SCHOOL OF MATHEMATICAL, PHYSICAL AND COMPUTATIONAL SCIENCES

A Study of Relational Structure in Multi-Discrete Action Spaces in Reinforcement Learning

by

Perusha Moodley

Thesis submitted for the degree of Doctor of Philosophy

PhD

COMPUTER SCIENCE

January, 2024

UNIVERSITY OF READING

To my Mum and Dad
Tilly and Reuben

Acknowledgements

Professor Xia Hong, thank you for taking me on as a PhD student; working with you over the years has shown you to be a very open-minded, curious person who gives genuine feedback and who has the best interests of her students at heart. I also appreciate that you pushed me to start writing early and for all the guidance you provided. I have a feeling that if I listened to you from the start, my journey would have been easier but apparently I had to do it my way!

Professor Benjamin Rosman, it is no exaggeration to say that without your supervision I would have given up a long time ago. I am grateful to the DL Indaba for introducing us and forever grateful to you for giving up your time, without hesitation, over so many years to coach me on a bi-weekly basis. Your positivity, wisdom, technical and other guidance picked me up and helped me to stay the course. I am very appreciative to you for what you did for me and will not forget it. Thank you too for making the effort to introduce me to people, some of whom have made my PhD life richer and are still friends.

I would also like to thank my PhD advisors Pat Parslow, Guiseppe di Fatta and Carmen Lam for supporting me over the years and turning what could have been gruelling monitoring sessions into interesting discussions. In addition I would like to thank Mark Trovinger for sharing his knowledge and experience with GPU related hardware and software systems that was very much appreciated! I would also like to thank Pramod Kaushik who pushed us to submit a paper to a NeurIPS workshop that was accepted! It was an amazing experience!

I met many kindred spirits on this journey, some who studied with me, others who collaborate with me even now. This has not been an easy journey; I therefore feel very lucky to have met and interacted with such a lovely group of collaborators, Maria Jacob, Saeed Taffazol, Pramod Kaushik, Mark Trovinger and Dhillu Thambi, who push me, advise me, help me understand, allow me to speak and most importantly, listen to me without judgement.

Two organisations that I felt really helped me include the Google Developer Group (GDG) and the ML Collective (MLC). As someone who has had trouble fitting in all her life, I felt like I discovered community for the first time when I joined the GDG as an organiser, running huge technical events drawing up to 300 people at times. My PhD journey started in the GDG and I think it is a brilliant endeavour! I realised quite late in my PhD the power of collaborating with other researchers and fortuitously discovered the MLC at a conference. I met most of my collaborators via the MLC which is a great place to meet researchers of various skill levels and offers opportunities to grow in a research environment.

Finally, and importantly, I would like to thank Dr Omer Dawelbeit, who sparked this PhD journey and was generous with his friendship and inspiration. He just assumed I could do it and had confidence in me. I hope I can pass that forward.

Personal Acknowledgements

I would like to acknowledge and thank my parents, Tilly, the pragmatist and Reuben, the visionary, responsible for the start of this unusual journey. My Father inspired me all his life; he had more spunk and get-up-and-go than anyone I have ever met. He made himself a man to be proud of despite numerous setbacks and hardships. He taught me the power of knowledge, to be a free thinker, to always get back up and never give up. He was a fighter, had an unerring ability to understand whatever you spoke about and an undying passion for knowledge. He believed in his 3 daughters, always told us we could achieve anything if we set our minds to it and never doubted our capabilities. I am so grateful I knew him and that I could benefit from his wisdom and his spirit. Tragically he passed away while I was working on my dissertation and I regret very much that he did not get to see me finish.

My Mum was my rock throughout my life. She was a constant force in the swirling rush of life. She always believed in us and gave us the pragmatic stability that has rooted me and allowed me to move forward. It is easy to under-appreciate stability but a PhD is an endeavour that tests all of you and I am grateful for this skill. Thank you for your unconditional love. It makes all the difference in this world. Like most people, it took me years to realise that my parents were something special. They had few of the benefits in life they bestowed upon us but they did something special and they should be proud of that!

My husband, Mark, I thank for being himself, genuinely interested and fascinated with the world, always ready for a discussion about any topic, a friendly smile and easy-going style … except when he reviews documents! Thank you for reviewing my dissertation, I know the document is better for it, and I am thrilled that you enjoyed the process. There are many things I am grateful for but, most importantly, thank you for keeping me sane, for balancing my intensity and for allowing me to make this long journey without complaint, only support. Not many people would do that and still be smiling at the end of it.

To my sisters and their families: Thosha and Renze, those weekly golf sessions became a stable point for me and I enjoyed it immensely. To Davina, Jan and Michael, thanks for being patient while I finished up; I have a lot of catching up to do and many missed birthdays. My sisters were always supportive of my decision and allowed me to get on with it, inspiring us with their own lives.

Dear Anne, you are an unconventional mother-in-law; I wish you saw yourself as we see you. I thank you for your interest, support and faith in me; it is very much appreciated. I wish you could have had the opportunity to complete your PhD.

My technical female friends Maria Jacobs, Luisa Zintgraf, Kubra Harmankaya, Amanda Cavallaro and Siobhan Hall; I know it should not matter, but it is lovely to know female techies! I am so happy to know you, feel very supportive of you and wish you all the best in life! My longer term technical friends Lisa Smith, Shireen Khan and Sharan Foga, you have inspired me with your amazing lives! I love that you are all so quirky and interesting! You are great role-models!!

My friends helped keep me sane: they listened patiently when I was unable or too stressed to meet them, dropped by for games of Die Crew, Pandemic, Quacks and Dominion, gave me kefir grains, sourdough starters and home made vinegar, generously invited us for a much needed holiday to Turkey, housed and fed us for the amazing Hay Book Festival, shared the precious Ted and Reggikins with us, and are just some of the most wonderful people I know. Janet, Julie, Steve you guys rock! Thank you for being such good friends to us and inspiring us to be better!

Finally, to the people who made that work of art, Ted Lasso - I Believe!!

Declaration: I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.
Signed: Perusha Moodley

Abstract

This thesis proposed three novel methods for learning and exploiting relational structure in the action space of reinforcement learning (RL) environments. It provides initial evidence across model-free online and offline RL algorithms that mechanisms adapted for extracting and exploiting action structure can mitigate key challenges in multi-discrete domains like sparse rewards and large action spaces. It is demonstrated that the proposed techniques could significantly improve the performance of RL algorithms in multi-discrete action spaces.

Firstly, a multi-task approach trains agents across diverse procedural tasks, using state-action visitations to identify task-agnostic action space structure derived from bottlenecks. Count matrices reveal underlying action clusters that are transferred to enhance exploration in new tasks. The approach for extracting and transferring structure is novel compared with other work in multi-task RL and prior information transfer. The proposed approach successfully demonstrates the transfer of task and context agnostic action structure to new tasks and significantly improves convergence over baselines.

Secondly, an auxiliary module for proximal policy optimisation (PPO) uses a self-supervised signal from successful state transitions to shape action representations around beneficial relationships. Compared with related work, the relational auxiliary objective is an uncomplicated approach to extracting action structure from multi-discrete action spaces online. The shaped representations demonstrate faster adaptation to complex tasks and better generalisation.

Finally, Decision Transformers are adapted through novel multi-token expansions of multi-discrete actions, exposing more mixing opportunities. Comparisons against single token variants reveal consistent gains in the Deadly Corridor scenario within the ViZDoom platform. Further analyses confirm individual actions are actively attended to by the Decision Transformer after multi-tokenisation.

In summary, the contributions in this thesis provide good evidence that mechanisms that expose and exploit relational attributes can enhance sample efficiency and generalisation in multi-discrete action spaces. Multi-tokenisation and auxiliary modules are two particular methods that show promise for further exploration for leveraging structure. Further work remains in validating and interpreting learned relationships, however this research direction appears fruitful.

Table of Contents

List of Tables

List of Figures

List Of Acronyms

AI	Artificial Intelligence	65
API	Application Programming Interface	138
APPO	Asynchronous-PPO	163
BC	Behaviour Cloning	45
BERT	Bidirectional Encoder Representations from Transformers	49
CASC	Concurrent Action Structure using Clustering	73
CoT	Chain of Thought	145
CNN	Convolutional Neural Network	27
DC	Deadly Corridor	156
DL	Deep Learning	27
DQN	Deep Q-Network	4

DRL	Deep Reinforcement Learning	15
DT	Decision Transformer	141
GAE	Generalised Advantage Estimate	35
GAN	Generative Adversarial Network	67
GANs	Generative Adversarial Networks	64
GN	Graph Network	66
GPI	Generalised Policy Improvement	39
GPT	Generative Pre-trained Transformer	4
GPU	Graphics Processing Unit	48
HGS	Health Gathering Supreme	157
ICL	In-Context Learning	145
ICM	Intrinsic Curiosity Module	53

IL Imitation Learning 44

IRL Inverse Reinforcement Learning 45

KL Kullback Leibler divergence 39

LLM Large Language Model 183

LLMs Large Language Models 4

LSTM Long-Short Term Memory 48

MARL Multi-Agent Reinforcement Learning 41

MAT Multiple Action Tokens 168

MCMC Markov Chain Monte Carlo 59

MDP Markov Decision Process 16

ML Machine Learning 27

MLP Multi-layer perceptron 27

M-SAT Multi-State Action Tokenisation 139

MSE mean square error 29

NN Neural Network 27

PCA Principal Component Analysis 94

PE Positional Encoding 167

POMDP Partially Observable Markov Decision Process 17

PPO Proximal Policy Optimisation 4

RL Reinforcement Learning 1

RLHF Reinforcement Learning from human feedback 4

RN Relational Network 65

RNN Recurrent Neural Network 27

RTG return-to-go 141

SARSA	State Action Reward State Action	25
SF	Sample Factory	163
SOTA	State of the Art	49
SVD	Singular Value Decomposition	127
TD	Temporal Difference	23
TRPO	Trust Region Policy Optimisation	33
UVFA	Universal Value Function Approximators	55
VAE	Variational Autoencoders	65

Chapter 1

Introduction

Reinforcement Learning (RL) is a framework for learning how to make decisions while interacting with an environment and receiving a reward feedback signal. It leads to the development of an autonomous agent that is knowledgeable about the specific environment and can make decisions about how to behave in the environment. To make this a little more concrete consider a robotic maid: the environment could take the form of a kitchen with utilities and utensils and the agent will interact with the environment until it is able to cook a meal, wash dishes or achieve some specified goal. The activities are known as tasks and actions are taken to perform each task. A decision is taking an informed action based on a given situation, while working towards completing the task. Feedback in the form of rewards is available from the environment, sometimes frequent but often sparse, for instance washing dishes might provide a reward signal per dish washed but cooking a meal could provide a reward only at the end, once the meal is fully prepared. The RL framework (Sutton and Barto, 1998b) formalises these concepts and provides the basis for developing algorithms to train the aforementioned agents in environments with a multitude of complexities.

Figure 1.1 illustrates the core elements of the RL framework applied to a simple

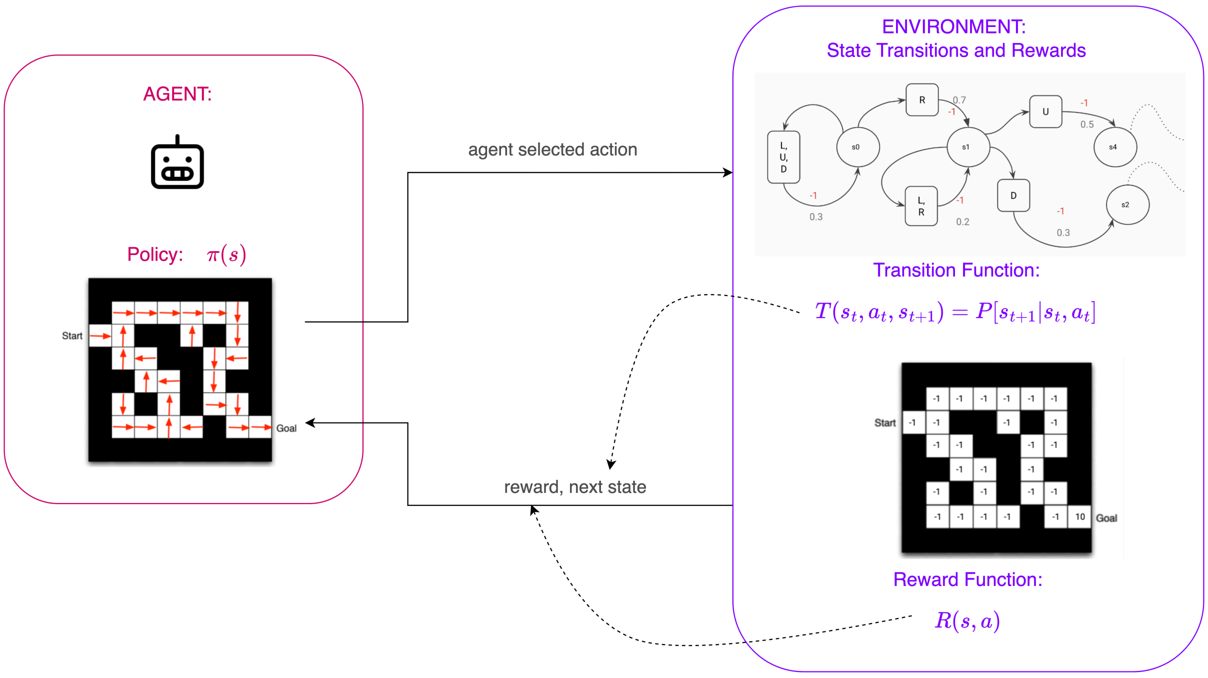


Figure 1.1: An overview of the RL process, with the agent interacting with a maze environment. The environment is characterised by a reward function and a transition function. The agent is characterised by a policy, which can generate the actions [Up, Down, Left, Right]. The current policy's action for each state is displayed on the maze for clarity. Likewise, the reward for each state in the maze is illustrated on the right. Source: (Silver, 2015)

maze environment. In this image, the agent is represented by a policy π and the environment consists of a reward function and a transition function. The agent's policy is trained by interacting with the environment, starting at an initial state or observation. Observations refers to the information the environment provides about the state at each timestep. In this case the observations will be the agent's position in the maze. Observations provide insight into the state, but the state fully describes the environment and may have additional information. In this environment, the states and observations are the same.

In the maze, the start state is clearly defined, and the agent needs to make the first decision, viz. which action to take at that state. The policy is used to make decisions, mapping states to actions, and controls how the agent behaves. The decision or action is sent to the environment which responds with the reward feedback signal and an observation of the new state. Initially, the policy is probably bad at solving the maze, however an RL algorithm will update the policy over time and eventually reach a better policy. The best policy is known as the optimal policy, and the policy shown in the image is optimal for the maze, with the actions for all states directing the agent towards the goal state. In the maze environment the states will be the co-ordinates of the agent, the actions are discrete (where an integer is associated with each individual action) and rewards are scalar and dense, with a reward on every step.

Defining the environment is important and requires domain knowledge to correctly determine the observations, actions and rewards that the agent will encounter and the tasks to solve. Observations could also be images from a video game, images from an MRI scan, joint information from a robotic arm and so on. The reward signal is very important in RL; providing too much information can be misleading but too little information makes the problem harder to solve. The reward signal allows complex behaviour to emerge in the trained agent.

RL is used to develop agents for a variety of environments, however it is probably most famous for achievements in games such as Chess, Go, Atari and StarCraft II. These game environments are complex and can be useful playgrounds for algorithmic development, within which real-world features may be simulated and problems tackled. The potential for RL is still being realised as it is gradually used for problems that are too complex to model, control or programmatically manage. For example a robotic simulation environment used to develop algorithms and train agents can be particularly beneficial if the physical robots are expensive or potentially dangerous. Areas where it has been applied include microchip placement (Mirhoseini et al., 2020), optimising data-center cooling (Luo et al., 2022) and medicine (Raghu et al., 2017; Fatemi et al., 2021; Yu et al., 2019). More recently RL was applied to Large Language Models (LLMs) to incorporate rewards and human feedback into the language generation process, known as Reinforcement Learning from human feedback (RLHF). This has led to exceptional improvements in the performance of LLMs and the proliferation of the Generative Pre-trained Transformer (GPT) (Wikipedia contributors, 2023a) range of models.

Modelling dynamic systems is essentially hard, so any algorithm that encapsulates the elements for modelling, or “knows how to model”, will be useful and RL algorithms fall into this category. These elements mean that, if the environment is well defined, a set of generic RL algorithms can be applied to vastly different environments solving problems that seem on the surface very different from each other. For instance the Deep Q-Network (DQN) algorithm (Mnih et al., 2013) used to solve Atari has also been applied to analysing cancer data and generating medical dosing strategies (Yu et al., 2019). The collection of available generic RL algorithms has become quite extensive, including actor-Critic algorithms such as Proximal Policy Optimisation (PPO) (Schulman et al., 2017), DQN (Mnih et al., 2013) and variants, and Transformer-based algorithms (Agarwal et al., 2023). Likewise the available techniques in RL have expanded to include, amongst others: learning from the most recent interactions versus learning from batched data; learning from interactions with the environment versus learning from experts; incentivising learning using intrinsic rewards or auxiliary signals and optimising exploration techniques.

Any generic framework that can tackle a wide variety of problems is subject to challenges specific to the nature of the framework and RL is no different. The agent needs to interact with the environment to learn how to behave, guided by pre-defined task-reward definitions. A behaviour policy needs to learn what actions are best to take for every state, mapping states to actions. Considering the robotic maid scenario once more, the untrained agent needs to implement an exploration strategy to experience as many states (locations, utilities, etc) as possible and determine which actions are effective in those states (place clothes in washing machine, wash dishes, stack dishes, etc.) Exploration ensures the agent experiences many state-actions and learns knowledge for a more generalised policy. Generalisation is a concept that refers to how well a trained policy performs when tested on new tasks or environments that differ from the training task or environment. For instance different types of plate should not affect the robot maid, because its internal representation of plates is general enough to manage this slight difference.

As RL is reward driven, the agent will receive little reward just for exploration and needs to balance exploring with exploiting the knowledge it has gained, using it's policy to make decisions and gain rewards. Say the house is a hotel with many rooms, similar but in slightly different states of disarray; the size of the state-action space has increased and this directly impacts exploration. The curse of dimensionality (Bellman, 1954) has a high impact in RL where increases in the state and/or action space makes learning harder. Ideally the similarity of tasks should mitigate this increase in problem complexity but it turns out that training an agent to generalise over many different tasks successfully requires a significant amount of diverse interaction with the environment and the assistance of appropriate strategies for exploration and updates. The observation space can be large especially if it is image based, so techniques to reduce dimensions and improve representations are necessary, increasing model size. Larger models can require significant compute time taking days or months of training, all of which makes the attainment of a general RL agent non-trivial.

So RL algorithms are generic and can solve diverse problems. The downside is that applying RL to real world problems is complicated by many factors from the dimension and visibility of the observation space to balancing exploring and exploiting, all the while updating a policy towards the goal of maximising returns. This statement hides many details: efficiently exploring the state-action space is often non-trivial and sometimes task dependent; algorithms require tuning to provide the best performance; reward sparsity can massively complicate the above. Whether the agent has full visibility of the environment or partial can likewise extend the training process. Unless the agent is trained over a diversity of tasks to be as generalisable as possible, the agent is often over-specialised for the environment and/or task such that a change of scenery is enough to make training obsolete. Given this, any optimisations that simplify training are welcome.

This dissertation contributes to the research efforts focused on the action space, in particular the multi-discrete or concurrent action space, proposing methods for leveraging structural information within the action space to improve the efficiency of simultaneous action selection. In the multi-discrete action setting, the agent selects multiple discrete actions per timestep as opposed to a single action per timestep. Multi-discrete actions are commonplace in RL environments, but the action space increases exponentially with the number of concurrent actions and problem complexity increases with the size of the action space. This makes the mapping at the algorithmic level from state to action more complicated, and entangled, which is often managed by not focusing on the individual discrete actions,

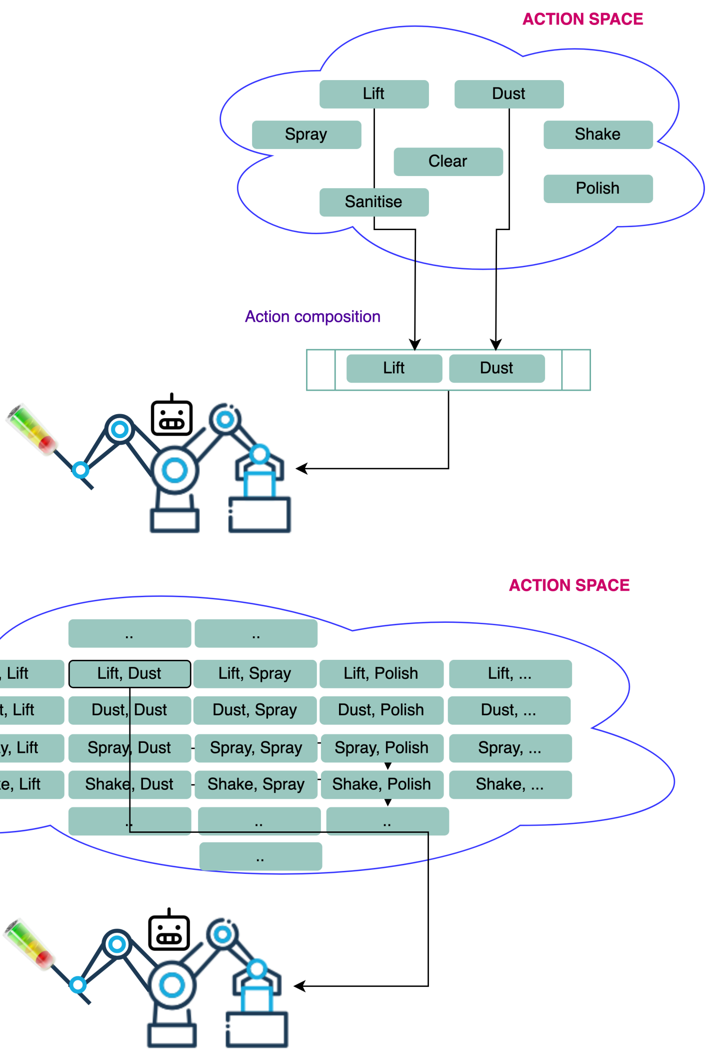


Figure 1.2: Image demonstrating the difference between discrete and multi-discrete actions. In the top image the robot maid composes multi-discrete actions from an action space of individual discrete actions. In the bottom image the robot maid selects a discrete action from an action space of discrete actions. In the bottom image, the action space is expanded to hold all possible actions the robot can take.

but on the combined action. To make the multi-discrete action more concrete, say the robot maid needs to lift a vase off a table to dust. It could move the vase then dust as two separate actions, but being a superior robot maid with multiple limbs it can perform multiple actions at once, in this case lifting with one limb and dusting with the other. As illustrated at the top of Figure 1.2 the robot could compose the action [[Lift], [Dust]] when needed, or alternatively select the single [Lift, Dust] action from the action space as shown at the bottom of Figure 1.2. There are disadvantages to both approaches; for example if the number of individual actions is large, the action space in the bottom image will grow faster, however learning to compose actions is non-trivial for the RL agent.

Conventional approaches to solving problems in multi-discrete action spaces (Kanervisto et al., 2020) tend to treat the multi-discrete action as a single entity and not as individual discrete actions that can interact and have relationships with each other. This is partly as a result of the nature of the most commonly used RL algorithms and the desire to reduce the complexity of training where possible. In the example above, "lift and dust" appears as a single action. The downside of this is if all actions are expanded in this way, the action space is still large, so supplementary techniques are adopted such as eliminating useless combinations (Zahavy et al., 2018) or embeddings (Chandak et al., 2019), primarily to reduce the size of the action space. In this dissertation the benefits of explicitly considering individual actions in multi-discrete settings and exploiting the potential relations between them is explored. This is limited to environments where action relations are possible, such as Minecraft (Fan et al., 2022) and ViZDoom (Kempka et al., 2016). The focus is first to identify and learn relational structure from actions and second to leverage this information in the action selection process. Relational structure refers to how actions interact with each other, for example, synergistic or contradictory actions.

In the real world, actions interact in most multi-discrete environments, from the low level basic coordination of limbs while walking to high level actions like tying shoelaces, a multi-drug treatment plan for patients, twisting the lid on/off a jar or opening a door and walking through it. In the RL world where the agent has to select multiple actions from a pool of actions, determining relationships between actions is a complex task. The task of the agent is already complicated and resource intensive: learning from interactions, balancing exploring and exploiting, updating a policy at the same time as using it to generate future learning data are standard aspects of training agents. Often the focus is on the observation space or the learning signals and the action space is less explored. An approach that can learn relational structure and use this to capitalise on combinations of actions that interact would speed up learning and enhance the agent's capacity to generalise to new objects and tasks.

The specific research questions in this dissertation focus on learning relational structure in multi-discrete action spaces to improve the agent's ability to solve a task, with minimal impact on the RL algorithm. Environments are increasingly complex in nature, with vast state spaces and richly connected action spaces. Multiple actions per timestep increases the size of the action space and an increased space requires more exploration and longer training times. To deal with the increased problem space, complex modellers and large amounts of computing power are used to train agents in these environments, however there are problems associated with using such systems. Models with a large learning capacity need a lot of diverse data to prevent the model from overfitting (Kirk et al., 2023). The model must have access to as much of the problem space during training as possible which is where both task diversity and exploration strategies for the space become crucial.

The review of literature in the area of multi-discrete or concurrent action spaces in RL environments reveals that few works attempt to exploit potential relational structure between the individual discrete actions. The few works that do focus on the multi-discrete action space are limited by the availability of expert data (Harmer et al., 2018; Tennenholtz and Mannor, 2019) or some form of manual manipulation or decomposition of the action space (Sharma et al., 2017; Jain et al., 2022). It turns out that generalisation is a problem in most of these works (Sharma et al., 2017; Tennenholtz and Mannor, 2019; Jain et al., 2022; Li et al., 2023b; Harmer et al., 2018). In general, there is no established approach in RL for extracting and utilising action relationships. While there is some work on applying relational reasoning to complex tasks in RL, most of these works target the state or task space (Zambaldi et al., 2018; Battaglia et al., 2018; Garnelo et al., 2016; Hill et al., 2019) and not the multi-discrete action space. RL algorithms like PPO perform independent action sampling and do not exploit relational reasoning in the multi-discrete action space. Most work in multi-discrete action spaces expands the space to include all combinations of the discrete actions, preferring to manage the growth of the action space. This, however, creates exploration and convergence issues. The most common methods for dealing with exploration include intrinsic signals and count-based bonus methods (Pathak et al., 2017; Tang et al., 2017) that ignore action structure. Multi-task methods that transfer task-agnostic prior knowledge (Rosman and Ramamoorthy, 2012) could be effective in many situations however no research in the multi-discrete action space was found.

Some of the key research challenges that this thesis is aiming to address includes:

- Multi-discrete action spaces are common in RL environments, but algorithms typically do not try to exploit potential relational structure between the individual discrete actions. This misses opportunities for more efficient learning.

- Large action spaces pose exploration and convergence issues, and naively expanding multi-discrete spaces exacerbates these challenges. Action structure could inform more principled exploration.

- There is no established approach in RL for extracting and utilising action relationships in online or offline settings. Prior techniques rely more on manual factorisation or offline expert data.
- Model-free RL algorithms like PPO perform independent action sampling, making it difficult to directly capture action dependencies. The algorithms require modifications to enable relational learning.
- Complex neural network policies in RL are difficult to analyse in terms of what relational concepts are learned and how these aid task performance. Transparency is limited.

The overall aim of this thesis is to develop techniques that enable RL agents to extract and exploit relational attributes in multi-discrete action spaces. Designing and assessing techniques to handle multi-discrete action dependencies, where individual actions can have relations, poses algorithmic, architectural and empirical challenges that this thesis works to address. Both online and offline RL algorithms are considered in the model-free space. Specifically, the objectives are:

- Learn task-agnostic action structure via multi-task training and transfer the relational attributes to improve exploration efficiency in new tasks (Chapter 4).
- Develop an online relational auxiliary module for PPO that reinforces beneficial action relationships during training, speeding up convergence (Chapter 5).
- Apply Decision Transformers, which specialise in detecting relationships in sequential data, to multi-discrete actions by expanding actions into multiple tokens. Assess whether this multi-token approach improves performance and attention to action structure (Chapter 6).

This thesis hypothesises that increasing the visibility of individual actions and providing explicit mechanisms to exploit relational insights will enhance sample efficiency and task performance. Both online and offline model-free methods are explored. The evaluation metrics focus on improved convergence times, attention patterns, and generalisability.

1.1 Thesis organisation

This thesis studies mechanisms for learning and leveraging relational structure in multi-discrete action spaces in an RL context. Three different mechanisms are presented that consider the problem in specific RL settings with the underlying goal of improving the performance of the algorithm by better leveraging action structure.

The thesis is organised as follows:

- Chapter 2 is a summary of RL background information.
- Chapter 3 reviews the literature in the action structure and representation space, building the case for the action structure problem.
- Chapter 4 considers the problem of learning and transferring task-agnostic relational action structure as prior knowledge to improve exploration in the multi-task, multi-discrete action space context. The proposal is to exploit the multi-discrete nature of the action space to better inform exploration using action priors. More concretely, Chapter 4 uses a multi-task setting to demonstrate that task-agnostic relational action structure can be revealed by bottlenecks. Bottlenecks are states that are an essential part of the task solution, viz. a key decision state. Identification of key actions in turn highlights key relationships and the relationships extracted are transferred to future tasks, increasing the exploration efficiency of the agent on the new

tasks significantly and demonstrating an element of generalisation. This approach focuses on context-free actions, not associated with any particular state but generally useful in the environment. This work was accepted for publication and presented at a conference [1].

- Chapter 5 targets the relational action problem in the online RL setting, specifically how to capture action relationships in algorithms like PPO. The proposal in this chapter is to use an auxiliary, self-supervised signal and objective loss to train alongside PPO that contributes towards relationally shaping the action representation. While the approach in Chapter 4 was effective, the method was limited to small environments and relied on a 2-phase approach, namely extensive task generation and pre-training to extract the action structure before it could be transferred and applied to new tasks. In this chapter, a relational auxiliary signal derived from positive transitions was used to shape the agent policy online. Results indicate the shaping provided by this component was effective and achieved earlier convergence than the baseline algorithm.

- Chapter 6 tackles the relational action problem in a multi-discrete offline RL setting, using the Transformer (Vaswani et al., 2017) to exploit action relationships. A multi-token expansion with state associations is proposed for better transparency and interpretability at the multi-discrete action level. This chapter considers using attention models for RL, in particular the Transformer model which has demonstrated proficiency for learning relationships in sequential data. This aligns with the nature of the data generated in RL. This chapter applies the Transformer model to the problem of learning action structure and relationships in multi-discrete action settings, and focuses on techniques for increasing action communication. The key idea is

[1]Perusha Moodley, Benjamin Rosman, and Xia Hong. 2019. Understanding structure of concurrent actions. In International Conference on Innovative Techniques and Applications of Artificial Intelligence. Springer, 78–90

to increase action interaction and information exchange in Decision Transformers (Chen et al., 2021), a variant of the Transformer model adapted for RL. Results show an improvement in overall performance. An analysis of the inner workings of the Decision Transformer is attempted to determine where the performance gains originate using several techniques, including Mechanistic Interpretation (Elhage et al., 2021).

- Chapter 7 concludes with a discussion of findings and future research directions stimulated by this dissertation.

1.2 Summary

Reinforcement learning is a powerful and generic framework for developing autonomous agents that can learn complex behaviour by interacting with environments and receiving feedback signals. Challenges arise, however, in applying RL to real-world problems, including the curse of dimensionality from large state and action spaces, difficulties balancing exploration and exploitation, and issues generalising learned policies across diverse tasks. This thesis focuses on multi-discrete action settings where an agent selects multiple discrete actions per timestep, hypothesising that explicitly modelling relational structure between individual actions can enhance sample efficiency, task performance and generalisability. Three mechanisms are studied across online, offline and multi-task RL settings - spectral clustering for pre-training relational priors, an auxiliary shaping module for online learning, and multi-token actions to improve relational encoding. Evaluation methods assess convergence rates, attention analysis, and transfer learning capabilities. By increasing visibility into action dependencies and providing algorithms with the ability to leverage this relational information, the goal is developing more adaptable RL agents suited to complex, interactive environments.

Chapter 2

Preliminaries

2.1 Introduction

This chapter provides an overview of the theory and frameworks used in this dissertation. It includes a brief overview of the RL framework (section 2.2) and outlines the two most commonly used algorithmic approaches in RL viz. value-based algorithms (section 2.3) and policy-based algorithms (section 2.5), including extensions of these algorithms to Deep Reinforcement Learning (DRL), viz. deep Q-Networks (Mnih et al., 2013) (section 2.4.1) and proximal policy optimisation (Schulman et al., 2017)(section 2.5.1). A brief overview of transfer in RL is included in section 2.6, followed by an introduction to using auxiliary signals (section 2.7), supporting Chapter 5. Sections 2.8 and 2.9 provide some foundations to support Chapter 6. Finally section 2.10 provides an introduction to transformer models (Vaswani et al., 2017) used as the basis for Decision Transformers (Chen et al., 2021).

2.2 The RL framework

The classic RL framework (Sutton and Barto, 1998a; Alekh et al., 2022) describes how an agent interacts with an environment and learns a policy for how to behave

or take actions within the environment. The framework is based on a Markov Decision Process (MDP) (Bellman, 1954) formulation for ∞-horizons with discounting defined by the tuple:
$\mathcal{M} = (S, A, T, R, \gamma)$ where
S is the space of all possible states the agent can take in the environment,
A is set of n primitive actions such that $a_i \in A$,
T is the transition function $T : S \times A \times S \rightarrow [0, 1]$ that defines the dynamics of the environment, i.e. how to move through the environment from state to state. The transition $T(s'|s, a)$ is the probability of moving from a state s to the next state s' after taking action a,
R is the reward function $R : S \times A \rightarrow \mathbb{R}$ such that $R(s, a)$ is the expected reward received when taking action a at state s, and
$\gamma \in [0, 1]$ is a geometric discount factor used in the return over the ∞-horizon.

The ultimate aim of the agent is to learn a behaviour policy, denoted as $\pi : S \times A \rightarrow [0, 1]$ where $\pi(s, a)$ is the probability of selecting an action a while in state s.

The agent begins an episode or trajectory, τ, at the initial state, denoted s_0, selects an action a_0 according to the policy $\pi(s_0)$ and transitions to the next state according to the transition function: $T(s_1|s_0, a_0)$. A reward, r_0 is received from the environment based on the reward function: $R(s_0, a_0)$. The process repeats with the next state, s_1, used to select the next action, a_1, and so on, generating a trajectory with a horizon. The horizon, H is the length of the episode or trajectory. In practice, the RL horizon is finite, however the core RL framework is based on discounted ∞-horizon MDPs, described in more detail by Alekh et al. (2022).

2.2.1 Partially Observed Environments

The RL environments referenced in this dissertation are described as partially observed, i.e. some aspects of the environment are not visible to the agent. In a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) the agent cannot directly observe the underlying state of the environment and instead receives observations. States fully describe the environment, while observations provide partial or noisy information about the state. Most RL algorithms applied to the POMDP rely on approximations and assumptions, such as assuming observations provide enough information to identify the underlying state and learning a compressed representation of the state that is sufficient for policy optimisation.

More common is the use of memory-based architectures. As the current observation does not fully describe the underlying state, the POMDP requires a memory of past observations to find optimal policies. Methods that can process sequences of history or utilise memory are commonly adopted (Ni et al., 2022) to solve such environments in RL. The Decision Transformer is another architecture that can process sequences of historical data to learn state representations based on past observations, for the purpose of guiding the agent in a POMDP. This topic is described further in section 2.10.

2.2.2 Action Spaces

This dissertation focuses on a distinctive type of action, the multi-discrete action, so a brief overview of the most common types of action spaces used in RL follows, including discrete, multi-discrete, continuous, hybrid and parameterised action spaces. This dissertation will only consider discrete and multi-discrete action spaces.

Discrete Actions: The discrete action is arguably the simplest form of action, where each action belongs to a countable set of independent actions $a \in \{0, 1, 2, ..., N-1\}$ and is an integer, with N possible actions to choose from. The integers represent actual actions, for example the agent can select an integer representing the following actions in the environment: $\{0 - \text{Up}, 1 - \text{Down}, 2 - \text{Left}, 3 - \text{Right}, 4 - \text{Attack}\}$. In the discrete action space only one action is selected per timestep and it is represented as a one-hot vector.

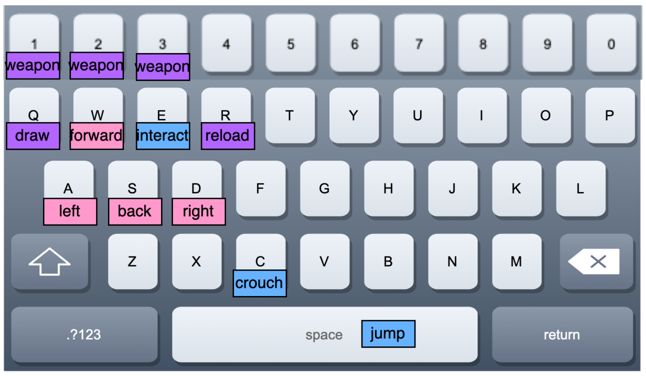


Figure 2.1: Keyboard mappings for an interactive game, demonstrating how multi-discrete actions may be generated. Pink keys are navigation actions, purple keys are weapons related actions and blue keys are other actions. Any number of these keys may be selected together in a timestep, resulting in a multi-discrete action.

Multi-Discrete Actions: The multi-discrete action is a vector of discrete actions: $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n)$ where $\mathbf{a}_i \in \{0, 1, ..., N_i\}$ for $i = 1, 2, ..., n$. In each timestep the agent composes an action by selecting actions from subsets of discrete actions, for example in the multi-discrete environment illustrated in Figure 2.1 an agent could select the multi-discrete action $[A - \text{Left}, W - \text{Forward}, C - \text{Crouch}]$ where:

$\text{Left} \in \{A - \text{Left}, D - \text{Right}\}$, $\text{Forward} \in \{W - \text{Forward}, S - \text{Back}\}$, and

$\text{Crouch} \in \{E - \text{Interact}, C - \text{Crouch}, \mathit{Space} - \text{Jump}\}$.

Multi-discrete actions are managed in various ways; two common approaches are to either model each discrete action separately and use categorical distributions to generate multi-discrete actions or to convert the multi-discrete action into a single discrete action (Kanervisto et al., 2020). The size of the discrete action space is the number of possible unique action combinations that can be taken. As the number of actions that can be taken simultaneously increases, the number of dimensions of the action space increase. Consequently the size of the action space increases exponentially with the number of discrete actions.

Continuous Actions: In continuous action spaces, the action $a \in \mathbb{R}$ is a real number or real vector. This is the most common type of action in the real world, for example actions that represent the speed or angle of movement or any kind of fine-grained control action.

Hybrid actions and Parameterised actions: In real-world scenarios combinations of discrete and continuous actions occur frequently, and are managed in RL in two slightly different formats, namely hybrid actions and parameterised actions. In hybrid action spaces both discrete and continuous components exist and the agent's policy will be required to select both the discrete action and the continuous parameters. In parameterised action spaces, discrete actions have associated continuous parameters, but these are kept distinct from the discrete actions. The agent first selects the discrete action, then the continuous parameters. For more details on hybrid and parameterised actions the reader is referred to Delalleau et al. (2019) and Masson et al. (2016).

Action Relationships RL agents explore both the state space and the action space; the total search space is the product of the two spaces. As either space becomes large it becomes less practical for the agent to explore the whole space. There will, however, normally be relationships between actions, for example:

- actions which to some degree synergise (enhance the effects of each other) e.g. Forward and Run
- actions which conflict with each other e.g. Forward and Backwards
- actions which in some contexts can replace each other e.g. Turn Left and Strafe Left (either can be used to aim at an enemy to the left of center, but only one will look around a corner)

These relationships can be treated as structure within the action space. As described above, as the number of discrete actions taken in a multi-discrete environment increases, the size of the action space can quickly become large. A hypothesis of this thesis is that learning the structure of the action space can help a model explore such a large action space more efficiently.

2.2.3 Model-Free and Model-Based RL

RL algorithms update the behaviour policy from an initial policy to an optimal version of the policy that produces the optimal or best action for a given state, and this is true for all states. There are multiple approaches to determine the optimal policy, and various algorithms within each approach. A brief characterisation of the two main branches, model-free and model-based, is provided below.

The model in the context of model-free or model-based RL refers to the functions that describe the environment, viz. the transition and reward functions. When the model is known, i.e. given the transition and reward functions of an environment, it is possible to determine the optimal policy using, for example, a dynamic programming algorithm (Bellman, 1954). When the model transition and reward functions are unknown, the model-free approach generates data by interacting with the environment which it uses at intervals to update its policy in an iterative process. While the aim of model-free approaches is to learn the behaviour policy, in the model-based approach the aim is to first learn a representation of the model from interactions, then use the model to solve the environment. Model-based RL is not in scope in this dissertation where model-free approaches are used.

2.2.3.1 Model-Free RL

There are a few elements to consider in model-free learning, including data collection for training an agent, the functions that represent the agent and the algorithms used to learn these functions.

Consider an agent that is initialised in an environment, with a task defined as follows: navigate a room and find an object, for a reward of 10 points. The environment is initialised and provides the agent with an initial state s_0. The agent selects an action based on its policy and sends this to the environment, that responds with the next state, s', the reward and the done-flag that indicates if the episode has terminated. This interaction is described by a tuple of data, (s_t, a_t, s_{t+1}, r_t) where $s_{t+1} \equiv s'$ and $s_t \equiv s$, and is known as the trajectory tuple at timestep t. The agent will continue to take actions, generating a trajectory of data points, until the goal is reached, the agent is destroyed or the episode times out. In model-free RL, this interaction is used to derive a policy; when a value function is learnt first, this is known as a value-based method.

In value-based methods the agent learns a value function such that, given any state, the value at that state is an indicator of how good or bad it is to be in that state. The value of a state is defined by Equation 2.1 as the expected discounted return from a given state s, taking actions based on the policy π from that state onwards and summing the discounted rewards:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s, \pi\right] \tag{2.1}$$

The Bellman policy evaluation Equation 2.2 (Bellman, 1954; Russell and Norvig, 2010) expands the expected return into a more useable form, given the reward function $R(s, \pi(s))$ and the transition function $P(s' \mid s, \pi(s))$. The reformulated equation introduces a relationship between the values of the current state s and the next state s' that enables a recursive iterative optimisation algorithm like dynamic programming to update the value function towards an optimal solution.

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P\left(s' \mid s, \pi(s)\right) V^{\pi}\left(s'\right) \tag{2.2}$$

The Bellman Optimality Equation 2.3 (Sutton and Barto, 1998b; Alekh et al., 2022) expands the definition above to find a stationary and deterministic (but not unique) policy that is optimal for all states. Stationarity refers to the dynamics, with the transition and reward functions remaining static over time. The non-stationary condition is more generalised and complex to solve and is out of scope in this document. A deterministic policy will always produce the same actions for a given set of states while a stochastic policy is impacted by uncertainty that results in a potentially different outcome each time. While Bellman policy evaluation is about prediction, i.e. determining the value of policies, Bellman optimality is about control, i.e. finding the optimal policy. Note the $*$ denotes the optimal value function.

$$V^{*}(s) = \max_{a \in A} \left(R(s, a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \left[V^{*}\left(s'\right) \right] \right) \tag{2.3}$$

2.3 Introducing the Q-value function:

The value function in Equation 2.1 may be expanded to be more granular at the action level, viz. the action value function, $Q(s, a)$, defined as the expected value of being at a state s, taking action a and then following the policy π to the end of the trajectory, defined in Equation 2.4.

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r^{t} \mid s_1 = s, a_1 = a, a_{2:\infty} \sim \pi\right] \tag{2.4}$$

This leads to the Q-value form of the optimality Equation 2.5 from which it is easier to derive the optimal policy, Equation 2.6, because it is at the action level and the argmax of the Q-value selects the optimal action.

$$Q^{*}(s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(s,a)}\left[\max_{a' \in A} Q^{*}\left(s', a'\right)\right] \tag{2.5}$$

$$\pi^{*}(s) = \operatorname*{argmax}_{a' \in A} Q^{*}\left(s, a\right) \tag{2.6}$$

2.3.1 Temporal Difference (TD)

From these base definitions a number of classic value-based algorithms emerge, including the commonly used Q-learning algorithm, introduced by Bellman (Bellman, 1954). A key element in value-based model-free RL is the temporal difference concept, where the value function is re-written in the form shown in Equation 2.7. In this equation the value is updated by taking a step, obtaining a reward and adding this to the estimated value of the next state. The reward plus value of the next state, $r + \gamma V\left(s'\right)$, is known as the target and may be viewed as the more up-to-date value of the current state $V\left(s\right)$, having taken a step and received real feedback from the environment. The term $\left(r + \gamma V\left(s'\right) - V\left(s\right)\right)$ is known as the Temporal Difference (TD)-error, the difference between the current value and the most up-to-date value of the state. The TD-error is used to update the current value $V(s)$ towards the target.

$$V\left(s\right) \leftarrow V\left(s\right) + \alpha\left(r + \gamma V\left(s'\right) - V\left(s\right)\right) \tag{2.7}$$

The value estimates are based on taking steps in the environment and making updates to the value function from samples in an iterative process. The agent interacts with the environment, collects a tuple of data (s, a, s', r) and uses the Bellman equations to update the estimated value for the current state, s. The value at the next state, s', is also an estimate, based on the previous iteration. Using an estimate of the value function as a target is referred to as bootstrapping.

2.3.2 Exploration

In model-free value-based methods, the quality of the estimated value function is based on the quality of the data collected. Specifically there needs to be a mechanism for ensuring that as much of the state-action space is experienced as possible to ensure the value function is representative of the space. In RL this process is known as exploration. Exploration is a function of the state-action space that makes it challenging in environments with large discrete state-action spaces or continuous spaces where it is impossible to experience all possible states and actions. Exploration in RL is a significant field of study in itself (Kuleshov and Precup, 2014; Yuan et al., 2018; Tang et al., 2017; Ecoffet et al., 2021) and will be referred to again later with respect to transfer in a multi-task context.

Once the Q-function is a fair representation of the value of the state-action space, it is used to make optimal decisions for acting in this space. Establishing when the Q-function is good enough introduces the so called exploration-exploitation dilemma: while initially the focus should be on exploring as much of the space as possible, over time exploration should be reduced in favour of exploiting the knowledge learnt. A commonly used exploration method is ϵ-greedy exploration (Sutton and Barto, 1998b) where the ϵ is the percentage of exploration applied in the algorithm and $(1-\epsilon)$ is the percentage of exploitation. ϵ is generally initialised to a high value $(90 - 100\%)$ and annealed over the course of training to close to 0, where exploitation takes over almost completely. While ϵ-greedy is a very basic approach to exploration, it is proven to be a very effective method

(Kuleshov and Precup, 2014).

2.3.3 Value Based algorithms

This section describes some of the classic model-free value-based algorithms, including Q-learning, State Action Reward State Action (SARSA) and TD-λ (Sutton and Barto, 1998b). These three algorithms cover a variety of the classic RL problem space: Q-learning is an off-policy algorithm while SARSA is on-policy and TD-λ capitalises on the powerful temporal difference methods. Q-learning is used in this dissertation and is expanded on further below.

The Q-function update equation for SARSA (Equation 2.8) is derived in the same way as the value function update in Equation 2.7, but at the state-action level and therefore is more directed towards control. The target used to update the Q-function is the approximated Q-value of the next state and action, taken from a previous iteration of the Q-function. This is an on-policy algorithm, where the policy used to select actions (behaviour policy) is the same as the policy updated in the Q-function update.

$$Q\left(s,a\right) \leftarrow Q\left(s,a\right) + \alpha\left(r + \gamma Q\left(s',a'\right) - Q\left(s,a\right)\right) \tag{2.8}$$

The Q-learning update function in Equation 2.9 has a similar form to SARSA, with the primary difference being the target that the Q-function is updated towards is the maximum Q-value of the next state. This is equivalent to finding the optimal action from the next state and updating the Q-function towards this optimal value. Q-learning is an off-policy algorithm because the optimal Q-value of the next state is in effect different from the behaviour policy that was used to generate the actions. This algorithm improves the Q-function over each iteration, always heading towards the optimal Q-function, $Q^*(s,a)$.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a \in A} Q(s',a) - Q(s,a) \right) \tag{2.9}$$

When the Q-function is represented by a matrix, this is referred to as a tabular representation, where the rows are states and columns are actions. When the state-action space is very large and tabular methods that calculate the Q-function for the entire state-action space are not feasible, the Q-function is modelled using neural networks or similar function approximators. The next section provides a brief overview of neural networks and deep learning before returning to the Deep Q-Network, an extension of the Q-learning algorithm using neural networks to model the Q-function.

2.4 Neural Networks, Deep Learning and function approximators

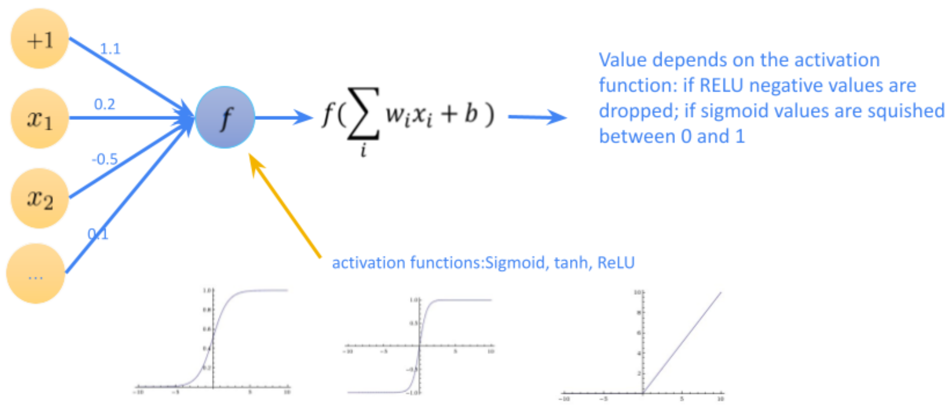

+1
x_1
x_2
...
1.1
0.2
-0.5
0.1
f
$f(\sum_i w_i x_i + b)$
Value depends on the activation function: if RELU negative values are dropped; if sigmoid values are squished between 0 and 1
activation functions:Sigmoid, tanh, ReLU

Figure 2.2: A single neuron, with a linear function passed into one of several possible activation functions (including sigmoid, Tanh and ReLU) is illustrated, where x_i are the inputs to the neuron and w_i and b are the trainable neuron weights.

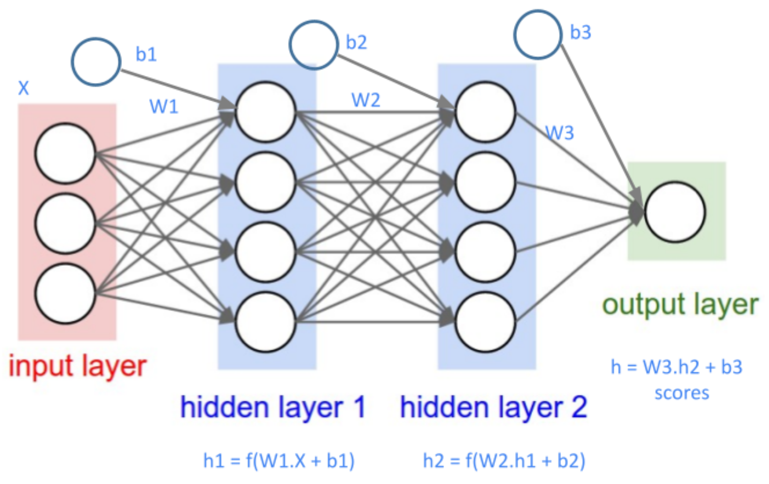


Figure 2.3: A sample neural network architecture with multiple layers of neurons. This is a fully connected network where neurons in one layer are connected with all neurons in the next layer. Neurons in the same layer do not have any associations. The internal layers are referred to as hidden layers, where x is the input to the first layer, h_i is the output from the ith hidden layer and the input to the subsequent layer and W_i and b_i are the weight matrices for each layer.

A Neural Network (NN), depicted in Figure 2.3 is comprised of layers of connected neurons (Figure 2.2), designed for modelling a subset of input data provided from a source distribution. The NN is trained using the Machine Learning (ML) framework, that defines the principles for training and fitting a learning algorithm to a subset of data with the intention of learning sufficient patterns in the data to allow the algorithm to generalise to new, unseen data. Deep Learning (DL) is a branch of ML that trains large volumes of data through deep network architectures, such as Multi-layer perceptron (MLP) (Figure 2.3), Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN). Readers may refer to Bishop (2007) and Goodfellow et al. (2016) for the fundamentals of machine learning and deep learning.

The NN modeller is also referred to as a function approximator because it is effectively fitting a function to the data. The function comprises a linear function

(Figure 2.2), $\mathbf{wx} + \mathbf{b}$, where the input data $\mathbf{x}$ is weighted by trainable parameters $\mathbf{w}$ and a bias parameter $\mathbf{b}$. To model not only linear but non-linear data, the linear function is passed through a non-linear activation function such as a ReLU (Agarap, 2018). This allows the NN to model complex non-linear patterns in data more effectively. During training, batches of data are passed through the network over multiple iterations, and a loss objective calculation is performed on each pass. Using backpropagation (Goodfellow et al., 2016), an algorithm used to calculate the gradients or derivative of the loss function with respect to the weights and biases of the neural network, an error term is fed backwards from the output layer through all the layers. Backpropagation calculates the error contribution of each neuron using the chain rule from calculus (Kreyszig, 2006) and this error signal is used to update the weights and biases in each layer to minimise the total error of the model.

A number of factors influence training and generalisation of DL models, including the depth of the network, the width of the layers (number of neurons per layer) (Cheng et al., 2016), how the weights are shared, the volume and distribution of the input data and a number of configurable hyperparameters. Deep learning is employed for supervised learning, where a learning signal is provided with the training data, unsupervised learning, where there is no learning signal but algorithms attempt to extract patterns from the source data, and reinforcement learning, where a reward signal is provided in an interactive setting. One of the complexities of using neural networks and machine learning in reinforcement learning is the nature of the data. A key requirement in ML is that the data is i.i.d., independent and identically distributed. Independence refers to the random variables, statistical concepts (Bishop, 2007), in input data being independent by nature, which is often not the case in sequential RL. Identically distributed means the training data the model is trained on being generated by an identical distribution, however this too is complicated by the RL process constantly generating and learning from data derived by a new, updated policy (Mnih et al., 2015). The next section 2.4.1 describes the use of NNs in the DQN and how some of these problems are managed.

2.4.1 Deep Q-Network (DQN)

When the state-action space is large enough such that the agent is unable to visit all state-action combinations, value function approximators are used, $V(s;\theta)$ or $Q(s,a;\theta)$, where the function approximator is modelled as a neural network. DRL (Francois-Lavet et al., 2018) uses deep neural networks to represent the value or policy function in large or complex environments.

The DQN algorithm (Mnih et al., 2013, 2015) is an extension of the Fitted-Q iteration algorithm (Sart-Tilman, 2005) adapted for DL (Goodfellow et al., 2016). DQN transforms the Q-learning algorithm to a supervised learning type of algorithm that uses a neural network to represent the Q-function. There are strong reasons for using DL, including the ability to model a state-action space that is very large and to draw on the representational power of DL neural networks (Deng et al., 2022) given raw inputs such as images instead of a pre-processed observation. In tabular models the Q-function is effectively modelled as a matrix, where taking the "argmax" function used in Q-learning over all actions in a state is possible. The tabular model is insufficient to represent a continuous state or action space. The capacity for generalising and modelling more complex representations in a tabular format is quite limited, hence the use of more generic function approximators like neural networks.

The DQN algorithm collects and stores trajectory tuples in a replay buffer. Batches are sampled from the replay buffer and the loss or objective function is calculated for the batch and optimised using stochastic gradient descent. The objective function for the DQN is modelled on the mean square error (MSE) between the predicting function (neural network in this case) and the target. In typical supervised learning processes the target is sourced from an i.i.d dataset. In the DQN the target is not i.i.d., but several modifications were applied to the algorithm to correct for this. The sampling of batches from the replay buffer helps break the correlations between data points that would be found in sequential trajectory data. The DQN target is bootstrapped, i.e. it is also based on the Q-function that is being estimated. To stabilise the DQN algorithm, a separate target network is initialised from the Q-network, then fixed during training for a period of time, after which it is updated from the latest Q-network; this process repeats, ensuring the DQN has a stable target. The DQN dataset will never be identically distributed as the replay buffer is continuously populated from newer versions of the policy and will typically contain data generated by multiple policies at any point in time. There are no guarantees the algorithm will converge but in practice it does perform reasonably well.

The DQN is not directly used in this dissertation but is covered to provide some back-ground for the actor-critic methods related to policy gradient algorithms.

2.5 Policy Gradient algorithms

Policy-based algorithms are viewed as a more direct approach to learn the policy (Sutton and Barto, 1998b; Sutton et al., 1999) than value-based methods, where the objective function for policy gradients is derived directly from maximising the expected reward $R(\tau)$ for a trajectory τ, as defined in Equation 2.10. Trajectories, τ, are sampled from the policy π_θ, parameterised by θ, and $J(\pi_\theta)$ represents the policy performance. The objective is to learn the policy that optimises the performance $J(\pi_\theta)$.

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] \tag{2.10}$$

To optimise $J(\pi_\theta)$ using gradient-based learning to learn the parameters θ requires the derivation of the gradient of the objective function $J(\pi_\theta)$. Taking the gradient of J with respect to θ ($\nabla_\theta J$) leads to a formulation of the policy gradient, derived from Equation 2.10 (Achiam, 2018) analytically to the form

$$\nabla_\theta J = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\nabla_\theta \log P(\tau \mid \theta) R(\tau)\right]$$

where τ, the trajectory, is sampled from the policy π_θ with distribution $P(\tau \mid \theta)$. This formulation is derived by expanding the expectation, using the log-derivative trick and then returning to expectation form. In practice the expectation is estimated using samples generated by interacting with the environment. The more common form of the policy gradient is therefore Equation 2.11. The intuition is the return $R(\tau)$ scores the gradient of the policy.

$$\nabla_\theta J\left(\pi_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) R(\tau)\right] \tag{2.11}$$

Policy gradient algorithms are based on the policy gradient equation derived above. Samples of data are generated by interacting with the environment for an episode and collecting the tuples of data. The return for each timestep is calculated from the rewards. Working backwards from the policy gradient function (equation 2.11), the loss function required to produce this gradient simplifies to Equation 2.12. This is a simpler, practical form of the objective that involves plugging the returns and sampled states into the objective function and allowing the optimiser to manage the back-propagation.

$$J\left(\pi_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \log \pi_\theta\left(a_t \mid s_t\right) R(\tau)\right] \tag{2.12}$$

The REINFORCE algorithm (Sutton et al., 1999) described above is an online algorithm, that collects data and uses it to update the policy whereafter it is discarded. As a result it is less sample efficient than value based methods that use a replay buffer and can potentially train over data more than once. In the format above, full episodes are required to calculate the return and overfitting is a known problem. Several changes were made to the basic policy gradient algorithm to overcome these problems, resulting in the so-called Actor-Critic methods (Konda and Tsitsiklis, 1999).

Actor-Critic methods (Konda and Tsitsiklis, 1999) have both a parameterised policy and value function that are trained from the same data collected from the environment. The value function (critic) estimates the value that is used to guide the gradient in Equation 2.12. It is trained against the return calculation using a mean squared error loss component (equation 2.16). The critic was introduced to reduce the variance that is endemic in pure policy gradient methods like REINFORCE above. An entropy loss component, H in Equation 2.13, addresses the problem of overfitting: if the policy becomes too certain about an action, the entropy of the policy, $H(\pi)$, becomes small and the entropy loss component acts to punish the policy for being too certain.

$$H\left(\pi\left(\cdot \mid s_t\right)\right) = -\sum_{a \in A} \pi\left(a \mid s_t\right) \log \pi\left(a \mid s_t\right) \tag{2.13}$$

The loss function ($L^{PG+VF+S}$) for a typical Actor-Critic model in Equation 2.14 combines all 3 components, viz. the policy gradient (L^{PG}), the value function loss (L^{VF}) and the entropy loss ($H\left[\pi_\theta\right]$).

$$L^{PG+VF+S}(\theta) = \mathbb{E}\left[L^{PG}(\theta) - \alpha L^{VF}(\theta) + \beta H\left[\pi_\theta\right](s)\right] \tag{2.14}$$

where

$$L^{PG}(\theta) = \mathbb{E}\left[\log \pi_\theta\left(a \mid s\right) A\right] \tag{2.15}$$

and

$$L^{VF}(\theta) = \left(V_\theta(s) - V^{target}\right)^2 \tag{2.16}$$

and α and β are co-efficients for each additional loss component. The policy gradient loss in Equation 2.15 uses the advantage, A, defined as $A(s, a) = Q(s, a) - V(s)$. The advantage is a common formulation of the return that removes the state return, V, from the state-action return, Q, providing information on how much better or worse actions are on average from each state. The critic or value function loss in Equation 2.16 is determined by the difference between the estimated and target returns, where the target return is often also an estimate, as described in section 2.4.1. This model is a baseline policy gradient model from which other more complex methods are derived, for example PPO (Schulman et al., 2017), described next.

2.5.1 Proximal Policy Optimization (PPO)

While policy gradient algorithms are very popular and experience reasonable performance in many applications, there is a tendency for these algorithms to suffer from performance collapse (Schulman et al., 2017; Achiam et al., 2017), a situation where the policy worsens over training. In policy gradient methods the latest policy is used to generate data for the next policy update; if the policy resides in an unfavorable region of the policy space, the resulting data and subsequent policy update may also be unfavorable. If this trend continues, the policy does not recover and this leads to what is known as performance collapse. Prevention of performance collapse led to the rise of algorithms such as PPO and Trust Region Policy Optimisation (TRPO) (Schulman et al., 2017).

The PPO algorithm tackles performance collapse by designing the objective function directly on the policy performance, $J(\pi_\theta)$. It turns out that performance collapse is due to the indirect optimization of the policy via the policy parameters. A step in the policy's parameter space can have very different results in the policy space, for instance moving from parameters θ_k to θ_{k+1} has no real control over the change from policy π_{θ_k} to $\pi_{\theta_{k+1}}$. In other words, a step in parameter space could translate to a larger or smaller step in policy space, or a step into a very different part of policy space.

A surrogate objective is derived based on the performance difference between two consecutive policies, $J(\pi') - J(\pi)$, leading to the clip component in Equation 2.17.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2.17}$$

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_{t\geqslant 0} A^{\pi}\left(s_t, a_t\right)\frac{\pi'\left(a_t \mid s_t\right)}{\pi\left(a_t \mid s_t\right)}\right] = J_{\pi}^{CPI}\left(\pi'\right) \tag{2.18}$$

The $r_t(\theta)$ term is the ratio of the new (π') and old (π) policies (as in Equation 2.18), applied as an importance sampling based adjustment of the advantage return, $\hat{A}_t$, where $\hat{A}_t$ is the estimated version of the advantage A. The objective is named J_{π}^{CPI} for conservative policy improvement. The minimization in Equation 2.17 comes from the idea of constraining the distance the new policy moves away from the old policy to a radius of ϵ, i.e. calculate the importance sampling ratio of the new and old policies and clip it if necessary before calculating the clip loss component. The complete PPO loss is a summation of the clip loss component, the value loss L^{VF} and entropy loss S, shown in Equation 2.19. The final form of the PPO loss function is very similar to the Actor-Critic loss in Equation 2.14 except for the first term where the policy gradient is replaced by the surrogate objective, L^{CLIP}.

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right]\left(s_t\right)\right] \tag{2.19}$$

The algorithm (Schulman et al., 2017) is outlined below.

There are effectively two stages: a data collection stage and an update stage.

Algorithm 1: PPO Algorithm (Schulman et al., 2017)

```
For iteration=1, 2,... do
    for actor=1,2,....,N do
        Run policy $\pi_{\theta_{old}}$ in env for T timesteps
        Compute advantage estimates $\hat{A}_1, ...., \hat{A}_T$
    end for
    Optimise surrogate $L$ wrt $\theta$, with K epochs and minibatch
    size M $\leq$ NT
    $\theta_{old} \leftarrow \theta$
end for
```

PPO makes more efficient use of the data collected because unlike the policy gradient algorithm above, it trains over the data for multiple epochs before discarding. The PPO agent collects data from multiple environments running concurrently to ensure decorrelation of data. In a discrete action space, the agent's policy is based on the categorical distribution. In the multi-discrete action space, a list of categorical distributions is held, one for each action in the action tuple. Actions are sampled from the distributions during action selection and submitted to the environment. The trajectories generated are stored in a buffer until the maximum timestep is reached, whereafter the update phase begins.

The update phase samples data from the buffer in batches and calculates the loss components in Equation 2.19. The update of the Critic value function requires the calculation of a return target. Multiple options are available for calculating the target return but the most commonly used is the Generalised Advantage Estimate (GAE) (Schulman et al., 2017) as it was found to be more performant (Andrychowicz et al., 2021). The update phase runs for 1 or more epochs over the full data buffer, updating the actor and critic models. At the end of this phase, the buffer is cleared and the cycle begins again until the maximum step count is reached.

2.6 Transfer

Transfer in RL involves using the knowledge gained from one environment or task to improve learning in a new, related environment or task. Some of the key goals of transfer in RL include improving sample efficiency by solving new tasks faster using prior knowledge instead of learning from scratch, and learning tasks that are too complex to learn from scratch by re-using knowledge. A task in RL is equivalent to the MDP, characterised by the transition dynamics and the reward function. This section defines a few concepts, considerations and methods for using transfer in RL.

Transfer occurs between one or more **source** domains to a **target** domain. What information is transferred, varies, ranging from environmental or domain knowledge (for example transition dynamics), policies on which actions are useful or not, value functions with knowledge of which state-action combinations are good and common features or latent representations that have been shown to be effective (Shelhamer et al., 2016). The key idea in transfer is the extraction and use of generalised knowledge to improve the training speed or performance of a similar task.

If the source policy can simply be transferred to a target domain and used immediately, this is referred to as **zero-shot** transfer. In Zero-shot (Palatucci et al., 2009) the model has not been exposed to this data previously so the transferred model is probably generalised, with latent features that can map new objects encountered. In **one-shot** (Fei-Fei et al., 2006) transfer some training is required but the model requires very few samples and relies on its ability to generalise. By contrast, in **fine tuning**, the model is transferred but the final layers are retrained over more data from the new task.

Some of the problems with transfer are described next, including the need for diverse datasets and the problem of domain shift. The training dataset should be sufficiently diverse to produce models that generalise well so that when the model encounters a new task, it is not so foreign that the model fails to handle it. This is difficult to achieve in reality as environments are large and complex, for example training a robotic agent to perform tasks in a domestic setting, where the average human home has a diverse array of objects and associated tasks. The high volume of data required to train generalisable models often requires the use of simulators for data generation and then model transfer to the real world (Chebotar et al., 2018). This approach introduces the problem of **domain shift** (Higgins et al., 2017) where the simulated source representations differ from real world images and makes transfer problematic. In other scenarios it is the MDP or dynamics of the environment that are different in the source and target tasks. Work on domain adaptation and randomisation focus on these types of problems. Domain adaptation acknowledges the differences in the source and target domains, and learns domain mapping functions or adds reward shaping to manage the differences (Eysenbach et al., 2021; Tzeng et al., 2014). For more approaches for managing these problems in transfer refer to the review by Zhu et al. (2023). This dissertation focuses on a few transfer approaches to support the following chapters, including multi-task transfer, learning from demonstrations and policy distillation.

2.6.1 Multi-Task RL and Meta-RL

Multi-task RL learns policies for a fixed set of tasks by exploiting commonalities and shared structure between the tasks. The goal is to leverage knowledge gained from each task to improve learning on other tasks by exploiting relationships and overlaps between tasks. The training process is a joint training process across multiple tasks in parallel rather than in isolation. This increases sample efficiency, making better use of available data compared with learning each task individually.

Meta-RL is another variant of RL that trains across multiple tasks that share similarities. The purpose in Meta-RL is to learn a generalised model that learns new tasks faster. The training process for Meta-RL is different in a few very specific ways. First the tasks are sampled from a distribution of tasks to train a model that learns how to identify the task from the data. The goal here is to learn how to quickly adapt to new tasks drawn from this task distribution. A second phase, the test phase, evaluates the model on new unseen tasks. The model is expected to perform few-shot generalisation, that is, with only a few samples of a new task it should solve the task.

Meta-RL tackles the problem of learning and transferring structure simultaneously. The agent is trained over a distribution of tasks, $M_i \in M$, from which the training and test tasks are drawn. The model's MDP tuple is $M_i =< S, A, P_i, R_i >$ where P_i and R_i, the transition and reward functions, depend on the sampled task M_i. The policy depends on the previous action and reward from the sampled model in addition to the state:
$\pi_\theta(a_{t-1}, r_{t-1}, s_t) \rightarrow a$ distribution over $\mathcal{A}$. This means the environmental and task structures are built into the policy directly requiring no other transfer mechanism. The need to learn from historical context means recurrent neural networks are often used to model the policy which effectively captures the dynamics in the model. When faced with a completely new task in the same domain the agent is able to select from its pool of task dynamics. Some key related works in Meta-RL include Wang et al. (2016); Finn et al. (2017). Meta-RL is mentioned to clarify the difference with multi-task methods; it is not used in this dissertation and is mentioned only for completeness.

2.6.2 Learning from Demonstrations and Distillation

Learning from demonstrations (Schaal, 1996) is used when the source and target MDPs are closely aligned. Expert or sub-optimal data is provided, from human experts or an optimal/sub-optimal policy, for offline supervised training of a policy or value function or for training a complete agent (Chen et al., 2021). Offline RL, covered in more detail in section 2.8, is a branch of RL that involves learning from data without further interaction with the environment.

By comparison, policy distillation or policy re-use learns from multiple source policies (teacher) and transfers knowledge to a target policy (student). Distillation (Hinton et al., 2015) samples from either the teacher policies or the student policies, using techniques like supervision to push the student features closer to the teachers, or Kullback Leibler divergence (KL) to keep the student and teacher policies close together. In policy re-use, Generalised Policy Improvement (GPI) is applied to improve the policy greedily from the set of teacher policies, resulting in a student policy that is at least as good as the best teacher at every step.

This completes a very brief outline of some of the transfer approaches that will be referred to in this dissertation.

2.7 Auxiliary Tasks

In RL the agent learns from a reward signal, but in many domains and environments there are more signals available for learning. In Jaderberg et al. (2017) auxiliary tasks incorporating other learning signals supplemented learning, improving the speed, performance, stability and efficiency of the RL agent.

The earliest mention of auxiliary-like signals is the “hint” (Suddarth and Kergosien, 1990) injected into a neural network to influence training. Suddarth and Kergosien (1990) noticed that the hint had the effect of flattening local minima but also stressed that the network needed sufficient capacity to be able to encode both the hint and the original data for this technique to be effective. The idea was that the hint provided additional information to the input data that was encoded into the network during training and made the network more effective. At inference the hints were not required because the network had already captured the additional structure and relations.

More recently Jaderberg et al. (2017) demonstrated the use of auxiliary tasks $\mathcal{T}_{aux}$ supporting the main task $\mathcal{T}_{main}$ with both control and feature learning. Specifically, they introduce auxiliary control and auxiliary prediction tasks based on alternative sensory streams to improve learning in complex environments, such as a first-person visual maze with sparse rewards. Predictions were converted into alternative signals, such as reward prediction or end of episode prediction, which could work across many domains more generally. Jaderberg et al. (2017) made innovative use of the visual stream, monitoring and encouraging large changes in pixel maps, or measuring the size of activations in a layer of the network and converting this into a learning signal. The methods adopted are outlined briefly to provide more context on how auxiliary signals can be used to affect control and feature shaping, the latter of which is more commonly used.

Auxiliary control tasks use an alternate reward or pseudo-reward signal in the same environment as the agent to improve learning about the dynamics of the system. Formally, the auxiliary control task $c \in \mathcal{T}_{aux}$ is defined by a reward function $r^c : S\text{x}A \rightarrow \mathbb{R}$. The control task has policy $\pi^{(c)}$ and the task objective is added to the main task, controlled by a task weight λ_c. Equation 2.20 shows the final objective with the main task and the auxiliary control tasks where $R^c_{1:\infty}$ is the discounted return for the auxiliary task.

$$\underset{\theta}{\text{argmax}} \mathbb{E}_{\pi}\left[R_{1:\infty}\right] + \lambda_c \sum_{c \in \mathcal{C}} \mathbb{E}_{\pi_c}\left[R^{(c)}_{1:\infty}\right] \tag{2.20}$$

The task weights λ_c balance the auxiliary tasks with the main task during training. In Jaderberg et al. (2017) two forms of auxiliary control signals were employed: pixel control and feature control. In pixel control the reward signal is derived from pixel changes or changes in the perceptual stream, while feature control monitors changes in network features; specifically it rewards and encourages higher activations in a hidden layer that contains features of interest.

Auxiliary prediction tasks focus on increasing the rewards gained versus improving learning about the system dynamics. Ultimately a good RL agent needs good feature representations (Shelhamer et al., 2016) and the ability to recognise states and actions that lead to high values. Auxiliary prediction tasks help shape the features of the agent and are often unsupervised, using the trajectory data already available. In Jaderberg et al. (2017) a reward predictor is used to predict the reward at the next state when passed a stream of states and reward history. The reward predictor is especially designed to supplement the value function training: the reward predictor is trained over a positively-biased dataset and can inject additional reward information into the value function network without affecting the core RL algorithm (as in auxiliary control). This example shows how an auxiliary task can feed really directed information (positive reward bias in the example above) from the same system to support the core algorithm.

An important part of the agent and auxiliary task design in (Jaderberg et al., 2017) is that the agent network shares at least some parameters with the auxiliary tasks to benefit from the additional signals. This concept of parameter sharing is more clearly illustrated by Hernandez-Leal et al. (2019) in figure 2.4 showing the splitting of the network head into the critic, actor and auxiliary tasks, where the base network is a CNN plus two fully connected layers. Hernandez-Leal et al. (2019) use auxiliary tasks in a Multi-Agent Reinforcement Learning (MARL) context to learn models of the other agents while training the main RL algorithm (a fully online algorithm, A3C). This is known as agent modelling (learning models of other agents in a MARL system) and is useful for informing the agent in cooperative or competitive scenarios. Two different architectures for learning agent modelling are proposed to supplement the main task, which is an online A3C agent, viz.:
AMS - using parameter sharing (Foerster et al., 2016) to learn the opponent or

teammate policy as an auxiliary task together with the actor and critic

AMF: agent modelling learning latent policy features

These architectures, illustrated in Figure 2.5, provides some background on the architectures adopted in this dissertation.

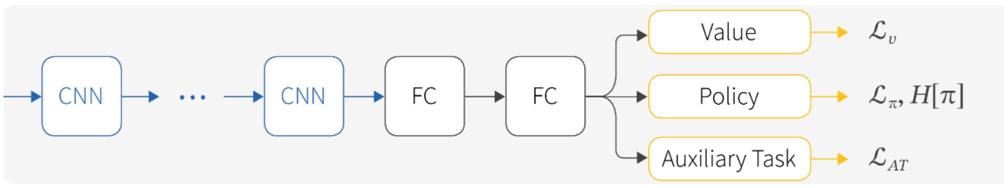


Figure 2.4: Parameter sharing with auxiliary tasks (Hernandez-Leal et al., 2019) Source: (Hernandez-Leal et al.)

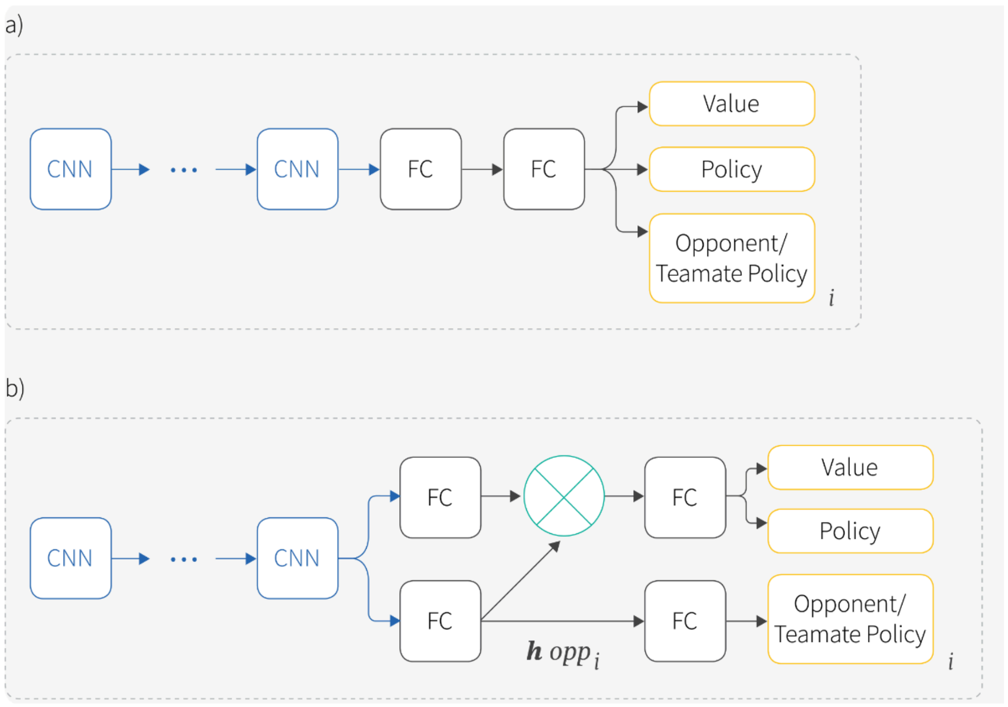


Figure 2.5: Parameter sharing (AMS) and policy features (AMF) architecture in Hernandez-Leal et al. (2019) Source: (Hernandez-Leal et al.)

2.8 Offline RL

Offline RL (Lange et al., 2012), also known as batch RL, is a variant of reinforcement learning where the agent is provided with a static batch of generated

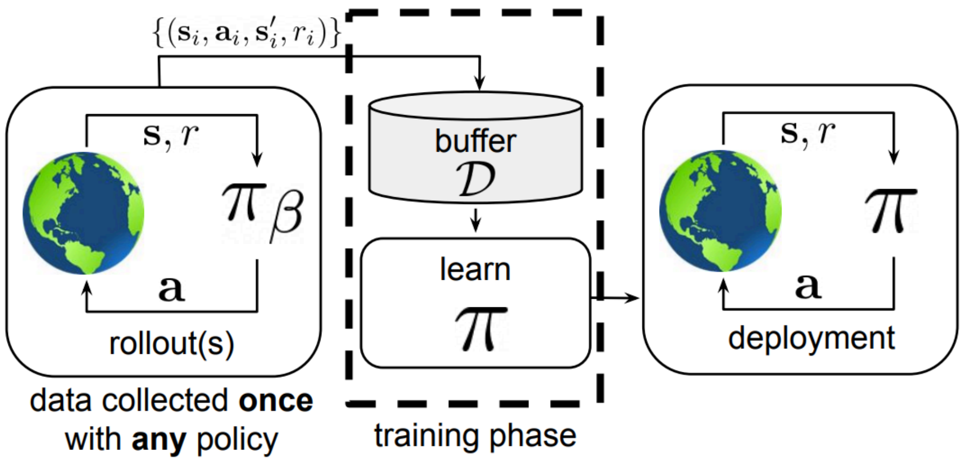


Figure 2.6: In offline RL a dataset $\mathcal{D}$ is collected by an unknown behaviour policy π_β. A policy is trained over the static dataset without further interaction with the environment and is only deployed after being fully trained. Source: (Levine et al., 2020)

trajectory data and must learn the best policy based on this data without any interaction with the environment. Training purely from the dataset means offline RL most resembles supervised learning methods.

To formulate the offline RL problem (refer to Figure 2.6), a dataset of trajectory tuples

$$\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}$$

is provided from an unknown generating or behaviour policy π_β. The trajectory data could be used to calculate a supervised-style Q-value target for training a Q-network, similar to how the DQN is trained (section 2.4.1), where for each datapoint (s, a, s', r) the target y is defined as :

$$y(s,a) = r(s,a) + \mathrm{E}_{a' \sim \pi_{new}}[Q(s', a')]$$

where π_{new} is the new policy derived from the Q-network, and the Q-network loss is

$$\mathcal{L} = \min_Q \mathrm{E}_{(s,a)\sim\pi_\beta(s,a)}[(Q(s,a) - y(s,a))^2]$$

This formulation is based on two policies, viz. the sampled datapoints are from π_β while the target $y(s,a)$ uses π_{new}. The goal is to make π_{new} better than the original behaviour policy π_β. If the two policies are similar, accuracy is good, but Kumar et al. (2019) demonstrated that there are several sources of distribution shift that affect how well offline Q-learning methods perform. Particularly they show that the argmax operation used in Q-learning exacerbates the shift problem when it is used to determine π_{new}.

$$\pi_{new} = \underset{\pi}{\operatorname{argmax}} \, \mathrm{E}_{a \sim \pi(a|s)}[Q(s,a)]$$

a' is sampled from π_{new} for the target calculation and if π_{new} differs from π_β, the argmax makes the target values more extreme.

Various constraining methods (Levine et al., 2020) were developed to counteract the distributional shift problem, including applying the KL-divergence to constrain the policies such that the divergence between the two policies is less than or equal to some small value ϵ,

$$D(\pi_{new}(a|s), \pi_\beta(a|s)) \leq \epsilon$$

which effectively limits how different the new policy can be from the original behaviour policy. This constraint is similar to the policy constraint used in PPO (section 2.5.1) and limits the out-of-distribution actions that are generated by the offline model, keeping it closer to the behaviour policy and trying to contain shift. Refer to Levine et al. (2020) for a comprehensive overview of modern offline methods.

2.9 Imitation Learning (IL)

A brief outline of the core concepts in Imitation Learning (IL) is provided. Imitation learning (Schaal, 1999) emerged from the humanoid robotics field of study, a response to learning complex tasks requiring manual input in the form of rewarding functions, constraints, skills and so on. The premise behind IL is instead of specifying the complexities of training, the agent learns how to behave or perform a task from expert demonstrations, human or machine generated. There are two main branches, Behaviour Cloning (BC) (Bain and Sammut, 1995) and Inverse Reinforcement Learning (IRL) (Russell, 1998).

In BC, supervised learning is used to learn from expert data in the form of state-action pairs called demonstrations. The agent is trained to learn how to map from states to actions and mimic the expert, without reference to rewards. When fully trained on the data, the agent should be able to generate actions even for new, previously unseen states. A common problem in BC is that the actions generated may lead to out-of-distribution states and the distribution shift problem described in section 2.8 is experienced here too. Various constraint techniques are applied to contain the distribution shift, such as using the KL-divergence over the policies. A number of methods are based on the DAgger paper (Guo et al., 2014) that addresses the problem of shift by using an interactive process that alternates between training the policy on the currently aggregated dataset and rolling out that policy to collect more state-action samples.

In contrast to BC, that learns to map from states to actions, IRL (Russell, 1998) uses expert demonstrations to learn a reward function, employing an iterative process that alternatively learns, then evaluates a reward function. This is an interactive process with the agent communicating with the environment, receiving feedback and making updates. IRL methods are potentially more complex than BC because reward function recovery from data, especially in sparsely rewarded environments, is non-trivial.

Refer to the surveys of Zare et al. (2023); Zheng et al. (2022) for more details of imitation learning methods.

2.10 Transformers

To conclude this chapter, a brief introduction to Transformer models is provided to support Decision Transformers (Chen et al., 2021) referenced in Chapter 6, an example of an offline RL method that is competitive with BC.

The Transformer architecture was introduced by Vaswani et al. (2017) to process sequential data such as text or time series data, used either for modelling and understanding the data, or for the purpose of generating the next item in the sequence. The transformer parses fairly long bodies of text for translation, summarisation or to generate text that is similar in context. It is composed of multiple layers of multi-headed attention and is capable of both self-attention and causal-attention. These concepts are expanded upon below in more detail.

The original Transformer in Vaswani et al. (2017) has an encoder-decoder architecture. The encoder and decoder both receive batched, parallelised data but apply a different type of attention to the data. The encoder model uses what is known as self-attention due to having visibility over the entire context. The context is a fixed length window of input word tokens (in the case of language modelling) and the attention mechanism of the encoder can see and attend to all words in the context, hence the term self-attention. The decoder employs a slight modification to self-attention, known as causal attention, over the same data. In causal attention, each word in the context is revealed one token at a time, using a mask, and the attention head is unable to see words ahead of the current word (future words), only preceding words (the past). The decoder is trained to predict the next word given only the history, forcing it to learn how to generate text token by token, which is a core aspect of the generative ability of the Transformer. The model is auto-regressive because it uses previously generated tokens in the context when generating new tokens.

The attention mechanism is designed to learn relationships between tokens, i.e. encoded sequential data, in a context. When a batch of contexts passes through the attention layer, an attention score is calculated for each token that indicates how much it attends to every other token in the context. The attention mechanism has multiple layers with multiple heads, where each head has its own set of query, key and value weights, W_Q, W_K, W_V respectively, that are shaped during training. Each layer has multiple heads so that different heads can specialise in different aspects of the data and task. Head specialisation means that each head potentially extracts different information pertaining to the task, which is fed to the heads in the next layer for further processing in a recursive manner. This is important because it effectively disentangles task-related information into composable, reuseable elements, a desirable feature for foundation models (Yang et al., 2023). A batch of input data is first passed through the weight matrices (W_Q, W_K, W_V) to generate the Q, K and V matrices. The attention score is derived from the dot product of the Q and K matrices, matching query information in the input data with key information already learned by the model. The softmax of the attention score in the Transformer model is displayed in the attention Equation 2.21 from Vaswani et al. (2017)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.21}$$

The core idea is that an attention score is indicative of how strongly the current token is connected to, or influenced by, other tokens in the context. If the score is high, this implies a relationship exists that should be used to influence the outcome, for example, generating the next word in a sentence. The attention score is used for this purpose, to weight the value matrix and influences how much of the value of a token should be used when completing the task.

Each layer has layer normalisation and an MLP before rejoining the backbone, also referred to as the residual stream. The Transformer is supported by a backbone of residual connections that link multiple attention layers to an embedding layer at the start and an unembedding layer at the end. A simplified version of the core components of the Transformer, highlighting the residual stream backbone is in Figure 2.7. It turns out that the multiple layers of multi-attention heads in a Transformer, together with the MLPs, form circuits (Elhage et al., 2021) that achieve complex behaviours via communication both between tokens and between layers using the residual stream.

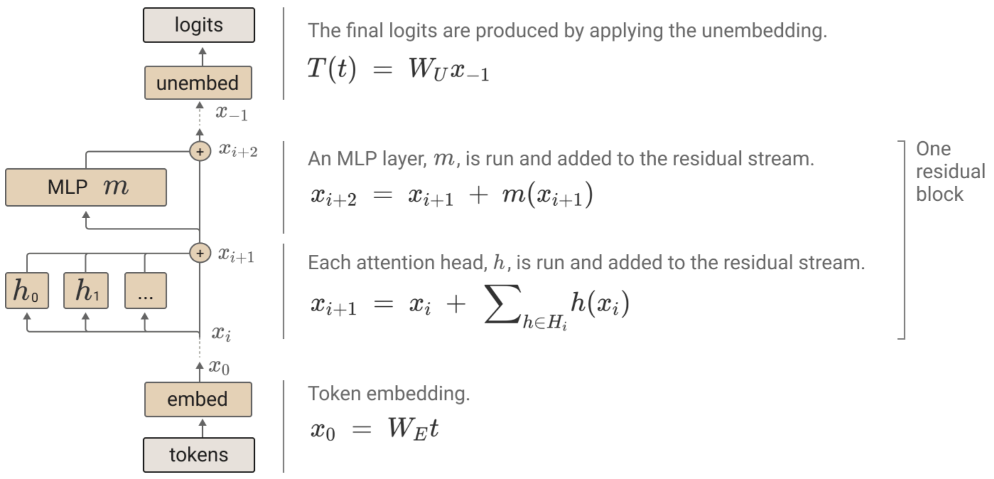

logits
unembed
x_{-1}
x_{i+2}
MLP m
x_{i+1}
h_0
h_1
...
x_i
x_0
embed
tokens
The final logits are produced by applying the unembedding.
$T(t) = W_U x_{-1}$
An MLP layer, m, is run and added to the residual stream.
$x_{i+2} = x_{i+1} + m(x_{i+1})$
Each attention head, h, is run and added to the residual stream.
$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$
Token embedding.
$x_0 = W_E t$
One residual block

Figure 2.7: A simplified 1-layer Transformer, displaying the residual stream as the backbone of the Transformer architecture. Source (Elhage et al., 2021)

The Transformer effectively replaced the Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) family of sequential text processing models (language modelling) for several reasons, one of which is better parallelisation but also because of the significant improvement it contributed to translation (Devlin et al., 2019) and text generation (Brown et al., 2020), leading to the so called family of LLMs. Transformers differ from LSTMs in several ways: the LSTM receives data (for example words in a sentence) sequentially, while the Transformer has parallelised this process, adopting a word position encoder to retain the sequential positioning of words instead. The parallelisation of the Transformer model results in better performance on Graphics Processing Unit (GPU) (Vaswani et al., 2017; Popel and Bojar, 2018) that benefit from parallelised operations in particular.

The Transformer of Vaswani et al. (2017) is both an encoder and decoder model, with the attention learnt by the encoder accessible to the decoder as cross-attention. The most popular LLMs are decoder-only, the GPT (Brown et al., 2020) models, while the BERT-style (Devlin et al., 2019) models are encoder only. While Bidirectional Encoder Representations from Transformers (BERT) models excel at various language understanding tasks (for example semantic understanding, sentiment, etc.), GPT models are more commonly used for generation and are of interest in this dissertation. Generative models such as the GPT models require prompting, a term that refers to the provision of some context for the generation process, for example a prompt could be a recipe request with some details of desired ingredients.

Finally, a few of the key aspects of Transformer architecture and design responsible for the leap in performance in language modelling are specifically mentioned below:

- the attention mechanism (Bahdanau et al., 2014; Vaswani et al., 2017) gives the Transformer visibility over the whole context in encoders and historical context in decoders. Crucially the attention mechanism allows for relationships between tokens to be derived using information passing between tokens in a context.

- causal attention in the decoder is crucial for the generative aspect of the Transformer model. Language modelling has seen two main variants of the Transformer emerge, viz. the BERT-style models (Devlin et al., 2019) that are encoder-only and the GPT-style models (Brown et al., 2020) that are decoder only. The latter employs the causal mechanism which limits access to future tokens, enabling visibility of historical tokens only. The decoder is trained to predict the next word in the sequence, the basis of the generative capabilities of the GPT-style models.

- parallelisation of Transformer data. Previous State of the Art (SOTA) language models were based on variants of the LSTM (Hochreiter and Schmidhuber, 1997) architecture, where the sequential nature of the data limited the use of GPUs for training. The Transformer replaced the sequential training data with batches of position encoded data, significantly improving the parallelisability of training and the use of GPUs (Vaswani et al., 2017).

- the residual stream, that forms the backbone of the Transformer models where residual connections (He et al., 2016) allow complex interactions between layers of the Transformer. The residual stream is said to contain a number of subspaces (Elhage et al., 2021) and token information is copied from the stream to the attention heads and back again, sharing specific information about tokens. The residual connection is also associated with vanishing gradient problems, that are common in very deep networks (He et al., 2016).

2.11 Summary

This chapter reviews the background topics that support this dissertation. It begins by outlining the definitions, principles and algorithms of the RL framework including value and policy based approaches to solving the RL problem. An overview of the different types of action spaces is provided, with a focus on the multi-discrete action space studied more closely in this dissertation. A very brief overview of neural networks is followed by sections on DQN and PPO, both of which support neural network function approximators. Transfer in RL, especially in multi-task contexts is outlined to support Chapter 4. Fundamental concepts for auxiliary tasks are defined to support Chapter 5. Finally a very brief overview of offline RL, imitation learning and Transformers closes this chapter, in support of Chapter 6.

Chapter 3

Literature Review

The key research problems addressed in this thesis are related to learning and leveraging potential relational structure between individual actions in multi-discrete action spaces in RL environments. This chapter covers some of the literature that motivates the work in this dissertation. The first area (section 3.1) reviews transfer in RL, specifically work that explores how to extract and transfer prior knowledge, methods for improving exploration in new tasks and transfer in the multi-task context. The second area (section 3.2) reviews related research for learning and exploiting structure in RL including the factorisation and decomposition of state-action spaces and the use of options and macro-actions as composite action structures. The section ends with a review of work in the multi-discrete action space with a focus on structure, a topic of specific interest in this dissertation. The final section (section 3.3) reviews work in the area of relational reasoning in RL. The area of relational reasoning is vast so only a subset of topics are reviewed, including multimodal data, disentanglement of learned concepts for composition, architectures for reasoning including attention-based architectures and auxiliary signals.

3.1 Transfer

3.1.1 Transfer of prior knowledge

A key idea for accelerating RL is by reusing prior knowledge rather than exploring and learning from scratch. As environments grow in complexity and size several works have looked at ways of supplementing the knowledge or ability of the agent in the form of prior knowledge, sometimes applied during exploration or as structure added to the model (Rosman and Ramamoorthy, 2012; Wingate et al., 2011; Sherstov and Stone, 2005).

Rosman and Ramamoorthy (2015) proposed learning a model that outputs action priors, a weighting over possible actions for a given state that encodes useful knowledge about actions that are used in new tasks to bias the agent's exploration toward more promising actions. Leveraging these learned priors accelerates learning on the new task compared to unbiased exploration or random action selections. Typically these types of methods will be limited to environments where the source and target tasks have a level of similarity and a diverse range of tasks is needed to generate effective priors.

In all methods that transfer prior knowledge, negative transfer is possible and safeguards are required to prevent this. Sufficient diversity in the source tasks is crucial.

3.1.2 Exploration Methods

Exploration is core to the RL process and its efficiency is linked to changes in the nature and size of the state or action space. Sparsely rewarded settings make the task of exploration more important because RL is traditionally reliant on reward signals. The works reviewed below cover some of the more common approaches to improving exploration including intrinsic motivation and count-based exploration. Note this is not an exhaustive review of exploration in RL and the reader is directed to Amin et al. (2021) and Ladosz et al. (2022a) for a more comprehensive treatment of this topic.

The complexity of a task determines how intelligent exploration strategies should be; random exploration is sufficient for a simple task in a small state space, while a complex task requires better exploration strategies for fast and efficient coverage of the space. A survey of exploration in RL conducted by Amin et al. (2021) roughly categorises exploration into two groups: un-directed and directed exploration. Un-directed exploration is uninformed, not using exploration-specific information and includes random exploration; this is effective in limited settings such as small environments. Most work in exploration targets exploration in complex environments or tasks, where direct or informed exploration is more useful, using exploration-specific information to assist the agent.

The process of exploration entails selecting an action to perform at a given state. Random exploration selects the action at random and does not use information from the environment to do so. A common exploration method used in value-based algorithms is $\epsilon-$greedy exploration (Sutton, 1995), described in section 2.3.2. In this method, the exploration generates an action randomly and is balanced with exploitation, that selects an action using the current value or policy function. Intrinsic reward methods are similar to $\epsilon-$greedy as these methods do not use the extrinsic reward from the environment to select an action either. However, unlike $\epsilon-$greedy exploration, intrinsic methods use other information from the environment (i.e. directed exploration) to encourage exploration of the state-action space or improve learning when rewards are sparse. Pathak et al. (2017) propose the Intrinsic Curiosity Module (ICM) for generating an intrinsic learning signal wholly disconnected from the extrinsic reward in the environment. Instead the prediction of the state features is used to generate an error signal that helps the agent explore the space in a controlled way, by predicting something and using that prediction error as a reward to indicate something is novel. The

ICM is an example of an exploration bonus method, described next, that is self-limiting because once the novelty (prediction error signal) dies down, so does the extra signal (intrinsic reward). The method generalises well and exhibits faster learning in many environments but performance is sensitive to how these models are architected and tuned. Sub-optimal models may not produce useful intrinsic reward signals.

Count-based exploration methods (Tang et al., 2017) are also bonus based methods where a bonus signal is derived from the environment to supplement the extrinsic reward. The traditional use of count-based bonuses was to measure state-action visitations and use this as a signal to prompt more exploration of less frequently visited state-action pairs. Bellemare et al. (2016) extended this approach to modern DRL methods, generating count-based exploration bonuses by deriving generalised pseudo-counts from density models, and converting pseudo-counts into intrinsic rewards for exploration. The core idea with these methods is the derivation of a new signal from the environment or task to improve exploration and supplement learning, especially useful when rewards are sparse. While exploration bonus methods are effective, the signal provided is non-stationary as counts change over time. This will impact the learning process and make optimisation harder. Furthermore in very large or noisy state spaces where novelty does not reliably die down there could be stability issues.

Finally, Zahavy et al. (2018) propose an action elimination method that affects exploration by using an additional network to filter out sub-optimal actions. The filtered set of actions is provided to the main RL algorithm, DQN in this case, that explores and selects actions from the filtered set. The Action Elimination Network (AEN) predicts invalid actions using an external elimination signal provided by the environment. The method is designed for large discrete action spaces, however it requires an additional elimination signal from the environment that is generally unavailable.

3.1.3 Multi-Task RL

In multi-task RL (Vithayathil Varghese and Mahmoud, 2020), task-agnostic structure is learnt by training over a diverse range of related tasks to induce task invariance. Representation transfer is based on the idea that there is some invariance in the task-space that can be exploited for transfer, i.e. if the source and target domains disentangle one of the state, action or reward spaces into orthogonal components, there will be a shared set of components between the source and target domains that can be re-used. Representation transfer boils down to the transfer of features, achieved either by directly re-using features or disentanglement of features into a set of shared sub-features. In pure feature re-use, the problem relates to task association: some features will be associated more strongly with some tasks and this association is necessary for transfer. For example Progressive Networks (Rusu et al., 2016) add one task at a time, freezing existing layers then adding and training a new layer for the new task. PathNet (Fernando et al., 2017) uses a fixed network rather than adding layers for each task, but the weights are divided into subsets at the task level so there is still a task association.

In disentanglement (Higgins et al., 2018), the key idea is to break down one of the main functions (value, policy or reward) into two parts, one that will change with every task and one that will not. If the static part is transferred, it reduces the time needed to learn the task-specific part. Successor representations (Barreto et al., 2017) and Universal Value Function Approximators (UVFA) (Schaul et al., 2015b) are examples of these approaches. The successor representation disentangles the state occupancy and the reward function. This method assumes that the dynamics are the same across the source and target domains but the reward functions differ.

The concept of invariance is also used in task-agnostic RL, that learns structure from tasks that can be applied to downstream tasks, reducing the training time for new unseen tasks. There is an underlying assumption that there is common or related structure between tasks that may be exploited in new tasks, much like the invariant representation learning above. A key part of this approach relies on training over a range of related tasks and generating a diversity of data that results in inducing task invariance. In these methods, a task variable is used to provide context for the state; the policy conditions on both the state and the task.

3.2 Structure in Action Spaces

This section provides a review of methods that specifically consider leveraging structure in RL including factors and decomposition, abstractions such as options and embeddings followed by work on multi-discrete action spaces.

The concept of "structure" as it pertains to RL and action spaces requires further clarification. As per the Oxford English Dictionary (University, 2016), structure is formally defined as "the arrangement of and relations between the parts or elements of something complex". The structure of interest in the context of this dissertation is the inter-relations between individual discrete actions in the multi-discrete action space of an agent. Structure between actions is quite intuitive, for instance, moving a jointed object towards some goal often requires co-ordinated movements across multiple joined limbs where the choice of action for one joint dynamically influences the subsequent action of another joint. The relationship between actions can be important in some environments however this is often not taken into account for various reasons. Nevertheless, recent works have proposed mechanisms to incorporate action structure, in the form of abstractions, directly into the learning process. Transfer, extracting and using prior knowledge, can be very domain specific necessitating the use of very different approaches to abstractions at various levels within the RL framework (Thrun and Schwartz, 1995; Pazis and Parr, 2011; Precup, 2000; Singh, 1992; Dean et al., 1998; Wang and Yu, 2016). These different approaches are outlined below.

3.2.1 Structure from Factorization and Decomposition

Methods for factoring state-actions spaces were adopted for managing large spaces and exposing more meaningful representations.

The process of reformulating the large state and action space MDP into a reduced, factored form is known as model minimisation (Dean et al., 1998). This entails extracting from the original MDP, M, an MDP M', whose states and actions are factored sets of states and actions such that the optimal value function V^* of M' may be interpreted as the optimal value function of M. The smallest M' is sought in the process of model minimisation. Factored state and action space MDPs are used to construct minimal models. In the factored model, the state space Q is divided into J state variables, $\{X_i\}$ such that

$$Q = \prod_{i=1}^{J} \Omega_{X_i} \quad \vec{X} = (X_1, \ldots, X_J)$$

Likewise the action space A is represented by K parameters, $\{U_i\}$ such that

$$A = \prod_{i=1}^{K} \Omega_{U_i} \quad \vec{U} = (U_1, \ldots, U_K)$$

where Ω_{U_i} and Ω_{X_i} are the sets of possible values or decompositions for U and X respectively. This forms the basis for factoring a large state and action space. Once the factored representation is available the value iteration process is applied to the new representation and the minimised MDP, i.e. the equivalent MDP. The key idea is to partition the state and action spaces, i.e. create factors, such that the transition behaviour of the factors matches the transition behaviour of the primitive states and actions. Model minimisation works because the transition function of the minimised factored MDP is consistent with the transition function of the original MDP, implying similar states are grouped together and similar actions are grouped together to achieve a similar outcome. Minimisation reduces the exploration space and the amount of time the agent needs to explore. Factoring or partitioning approaches rely on actions within the set having some level of similarity. Early work on factored action spaces (Dean et al., 1998) extends previous work on online model minimisation algorithms and MDPs that deal with large state spaces.

Pazis and Parr (2011) look into the evaluation of factored action spaces, extending the concept of the value function in the RL framework to a more generic form. They also demonstrate how to build a structure into the value function that facilitates faster action selection. The premise is that selection in large action spaces is limited by the value function representation. Pazis and Parr (2011) observe that $V(s)$ and $Q(s,a)$, the traditionally used value function representations, are but two out of a distribution of possible representations. $V(s)$ holds the value of **all actions** for a state s whereas $Q(s,a)$ holds a value at state s for **every single action** a. If the action space is partitioned into sets and the value at each state is the value for the set, then a distribution of value function representations is possible as the number of sets and actions in a set vary. A new value function representation is suggested, based on a hypercube mechanism for efficient storage and retrieval of information and the problem is framed as an optimisation problem soluble using linear programming. In this form the value function is optimised subject to Bellman constraints. Approximate linear programming is invoked. The size of the action space means a high number of constraints so constraint sampling is used and the samples are generated via an RL algorithm. The use of the hypercube as a structure for efficient storage and fast retrieval of actions is interesting especially as it allows for parallelisation of processing. This method of adding structural placeholders into the value function is adopted in several other works, some of which are described below. Although this is a batch-based approach rather than online, there are several novel ideas in this paper demonstrating how to build structures into the RL framework.

Similarly, Schaul et al. (2015a) build structure into the value function, adding a more generic placeholder. They use the value function to extract structure from goals, suggesting that goals are structure-rich and should be included in value functions in addition to the state. The UVFA is a value function approximator that generalises over the goal in addition to the state: $V(s, g, \theta)$. This addition increases the size of the space so a few different architectures are considered to manage this, including a two stream factorised embedding structure where the state and goal matrix is factorised into a lower dimensional space. Using a four room environment they show how the resulting structure learnt by the embeddings adequately map to the environmental structure. This is a multi-goal method (as opposed to multi-task) and is designed for use within an MDP for purposes such as the extraction of options Precup (2000) or predictive representation. Nevertheless it is an extension of the value function that makes it a more useful and composite unit that can be used in other contexts.

Factoring a state or action space is also used for task decomposition and skill discovery. Some of the works below focus on the action space in particular where implicit and explicit relational structure in the action space emerges.

Sallans and Hinton (2004) consider a factored representation to deal with large state or action spaces, using a Product of Experts (PoE) to represent the value function approximation and uncover any implicit structure in the space. They use Markov Chain Monte Carlo (MCMC) (Robert and Casella, 2011) sampling to sample action values relying on the MCMC to converge to the optimal representation of the value function. Sallans and Hinton (2004) test against two tasks, a task with a large number of action and a co-operative multi-agent game called

Blocker which is a concurrent action scenario. The PoE performs well by learning hidden representations that appear like macro-actions when analysed and seem to automatically eliminate less useful actions. The MCMC algorithms selected were designed for finding relevant regions in large spaces.

The works below are also factor or decomposition based but have a relational or compositional component added. Wang and Yu (2016) decompose an action into sub-action components and focus specifically on learning the relationships between sub-actions. They build a novel structure to model the parameters of the sub-action components in a maximum a posteriori setting to induce relations between sub-actions. Similarly Sharma et al. (2017) first factor the action space manually, then change the representations for the policy (in A3C) or value function (for DQN) to be based on the corresponding factors. The agent learns how to compose concurrent actions or compositions of actions based on these pre-determined factors. Harmer et al. (2018) propose a network architecture that outputs multiple actions per timestep in a deep RL setting. The network is trained using auxiliary signals from experts resulting in an online approach and is one of the few on-line models in the area of action structure. The agent benefits from having a concurrent action structure as it can model and learn from experts without restriction. While this approach requires expert input, it is very effective. Two interesting features of this work include building the capacity for action concurrency into the model structure and the use of an auxiliary expert signal. The availability of an expert signal will be a limitation in most environments.

Most of the factored action space work benefits from a reduced exploration space. The next two papers learn lower dimensional embeddings to extract structure instead of factoring the space. Tennenholtz and Mannor (2019) developed a context-based action embedding representation, Act2Vec, modelled on word embeddings using skip-grams (Mikolov et al., 2013). The action embeddings, once trained, were used to enhance Q-function learning and to cluster similar actions together, thereby reducing the exploration space. The embeddings were generated from optimal demonstrator trajectory data. The paper used this representation and a new measure of similarity to consider a broad spectrum of analysis that included concurrent actions and exploration across clusters. Chandak et al. (2019) learnt a lower dimensional action representation, and a transformation function between the representation policy and original policy. The embedding is used for training the agent in the lower dimensional space and makes use of underlying structure, similar to Tennenholtz and Mannor (2019).

Representing structure in action spaces is useful for skill learning, generalisation and transfer to new tasks and domains. Ganin et al. (2018) show that the agent can learn very effectively from external structure and train an agent to compose images by effectively writing programs for a graphics engine. The graphics engine contained all the structures needed to produce images so the agent architecture did not need to build this in separately. This paper shows that the structure can be external to the system, and that structure learning modules can be used as external support for other systems. For real world problems this may be a very effective solution.

3.2.2 Options

Action abstractions and structure are often analysed at the task, skill or subgoal level. Some of the earliest work in this area includes options Precup (2000), skills Thrun and Schwartz (1995) and macro-actions Randlov (1999). A natural extension of the primitive action is the macro-action or option, which is a multi-step action that (often representing a skill) is re-useable and associated with a sub-task. Macro-actions and options are very useful in hierarchical decision making where the agent can abstract at a higher level and solve more complex, multi-layered tasks. Options are temporal abstractions and are composed of a sequence of actions, the start states for which the option is eligible and a completion probability.

Options may be added to the pool of primitive actions to give the agent the advantage of selecting a skill where appropriate. Option discovery is an on-going area of research Harb et al. (2018); Barreto et al. (2019). Options differ from concurrent actions in that options are generally attached to sub-goals or sub-tasks while this is not necessarily the case with concurrent actions and options are multi-step while concurrent actions are single step.

3.2.3 Multi-Discrete Action Spaces

This section describes work that focuses on the multi-discrete or concurrent action space in particular. Several approaches are adopted when working with concurrent or joint actions including learning intuitive embedding representations (Tennenholtz and Mannor, 2019; Chen et al., 2019), decomposition of joint into factored actions (Sharma et al., 2017), intrinsic motivation as an auxiliary signal (Chitnis et al., 2020) and imitation learning (Harmer et al., 2018).

Sharma et al. (2017) leverage compositional structure in action spaces by learning factored policy representations. Their method decomposes the policy learned by a deep reinforcement learning agent into factors that can be learned in parallel, allowing efficient learning about multiple actions simultaneously. The overall action is then composed from the outputs of the modular factored policies and shows performance improvements in the Atari environment.

Harmer et al. (2018) adopt a multi-pronged method to work with multi-action environments. They combine expert data with environment interactions instead of pre-training on expert data alone. Using mixed batches of expert and real data, a model designed to handle concurrent actions is trained to benefit from knowledge of how humans select actions concurrently. Chen et al. (2019) learn action embeddings from semantics of actions in an outcome-based approach. The multi-task space features different state and action spaces where shared rewards and goals are used to drive similar embeddings across the tasks.

Chitnis et al. (2020) consider concurrent actions in a sparsely rewarded multi-agent, co-operative setting and use intrinsic rewards to encourage synergistic behaviour when selecting actions. The intrinsic reward is based on the deviation between the predicted outcome of the joint action versus the predicted outcome of the combination of the individual actions. The intrinsic motivation reward is used to incentivise agents to take joint actions that have synergistic effects.

Finally, recent work by Li et al. (2023b) introduces a method for solving structured action spaces, defined in the paper as having two properties:

- Composability, where the action space is composite, consisting of multiple primitive action components, such as a sequence or set of primitive actions
- Local dependencies, where there are local correlations between primitive actions

This definition is very similar to the topic of this dissertation. Li et al. (2023b) propose Diverse Policy Optimization (DPO), a method that models policies as energy-based models (EBM) within a probabilistic RL framework. DPO uses a generative model, GFlowNet, for sampling diverse policies in structured action spaces. GFlowNet is used because sampling policies in structured action spaces is complicated by the high dimensionality of such spaces. In the proposed process, primitive actions are generated and composed by GFlowNet until the global action, composed of multiple primitives, is constructed, to match a target policy distribution. This is a creative use of EBM and generative methods in RL.

3.3 Action Relations in Reinforcement Learning

This section reviews work that focuses on learning relationally in RL, beginning with a brief review of relational reasoning, followed by where relational learning has been applied and finally closing with some of the architectures developed for relational approaches.

The motivation for relational learning in RL arises from the complexity of some environments, modelled using games such as Atari (Bellemare et al., 2015), Go (Silver et al., 2017), StarCraft II (Vinyals et al., 2017), MineCraft (Guss et al., 2019), Poker (Brown and Sandholm, 2019) and Diplomacy (, FAIR). As environments approach the complexity of real-world environments, so too do the states, actions and rewards. Visually-based observation spaces are high dimensional by nature requiring large volumes of data and high amounts of compute to train DRL agents. As environments become more realistic, additional sensory data is provided making the observation space more complex and prompting work that focuses on better abstractions. Some types of abstraction were discussed in the previous section 3.2 such as embeddings and temporal abstractions such as options, often used to learn a set of latent features that are relevant for solving the RL task. Another form of abstraction is to identify relational information inherent in the data that is useful for solving complex tasks. This may be multimodal data, combining images, text, video or other inputs (Reed et al., 2022) or in the space of multiple agents (Mathieu et al., 2023) all required to solve related tasks. This topic is vast and, at the time of this dissertation, very topical. A brief summary of work relevant to this thesis is provided.

Relational reasoning is emerging as a fundamental component required for learning in large, complex systems (Battaglia et al., 2018; Garnelo et al., 2016; Hill et al., 2019; Lampinen et al., 2022). There is divided opinion on how best to learn relational abstractions. Some approaches advocate building structure into the architecture (Zambaldi et al., 2018; Battaglia et al., 2018; Shanahan et al., 2020), others into the input (Sanchez-Gonzalez et al., 2018), yet others demonstrate that all structure can be external to the agent (Ganin et al., 2018). Hill et al. (2019) show that careful preparation of training data can be used to learn complex relational reasoning patterns and suggest training Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) to generate data of the nature required.

Andreas et al. (2017) show that a coarse symbolic outline of the relations between tasks is enough to learn components that generalise and improve performance.

Learning compositional representations is complex. While DL is able to take raw input data and produce representations in the intermediate layers, it turns out that different representations are learnt depending on the type of training (Garnelo et al., 2016). Garnelo et al. (2016) find that learning disentangled representations alone are not enough when building compositional systems, and incorporating relational structure during training moulds the representation into components that work compositionally. Garnelo et al. (2016) specifically looked at combining symbolic Artificial Intelligence (AI) and DL. Symbolic AI requires a compositional model with objects and a language for making queries. Object representations are often trained using auto-encoders however when training representations from raw data using an auto-encoder, entangled representations emerge in the intermediate layers. Variational Autoencoders (VAE) (Kingma and Welling, 2014) are able to disentangle representations, but a further problem emerges when using these representations as components in tasks that require reasoning about objects and their relations. It turns out that when composition is required, the representation training process must impose structure so that the representation is primed or tweaked for composition. They consider two similar mechanisms for forcing the learning of relational information: Relational Network (RN) (Santoro et al., 2017) and self-attention (Vaswani et al., 2017). Both of these approaches learn pairwise relations such that if there are n objects, n^2 relations are learnt.

Andreas et al. (2017) look at compositional reasoning models in RL and introduce the concept of policy sketches, a coarse definition or outline of the subtasks in a sequential task. No details of the components of the subtask are provided, just the sequence of the subtask (in symbolic form) in the task, with re-use of subtasks across multiple tasks. This paper demonstrates that simply a high-level outline of the abstract relational structure of a task allows the agent to learn re-usable subtask policies and achieve good performance improvements for sequential tasks in multi-task environments compared with other approaches.

In “Relational Deep Reinforcement Learning”, Zambaldi et al. (2018) build relational reasoning into the architecture of a DRL agent, to allow the agent to reason about relations between entities in a scene using a self-attention mechanism. Zambaldi et al. (2018) apply self-attention to iteratively reason about the relations between entities in a scene, specifically, self-attention is used to update the agent's relational representations between entities to improve its understanding of the relations in the scene. They show that their method works well, inducing relational inductive biases in StarCraft II and allows an element of interpretability, however some limitations were observed, including a tendency to overfit, a lack of generalisability and limitations on the nature of the environment, requiring objects and relations to be clearly detectable. Other neural network architectures in the area of relational structure include the RN (Santoro et al., 2017), interaction network (Battaglia et al., 2016), the Transformer self-attention approach (Vaswani et al., 2017) and graph networks (Sanchez-Gonzalez et al., 2018; Battaglia et al., 2018). Recent work on Graph Network (GN) by Battaglia et al. (2018) was developed to support relational reasoning over graph structures, designed to be added to any learning process as an architecture that can accept a graph. Sanchez-Gonzalez et al. (2018) use GNs to model articulated bodies as a graph of limbs, joints and some global attributes, showing how effective the GN is at learning the relational structure and updating the graph representation. This work illustrates how more complex systems may be modelled in a more intuitive manner by using these architectural components in deep neural networks.

By contrast to using architectures, Hill et al. (2019) find, using a theory from analogical reasoning, viz. Structure Mapping Theory (SMT), that relational abstractions can be learnt from only the input data. Analogies require relational structure and Hill et al. (2019) believe it is a key element to human general intelligence, i.e. abstracting the core relations from an input source and applying the relational abstraction to a completely different set of data. They believe in inducing analogical reasoning from the input data instead of architecture, and introduce a new method of training (LABC) that can be applied to any neural network. A key feature of this work is the careful generation of training data to expose the type of learning desired, providing a new dataset that uses the interplay between relations, domains and values to create many different tasks for this purpose and later train a Generative Adversarial Network (GAN) to automatically generate data for future training. This work provides an alternative to architecting for structure by relying on data instead, and proposes a means for an agent to automatically generate the data it needs to learn the structure, opening up new possibilities for agents to perform more intelligent pre-exploration of the space autonomously.

In “Measuring abstract reasoning in neural networks” Barrett et al. (2018) tackles the question of whether relational modules like the relational NN are actually learning relational reasoning. Experiments based on the Raven grids used for IQ testing are designed and a variant of the RN model from Santoro et al. (2017) is used to learn relations and train an auxiliary model that tests whether valid relations are learnt. Barrett et al. (2018) find that when the agent makes a selection the relation metric is higher than the other metrics, pointing positively to relational reasoning driving action selections.

A common feature in Garnelo et al. (2016) and Barrett et al. (2018) is the use of an auxiliary task for grounding learning. Garnelo et al. (2016) are interested in symbolic representations and notice that training the query with symbols seems to improve the nature of representations learnt. The same observation is made in Barrett et al. (2018), that an auxiliary model trained to predict features that agents focus on in reasoning tasks in a more granular way, produced improved results.

3.4 Summary

A summary of the related work and the gaps identified is outline below.

Transfer in Reinforcement Learning (Section 3.1)

• Transfer of prior knowledge to accelerate learning in new tasks

• Exploration methods for sparse and delayed reward settings

• Multi-task RL for learning common knowledge across tasks

Gaps Identified

• In general, most transfer methods are limited when source and target tasks lack similarity

• Sufficient diversity in source tasks is crucial to avoid negative transfer or overfitting

Structure in Action Spaces (Section 3.2)

• Factorisation and decomposition of state-action spaces

• Abstractions such as options and embeddings

• Approaches tackling multi-discrete action spaces

Gaps Identified

• Factorisation and decomposition rely on actions having similarity within sets and methods are often manual

• Generalisation remains a challenge in most works on multi-discrete action spaces

Action Relations in Reinforcement Learning (Section 3.3)

• Representing and leveraging relationships between actions

• Frameworks combining relational reasoning with DRL

• Architectures for relational learning like attention mechanisms

Gaps Identified

• While there is work on relational reasoning in RL, most target the observation or task space, not the action space. Model-free RL algorithms tend not to exploit relational reasoning in multi-discrete action spaces.

• In general, there is no established approach for extracting and utilising action relationships in online, offline or multi-task RL settings. Expert data or manual manipulation of the action space is often required.

• Architectures for relational learning like attention are promising but tend to overfit, lack generalisation, and have limitations based on the environment. Graph networks are an emerging direction.

In summary, the related work covers key areas related to structure and relationships in RL action spaces and identifies significant remaining gaps, especially in leveraging action relations without expert data, improving generalisation, and developing architectures for more complex environments. Addressing these open problems is a key motivation for the research in this paper.

Chapter 4

Multi-task transfer of action structure in multi-discrete action spaces

4.1 Introduction

One of the challenges in RL is to ultimately obtain enough of a signal to facilitate learning, especially in environments where rewards are sparse (Shelhamer et al., 2016; Ladosz et al., 2022b; Eysenbach et al., 2019; Raileanu and Rocktäschel, 2020; Pathak et al., 2017). An important part of the RL process entails exploring the state-action space efficiently and effectively. The problem addressed in this chapter is to improve the exploration capabilities of an agent in sparsely rewarded environments that have multiple discrete actions per timestep, known as multi-discrete actions. The nature of multi-discrete actions and the types of relationships that can form between individual actions in the multi-discrete action are described in section 2.2.2. Multi-discrete actions increase the size of the exploration space combinatorially. A sparse reward signal further exacerbates the RL learning challenge, that is directly tied to the size of the state-action space

(Sutton and Barto, 1998b; Amin et al., 2021). Random exploration is often used as a starting point when training RL algorithms (Kuleshov and Precup, 2014; Houthooft et al., 2016), however this approach is problematic when rewards are sparse (Pathak et al., 2017; Shelhamer et al., 2016) and learning is dependent on the agent randomly encountering a reward signal.

Reward sparsity (refer to Chapters 2 and 3 for context), is not an unusual problem in RL and much work has already been proposed to tackle this problem (Pathak et al., 2017; Schaul et al., 2015c; Vecerik et al., 2017; Rengarajan et al.). In sparsely rewarded settings learning can be very slow until exploration provides some positive signal. When the extrinsic reward signal is unavailable alternative approaches are adopted to guide the agent, including auxiliary signals (Jaderberg et al., 2017; Shelhamer et al., 2016), intrinsic rewards (Machado et al., 2017; Pathak et al., 2017; Raileanu and Rocktäschel, 2020), providing the agent with additional knowledge about the environment (Goyal et al., 2019; Salimans and Chen, 2018) or task (Zhang and Yang, 2022; Chitnis et al., 2020).

In multi-task RL (section 2.6.1) there is some level of redundancy and invariance across tasks (Du and Narasimhan, 2019; Zhang and Yang, 2022; Zintgraf et al., 2019) that could speed up the learning of future, unseen tasks if exploited. This finding motivates the approach in this chapter that considers learning task-invariant action structure in a multi-task setting and transfers this knowledge to support exploration when training new tasks. The exploration space is normally increased by the multi-discrete action setting, so the invariant structure is used to enhance action selection by reducing the size of the exploration space.

In a synergistic action combination the outcome of the joint action exceeds the sum of the individual action outcomes. In multi-discrete environments where synergistic action combinations are required to transition some states on the optimal trajectory, these states are referred to as bottleneck states. The concept of bottleneck states arises from the skill discovery literature (Bacon; Goyal et al., 2019; Kulkarni et al., 2016), and refers to a state that may be used as a goal state because it is on the critical path. By nature, bottlenecks are states that occur frequently on successful trajectories and less frequently on unsuccessful trajectories, a concept that is exploited in this chapter.

The proposed approach in this chapter is to explicitly learn synergistic relational structure in multi-discrete action spaces in sparsely rewarded environments. This structure is learnt in a task-agnostic way that leverages bottleneck states (Goyal et al., 2019; Kulkarni et al., 2016) and is transferred to new tasks to bias the exploration process. Bottlenecks are used to identify effective action combinations across a diverse range of tasks. The actions linked to bottleneck states are mined for structural information, specifically relational structure or action affinities. By accessing the structural information from the actions rather than the states, the context problem associated with a task is reduced. In each task the specific states and transitions are different, however, context-free action structure is transferable to new tasks. This approach is limited to problems with a static action space that remains consistent across all tasks.

Context-free structure is achieved by first establishing common action structure across multiple tasks using bottleneck states, then clustering the action space to group individual discrete actions that demonstrate affinities. The clusters are used as transferable structure to modify the exploration process, impacting how actions are selected. During exploration for new, unseen tasks, actions are selected from within these clusters.

A four-room grid-world environment with a multi-discrete action space was used to demonstrate the effectiveness of this approach, using a Q-learning RL algorithm. The results were compared with a baseline Q-learning algorithm and an action elimination approach. The empirical results demonstrate that when the clusters were applied to the exploration process of downstream tasks, exploration efficiency improved, enabling the agent to converge much faster on new tasks. For convenience the proposed approach is referred to as Concurrent Action Structure using Clustering (CASC) for the duration of this document.

The specific contributions of this chapter include:

- learning task-agnostic and context-free relational action structure across multiple RL tasks by leveraging bottleneck states
- transferring action structure in the form of clusters to new unseen tasks to improve exploration efficiency
- comparisons of CASC against baseline and action elimination methods over the same set of tasks
- demonstration of faster convergence on new tasks by biasing exploration to use learned clusters

The chapter is organised as follows: section 4.2 provides some related literature, followed by section 4.3 that provides background material on the spectral clustering method used to learn structure; section 4.4 provides the motivation and details for the proposed approach, including the extraction and transfer of action structure to new, unseen tasks and details of the implementation; section 4.5 outlines particular experiments performed and results and finally section 4.6 concludes the chapter.

4.2 Related Work

Chapter 3 provides an overview of literature relating to exploration and reward sparsity (section 3.1.2), multi-task transfer learning (section 3.1) and structure in RL (section 3.2). This section reviews some literature more specific to this chapter, particularly how other approaches manage structure in concurrent action spaces.

Several works use factors or decomposition of the action space in concurrent action environments although the approaches for obtaining the factors and the methods of composition vary across approaches. Wang and Yu (2016) use a regularisation technique to model and exploit the relationships between multiple concurrent actions to improve sample efficiency. The regularisation encourages the values of related actions to be similar and allows the agent to learn relationships between actions instead of treating each action independently. The method requires manual intervention to decompose actions into components and it is uncertain how well this approach would scale to larger action spaces. Sharma et al. (2017) also decompose actions into a set of action-factors forming a factored action representation. The agent is trained to compose concurrent sets of actions based on the factors. As with Wang and Yu (2016), the decomposition is manual. Harmer et al. (2018) adopt the composition approach too and propose a network architecture that outputs multiple actions per timestep in a DRL setting. The network is trained using auxiliary signals from experts resulting in an online approach, one of the few online models in this problem space. The agent benefits from having a concurrent action structure that means it has the capacity to learn relational information from expert data. This approach seems very effective but does require expert data.

Rosman and Ramamoorthy (2015) and Zahavy et al. (2018) look at prior knowledge and action elimination to bias the agent's learning of new behaviour. In Rosman and Ramamoorthy (2015) action priors are modelled using Dirichlet distributions where the concentration parameters are the counts for a task. Zahavy et al. (2018) propose reducing the size of the relevant actions per state using a separate network, viz. a model that is trained with an externally provided action elimination signal, controlling which actions to eliminate in the main DQN agent. The motivation is to remove unnecessary actions and improve sample efficiency.

Tennenholtz and Mannor (2019) and Chandak et al. (2019) focus on structure in action spaces specifically, both learning embedded action representations. Tennenholtz and Mannor (2019) developed an action-context embedding representation, Act2Vec, modelled on word embeddings in the manner of language-based skip-grams models Mikolov et al. (2013). Action embeddings were trained in the first phase, and used to enhance Q-function learning in the second phase. Actions were clustered, by similarity of outcome, to reduce the exploration space and improve performance. To learn the embeddings, expert trajectory data was provided. Chandak et al. (2019) also learn a lower dimensional action representation and a transform function between the lower dimensional representation policy and original policy. The embedding is used for training the agent in the lower dimensional space and makes use of underlying structure, similar to Tennenholtz and Mannor (2019). The choice of action structure is left to the implementer and the transformation function from embedding to real actions makes an assumption that is related to the convergence guarantees. In practice, this method was tricky to implement successfully.

4.3 Spectral Clustering

This section provides a brief overview of spectral clustering to support the algorithm used in this chapter (Ng et al., 2002). Spectral clustering is a non-generative clustering method based on the eigenvectors of a matrix representing the similarity or affinity between data points. There are many ways to generate the affinity matrix, as described by Luxburg (2007).

The affinity matrix may be viewed as a graph of nodes that reflect the data points and edges that reflect the strength of connection between the points. For example, if the affinity matrix is defined as $A = (a_{ij})_{i,j=1,\dots,n}$ where $a_{ij} = 0$, this means the ith and jth points have no connectivity.

The degree matrix is a diagonal matrix, D, whose elements are defined by

$$d_i = \sum_{j=1}^{n} a_{ij}$$

The graph Laplacian matrix is defined from this matrix. There are various forms of Laplacian matrix definitions. The definition used in Ng et al. (2002) is the normalised Laplacian:

$$L := D^{-1/2} A D^{-1/2}$$

There are two approaches for using the eigenvectors of the Laplacian to cluster the data, viz. spectral graph partitioning or clustering of k eigenvectors. In the former the eigenvectors are used to make graph cuts. The latter approach is used here and described next. A preceding step to re-normalise the k eigenvectors to form tighter clusters is used. If $X = \{x_1, x_2, ..., x_k\}$ are the k eigenvectors of the Laplacian L, calculate Y, the renormalised eigenvectors as:

$$Y_{ij} = X_{ij} / \left(\sum_j X_{ij}^2 \right)^{1/2}$$

Finally k-means clustering was applied to the matrix Y to generate clusters and allocate the original data points according to the clustering.

There are several clustering methods that are applicable for data that exhibit affinities, including Agglomerative Clustering and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Pedregosa et al., 2011). These two methods require feature vectors for the Euclidean distance calculations whereas with spectral clustering it was possible to provide an affinity matrix directly.

4.4 Proposed Approach: Concurrent Action structure using clustering (CASC)

In this section the proposed approach for extracting task-agnostic, relational action structure from multi-task data, thereby enhancing action selection for improved exploration of new tasks, is defined.

This problem is cast in the multi-task space where a range of tasks, similar in nature, are generated and share some task-invariant features. This is also a multi-discrete action space where the agent must select multiple actions in one timestep. In this work, the agent is limited to two actions per timestep. In this setting, relationships between individual actions in a concurrent action are possible, such that some action combinations are synergistic in nature and typically associated with bottleneck states.

The remaining sections outline the CASC approach:

- Section 4.4.1 describes the proposed approach and motivation
- Section 4.4.2 describes the process used to generate data and extract task-agnostic structure in a multi-task setting
- Section 4.4.3 defines the spectral clustering method for extracting relational structure from the action space
- Section 4.4.4 elaborates on how action structure is transferred to enhance exploration
- Section 4.4.5 provides details of a comparative action elimination method using the same multi-task data
- Section 4.4.6 places this work in the context of the related work in this area
- Section 4.4.7 provides an overview of implementation details

4.4.1 Approach and Motivation

The proposed approach in this chapter is to generate data from multiple tasks and use bottlenecks to highlight which actions are important. The hypothesis is that the high state visitations that accompany bottleneck states in successful trajectories effectively identifies actions whose structure might be useful. The high state visitations associated with bottlenecks are used to divide the action space into regions of high affinity actions. The implication is if this action is important then the composition of actions is useful to remember and might be useful for other tasks too. Spectral clustering (Ng et al., 2002) is used to categorise and divide the action space. The important action relations are retained and indirectly suppresses relations that are less important. Both clustering and action elimination are applied to a modified exploration process as described below in sections 4.4.3 and 4.4.5.

Intuitively, categorisation and elimination are very human concepts. Almost the first thing we do, as humans, is to perform a mental categorisation of data when learning something new (Chi et al., 1981). Similarly, the human brain eliminates an enormous amount of sensory and other data to focus on the essentials. These two approaches lead to an implicit and explicit perspective on extracting, learning and using structure. Action elimination may be viewed as implicit, i.e. not including any particular architecture for learning structure. The idea is that task invariant actions are associated with features in the environment even if the specific features are unknown. This implicit structure is applied as prior knowledge during action selection to new tasks.

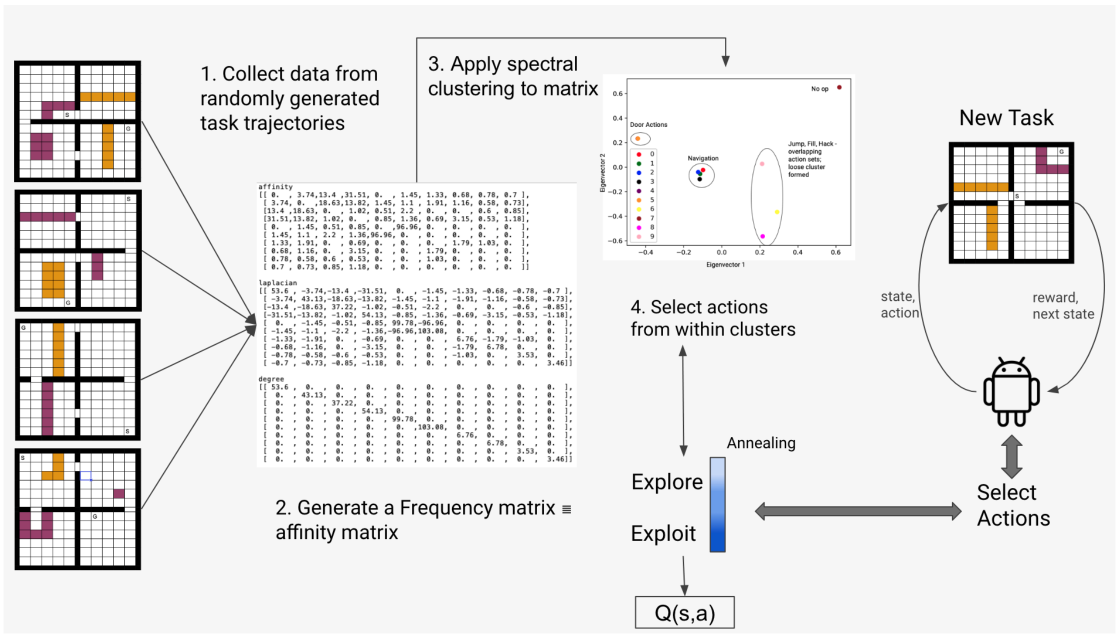
Clustering on the other hand is more explicit, i.e it separates the action space based on inherent structure in the space and the model is architecturally modified to adopt this structure during action selection. The agent is trained to learn explicit structure in the action space in the form of clusters. The premise is that high counts of some action sets imply an underlying relationship or affinity between the primitive actions in those sets, that can be used to separate the action space into clusters. During exploration and action selection the agent would select concurrent actions from within a cluster on the basis that there is a higher likelihood of picking an effective action set.

A comparative approach was not easy to find because a common way to deal with multi-discrete action spaces is to transform the space into a discrete action space (Kanervisto et al., 2020). If the resultant action space was too large, less common actions were dropped in a process known as action elimination (Even-Dar et al., 2003; Zahavy et al., 2018). Kanervisto et al. (2020) actually find that multi-discrete actions perform better overall than the discrete transformation nevertheless, empirically, action elimination can be an effective approach and is used as a comparative method against the proposed clustering approach. This approach conceptually improves sample efficiency by applying prior knowledge during exploration. A possible complication for action elimination in the multi-task setting could be a lack of diverse data leading to some actions being eliminated prematurely.

The sections 4.4.2-4.4.4 below describe the proposed CASC algorithm, that first extracts structure then transfers it to train new tasks. A sub-set of this process (section 4.4.5) is used to implement the action elimination method used for comparison. Refer to Figure 4.1 for an overview of the spectral clustering process and Figure 4.2 for an overview of the action elimination process flow.

4.4.2 Extracting task-agnostic structure in a multi-task setting

This section describes the proposed method for collecting data across multiple tasks and extracting task invariant structure by averaging action frequency data,


1. Collect data from randomly generated task trajectories
3. Apply spectral clustering to matrix
No op
Door Actions
Navigation
Jump, Fill, Hack - overlapping action sets; loose cluster formed
Eigenvector 1
Eigenvector 2
New Task
state, action
reward, next state
4. Select actions from within clusters
Annealing
Explore
Exploit
Select Actions
Q(s,a)
2. Generate a Frequency matrix ≡ affinity matrix

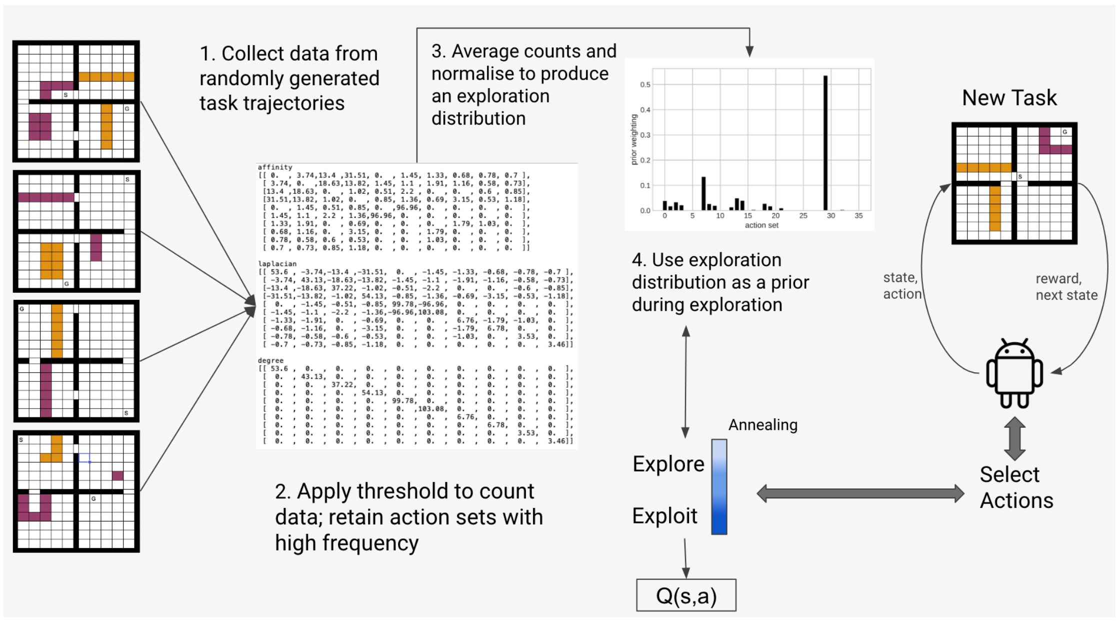
```
affinity
[[ 0.  , 3.74,13.4 ,31.51, 0.  , 1.45, 1.33, 0.68, 0.78, 0.7 ],
 [ 3.74, 0.  ,18.63,13.82, 1.45, 1.1 , 1.91, 1.16, 0.58, 0.73],
 [13.4 ,18.63, 0.  , 1.02, 0.51, 2.2 , 0.  , 0.  , 0.6 , 0.85],
 [31.51,13.82, 1.02, 0.  , 0.85, 1.36, 0.69, 3.15, 0.53, 1.18],
 [ 0.  , 1.45, 0.51, 0.85, 0.  ,96.96, 0.  , 0.  , 0.  , 0.  ],
 [ 1.45, 1.1 , 2.2 , 1.36,96.96, 0.  , 0.  , 0.  , 0.  , 0.  ],
 [ 1.33, 1.91, 0.  , 0.69, 0.  , 0.  , 0.  , 1.79, 1.03, 0.  ],
 [ 0.68, 1.16, 0.  , 3.15, 0.  , 0.  , 1.79, 0.  , 0.  , 0.  ],
 [ 0.78, 0.58, 0.6 , 0.53, 0.  , 0.  , 1.03, 0.  , 0.  , 0.  ],
 [ 0.7 , 0.73, 0.85, 1.18, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]]

laplacian
[[ 53.6 , -3.74,-13.4 ,-31.51,  0.  , -1.45, -1.33, -0.68, -0.78, -0.7 ],
 [ -3.74, 43.13,-18.63,-13.82, -1.45, -1.1 , -1.91, -1.16, -0.58, -0.73],
 [-13.4 ,-18.63, 37.22, -1.02, -0.51, -2.2 ,  0.  ,  0.  , -0.6 , -0.85],
 [-31.51,-13.82, -1.02, 54.13, -0.85, -1.36, -0.69, -3.15, -0.53, -1.18],
 [  0.  , -1.45, -0.51, -0.85, 99.78,-96.96,  0.  ,  0.  ,  0.  ,  0.  ],
 [ -1.45, -1.1 , -2.2 , -1.36,-96.96,103.08,  0.  ,  0.  ,  0.  ,  0.  ],
 [ -1.33, -1.91,  0.  , -0.69,  0.  ,  0.  ,  6.76, -1.79, -1.03,  0.  ],
 [ -0.68, -1.16,  0.  , -3.15,  0.  ,  0.  , -1.79,  6.78,  0.  ,  0.  ],
 [ -0.78, -0.58, -0.6 , -0.53,  0.  ,  0.  , -1.03,  0.  ,  3.53,  0.  ],
 [ -0.7 , -0.73, -0.85, -1.18,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.46]]

degree
[[ 53.6 ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  , 43.13,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  , 37.22,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  , 54.13,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  , 99.78,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,103.08,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  6.76,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  6.78,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.53,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.46]]
```

Figure 4.1: Process Flow for CASC: In step 1 tasks are randomly generated by varying the locations of the start (S), goal (G) and special states (coloured states). Data generated from solving these tasks using an RL algorithm is processed in step two to produce the affinity matrix. Step three applies spectral clustering to the affinity matrix to create clusters of primitive actions with high affinity. Finally, in step four a Q-learning algorithm is modified to select actions from the clusters during exploration for new tasks.

generating the frequency or count matrix for section 4.4.3.

A key feature of this approach is that the action structure is task invariant and context-invariant, which means it is transferable to other tasks without concern about the nature of the source or destination tasks, or about state associations. The idea is to extract and transfer only the elements common to multiple tasks that are not conditional on task specific elements such as the transition dynamics or goal states. This is achieved by first using a pre-training phase to collect data from multiple tasks so that common structural information can be harvested, as outlined in algorithm 2.

In step 1, the tasks are trained using a Q-learning algorithm with $\epsilon-$greedy exploration to generate trajectories. A sample of successful trajectories is collected, where success is defined by whether the goal state is reached before the maximum


1. Collect data from randomly generated task trajectories
3. Average counts and normalise to produce an exploration distribution
prior weighting
action set
New Task
4. Use exploration distribution as a prior during exploration
state, action
reward, next state
2. Apply threshold to count data; retain action sets with high frequency
Annealing
Explore
Exploit
Select Actions
Q(s,a)

```
affinity
[[ 0.  , 3.74,13.4 ,31.51, 0.  , 1.45, 1.33, 0.68, 0.78, 0.7 ],
 [ 3.74, 0.  ,18.63,13.82, 1.45, 1.1 , 1.91, 1.16, 0.58, 0.73],
 [13.4 ,18.63, 0.  , 1.02, 0.51, 2.2 , 0.  , 0.  , 0.6 , 0.85],
 [31.51,13.82, 1.02, 0.  , 0.85, 1.36, 0.69, 3.15, 0.53, 1.18],
 [ 0.  , 1.45, 0.51, 0.85, 0.  ,96.96, 0.  , 0.  , 0.  , 0.  ],
 [ 1.45, 1.1 , 2.2 , 1.36,96.96, 0.  , 0.  , 0.  , 0.  , 0.  ],
 [ 1.33, 1.91, 0.  , 0.69, 0.  , 0.  , 0.  , 1.79, 1.03, 0.  ],
 [ 0.68, 1.16, 0.  , 3.15, 0.  , 0.  , 1.79, 0.  , 0.  , 0.  ],
 [ 0.78, 0.58, 0.6 , 0.53, 0.  , 0.  , 1.03, 0.  , 0.  , 0.  ],
 [ 0.7 , 0.73, 0.85, 1.18, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]]

laplacian
[[ 53.6 , -3.74,-13.4 ,-31.51,  0.  , -1.45, -1.33, -0.68, -0.78, -0.7 ],
 [ -3.74, 43.13,-18.63,-13.82, -1.45, -1.1 , -1.91, -1.16, -0.58, -0.73],
 [-13.4 ,-18.63, 37.22, -1.02, -0.51, -2.2 ,  0.  ,  0.  , -0.6 , -0.85],
 [-31.51,-13.82, -1.02, 54.13, -0.85, -1.36, -0.69, -3.15, -0.53, -1.18],
 [  0.  , -1.45, -0.51, -0.85, 99.78,-96.96,  0.  ,  0.  ,  0.  ,  0.  ],
 [ -1.45, -1.1 , -2.2 , -1.36,-96.96,103.08,  0.  ,  0.  ,  0.  ,  0.  ],
 [ -1.33, -1.91,  0.  , -0.69,  0.  ,  0.  ,  6.76, -1.79, -1.03,  0.  ],
 [ -0.68, -1.16,  0.  , -3.15,  0.  ,  0.  , -1.79,  6.78,  0.  ,  0.  ],
 [ -0.78, -0.58, -0.6 , -0.53,  0.  ,  0.  , -1.03,  0.  ,  3.53,  0.  ],
 [ -0.7 , -0.73, -0.85, -1.18,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.46]]

degree
[[ 53.6 ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  , 43.13,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  , 37.22,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  , 54.13,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  , 99.78,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,103.08,  0.  ,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  6.76,  0.  ,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  6.78,  0.  ,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.53,  0.  ],
 [  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  3.46]]
```

Figure 4.2: Process Flow for Action Elimination: In step 1 tasks are randomly generated by varying the locations of the start (S), goal (G) and special states (coloured states). Data generated from solving these tasks using an RL algorithm is filtered in step two to retain only the highest performing actions. Step three converts the highest performing actions into an exploration policy. Finally, in step four a Q-learning algorithm is modified to select actions using the exploration policy during exploration for new tasks.

allowed step count (a hyperparameter). This process is effectively collecting expert or optimal data from multiple tasks to refine the quality of data transferred to future tasks.

In step 2, the frequency counts of actions from optimal trajectories are processed for transferable structural data. The intuition is that optimal trajectories will have the highest percentage of bottleneck states, with corresponding useful actions, which is a good starting point for analysing structural information in a multi-discrete action setting.

Step 2b in algorithm 2 outlines how the count data is processed for the clustering approach. The counts per set of actions are collected across multiple tasks, retaining only the most frequently used actions, controlled by a threshold hyperparameter, t. The threshold and top counts hyperparameters are used to control

```
Algorithm 2: Processing Count Data
Inputs
$N$ - Number of primitive actions
Hyperparameters: threshold $t$, no. of top counts $NC$
1. Generate Counts matrix
Generate trajectories from multiple tasks using RL algorithm (eg.
Q-learning)
Collect count of each action set selected per state for successful
near-optimal episodes
Average over multiple tasks and runs to generate a count matrix by
state and action set, $Counts(s, a, a)$
2. Process Action Counts
Reshape Counts from $S$x$N$x$N$ to $S$x$N^2$, unrolling action sets per
state
    2.a. Process Action Elimination Count Matrix
        Set threshold $t > 0$ and $NC$ to low number
        Apply threshold filter to Counts and accumulate only the
        remaining top $NC$ action counts across all states
        Combine count matrices for all tasks and normalise the
        vector formed: $probs(A_i)$
        Return Vector $probs(A_i)$ with dim 1x$N^2$
    2.b. Process Clustering Count Matrix
        Set threshold $t = 0$ and increase $NC$
        Apply threshold filter to Counts and accumulate the top
        $NC$ action counts across all states
        Reshape to form matrix $W$ with dimensions $N$ x $N$
        Return Matrix $W$, with dim $N$ x $N$
```

the sparsity of the action frequency count matrix. The process of filtering only the best trajectories and averaging over all tasks builds a collection of task-invariant actions. Specifically, the action counts are averaged across all tasks and the action frequency count matrix is re-formed to a $N \times N$ matrix, where N is the number of primitive actions.

For example, if the primitive action set is $A = \{0, 1, 2, 3, 4, 5\}$ in a multi-discrete action space of MultiDiscrete($[6, 6]$) such that a sample action $[3, 0]$ that has an averaged count of 23 over multiple tasks, that action will contribute 23 to row 3 and column 0 of a sample 6x6 action frequency matrix,

$$\begin{bmatrix} 10 & 2 & 30 & 1 & 0 & 0 \\ 45 & 20 & 19 & 1 & 0 & 15 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ \mathbf{23} & 0 & 9 & 40 & 0 & 0 \\ 1 & 0 & 0 & 1 & 50 & 0 \\ 7 & 0 & 0 & 1 & 0 & 10 \end{bmatrix}$$

The output of this algorithm is the averaged action affinity matrix that is used as the input to the clustering algorithm 3. The count frequencies are likened to action affinities: a high frequency or count is taken to mean a high action affinity which implies a strong relation between the two actions in the set. In this first phase, averaging the counts over multiple tasks makes this a task-agnostic, context-free approach so that there is no direct association with states or tasks. The task-agnostic matrix of action frequencies is therefore used as an action affinity matrix that denotes the strength of the action relationships. The next step will extract the transferable structural information from the task-agnostic matrix of action frequencies.

4.4.3 Using spectral clustering to exploit relationships between actions

This section describes the proposed method for clustering the action space by applying spectral clustering over the average frequency counts or affinity matrix.

The output from the pre-processing in algorithm 2 is a task-agnostic, averaged action affinity matrix. The spectral clustering algorithm 3 receives this matrix as input. Relationships in the action space are important in this problem setting so the main idea is to extract the relational behaviour and transfer it to new tasks for faster learning. Spectral clustering separates data using the eigenvectors of the Laplacian of an affinity matrix (Luxburg, 2007). The affinity matrix may be

```
Algorithm 3: Spectral Clustering of Actions
Input:
Count matrix for action sets, $W$, with dim $N$ x $N$, where $N$ is no. of primitive actions
1. Prepare Affinity Matrix:
Check for symmetry, zero diagonals
2. Apply Spectral Clustering algorithm (Ng et al., 2002)
Degree matrix: $D$ where $d_{ii}$ is the diagonal sum of row $W[i]$
Normalised Laplacian: $L = D^{-1/2}WD^{-1/2}$
$V$ is a matrix formed from the top $k$ of eigenvectors of $L$
$Y$ is the matrix $V$ normalised
Apply k-means to $Y$ for $k$ clusters and create list of clusters, $C$
3. Post Process Clusters
Remove clusters with low silhouette score
Remove single item sets
Return List of clusters $C$
```

viewed as a graph where each action is a node and the edge is the strength of the relationship between actions, the strength in this case is reflected by the averaged counts in the frequency matrix. Spectral clustering was performed on the frequency matrix to find clusters in the action space. This results in the generation of clusters with elements consisting of actions that are grouped by strength of relationship. Specifically this means that if three actions display relationships and this manifests in the training data, these three actions are likely to be clustered together.

Algorithm 3 describes how the spectral clustering algorithm from Ng et al. (2002) is applied to cluster actions using the affinity (frequency) matrix, including some conditions that the affinity matrix should meet. Step 1 in Algorithm 3 transforms the count matrix into the appropriate form. In step 2 the eigenvalues and eigenvectors of the normalised Laplacian are calculated; typically cluster blocks are revealed in this step. K-means clustering (Pedregosa et al., 2011) was used to group the top eigenvectors. The number of clusters k is a hyperparameter, however it is also determinable using an eigengap heuristic (Luxburg, 2007; Ng et al., 2002) that monitors the gaps between the eigenvalues for distinct jumps or steps. Step 3 is a processing step that uses a cluster measure to check the validity of each cluster and retains only the most confident clusters based on the silhouette score. Given the nature of how the clusters are used for action selection, single element clusters are removed. This will be described further in algorithm 4. The output of this algorithm is a set of clusters for the action space.

It should be noted that although spectral clustering was applied to this data, another clustering method could be used instead. Spectral clustering was found to be convenient for this purpose as the frequency matrix functionally aligns with the requirements for an affinity matrix.

4.4.4 Transferring action structure to enhance exploration

This section describes the proposed method for transferring the clusters defined in algorithm 3 to enhance the exploration process of new tasks.

Algorithm 4: Action Selection - Clustering

```
Input:
    $C$: list of clusters
    $Q(s, A_i)$: current Q-function
    $np$: number of individual actions in a concurrent action set
If Exploration:
    Sample a cluster $c_i$ from $C$ randomly
    Select $np$ actions from within $c_i \rightarrow A_i$
Else if Exploitation:
    Find action set with max value in $Q(s, A_i) \rightarrow A_i$
Return $A_i$
```

The previous algorithm 3 extracted structural information from the action space in the form of clusters of related actions. Algorithm 4 describes the approach for transferring this structure to new tasks to improve performance. This algorithm targets the exploration process in multi-discrete action spaces because multi-discrete actions increase the size of the exploration space (Kanervisto et al., 2020; Sutton and Barto, 1998b). The approach modifies the action selection process to use the clusters derived in algorithm 3 to improve exploration. In Algorithm 4 actions are selected by, first, randomly selecting a cluster, then second, randomly selecting actions from within that cluster. The actions within each cluster should be related actions with strong affinities. In practice, this reduces illegal or useless sets generated by random action selection during exploration.

4.4.5 Action elimination in a multi-discrete action setting

This section describes the proposed method for implementing action elimination in a multi-discrete action context. The method takes as input the vector of probabilities calculated by algorithm 2 and uses this as an exploration policy for the exploration of new tasks. The exploration policy provides biased input into which actions are good or bad.

Algorithm 5: Exploration with Action Elimination Prior

```
Input:
    Exploration policy $probs(A_i)$
    Current Q-function $Q(s, A_i)$
If Explore:
    Sample an action set from $probs(A_i) \to A_i$
Else if Exploit:
    Find action set with max value in $Q(s, A_i) \to A_i$
Return $A_i$
```

An action elimination (Zahavy et al., 2018; Even-Dar et al., 2003) method was selected as a comparative method for the main proposed clustering approach defined in algorithms 3 and 4.

A policy, called the exploration policy, is extracted from the frequency count matrix as described in step 2a of algorithm 2. The algorithm describes how the frequencies per action set are collected across multiple tasks in step 1. Step 2a (algorithm 2) describes how only the most frequently used action sets are retained, controlled by a threshold hyperparameter, t. Frequencies below this threshold will be filtered out, effectively pruning the matrix. The matrix is unrolled to a more use-able probability vector (exploration policy) format, after normalising the data to create a distribution, with dimensions $1 \times N^2$, a distilled set of actions averaged across multiple tasks.

Algorithm 5 defines how the exploration policy is transferred to impact action selection in new tasks as part of a Q-learning algorithm. The biased exploration policy is sampled for actions during the explore phase while the exploit phase is unmodified. Normally during exploration all actions would be available for selection so this may be viewed as providing the agent with prior expert information on which actions are better. The intuition is that high frequency action sets are probably more useful and could be used as a basis for action elimination. Action elimination is, however, potentially dangerous to the agent if useful actions are removed due to inadequate pre-training data. The task-agnostic manner in which this data was collected should help to minimise bias towards any one task and remove generally useless actions by favouring the task-invariant sets, assuming task diversity is maintained. The action elimination process amounts to injecting implicit structural knowledge about the action space into the system versus the explicit structure transferred in the spectral clustering approach (Section 4.4.3).

4.4.6 Discussion

In CASC, action clusters are proposed to speed up exploration by reducing the number of elements to select. In general, exploration should have an element of randomness to prevent falling into local optima too early (Sutton and Barto, 1998a). The action selection process in CASC tries to maintain this randomness, by first randomly selecting a cluster, then randomly selecting actions within the cluster. The structural information imparted by the clusters makes the exploration space more meaningful, and potentially more interpretable, while allowing actions to be selected at random. This method is similar to the k-Exp exploration approach adopted in Tennenholtz and Mannor (2019) but in that work the purpose for the clustering was different; Tennenholtz and Mannor (2019) used clusters to reduce the size of the action space by grouping similar actions together, whereas CASC is using clusters to denote relational structure, grouping actions that synergise instead.

This chapter has expanded on some of the ideas in Rosman and Ramamoorthy (2015), but in a concurrent action setting, specifically extracting task-invariant structure into a prior for action exploration. There are a few similarities with the approach of Tennenholtz and Mannor (2019) who use supervised embedding and clustering of the action space versus the unsupervised, count-based spectral clustering approach of CASC; both works consider the action-only context. The use of the clusters during exploration is also different; Tennenholtz and Mannor (2019) cluster to prevent redundant selections of actions whereas CASC clusters effective action combinations with high affinities.

4.4.7 Implementation Details

4.4.7.1 Environment

A requirement for testing the approaches mentioned above is a rich action space with actions that interact and have relations. Specifically, unique synergies exist where some actions, when combined, will achieve a state transition that no other action combinations are capable of making. The environment should therefore have some states that require special combinations of actions to transition to the next state.

A four-room 10×10 grid-world environment (Figure 4.3) was modified to sup-

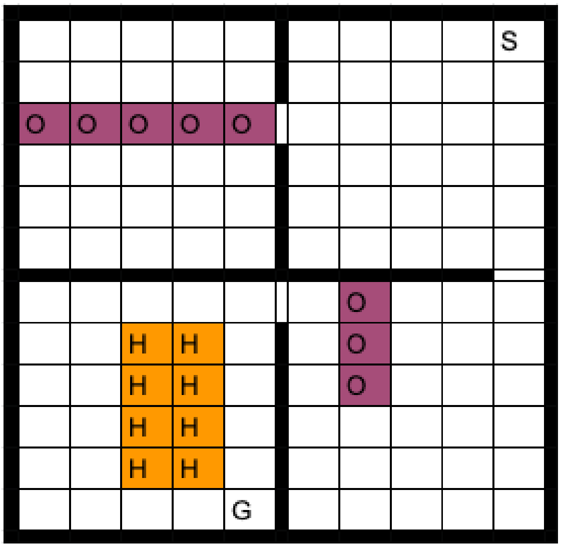


Figure 4.3: Grid-World requiring multi-discrete actions on every step, where S-Start, G-Goal, H-Holes, O-Obstacles and gaps in the walls indicate doors

Table 4.1: Action-State Categories; the index of the action and an abbreviated form is also provided. Multiple actions may be selected from within and across categories.

State	*Action Set*	*Category*
Normal	Up (U-0), Down (D-1), Left (L-2), Right (R-3)	Navigation
Obstacle	Hack (H-7), Jump (J-6)	Obstacles
Doors	Open (O-4), Enter (E-5)	Doors
Holes	Fill (F-8), Jump (J-6)	Holes
Any	No-Op (N-9)	None

port these requirements. To support the generation of multiple tasks for pre-training, the environment was designed to be configurable, i.e. the start (S) and goal (G) positions, internal walls and special states are moveable to support random task generation. A single goal state (G) is located in one of the rooms for each task and the agent is allocated a start state (S).

The set of ten primitive actions is fixed across all tasks and listed in Table 4.2. The environment accepts a multi-discrete action on every step and has special states with corresponding special or synergistic action combinations, as detailed in Table 4.1. For example, a set of door related actions [Open, Enter] are associated

Table 4.2: The full set of primitive actions with action index

0	1	2	3	4	5	6	7	8	9
U	D	L	R	O	E	J	H	F	N

with a special type of door-state such that only this set of actions can transition through a door state. The expectation is that the agent would learn that the Open and Enter actions are door-related and create a cluster for door-related actions that is independent of, say, navigation actions [Up, Down, Left, Right]. Two additional special state categories add a further element of complexity, viz. the holes and obstacles were designed to share a primitive action (the Jump action), to mimic possible real-world scenarios where actions overlap. Finally a no-op action was added that can be combined with any action but has no effect.
Rewards: there is a per-step penalty of -0.01 and the goal state has a reward of 10.

More details of the dynamics and other aspects of the environment are provided below:

- Special states require a particular action combination to successfully transition to the next state. There is no reward if the agent is successful and no penalty if the agent fails.
- Any action combination may be selected in any state. There is no reward penalty for selecting an incompatible set of actions in a state.
- The environment is sparsely rewarded with a positive reward received only on reaching the goal state. There are no other positive rewards.
- The every-step penalty serves to encourage the agent to be optimal in terms of timesteps.
- While the environment is small, the agent does not have location of the goal state, special states or walls included in the agent state. The agent state is

simply the one hot encoded position of the agent in the grid. This makes the problem partially observable or a POMDP.

- More details on action combinations:
  - [Open, Enter] is required to transition a door-state. Taking this action combination on another state is possible but has a neutral response (i.e. no state change).
  - Hole and Obstacle states share the Jump action. [Hack, Jump] transitions obstacle states while [Fill, Jump] transitions hole states. A navigation action is not required; the environment manages the direction. The agent needs to be facing the state in the correct direction in order to successfully transition.
  - The No-Op action can be combined with any action but is neutral. For example [Up, No-Op] will move up one step. There are no penalties for taking a No-Op action. Combinations of special actions and No-Op will also have a neutral result.

4.4.7.2 Task Setup

In this environment, a task is characterised by the location of the start and goal states, walls and door states. Tasks are configurable and are generated by randomly allocating the start and goal states, and the positions of the internal walls and doors. For each task configuration the agent was trained for 50 runs of 200 episodes; each episode was truncated after 200 timesteps. 10 tasks were used for the initial pre-training phase.

4.4.7.3 Training the RL algorithm

Due to the small number of states and actions, a tabular Q-learning algorithm was used to solve this environment. The multi-discrete actions were converted into discrete actions for evaluation in the Q-function, but actively composed from primitive actions during action selection based on the clustering mechanism defined in algorithm 4. Other value-based algorithms (Sutton and Barto, 1998b) would work as the approach is algorithm agnostic and just requires a method for incorporating the clusters into the exploration process. The agent was configured to select and take multi-discrete actions, in this case two actions per timestep. There were no limitations on the composition of the action sets, so potentially useless action sets such as [Up,Down] were possible.

There are three phases of training, viz. a pre-training phase for data collection across multiple tasks, an analysis phase for extracting structural information for transfer and finally the multi-task transfer phase with structural action information applied to new tasks.

Pre-training: This phase aligns with section 4.4.2, algorithm 2, step 1, where multiple random tasks were generated and trained using Q-learning with ϵ-greedy exploration, starting at 0.9 and annealed to 0. All data was collected in the form of trajectory tuples (s, a, r, s').

Analysis: during this phase, data from the pre-training phase was collated according to the approach in sections 4.4.2-4.4.5.

For CASC: the cluster frequency matrix generated as per algorithm 2, step 2b was subjected to spectral clustering as described in algorithm 3. The output of this process is a list of clusters in the action space.

For the action elimination approach: an exploration policy was generated as per algorithm 2, step 2a, ready for the transfer step.

Multi-task Transfer:

For CASC: the exploration process of the RL algorithm is enhanced to use the clusters generated, as described in algorithm 4.

For the action elimination approach: the exploration process of the RL algorithm is fitted with the exploration policy described in section 4.4.5, algorithm 5.

The CASC and action-elimination agents are evaluated over randomly generated tasks. The training performance of these algorithms is compared with a vanilla or un-modified Q-learning baseline algorithm (section 2.3.3) over the same set of tasks.

4.5 Experiments and Results

Experiments were designed to address several key questions:

- Is it possible to learn context-free, task-agnostic relational action structure from multiple tasks in the multi-discrete action space? (section 4.5.1)
- Does the transfer of this structure, applied to the exploration process, improve the training of new unseen tasks? (section 4.5.1)
- Is the performance achieved comparable with a high performing action elimination algorithm? (section 4.5.2)
- Does the method perform well compared with ablative random clustering methods? (section 4.5.3)

4.5.1 Concurrent Action structure using clustering - CASC

The multi-phase CASC algorithm was executed as described in section 4.4.7.3, specifically:

- Random tasks were generated for the pre-training, multi-task data collection phase as per algorithm 2 (step 1)
- Each task was trained using a Q-learning algorithm with ϵ-greedy exploration for 200 episodes and 50 runs

- The cluster frequency matrix was collated as described in Step 2b of algorithm 2 and spectral clustering was applied to this matrix to generate clusters of actions, as per algorithm 3.
- Next a vanilla agent's exploration process was enhanced with clustering as per algorithm 4, so that action selection during exploration was constrained to intra-cluster selections rather than inter-cluster. This meant the agent would select actions from within a door-related cluster or within a navigation cluster in a more intuitive way, versus free-format action composition.

Results in Figure 4.4 demonstrate that when spectral clustering was applied to the task-agnostic frequency matrix it revealed partitions in the action space according to expectations. The threshold was set to zero so no action sets were removed from the action space during the analysis phase. The eigenvalues generated by the algorithm gave an indication of how the matrix could be partitioned, particularly the second smallest eigenvalue (Ng et al., 2002). Plotting the eigenvectors of the first and second smallest non-zero eigenvalues, after reducing the dimension of the eigenvectors to 2D using Principal Component Analysis (PCA), shows a clear separation of the action space and Figure 4.4 shows clusters of actions distinguishable and associated with the types of state in Table 4.1.

The clusters identified above were transferred to a vanilla Q-learning agent to enhance action selection during exploration in new tasks, as described in algorithm 4.

The plots in Figure 4.5 show the average performance over multiple randomly generated tasks compared with other approaches, including transfer using action elimination and a baseline Q-learning algorithm. The plot shows a significant improvement in convergence times when CASC is compared to the baseline. The results support the hypothesis that the transfer of task-agnostic action structure in a multi-discrete setting can produce a significant reduction in training times over new, unseen tasks.

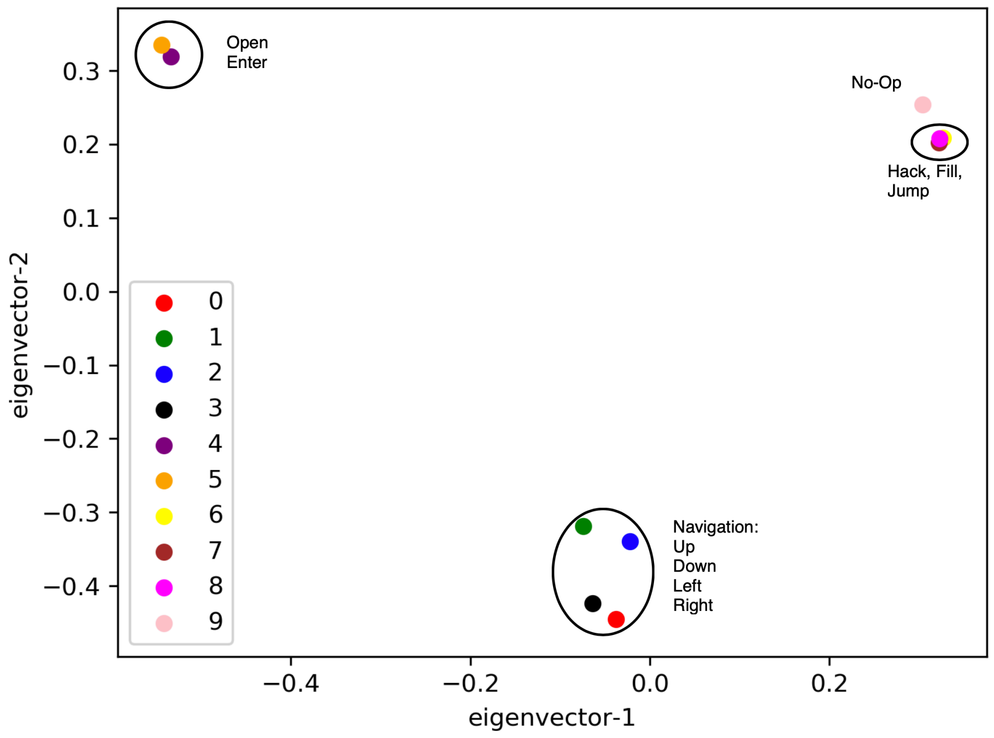


Figure 4.4: Three visible clusters generated in the 10-action space align with the states defined in Table 4.1

4.5.2 Action Elimination

The action elimination approach, a comparative method to CASC, was executed as described below:

- Random tasks were generated for the pre-training, multi-task data collection phase as per algorithm 2 (step 1)
- Each task was trained using a Q-learning algorithm with ϵ-greedy exploration for 200 episodes and 50 runs
- The exploration policy was derived as per algorithm 2 (step 2a); specifically the frequency counts were aggregated across all states for each action set and normalised, resulting in a vector of proportional representation (weighting vector or exploration policy) for all action sets. The threshold hyperparameter was set to 7 and the number of top actions to 2 to generate a sparse count matrix

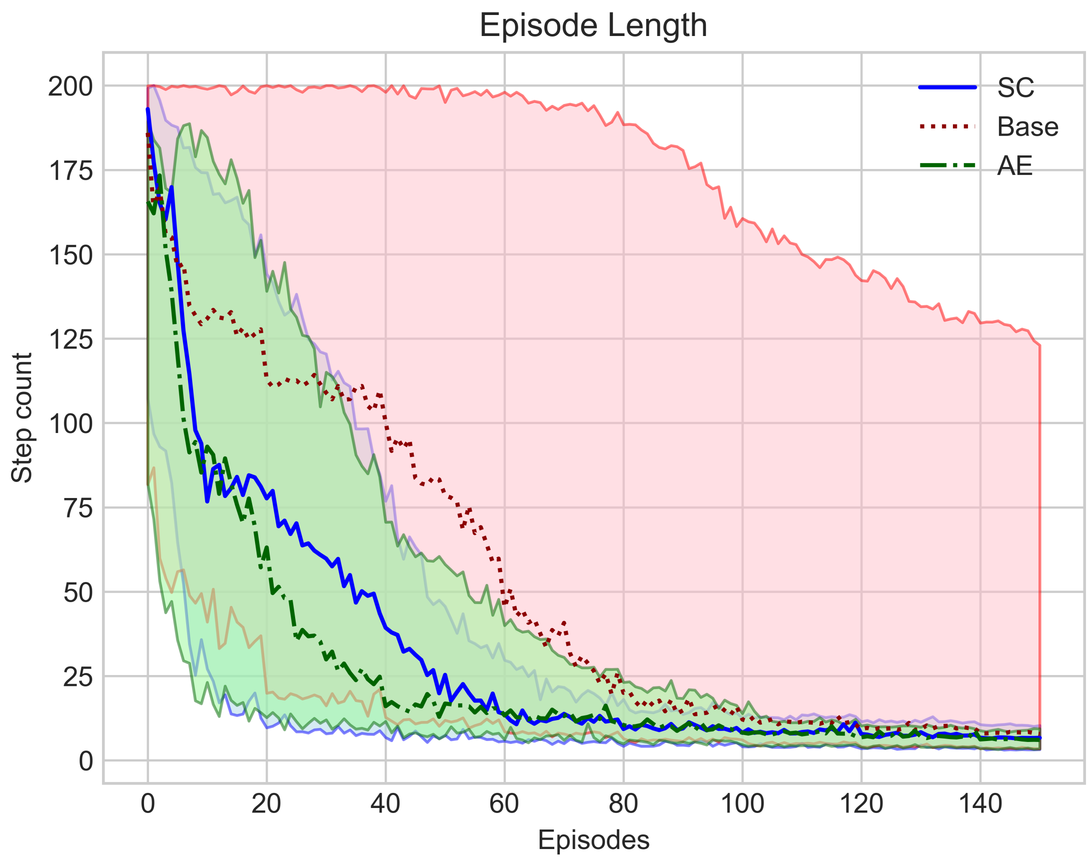


Figure 4.5: Comparison of task convergence using Q-learning (Base - red), spectral clustering enhanced Q-learning (SC - blue) and an action elimination based exploration policy (AE - green), over 10 randomly generated tasks.

- Next a vanilla Q-learning agent's exploration process was enhanced with the exploration policy, as per algorithm 5, so that action selection during exploration was influenced by the task-agnostic weightings learnt above

A sample of the resulting exploration policy or vector of proportions for the action sets is illustrated as a heatmap in Figure 4.6. Each location on the grid is a potential action combination, for example the first block on the top left is the action [Up, Up]. The intensity reflects how frequently an action set occurred in successful trajectories collected. The [Open, Enter] action set is relatively frequent, as is the [Down, Down] action set. The heatmap reflects some of the bias that arises from the data collection process, indicating how crucial it is to train over a diverse range of tasks. Increasing the number of tasks during the pre-training phase should reduce this bias but will increase the cost of pre-training.

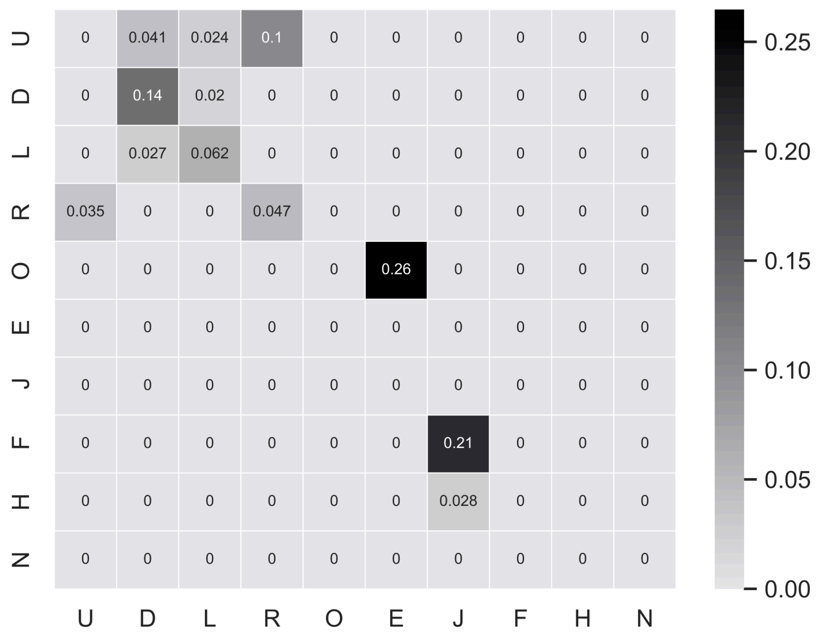


Figure 4.6: Sample of heatmaps for action elimination generated exploration policies (10 primitive actions)

The vector of proportions acts as an exploration policy, injected into the exploration process as prior knowledge and tested over new, randomly generated unseen tasks. Figure 4.5 compares the time to task convergence for new, unseen randomly selected tasks, with the spectral clustering approach in CASC and the baseline Q-learning algorithm. The plots show a significant reduction in time to convergence for both CASC and action elimination, supporting the effectiveness of action elimination during exploration in concurrent action spaces. The results demonstrate that the task-agnostic pre-training data collected can be useful as a basis for eliminating less useful action sets, without any contextual reference to the task.

4.5.3 Random clusters

To demonstrate that the effectiveness of the learned clusters in CASC is not random, an ablative experiment is performed that randomly separates the action space into clusters and injects this into the CASC algorithm in place of the learned clusters. In this experiment the action space is divided into 2 and 3 clusters respectively and injected into algorithm 4 as the action clusters. This experiment is evaluated in the same manner as the CASC experiment, over a set of randomly generated tasks.

Figure 4.7 displays the results of the random cluster ablations compared with the baseline Q-learning (red) and the spectral clustering (blue) approaches. The specific clusters used to generate these results are listed below:

The two cluster scenario, with 5 actions in each cluster (the green plot) - referred to as Cluster-A in discussions:
$[(0_{Up}, 1_{Down}, 2_{Left}, 3_{Right}, 9_{No-op}), (4_{Open}, 5_{Enter}, 6_{Jump}, 7_{Hack}, 8_{Fill})]$

The randomly generated set of 3 clusters (the pink plot) - referred to as Cluster-

B in discussions:

$[(0_{Up}, 4_{Open}, 2_{Left}), (7_{Hack}, 1_{Down}, 5_{Enter}), (6_{Jump}, 3_{Right}, 8_{Fill})]$,

The plot shows that spectral clustering demonstrates a significant performance improvement over the other methods, however, both the A and B clusters appear to outperform the base Q-learning algorithm. Given that Cluster-A provides a natural break between navigation and special actions, it is less surprising that this approach performs slightly better than the baseline Q-learning.

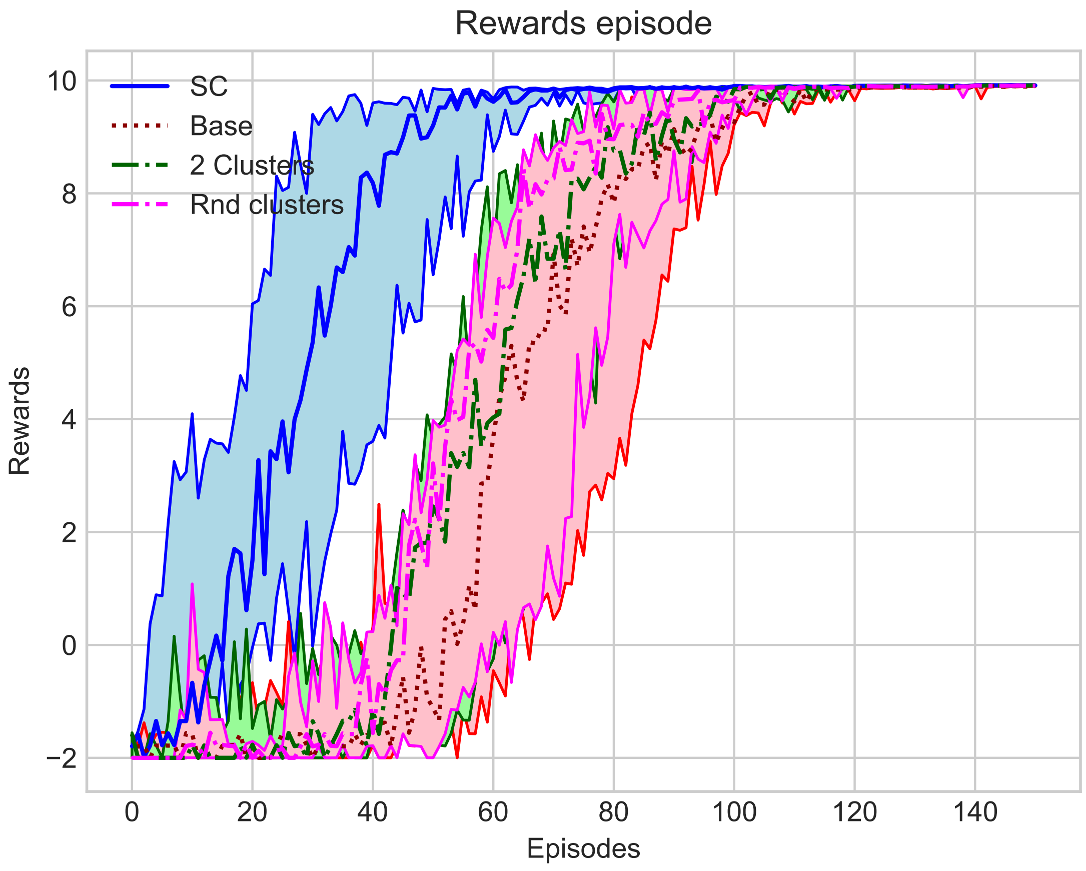


Figure 4.7: Comparison of task convergence over randomly generated tasks for base Q-learning (Base), with spectral clustering (SC), A with 2 clusters (2 Clusters) and B with 3 random clusters (Rnd clusters)

4.5.4 Discussion

Figure 4.4 demonstrates that CASC was able to identify actions with strong relationships, with the clusters showing which actions displayed affinities. The task-agnostic context-free approach adopted meant that a limited model capacity was sufficient, without the need to hold state-action relationships. In this simple environment, ten tasks were sufficient to generate data from which to extract task-agnostic structure. It is assumed that if the number of primitive actions in the action space is increased, the number of tasks would need to be scaled accordingly.

In Figure 4.5 the action elimination approach used as a comparative, highly performant method, is shown to converge slightly faster than CASC, in this environment. This is presumably because the exploration policy used in the action elimination approach contained effective action combinations, while CASC is subject to in-built stochasticity, by design. One of the limitations with the action elimination approach is a dependence on capturing sufficient task-agnostic behaviour without eliminating actions that were under-represented in the pre-training tasks. While the action elimination method was very effective, the results show that CASC is very competitive. While CASC is also susceptible to representation issues in the pre-training tasks, it is a less exclusive method as actions are not eliminated so there is less risk of prematurely rejecting actions with the clustering approach as compared with action elimination. It is envisaged that CASC will outperform action elimination in larger action spaces where the impact of elimination is potentially more harmful. Furthermore, as illustrated in Figure 4.7, even if the clusters are imperfect, there is evidence that CASC will still improve performance when compared to vanilla Q-learning. The clustering ablation compared CASC and the baseline method with a CASC variant using randomly generated clusters, where CASC outperforms the random cluster methods.

The transfer of structure approach adopted proved effective at maintaining a required level of stochasticity during exploration in the target task. Transferring a deterministic policy to the target task would have created difficulties for adequately exploring the new state-action space (Silver et al., 2014). CASC was effective at preventing this limitation, using higher-level structural priors over the action distribution in the form of clusters. Each cluster comprises a subset of related actions. By first randomly selecting a cluster, then randomly choosing an action within said cluster, sufficient stochasticity was induced to avoid fully deterministic selections. This two-tiered action sampling process retained exploratory noise while still benefiting from the guidance of the transferred policy clusters. This semi-structured stochasticity manages to circumvent the lack of exploration problem that would arise from using a fully deterministic source policy.

Finally a brief comment on how these methods fit into current exploration literature. The key exploration-related methods proposed and tested is CASC, that uses spectral clustering on a task-agnostic and context-free action frequency matrix to identify clusters of related actions. These clusters are then used to constrain action selection during exploration to sample actions from within a randomly selected cluster. The method is based on extracting and transferring task-invariant structure from the action space itself rather than using common intrinsic reward or count-based exploration techniques and is a novel way of guiding exploration in multi-discrete action spaces by analysing action relationships across tasks.

4.6 Conclusions

This chapter demonstrated that learning action structure and transferring it to support the training of new tasks can significantly reduce convergence times in a multi-task scenario. Task-agnostic data was collected from optimal trajectories across a range of tasks to improve diversity and generalisation, and spectral clustering was used to identify structure in the form of clusters in the action space. Plots of the eigenvectors (see Figure 4.4) revealed the existence of relational structures in the action space, manifesting as clusters of actions that exhibit high effectiveness when combined. This structural information was transferred to enhance the exploration process of a standard Q-learning algorithm and results showed a significant improvement was obtained from using the clusters during action selection. The action relationships were demonstrated to be as effective as the highly performant action elimination approach, evaluated on the same tasks. Comparative and ablative methods showed the clustering approach was effective at extracting task-agnostic structure with less risk of eliminating an action prematurely. While not uncommon in similar approaches (Tennenholtz and Mannor, 2019) a noteable disadvantage to the approach was the need for a pre-training phase that meant relationships were not learnt online, for which there is a preference in most competitive and widely used RL algorithms and processes (refer to section 2.5). Additionally, a tabular Q-learning algorithm was used, limiting the size of the state-action space of the environment and the generalisation capabilities of the model.

The next chapter addresses both these limitations and considers a more scalable method of learning and capitalising on structure in the concurrent, multi-discrete action space.

Chapter 5

Relational Representations in Multi-Discrete Action Spaces

5.1 Introduction

This chapter proposes a novel self-supervised approach to learning relational representations of actions during online RL in environments with multi-discrete action spaces. Such spaces, where multiple discrete actions must be selected concurrently per timestep, present an opportunity for incorporating relational information that could be beneficial for solving complex tasks.

Multi-discrete action spaces are common in RL environments, for example, controlling a humanoid robot with multiple limbs or managing the actions of multiple teams in strategy games. Naively expanding the multi-discrete action space can lead to an intractably large joint action space and, importantly, fails to utilise relational structures between the concurrent actions that could be useful for solving the task.

Prior works (Harmer et al., 2018; Sharma et al., 2017; Moodley et al., 2019) have shown the potential benefits of leveraging the multi-discrete nature of the action space for learning and exploiting relationships between actions. Incorporating relational information, such as which actions can and cannot be used together, directly into the action space in an online manner has proven to be challenging however, leading to pre-factorisation of the action space (Sharma et al., 2017) or using imitation learning (Brys et al., 2015) to extract relational information (Harmer et al., 2018).

Although many RL environments have multi-discrete action spaces, it is customary for the action space to be transformed into a single discrete action space (i.e. a single compound discrete action per timestep) by expanding out the space to the full combination of all actions (Kempka et al., 2016; Vinyals et al., 2017; Fan et al., 2022). For example, a multi-discrete action space with elements $\mathbf{A} = \{a_1, a_2, a_3\}$ would be expanded to the following set of all action combinations [1]:
$\mathbf{A_{exp}} = \{(a_1, a_1), (a_1, a_2), (a_1, a_3), (a_2, a_1), (a_2, a_2), (a_2, a_3), (a_3, a_1), (a_3, a_2), (a_3, a_3)\}$
In the naive case multiple action selections would require multiple value or policy networks, one for each multi-discrete action, or a more complex change to the algorithm as mentioned in Kanervisto et al. (2020). The expansion can, however, drastically increase the size of the action space. There are incentives to reduce the size of the increased action space because of the direct impact on the amount of exploration required if the state-action space is large, which directly affects the performance of algorithms (such as DQN) that need to learn the value of state-action combinations, often requiring the agent to experience each state-action combination multiple times. This leads to solutions involving the manual removal of actions to achieve a more manageable size (Kanervisto et al., 2020; Harmer et al., 2018; Fan et al., 2022), learned action elimination (Zahavy et al., 2018; Even-Dar et al., 2003) or using action masking (Huang and Ontan'on, 2020; Bamford and Ovalle, 2021) to limit the actions available during training.

In multi-discrete action spaces with relational actions there is, as yet, no convenient method to leverage the relationships between individual primitive actions.

[1]assuming two actions are selected per timestep

Learning relational knowledge during training can provide a useful inductive bias for shaping the action representation and accelerating learning that RL algorithms focused purely on maximising returns may fail to detect. The core motivations for this chapter are summarised below, including:

- to retain native multi-discrete action spaces to facilitate learning and use of the relationships between actions
- to derive a self-supervised signal from trajectory training data that identifies positive relationships
- to incorporate a relational objective in an online RL algorithm that shapes action representations and results in faster learning on multi-discrete tasks that benefit from relational knowledge

The goal is an integrated framework allowing online relational learning in natively multi-discrete spaces, without external guidance or demonstrations.

The approach proposed in this chapter is a novel method for learning relational representations online that are relevant to an RL task, using a relational auxiliary module that is trained alongside a multi-discrete RL algorithm. The relational module derives a self-supervised signal from online trajectory data which is used to identify and reinforce effective action relationships while the agent is being trained. This signal is used to train a relational loss objective that is incorporated into the overall reinforcement loss and contributes to shaping the action representation (Figure 5.1).

The hypothesis is that to benefit from relational structure requires an action space that supports relationships, viz. a multi-discrete action space, and that to provide relational shaping of the representation requires a relationally-directed loss component.

The multi-discrete PPO algorithm (Huang et al., 2022) was used due to the availability of existing coded implementations however the approach is not limited to this algorithm. Unlike existing techniques such as factoring the action space, using embeddings or expert data, the proposed approach focuses on directly learning the relational structure during the training of the PPO algorithm. The relational agent is compared with a vanilla PPO baseline that does not exploit action relations, and the relational agent shows faster convergence on average. The development of the representations learnt by both agents is tracked and visualised over several updates and the results demonstrate that the relational agent benefits from the structured representation shaping, contributing to faster convergence.

The specific contributions in this chapter are:

- the development of an online auxiliary module for PPO to reinforce beneficial action relationships consisting of a derived self-supervised signal for shaping action representations, and a relational loss optimised alongside PPO
- analysis of the emergence of relational structure in action representations

There is currently no similar contribution that specifically targets the multi-discrete, concurrent action space, to the best of the author's knowledge. The approach in this chapter is limited to environments with multi-discrete action spaces where actions have dependencies or relationships that would benefit task-solving.

The rest of the chapter is organised as follows: a brief overview of related research is provided in Section 5.2, followed by Section 5.3 providing an overview of the multi-discrete PPO algorithm used in the implementation. The approach in Section 5.4 outlines the proposed relational auxiliary method, providing details of the self-supervised signal and relational auxiliary objective, followed by implementation details. Section 5.5 provides details of the experiments conducted and the results, concluding with Section 5.6.

5.2 Related Work

There are two steps required when solving relational action structure problems in RL tasks, namely choosing an action space that is conducive to learning relational attributes, and modifying the RL algorithm to manage actions with relations.

5.2.1 Multi-discrete and large action spaces

Although many environments have multi-discrete action spaces, such as Vizdoom (Kempka et al., 2016) or StarCraft (Vinyals et al., 2017), it is customary to convert this into a discrete action space by expanding out the full combination of all actions as outlined in (Kanervisto et al., 2020; Harmer et al., 2018). This, however, results in a larger action space that is often managed by the manual removal of actions to achieve a more manageable size (Kanervisto et al., 2020; Vinyals et al., 2017), the learned elimination of useless actions (Zahavy et al., 2018) or representing actions with a reduced latent representation (Tennenholtz and Mannor, 2019; Dean et al., 1998; Dulac-Arnold et al., 2015; Chandak et al., 2019). In general large action spaces often result in significantly slower convergence times which is already a challenge in DRL (Irpan, 2018; Achiam, 2018).

In contrast to this expansion to a single discrete space, other approaches have proposed exploiting the benefits of the multi-discrete nature of the action space in RL (Sharma et al., 2017; Harmer et al., 2018; Wang and Yu, 2016). However, incorporating relational information, such as which actions can and cannot be used effectively together, directly into the action space in an online manner is a challenging task. To address this issue, Sharma et al. (2017) proposed a method that factorises the action space, converting 18 discrete Atari actions into multi-discrete actions with 3 dimensions (Up/Down, Left/Right, Fire). The factorisation is extended to the policy or value function, to learn values at the factored action level. This approach allows action relationships to be leveraged in an online, on-policy algorithm such as A3C, however the factorisation process requires human intervention to define the factors.

Another approach (Harmer et al., 2018) models the multi-action space as a multi-discrete Bernoulli policy consisting of independent actions and injects useful action information into an online algorithm using an auxiliary imitation signal based on expert data. This technique combines imitation learning with RL, enabling the algorithm to learn from both its own experiences and the expert demonstrations. Importantly the multi-discrete nature of the space gives the agent the capability to benefit from relational structure exhibited by the experts. By taking advantage of the structure and relationships within the action space, these approaches provide a way to learn more efficiently and effectively in complex environments. While there is no consensus on what the best algorithm should be, manual factorisation of the action space and the use of expert data are limitations that motivate the approach in this chapter to pursue an online solution without manual or expert intervention.

Zambaldi et al. (2018) introduces a framework called relational deep reinforcement learning that combines DRL with relational reasoning and uses self-attention (Vaswani et al., 2017), a relational mechanism, to align with the relational nature of the task. Zambaldi et al. (2018) use attention in a relational reasoning RL approach, learning relationships between entities in a scene in the BoxWorld and StarCraft II environments, where multiple objects exist and knowledge of where objects are in relation to each other is important to the agent. The central problem addressed by this work is that DRL struggles to solve problems that require relational reasoning, the ability to understand and reason about the relationships between entities. This can be crucial for problems with multiple entities or objects where reasoning can provide valuable information about relationships (Battaglia et al., 2016; Barrett et al., 2018; Hu et al., 2017). In this work, the policy is trained using a combination of RL and supervised learning, where the RL component learns to maximise the expected reward, while the supervised component learns to predict the relationships between entities in the environment.

5.2.2 Action Relations

Existing works in RL (Kerg et al., 2022; Webb et al., 2020, 2021; Shanahan et al., 2020; Santoro et al., 2017) have sought to learn relationships in the state and task space where the agent could benefit from related objects in a state or related tasks. The pursuit of relationships between actions is less common. A few of the works most relevant to this chapter are described next.

Jain et al. (2022) explore action relations but focuses on a varying action setting where, at any point in time, only a subset of actions are available to solve a task. The approach adopted relates actions that achieve similar outcomes so that, for example, if a hammer is not available but a nail is, an action similar to hammering should be selected. Graph Attention Networks (Veličković et al., 2018) are used to learn the relational representation. This approach is conceptually similar to the Natural Language of Actions (NLoA) by Tennenholtz and Mannor (2019) where actions are grouped by similar outcomes, effectively reducing the dimension of the action space to a more compact representation. This provides a method for allocating even new, unseen actions to possible outcomes. Other methods that focus on reducing the dimensions of the action space to a more manageable and generalisable representational form include Dean et al. (1998); Dulac-Arnold et al. (2015); Chandak et al. (2019). In these approaches the reduced action representations will contain elements of structural or relational information available in the action space.

Chitnis et al. (2020) investigate learning synergistic actions in a multi-agent setting, comparing the actions of the joint agent (with a joint action space) with the composition of actions of the individual agents. Here a synergistic set of joint actions will have an outcome that is greater than the combined outcomes of the individual actions, taken independently. The most synergistic outcome is favoured and converted to an intrinsic reward signal for training, so the agent learns how to select the most synergistic combinations of actions. This approach requires pre-trained data from the individual agents as well as the joint agent and is therefore not an online process. Synergistic action relations are a very desirable form of action relation learning (Moodley et al., 2019) and is similar to the goals of this chapter.

5.3 Multi-discrete Proximal Policy Optimisation

A brief overview of multi-discrete PPO follows, extending the original algorithm defined in section 2.5.1. This section provides the fundamentals of the algorithm modified in the proposed approach, section 5.4.

The PPO algorithm is divided into two phases: a data collection phase and an update phase. In the data collection phase the algorithm spawns multiple instances of the environment, parallelising the interaction of the agent with the environments. The trajectory data of T timesteps over N parallel environments is collected in a buffer of size NxT. Once the buffer is collected, the update phase begins, generating batches from the buffer in one or more epochs and passing this to the PPO algorithm. Recall the PPO loss from Section 2.5.1 that is rewritten here for convenience:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t) \right]$$

The agent parameters are updated using stochastic gradient descent over the loss. After a number of update epochs performed on this buffer, the buffer is discarded and the process repeats with a new collection phase. The number of epochs, timesteps, environments and batch size are all hyper-parameters of the PPO algorithm.

The pseudocode for the multi-discrete PPO algorithm provided by Huang et al. (2022) is illustrated in Algorithm 6, describing the splitting of the logits to manage multi-discrete actions.

Algorithm 6: Multi-Discrete PPO Algorithm (Schulman et al., 2017; Huang et al., 2022).

```
Initialise Actor's policy network $\pi(a|s;\theta)$ and Critic's value
network $V(s;\theta_v)$ with weights $\theta$, $\theta_v$
Initialise environments with multi-discrete action space $A$
For each iteration:
    Get initial state s from environments
    For each timestep in trajectory, starting from $t = 0$:
        // Get action logits for each discrete action
        logits from $\pi(s;\theta)$
        // Split logits into each discrete action component
        action_logits = split(logits)
        // Sample action for each component
        For each discrete action i:
            action[i] ~ Categorical(action_logits[i])
        Take multi-discrete action generated, observe r, s′
        Store (s, a, r, s′) in buffer
    Compute discounted returns R and advantage
    // Update policy
    For each epoch:
        Shuffle buffer
        For each minibatch:
            // Calculate PPO, value and entropy losses
            $L = L^{CLIP} + L^{VF} + S$ .... Eqn 2.19
            // Update policy and value networks
            $\theta \leftarrow$ Adam($\nabla\theta L$)
            $\theta_v \leftarrow$ Adam($\nabla\theta_v L$)
```

The agent has an actor-critic architecture with a separate neural network for the actor and the critic. In the multi-discrete version of the algorithm the actor generates a tuple of actions versus a single discrete action on each timestep, for example, a tuple of two discrete actions could be $[0, 1]$ or four discrete actions could be $[1, 0, 2, 1]$ where each action $a \in \{0, 1, 2, ..., N\}$.

Multi-discrete actions are generated by feeding a batch of states through the actor network to generate action predictions, specifically a set of logits for each multi-discrete action. The logits are used as the basis for a corresponding set of categorical distributions that span the action space, one distribution per set of logits. Actions are sampled from each categorical distribution to compose the multi-discrete action set, for example $[a_i, a_j]$ where $a_i \sim \delta_i$ and $a_j \sim \delta_j$ in a multi-discrete setting with two categorical distributions, referred to as $[\delta_i, \delta_j]$. It is worth noting that each action is independent of each other action in the tuple. The critic in the multi-discrete PPO algorithm is unmodified from the original PPO algorithm; it outputs the value of the state and is only indirectly impacted by the action space.

5.4 Proposed relational auxiliary module

This section provides details of the proposed relational auxiliary module designed to be trained alongside an online RL algorithm to induce relational representations of actions in multi-discrete action spaces. The multi-discrete PPO algorithm used in this chapter (Huang et al., 2022) is modified to include the self-supervised relational module.

There are three core aspects of the proposed approach and these are outlined below:

1. A multi-discrete action space is adopted that facilitates relationships between actions, instead of expanding to a single discrete action space, providing the capacity to represent and leverage relationships.

2. A self-supervised relational auxiliary module is introduced that derives a signal from the agent's training data to identify positive relationships between concurrently selected actions. This signal is used to train a separate relational loss objective, described in more detail in section 5.4.1.

3. The relational loss is added to the PPO loss as an auxiliary objective, influencing the shaping of the action representation during training. This aligns

the representation with the relational nature of the task. Section 5.4.2 elaborates on the auxiliary objective.

5.4.1 Self-supervised signal

The self-supervised module derives a learning signal from trajectory data collected in the buffer for PPO updates (referenced in Algorithm 6). The signal determines which relationships between discrete actions in a multi-discrete action tuple to reinforce during training. Together with the relational auxiliary objective (section 5.4.2) the self-supervised module helps shape the action representation learnt by the agent.

The basis for the signal is successful transitions. In domains with many ineffective action combinations or sparse rewards, a successful state transition to a different, next state is a useful signal. This derived, self-supervised signal is used to train a relational auxiliary component (see Figure 5.1), however, the signal is noisy and not all state transitions are optimal, which is normal in a developing policy. An example of a non-optimal transition is action $[Forward, No\text{-}Op]$ which is less effective than, say, $[Forward, Jump]$. The inclusion of signals like this create noise. Without knowing how effective a transition is, it will still be flagged as positive, i.e. reinforce this relationship, for both actions. Nevertheless, using this noisy, non-optimal signal for learning relationships results in a performance improvement over the baseline algorithm. For example, in an environment with actions a ∈[..., turn_on_heating, turn_on_ac, turn_on_fan, no_op, ...], the action [turn_on_heating, turn_on_ac] would not lead to a successful state transition while the action [turn_on_ac, turn_on_fan] would, as would the partially successful action [turn_on_ac, no_op]. This models that there is a contradictory relationship between turn_on_heating and turn_on_ac and these are excluded by the signal.

The self-supervised signal for determining which action relationships to reinforce is based on a filter criterion applied to the trajectory dataset. A positive transition occurs when an action taken leads to a successful change in state, while a negative transition occurs when it does not. Although some environments provide an appropriate reward signal for positive transitions, in sparsely rewarded environments, rewards are often unavailable, making positive transitions a useful generic indicator for positive data points. By applying a filter that selects only records with successful state transitions, those data points with positive transitions are transformed into an auxiliary signal that reinforces the agent's relational structure.

The filtered records are reinforced with a similarity loss to induce relational structure in the action space. The state is fed to the actor to obtain logits which are used to predict actions for the state. The logits are used to compute both the PPO loss objective and the new relational objective. After a comparison of similarity losses, a relational loss based on the dot product (equation 5.1) (Kreyszig, 2006) preceded by z-score batch normalisation was adopted. The rationale behind using the dot product as a relational measure is that it captures the degree of similarity between actions, inducing similar features in the representation of the actions that have a high affinity for each other and are likely to result in positive transitions. The relational objective is thus based on the log sigmoid of the dot product of the logits (equation 5.2), with the sigmoid ensuring that the dot product is in the interval $[0, 1]$ and suitable for the log loss.

$$S_C(\mathbf{l}_1, \mathbf{l}_2) = \frac{\mathbf{l}_1.\mathbf{l}_2}{||\mathbf{l}_1||.||\mathbf{l}_2||} \tag{5.1}$$

$$L^{REL} = \hat{\mathbb{E}}[-log(\sigma(\mathbf{l}_1.\mathbf{l}_2))] \tag{5.2}$$

where $\mathbf{l}_1$ and $\mathbf{l}_2$ refer to the raw logits for each action prediction after batch norm and $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

The modifications to the update section of the multi-discrete PPO (described in Algorithm 6) are shown in Algorithm 7.

Algorithm 7: Update section of Multi-Discrete PPO Algorithm (Schulman et al., 2017; Huang et al., 2022) modified to support relational component.

```
// Update policy
For each epoch:
    Shuffle buffer
    For each minibatch:
        Calculate $L^{REL}$:
            • positive_filter ← Filter records with positive
            transitions only
            • logits ← $\pi(s;\theta)$
            • action_logits = split(logits)
            • $L^{REL}$ ← similarity_loss(action_logits[positive_filter])
        // Add ppo, value, entropy losses to relation loss
        $L = L^{CLIP} + L^{VF} + S + L^{REL}$ ... Eqn 5.3
        // Update policy and value networks
        $\theta \leftarrow \mathrm{Adam}(\nabla\theta L)$
        $\theta_v \leftarrow \mathrm{Adam}(\nabla\theta_v L)$
```

5.4.2 The relational auxiliary objective

Prior knowledge of relationships allows the actor to more effectively combine actions earlier in the training process, leading to faster convergence than the vanilla PPO algorithm. Figure 5.1 illustrates the placement of the relational module in the actor-critic model. The relational auxiliary module influences the training of the standard PPO algorithm and improves convergence times by shaping the actor network's representation to include a relational aspect. The relational loss, shown in Equation 5.3, positively reinforces action relations based on a self-supervised signal, which is incorporated into the standard PPO loss (equation 2.19). The relational module is attached to the head of the actor network as it is based on the actor's logits and affects the actor's weights, thus inducing relational structure in the agent's action representation. This is a common method used to add auxiliary signals to shape the actor or critic networks (Jaderberg et al., 2017),

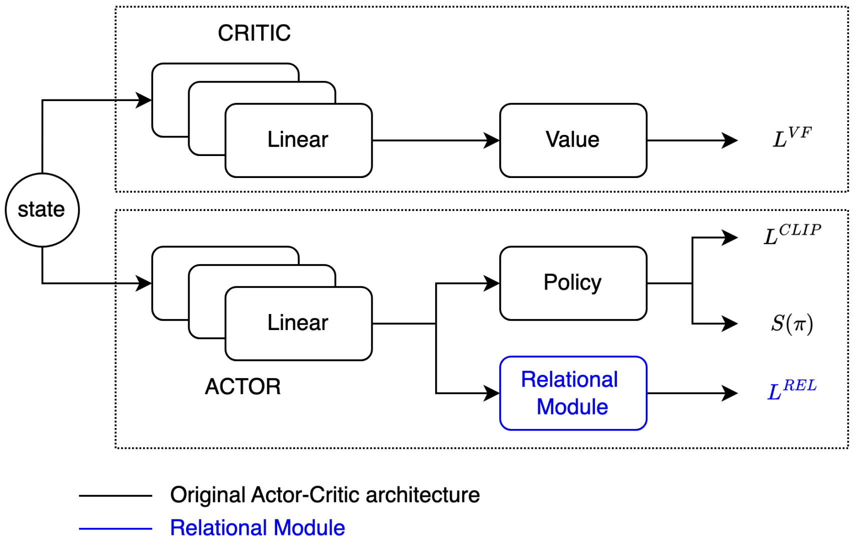


Figure 5.1: Relational auxiliary component fitted in an Actor-Critic model. The state feeds the Critic network (top row) and the Actor network (bottom row). The Actor's head splits to produce the clipped policy and entropy losses and, separately, the proposed relational loss.

also described in Chapter 2. The critic is left unmodified since the intention is to retain it as a separate network that provides strong direction to the actor, and because it is harder to directly involve the critic in the action structure without more modifications to the algorithm.

$$L_t^{CLIP+VF+S+REL}(\theta,\theta_v)=\hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta)-c_1L_t^{VF}(\theta_v)+c_2S\left[\pi_\theta\right](s_t)+c_3L_t^{REL}(\theta)\right] \tag{5.3}$$

The relational loss is added to the PPO loss with a hyperparameter coefficient that controls the scale of the loss.

5.4.3 Discussion

1. The approach proposed in this chapter is similar to the approaches of Wang and Yu (2016), Sharma et al. (2017) and Harmer et al. (2018) in adopting a multi-discrete action space to capitalise on relationships between actions.

The specific approach proposed makes use of auxiliary signals as described in Jaderberg et al. (2017) but not expert data in contrast with Harmer et al. (2018) and no factoring or embedding of the action space in contrast to Sharma et al. (2017), Chandak et al. (2019) and Tennenholtz and Mannor (2019). Like Zambaldi et al. (2018), the proposed approach modifies the architecture of the agent to include a relational component. In this chapter a self-supervised relational mechanism is trained alongside the agent, unlike the Actor Critic relational attention mechanism used by Zambaldi et al. (2018). The inspiration for a mechanism dealing with relationships was drawn from the theory of Xu et al. (2021) that says if the network is aligned with the algorithm required to solve a task, the network will be able to generalise and extrapolate well. The relational component added to the PPO agent serves to align the agent with the relational nature of the task.

2. There are several ways to provide reinforcement signals in sparsely rewarded environments, and the approach described in this chapter is just one example. Other options for generating intrinsic rewards include using curiosity-based approaches (Pathak et al., 2017) or prediction error signals (Jaderberg et al., 2017). Pseudo-rewarding bottlenecks (Parisi et al., 2021), exploration bonuses for rarely visited states (Bellemare et al., 2016; Raileanu and Rocktäschel, 2020), and oversampling positive data points (Schaul et al., 2015c) are other approaches that have been adopted in sparsely rewarded settings. The choice of which filter to use as a reinforcement signal should depend on the specific environment or task being trained. The positive transition filter used in this chapter is available in most multi-discrete RL environments because transitioning to new states is a common feature and may be viewed as a metaphorical lower bound on the amount of information that can be extracted from any RL trajectory.

5.4.4 Implementation Details

5.4.4.1 Environment

A toy environment was designed to test the relational model. The 10x10, four-room, multi-discrete grid-world environment shown in Figure 5.2 has a relational action space with 10 primitive actions that are combined to form multi-discrete actions. In this domain, the agent must select a tuple of two actions at each timestep. To generate a diverse set of tasks, the environment is configurable by randomly moving the walls, start, goal, and special states, while the action space remains static. The environment is sparsely rewarded, providing only a reward of 10 if the agent reaches the goal state. There are no other rewards or penalties. The state is purely the co-ordinates of the agent in the grid and is one-hot encoded.

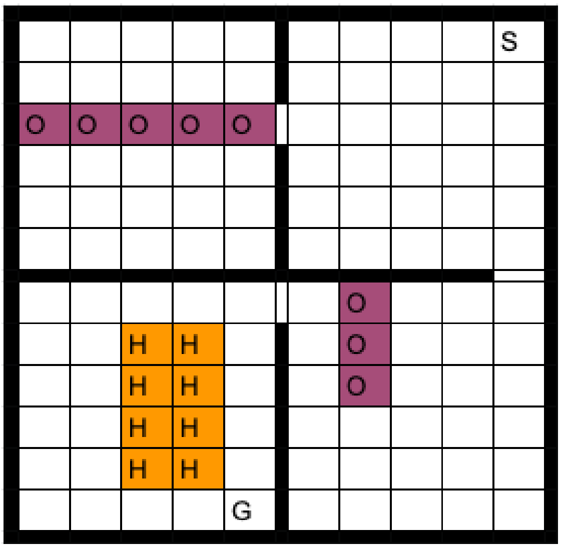


Figure 5.2: Grid-World requiring multi-discrete actions on every step, where S-Start, G-Goal, H-Holes, O-Obstacles and gaps in the walls indicate doors

5.4.4.2 Actions

The environment features five types of actions mapped to different categories of state, as illustrated in Table 5.1, where the set of primitive actions is listed in Table

Table 5.1: Special states and corresponding action combinations

State	*Action Set*	*Category*
Normal	Up (U), Down (D), Left (L), Right (R)	Navigation
Obstacle	Hack (H), Jump (J)	Obstacles
Doors	Open (O), Enter (E)	Doors
Holes	Fill (F), Jump (J)	Holes
Any	No-Op (N)	None

Table 5.2: The full set of primitive actions with action index

0	1	2	3	4	5	6	7	8	9
U	D	L	R	O	E	J	H	F	N

5.2. A multi-discrete action tuple is composed of combinations of primitive discrete actions, whose effectiveness varies. Most of the states in the task (room) are classed as normal and traversed using navigation actions. There are 3 special types of state, viz. obstacles, doors and holes where only the corresponding synergistic action combination results in successfully changing state, for example, taking the multi-discrete action combination [Open, Enter] at door states results in the agent moving through the door. Likewise taking the action combination [Fill, Jump] results in the agent jumping over the hole states.

There are no constraints on action selection, so any two actions may be combined to form a tuple or multi-discrete action. If a navigation action is selected with any of the special actions, the environment will perform the navigation action if possible. While any action can be combined with any other action, including itself, some combinations may be redundant or inefficient. Where [Up, Down] would result in the agent standing still and is normally a redundant action, it has a different effect when applied against a wall, where the Up action does nothing but the Down action moves the agent away from the wall. This makes it more difficult for the agent to work out the action relationships within the action tuple and generates noisy data. Finally, to test the agent's ability to learn shared combinations, the hole and obstacle states share the Jump action. The agent must learn two relationships associated with the Jump action, adding an element of complexity to the relational representation.

5.4.4.3 RL Algorithm

The PPO algorithm (Schulman et al., 2017) was used in this chapter but any similar actor-critic algorithm could be used. PPO was selected because it meets the online learning requirement of this chapter to investigate relational learning in an online setting and, importantly, it is less sensitive to hyperparameter tuning than other algorithms (Schulman et al., 2017). A multi-discrete implementation provided by (Huang et al., 2022) was used. This same version was modified to incorporate the relational model and provide a like-for-like comparison.

5.5 Experiments and Results

This section outlines the experiments carried out to test the relational auxiliary model in tasks with increasing levels of difficulty to demonstrate that the relational module is most effective in complex tasks where base PPO struggles to learn. Analysis of the action representation is compared with base PPO to show the contribution of the relational module to the shaping of the representation. The learning dynamics of the base and relational models are compared in a final section that tests the generalisation ability of each model.

The baseline model is the vanilla multi-discrete PPO referenced in section 5.3. In the experiments that follow, both the base and relational model were trained on each task for 10 randomly generated seeds and the average convergence performance was compared. The random seed impacts a number of areas in training deep learning models, including weight initialisation, sampling of batches, the order of data and sampling of actions from policies. Refer to Patterson et al. (2023) for more complete guidelines on training RL models.

5.5.1 Relational Auxiliary Task Setup

The environment is configurable and able to generate random tasks programmatically simply by varying the start and/or goal states and controlling the position and number of each of the special states as per Table 5.1. Using these parameters, 3 different categories of tasks were defined:

- easy, featuring two special states from Table 5.1, for example doors and holes
- intermediate, featuring all three categories of special states from Table 5.1
- hard/complex, featuring all three categories of special states and one less door.

For examples of each type of task, see Figure 5.3.

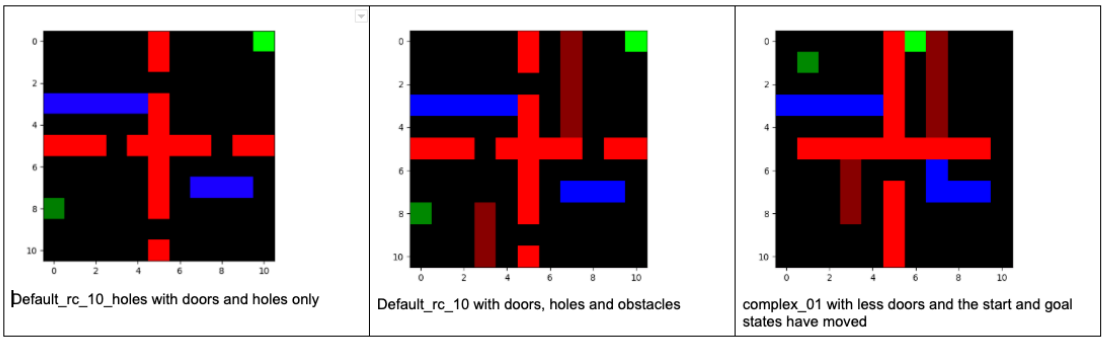

Default_rc_10_holes with doors and holes only
Default_rc_10 with doors, holes and obstacles
complex_01 with less doors and the start and goal states have moved

Figure 5.3: From left to right: easy (with 4 doors and holes), intermediate (with 4 doors, holes and obstacles), hard tasks (with 3 doors, holes and obstacles; the agent start location is furthest from the goal)
Legend: Walls - bright red; obstacles - pale red; start state - pale green; goal state - green; holes - blue

5.5.2 Results - Relational Model Performance

The results show that the relational model is able to solve the complex task faster than the baseline model on average. Given that both models solve the easy and intermediate tasks (section 5.5.2.1) at a similar rate, the result for the complex task (section 5.5.2.2) suggests the extra relational information proved useful to the agent's ability to converge.

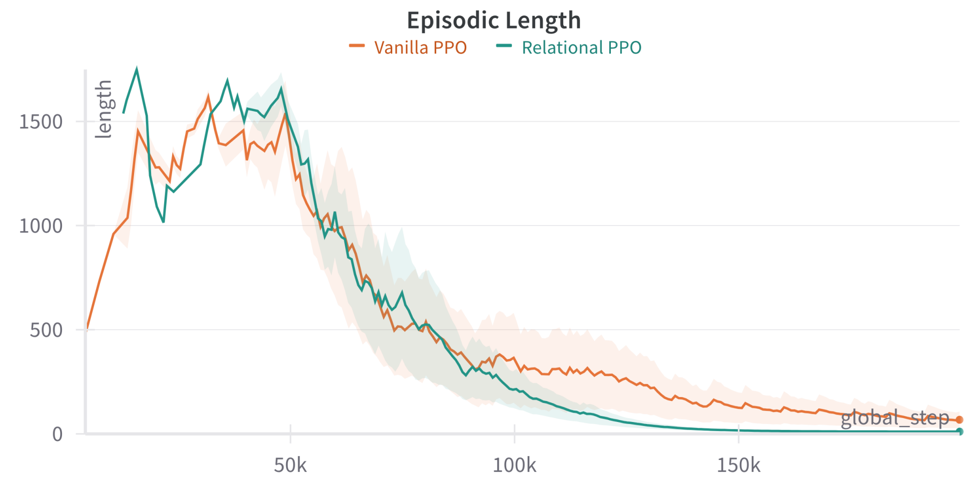
5.5.2.1 Easy and Intermediate Tasks

In this experiment the relational model and the baseline PPO model were trained over the easy task defined in section 5.5.1. Each model was run for 10 seeds and the mean result across all seeds, with standard deviation, is plotted. The episodic length and return plots illustrate how fast the model converges, indicated by the episode length approaching the optimal length (14 steps) and the episode return approaching the maximum of 10. The length and return together determine whether a run is efficient and therefore whether the agent has learnt the relationships between actions effectively enough to solve the task optimally. An agent can achieve a high return but take a less optimal route with a longer episodic length, so both measures are used to indicate optimal performance. Of the loss plots, the value loss is most illustrative of how well the model is performing, with the loss expected to approach 0.0.

Figure 5.4 shows that the relational model only slightly out-performs the vanilla PPO over the 10 seeds in the tasks with easy complexity. In the episodic length plot the trajectory length for both models reduces to close to optimal length however the relational plot descends earlier and achieves a better length on average. The difference in the return plots is less significant. On average the vanilla model does not converge as evidenced in the value loss plot in Figure 5.5, which shows the vanilla PPO model average value loss remains higher than the relational model. Nevertheless, this is a very easy task with only two special actions and not a particularly good test of the relational model's capabilities.

In the intermediate task not all seeds converge for both the relational and the vanilla models (Figure 5.6), unlike in the easy task, with both algorithms performing very similarly. While the easy task was run for a total of 200k timesteps, the intermediate tasks were run for a total of 250k timesteps.

The impact of the relational component is less clear in the intermediate task than in the easy task, however the key question is whether the relational component



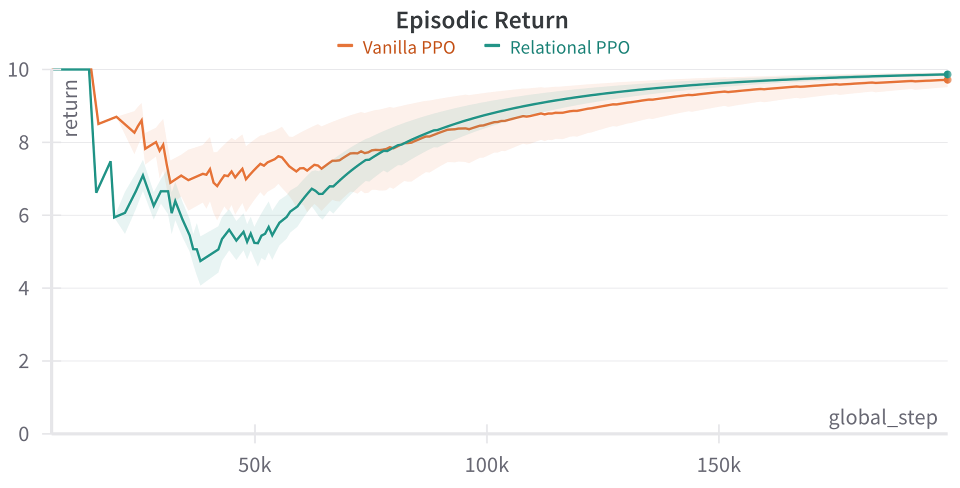


Figure 5.4: Easy Task - Episodic length and Returns: comparison of relational (orange) and vanilla ppo models (green) over 10 seeds.

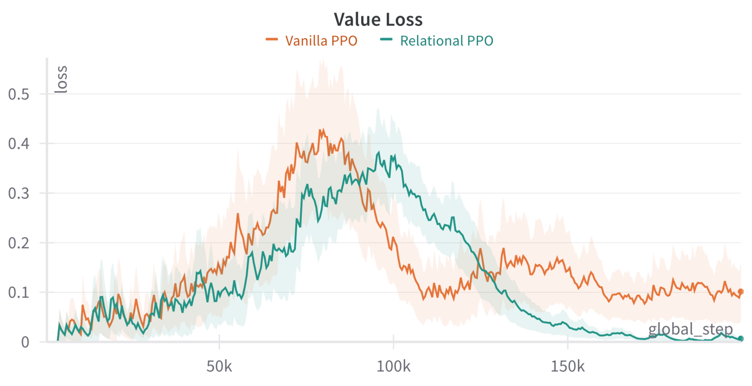
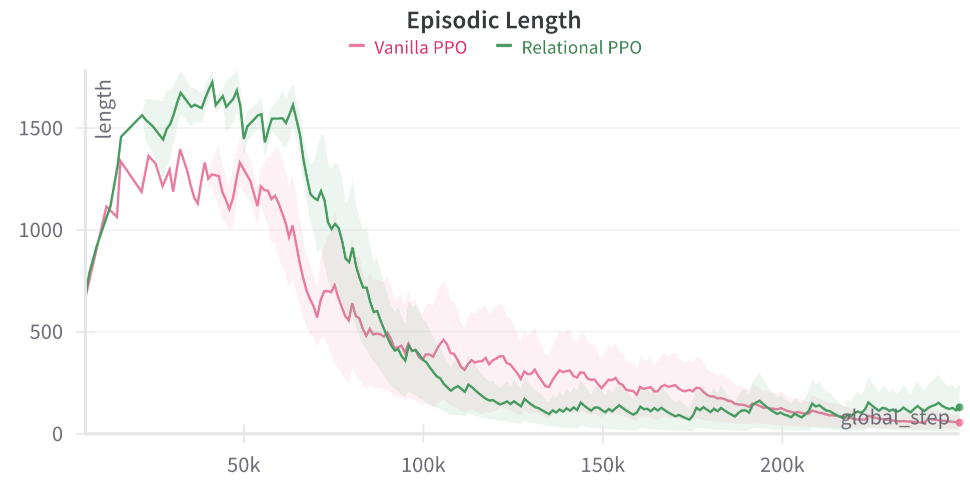


Figure 5.5: Easy Task - Value loss: comparison of relational (green) and vanilla ppo models (orange) over 10 seeds



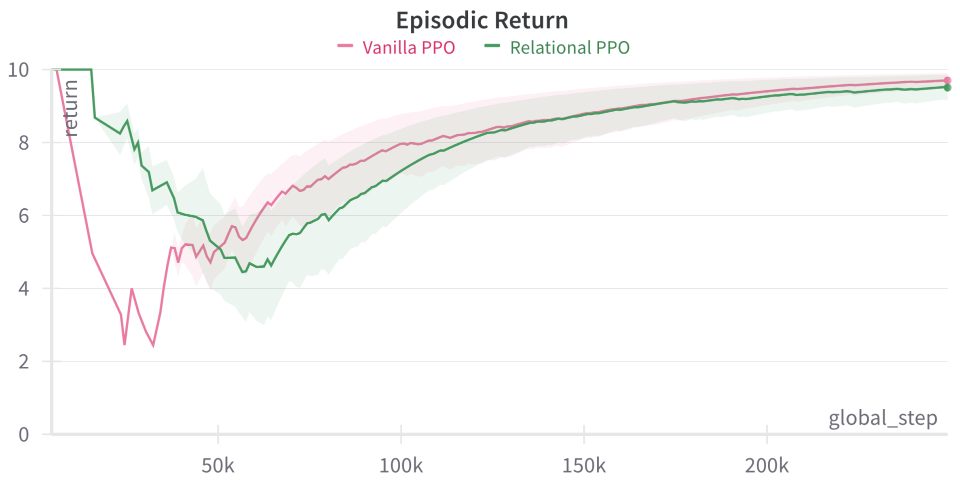


Figure 5.6: Intermediate Task - Episodic length and Returns: comparison of relational (green) and vanilla PPO models (pink) over 10 seeds

has a positive impact on convergence when task complexity increases.

5.5.2.2 Hard/Complex Task

For the easy and intermediate tasks, the benefits of using the relational model are not immediately apparent. This is likely due to the relative simplicity of the tasks. Vanilla PPO is already effective at solving these tasks, suggesting that the agent had enough time to learn the best action combinations through the normal exploration process of PPO. The focus of this section, however, is to examine the effect of the relational model on tasks that are harder for vanilla PPO to solve. This experiment asks the question "can the agent benefit from knowledge of which actions relate well to one another?"

In this experiment the agent is faced with a complex task as per Figure 5.3 that requires the agent to learn all the special action combinations described in Table 5.1. The goal state is boxed in between obstacles and a wall (Figure 5.7), making it unreachable without knowing the correct action combinations. Furthermore, the agent is initially trapped behind a different type of special state, making it impossible to complete the task without learning all special actions. Finally, one door is removed to increase the time required to reach the goal. For this task, the total number of training timesteps was increased to 300k.

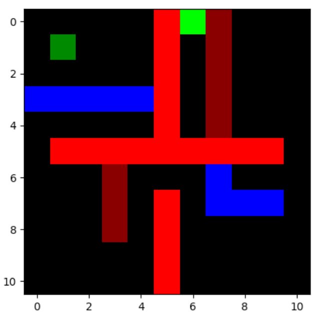


Figure 5.7: Complex task. Walls - bright red; obstacles - dark red; start state - dark green; goal state - bright green; holes - blue

In Figures 5.8 and 5.9 the performance of the vanilla multi-discrete PPO vs the relational multi-discrete PPO algorithms on the complex task is compared.

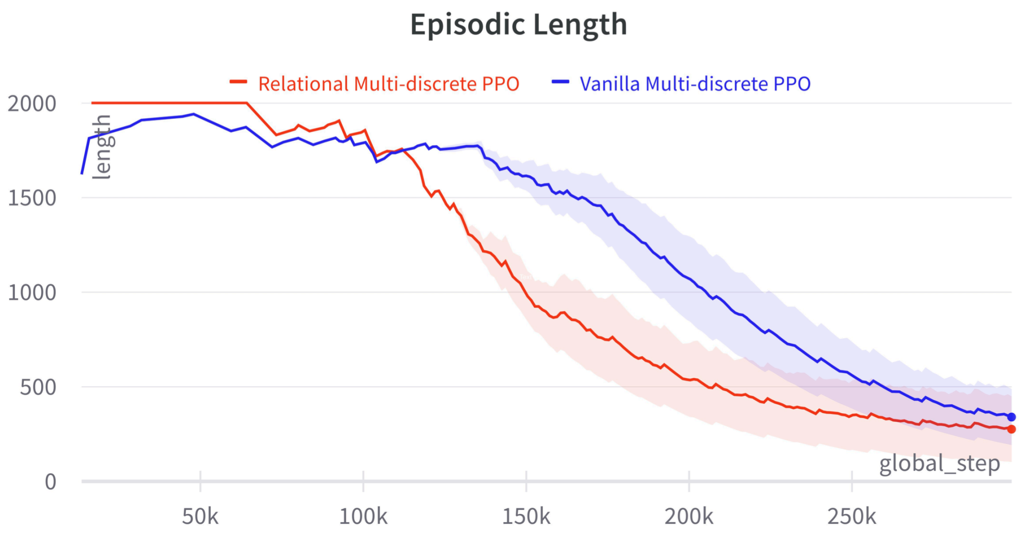


Figure 5.8: Complex Task - Episodic length: comparison of relational (red) and vanilla PPO models (blue) over 10 seeds

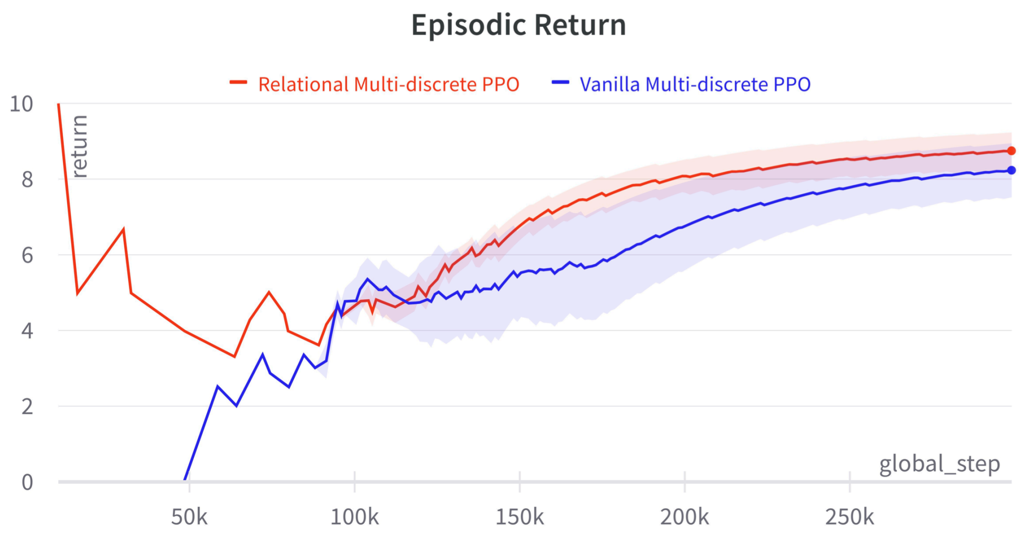


Figure 5.9: Complex Task - Episodic returns: comparison of relational (red) and vanilla PPO models (blue) over 10 seeds

In both plots the relational model learns faster than the vanilla model on average.

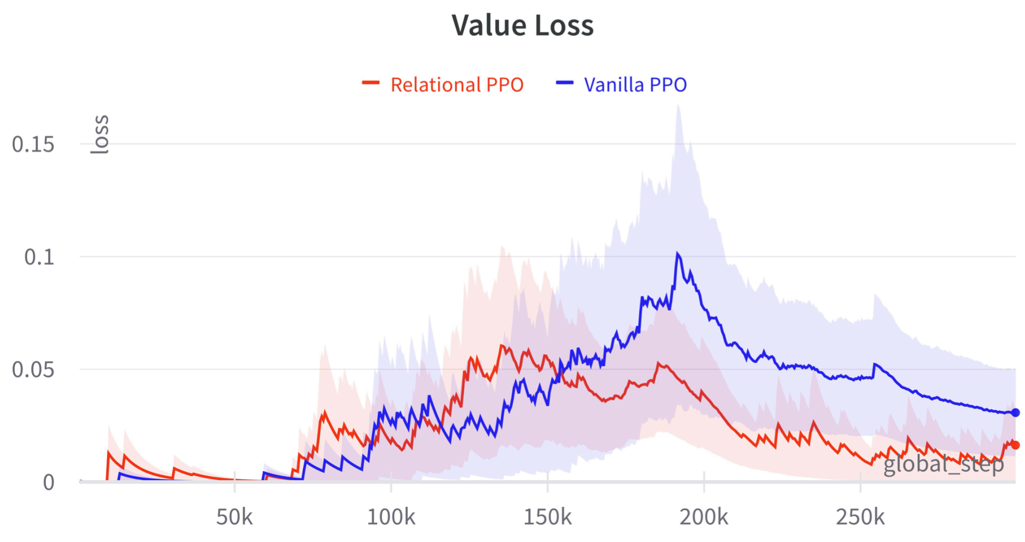
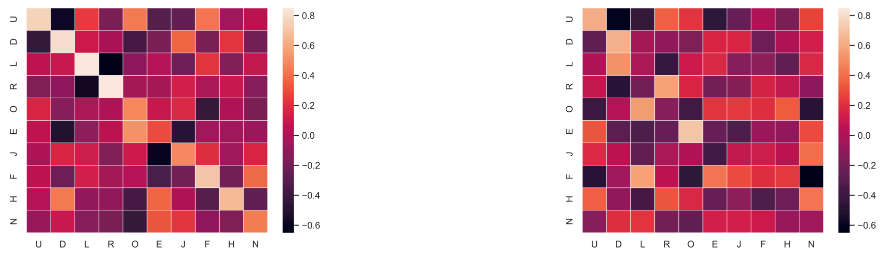


Figure 5.10: Complex Task - Value loss: comparison of relational (red) and vanilla PPO models (blue) over 10 seeds

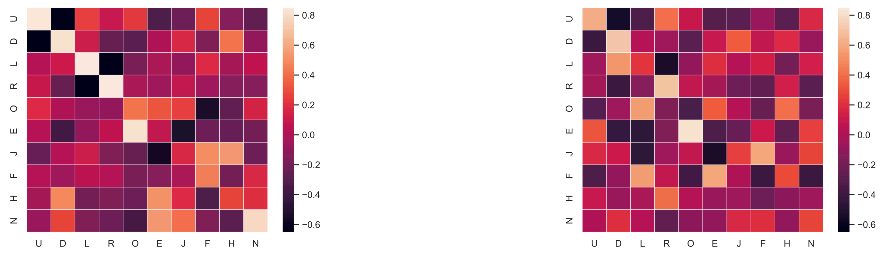
Figure 5.10 looks at the averaged value function loss for the two models. The value or critic loss is a good indication of whether the model converged. Overall the relational model appears to perform better than the vanilla PPO, with the relational model converging faster compared to the baseline

5.5.3 The Action Representation

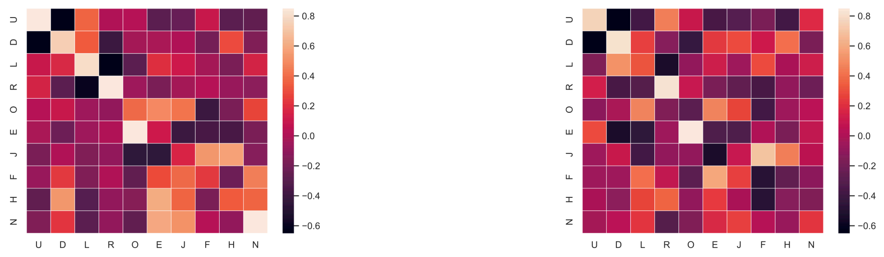
The underlying hypothesis in this chapter is that the relational model influences the shaping of the action representation in a way that is beneficial to the relational nature of the actions in this domain. The analysis that follows looks more closely at whether the model actually promotes the learning of the relational structure and how this is useful compared to the vanilla model. First, the action representation was analysed for signs of the relational structure that might be expected in this domain. Next, the learning dynamics of the network was explored using Singular Value Decomposition (SVD) (Golub and Reinsch, 1970). Finally, the representa-



Update 300



Update 400



Update 500

Figure 5.11: Heatmaps for Relational PPO (left) vs Vanilla PPO (right) taken at various policy model updates (300, 400 and 500) show the relational model has already learnt more structure at update 300 than the vanilla model. Lighter squares indicate strong affinities between the set of actions.

tions from the relational and vanilla models were assessed for generalisability on a new complex task.

To assess whether the relational component actually impacts the outcome of the relational model, the actor's weights were analysed. The final layer of the actor or policy network may be analysed to determine how the action representation evolves during training (Saxe et al., 2013). The layer weights may be analysed to see if expected structure patterns have developed. Monitoring the development of this layer over multiple model updates gives some visibility of when patterns develop over time. The weights from the final layer with dimensions 128×20 is a mapping from the previous hidden layer to the action set $[a_i, a_j]$. Each of a_i and a_j has 10 possible discrete actions. A trained actor should have a representation for each discrete action represented in these weights.

One way to demonstrate relationships between the action representations is to use a similarity measure. The cosine similarity of the action representation tensors is plotted as a heatmap to highlight relationships in an interpretable way. The heatmap was captured at various updates during training to view the progression of relational structure learnt by the models. The plots in Figure 5.11 compare the relational and vanilla model's latent representation at updates 300, 400 and 500 (by which time both models converge). The figures show that the relational model discovers useful structure earlier than the vanilla model (around update 300) while the development of similar structure in the vanilla model forms later (around update 500).

A closer inspection of the heatmaps reveals the relational model displays a strong preference for the navigation action combinations [Up, Up], [Down, Down], [Right, Right], [Left, Left] and a very negative response for [Up, Down] and [Right, Left] actions very early on. These responses make sense, especially [Up, Down] and [Right, Left], that would otherwise waste valuable learning time. The double action combinations are particularly strong early on but over time become less so as other navigation combinations become more useful. As both models converge, it is clear that the vanilla model does learn the required structures by the end of the run. This comparison serves to show the performance benefit of the relational component, viz. the early formation of desirable action structures by the actor network.

5.5.3.1 Learning Dynamics

The emergence of early relational structure can also be observed through an analysis of the learning dynamics of the network using SVD as per Saxe et al. (2013). This method involves accumulating the network weights and performing SVD over the cumulative weights, producing a set of orthogonal vectors that represents disentangling a deep and wide neural network into multiple deep narrow linear networks (Saxe et al., 2013) and a matrix of singular values (Σ), as shown in Equation 5.4. The singular values, or modes, are reflective of the key concepts learned by the network during training. If the weights are collected for each training update, the modes can be plotted to show the progression of learning concepts over time.

$$W^{Tot} = U\Sigma(t)V^T \tag{5.4}$$

This approach is applied to the actor network, where the weight matrices are accumulated during training with final dimensions 121×20, split into two 121×10 matrices, one matrix per action in the multi-discrete tuple. SVD is applied and Figure 5.12 plots the singular values for both the relational (left) and vanilla models (right). The plots for the relational model show the development of 10 singular values or modes, illustrated in different colours. The modes develop at different rates and converge, flattening off from around update 400. In the vanilla model, a similar separation of the modes is observed but convergence is less developed, with less flattening off towards the end of training. The plots show that the relational model learns earlier and stabilises faster than the vanilla model. Unfortunately it is not possible to gauge which actions develop first from this method, as Saxe et al. (2013) demonstrate in their work, for example, would navigation-related actions be learnt first, followed by door-related actions? This type of analysis is left for future work.

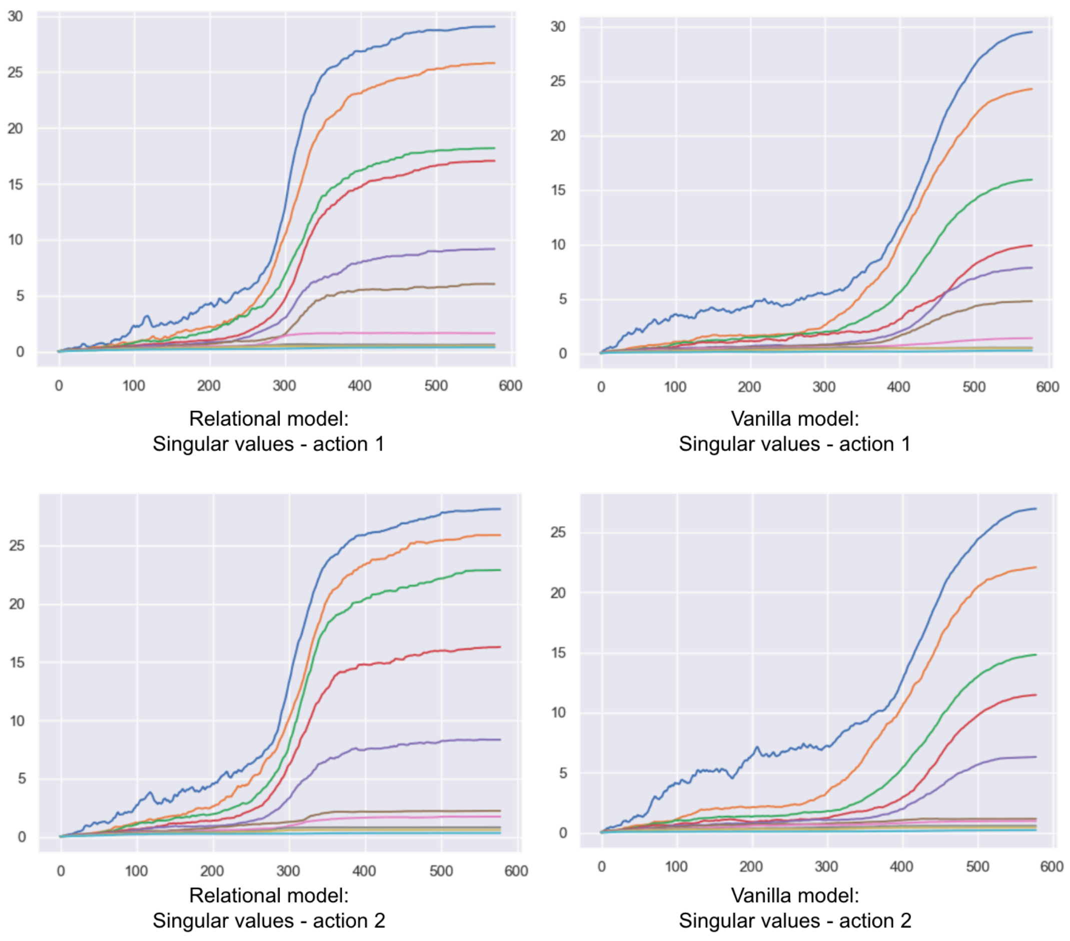


Figure 5.12: Singular values plotted for action representations over training period. X-axis is the update number, Y-axis is the singular value or mode

While the return and value plots for the vanilla model (Figures 5.9, 5.10) indicate that the network has converged, the SVD plots provide further insight into whether it really has. The difference between the relational and vanilla SVD plots suggests that the relational model has learnt the representation but the vanilla model has not, as it is only just starting to stabilise. The next set of tests aim to determine if this is the case.

5.5.3.2 Generalisation test

Both models were tested on a new particularly complex task to determine if the representations learnt by the models were developed enough to generalise well. The final layer of weights from each model was extracted and ported to a new vanilla model. After being initialised with each model's weights the new model was trained on the complex task to determine if the "injected" representation provided any advantage and improved convergence. The plots in Figure 5.13 show that the original vanilla (dark blue) and relational (deep red) models fail to solve this task from scratch. The light blue and violet plots are from the new vanilla and relational models, respectively, retrofitted with the transferred representations. The retrofitted relational transfer (violet) plot is the only successful plot, showing almost zero-shot transfer in the episodic return. The retrofitted vanilla transfer plot (light blue) does not show similar performance. In this experiment, the representation learned by the relational model performs better than the representation from the vanilla model.

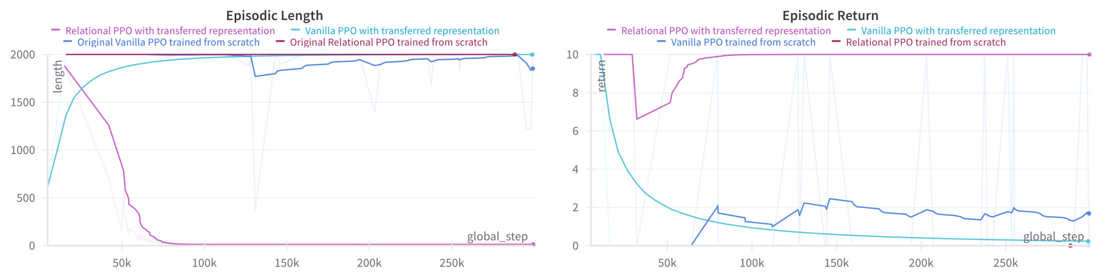


Figure 5.13: Testing the generalisability of the trained representation on a new complex task. Dark blue (vanilla) and red (relational) are the from-scratch runs training over the new task. Light blue (vanilla) and violet (relational) are the runs using the transferred representation.

A second complex task was designed to be sufficiently different from the previous tasks to test that the network is not simply benefiting from familiarity of state dynamics. The results in Figure 5.14 show once again that the relational representation is more successful at generalising to the new task and converging.

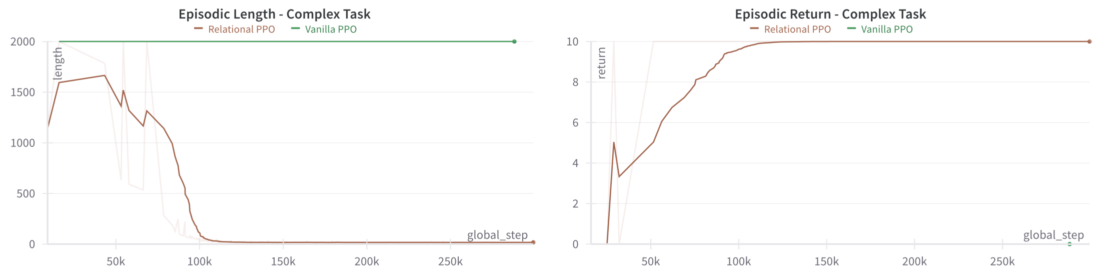


Figure 5.14: The vanilla plot (green - based on the representation from the vanilla model) does not perform well while the relational (brown - based on the representation from the relationally-trained model) performs well and converges

The results indicate that there is a benefit to using the relational representation versus the vanilla representation in terms of generalising to new domains.

5.6 Conclusions

This chapter introduced a novel self-supervised relational auxiliary module that shapes action representations by optimising a relational loss trained alongside the multi-discrete online PPO algorithm. A range of simulated tasks from a multi-discrete grid-world environment was used to test the effectiveness of the proposed online relational model. Results showed that the relational module improved the performance of the algorithm compared with a baseline. Analysis to determine if the multi-discrete, concurrent action model learnt a relational representation was performed utilising heatmaps of the action space, showing the emergence of structured representations. Furthermore, training dynamics analysis revealed that the relational model converged faster than the baseline version of the algorithm. Generalisation tests demonstrated successful transfer of learned representations to new tasks. The results indicate that the self-supervised auxiliary relational module could provide an advantage to agents in a multi-discrete action space that supports relationships.

While the relational model shows promise, it is limited by the noisy self-supervised signal used. Ideally the model needs to be "smarter", able to distinguish trends and patterns without being directed on how to achieve this. The next chapter looks at attention mechanisms, and the Transformer model in particular, as an architecture that can handle massively complex processes and decision making.

Chapter 6

Action Structure in Decision Transformers

6.1 Introduction

In a multi-discrete action space, the agent must learn to select multiple discrete actions per timestep. Methods for solving environments with multi-discrete action spaces often convert multi-discrete actions to a single action representation, seldom looking for relationships between the individual discrete actions. In environments such as Minecraft, Starcraft II and Vizdoom with multi-discrete actions spaces, relationships between actions exist and could be exploited to more closely model human behaviour. Human players are much more likely to select individual actions based on which action combinations make sense in the current state. This chapter is motivated by the desire to model expert behaviour using a model capable of extracting and utilising relational action insights when addressing environments with multi-discrete action spaces.

Transformers by Vaswani et al. (2017) are very effective at modelling sequential data due to extensive use of the self-attention mechanism (Bahdanau et al., 2014) and the residual backbone (Elhage et al., 2021). These features give Transformers the ability to extract relational information between positions in sequential data. At the time of this dissertation Transformer-based models are still a very active area of research (Wikipedia contributors, 2023b; Han et al., 2023) having revolutionised the field of LLMs, exhibiting expert behaviour at translation, semantic understanding and generative tasks. More generally LLMs are an example of foundation models, i.e. machine learning models trained on vast quantities of data, often using self-supervised learning methods, making them adaptable for disparate tasks. The success achieved by Transformers in language has led to similar attempts in vision (Dosovitskiy et al., 2021) and in RL, where Transformers have been used in multiple contexts from modelling the value or policy functions (Agarwal et al., 2023; Li et al., 2023a) or contributing to the task in another way (Zambaldi et al., 2018). Subsequently Transformers were applied to offline RL (Chen et al., 2021; Janner et al., 2021) and used to model offline sequential trajectory data in a self-supervised manner. Offline RL (Levine et al., 2020) is a branch of RL that extracts behavioural policies from offline data without having further interaction with the environment and is more closely aligned with supervised learning than other RL approaches. Transformers were applied to offline RL largely because of its ability to model long term dependencies in sequential data, as demonstrated by LLMs. This ability to derive relationships from sequential data makes the offline RL Transformer an interesting framework for the relational action problem. While both the Decision Transformer (Chen et al., 2021) and the Trajectory Transformer (Janner et al., 2021) are offline RL approaches that learn agent behaviour from both optimal and sub-optimal offline data, the Decision Transformer was selected for this chapter. The reasons are twofold: first, it follows a model-free RL approach that is less complicated to adapt than the Trajectory Transformer's model-based approach and, second, it has been more extensively adopted by the RL research community with findings that this dissertation could leverage.

Much of the literature relating to Decision Transformers tends to focus on methods for prompting (Xu et al., 2022; Jiang et al., 2023), masking (Carroll et al., 2022; Wu et al., 2023a; Liu et al., 2022) and in-context learning (Jia et al., 2023; Laskin et al., 2023) for RL. Prompts are like hints and in LLMs prompts are used to communicate with a trained model to steer or guide its behaviour towards a desired outcome without updating the model weights, i.e. in-context. The prompt can be regarded as an instrument for extracting pre-existing knowledge from the LLM. In the Decision Transformer the return-to-go, the calculated expected return at a given state, is used as a simple scalar prompt but as prompting is the key to drawing out behaviour, multi-step prompts (Xu et al., 2022; Laskin et al., 2023) and in-context learning are highly desirable avenues of research relating to Decision Transformers. An example of a multi-step prompt could be a selection of images from the desired task, such an egg, an egg-cup, an egg in the egg-cup. The information is usually sufficient to characterise the task, like a signature. Trajectory stitching (Emmons et al., 2022; Brandfonbrener et al., 2022) is another key area of research effort, referring to the ability to generate trajectories by stitching from optimal, sub-optimal and multi-task data. Masking (Carroll et al., 2022; Wu et al., 2023a; Liu et al., 2022) trains a model by first masking some elements then training the model to predict the missing elements, resulting in the model more closely matching the dataset and therefore features more enhanced generation capabilities.

Research in RL-style foundation models is fairly new as of the time of this dissertation however, with only a few papers that look at environments with complex state and action spaces that might require non-standard tokenisation of trajectory data. A tokeniser converts input data, potentially multimodal data of varying dimensions, into a tensor of uniform dimension, in preparation for training a Transformer-style model. In the Decision Transformer tokenisation is at the mode level, with one token per mode per timestep, such that a multi-discrete action would typically be represented by a single token. It is not uncommon for the tokeniser to be a neural network that compresses higher dimensional data into an embedded feature representation. Mathieu et al. (2023); Wang et al. (2023); Reed et al. (2022); Jiang et al. (2023) work with complex environments with multimodal trajectories or complex state or action spaces. Mathieu et al. (2023) generate actions auto-regressively in the AlphaStar model where the action space itself is multimodal. Wang et al. (2023) take an entirely different approach to actions by using a skill library for MineCraft. The library turns skills (collections of actions that can perform a task) into Application Programming Interface (API) calls, a construct used in application development. Abstracting away the action detail allows for high level planning and control and makes sense in real-world foundation models, of which Gato by Reed et al. (2022) is another example. In the Gato approach, discrete and continuous actions, images, text and other modalities are tokenised individually, then combined, with the focus more on how to combine multimodal data in real-world scenarios. Finally, in VIMA, Jiang et al. (2023) expand multimodal data in a robotic environment, including visual goals, visual constraints, one-shot demonstrations and textual grounding inputs, for the purpose of more effective prompting. VIMA uses object detection to monitor and learn relations between objects that it tracks over trajectories. The modalities in these works are novel and complex however the focus is on the multimodal prompting of the robot arm and not the specific treatment of the action space.

Prior work has demonstrated the potential complexities in the state-action space of environments trained using Transformer models but also reveal there is no established way to manage inputs and modalities at this time. In this chapter, an offline RL sequential modeller known as the Decision Transformer (Chen et al., 2021) is trained on a multi-discrete action space in the ViZDoom environment (Kempka et al., 2016). Transformers (Vaswani et al., 2017) are very effective at learning relationships between tokens, traditionally between words in language modelling (Brown et al., 2020; Devlin et al., 2019). The hypothesis is that Decision Transformers would therefore be capable of modelling relationships between individual discrete action tokens in multi-discrete action spaces. The proposed approach considers techniques for exploiting the relational capability of a Transformer, while training over offline RL data, to extract relational information from the action space.

Multi-discrete actions are tokenised to encourage state-action and action-action relationships prior to being processed by the Transformer. This is achieved by first decomposing multi-discrete actions, then tokenising actions with states, resulting in multiple action tokens per multi-discrete action. The approach is referred to as Multi-State Action Tokenisation (M-SAT) for the duration of the chapter. The expanded tokenisation is compared with a Decision Transformer trained using the more typical single action token, on the ViZDoom (Kempka et al., 2016) multi-discrete action scenario, Deadly Corridor. The application of tokenisation configurations for multi-discrete action spaces, for the purpose of increasing action awareness, is novel as far as the author is aware. Model performance is compared by evaluating the robustness of each model and how well it solves the ViZDoom scenario. When comparing the overall training performance of the models results show that the increased tokenisation of M-SAT performs consistently better compared with the single token model and the methods used for ablation. The models are further interrogated at the attention head level. Analysis of the attention heads to assess whether the additional action tokens are actually used and how often, using mechanistic interpretation (Elhage et al., 2021) techniques, reveals that the attention heads do focus on the additional tokens, however the Transformer is a complex model to interpret so it is difficult to establish a concrete causal link with the performance improvements observed. Nevertheless the results demonstrate that M-SAT generally performs better than the single-action token Transformer and suggests the attention mechanism of the Transformer is able to establish more connections as a result of the additional tokenisation. The injection of state-action information in particular demonstrates that the Transformer benefits from additional relational information.

The specific contributions include:

- novel tokenisation of multi-discrete actions (M-SAT), highlighting both action-action and state-action relationships in a Decision Transformer model to improve the visibility of these relationships to the attention mechanism
- creation of more mixing opportunities to leverage Transformer relational abilities
- demonstration of the M-SAT approach in a ViZDoom multi-discrete environment
- use of mechanistic interpretation methods to interrogate the internal workings of the M-SAT Decision Transformer approach.
- ablation experiments performed including removing position encodings, removing state-action relationships and adding more position encoding for actions

The remainder of the chapter is organised as follows. Section 6.2 provides some background material for the Decision Transformer and is followed by Section 6.3 with related literature. Section 6.4 presents the proposed M-SAT approach, i.e. multi-token modification to the Decision Transformer and comparative approaches and includes implementation details. Section 6.5 presents details of the experiments and results. Section 6.6 is devoted to conclusions.

6.2 Preliminaries

This section describes the application of GPT Transformer models to RL, learning from multimodal, offline, RL trajectory data to generate decisions (actions), viz.

the Decision Transformer (DT) (Chen et al., 2021). Refer to section 2.10 for details of the Transformer model.

6.2.1 Decision Transformers

The Decision Transformer (Chen et al., 2021) is a Transformer adapted to train over offline trajectory data generated by an RL algorithm such as PPO or DQN, rather than textual data. It is an offline RL method trained entirely on pre-generated, often expert data, without any further interaction with the environment during training. Sometimes the data is collected from human or other sources. However the data is generated, this is all the offline RL algorithm has access to during training which means it is unable to receive corrections or feedback from the environment, unlike in online RL. The premise behind the Decision Transformer is that typical RL trajectory data is sequential data, similar to textual sequences in language modelling, and therefore a Transformer should be able to model this data.

The adaptations required to make the Decision Transformer effective as an RL agent are described. The word and sentence structure in language models is equated to the timestep and trajectory in RL trajectory data. The RL trajectory is defined as $\tau = \{(s_0, a_0, r_0, d_0), ..., (s_t, a_t, r_t, d_t), ..., (s_T, a_T, r_T, d_T)\}$ with T timesteps of state, action, reward and done-flag tuples, covering one or more episodes. The Decision Transformer uses the return-to-go (RTG) in preference to the every step reward, r_t. The return-to-go (RTG) is the expected sum of future rewards from a given state-action pair. It is calculated for each trajectory step, accumulating the returns from that step forward, to the end of the trajectory or window of evaluation. The trajectory used by the Decision Transformer is adjusted to $\tau = \{(s_0, a_0, \hat{r_0}), ..., (s_T, a_T, \hat{r_T})\}$ where $\hat{r_t}$ is the RTG and the done flags are subsumed by the RTG.

The equivalent to the “word” in RL is the multimodal timestep data comprising the state, action, reward and done flag for each step. The multiple modes typically require individual tokenisation for each mode with the tokeniser chosen to match the type of data, for example, if the state is a pixelated image then a CNN tokeniser is used; discrete actions and RTGs would be mapped using an embedding matrix or an MLP. After tokenisation each mode is represented by a fixed length vector of uniform dimension known as the model dimension. The timestep represents the position in the sequence and is encoded and added to each of the embedded modes to provide positional information; this effectively stamps all modes (i.e. state, action and RTG) in the timestep with the same time or position (illustrated in Figure 6.1). The injection of the sequential positional information is a factor that indirectly contributes to the parallelisability of the Transformer. The attention mechanism's ability to process all tokens in a context simultaneously makes the Transformer parallelisable, however pure attention does not consider the positions of tokens. By encoding the positional information into the data, the parallelisability of Transformer models is enabled, while retaining positional awareness.

The Decision Transformer is a return-conditioned, decoder based Transformer model (GPT model) that takes trajectory data as input and generates actions, mimicking an RL agent. The GPT model is adapted in a few ways to suit RL trajectories, viz. word tokens are replaced by multimodal trajectory tokens of states, actions and RTG. The RTG is derived from trajectory data and used as the prompt. During inference, the model receives an initial RTG prompt and an initial environment state from which it generates the next action to send to the environment. The generated action is appended to the sequence of RTG and state and the process repeats, generating the context for the next call to the Transformer, where the RTG prompt is reduced by the environment reward on every step and the next state is obtained from the environment. The output of the Decision Transformer is a set of logits for every token in the context. Actions are generated from the mode prior to the action in the trajectory, namely the state logits. Logits for the other modes are currently not used in the standard Decision Transformer configuration. The state logits are also used to generate the action predictions for the calculation of the training loss, as illustrated on the top left of Figure 6.1, while the top right section illustrates the action generation process. In the multi-discrete action scenario, the state logits will be split into the multiple segments, one per action. Each segment supports a distribution, from which the associated action is sampled. Combining the actions generated from each segment produces the multi-discrete action.

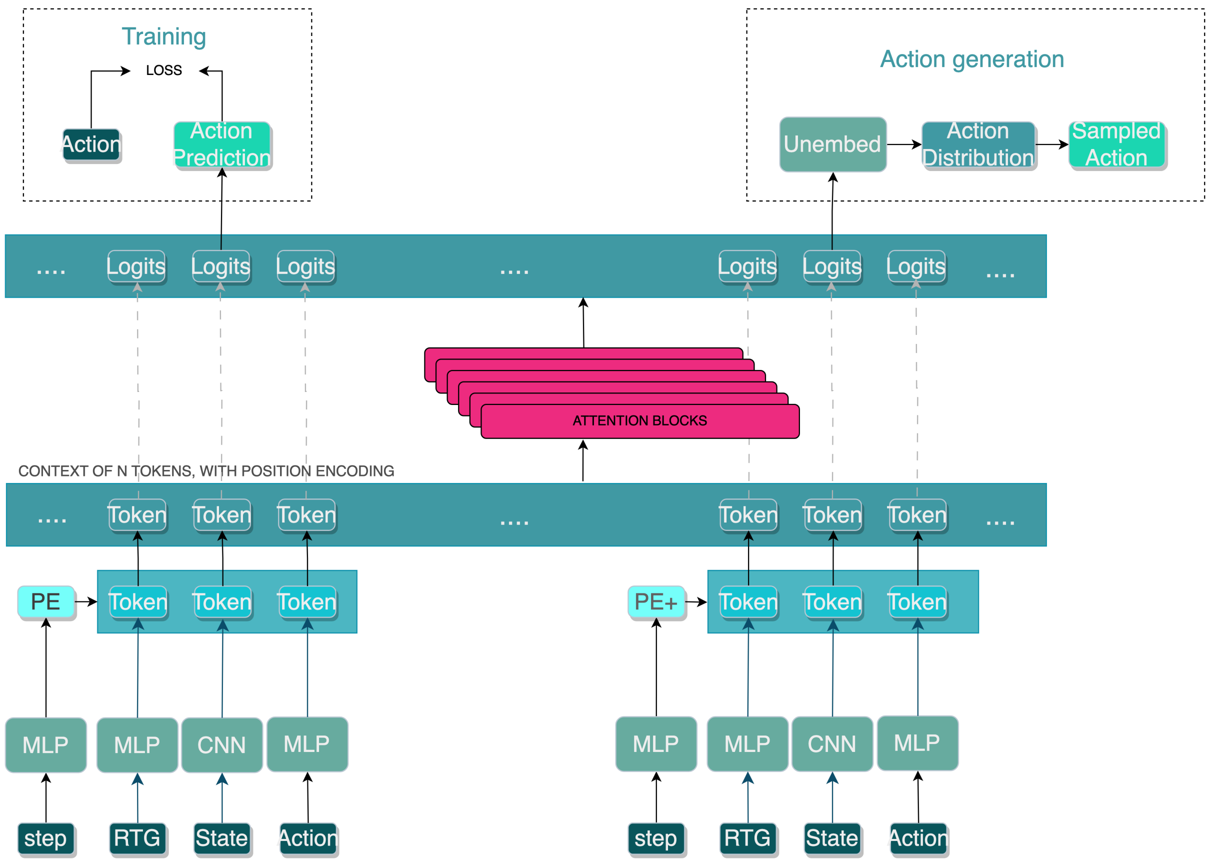


Figure 6.1: Flow of data through the Decision Transformer (Chen et al., 2021), from trajectory steps (timestep, RTG, state, action) to either a loss calculated during training OR to action generation. The position encoding is added to each of state, action and RTG tokens. The state logits are used to generate actions, which are next in the sequence. The feint grey lines indicate the transformation of tokens to logits in the context. The use of the state logits when generating actions or the training loss is also indicated.

6.2.1.1 Prompting

The main idea behind prompting is to extract knowledge already learnt from a trained Transformer in the most effective way, by setting an appropriate context. In language, this would mean phrasing the query to be very specific to the desired task. Prompting is used to extract behaviour from a trained model without updating the weights of the model and is viewed as an empirical science. In RL, a prompt refers to either a single or multiple steps of context provided to a Transformer model that triggers the model to respond in a particular way, i.e. in the direction of the prompt. Prompting is also known as conditioning, and the Decision Transformer is a return-conditioned model, using the RTG calculated from episodic trajectory rewards as the prompt. In the Decision Transformer, the scalar RTG represents the desired task. The prompt can be dynamic or static: in the Decision Transformer the RTG is a single, static prompt provided at the start of evaluation and is not recalculated after the generation step, however it is updated with the step reward obtained from the environment. In dynamic prompting (Xu et al., 2022; Jia et al., 2023) the prompt is multiple steps and may be updated or recalculated on every step (i.e. dynamically). For dynamic prompting to be possible, the prompt generating model also needs to be trained alongside the Transformer as unlike in language, there is no user provided prompt.

6.3 Related Work

The Decision Transformer (Chen et al., 2021) uses concepts from language modelling in the RL domain to train a GPT-style Transformer (Brown et al., 2020) to model and generate RL behaviour from trajectory data. The Decision Transformer therefore inherits causal, generative features that, when applied to trajectory data, models the data and generates RL behaviour. The Decision Transformer is very closely aligned with the GPT Language Model (Brown et al., 2020), more so than the Trajectory Transformer (Janner et al., 2021), that was published around the same time. The Trajectory Transformer uses the Transformer to learn a model of trajectory data and then uses the model for planning; this follows a model-based RL approach. By comparison, the Decision Transformer adopts a model-free RL approach using expert data, and is more comparable with imitation learning (Brys et al., 2015) techniques such as behaviour cloning, an offline RL method (Levine et al., 2020) where the agent learns how to behave using supervised learning concepts over expert data.

At the time of this dissertation one of the few papers that attempts to solve environments with complex state and action spaces is Alphastar by (Mathieu et al., 2023). The StarCraft II environment's states and actions are multimodal and are tokenised with either an MLP, residual CNN or Transformers (used as tokenisers). Actions are sampled auto-regressively, so each sampled action is embedded and then used to sample the next action. This work provides useful details on the complexities of adapting the Transformer for such a complex environment however no special tokenisation is performed on the input actions.

6.3.1 Prompting and In-context learning (ICL)

The prompt represents a task, which is less obvious in the Decision Transformer's scalar RTG than the multiple steps of goal state prompting in PromptDT (Xu et al., 2022) or Chain of Thought (CoT) (Jia et al., 2023). These works are variants of the Decision Transformer, training a dynamic prompting model alongside the Decision Transformer that can generate a prompt on each step during action generation. The multi-step prompts in PromptDT are trained from a subset of expert trajectories, effectively summarising tasks the agent will be exposed to. In CoT the prompt is a plan, a set of subgoals that are associated with solving the task. This paper is a step towards In-Context Learning (ICL) (as opposed to in-weight learning), i.e. gradient-free learning that takes place in the context passed to a Transformer model, where instead of inferring tasks from a short prompt, the model infers the task from the context. Laskin et al. (2023) attempt to show that the Decision Transformer model can learn how to be a trainer by training over contexts that are long enough to demonstrate the mechanism of some algorithm (for example, Actor Critic (AC), DQN, Upper Confidence Bounds (UCB) (Auer, 2000)). The trained model is then passed a new task with the expectation that it will solve the task by applying this learned algorithm, without any further training. The model does not receive a prompt as such, and the task that is inferred is a whole algorithm. A key aspect of training this model is that extra long contexts are required, including trajectories up to four episodes long, to provide visibility of and encapsulate the whole learning process that the agent needs, to learn the algorithm. Effectively the model learns to mimic an RL algorithm so in theory it should be able to solve an RL task by applying the algorithm. This approach is viewed as an algorithm distillation method, typically trained over expert data.

6.3.2 Other aspects of Decision Transformers

ICL is a desirable feature in Transformer-based models but it requires the ability to extrapolate from available knowledge and perform sub-optimal trajectory stitching. While the authors suggest that the Decision Transformer (Chen et al., 2021) is able to extrapolate and stitch trajectories using the RTG as a prompt, Brandfonbrener et al. (2022) performed a detailed study showing that stitching is not possible in the original Decision Transformer model. Wu et al. (2023b) proposed a variant called Elastic Decision Transformer to facilitate stitching, using histories with varying lengths. This remains, however, a complex problem for Decision Transformers to solve.

Emmons et al. (2022), in “RvS: What Is Essential for Offline RL via Supervised Learning”, perform a systematic study of RL via Supervised Learning (RvS) investigating whether a supervised MLP could model RL data as effectively as a TD-algorithm (Tsitsiklis and Van Roy, 1997) or a Decision Transformer (Chen et al., 2021), and if so, what expertise of data or conditioning was required. Conditioning is also known as prompting in Transformer terminology and the paper compares the scalar reward-based prompt with a goal-state prompt. Emmons et al. (2022) find that goal-state conditioning performs better in some environments, noteably multi-task, goal-conditioned environments such as Four-Room Gridworlds (Chevalier-Boisvert et al., 2023), Pusher (Sawyer robot) (Nair et al., 2018), AntMaze (Fu et al., 2020), etc., while RTG conditioning performs better in locomotion-based environments like Half Cheetah, Hopper and Walker (Fu et al., 2020) that do not have specific task goals such as retrieving a key or opening a door.

Generalisation is challenging in RL and Lee et al. (2022) assess whether it is possible to train a generalist agent from diverse experience across multiple games, even if not to expert capabilities. Both online and offline RL models, including the Decision Transformer (Chen et al., 2021), BC (Levine et al., 2020) and offline TD methods were compared and the Decision Transformer showed the best overall performance and scalability, supplemented with additional expert guidance from a discriminator-like predictor trained on expert data. A key finding was that the Decision Transformer performed well on expert and non-expert data, outperforming BC on expert data which is a highly desirable quality.

Kazemnejad et al. (2023) show position encoding is not necessary and even restrictive at times in the standard Transformer language model however this has not as yet been demonstrated to be the case for the Decision Transformer. It may be the case that the multimodal nature of trajectory data benefits from some form of position encoding, as seen in the Gato model (Reed et al., 2022) that features multiple modes and uses extra levels of positional encoding. These aspects of Decision Transformers are less studied as of the time of this dissertation.

An interesting finding by Lee-Thorp et al. (2022) highlights the importance of the tokens in Transformer models. Their FNET model successfully replaces the attention mechanism with Fourier mixing, demonstrating that a key purpose of the dot product in the attention mechanism is to facilitate the mixing of token information. This has implications for preparing data for Transformers, specifically for the creation of more information mixing opportunities.

The Perceiver paper (Jaegle et al., 2021) hypothesised that special tokenisers were unnecessary to process multimodal inputs. Instead, Perceiver uses an attention bottleneck to process inputs with explicit position and modality information as these are deemed essential to conveying the relevant structure of the data to the Transformer, using cross-attention. The bottleneck features have a reduced model dimension d_{model} that restricts the capacity of the Transformer architecturally; this is circumvented by iteratively processing over the inputs. The purpose is to keep the Transformer architecture as generic as possible but highlights the importance of retaining position and modality information from the input data.

To summarise, to successfully use Decision Transformers to model RL processes requires environments that facilitate task generation sufficient for generalisation, careful choice of the conditioning mechanism based on the nature of the task and an architecture that induces stitching from optimal and non-optimal data. When multimodal data is involved, the model benefits from knowledge of supplemental structure and other characteristics of the data.

6.4 Proposed Approach: Multi-State-Action Tokenisation (M-SAT) in Decision Transformers

In this section Multi-State-Action Tokenisation (M-SAT) is presented, a proposed approach for dealing with multi-discrete actions using Decision Transformers in an offline RL context, to facilitate the formation and exploitation of discrete action-action and state-action relationships. This is achieved by increasing access to the action information in the trajectory using tokenisation techniques. Further methods are proposed to assess the impact of increasing the action contributions, including ablative studies and techniques from mechanistic interpretation (Elhage et al., 2021), a new field focused on the internal mechanisms and explanability of Transformers, to determine the effectivity of the proposed methods.

This section first describes the sequential modelling framework the chapter is based on, followed by the tokenisation approach and motivation in more detail. The environment, model and data collection are explained next, followed by methods of analysis and ablations.

6.4.1 Sequence Modelling using Decision Transformers

In this chapter an offline RL approach using sequential modelling is adopted, where RL trajectory data is modelled as a sequence using the Decision Transformer from Chen et al. (2021). The Decision Transformer receives a context of trajectory data and uses this prompt to generate the next action. The trajectory, τ is represented as: $\{(s_0, a_0, \hat{r_0}), ..(s_t, a_t, \hat{r_t}).., (s_T, a_T, \hat{r_T})\}$ where t is the timestep and $\hat{r_t}$ is the return-to-go or RTG.

Each element of the trajectory is pre-processed into tokens of the same dimension for the Transformer, with a post-processed batch of dimensions batch size$\times$context length×model dimension, where batch size, model dimension and context length are hyperparameters.

The RTG is calculated as per the original Decision Transformer (Chen et al., 2021) and there are no deviations from the training or action generation mechanism.

A minor modification is applied to the trajectory, τ to cater for multi-discrete actions as follows:

$$\{(s_0, \mathbf{a}_0, \hat{r_0}), ..(s_t, \mathbf{a}_t, \hat{r_t}).., (s_T, \mathbf{a}_T, \hat{r_T})\}$$

where $\mathbf{a}_t$ is a vector of discrete actions at timestep t.

6.4.2 M-SAT Approach and Motivation

A change to the way multi-discrete actions are pre-processed by the Decision Transformer is proposed. Multi-discrete actions are expanded to generate multiple action tokens per timestep, instead of the more typical single action token per timestep. Figure 6.2 demonstrates the tokenisation of multi-discrete actions to a single token, compared with the multiple tokens proposed in this chapter, illustrated in Figure 6.3.

In M-SAT a novel action tokeniser is presented that

- individually tokenises the discrete actions in each multi-discrete action
- concatenates each discrete action with the preceding state before encoding

The tokenisation process effectively imprints each individual action within the multi-discrete action with the preceding state features. This is analogous to the mechanism of positional encoding, where each token is tagged with its corresponding timestep. The motivation of the token expansion is to increase the visibility of actions in the attention heads to make more efficient use of the attention blocks

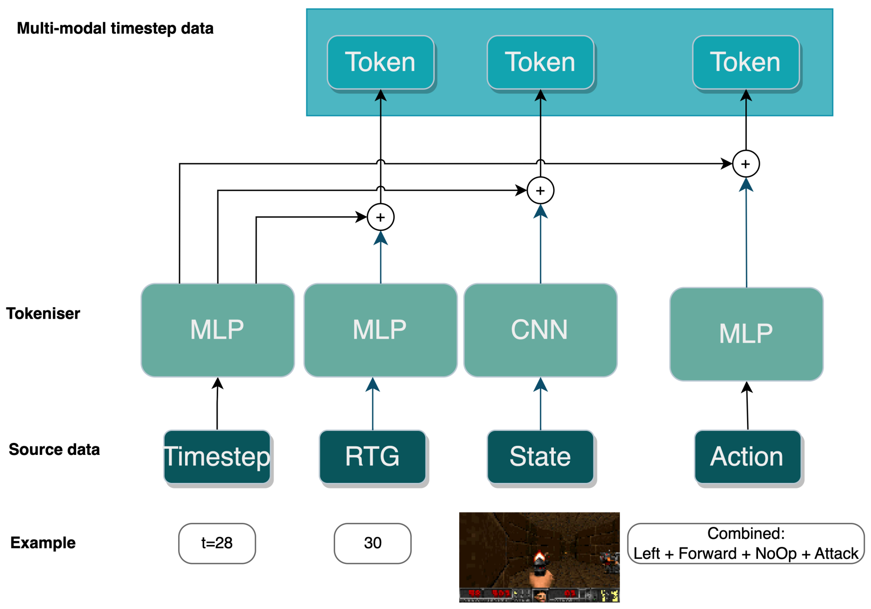


Figure 6.2: Tokenisation of the RTG, state and multi-discrete action in the standard Decision Transformer model to the model dimension. The timestep is encoded and added to each token as the position encoding.

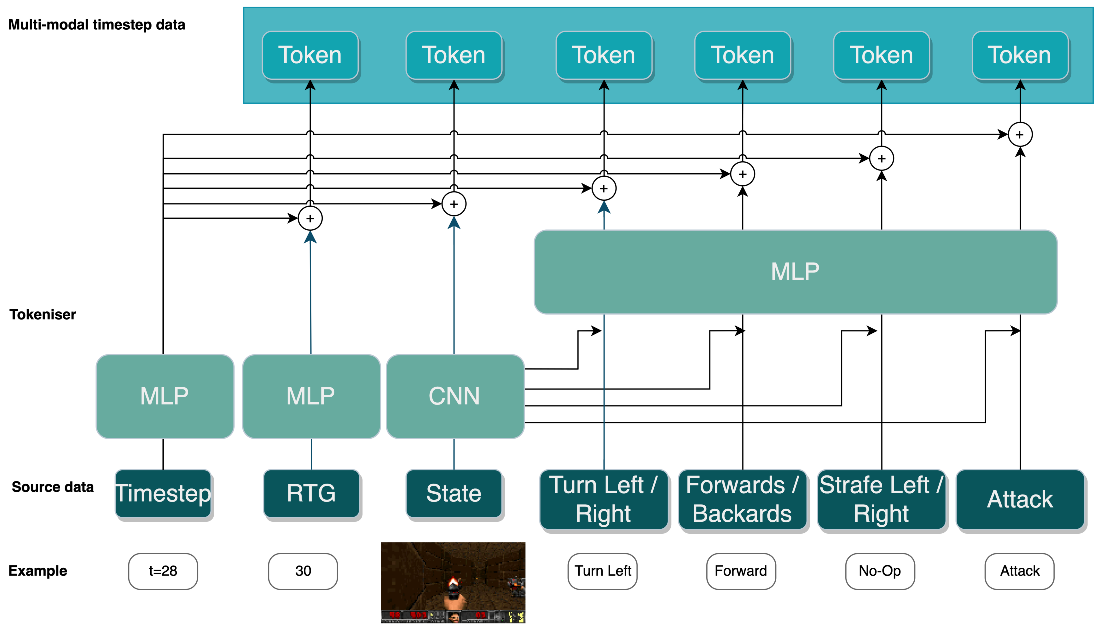


Figure 6.3: Tokenisation of the RTG, state and multi-discrete actions in the M-SAT Decision Transformer, with the proposed multi-token actions .

in the Decision Transformer. In M-SAT the attention heads of the Transformer will see multiple action tokens per timestep as part of the context, meaning the heads can formulate relationships between the individual actions, and between individual actions and the state and RTG tokens. The state-action associations are used to convey structural hints or suggestions to the attention blocks, such as particular relationships of interest. The intuition is the attention heads should have maximum visibility of all information (structural and relational) that could foster the development of relationships between actions, a desirable outcome in many scenarios (Moodley et al., 2019; Chitnis et al., 2020; Zambaldi et al., 2018; Jain et al., 2022).

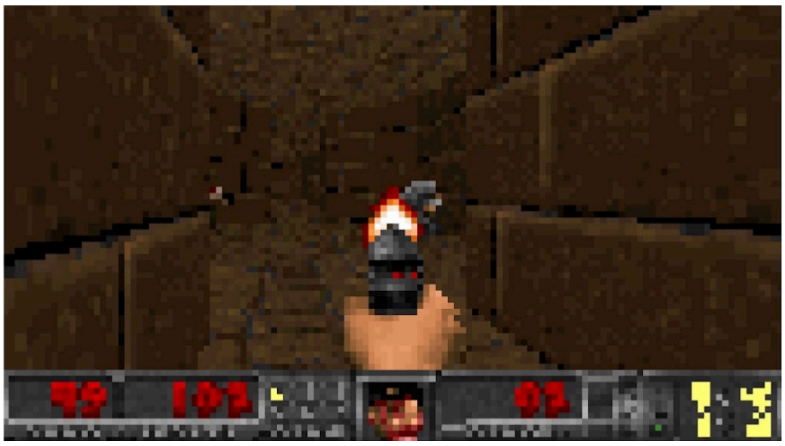


Figure 6.4: Sample state from the ViZDoom Deadly Corridor environment

For example, in ViZDoom's Deadly Corridor scenario (see Figure 6.5), given a state with a monster to the right of the agent, as in Figure 6.4 and a multi-discrete action [Right, Forward, No-Op, Attack], each discrete action will be associated with this state. A desired outcome is that the attention heads form an association with the appearance of the monster and the attack action, and with the position of the monster and the navigation or manoeuvring actions. State-action information allows the attention heads to form more granular relationships in the state-action space and this association is made possible with the additional tokenisation of the multi-discrete action. Transformer attention blocks learn relationships between sequential tokens in a context. Similarly, in the Decision Transformer, relationships between timesteps are learned and also between state, action and return-to-go tokens. The causal nature of the Decision Transformer means current timesteps refer to historical tokens or context to learn which tokens to focus on when generating a new action. As previously generated actions form part of the context when generating new actions, this process is auto-regressive. In the M-SAT approach the Transformer has a more detailed view of multi-discrete actions, with past actions more directly informing future actions. It is proposed that the increased action visibility allows the attention heads to learn relationships between individual discrete actions and to benefit from any action level structure available.

In RL, a common approach adopted in multi-discrete action spaces (Kanervisto et al., 2020) is to treat the multi-discrete action as a single action and not as separate discrete actions that can interact and have relationships. There are environments where relationships between actions exist (Fan et al., 2022; Kempka et al., 2016; Vinyals et al., 2017; Jain et al., 2022; Chitnis et al., 2020) and could be exploited to produce more intuitive and performant agents. While Decision Transformers have been used on environments with multi-discrete actions, the few examples observed to date have not focused on the action space in particular (Mathieu et al., 2023; Reed et al., 2022). Applying Decision Transformers to the action relationship problem is of interest for several reasons: firstly, the attention mechanism could focus on individual actions in the proposed M-SAT approach, facilitating the development of interactions and relations between individual actions; secondly, in this setup, beneficial state-action associations could also be detected by the attention mechanism and exploited at the individual action level.

6.4.3 Tokenisation and action generation

The role of the tokeniser in the Decision Transformer is to encode the trajectory elements, i.e. states, actions and RTGs, into a vector of fixed dimension known as the model dimension d_{model}. Decision Transformer tokens are multimodal including a set of state, action and RTG tokens, compared with tokens in natural language sequences that are all word tokens. The tokeniser used for each of the states, actions and RTGs depends on the nature of each mode, for instance a CNN would tokenise images while an MLP would be used to tokenise the discrete and continuous values typically associated with actions and RTGs. The output of each tokeniser has the model dimension, d_{model}. Individual tokenisation (as illustrated in Figure 6.3) of each discrete action in the multi-discrete action is proposed, that splits the multi-discrete action into the same number of action tokens. In the proposed approach, first a multi-discrete action with N discrete action elements is split into N one-hot encoded vectors representing each discrete action element; second, each one-hot encoded action is concatenated with the preceding state's feature vector, then tokenised using an MLP to generate N action tokens of dimension d_{model} for N discrete actions. The pseudo-code for this process is provided below.

Pseudo-code for M-SAT action tokenisation

Inputs: States with dimension $(B, T, n_{channels}, height, width)$,

Actions with dimension $(B, T, n_{actions})$

Where $n_{actions}$ is the number of elements in the multi-discrete action,

B is the batch size and T is the context length

d_{model} is the model dimension

d_{action} is the dimension of the multi-discrete action

$(n_{channels}, height, width)$ are the dimensions of the state images

(Refer to Table 6.2)

Method: Action Tokenisation

```
1. Tokenise state through CNN to get state tokens $(B, T, d_{model})$
2. One-hot encode the actions individually such that the new action
   dimensions are $(B, T, n_{actions}, d_{action})$ and each individual action has
   effective dimension $(B, T, d_{action})$
3. Concatenate the state token $(B, T, d_{model})$ to each one-hot encoded
   action $(B, T, d_{action})$; the resultant dimension is
   $(B, T, d_{model} + d_{action})$
4. Tokenise the concatenated state-action batch through the
   State-Action embedding MLP to get action tokens of dimension
   $(B, T, n_{actions}, d_{model})$
```

Despite the additional action tokens, the action generation process is unmodified. The M-SAT process only affects the processing of the input data into tokens and has not changed how actions are generated or how the loss is calculated. Multi-discrete actions are generated by sampling from categorical distributions supported by the state logits, as per Figure 6.1.

6.4.4 Implementation Details

A brief overview of the environment, including the action and state spaces is provided in section 6.4.4.1, followed by the setup and implementation details related to the Decision Transformer in section 6.4.4.2. The generation of offline data is non-trivial as sufficient volumes of suitable quality data are required; the processes designed for offline data generation are outlined in section 6.4.4.3.

6.4.4.1 Environment

The ViZDoom environment (Kempka et al., 2016) was designed for use in machine learning modelling initiatives including RL. The environment is configurable, with multiple scenarios with varying layouts, action spaces and reward functions possible. The ViZDoom Deadly Corridor (DC) scenario has a multi-discrete action space and is used in this chapter. The task involves the agent walking down a corridor, populated by attacking adversarial entities, to reach the goal, a green vest, at the end.

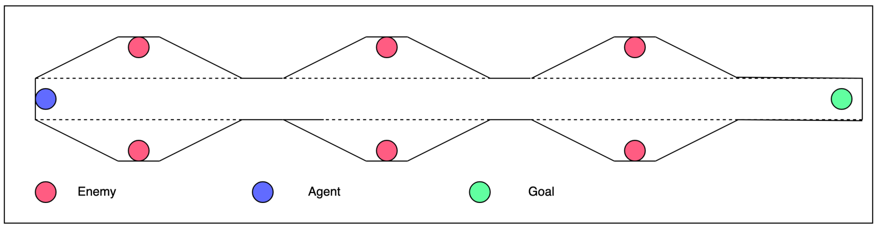


Figure 6.5: Simple map outline of ViZDoom's Deadly Corridor scenario showing the locations of the agent, enemies and the goal.

Rewards in ViZDoom A per step reward is calculated, based on the current coordinate of the agent, with the agent starting at coordinate 0, and the end of the tunnel is coordinate 1300. If the agent reaches the goal, it receives an additional +1000 reward (on top of the "x-coordinate" reward). There is no specific incentive

Table 6.1: ViZDoom Deadly-Corridor multi-discrete action space

a_0:	No-Op	Left	Right
a_1:	No-Op	Forward	Backward
a_2:	No-Op	Strafe Left	Strafe Right
a_3:	No-Op	Attack	-

for attacking monsters, however there is a death penalty of -100 if the agent dies. Finally, there is a time limit of 2100 timesteps to reach the end of the corridor before the episode terminates.

The Health Gathering Supreme (HGS) has a discrete action space so while it is not the main focus of this chapter it is occasionally used to provide context and act as a comparative scenario. In HGS the agent collects healthpacks to survive for 2100 timesteps. Rewards are immediate, +1 for living on every step, -100 death penalty; there is no direct reward for the collection of healthpacks.

Multi-discrete actions in ViZDoom - Deadly Corridor Multi-discrete actions are vectors of discrete actions, requiring the agent to select multiple discrete actions per timestep:
$\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) \quad \text{where} \quad \mathbf{a}_i \in \{0, 1, ..., N_i\} \quad \text{for} \quad i = 1, 2, ..., n$
(refer to section 2.2.2 for a reminder of discrete vs multi-discrete actions).

In Deadly Corridor the multi-discrete action space has 4 possible discrete actions to select on each timestep, described in Table 6.1.

The multi-discrete action has the form: $\mathbf{a} = [\mathbf{a_0}_t, \mathbf{a_1}_t, \mathbf{a_2}_t, \mathbf{a_3}_t]$ where t is the timestep and $\mathbf{a_0}$ - $\mathbf{a_3}$ are selected from the appropriate row in Table 6.1. For example, an action with Left, Forward, No-Op and No-Op is represented as $[1, 1, 0, 0]$.

States in ViZDoom The ViZDoom environment provides a state with two components, viz. an image (dimensions $3 \times 72 \times 128$) and an environment feature vector with the agent's health. In this work, only the image was used as the state when training the Decision Transformer. Some commonly used image preprocessing techniques derived by Mnih et al. (2013) for pixel-based image states in Atari were applied to the ViZDoom images, including resizing and frame skipping. A frame skip of 4 was adopted as per Kempka et al. (2016) who demonstrated that higher skip-rates learnt faster in general but suffered with poor fine-grained control. Also in keeping with Kempka et al. (2016), colour images were used rather than the gray-scaling often performed on Atari images.

6.4.4.2 Decision Transformer Related Setup and Implementation

A few aspects pertaining to the Decision Transformer are discussed below, including the context length and further details related to tokenisation. This is followed by a training flow for the Decision Transformer customised with the M-SAT implementation.

Context Length The context length is the number of timesteps the Decision Transformer sees during training, resulting in an input batch dimension of [batch size$\times$context length$\times$token dimension]. The context length is a hyperparameter that is optimised to suit the task, for instance in HGS the optimal context was between 40-50 while for DC a minimum context of 60 was adopted. In HGS rewards are obtained directly on every step compared with DC where the rewards can be sparse. Chen et al. (2021) comment that the context length can be important for enabling credit assignment and mention that the Key-to-Door environment was more complex, requiring a context of at least the length of a full episode. Similarly, Laskin et al. (2023) found that in-context learning in RL was more likely when the context length covered a full episode, adding that the context often needed to span at least 2-4 times the episode length. These findings indicated that it is crucial to determine an effective context length for DC due to the sparsity of rewards and the maze-like nature of the task, i.e. the Transformer needed to see the attainment of the goal at the end of the trajectory within the context. Based on these findings a context length of 60 was selected after analysing the offline trajectory data and ensuring sufficient episode trajectories were available where the agent reached the end of the corridor and obtained a reward.

Tokenisation In the Decision Transformer the tokens are multimodal, with sequential trajectories of timesteps comprised of return-to-go (RTG), state and action tokens. A single timestep contains a sequence of the RTG token, state token and action token encoded with the timestep, as illustrated in Figure 6.3.

A separate tokeniser is used for each mode to match the mode data. Specifically, image-based ViZDoom states are processed by a 3-layer CNN producing a token vector of dimension 128. Similarly, the continuous-valued RTG is processed by an MLP generating a vector of 128. In the original Decision Transformer specification discrete actions were processed by an MLP to produce a single token vector of dimension 128. Multi-discrete actions can similarly be processed through an MLP to produce a single action token vector of dimension 128. This configuration of tokeniser for multi-discrete actions, that converts N-discrete actions to a single embedded token, will be referred to as the Baseline and referenced in the experiments in section 6.5. In the proposed state-action M-SAT approach, multi-discrete actions with N dimensions are processed with the preceding state by an MLP to produce N tokens of dimension 128, one token per discrete action.

Finally, a few points are clarified:

- the model dimension, d_{model} is 128, so each token has a dimension of 128 but the tokens have different modalities. It is not the case that a single token comprises a concatenation of RTG, state and action with a dimension of 3×128.
- the context length is doubled to accommodate the extra tokens introduced by the M-SAT approach such that a trajectory with 10 timesteps that would

normally have a context length of 30 (3 modes $\times$ 10 timesteps) is now expanded to a context length of 60 (6 modes $\times$ 10 timesteps), allowing for the three additional action modes.

- for completeness it should be noted that during inference when the Transformer is generating the next action, the final timestep is composed of the RTG followed by the state token only.

Training the Decision Transformer The model used by the Decision Transformer is the GPT model by Radford et al. (2018). The code is largely vanilla except for modifications by Chen et al. (2021) to the tokenisation to cater for the multiple modalities of RTG, states and actions. This dissertation performed further modifications, specifically to the action tokenisation, to support the methods outlined in the Approach (section 6.4). A brief overview of the data generation, training flow and inference processes for the Decision Transformer is provided below with customisations highlighted.

1. Generate trajectory data (custom): generates at least 1 million timesteps, stored as individual files for states, actions, rewards and done flags. Section 6.4.4.3 provides more details of this process.

2. Training Flow: The code is divided into a few modules, viz. a runner process that controls the entire training flow, the data generation module that prepares the training data, the GPT model that defines the architecture and finally the trainer that manages training and evaluation of the model. A few details of the main processes is provided below:

   - Data generation: the runner reads the trajectory data and generates the training dataset (custom); this method parses the above-mentioned trajectory data, collating episodic data into state, action, done flag, RTG and timestep arrays to support and interface with the dataloader code provided by Chen et al. (2021).

- Trainer: the runner creates the trainer and the model objects and calls the trainer. The trainer initialises the dataloader and feeds batches of data to the model to obtain the logits (training and inference mode) and loss (training mode). The model is updated by the optimiser and the process repeats for the requested number of epochs.
- GPT Model: The model is largely the original code from Chen et al. (2021) with minor modifications to support different action tokeniser networks (custom) that are plugged into the model. The M-SAT approach affects the way tokens are composed in the context; instead of adding a single action token, multiple action tokens are inserted into the context. This modification is self-contained within the GPT model and is largely transparent to the rest of the process.
- Evaluation: The model is evaluated after each epoch. Unlike training, that has no contact with the environment and relies entirely on offline data, evaluation (custom) interacts with the environment to assess the agent's performance. This process is managed by the trainer.

3. Inference or generation:

   - The initial prompt is composed of a desired RTG; the initial state from the environment is also provided at this time. The RTG is manually derived from an analysis of the training data reward distributions. An optimal to semi-optimal RTG is used for evaluations.
   - The model receives the prompt and returns logits that are converted into a categorical distribution from which the action is sampled. In the multi-discrete action scenario, this process is repeated for each discrete action.

- The action is passed to the environment and the next state, reward and done flag are received from the environment and appended to the context to form the next prompt. The RTG for the next prompt is the previous RTG less the environment reward. The model receives the new prompt and the cycle continues.

Details of the key hyperparameters for the Decision Transformer are provided below: (The best model performance for both the single-action-token and multi-action-token models was achieved using the following hyperparameters)

Decision Transformer Hyperparameters for training:

env = Deadly Corridor

layers = 8

heads = 8

number of timesteps of training data = 500k

epochs = 5

embed dropout = 0.1

residual dropout = 0.1

attention dropout = 0.3

batch size = 128 or 64

learning rate = 0.005

warmup = 2000

context length = 60

eval target = 29

6.4.4.3 Data Generation

The volume and nature of offline data required to train the Decision Transformer are important factors as considerable data is required. In the experiments that follow expert offline data was generated using a PPO algorithm and used to train the Decision Transformer.

The Asynchronous-PPO (APPO) (Petrenko et al., 2020) was used to generate offline data for training the Decision Transformer. The APPO algorithm uses a combination of parallelisation of training components, including multiple instances of the environment and multiple workers for rollouts and inference, as illustrated in Figure 6.6. Queues are used to reduce the waiting time for workers, resulting in high speed training and data generation. Furthermore the APPO implementation in the Sample Factory (SF) library provides wrappers for several ViZDoom scenarios including DC, with hyperparameters for training the scenarios to convergence (expert performance). For the DC scenario the APPO agent was trained for 200 million frames until reward evaluations stabilised (optimal or close to optimal), after which approximately 1 million frames of data was collected. The training curves for APPO are provided in Figures 6.7, 6.8 and 6.9.

Trajectory data from SF comprised states (colour images), actions, rewards and done flags and required some manipulation before it was useable as an offline dataset. When generating data for the Decision Transformer it is important to maintain the sequential structure of the data. The dataset generation process is very closely modelled on the original code provided by Chen et al. (2021), described below. First whole episodes were extracted from the trajectory files, producing separate, sequentially matched arrays for states, actions, done indices, timesteps and RTGs. Sequentially matched means trajectories for an episode could be compiled by reading steps across the data files and down the files to generate an episode. The done indices signal the end of each episode. The timesteps files indicates the position of the episode within the larger array of 1 million datapoints. The RTG is calculated for each step of the episode at this time, using information in these files. The timesteps and RTG are generated specifically for the Decision Transformer.

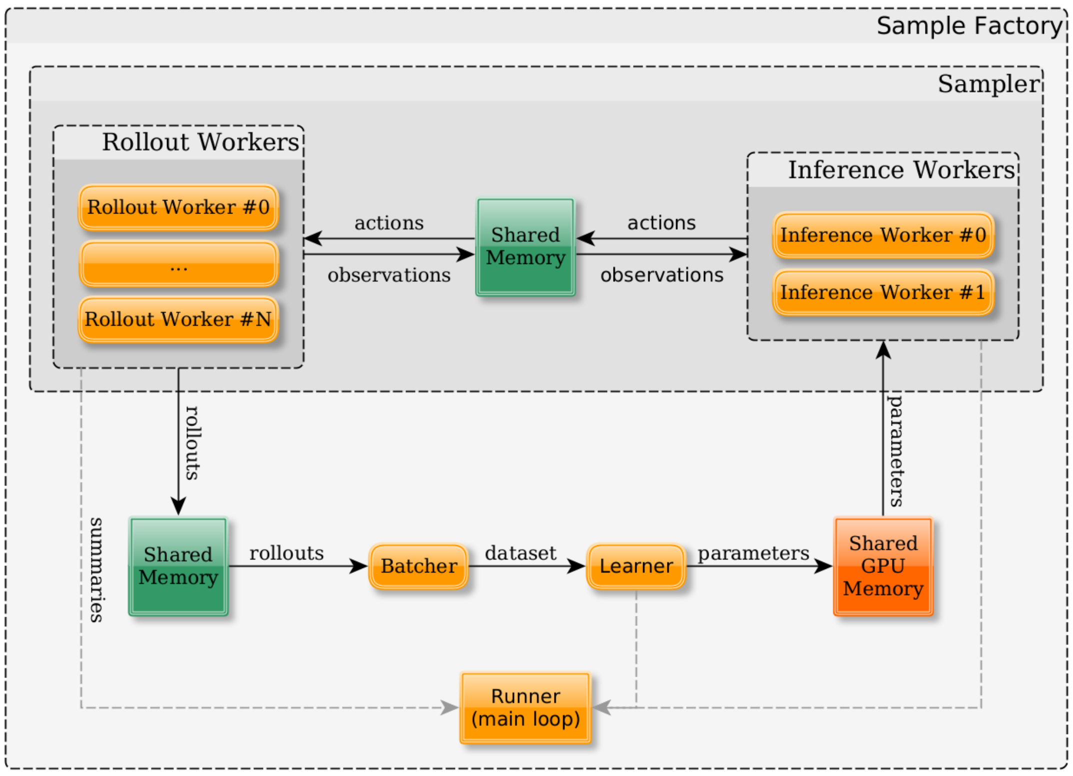


Figure 6.6: Sample Factory architecture overview - Source: (Petrenko et al., 2020)

The custom dataloader reads the source arrays and generates batches based on the batch size and context length. In keeping with the Atari approach from Chen et al. (2021) there is no padding of contexts. The data is a continuous stream of trajectories, and segments of context length are extracted from this stream (at a randomly generated index) to compose batches of dimension batch size×context length.

The box below provides the key hyperparameters required for generating the offline data in SF. The data was generated on a server with 40 CPU cores, 200GB RAM and an Nvidia GPU (RTX A5000 with 24GB RAM). These specifications were used to design the hyperparameters and achieve optimal performance from the APPO algorithm, as recommended by SF.

Relevant command line arguments to replicate data collection in Sample Factory:

```
env=doom-deadly-corridor
num-workers=80
num-envs-per-worker=8
batch-size=4096
train-for-env-steps=200000000
with-wandb=True
rollout=128
use-rnn=True
```

6.4.5 Methods of Analysis

A few comparative methods are outlined followed by ablative and analysis approaches adopted.

The key idea of the M-SAT approach is to increase the amount of token information the Decision Transformer has access to in multi-discrete environments by expanding out multi-discrete actions to multiple tokens. If action relationships and structure are the aim, the position of the tokens could provide further relevant information to the attention mechanism. In this vein, the first comparative methods consider the impact of position encoding by first removing position encoding to allow greater mixing of action tokens and, second, by adding more positional information at the action level. The next method determines if the state-action information is actually useful and removes states when tokenising the actions.

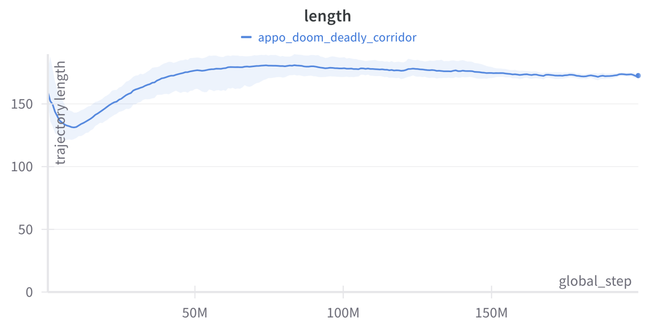


Figure 6.7: APPO Training curves for Deadly Corridor training - length (Sample Factory)

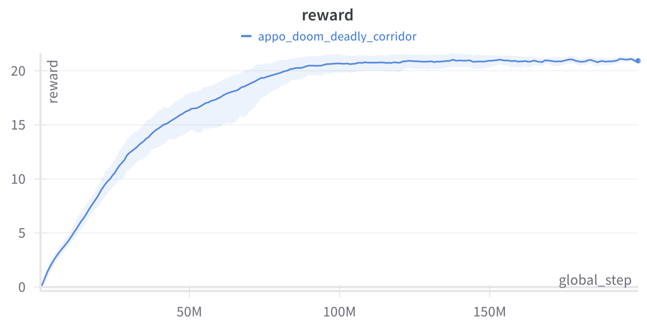


Figure 6.8: APPO Training curves for Deadly Corridor training - reward (Sample Factory)

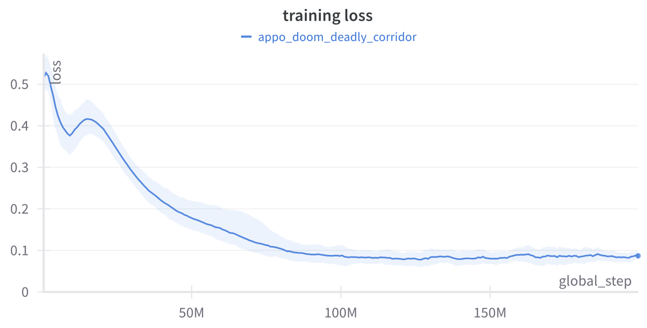


Figure 6.9: APPO Training curves for Deadly Corridor training - loss (Sample Factory)

6.4.5.1 Position Encoding

Positional Encoding (PE) is how the Transformer is informed about the sequencing or position of a token. In the Decision Transformer, position encoding is applied at two levels: first, an encoding of the timestep within the training context is applied such that each set of state, RTG and action tokens are encoded with the same timestep position information; second, a global encoding is applied that captures the global timestep position of the datapoint in the global dataset.

No PE: To further facilitate the development of relationships between tokens, the removal of all position encoding is proposed. The position encoding effectively ties the state, actions and rewards together and might therefore impact the visibility of the newly proposed action tokens. Kazemnejad et al. (2023) performed a detailed study of the effects of positional encoding on increasing context lengths in LLMs and found that when position encoding was entirely removed, the network learned a form of relative encoding to replace it. Following a similar vein to Kazemnejad et al. (2023), position encodings were removed from the Decision Transformer in this method to allow for more mixing opportunities between all tokens, facilitating intra-timestep interaction.

Action PE vs state-action: In M-SAT, states are pre-pended to discrete actions before tokenisation in a method reminiscent of position encoding, where tokens in a single trajectory step are all tagged with the same timestep encoding. The question arises, would simply adding a different form of action position encoding be sufficient; is the state information specifically necessary?

Two ablative methods were derived from this question:

- No-State-Action (MAT): this method is a variant of the M-SAT method without the state-action tokenisation; it retains the multiple action tokens but takes the one-hot encoded discrete action only as input to the MLP. The multi-discrete nature of the action token space is maintained. The purpose is to determine if the state-action association is providing useful information to the attention heads. This method will be referred to as Multiple Action Tokens (MAT) for the duration of this chapter. Figure 6.10 illustrates the tokenisation of the MAT model in more detail.

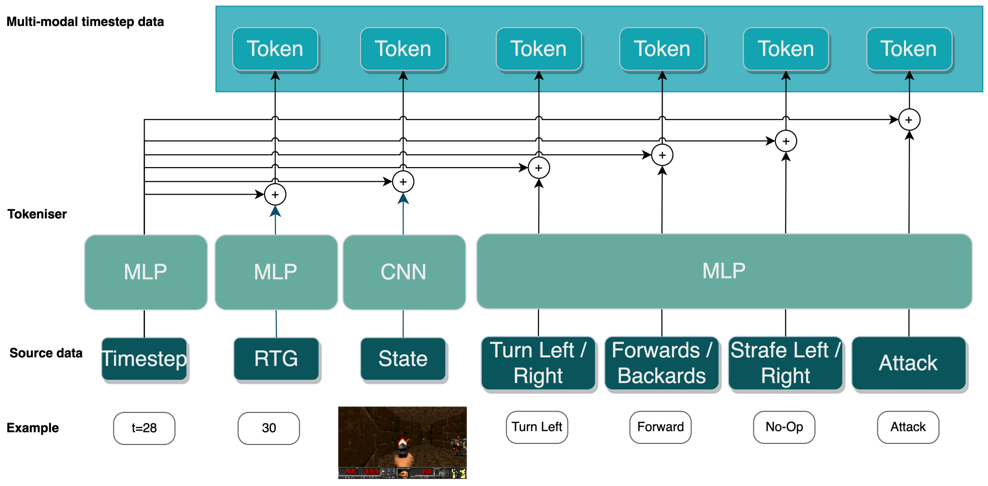


Figure 6.10: Tokenisation of the RTG, state and multi-discrete actions to the Decision Transformer model dimension, with the proposed multi-token action for MAT. Notice the actions are not encoded with the state, as in M-SAT (Figure 6.3)

- Action-PE: this method was inspired by the Perceiver (Jaegle et al., 2021) paper that used an attention bottleneck to extract features for position and modality because it was essential to convey the relevant structure of the data to the Transformer. Similarly, this method acknowledges the importance of position information in the modes by introducing additional position encoding specific to the action tokens. This method is applied to the MAT method above, with multiple action tokens but no state-action associations. Instead a Fourier transform based calculation is used.

6.5 Experiments and Results

Experiments were designed to address several key questions:

Table 6.2: ViZDoom Deadly-Corridor 11-dim multi-discrete action where N is NoOp, L Left, R Right, SL Strafe Left, SR Strafe Right, F Forward, B Backward, A Attack

0	1	2	3	4	5	6	7	8	9	10
N	L	R	N	F	B	N	SL	SR	N	A

- Does a Decision Transformer trained with multiple action tokens outperform single action tokens in a multi-discrete action scenario? (sections 6.5.2 and 6.5.4 versus section 6.5.1)
- In the multiple action token configuration, does reinforcing state-action relationships aid the attention mechanism? (section 6.5.4)
- Does the multi-token model benefit more from more generic position encoding than from state information? (section 6.5.5)
- Does removing all position encoding improve the multi-token model by increasing information mixing? (section 6.5.3)

6.5.1 Baseline Method

The baseline approach is a Decision Transformer modified to handle multi-discrete actions by tokenising them to a single action token of embedding dimension 128. The action is first one-hot encoded into an 11-dimension vector according to Table 6.2, then passed through an MLP.

The Deadly-Corridor scenario is trained with the standard Decision Transformer model provided by Chen et al. (2021) with minimal modifications to replace the action tokeniser as described in section 6.4.4.

Table 6.3: Table of Results: Baseline (single action token) and M-SAT are compared over 100 evaluation runs and multiple seeds. A version of M-SAT without state-action associations (MAT) tests how effective multiple-action tokens are without any reference to the state during action tokenisation. Two Position Encoding (PE) variants are included: first, PE is removed (No PE) and second, additional action PE is added (Action PE).

Model Type	Ave Eval Rewards	No PE
Baseline	10.22 ± 3.2	3.60 ± 1.5
M-SAT	14.88 ± 0.44	16.17 ± 0.33
MAT	12.28 ± 3.1	4.68 ± 2.79
Action PE	8.0 ± 2.77	-

6.5.2 Decision Transformer trained with Multi-Action Tokens (M-SAT)

In this method, 4-dimensional multi-discrete actions are tokenised into 4 separate tokens of 128 dimensions, one per discrete action. The actions use a shared embedding network (a 2 layer MLP) as a tokeniser, converting a 1×4 multi-discrete action vector into a $1 \times 4 \times 128$ set of action tokens. This has the effect of doubling the size of the training batch, adding an extra three tokens per timestep, converting a context with 30 timesteps to a context of $180 = 30 \times 6$ (RTG, state, action1, action2, action3, action4). The standard Decision Transformer training process was modified to perform the additional tokenisation inside the model to cater for the change to context length.

Furthermore, in M-SAT, the tokeniser MLP takes as input the preceding state concatenated with the discrete action when generating the action tokens (Figure 6.3). The state is first embedded with its own tokeniser, generating a feature vector of dimension $d_{model} = 128$ that is concatenated with each discrete action associated with the timestep.

6.5.3 No Position Encoding

In this experiment M-SAT (section 6.5.2) was modified to remove all position encoding to determine if this resulted in improved information mixing opportunities in the attention mechanism, given the increased number of action tokens. For completeness, the experiment is also performed on the baseline or single action token (section 6.5.1) and the MAT model (section 6.5.4).

6.5.4 No State-Action (MAT)

This method is a variant of the M-SAT method that retains the multiple action tokens but removes the state-action associations (Figure 6.10). The tokeniser MLP takes only the discrete action as input when generating the action tokens, generating a feature vector of dimension $d_{model} = 128$. In this configuration, both PE and no PE are considered.

6.5.5 Action PE

This is a second contrastive experiment related to position encoding, adding extra positional information to the action tokens. Position vectors were generated for each discrete action using Fast Fourier Transforms (FFT), as per FNET (Lee-Thorp et al., 2022) and Perceiver (Jaegle et al., 2021). This experiment was applied to the MAT variant, to determine the importance of the state-action association to M-SAT during tokenisation.

6.5.6 Discussion and Analysis

Results from Table 6.3 show that M-SAT out-performs the baseline model, averaged over 100 evaluation runs and multiple seeds. The MAT model also performs better than the baseline model, indicating that purely expanding the multi-discrete action into multiple tokens improves performance. The motivation for increasing the number of action tokens in M-SAT was to defer the problem of extracting structure and developing relationships in the multi-discrete action space to the attention mechanism, that specialises in this sort of task. The hypothesis was that the additional tokens would increase the amount of information available for mixing, resulting in better communication within the attention layers. The results suggest that the increased information available via the extra action tokens does improve the performance of the Decision Transformer. This is likely because the attention mechanism determines action importance instead of the tokenising MLP. In M-SAT, the individual discrete actions are first encoded by an MLP but are received by the attention heads as individual tokens; the attention heads learn relationships at the token level and can form state-action, action-action or RTG-action relations with the action tokens. In the baseline model, the single action tokenising MLP must determine how to encode the relationship between the actions when generating the action token; the attention heads have much less information to formulate relationships from, by comparison. The results suggest deferring complex processing, such as relationships, to the attention heads rather than the tokenisers. M-SAT features cross-modal interactions, making better use of the attention mechanism with good results.

The state-action associations are only possible because of the multi-action tokenisation but results show that the state-action model M-SAT is an improvement over the MAT model, suggesting that the attention heads have detected and used additional structure found in the state-action tokens. The variance of the M-SAT model is also less than the other models, indicating a higher level of stability across seeds.

Table 6.3 also displays the results for the position encoding (PE) variants, including the additional Action PE and the No PE models. The Action PE experiment was meant as a comparison to the state-action association used in M-SAT. Both Action PE and M-SAT may be viewed as supplying some form of action structure to a base multiple action token model (MAT). In the former, the individual discrete actions are supplemented with FFT positional encodings while in the latter, state-related encodings are provided. The state-action scenario (M-SAT) outperforms the Action PE model quite considerably, suggesting that the nature of the supplemental information provided to the Transformer is important.

Finally, the removal of all position encoding from both the baseline and M-SAT models are compared resulting in vastly differing outcomes. The baseline model performs much worse when position encoding is removed, while M-SAT performs better, actually improving on the performance of the M-SAT model and displaying an even smaller variance across seeds. A possible hypothesis for why M-SAT performs better without any position encoding is that the state information supplied during the encoding or tokenisation of actions serves as a sufficient replacement for position encoding. The position encoding would normally maintain the sequence for the Transformer; in M-SAT perhaps the state information, whose features encode aspects of the scene, contains sequential information too. This result suggests the findings of Kazemnejad et al. (2023) in LLMs might be extended to Decision Transformers and warrants further study.

6.5.6.1 Raw attention and Attention Flow

This section looks more closely at the models for indicators that demonstrate that the additional tokens are actively contributing to the development of the M-SAT model. Specifically the attention scores and value contributions of individual tokens are considered for each attention head and layer. There are eight layers with eight attention heads each and it is not immediately obvious how to analyse the attention mechanism and combine data across the heads and across layers. A few different methods are listed below, but the main idea in this section is to present a summarised view of the attention contribution or participation of each token in the evaluation run.

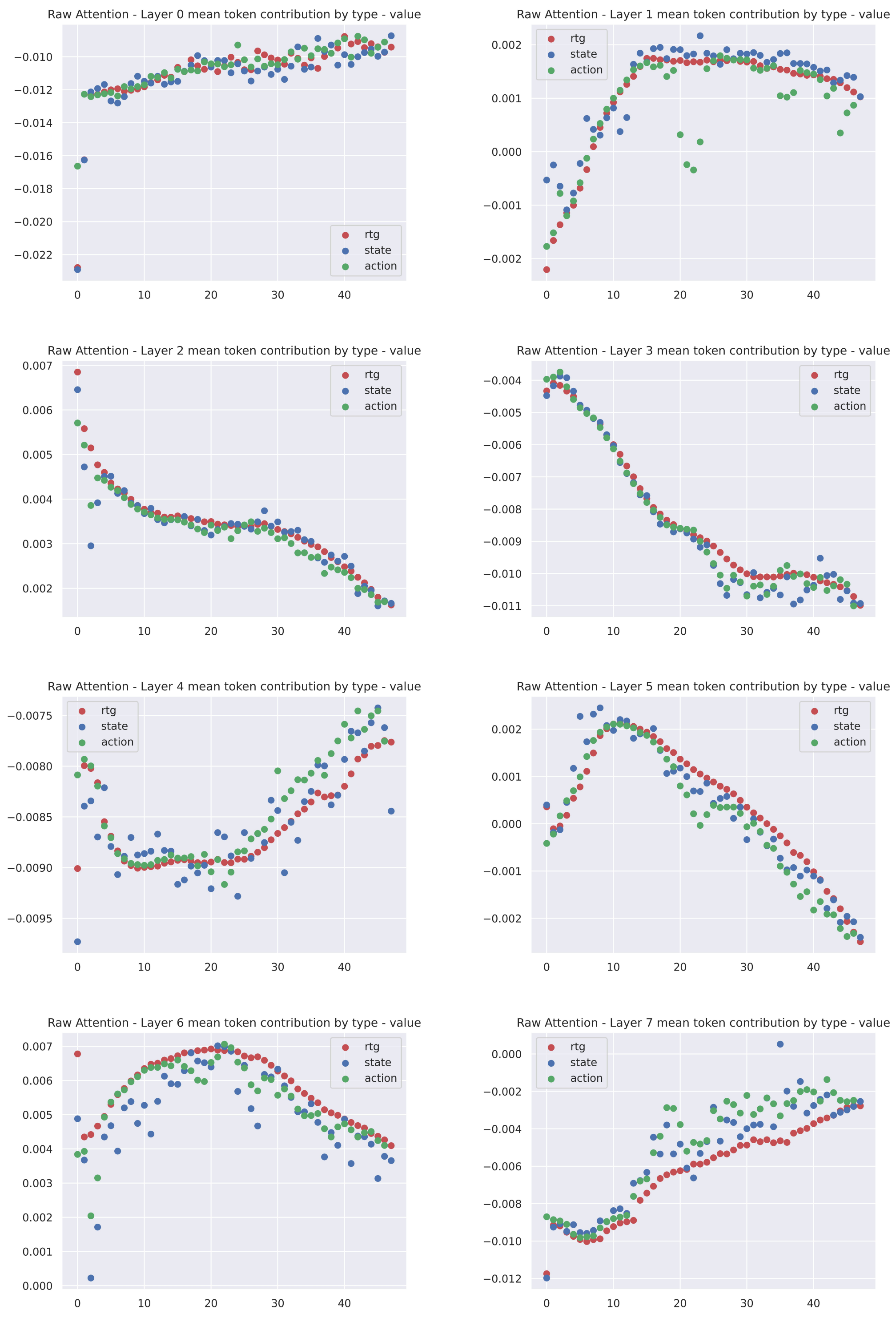


Figure 6.11: **Baseline model:** Each plot is the value contributed by each token, summarised by layer. The value is averaged across all attention heads, then averaged across the last dimension. The x-axis is the timestep for an evaluation run and the y-axis is the averaged value. The value is separated by token type with a different colour for states, RTGs and actions.

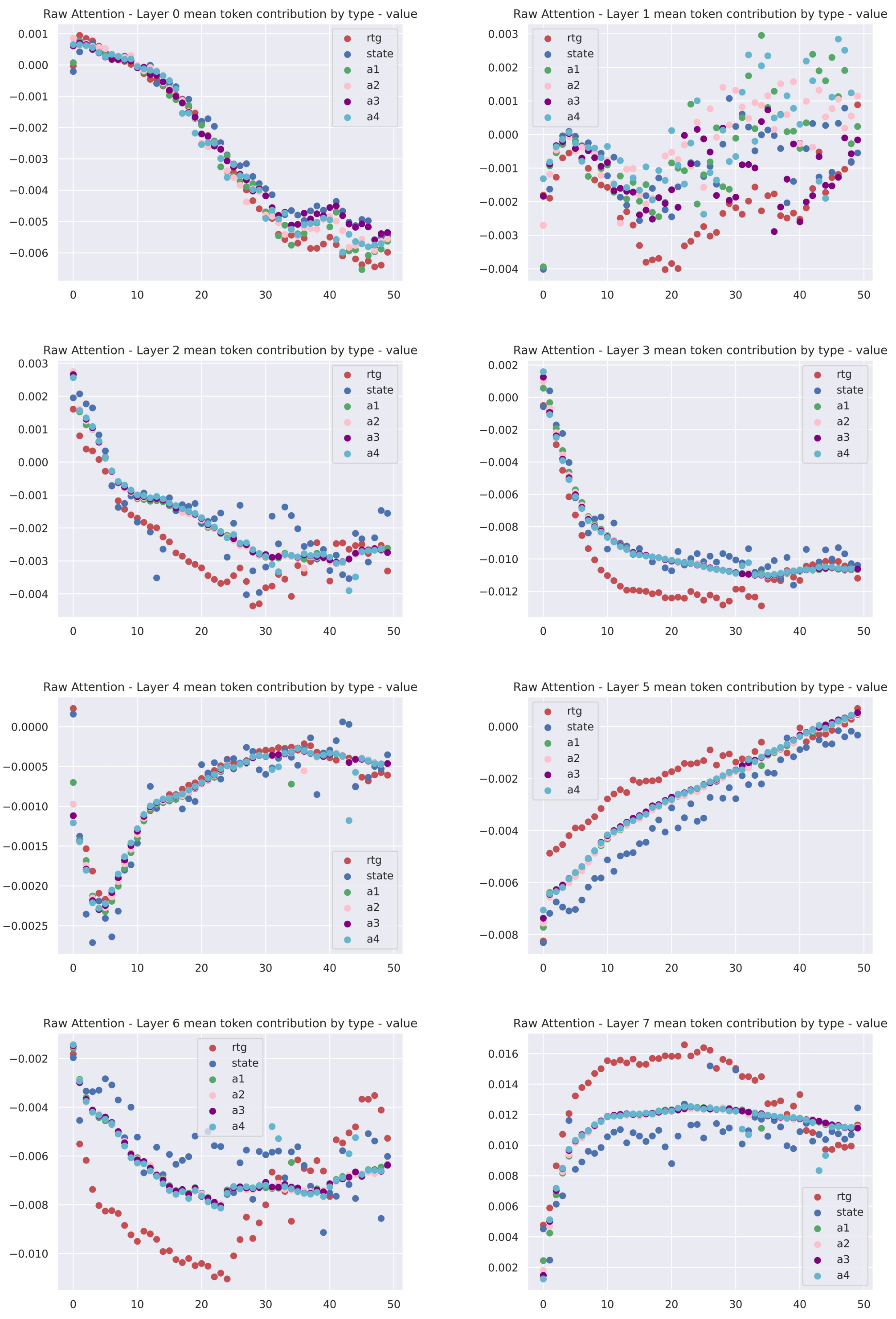


Figure 6.12: **Multiple Action Tokens (MAT) model - no State Action:** Each plot is the value contributed by each token, summarised by layer. The value is averaged across all attention heads, then averaged across the last dimension. The x-axis is the timestep for an evaluation run and the y-axis is the averaged value. The value is separated by token type with a different colour for states, RTGs and actions.

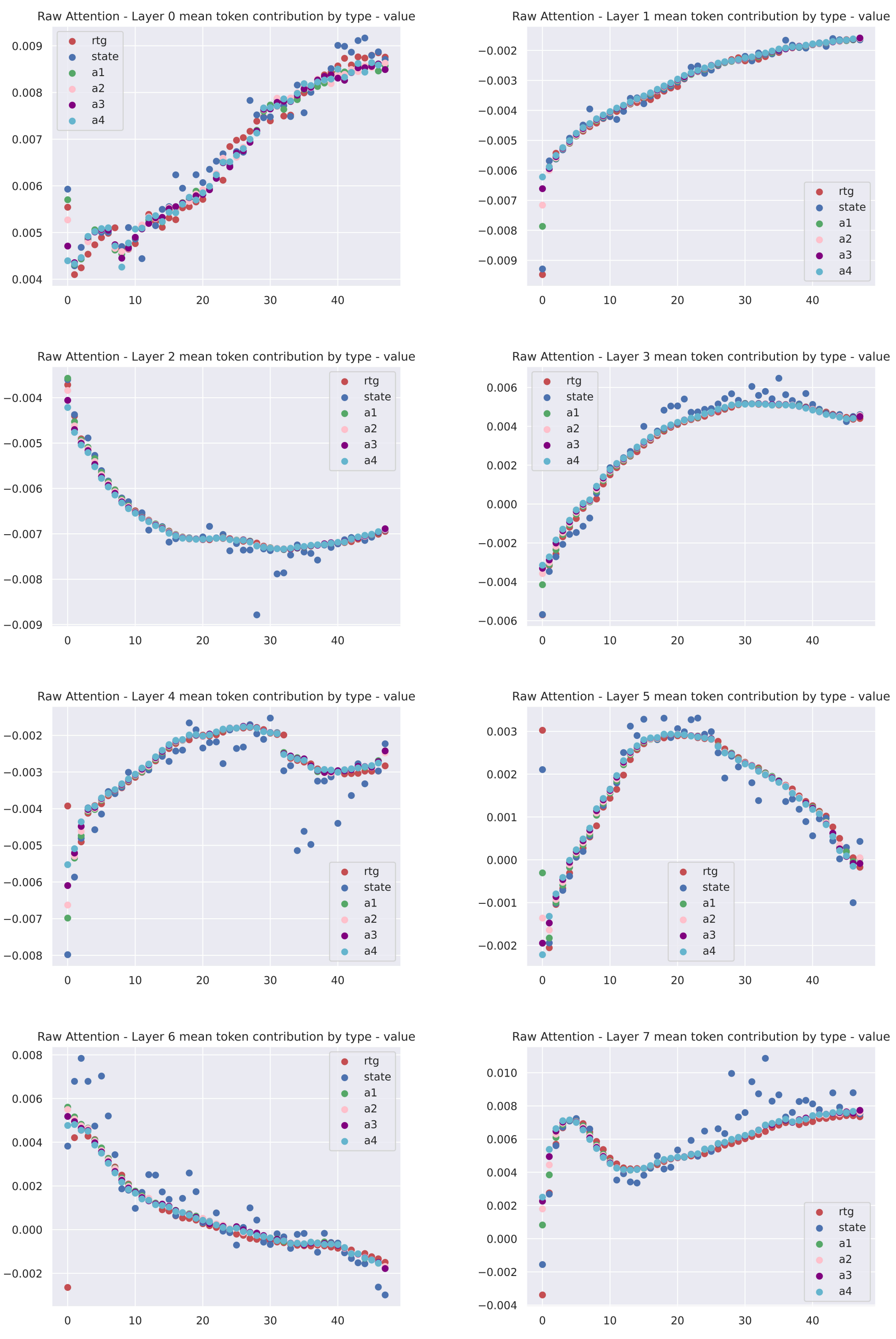


Figure 6.13: **M-SAT model:** Each plot is the value contributed by each token, summarised by layer. The value is averaged across all attention heads, then averaged across the last dimension. The x-axis is the timestep for an evaluation run and the y-axis is the averaged value. The value is separated by token type with a different colour for states, RTGs and actions.

The first set of plots in Figures 6.11, 6.12 and 6.13 summarises the attention values by layer. The heads are fused by taking the average across all heads; for attention values this results in a matrix of batch size × context length × model dimension. To produce the plots the final dimension was averaged in each case. The result provides an averaged view of the value associated with each token (state, RTG, actions) by layer. The plots are obtained by generating an evaluation run from a trained model in the Deadly Corridor ViZDoom environment, then extracting the relevant attention scores and values from each attention head using hooks, a concept in PyTorch for interrogating models. The x-axis represents each timestep in the evaluation run and each datapoint represents the attention value associated with that particular point, for instance states are illustrated in red, RTGs in blue and actions depend on the model type. What type of plot is expected? Key areas of the corridor where an enemy is encountered should have higher value and attention scores than other areas; otherwise timesteps closer to the end should provide more information than earlier timesteps, in general. The averaged view hides the details of the heads but provides some idea of the trends in each layer. For the M-SAT model (Figure 6.13), layers 0,1,3,4,5,7 show later tokens have more value than earlier tokens. Layers 6 and 2 are more complex; these layers may be removing information rather than actively hindering the attention mechanism. This pattern is also visible for the baseline and MAT models. M-SAT plots have less variance than the other two models, with MAT showing the most variance and a clear separation of the RTG from the state and action tokens.

A summary over all layers is provided in Figure 6.14 for each model, reflecting the averaged value contributions and also the averaged attention scores per token. The baseline and M-SAT models show similar trends for the value contributions, with later tokens contributing more value than earlier. M-SAT displays a very concise plot implying all tokens have a similar contribution save for a set of state tokens at particular timesteps. The peak states correspond with periods of enemy attack implying these states contribute more strongly to the final outcome. The attention scores for M-SAT are more varied across the types of token, showing contributions from all four actions that becomes more pronounced in the latter half of the trajectory, with each action type fluctuating independently at each timestep. The independence of the action tokens is much more pronounced in the MAT plots (middle row), with the baseline model also showing strong action activity.

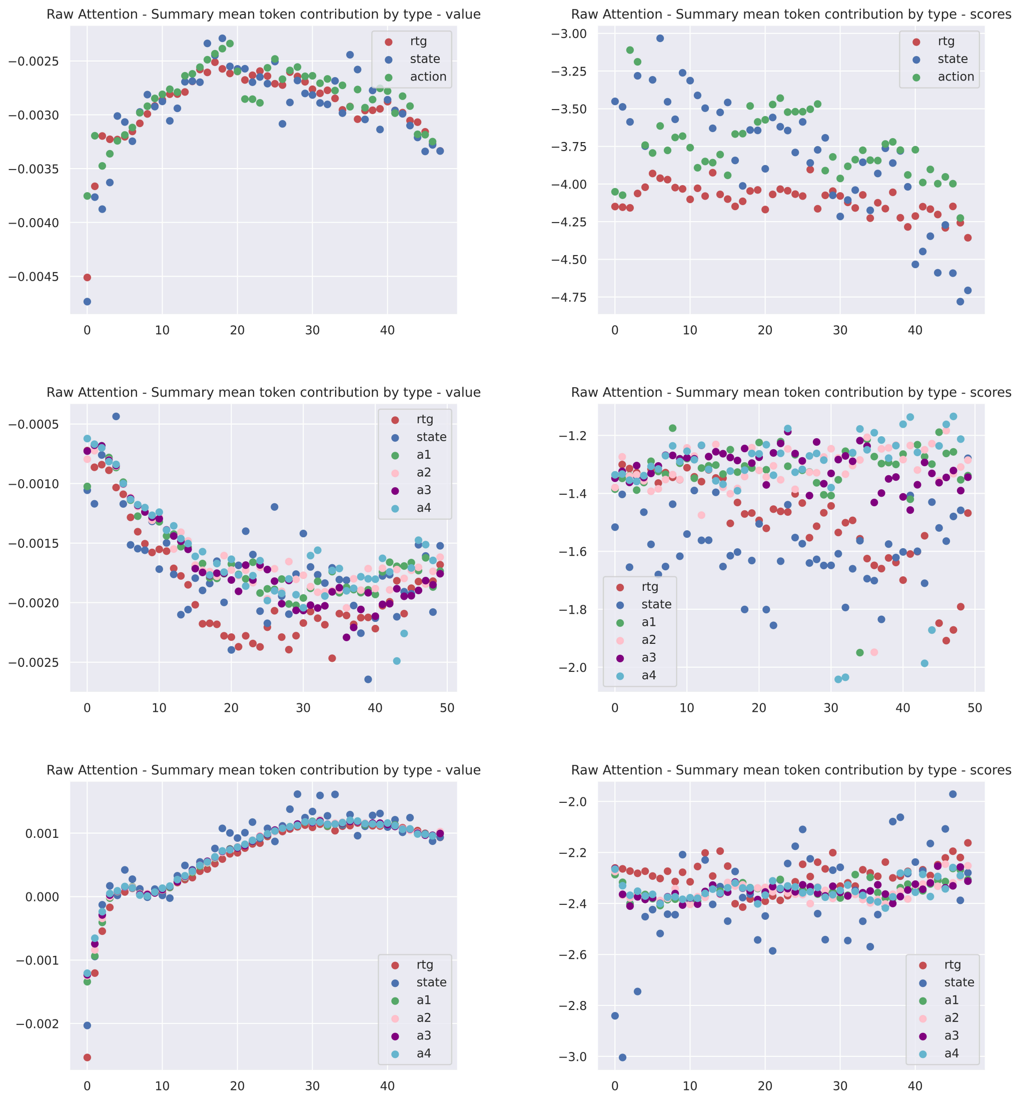


Figure 6.14: The first row of plots is the baseline, the middle row is MAT and the last row is M-SAT. Each plot is summary over all layers of the value on the Left and the attention scores on the Right. The x-axis is the timestep for an evaluation run and the y-axis is the averaged value (Left) or attention score (Right).

Averaging the raw attention over heads and layers is a crude measure. The query-key dot product measures the attention score between tokens; the score is used to weight the value of the current token and the weighted value becomes the input to the next layer, where the process repeats. This implies that when summarising the influence of a token, the attention layers should be multiplied across layers rather than averaged and the residual connections also have an impact on this process. Attention flow and attention rollout (Abnar and Zuidema, 2020), adjusts the raw attention weights for tokens to take into account the effect of each attention layer on the token and the residual connections. Abnar and Zuidema (2020) were able to demonstrate that both attention flow and attention rollout measures correlate with tokens of importance, derived by performing token ablations, while raw attention weights were unable to establish a similar link. Attention rollout was applied as an alternate measure to the raw attention measures plotted above. Attention rollout was chosen because it was easier to implement than attention flow, but provided similar information.

Figure 6.15 displays the attention rollout for each model. The attention heads are fused using averaging and a discard ratio of 0.9 was used as per Gildenblat (2020).

The difference that stands out the most between the three attention rollout plots is the state tokens. In the baseline, there are fewer high impact state tokens compared with M-SAT and a much larger percentage of high impact action and RTG tokens. Given the nature of the environment, i.e. fairly sparse rewards, it is conceivable that the state tokens would contribute more strongly to the agent's behaviour, with the other modalities providing support. Given, instead, the high percentage of attention on actions by the baseline mode, it is possible that the baseline model has memorised the environment and not solved it, reciting actions from memory. The MAT model is more in line with expectations, with more high impact states displayed, however there is very little action contribution visible, im-

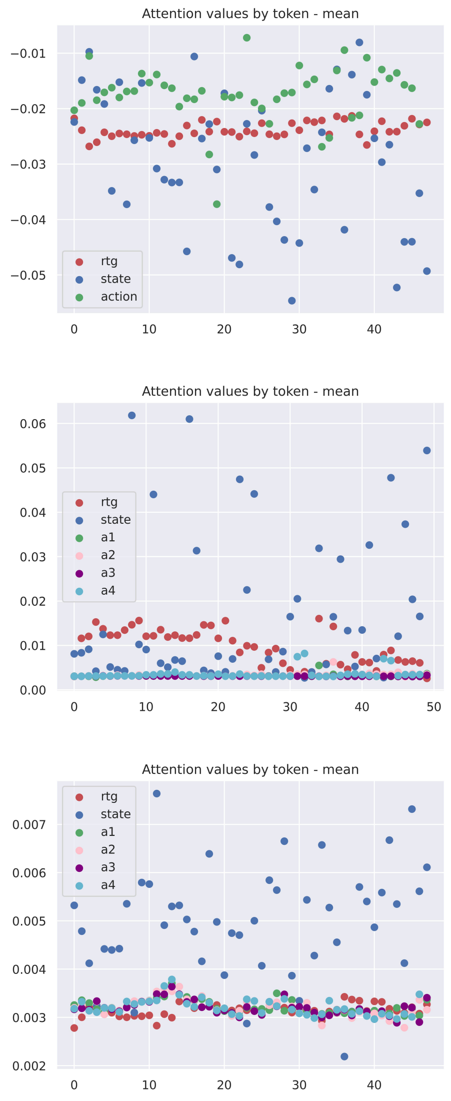


Figure 6.15: Each plot displays the attention rollout (Abnar and Zuidema, 2020) for each model: baseline (Top), MAT (Middle) and M-SAT (Bottom). The rollout is calculated by fusing attention heads using the mean. The x-axis is the timestep for an evaluation run and the y-axis is the attention rollout value.

plying that merely increasing the number of action tokens is insufficient. Finally the M-SAT model once again displays high impacting states but also more contribution from the individual action tokens. Close inspection of the action activity shows definite activity in a number of timesteps suggesting the action tokens are making individual contributions.

6.5.6.2 Interpreting the Transformer

Transformers are complicated to analyse from an explanability point of view. Nevertheless the growing field of Mechanistic Interpretation (Elhage et al., 2021; Bricken et al., 2023; Sharma et al., 2023) has been making progress towards the understanding of LLMs. The same is not yet true for Decision Transformers but some techniques that may be applied towards understanding the behaviour of Transformers more generally are applied below. First, embeddings learnt for the observations or states are analysed using EigenCAM (Bany Muhammad and Yeasin, 2021), a non-gradient based method that highlights the activated portions of the image focused on by the Transformer during inference. Next the attention heads are interrogated and states from empirical analyses are used to identify particular heads that seem to align with a task.

EigenCAM EigenCAM (Bany Muhammad and Yeasin, 2021) is a visual explainability approach used to analyse representations learned during training. Often these methods are applied to visual representations derived from convolutional neural networks with image inputs. EigenCAM works by calculating the principle components of the representation. These are then superimposed on the original image to highlight the areas most activated or focused on during inference. EigenCAM was applied to states from evaluation runs to determine if the representations learnt aligned with the task. Figure 6.16 displays several key states involving enemy attacks in Deadly Corridor alongside the same images with EigenCAM images superimposed over them. The activations align with expectations in these cases, highlighting the location of the enemy in both the M-SAT and baseline models, with a few exceptions at the end of the corridor when the agent is approaching the goal. This is illustrated in Figure 6.17 where there are hints once again that the baseline model has memorised actions. Towards the end of the corridor Eigen-CAM shows that the baseline model has lost its focus on the goal but in the final timesteps still manages to complete the episode by strafing (walking sideways) and thus obtaining the reward. Compare this with M-SAT in Figure 6.18 where the model maintains focus on the goal while moving towards it.

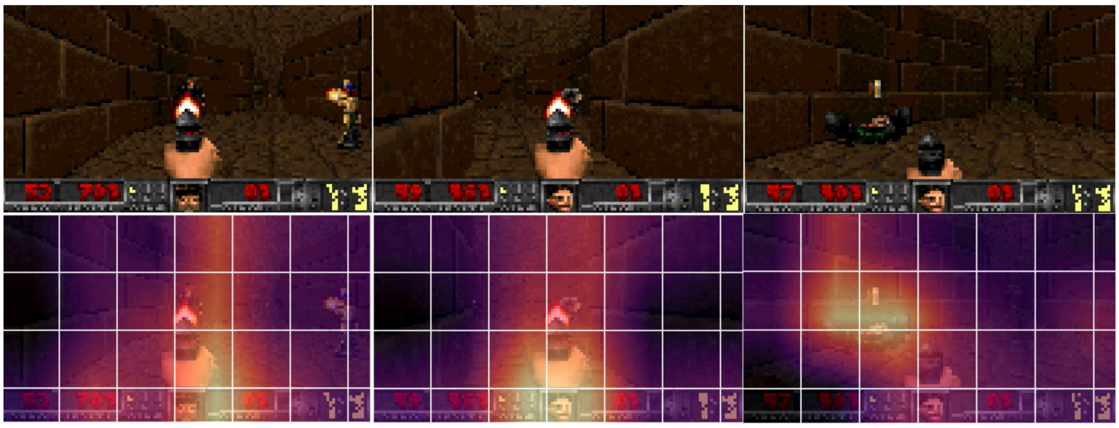


Figure 6.16: Original images (top row) and EigenCAM images (bottom row) samples displaying the model's focus when enemy is located in the image.

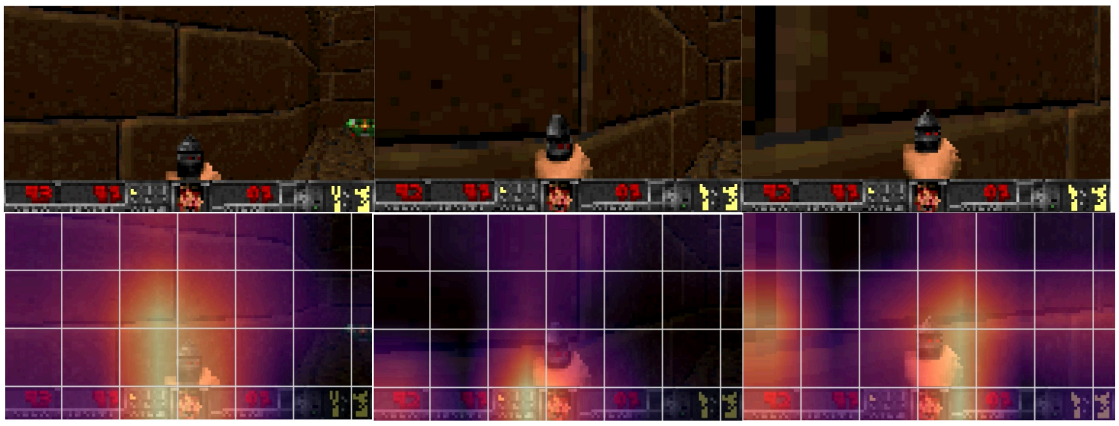


Figure 6.17: Original images (top row) and EigenCAM images (bottom row) for the baseline model at the end of the corridor, where the agent loses track of the goal but somehow achieves the goal

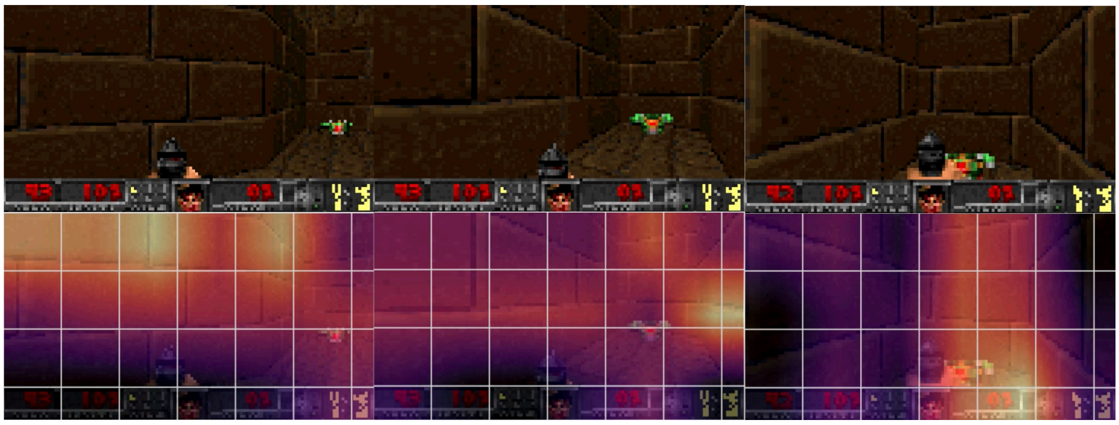
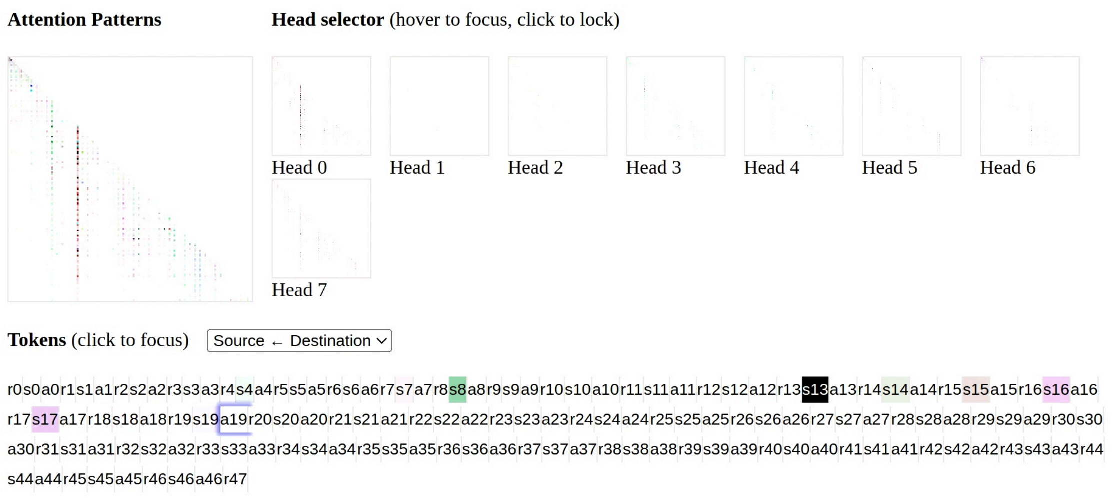


Figure 6.18: Original images (top row) and EigenCAM images (bottom row) for the M-SAT model at the end of the corridor, where the agent keeps track of the goal until it is reached

Attention heatmaps This section uses tools that visualise the attention head patterns in each layer to assess whether the Transformer model benefits from the multi-token action configuration of the MAT model. While it is not possible to directly compare the single vs multi-token Transformer models, it is possible to illustrate that the individual action tokens are used by the attention heads. CircuitsViz (Cooney, 2022) is a utility provided to visualise the attention between tokens in a sequence of text when passed through a trained Large Language Model (LLM) Transformer model in inference mode. CircuitsViz requires as input the sequence of text parsed and the attention pattern from each layer. The attention pattern may be retrieved using PyTorch (Paszke et al., 2019) hooks or other tools provided for this purpose such as Transformerlens (Nanda and Bloom, 2022) or Torchlens (Taylor and Kriegeskorte, 2023).

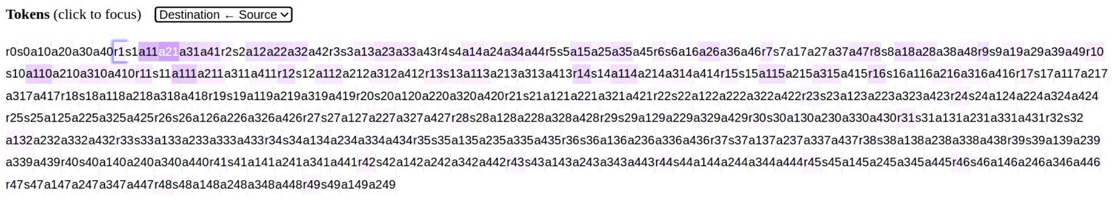
Evaluation runs for the Deadly Corridor ViZDoom scenario were stored and passed through a fully trained Decision Transformer model and the attention pattern was extracted using Torchlens (Taylor and Kriegeskorte, 2023). A sample CircuitsViz output is provided in Figures 6.19 and 6.20 for attention heads from a single-action token model. The attention heads are displayed in the upper section and the sequence of tokens in the lower section in Figure 6.19. Hovering over a token highlights the influence it has, as captured in the screenshot in Figure 6.20

or the influencers, as in Figure 6.19. Figure 6.20 also demonstrates how the tokens are displayed for a multi-token model where each action has format aij where i is the action index from 1-4 and j is the trajectory timestep. For convenience only the sections displaying the highlighted tokens are shown.



r0s0a0r1s1a1r2s2a2r3s3a3r4s4a4r5s5a5r6s6a6r7s7a7r8s8a8r9s9a9r10s10a10r11s11a11r12s12a12r13s13a13r14s14a14r15s15a15r16s16a16
r17s17a17r18s18a18r19s19a19r20s20a20r21s21a21r22s22a22r23s23a23r24s24a24r25s25a25r26s26a26r27s27a27r28s28a28r29s29a29r30s30
a30r31s31a31r32s32a32r33s33a33r34s34a34r35s35a35r36s36a36r37s37a37r38s38a38r39s39a39r40s40a40r41s41a41r42s42a42r43s43a43r44
s44a44r45s45a45r46s46a46r47

Figure 6.19: Sample CircuitsViz output showing each of 8 attention heads in the upper section and all tokens in the sequence in the lower section. All timesteps have 3 modes, for example timestep 0 has 3 tokens, viz. r0-RTG, s0-state and a0-action. In the Figure the token a19 is selected resulting in several states (s8,s13,s14,s15,s16,s17) highlighted according to the amount of attention contributed to the action by the head (where each head is denoted by a different colour). The intensity of the highlighted tokens convey the amount of influence other tokens in the sequence have on each other.

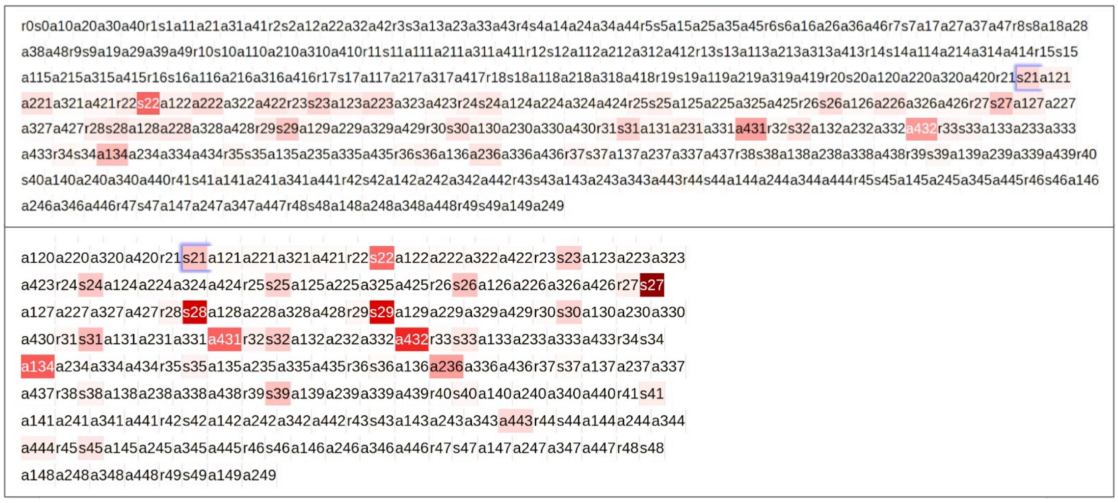


r0s0a10a20a30a40r1s1a11a21a31a41r2s2a12a22a32a42r3s3a13a23a33a43r4s4a14a24a34a44r5s5a15a25a35a45r6s6a16a26a36a46r7s7a17a27a37a47r8s8a18a28a38a48r9s9a19a29a39a49r10
s10a110a210a310a410r11s11a111a211a311a411r12s12a112a212a312a412r13s13a113a213a313a413r14s14a114a214a314a414r15s15a115a215a315a415r16s16a116a216a316a416r17s17a117a217
a317a417r18s18a118a218a318a418r19s19a119a219a319a419r20s20a120a220a320a420r21s21a121a221a321a421r22s22a122a222a322a422r23s23a123a223a323a423r24s24a124a224a324a424
r25s25a125a225a325a425r26s26a126a226a326a426r27s27a127a227a327a427r28s28a128a228a328a428r29s29a129a229a329a429r30s30a130a230a330a430r31s31a131a231a331a431r32s32
a132a232a332a432r33s33a133a233a333a433r34s34a134a234a334a434r35s35a135a235a335a435r36s36a136a236a336a436r37s37a137a237a337a437r38s38a138a238a338a438r39s39a139a239
a339a439r40s40a140a240a340a440r41s41a141a241a341a441r42s42a142a242a342a442r43s43a143a243a343a443r44s44a144a244a344a444r45s45a145a245a345a445r46s46a146a246a346a446
r47s47a147a247a347a447r48s48a148a248a348a448r49s49a149a249

Figure 6.20: Switching the focus from Destination to source and selecting RTG r1 shows the influence r1 has on other tokens. Note in this example the tokens are from a multi-token action model where, for example, the action for timestep 0 is represented by $[a10, a20, a30, a40]$ and timestep 20 is $[a120, a220, a320, a420]$

Some sample results are presented next that highlight that the attention heads acknowledge the individual actions in the multi-token action Decision Transformer model. Figure 6.21 exhibits the attention from two heads, layer 1 head 0 and layer

3 head 0. These segments show that these heads view state s21 as important by allowing it to influence both states and actions that follow. Figure 6.22 shows state s21 to be firing upon an enemy. The heatmaps imply that some information from this state is transmitted to future states and also to three action tokens, notably the a4 (attack action position) tokens for timesteps 31, 32. States 32 and 33 show the impact of those actions, with the enemy coming into range again and the agent firing. The empirical data suggests the firing state s21 influences the firing action a4 directly in at least two attention heads, implying the multi-token action representation allows for a granular level of influence that the single-token action representation will not be able to experience or reveal. This pattern is repeated in several heads but the figure displays a sample to demonstrate the concept.



```
r0s0a10a20a30a40r1s1a11a21a31a41r2s2a12a22a32a42r3s3a13a23a33a43r4s4a14a24a34a44r5s5a15a25a35a45r6s6a16a26a36a46r7s7a17a27a37a47r8s8a18a28
a38a48r9s9a19a29a39a49r10s10a110a210a310a410r11s11a111a211a311a411r12s12a112a212a312a412r13s13a113a213a313a413r14s14a114a214a314a414r15s15
a115a215a315a415r16s16a116a216a316a416r17s17a117a217a317a417r18s18a118a218a318a418r19s19a119a219a319a419r20s20a120a220a320a420r21s21a121
a221a321a421r22s22a122a222a322a422r23s23a123a223a323a423r24s24a124a224a324a424r25s25a125a225a325a425r26s26a126a226a326a426r27s27a127a227
a327a427r28s28a128a228a328a428r29s29a129a229a329a429r30s30a130a230a330a430r31s31a131a231a331a431r32s32a132a232a332a432r33s33a133a233a333
a433r34s34a134a234a334a434r35s35a135a235a335a435r36s36a136a236a336a436r37s37a137a237a337a437r38s38a138a238a338a438r39s39a139a239a339a439r40
s40a140a240a340a440r41s41a141a241a341a441r42s42a142a242a342a442r43s43a143a243a343a443r44s44a144a244a344a444r45s45a145a245a345a445r46s46a146
a246a346a446r47s47a147a247a347a447r48s48a148a248a348a448r49s49a149a249
```

```
a120a220a320a420r21s21a121a221a321a421r22s22a122a222a322a422r23s23a123a223a323
a423r24s24a124a224a324a424r25s25a125a225a325a425r26s26a126a226a326a426r27s27
a127a227a327a427r28s28a128a228a328a428r29s29a129a229a329a429r30s30a130a230a330
a430r31s31a131a231a331a431r32s32a132a232a332a432r33s33a133a233a333a433r34s34
a134a234a334a434r35s35a135a235a335a435r36s36a136a236a336a436r37s37a137a237a337
a437r38s38a138a238a338a438r39s39a139a239a339a439r40s40a140a240a340a440r41s41
a141a241a341a441r42s42a142a242a342a442r43s43a143a243a343a443r44s44a144a244a344
a444r45s45a145a245a345a445r46s46a146a246a346a446r47s47a147a247a347a447r48s48
a148a248a348a448r49s49a149a249
```

Figure 6.21: The top snippet is from Attention layer 1, head 0 and the bottom snippet from layer 2, head 0 - both show the influence of state s21 on future tokens, including individual action tokens. Refer to Figure 6.22 to view a sample of the associated states from this evaluation trajectory.

Other interesting patterns observed in the attention head heatmaps are described below. Figure 6.23 illustrates another common pattern with a strong focus on certain key states. States s5 (adversary detection), s21 (attack) (also in Figure 6.21, s25 (post-attack) are key states to successfully attack and advance towards the goal state. The focus on key states is a repeated pattern amongst the heads, implying there is some redundancy in what specialisations the heads develop.

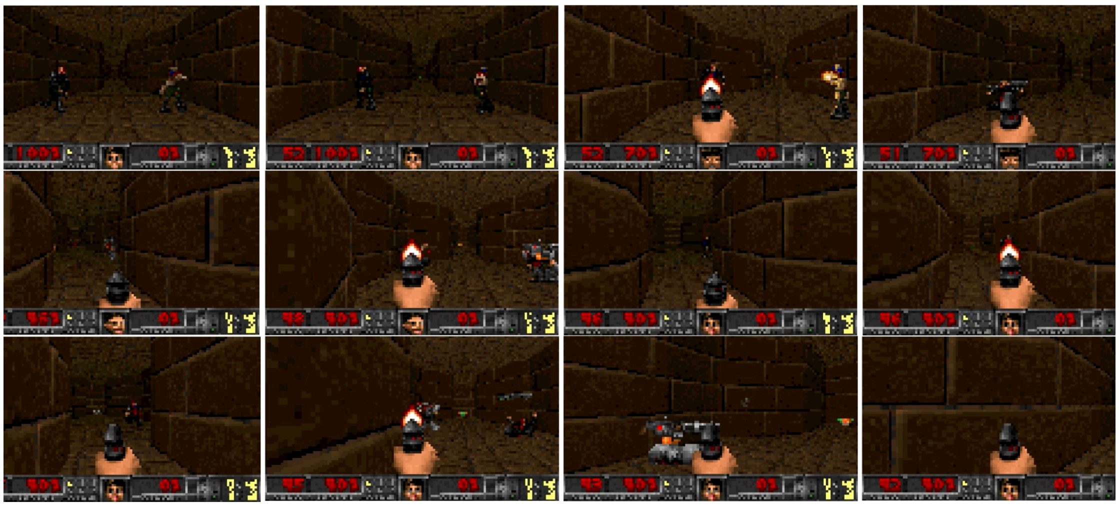


Figure 6.22: states from ViZDoom dataset, supporting attention heatmap snippets (Figure 6.21) from left to right, top to bottom: S1, S2, S5, S7, S16, S21, S31, S32, S33, S35, S40, S44

Some heads focus on the RTG and actions instead of the states, examples of which are available in Figure 6.24. The heads either focus on RTG at the start of the trajectory (L2H7, L5H7) or at the end (L1H0, L3H0, L6H4). The strong focus on the RTG at the end of the sequence may make sense in this sparsely rewarded environment as the goal state is approached from states s40 to s44. In L3H0 the fallen enemy and the goal are in view (s40) and the attention head fixates on this state until the end of the corridor with RTGs referring back to s40. L5H7 displays early RTG influence but this wanes mid-trajectory. Nevertheless this is a pattern observed for multiple heads implying that RTG activity is high at the start and end of the trajectory. Finally, L6H4 shows a mixture of actions and RTGs that focus on the state s24, which has an enemy in the scene. The next action (in s25) is an attack and several of the following highlighted tokens are also attack-relevant scenes, implying that attack-specific information is being conveyed to these tokens.

The attention map visualisations illustrate that the additional action tokens are actively attended to by the attention heads, especially the attack action that receives information from related states.

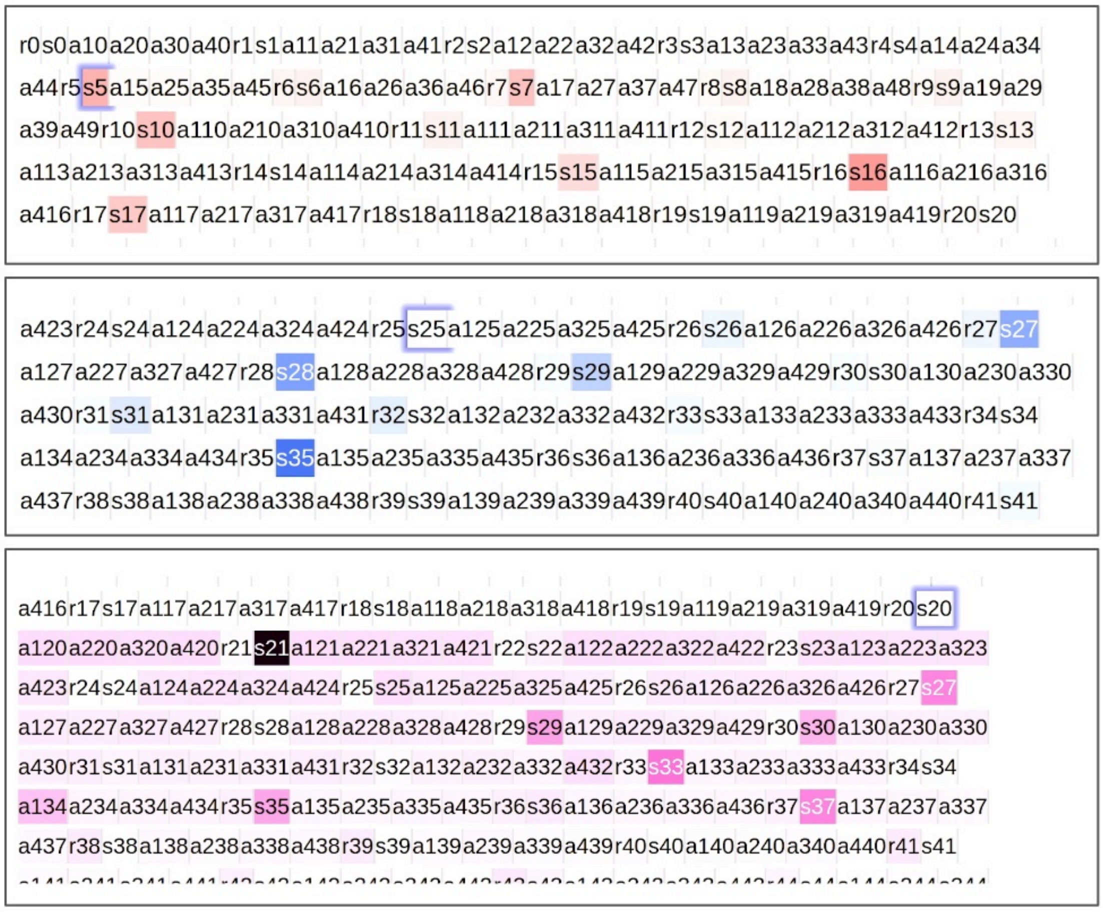


Figure 6.23: Top to bottom: L3H0, L7H5, L7H7

Training and loss analysis Training the Decision Transformer for the Deadly-Corridor scenario proved more complicated than scenarios with a single discrete action per timestep and plentiful reward information, like Health Gathering Supreme. The relatively sparse rewards and extra actions in Deadly Corridor made tuning the hyperparameters and training more challenging. The learning rate schedule proved to be the most important factor for achieving model convergence. Once the model was trained and evaluation improved significantly, symptoms of over-fitting were encountered with evaluation runs displaying high initial scores that gradually worsened with each epoch. Dropout rates were increased to compensate for this and control over-fitting. There are three levels of dropout applied to the Transformer model, implying that over-fitting is a known feature of this type of model, including embedding, residual and attention dropout. Dropout rates were

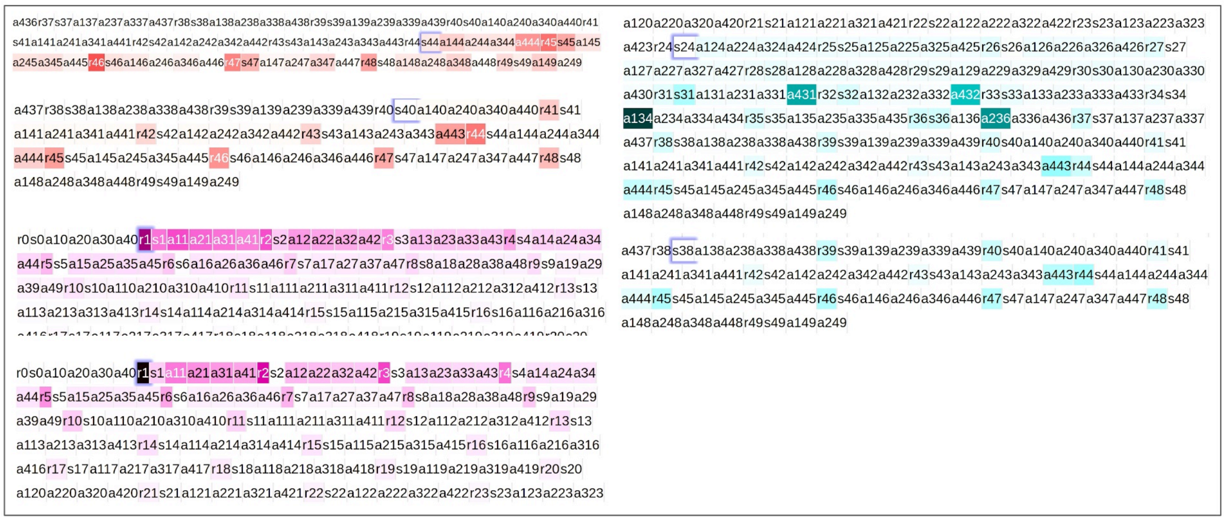


Figure 6.24: Left, top to bottom: L1H0, L3H0, L2H7, L5H7. Right, top to bottom: L6H4, L6H4

tuned, particularly the attention regularisation from the standard dropout rate of 0.1 to 0.3 (i.e. 30%). This improved evaluation scores when the batch size was increased to 128, and the drop off in performance by epoch was stabilised. It is postulated that other mechanisms for increasing regularisation that have been used recently in Decision Transformers with good effect (Liu et al., 2022; Wu et al., 2023a) would be useful, including masking. The model that performed best and continued to improve over epochs across multiple seeds was the M-SAT model. The state-action tokenisation stabilised the model significantly compared with the baseline and MAT models.

The loss plots display phases consistent with Transformer training curves as per (Olsson et al., 2022), including two steep drops (Figure 6.25), followed by a gradual decay.

Drop zone: there are two distinct drops in the loss curves as shown in Figure 6.25. It turns out that the learning rate schedule impacts the nature of the drop in the loss curve. The smaller the learning-rate related hyperparameter, viz. warmup (refer to Table 6.4.4.2), the steeper the learning rate schedule and the steeper the loss curve. Runs from the previous learning rate schedule have a gentler slope (see Figure 6.25, light green and violet plots). The steep learning rate schedule showed a dramatic improvement in performance, suggesting that a steep initial learning rate may be key to achieving the desired phase changes that are associated with in-context and induction learning as per Olsson et al. (2022), features of Transformers that are associated with generalisation. In order to succeed in Deadly Corridor, the agent requires two essential skills: navigation of the corridor and counter-attack. Without these skills, the agent would be terminated, so it is highly likely these skills are learnt early and result in the phase change visible in Figure 6.25.

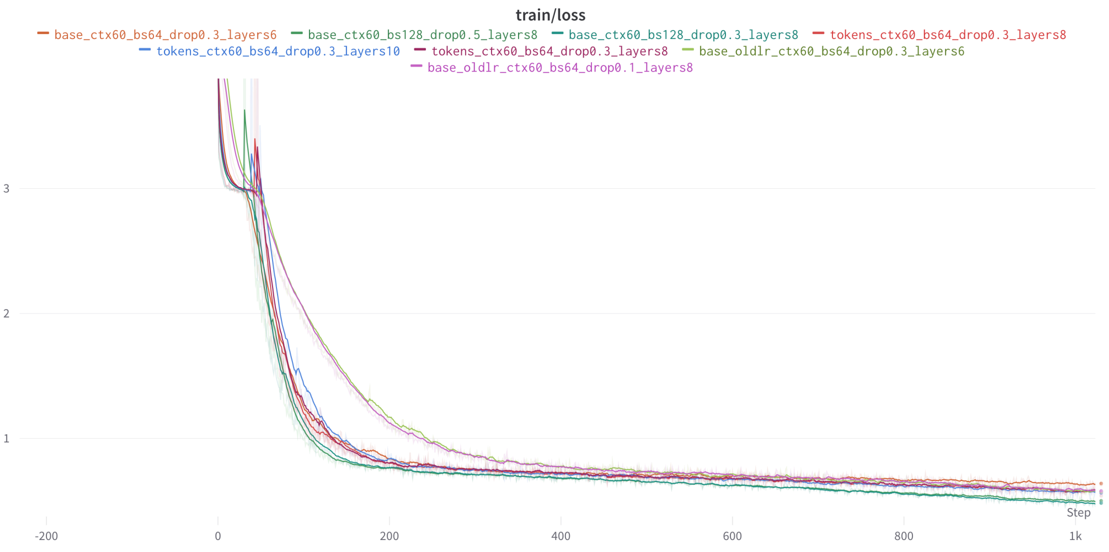


Figure 6.25: Transformer loss curve showing the initial drop corresponding to early learning rate increase

6.6 Conclusions

In this chapter, Decision Transformers were applied to the relational action problem in multi-discrete action spaces. The hypothesis was that expanding the multi-discrete action to multiple action tokens would make individual actions more visible to the attention mechanism of the Decision Transformer, leading to action-action and state-action relationships emerging, providing more information mixing opportunities to leverage the Transformer's relational abilities. Experiments were conducted using the multi-discrete ViZDoom Deadly Corridor scenario. A comparison of results with a baseline single action token model and ablative methods demonstrated that multiple action token models performed better overall, with the proposed M-SAT method outperforming other approaches. A further benefit of the multi-token models was the improved interpretability opportunities; with all actions exposed, attention heads were analysed at the individual action level.

Chapter 7

Conclusion and Future Work

Reinforcement learning agents must often select multiple discrete actions per timestep in environments with multi-discrete action spaces. Relationships often exist between these individual actions that can be utilised to improve agent performance. However, RL algorithms typically treat multi-discrete actions as single actions, missing opportunities to leverage relational insights.

The research in this thesis focuses on approaches in RL for exploiting action relationships in multi-discrete action spaces in online, offline and multi-task contexts. Methods were proposed and evaluated for each context to satisfy the original objectives outlined in Chapter 1, including

- learning and transferring task-agnostic, context-free action structure to enhance exploration in a multi-task, multi-discrete scenario
- the development of a method to support relational learning in an online multi-discrete RL algorithm
- an exploration of effective methods for managing relational structure in multi-discrete action spaces in an offline attention-based RL algorithm

A summary of each chapter is provided below, outlining the methods used, analyses performed and future research directions.

In Chapter 4 a multi-task approach (CASC) was designed to address the challenge of using relational structure in action spaces to improve exploration. The approach first extracted task-agnostic action structure in the form of clusters from data generated by a diverse range of tasks. Spectral clustering was used to extract relational structure in the action space. This relational structure was then transferred to a new agent, improving training performance for new, unseen tasks. The proposed approach was compared with other methods in the same multi-task, multi-action context, including action elimination and random clustering ablations. The clustering approach was competitive, showing much better performance than the baseline and random clustering ablations.

Chapter 5 addressed the challenge of adapting model-free online algorithms to capture action dependencies during training. An online auxiliary module for PPO was developed to reinforce beneficial action relationships and shape action representations by optimising a relational loss alongside PPO, demonstrating faster convergence over vanilla PPO. A self-supervised signal derived from training data determined which relationships to reinforce. Training dynamics techniques were used to analyse the emergence of structured representations and demonstrated that the relational module contributed to the earlier convergence.

Finally in Chapter 6 an offline RL approach utilising a Decision Transformer was applied to the relational action problem. Multi-discrete actions were expanded to multiple action tokens to facilitate the formation of relationships and support enhanced token mixing opportunities. The proposed model, M-SAT, outperformed the single action token baseline model. The state-action tokenisation proposed successfully provided more nuanced information to the Decision Transformer than no information or generic position encoding information. The multi-token approach improved the interpretability of the model. The use of attention rollout (Abnar and Zuidema, 2020) and attention head visualisation indicated that the attention heads made use of the individual action tokens when making decisions. This chapter demonstrated that applying different methods of tokenisation to multi-discrete action spaces processed by a Decision Transformer improved overall performance.

7.1 Future Work

Some future works, following the above chapters, are discussed below:

Multi-task transfer of action structure in multi-discrete action spaces

- A disadvantage of the proposed approach in Chapter 4, CASC, was the need for a pre-training phase that meant relationships were not learnt online. A potential avenue of research would be to make the existing algorithm more online, either by adopting an online clustering algorithm or by clustering intermittently during training. In the former case, modern data streaming applications have resulted in the adaptation of clustering algorithms for online, streaming scenarios (Yoo et al., 2016). The data generated by the RL agent could be envisioned as a stream, allowing the implementation of an online spectral clustering algorithm. The latter scenario, running spectral clustering every N runs, is potentially more easily implemented. Applying dynamically changing structure, however, to an RL agent during training would probably be destabilising. Some constraints might be necessary to influence either how much the structure was allowed to change, based on a clustering metric, or how much to use this structure during early stage training.

- Another area that would benefit from future work is the data collection phase. CASC is dependent on having good state-action space coverage. Traditional count-based bonus methods are used to improve coverage, providing a signal that diminishes over time as more of the space is explored. Merging CASC with a count-based method could help improve coverage efficiency and improve performance of the proposed approach generally.

Relational Representations in Multi-Discrete Action Spaces

- The general design of the relational module adopted in Chapter 5 was a relational reinforcing objective in the multi-discrete action space. It was supported by a derived signal indicating which relationships to reinforce. A positive signal was based on a positive transition and vice versa and the filter selected only successful transitions as the auxiliary signal. The negative signal was ignored and a more contrastive learning approach is proposed for future work. Contrastive learning (Chen et al., 2020) models both the similarity and dissimilarity of data points through an objective function. At a high level this would work by pushing similar actions closer together while pushing dissimilar actions apart. The inspiration for this approach was derived from skipgrams with noise contrastive estimation (NCE) Mikolov et al. (2013), that works by pushing a central word closer to other words in a window of words, while simultaneously pushing the central word away from words not in the window (noise). This could improve the performance of the current relational module proposed in Chapter 5.

- The interpretability of this model could be expanded, especially relating to the dynamics plots in Figure 5.12 that demonstrate the earlier convergence of the relational model compared with the baseline. Following the work by Saxe et al. (2013) it would be interesting to understand which action relationships formed first and what part of the architecture facilitated this, for example, would navigation-related actions be learnt first, followed by door-related actions? How does the availability of bottleneck states impact the relationships that are learnt first? This type of analysis would be interesting and provide better insight into how relational models learn online.

Action Structure in Decision Transformers

- The M-SAT approach as proposed in Chapter 6 increases the size of the context length by the number of discrete actions in a multi-discrete action, using a naive decomposition of actions. This will have an impact on the amount of compute required to train the model especially for multi-discrete action spaces with a large number of individual actions. Future work should consider reducing the number of action tokens by grouping actions and generating a token per group. Groups could be determined by clustering actions manually or similar unsupervised methods using the available training data. The M-SAT method demonstrates that structural information may be injected into Decision Transformer models successfully during tokenisation. Given this, it is envisaged that the grouping of actions can maintain the benefits from the increased number of tokens without degrading performance.

- Generalisation of transformer-style models requires environments where a diversity of tasks can generate data for training. In Chapter 6, the focus was on providing additional structural information to the transformer to foster the formation of state-action and action-action relationships. The generalisation of this approach is left for future work. It is envisaged that a procedurally generated environment would be more suitable for this type of work, such as ProcGen (Cobbe et al., 2020) but it is likely that more computational resources would be required.

- The interpretation of the shape of the training loss plots is proposed as future work, with a focus on plotting the loss by token as performed by Olsson et al.

(2022) to determine which tokens result in the initial steep drop in the loss. It is hypothesised that tokens in the vicinity of enemy attacks lead to the initial steep drops in training loss, possibly as a result of penalties when the agent is unsuccessful and is terminated. This analysis, and others in the area of interpretation, is left for future work.

The approaches proposed in this dissertation could be applied to many multi-discrete, real-world scenarios such as high level planning and robotic tasks. In general there are potential applications in transportation, language and recommendation systems too where enhancing learning in these structured action spaces can lead to more efficient and intuitive solutions.

In summary, this thesis has explored techniques to enable reinforcement learning agents to extract and utilise relational structure in multi-discrete action spaces across online, offline, and multi-task settings. The overarching goal was to improve sample efficiency, task performance, and generalisability by increasing the visibility of individual actions and providing algorithms with the capacity to exploit any relationships discovered. Overall, the thesis provided promising evidence that tailored mechanisms to expose and exploit relational attributes can enhance sample efficiency and generalisation. Multi-tokenisation and auxiliary modules are two techniques that warrant further exploration for leveraging structure in multi-discrete action spaces.

While challenges remain in validating and interpreting learned relationships, this research direction appears positive. Future work could expand evaluation to more complex domains and pursue methods of interpretation to better understand what concepts algorithms encode regarding action interactions. By improving algorithmic support for leveraging structure inherent in multi-discrete actions, RL systems may emerge that more efficiently solve tasks requiring coordinated behaviour.

Bibliography

S. Abnar and W. Zuidema. Quantifying attention flow in transformers. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4190–4197, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.385. URL https://aclanthology.org/2020.acl-main.385.

J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In D. Precup and Y. W. Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 22–31. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/achiam17a.html.

A. F. Agarap. Deep learning using rectified linear units (relu). CoRR, abs/1803.08375, 2018. URL http://arxiv.org/abs/1803.08375.

P. Agarwal, A. A. Rahman, P.-L. St-Charles, S. J. D. Prince, and S. E. Kahou. Transformers in reinforcement learning: A survey. July 2023. URL https://arxiv.org/abs/2307.05979.

A. Alekh, J. Nan, M. K. Sham, and W. Sun. Reinforcement Learning: Theory and Algorithms. Jan. 2022.

S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup. A survey of exploration methods in reinforcement learning. arXiv preprint:2109.00157, 2021.

J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In D. Precup and Y. W. Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 166–175. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/andreas17a.html`.

M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In International Conference on Learning Representations, 2021. URL `https://openreview.net/forum?id=nIAxjsniDzg`.

P. Auer. Using upper confidence bounds for online learning. In Proceedings 41st Annual Symposium on Foundations of Computer Science, pages 270–279, 2000. doi: 10.1109/SFCS.2000.892116.

P.-L. Bacon. Incremental skills discovery based on the bottleneck concept.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL `http://arxiv.org/abs/1409.0473`. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

M. Bain and C. Sammut. A framework for behavioural cloning. In Machine Intelligence 15, 1995. URL `https://api.semanticscholar.org/CorpusID:10738655`.

C. Bamford and A. Ovalle. Generalising discrete action spaces with conditional action trees. 2021 IEEE Conference on Games (CoG), pages 1–8, 2021. URL `https://api.semanticscholar.org/CorpusID:233240936`.

M. Bany Muhammad and M. Yeasin. Eigen-cam: Visual explanations for deep convolutional neural networks. SN Comput. Sci., 2(1), jan 2021. doi: 10.1007/s42979-021-00449-3. URL `https://doi.org/10.1007/s42979-021-00449-3`.

A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf`.

A. Barreto, D. Borsa, S. Hou, G. Comanici, E. Aygün, P. Hamel, D. Toyama, J. Hunt, S. Mourad, D. Silver, and D. Precup. The option keyboard: Combining skills in reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 13031–13041. Curran Associates, Inc., 2019.

D. Barrett, F. Hill, A. Santoro, A. Morcos, and T. Lillicrap. Measuring abstract reasoning in neural networks. In J. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 511–520. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/barrett18a.html`.

P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper_files/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf`.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Çaglar Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. M. O. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. ArXiv, abs/1806.01261, 2018.

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based exploration and intrinsic motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 1471–1479. Curran Associates, Inc., 2016.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, pages 4148–4152. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL http://dl.acm.org/citation.cfm?id=2832747.2832830.

R. Bellman. The theory of dynamic programming. Bull. Amer. Math. Soc., 60 (6):503–515, 11 1954. URL https://projecteuclid.org:443/euclid.bams/1183519147.

C. M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, 1 edition, 2007. ISBN 0387310738.

D. Brandfonbrener, A. Bietti, J. Buckman, R. Laroche, and J. Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=XByg4kotW5.

T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language models with dictionary learning. Transformer Circuits Thread, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

N. Brown and T. Sandholm. Superhuman ai for multiplayer poker. Science, 365:885 – 890, 2019. URL https://api.semanticscholar.org/CorpusID:195892791.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement learning from demonstration through shaping. In International Joint Conference on Artificial Intelligence, 2015. URL https://api.semanticscholar.org/CorpusID:1557568.

M. Carroll, O. Paradise, J. Lin, R. Georgescu, M. Sun, D. Bignell, S. Milani, K. Hofmann, M. Hausknecht, A. Dragan, and S. Devlin. Uni[MASK]: Unified inference in sequential decision problems. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=GisHNaleWiA.

Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. Thomas. Learning action representations for reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 941–950, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/chandak19a.html.

Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. D. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. 2019 International Conference on Robotics and Automation (ICRA), pages 8973–8979, 2018. URL https://api.semanticscholar.org/CorpusID:53046511.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 15084–15097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In H. D. III and A. Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 1597–1607. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Y. Chen, Y. Chen, Y. Yang, Y. Li, J. Yin, and C. Fan. Learning action-transferable policy with action embedding. CoRR, abs/1909.02291, 2019. URL http://arxiv.org/abs/1909.02291.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS 2016, page 7–10, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347952. doi: 10.1145/2988450.2988454. URL https://doi.org/10.1145/2988450.2988454.

M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. CoRR, abs/2306.13831, 2023.

M. Chi, P. J. Feltovich, and R. Glaser. Categorization and representation of physics problems by experts and novices. Cogn. Sci., 1981.

R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta. Intrinsic motivation for encouraging synergistic behavior. In International Conference on Learning Representations, 2020. URL https://openreview.net/forum?id=SJleNCNtDH.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In H. D. III and A. Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 2048–2056. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/cobbe20a.html.

A. Cooney. Circuitsviz, 2022. URL https://github.com/alan-cooney/CircuitsVis.

T. L. Dean, R. Givan, and K.-E. Kim. Solving stochastic planning problems with large state and action spaces. In International Conference on Artificial Intelligence Planning Systems, 1998. URL https://api.semanticscholar.org/CorpusID:2029722.

O. Delalleau, M. Peter, E. Alonso, and A. Logut. Discrete and continuous action representation for practical RL in video games. CoRR, abs/1912.11077, 2019. URL http://arxiv.org/abs/1912.11077.

H. Deng, Q. Ren, H. Zhang, and Q. Zhang. DISCOVERING AND EXPLAINING THE REPRESENTATION BOTTLENECK OF DNNS. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=iRCUlgmdfHJ.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Y. Du and K. Narasimhan. Task-agnostic dynamics priors for deep reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 1696–1705. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/du19e.html.

G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin. Reinforcement learning in large discrete action spaces. CoRR, abs/1512.07679, 2015. URL http://arxiv.org/abs/1512.07679.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. Nature, 590:580—586, 2021. URL https://doi.org/10.1038/s41586-020-03157-9.

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, and others. A mathematical framework for transformer circuits. Transformer Circuits Thread, 2021.

S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. Rvs: What is essential for offline RL via supervised learning? In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=S874XAIpkR-.

E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping conditions for reinforcement learning. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), pages 162–169, 2003.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In ICLR, 2019, 2019. URL https://openreview.net/pdf?id=SJx63jRqFm.

B. Eysenbach, S. Chaudhari, S. Asawa, S. Levine, and R. Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=eqBwg3AcIAK.

M. F. A. R. D. T. (FAIR), A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, A. P. Jacob, M. Komeili, K. Konath, M. Kwon, A. Lerer, M. Lewis, A. H. Miller, S. Mitts, A. Renduchintala, S. Roller, D. Rowe, W. Shi, J. Spisak, A. Wei, D. Wu, H. Zhang, and M. Zijlstra. Human-level play in the game of <i>diplomacy</i> by combining language models with strategic reasoning. Science, 378(6624):1067–1074, 2022. doi: 10.1126/science.ade9097. URL `https://www.science.org/doi/abs/10.1126/science.ade9097`.

L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2022. URL `https://openreview.net/forum?id=rc8o_j8I8PX`.

M. Fatemi, T. W. Killian, J. Subramanian, and M. Ghassemi. Medical dead-ends and learning to identify high-risk states and treatments. CoRR, abs/2110.04186, 2021. URL `https://arxiv.org/abs/2110.04186`.

L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(4):594–611, 2006. doi: 10.1109/TPAMI.2006.79.

C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. CoRR, abs/1701.08734, 2017. URL `http://arxiv.org/abs/1701.08734`.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In ICML, 2017.

J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. Lee,

M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf.

V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning, 2018. URL http://arxiv.org/abs/1811.12560. cite arxiv:1811.12560.

J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: datasets for deep data-driven reinforcement learning. CoRR, abs/2004.07219, 2020. URL https://arxiv.org/abs/2004.07219.

Y. Ganin, T. Kulkarni, I. Babuschkin, S. M. A. Eslami, and O. Vinyals. Synthesizing programs for images using reinforced adversarial learning. In ICML, 2018.

M. Garnelo, K. Arulkumaran, and M. Shanahan. Towards deep symbolic reinforcement learning. CoRR, abs/1609.05518, 2016. URL http://arxiv.org/abs/1609.05518.

J. Gildenblat. Exploring explainability for vision transformers, 2020. URL https://jacobgil.github.io/deeplearning/vision-transformer-explainability.

G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. Numerische Mathematik, 14(5):403–420, 1970. ISSN 0945-3245. doi: 10.1007/BF02163027. URL https://doi.org/10.1007/BF02163027.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, Advances in

Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. Adaptive computation and machine learning. MIT Press, 2016. ISBN 9780262035613. URL https://books.google.co.in/books?id=Np9SDQAAQBAJ.

A. Goyal, R. Islam, D. Strouse, Z. Ahmed, H. Larochelle, M. Botvinick, S. Levine, and Y. Bengio. Transfer and exploration via the information bottleneck. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=rJg8yhAqKm.

X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/8bb88f80d334b1869781beb89f7b73be-Paper.pdf.

W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al. The MineRL competition on sample efficient reinforcement learning using human priors. NeurIPS Competition Track, 2019.

K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang, and D. Tao. A survey on vision transformer. IEEE Transactions on Pattern Analysis amp; Machine Intelligence, 45(01):87–110, jan 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3152247.

J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup. When waiting is not an option: Learning options with a deliberation cost. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

J. Harmer, L. Gisslén, J. del Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin. Imitation learning with concurrent actions in 3d games. In 2018 IEEE Conference on Computational Intelligence and Games (CIG), page 1–8. IEEE Press, 2018. doi: 10.1109/CIG.2018.8490398. URL `https://doi.org/10.1109/CIG.2018.8490398`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

P. Hernandez-Leal, B. Kartal, and M. E. Taylor. Tutorial 4: auxiliary tasks in deep reinforcement learning. URL `https://www.borealisai.com/research-blogs/tutorial-4-auxiliary-tasks-deep-reinforcement-learning/`.

P. Hernandez-Leal, B. Kartal, and M. E. Taylor. Agent modeling as auxiliary task for deep reinforcement learning. In Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'19. AAAI Press, 2019. ISBN 978-1-57735-819-0.

I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In D. Precup and Y. W. Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 1480–1490. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/higgins17a.html`.

I. Higgins, D. Amos, D. Pfau, S. Racanière, L. Matthey, D. J. Rezende, and A. Lerchner. Towards a definition of disentangled representations. CoRR, abs/1812.02230, 2018. URL http://arxiv.org/abs/1812.02230.

F. Hill, A. Santoro, D. Barrett, A. Morcos, and T. Lillicrap. Learning to make analogies by contrasting abstract relational structure. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=SylLYsCcFm.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. CoRR, abs/1503.02531, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1503.html#HintonVD15.

S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

R. Houthooft, X. Chen, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/abd815286ba1007abfbb8415b83ae2cf-Paper.pdf.

R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. Learning to reason: End-to-end module networks for visual question answering. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 804–813, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society. doi: 10.1109/ICCV.2017.93. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.93.

S. Huang and S. Ontan'on. A closer look at invalid action masking in policy gradient algorithms. ArXiv, abs/2006.14171, 2020. URL https://api.semanticscholar.org/CorpusID:220055586.

S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. AraÃºjo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. Journal of Machine Learning Research, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

A. Irpan. Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In International Conference on Learning Representations, 2017. URL https://openreview.net/forum?id=SJ6yPD5xg.

A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira. Perceiver: General perception with iterative attention. In M. Meila and T. Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 4651–4664. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jaegle21a.html.

A. Jain, N. Kosaka, K. Kim, and J. J. Lim. Know your action set: Learning action relations for reinforcement learning. ICLR, 2022.

M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 1273–1286. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/099fe6b0b444c23836c4a5d07346082b-Paper.pdf.

Z. Jia, F. Liu, V. Thumuluri, L. Chen, Z. Huang, and H. Su. Chain-of-thought predictive control. In Workshop on Reincarnating Reinforcement Learning at ICLR 2023, 2023. URL `https://openreview.net/forum?id=TIV7eEY8qY`.

Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan. Vima: Robot manipulation with multimodal prompts. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org, 2023.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1): 99–134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL `https://www.sciencedirect.com/science/article/pii/S000437029800023X`.

A. Kanervisto, C. V. Scheller, and V. Hautamäki. Action space shaping in deep reinforcement learning. 2020 IEEE Conference on Games (CoG), pages 479–486, 2020. URL `https://api.semanticscholar.org/CorpusID:214775114`.

A. Kazemnejad, I. Padhi, K. N. Ramamurthy, P. Das, and S. Reddy. The impact of positional encoding on length generalization in transformers. ArXiv, abs/2305.19466, 2023. URL `https://api.semanticscholar.org/CorpusID:258987259`.

M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. 2016 IEEE Conference on Computational Intelligence and Games (CIG), pages 1–8, 2016. URL `https://api.semanticscholar.org/CorpusID:430714`.

G. Kerg, S. Mittal, D. Rolnick, Y. Bengio, B. A. Richards, and G. Lajoie. Inductive biases for relational tasks. In ICLR2022 Workshop on the Elements of Reasoning:

Objects, Structure and Causality, 2022. URL https://openreview.net/forum?id=BSgxIBuI5lq.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. jair, 76:201–264, Jan. 2023.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, Advances in Neural Information Processing Systems, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

E. Kreyszig. Advanced Engineering Mathematics. John Wiley, Hoboken, NJ, ninth edition, 2006. ISBN 0471488852 0471728977 9780471728979 0471726443 9780471726449 0471726451 9780471726456 047172646X 9780471726463 9780471488859.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. CoRR, abs/1402.6028, 2014. URL http://arxiv.org/abs/1402.6028.

T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. ArXiv, abs/1606.02396, 2016. URL https://api.semanticscholar.org/CorpusID:11965834.

A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c2073ffa77b5357a498057413bb09d3a-Paper.pdf.

P. Ladosz, L. Weng, M. Kim, and H. Oh. Exploration in deep reinforcement learning: A survey. Inf. Fusion, 85(C):1–22, sep 2022a. ISSN 1566-2535. doi: 10.1016/j.inffus.2022.03.003. URL https://doi.org/10.1016/j.inffus.2022.03.003.

P. Ladosz, L. Weng, M. Kim, and H. Oh. Exploration in deep reinforcement learning: A survey. Information Fusion, 85:1–22, 2022b. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2022.03.003. URL https://www.sciencedirect.com/science/article/pii/S1566253522000288.

A. K. Lampinen, N. Roy, I. Dasgupta, S. C. Chan, A. Tam, J. Mcclelland, C. Yan, A. Santoro, N. C. Rabinowitz, J. Wang, and F. Hill. Tell me why! Explanations support learning relational and causal structure. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 11868–11890. PMLR, 2022.

S. Lange, T. Gabel, and M. Riedmiller. Batch Reinforcement Learning, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_2. URL https://doi.org/10.1007/978-3-642-27645-3_2.

M. Laskin, L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. S. Hansen, A. Filos, E. Brooks, maxime gazeau, H. Sahni, S. Singh, and V. Mnih. In-context reinforcement learning with algorithm distillation. In The Eleventh International Conference on Learning Representations, 2023. URL https://openreview.net/forum?id=hy0a5MMPUv.

K.-H. Lee, O. Nachum, S. Yang, L. Lee, C. D. Freeman, S. Guadarrama, I. Fischer, W. Xu, E. Jang, H. Michalewski, and I. Mordatch. Multi-game decision transformers. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=0gouO5saq6K.

J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon. FNet: Mixing tokens with Fourier transforms. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4296–4313, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.319. URL https://aclanthology.org/2022.naacl-main.319.

S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. CoRR, abs/2005.01643, 2020. URL https://arxiv.org/abs/2005.01643.

W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye. A survey on transformers in reinforcement learning. Transactions on Machine Learning Research, 2023a. ISSN 2835-8856. URL https://openreview.net/forum?id=r30yuDPvf2. Survey Certification.

W. Li, B. Wang, S. Yang, and H. Zha. Diverse policy optimization for structured action space. In Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '23, page 819–828, Richland, SC, 2023b. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321.

F. Liu, H. Liu, A. Grover, and P. Abbeel. Masked autoencoding for scalable and generalizable decision making. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, Advances in Neural Information Processing Systems, volume 35, pages 12608–12618. Curran Associates, Inc.,

2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/51fda94414996902ddaaa35561b97294-Paper-Conference.pdf`.

J. Luo, C. Paduraru, O. Voicu, Y. Chervonyi, S. A. Munns, J. Z. Li, C. Qian, P. Dutta, J. Q. Davis, N. Wu, X. Yang, C.-M. Chang, T. Li, R. Rose, M. Fan, H. Nakhost, T. Liu, B. Kirkman, F. Altamura, L. Cline, P. Tonker, J. P. Gouker, D. Udén, W. B. Bryan, J. Law, D. Fatiha, N. Satra, J. Rothenberg, M. A. Carlin, S. Tallapaka, S. Witherspoon, D. Parish, P. Dolan, C. Zhao, and D. J. Mankowitz. Controlling commercial cooling systems using reinforcement learning. ArXiv, abs/2211.07357, 2022. URL `https://api.semanticscholar.org/CorpusID:253510192`.

U. Luxburg. A tutorial on spectral clustering. Statistics and Computing, 17(4): 395–416, Dec. 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. URL `http://dx.doi.org/10.1007/s11222-007-9033-z`.

M. C. Machado, M. G. Bellemare, and M. Bowling. A laplacian framework for option discovery in reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 2295–2304. JMLR.org, 2017.

W. Masson, P. Ranchod, and G. Konidaris. Reinforcement learning with parameterized actions. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1), Feb. 2016. doi: 10.1609/aaai.v30i1.10226. URL `https://ojs.aaai.org/index.php/AAAI/article/view/10226`.

M. Mathieu, S. Ozair, S. Srinivasan, C. Gulcehre, S. Zhang, R. Jiang, T. Le Paine, R. Powell, K. Żołna, J. Schrittwieser, D. Choi, P. Georgiev, D. Toyama, A. Huang, R. Ring, I. Babuschkin, T. Ewalds, M. Bordbar, S. Henderson, S. G. Colmenarejo, A. van den Oord, W. M. Czarnecki, N. de Freitas, and O. Vinyals. AlphaStar unplugged: Large-Scale offline reinforcement learning. Aug. 2023.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781, 2013. URL http://dblp.uni-trier.de/db/journals/corr/corr1301.html/abs-1301-3781.

A. Mirhoseini, A. Goldie, M. Yazgan, J. W. J. Jiang, E. M. Songhori, S. Wang, Y. Lee, E. Johnson, O. Pathak, S. Bae, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, A. Babu, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean. Chip placement with deep reinforcement learning. CoRR, abs/2004.10746, 2020. URL https://arxiv.org/abs/2004.10746.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. CoRR, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

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

P. Moodley, B. Rosman, and X. Hong. Understanding structure of concurrent actions. In International Conference on Innovative Techniques and Applications of Artificial Intelligence, pages 78–90. Springer, 2019.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/7ec69dd44416c46745f6edd947b470cd-Paper.pdf.

N. Nanda and J. Bloom. Transformerlens, 2022. URL https://github.com/neelnanda-io/TransformerLens.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, Advances in Neural Information Processing Systems 14, pages 849–856. MIT Press, 2002. URL http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf.

T. Ni, B. Eysenbach, S. Levine, and R. Salakhutdinov. Recurrent model-free RL is a strong baseline for many POMDPs, 2022. URL https://openreview.net/forum?id=E0zOKxQsZhN.

C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads. Transformer Circuits Thread, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell. Zero-shot learning with semantic output codes. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, Advances in Neural Information Processing Systems, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper_files/paper/2009/file/1543843a4723ed2ab08e18053ae6dc5b-Paper.pdf.

S. Parisi, V. Dean, D. Pathak, and A. Gupta. Interesting object, curious agent: Learning task-agnostic exploration. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, Advances in Neural Information Processing Systems, 2021. URL `https://openreview.net/forum?id=knKJgksd7kA`.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, High-Performance deep learning library. Dec. 2019.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In D. Precup and Y. W. Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 2778–2787. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/pathak17a.html`.

A. Patterson, S. Neumann, M. White, and A. White. Empirical design in reinforcement learning. Apr. 2023.

J. Pazis and R. Parr. Generalized value functions for large action sets. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pages 1185–1192, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

A. Petrenko, Z. Huang, T. Kumar, G. S. Sukhatme, and V. Koltun. Sample factory: Egocentric 3d control from pixels at 100000 FPS with asynchronous reinforcement learning. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 7652–7662. PMLR, 2020. URL http://proceedings.mlr.press/v119/petrenko20a.html.

M. Popel and O. Bojar. Training tips for the transformer model. CoRR, abs/1804.00247, 2018. URL http://arxiv.org/abs/1804.00247.

D. Precup. Temporal abstraction in reinforcement learning. 2000. URL https://scholarworks.umass.edu/dissertations/AAI9978540.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2018. URL https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

A. Raghu, M. Komorowski, I. Ahmed, L. A. Celi, P. Szolovits, and M. Ghassemi. Deep reinforcement learning for sepsis treatment. CoRR, abs/1711.09602, 2017. URL http://arxiv.org/abs/1711.09602.

R. Raileanu and T. Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In International Conference on Learning Representations, 2020. URL https://openreview.net/forum?id=rkg-TJBFPB.

J. Randlov. Learning Macro-Actions in reinforcement learning. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, Advances in Neural Information Processing Systems 11, pages 1045–1051. MIT Press, 1999.

S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. A generalist agent. Transactions on Machine Learning Research, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=1ikK0kHjvj. Featured Certification, Outstanding Certification.

D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration. International Conference on Learning Representations (ICLR). URL https://par.nsf.gov/biblio/10327543.

C. Robert and G. Casella. A short history of markov chain monte carlo: Subjective recollections from incomplete data. Statistical Science, 26(1), Feb. 2011. ISSN 0883-4237. doi: 10.1214/10-sts351. URL http://dx.doi.org/10.1214/10-STS351.

B. Rosman and S. Ramamoorthy. What good are actions? accelerating learning using learned action priors. In 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL), pages 1–6, Nov. 2012.

B. Rosman and S. Ramamoorthy. Action priors for learning domain invariances. IEEE Trans. Auton. Ment. Dev., 7(2):107–118, June 2015.

S. Russell. Learning agents for uncertain environments (extended abstract). In Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT' 98, page 101–103, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130570. doi: 10.1145/279943.279964. URL https://doi.org/10.1145/279943.279964.

S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 3 edition, 2010.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. CoRR, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

T. Salimans and R. Chen. Learning montezuma's revenge from a single demonstration. Dec. 2018.

B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. J. Mach. Learn. Res., 5(Aug):1063–1088, 2004.

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In J. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 4470–4479, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/e6acf4b0f69f6f6e60e9a815938aa1ff-Paper.pdf.

B. Sart-Tilman. Tree-based batch mode reinforcement learning. https://www.jmlr.org/papers/volume6/ernst05a/ernst05a.pdf, 2005. Accessed: 2021-7-10.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. CoRR, abs/1312.6120, 2013. URL https://api.semanticscholar.org/CorpusID:17272965.

S. Schaal. Learning from demonstration. In M. Mozer, M. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems, volume 9. MIT Press, 1996. URL https://proceedings.neurips.cc/paper_files/paper/1996/file/68d13cf26c4b4f4f932e3eff990093ba-Paper.pdf.

S. Schaal. Is imitation learning the route to humanoid robots? Trends in Cognitive Sciences, 3(6):233–242, 1999. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01327-3. URL https://www.sciencedirect.com/science/article/pii/S1364661399013273.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. Bach and D. Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 1312–1320, Lille, France, 2015a. PMLR.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. Bach and D. Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 1312–1320, Lille, France, 07–09 Jul 2015b. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. Nov. 2015c.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

M. Shanahan, K. Nikiforou, A. Creswell, C. Kaplanis, D. Barrett, and M. Garnelo. An explicitly relational neural network architecture. In Proceedings of the 37th International Conference on Machine Learning, ICML'20. JMLR.org, 2020.

M. Sharma, M. Tong, T. Korbak, D. Duvenaud, A. Askell, S. R. Bowman, N. Cheng, E. Durmus, Z. Hatfield-Dodds, S. R. Johnston, S. Kravec, T. Maxwell, S. McCandlish, K. Ndousse, O. Rausch, N. Schiefer, D. Yan, M. Zhang, and E. Perez. Towards understanding sycophancy in language models, 2023.

S. Sharma, A. Suresh, R. Ramesh, and B. Ravindran. Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. CoRR, abs/1705.07269, 2017. URL `http://arxiv.org/abs/1705.07269`.

E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. CoRR, abs/1612.07307, 2016. URL `http://arxiv.org/abs/1612.07307`.

A. A. Sherstov and P. Stone. Improving action selection in MDP's via knowledge transfer. AAAI, 2005.

D. Silver. Lectures on reinforcement learning. URL: `https://www.davidsilver.uk/teaching/`, 2015.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In E. P. Xing and T. Jebara, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL `https://proceedings.mlr.press/v32/silver14.html`.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. Nature, 550:354–, Oct. 2017. URL `http://dx.doi.org/10.1038/nature24270`.

S. P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. Machine Learning, 8(3):323–339, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992700. URL `https://doi.org/10.1007/BF00992700`.

S. C. Suddarth and Y. L. Kergosien. Rule-injection hints as a means of improving network performance and learning time. In Neural Networks, Lecture notes in computer science, pages 120–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, Advances in Neural Information Processing Systems, volume 8. MIT Press, 1995. URL `https://proceedings.neurips.cc/paper_files/paper/1995/file/8f1d43620bc6bb580df6e80b0dc05c48-Paper.pdf`.

R. S. Sutton and A. G. Barto. Reinforcement learning - an introduction. Adaptive computation and machine learning. MIT Press, 1998a. ISBN 978-0-262-19398-6. URL `https://www.worldcat.org/oclc/37293240`.

R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998b. URL `http://www.cs.ualberta.ca/~sutton/book/the-book.html`.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, Advances in Neural Information Processing Systems, volume 12. MIT Press, 1999. URL `https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`.

H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. #exploration: A study of Count-Based exploration for deep reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 2753–2762. Curran Associates, Inc., 2017.

J. Taylor and N. Kriegeskorte. Extracting and visualizing hidden activations and computational graphs of PyTorch models with TorchLens. Scientific Reports, 13(1):14375, 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-40807-0. URL https://doi.org/10.1038/s41598-023-40807-0.

G. Tennenholtz and S. Mannor. The natural language of actions. In K. Chaudhuri and R. Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 6196–6205, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/tennenholtz19a.html.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, pages 385–392. MIT Press, 1995.

J. N. Tsitsiklis and B. Van Roy. Analysis of Temporal-Diffference learning with function approximation. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems 9, pages 1075–1081. MIT Press, 1997.

E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. CoRR, abs/1412.3474, 2014. URL http://arxiv.org/abs/1412.3474.

O. University. Oxford English Dictionary. (2016) reference, v. 3. Online. Oxford University, 2016. URL http://www.oed.com/view/Entry/160845.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothorl, T. Lampe, and M. A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. CoRR, abs/1707.08817, 2017. URL http://arxiv.org/abs/1707.08817.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In International Conference on Learning Representations, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Starcraft II: A new challenge for reinforcement learning. CoRR, abs/1708.04782, 2017. URL http://arxiv.org/abs/1708.04782.

N. Vithayathil Varghese and Q. H. Mahmoud. A survey of multi-task deep reinforcement learning. Electronics, 9(9), 2020. ISSN 2079-9292. doi: 10.3390/electronics9091363. URL https://www.mdpi.com/2079-9292/9/9/1363.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. May 2023.

H. Wang and Y. Yu. Exploring multi-action relationship in reinforcement learning. In Pacific Rim International Conference on Artificial Intelligence, pages 574–587. Springer, 2016.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos,

C. Blundell, D. Kumaran, and M. M. Botvinick. Learning to reinforcement learn. CoRR, abs/1611.05763, 2016. URL http://arxiv.org/abs/1611.05763.

T. W. Webb, Z. Dulberg, S. M. Frankland, A. A. Petrov, R. C. O'Reilly, and J. D. Cohen. Learning representations that support extrapolation. In Proceedings of the 37th International Conference on Machine Learning, ICML'20. JMLR.org, 2020.

T. W. Webb, I. Sinha, and J. Cohen. Emergent symbols through binding in external memory. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=LSFCEb3GYU7.

Wikipedia contributors. Chatgpt — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=ChatGPT&oldid=1180697478, 2023a. [Online; accessed 19-October-2023].

Wikipedia contributors. Gpt-4 — Wikipedia, the free encyclopedia, 2023b. URL https://en.wikipedia.org/w/index.php?title=GPT-4&oldid=1192821203. [Online; accessed 1-January-2024].

D. Wingate, N. D. Goodman, D. M. Roy, L. P. Kaelbling, and J. B. Tenenbaum. Bayesian policy search with policy priors. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two, IJCAI'11, page 1565–1570. AAAI Press, 2011. ISBN 9781577355144.

P. Wu, A. Majumdar, K. Stone, Y. Lin, I. Mordatch, P. Abbeel, and A. Rajeswaran. Masked trajectory models for prediction, representation, and control. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org, 2023a.

Y.-H. Wu, X. Wang, and M. Hamaya. Elastic decision transformer. ArXiv, abs/2307.02484, 2023b. URL https://api.semanticscholar.org/CorpusID:259342857.

K. Xu, M. Zhang, J. Li, S. S. Du, K.-I. Kawarabayashi, and S. Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In International Conference on Learning Representations, 2021. URL `https://openreview.net/forum?id=UH-cmocLJC`.

M. Xu, Y. Shen, S. Zhang, Y. Lu, D. Zhao, J. Tenenbaum, and C. Gan. Prompting decision transformer for Few-Shot policy generalization. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 24631–24645. PMLR, 2022.

S. Yang, O. Nachum, Y. Du, J. Wei, P. Abbeel, and D. Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. ArXiv, abs/2303.04129, 2023. URL `https://api.semanticscholar.org/CorpusID:257378587`.

S. Yoo, H. Huang, and S. P. Kasiviswanathan. Streaming spectral clustering. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pages 637–648, 2016. doi: 10.1109/ICDE.2016.7498277.

C. Yu, J. Liu, and S. Nemati. Reinforcement learning in healthcare: A survey. CoRR, abs/1908.08796, 2019. URL `http://arxiv.org/abs/1908.08796`.

X. Yuan, M. Côté, A. Sordoni, R. Laroche, R. T. des Combes, M. J. Hausknecht, and A. Trischler. Counting to explore and generalize in text-based games. CoRR, abs/1806.11525, 2018. URL `http://arxiv.org/abs/1806.11525`.

T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 3562–3573. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/`

7615-learn-what-not-to-learn-action-elimination-with-deep-reinforcement-learn
pdf.

V. F. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. P. Reichert, T. P. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. M. Botvinick, O. Vinyals, and P. W. Battaglia. Relational deep reinforcement learning. ArXiv, abs/1806.01830, 2018. URL https://api.semanticscholar.org/CorpusID:46939951.

M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. ArXiv, abs/2309.02473, 2023. URL https://api.semanticscholar.org/CorpusID:261557281.

Y. Zhang and Q. Yang. A survey on multi-task learning. IEEE Transactions on Knowledge and Data Engineering, 34(12):5586–5609, 2022. doi: 10.1109/TKDE.2021.3070203.

B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen. Imitation learning: Progress, taxonomies and challenges. IEEE Trans Neural Netw Learn Syst, PP, Oct. 2022.

Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(11):13344–13362, 2023. doi: 10.1109/TPAMI.2023.3292075.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In K. Chaudhuri and R. Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 7693–7702. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/zintgraf19a.html.