models. Model θ is used for action selection where as θ' is used for calculating target values. Target model

is updated by weights of main model every 4 time steps.

$$\text{Double q learning target: } R_{t+1} + \gamma Q\left(S_{t+1}, \text{argmax}_a\, Q\left(S_{t+1}, a; \theta\right); \theta'\right) \quad (2.1)$$

- Deep Deterministic Policy Gradient (DDPG) (Lillicrap *et al.* (2019)):
  Unlike DQN, DDPG is specifically designed to tackle continuous reinforcement learning problems. In order to deal with continuous action space DDPG uses Q-network, target Q-network, deterministic policy network, and target policy network. This is similar to an actor-critic architecture where the actor is responsible for taking actions and the critic evaluates the action. In order to update actor and critic, we use values from the target network which are time-delayed copies of original actor and critic that undergo soft updates over time. DDPG outputs a continuous action value in contrast to probabilities for possible actions and choosing max out of those. DDPG works with the help of the following components:

  - Experience Replay: Similar to DQN, DDPG also uses a replay buffer which helps in training randomly sampled batch which is used to train the network.
  - Actor (Policy) and Critic (Value) Network Updates: The actor takes the state as the input and gives action as the output. The critic takes state and action as the input and outputs a q-value for the state-action pair. Actor and Critic are updated similarly to the Q-value update in DQN using the Bellman equation. Target networks are used to predict the information of the next state.
  - Target networks: At definite intervals, weights from the main network are copied to the target network which enables the use of time-delayed values

- Exploration: In order to promote exploration DDPG introduce randomness. Action values are subjected to adding Ornstein–Uhlenbeck noise which enables the agent to explore well.

2.3 C-DDPG Architecture

- Training Behaviors:
  - Hovering (Lunar Lander): In this behavior the agent keeps pumping the engine to fly around in any direction, making sure the agent doesn't crash or land on the ground. In order to achieve this behavior, the agent is positively rewarded for the timesteps its leg doesn't touch the ground and the agent doesn't crash. It get minor penalty for fuel usage. Agent is trained for 250 episodes and it gradually learns to fly around. It comes close to the ground but makes sure it doesn't crash or touch the ground. In order to train this behavior a DQN was implemented that would use double q learning and generalize it's experience to successfully learn the behavior.
  - Reach center (Lunar Lander): In this behavior the agent is supposed to learn how to get closer and closer to the landing area, irrespective of safe landing or crashing. In order to achieve this behavior, distance from the landing area is measured and as the distance decreases agent is awarded positively and as the distance increases agent is awarded negatively. DQN has been implemented with double Q learning to achieve this behavior.
  - Walk with big steps (Biped Walker): Our agent, the biped walker, needs to learn walking with bigger steps on grass terrain. In order to train this behavior we use DDPG algorithm. Depending on the action the agent

takes, it is negatively rewarded for falling down, going in the opposite direction and there's a minor penalty for torque as well. At the same time, agent is positively rewarded for taking steps forward with a condition that if the step is bigger it is rewarded more. As a result agent learn to take big and small jumps/steps and keep walking forward.

  - Balance (Biped Walker): In this behavior Biped Walker is supposed to learn to balance it's hull/head between absolute 0-30 degrees. It is allowed to jump and walk around without being rewarded. In this process it is given absolute negative reward if it goes beyond the specified range of angles. It scores a positive reward if angle is maintained between the given range. We use DDPG as the learning algorithm to train this behavior.

- Combining Behaviors:

  - Lunar Lander: Once we have trained behaviors for hovering around and reaching the center, our goal is to combine these into landing safely in the center. In order to achieve this we formulate a reinforcement learning algorithm using DDPG where the combination agent can take 2 actions which correspond to weights which are used to combine behaviors. Behavior 1 and behavior 2 predict a 4x1 probability vector from DQN which has been previously trained. Each behavior would try to assign maximum probability to the action that yields maximum individual reward. The combination agent predicts weight vector which is used for linear combination of behaviors. Taking argmax over the result of this linear combination, the Lunar Lander agent executes the action and collects the reward by interacting with the environment. For every state this learning strategy is able to generalize which combination would best maximize the overall reward for

the Lunar Lander environment.

– Biped Walker: Models from Biped Walker in grass terrain are trained and combined using the combination agent in Hardcore environment which consists of steps, pits and obstacles. In order to train the combination agent, each behavior outputs a command vector. Each command vector from a behavior is targeted at maximizing it's individual reward which is either walking with larger steps or balancing in our case. Combination agent uses DDPG to predict weight values which are used for linear combination of command vectors. This linear combination results into a resultant command vector that enables the agent to take advantage of balancing and walking on the difficult terrain.

Algorithm 1: C-DDPG Algorithm To Combine Behaviors

Train Behaviors $1, 2, 3...n$;

Save models for Behaviors $1, 2, 3...n$;

load models for Behaviors $1, 2, 3...n$;

foreach *episode* **do**

  Initialize S;

  foreach *step of episode* **do**

    $B_1, B_2...B_n \leftarrow$ predictions of models for Behaviors $1, 2, 3...n$ in S;

    Predict $[W_1, W_2, ...W_n]$ in S using policy derived from θ_{DDPG} ;

    Take action $A \leftarrow W_1 \times B_1 + W_2 \times B_2 + ...W_n \times B_n$, observe R, S';

    Update experience replay and learn θ_{DDPG};

    $S \leftarrow S'$;

  end

end

Putting it all together in a reinforcement learning loop, as we can observe in algorithm 1, the process starts with learning and saving models for individual behaviors. Behaviors are frozen which is analogous to the frozen architecture described by Lu *et al.* (2021). For every episode, at every step agent needs to learn to predict a set of weights such that linear combination helps the agent achieve required characteristics from different behaviors. For the linear combination, predicted values from saved models are multiplied by the set of weight vector given by the combination agent. As this process goes on, combination agent learns to predict relevant weight values for every state to achieve required combination and enables the agent to accomplish the goal.

Chapter 3

EXPERIMENTS, RESULTS AND ANALYSIS

3.1 Experiments

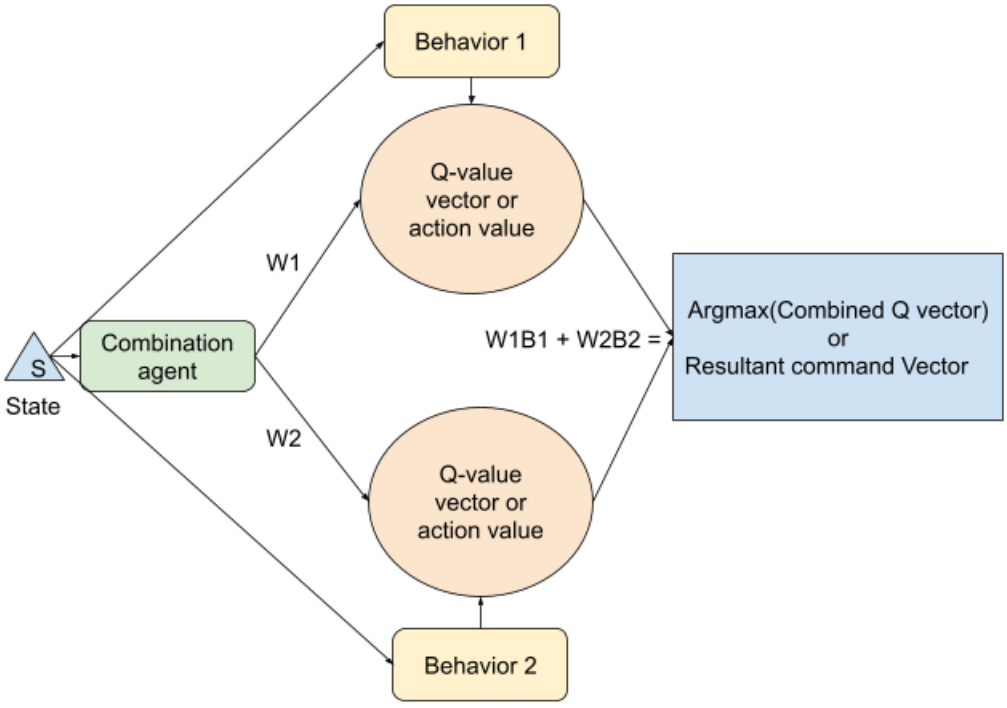
As we see in the methodology section, the implementation consists of 2 algorithms: DQN and DDPG for discrete and continuous scenarios respectively. As we see in the figure 3.1, each behavior outputs a q vector in discrete environment and action values in a continuous environment. The goal of the combination function is to combine these vectors in such a way that the agent learns the characteristics of both behaviors. We broadly divide the experiments into 2 categories, Discrete Action Domain and Continuous Action Domain.

3.1.1 Discrete Action Domain

In a discrete reinforcement learning environment, the agent is allowed to choose an action from a fixed number of actions. In this section, we describe how each experiment is divided and inner working during the training process.

- Training behaviors and a combination:
  - Lunar Lander Behavior 1: As described in the methodology section, the goal of this behavior is to fly around the environment. The reward function for this behavior is formulated using horizontal and vertical speeds and a boolean variable indicating if the leg of the agent has touched the ground. Using the resultant speed calculation, the agent is penalized for unstable speed and crashing. Using the boolean variable, we need to track whether

Figure 3.1: Combination Framework Using DDPG



the agent has touched the ground or not. If the agent flies again after the leg touches the ground it is rewarded negatively. Fuel consumption also constitutes a minor negative reward available from the openAI gym environment. DQN with input layer dimension = input state space = 8, 3 dense layers with 64 neurons each and relu activation, output layer dimension = action space = 4 and softmax activation is used to train using double q-learning update strategy. A learning rate of 0.001 and masked Huber loss is used in DQN. The simulation agent builds its experience over 250 episodes with 1000 steps. For exploration vs exploitation trade-off, ϵ-greedy policy is appointed. We are able to achieve the end goal which is to get a behavior where the agent can fly around without crashing or landing on the ground which can be reused in combination.

– Lunar Lander Behavior 2: As described in the methodology section, the goal of this behavior is to get closer to the center where the landing area is located. Reward function for this behavior is formulated using horizontal and vertical coordinates which helps in calculating Euclidean distance from the center which is located at (0,0). At every timestep, if the agent makes a move that enables it to minimize the distance, the agent is rewarded positively. In this behavior, the agent is not penalized for unstable speed and crashing. Fuel consumption also constitutes a minor negative reward available from the openAI gym environment. DQN with input layer dimension = input state space = 8, 3 dense layers with 64 neurons each and relu activation, output layer dimension = action space = 4 and softmax activation is used to train using double q-learning update strategy. A learning rate of 0.001 and masked Huber loss is used in this DQN. The simulation agent builds its experience over 350 episodes with 1000 steps. For exploration vs exploitation trade-off, ϵ-greedy policy is appointed. We are able to achieve the end goal which is to get a behavior where the agent can get closer to the center which can be reused in the combination.

– Lunar Lander Combination (C-DDPG): The goal of combining behaviors is to achieve a combination of flying safely and landing in the center without crashing. The combination is assumed to be an agent having 2 continuous actions. In order to train this agent, DDPG is used. These 2 actions correspond to 2 weights needed for a linear combination of the 2 behaviors. Precisely, behavior 1 and behavior 2 output q vectors such that the highest probability is given to the action that best suits the respective behavior. At the same time DDPG with actor configuration as input layer dimension = input state space = 8, 2 dense layers with 256 neurons each and

relu activation, output layer dimension = behavior space = 2 and softmax activation and critic configuration as input layer dimension = input state space = 8, 2 dense layers with 16 and 32 neurons respectively and relu activation, output layer dimension = 1, is appointed to weight each behavior. Linear combination of behavior 1 and behavior 2 results in a q vector out of which action corresponding to maximum q value is picked and executed. As the training process goes on, the combination agent outputs a q vector such that the maximum q value enables the agent to learn the combination. The agent is rewarded based on the original rewards from the gym environment. Training is done for about 1000 episodes.

- Basic RL:

  This is the baseline experiment where the agent is trained on the original Lunar Lander OpenAI gym environment. DQN with input layer (dimension = input state space = 8), 3 dense layers with 64 neurons each and relu activation, output layer (dimension = action space = 4) and softmax activation is trained for 800 episodes.

- Multi-objective RL with linear scalarization:

  In order to combine multiple q-vectors, MORL uses the traditional idea called linear scalarization. Weight values are fixed by the domain expert and linear combination is possible which enables the agent to pick an action with maximum q value. Here, w1=0.6 and w2=0.4 because flying is comparatively more important than getting to the center as crashing leads to a high negative reward. C-DDPG architecture on the other hand evaluates weight at every timestep and learns to map state to weight vector for the desired combination. As a result, weights change in C-DDPG architecture but remain constant in the linear

scalarization approach.

- Hierarchical machines inspired learning:

  In this technique, C-DDPG architecture for combination is used to predict weights but instead of linear combination, the behavior whose's weight value is higher is selected. As a result, only one of the trained behavior is executed at a time to take an action in the Lunar Lander environment where the reward is the original function from the OpenAI gym environment.

3.1.2 Continuous Action Domain

Command vectors are essentially the continuous actions the agent can take in the environment. Each model starts training from a known set of good weights for the Biped walker enabling the use of the bootstrap approach. Experimentation details on the Biped Walker environment can be given by the following:

- Training behaviors and a combination:

  Architecture for training behavior can be given by:

  - Actor: Input layer dimension = input state space = 8, 2 dense layers with 600 neurons in the first layer and 300 neurons in the second layer along with each activated by relu activation, output layer dimension = action space = 4 and tanh activation.
  - Critic: Input layer dimension = input state space = 8, 3 dense layers with 600 neurons in first layer and 300 neurons in second layer and third layer along with each activated by relu activation, output layer dimension = 1 and tanh activation.

  The following describes the training of each behavior and combination in detail:

– Biped Walker Behavior 1: As described in the methodology section, the goal of this behavior is to walk using long and short steps on a grass terrain without obstacles. Agent's position with respect to the previous step is measured and if the agent moves forward with a bigger step it gets a higher reward. If the agent moves backward or falls down, it scores a negative reward. After training the end goal is to use this behavior in association with another behavior to walk on terrain with obstacles.

– Biped Walker Behavior 2: The goal of the agent is to balance hull/head angle. In order to train such a behavior, DDPG is trained where the agent scores negative angle as the reward when the hull angle is beyond absolute(30) degrees and e^{-angle} as a positive reward when the angle is between -30 to 30 degrees. The agent is not restricted in moving direction and is free to walk and jump around as long as it manages to learn to maintain head angle. This behavior can be combined to balance in uneven terrain scenarios.

– Biped Walker Combination: The agent is placed in a grass terrain with obstacles like stairs, pits and stones. The goal of the combination algorithm is to combine walking with large and short steps and balancing behavior to walk a larger distance in "hardcore" terrain using the original rewards from the Biped walker hardcore gym environment. Biped walker hardcore v3 is a difficult environment that cannot be solved by DDPG but we intend to demonstrate behavior combination and agent's efforts to maximize the rewards. Behavior 1 outputs a command vector that is responsible for controlling the agent to walk and behavior 2 outputs a command vector that is responsible for maintaining a balance of the agent. At the same time DDPG with actor configuration as input layer dimension = input

state space = 8, 2 dense layers with 256 neurons each and relu activation, output layer dimension = behavior space = 2 and softmax activation and critic configuration as input layer dimension = input state space = 8, 2 dense layers with 16 and 32 neurons respectively and relu activation, output layer dimension = 1 is appointed to weight each behavior. As a result, the output is a linear combination of command vectors which we call the resultant command vector or the combination command vector. With the training process, the agent learns to weigh behaviors in order to get desired combined behaviors. In addition to this, we also experiment on combining 3 behaviors. The agent learns to combine balance, walking with short steps and walking with large steps in order to better overcome the obstacles in the difficult domain.

- Basic RL:

  The goal of the agent is to walk in a hardcore biped walker domain where it faces obstacles, stairs and pits. The agent uses the DDPG bootstrap model to train for 2000 episodes with 2000 steps in each episode. The agent follows the original reward from Biped walker hardcore OpenAI gym environment where it is given a high negative reward for falling down and positive rewards for overcoming obstacles. The architecture of the DDPG model can be given by the following:

  - Actor: Input layer dimension = input state space = 8, 2 dense layers with 600 neurons in first layer and 300 neurons in second layer along with each activated by relu activation, output layer dimension = action space = 4 and tanh activation.
  - Critic: Input layer dimension = input state space = 8, 3 dense layers with

600 neurons in the first layer and 300 neurons in the second layer and third layer along with each activated by relu activation, output layer dimension = 1 and tanh activation.

- Transfer learning:

  Using this approach, the agent has to reuse or adapt to the new domain of Biped hardcore version by using it's experience from walking on grass terrain. Initially, a bootstrap model is trained on grass terrain using original rewards from OpenAI gym environment. Later, this model is reused such that the model's dense layers(do not undergo training) are used for feature extraction but a new last layer is added which is responsible for adapting to new a environment by training and updating the weights. The skill of walking mutates to adapt to the obstacles in the target environment, such that the agent learns to overcome few obstacles. Since DDPG is unable to converge in this environment, we experiment to see how well does the transfer learning agent adapts to the target domain of a Biped Walker hardcore environment.

- Multi-objective RL with linear scalarization:

  Linear scalarization approach can also be used in scenarios where command vectors need a weighting mechanism to get the resultant vector. Deciding the trade-off can be difficult but it is necessary for the agent to walk a greater distance overcoming the obstacles in Biped Walker hardcore environment to walking behavior is multiplied with its weight of 0.7 and balancing behavior is multiplied with its weight of 0.3. Since the agent doesn't adapt the combination based on the current state information, it yields different results than the combination agent.

- Hierarchical machines inspired learning:

At a given time, HRL inspired technique chooses the command vector from the behavior whose weight is higher. So this strategy doesn't deal with a linear combination of command vectors but intends to apply particular behavior given the current situation. It uses the same DDPG network to learn and update weights as the combination but since only one behavior is active in the environment the experiment yields different results.

3.2 Results and Analysis

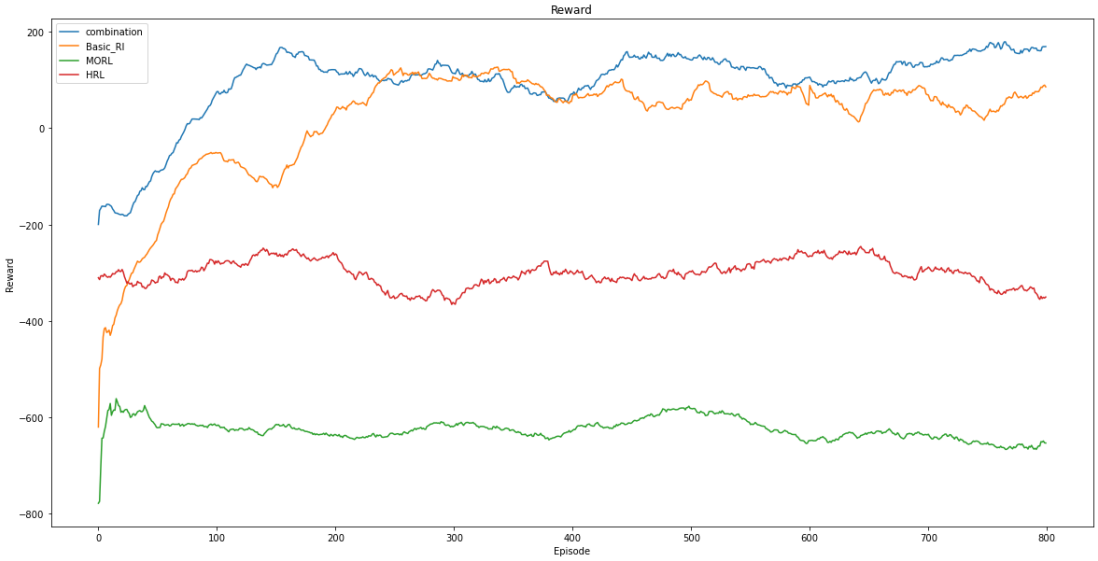
In order to evaluate the C-DDPG architecture, we evaluate how individual behaviors learn and compare the combination with basic RL, Multi-objective RL (Linear scalarization), Hierarchical machines inspired learning, and Transfer learning. Since C-DDPG works with continuous action space and discrete action space, we have divided this section based on discrete environment (Lunar Lander) and continuous environment (Biped Walker).

3.2.1 Lunar Lander

- Behavior 1 and 2:

  Training simpler behavior is easier than training an end-to-end agent on a lunar lander because the reward function is simpler and the agent begins to demonstrate learned behavior soon. These behaviors may not convey meaningful actions with respect to the environment's main goal but they contribute to be a part of the expected strategy. For example, when we divide the behaviors into hovering and reaching the center, we expect the agent to balance around (not crash) and also get closer and closer to the center. This makes it possible for the behaviors to be combined such that it can land safely in the center and achieve its goal.

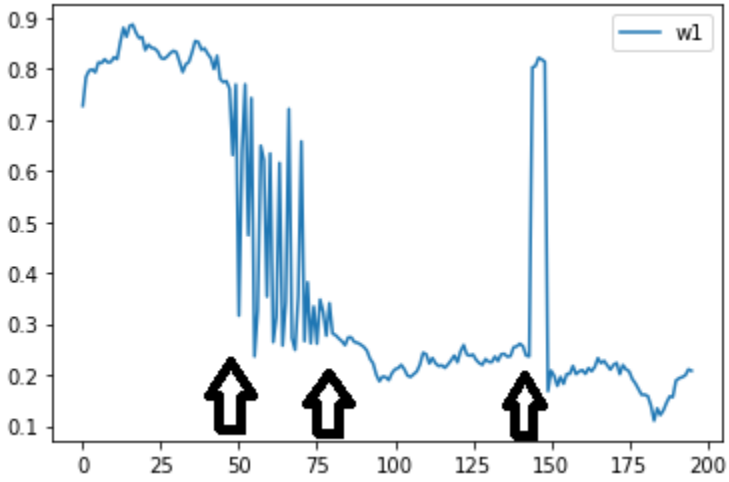
Figure 3.2: Lunar Lander Comparison [Combination Vs Basic RL Vs Multi-objective RL With Linear Scalarization Vs Hierarchical Machines Inspired Learning]



- Combination DDPG architecture outperforms basic RL, MORL (linear scalarization), and Hierarchical machines inspired learning. As we can see in the graph (figure 3.2), combination architecture is able to generalize weight mechanism depending on the state and gradually after about 170 episodes, we see that the agent successfully hovers around the center and lands safely. There are minor oscillations in the reward which can possibly be due to random initialization of the environment that make it difficult for the agent. The agent doesn't fail because of these difficulties but might take more time which impacts the rewards. Overall it rarely crashes and learns to land safely which maximizes the rewards.

  We can see in the Steps vs W1 plot (figure 3.3) that combination plays an important role. Initially, lander is free to give lower preference to hovering and more preference to navigate towards the center. Gradually, as it reaches closer to land, hovering is weighted higher to prevent crashing. After stabilizing or partly landing, if the agent is still not in goal position, center behavior spikes

Figure 3.3: Lunar Lander Steps Vs W1. (W1=Center, W2=Hovering)



and helps the agent achieve the goal by reaching the center and stabilizing in the end.

- Basic RL learns to accomplish the goal of the Lunar Lander environment (Landing safely in the center) after about 275 episodes as we can see in the figure 3.2. This approach also suffers from minor oscillations due to random initialization as mentioned earlier. Over a period of time, combination architecture and basic RL perform equally well such that the agent maximizes its rewards by achieving the end goal.

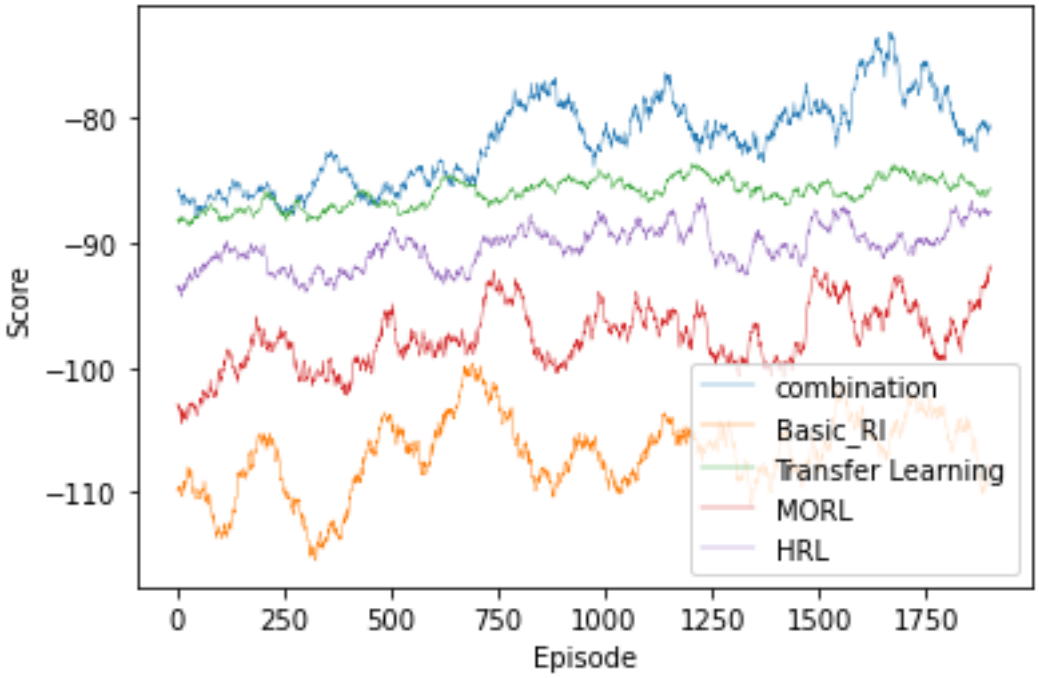
- MORL (linear scalarization) fails to achieve the right combination (As shown in figure 3.2) for the lunar lander environment. It is necessary to evaluate the position of the agent before determining which strategy can be helpful. For example,initially, the should fly stable and gradually proceed towards the center. Also, as it reaches the goal, hovering behavior should help the agent slow down for a safe landing. As going towards the center is not much important when the agent is about to crash, hover behavior needs to dominate such that after stabilizing the agent can fly towards the center and land safely. Since a

constant value is unable to realize the difference in the state of the agent, Linear scalarization approach needs to be modified by C-DDPG architecture to achieve good performance.

- Hierarchical machines inspired learning approach fails (As shown in figure 3.2) because of obvious reasons. Behaviors learned earlier are meant to be used in a combination and one behavior at a time is not sufficient. For example, when the agent prefers going towards the center behavior, it may lose balance or turn upside down. Even if the control goes to hovering behavior in the very next step, it fails to stabilize the heading angle and crashes. As a result, the agent collects negative rewards. This also demonstrates the need of combination rather than smaller modules individually trying to accomplish the bigger goal.

3.2.2 Biped Walker

Figure 3.4: Biped Walker Comparison [Combination Vs Basic RL Vs Multi-Objective RL With Linear Scalarization Vs Hierarchical Machines Inspired Learning]

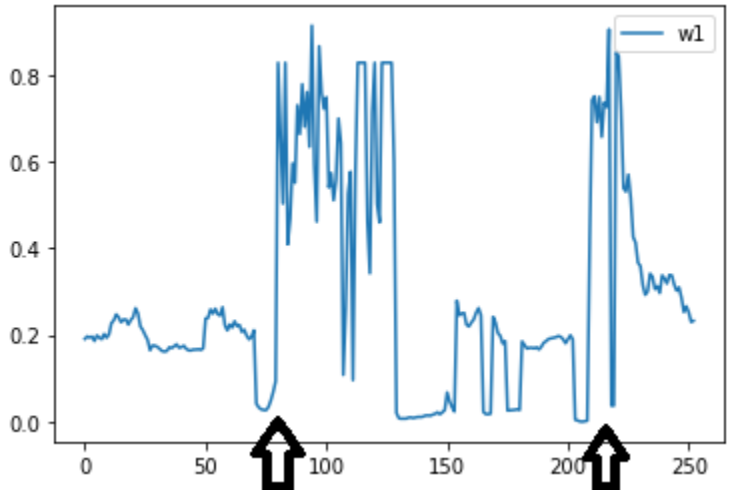


- Behavior 1 and 2:

  Learning to walk with big and small steps and balancing are simpler versions of biped walker environment since the agent channels its energy towards a simpler goal which is similar to the lunar lander environment. We are also able to inculcate a skill to take larger steps by hopping around and balancing while taking any action. Individually one behavior is not sufficient to make the agent demonstrate its skills in the hardcore version of biped walker environment.

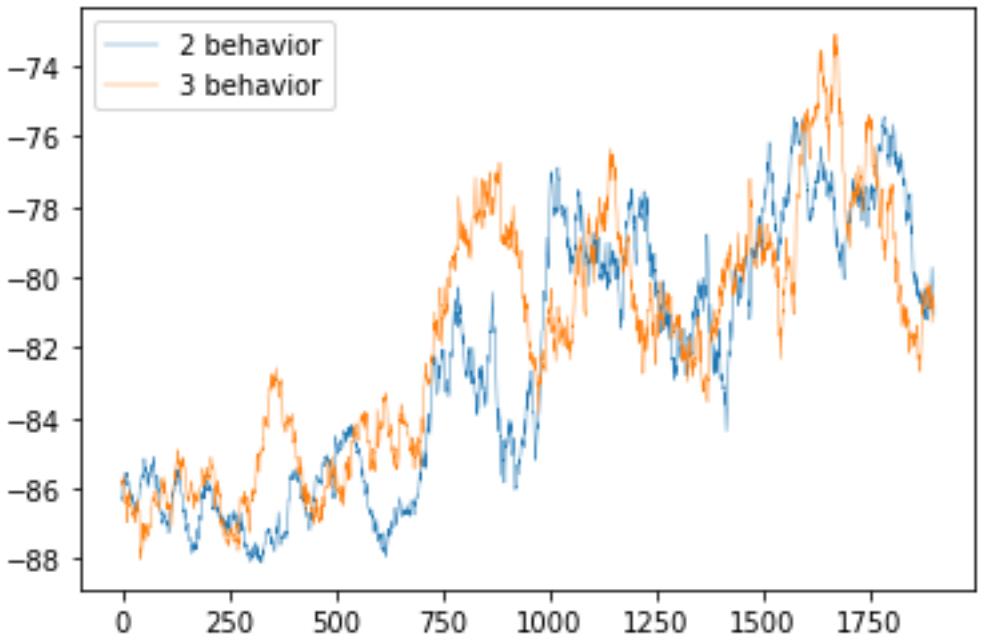
- Combination DDPG architecture enables the agent to overcome different types of obstacles using trained behaviors (As shown in figure 3.4). For example, if there are stairs followed by a pit then the agent is able to hop walking forward and balance its head angle helping it to overcome stairs and maintain the balance at the same time which helps the agent to be in a stable position while it encounters the next obstacle. The hardcore version 3 of biped walker cannot be solved by the DDPG algorithm but we successfully demonstrate combined behavior which enables the agent to overcome a variety of obstacles like stairs (going up or down), pits, or small/big stones.

Figure 3.5: Biped Walker Steps Vs W1. (W1=Balance, W2=Walk)



We can see in the Steps vs W1 plot (figure 3.5) that the agent prefers walking behavior initially but as soon as it encounters obstacles, the balance behavior spikes which help the agent stabilize and continue walking along with balancing. Later the agent runs into another obstacle and balances spikes up again but this time, unfortunately, the agent falls down and the episode ends.

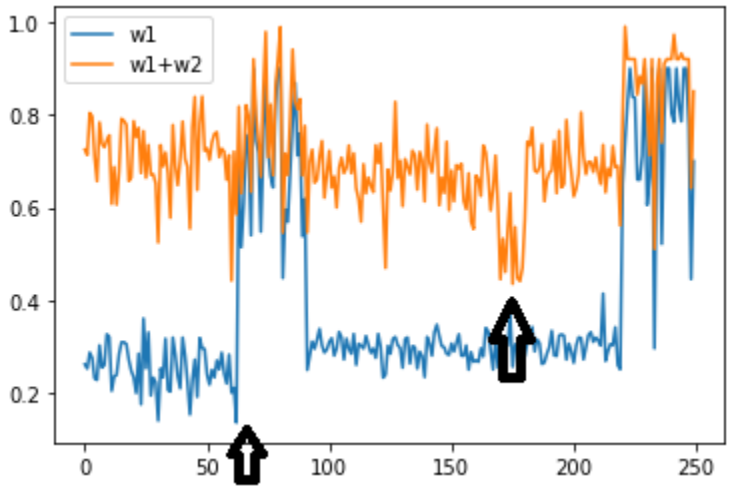
Figure 3.6: 3 Behavior Vs 2 Behavior Combination



- Combination DDPG architecture is also able to achieve 3 Behavior combination. The agent is also able to tackle various obstacles with the resulting combination of 3 behavior combination. When 3 behavior combination is compared with 2 behavior combination, it is observed that learning curve is similar. In some situations, the 3 behavior combination approach tends to collect better rewards than the 2 behavior approach as seen in figure 3.6.

  We can see in figure 3.7, Steps vs W1 and W1+W2 that the agent prefers walking behavior (with small steps or big steps) initially but as soon as it encounters obstacles the balance behavior (W1) spikes which helps the agent

Figure 3.7: 3 Behavior Steps Vs W1 And W1+W2



stabilize and continue walking along with balancing. We also observe that W1+W2 valley around step 170, representing more weightage to W3 which belongs to the behavior corresponding to walking with larger steps (helps the agent jump off an obstacle). The rest of the time agent combines balance and 2 walking behaviors to keep moving forward in a steady manner.

- Basic RL has very little prior knowledge from the bootstrap model and so most of the time the agent spends is learning to walk and at the same time dealing with obstacles. Since the agent keeps falling down due to imbalance and obstacles, it ends up collecting more negative rewards (As shown in figure 3.4). Since DDPG cannot solve this environment, we compare every technique for a fixed number of episodes and basic RL doesn't demonstrate much improvement over 2000 episodes.

- Transfer Learning is an interesting strategy where the agent already knows walking on the grass terrain and it needs to adapt to obstacles in the hardcore domain of biped walker. We observe that the agent is able to cover some distance where there are no obstacles which enable the agent to collect rewards

better than basic RL (As shown in figure 3.4), MORL, and hierarchical machines inspired learning. After a lot of training, the agent is still unable to overcome obstacles like stones and pits. This approach is the next best alternative to a combination DDPG architecture.

- MORL (linear scalarization) doesn't demonstrate a good working on the static combination for hardcore biped walker environment (As shown in figure 3.4). Similar to the lunar lander environment, there is a need to change combinations based on the state of the agent. For example, if the agent is climbing and if the agent is passing a pit, different walk and balance components are required. For climbing, walking is more important compared to the pit where balancing and walking are almost equally important. As a result, we do not see the agent adapting to obstacles in the domain and improving the performance over time.

From the experimentation and results, our analysis shows that the following makes C-DDPG architecture an interesting way of learning:

- We use a deep learning framework to learn behaviors and use those behaviors in a weighted combination to achieve a combination of behaviors. Learning weights also uses deep learning.

- This makes it difficult to compare with simpler domains like grid worlds but makes it easy to generalize to complex state space representations.

- C-DDPG framework doesn't work on image as a state input (This can be a possible extension). We rely on raw sensor input to learn policies.

- C-DDPG framework can be generalized for both continuous and discrete domains since learning weights is modeled as a deep RL task in addition to learning behaviors using deep RL.

- Multi-objective RL focuses on combination learning but to the best of our knowledge, it need not account for the state being a parameter for learning weight. On the other hand, HRL doesn't focus on combinations (Focuses on one subtask instead of a combination of subtasks). As a result, we combine HRL and MORL to learn combinations by building RL framework that learns how to combine behaviors.

- If a learned behavior needs additional skills, a new skill can be added by learning weights for combination and subtask to achieve that skill. This can enable learning safety and then learning complex moves in an agent.

Chapter 4

FUTURE WORK AND CONCLUSION

4.1 Future work

This work motivates us to think in the direction of modular learning strategies and combining behaviors. In the future, this method can be extended to real-world applications on the hardware. It would be a unique experiment if this method from simulation can make it to real-world learning like self driving cars, robots, etc. Modules for each individual behavior can also be used for more accurate performance depending on the shortcoming of a behavior and modified to incorporate safety. If each simpler module can address safety individually then then we only need to ensure that the combination does not violate any safety constraints.

This approach can also be extended towards making AI more interpretable. A key property of our combination approach is attributability. In particular, the combination allows us to easily attribute rewards to different behaviors. Consequently, methods can be developed for explainable AI ~Gunning and Aha (2019); Chakraborti *et al.* (2017a), for example, by generating explanations ~Chakraborti *et al.* (2017b); Zakershahrak *et al.* (2020a,b) to achieve explainable RL~Wells and Bednarz (2021).

In addition to this combination algorithm can be modified in a way such that instead of linear combination, complex non-linear combination can be experimented depending on the complexity of the task. Meta-learning can also be used in order to learn a better combination. As every deep-learning framework may suffer from problems like catastrophic forgetting, overfitting, local optima, etc, this approach may suffer from such drawbacks as well and so we leave it open for future researchers

to experiment more with this C-DDPG architecture and overcome the challenges.

4.2 Conclusion

In this thesis, the idea of combining behavior is simplified and a unique strategy to achieve modular and efficient combination is experimented. The process begins by simplifying the rewards to learn behaviors. Learned behaviors are saved and used in C-DDPG architecture where the strategy to combine involves reinforcement learning to assign weights for appropriate linear combination. This linear combination helps the agent take resultant action in the environment such that it demonstrates characteristics of both behaviors and enables the agent to achieve the goal faster.

It has been demonstrated that this approach can be useful in both continuous and discrete environments but since deep learning is a field that keeps evolving, a different method or architecture can be used with the same concept of combination in mind which would enable the growth of this area of research as well.

REFERENCES

Andreas, J., M. Rohrbach, T. Darrell and D. Klein, "Learning to compose neural networks for question answering", arXiv preprint arXiv:1601.01705 (2016).

Arkin, R., "Motor schema based navigation for a mobile robot: An approach to programming by behavior", in "Proceedings. 1987 IEEE International Conference on Robotics and Automation", vol. 4, pp. 264–271 (IEEE, 1987).

Bacon, P.-L., J. Harb and D. Precup, "The option-critic architecture", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 31 (2017).

Bahdanau, D., P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville and Y. Bengio, "An actor-critic algorithm for sequence prediction", arXiv preprint arXiv:1607.07086 (2016).

Balch, T. and R. C. Arkin, "Behavior-based formation control for multirobot teams", IEEE transactions on robotics and automation **14**, 6, 926–939 (1998).

Barreto, A., S. Hou, D. Borsa, D. Silver and D. Precup, "Fast reinforcement learning with generalized policy updates", Proceedings of the National Academy of Sciences **117**, 48, 30079–30087 (2020).

Barto, A. G. and S. Mahadevan, "Recent advances in hierarchical reinforcement learning", Discrete event dynamic systems **13**, 1, 41–77 (2003).

Brooks, R., "A robust layered control system for a mobile robot", IEEE journal on robotics and automation **2**, 1, 14–23 (1986).

Cai, Y., S. X. Yang and X. Xu, "A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments", in "2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)", pp. 52–59 (IEEE, 2013).

Chakraborti, T., S. Kambhampati, M. Scheutz and Y. Zhang, "Ai challenges in human-robot cognitive teaming", arXiv preprint arXiv:1707.04775 (2017a).

Chakraborti, T., S. Sreedharan, Y. Zhang and S. Kambhampati, "Explanation generation as model reconciliation in multi-model planning", CoRR **abs/1701.08317**, URL `http://arxiv.org/abs/1701.08317` (2017b).

Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume and J. Langford, "Learning to search better than your teacher", in "International Conference on Machine Learning", pp. 2058–2066 (PMLR, 2015).

D'Eramo, C., D. Tateo, A. Bonarini, M. Restelli and J. Peters, "Sharing knowledge in multi-task deep reinforcement learning", in "International Conference on Learning Representations", (2019).

Dietterich, T. G., "The maxq method for hierarchical reinforcement learning.", in "ICML", vol. 98, pp. 118–126 (Citeseer, 1998).

Doroodgar, B. and G. Nejat, "A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments", in "2010 IEEE International Conference on Automation Science and Engineering", pp. 948–953 (IEEE, 2010).

Google images, "Bipedal walker diagram", URL `https://miro.medium.com/max/1400/1*1hZSWhdg1BKy9iXlPA_luA.png`, [Online; accessed June 27, 2021] (2021a).

Google images, "Lunar lander diagram", URL `https://miro.medium.com/max/1346/1*i7lxpgt2K3Q8lgEPJu3_xA.png`, [Online; accessed June 27, 2021] (2021b).

Graves, A. and C. Czarnecki, "Design patterns for behavior-based robotics", IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **30**, 1, 36–41 (2000).

Gunning, D. and D. Aha, "Darpa's explainable artificial intelligence (xai) program", AI Magazine **40**, 2, 44–58 (2019).

Hospedales, T., A. Antoniou, P. Micaelli and A. Storkey, "Meta-learning in neural networks: A survey", arXiv preprint arXiv:2004.05439 (2020).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning", (2019).

Liu, C., X. Xu and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview", IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**, 3, 385–398 (2014).

Lu, K., A. Grover, P. Abbeel and I. Mordatch, "Pretrained transformers as universal computation engines", arXiv preprint arXiv:2103.05247 (2021).

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning", (2013).

Randlov, J., "Learning macro-actions in reinforcement learning", Advances in Neural Information Processing Systems pp. 1045–1051 (1999).

Roijers, D. M., P. Vamplew, S. Whiteson and R. Dazeley, "A survey of multi-objective sequential decision-making", Journal of Artificial Intelligence Research **48**, 67–113 (2013).

Schweighofer, N. and K. Doya, "Meta-learning in reinforcement learning", Neural Networks **16**, 1, 5–9 (2003).

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search", nature **529**, 7587, 484–489 (2016).

Sprague, N. and D. Ballard, "Multiple-goal reinforcement learning with modular sarsa (0)", (2003).

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

Vamplew, P., R. Dazeley, A. Berry, R. Issabekov and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms", Machine learning **84**, 1, 51–80 (2011).

Vamplew, P., J. Yearwood, R. Dazeley and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts", in "Australasian joint conference on artificial intelligence", pp. 372–378 (Springer, 2008).

van Hasselt, H., A. Guez and D. Silver, "Deep reinforcement learning with double q-learning", (2015).

Van Moffaert, K., M. M. Drugan and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques", in "2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)", pp. 191–199 (IEEE, 2013).

Vanschoren, J., "Meta-learning", in "Automated Machine Learning", pp. 35–61 (Springer, Cham, 2019).

Vezhnevets, A. S., S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning", in "International Conference on Machine Learning", pp. 3540–3549 (PMLR, 2017).

Vilalta, R. and Y. Drissi, "A perspective view and survey of meta-learning", Artificial intelligence review **18**, 2, 77–95 (2002).

Vithayathil Varghese, N. and Q. H. Mahmoud, "A survey of multi-task deep reinforcement learning", Electronics **9**, 9, 1363 (2020).

Watanabe, T. and T. Sawa, "Instruction for reinforcement learning agent based on sub-rewards and forgetting", in "International Conference on Fuzzy Systems", pp. 1–7 (IEEE, 2010).

Watkins, C. J. C. H., "Learning from delayed rewards", (1989).

Weitzenfeld, A., R. Arkin, F. Cervantes, R. Olivares and F. Corbacho, "A neural schema system architecture for autonomous robots", in "Proc. of 1998 International Symposium on Robotics and Automation, ISRA'98", pp. 245–252 (1998).

Wells, L. and T. Bednarz, "Explainable ai and reinforcement learning—a systematic review of current approaches and trends", Frontiers in artificial intelligence **4**, 48 (2021).

Wiering, M. A. and H. Van Hasselt, "Ensemble algorithms in reinforcement learning", IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **38**, 4, 930–936 (2008).

Xiaoqin, D., L. Qinghua and H. Jianjun, "Applying hierarchical reinforcement learning to computer games", in "2009 IEEE International Conference on Automation and Logistics", pp. 929–932 (IEEE, 2009).

Yu, Y., "Towards sample efficient reinforcement learning.", in "IJCAI", pp. 5739–5743 (2018).

Zakershahrak, M., Z. Gong, N. Sadassivam and Y. Zhang, "Online explanation generation for planning tasks in human-robot teaming", in "2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 6304–6310 (2020a).

Zakershahrak, M., S. R. Marpally, A. Sharma, Z. Gong and Y. Zhang, "Order matters: Generating progressive explanations for planning tasks in human-robot teaming", (2020b).

Zhang, Y. and L. E. Parker, "Iq-asymtre: Synthesizing coalition formation and execution for tightly-coupled multirobot tasks", in "2010 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 5595–5602 (2010).

Zhang, Y. and L. E. Parker, "Iq-asymtre: Forming executable coalitions for tightly coupled multirobot tasks", IEEE Transactions on Robotics **29**, 2, 400–416 (2013).

Zhang, Y., S. Sreedharan, A. Kulkarni, T. Chakraborti, H. H. Zhuo and S. Kambhampati, "Plan explicability and predictability for robot task planning", in "2017 IEEE International Conference on Robotics and Automation (ICRA)", pp. 1313–1320 (2017).