Deep Reinforcement Learning Methods for Automated Workflow Construction in Large Scale Open Distributed Systems

A thesis submitted in partial fulfilment

of the requirement for the degree of Doctor of Philosophy

Laura A. D'Arcy

May 2023

Cardiff University
School of Computer Science & Informatics

Abstract

Large-scale distributed and decentralized systems often require access to multiple services, leading to the construction of complex workflows that can be difficult to design manually. This thesis proposes to use Deep Reinforcement Learning (DRL) techniques to create the optimal workflow without human intervention. The proposed hypothesis is based on using DRL algorithms combined with various styles of encoding such as Symbolic Vector Architecture and Knowledge Graph Embeddings, to handle larger and more complex systems. The approach utilizes both hierarchical and multi-task reinforcement learning. The benefit of using DRL in workflow construction is its ability to adapt to dynamic systems, where services are continuously added or removed, and systems change in quality. Our proposed approach can learn to adapt to changes in the system and find suitable alternatives.

Contents

List of Publications

The work introduced in this thesis is based on the following publications.

- D'Arcy, L., Corcoran, P., Preece, A (2019). Deep Q-Learning for Directed Acyclic Graph Generation. *Learning and Reasoning with Graph-Structured Representations, ICML 2019.*
- Millar, D., Braines, D., D'Arcy, L., Barclay, I., Summers-Stay, D., Cripps, P.J. (2021). Embedding Dynamic Knowledge Graphs Based on Observational Ontologies in Semantic Vector Spaces. *SPIE Defense + Commercial Sensing.*

List of Figures

List of Tables

To Mum
for your endless patience and support.

Acknowledgements

I would like to acknowledge Padraig Corcoran, Alun Preece, Ian Taylor and Declan Millar for their invaluable advice throughout the course of this PhD.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF–16–3–0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Chapter 1

Introduction

Software has come a long way since its inception. It used to be created to complete a single task, such as performing basic mathematical calculations, managing a set of company accounts, hosting an information database or playing a game. Requirements were relatively static, and development was slow but it was easier to understand what the system was doing, how it was doing it and the services that it relied upon. Similarly, the people that used these systems had access to a limited set of data and services, they knew what was available and how it was provided and they had an inherent understanding of its trustworthiness and value. If the computer they used wasn't working or some part of the supply chain of that information was down they had to wait for repair or revert to manual or backup solutions.

As technology has advanced, people have access to vast amounts of data stored worldwide and millions of applications. We can interact with this data and these applications via a range of devices, from phones and computers down to smart doorbells and thermostats. To find that data or those applications, people use search engines and review sites to locate these resources and to (attempt to) assess their value and quality.

There is a wide array of choices. If a source for stock data is unavailable or slow we can use another. If Google Maps has been updated and not reporting traffic correctly, use Apple Maps. Similarly, the services we use are themselves using other services to achieve their goals. When you access a shopping site, before you have even seen a product,the site will have consulted multiple services to determine your location, your

language, your recent interests and your probable spending power before deciding what products to show you and what prices to charge for them.

The shopping site example above is a very simple example. In the above case, the services used can easily be pre-approved and selected from a simple list. In an ever more complex world we are trying to use technology to answer more and more complex questions to achieve goals that would have once been infeasible in a rapid or automated manner (e.g. drive a car, defend against a drone attack, reroute traffic after an accident to reduce traffic delays, monitor a chat room for inappropriate comments). In examples such as these we are striving for the best possible result and must be able to cope with the absence of a given service without a major degradation of output.

These complex systems often involve multiple components, each with their own owners and methods of access and use. These components are constantly changing, and as a result, the workflows that they are part of also change quickly. This requires a robust system architecture that is able to adapt to these constantly changing workflows, without needing human intervention for each change.

Many of these workflows cannot be designed by hand, as they are too complex and constantly evolving. Construction of these workflows can be formulated as a Reinforcement Learning (RL) problem. Reinforcement learning is an area of machine learning that is specifically designed to work within dynamic environments; it allows an agent to learn how to make decisions based on the feedback it receives from the environment. The main principle of RL is that an agent learns to respond to stimuli by maximizing a cumulative reward signal. This signal represents the long-term value of the agent's decisions and actions in the environment. The primary benefits of reinforcement learning are its ability to learn in complex and uncertain environments - a distributed system is both complex due to the size and uncertain due to the constant change of services within those systems. This makes RL the ideal tool for the automated construction of workflows.

Automated construction of workflows goes hand in hand with automated representation

of workflows, using representation learning. Representation learning is a technique that involves automatically learning useful features or representations of data. Rather than relying on handcrafted features, representation learning algorithms are able to discover features that are important for a particular task from raw data. This approach has been shown to be particularly useful for tasks such as image recognition and natural language processing.

In the context of distributed systems, representation learning can be used to represent workflows that combine multiple tasks and services across multiple machines. Due to its ability to capture complex relationships, representation learning enables the development of generalisable and scalable models that can handle diverse data types and variations. Workflows represented using learned representations can provide better insights into the distributed system's performance and improve the communication and coordination between services. Additionally, these representations can facilitate the automation of workflow optimization and orchestration, reducing the need for manual interventions and reducing the overall operational costs of the system. Therefore, representation learning is a promising approach for tackling the challenges of developing and managing large-scale distributed systems.

Furthermore, representation learning can be combined with reinforcement learning to improve the efficacy of decision making processes within a distributed system. By learning representations of the state and action spaces within the system, reinforcement learning agents can more effectively explore and navigate the system in order to achieve desired outcomes.

Two main methods of representation explored in this thesis are graph representation and vector representation. Workflows can be modelled as graphs, with nodes representing services within a distributed system and edges representing data flow. Geometric Deep Learning, which takes graphs as inputs, is a quickly developing area of machine learning that can be combined with Reinforcement Learning in order to develop workflows. Additionally, using machine learning graphs can be turned into vector represent-

ations which are euclidean. These euclidean vector representations can be useful not only for use in construction of workflows but also later analysis of workflows, such as for identifying emergent or unexpected behavior and seeking previously unaccounted for relationships, events, and groups.

We discuss relevant background in section 1.1. This provides key definitions in the areas of distributed systems, reinforcement learning and representation learning, all of which are referenced in the rest of the thesis. We then discuss the motivation and the research aims of the thesis in section 1.2. Finally we go over the thesis structure and contributions in section 1.3.

1.1 Background

To place the remainder of this thesis into its proper context, it is important to establish a common understanding of the scope of the major areas discussed in this thesis. First is a description of distributed systems and workflows, which are the general application and motivation of this thesis. Next is an introduction of basic Reinforcement Learning terminology, and then an introduction to representation learning terminology, both of which are used throughout the rest of the thesis.

1.1.1 Distributed Systems and Workflows

A distributed system can be defined as: 'a collection of autonomous computing elements that can appear to its users as a single coherent system' [121]. Distributed systems are used for a wide variety of applications, including P2P networks, cluster computing, large financial systems, and CCTV networks. As these systems are composed of a significant number of individual components, the order of accessing these separate services, or the workflow, is very important to the efficacy and efficiency of achieving the desired task. Tracking a specific car through a large CCTV network, for

example, would be extremely time consuming to do manually. Automated workflow construction is therefore an important open question in the area of distributed systems.

Consider a system that takes the name of a city as an input, and returns the percentage of residents that suffer from asthma, along with the nitrogen dioxide levels in each borough as an output, as shown in table 1.1. This system would have to consult multiple

Input	Output		
City	Borough	% asthmatic	$\mu g/m^3 NO_2$
London	Tower Hamlets	8.9	40
	Kensington & Chelsea	11.1	58
	Harrow	7.5	38
	...	...	...

Table 1.1: A fictitious example of the inputs and resulting outputs of a distributed system that accesses multiple components.

components, as described in figure 1.1:

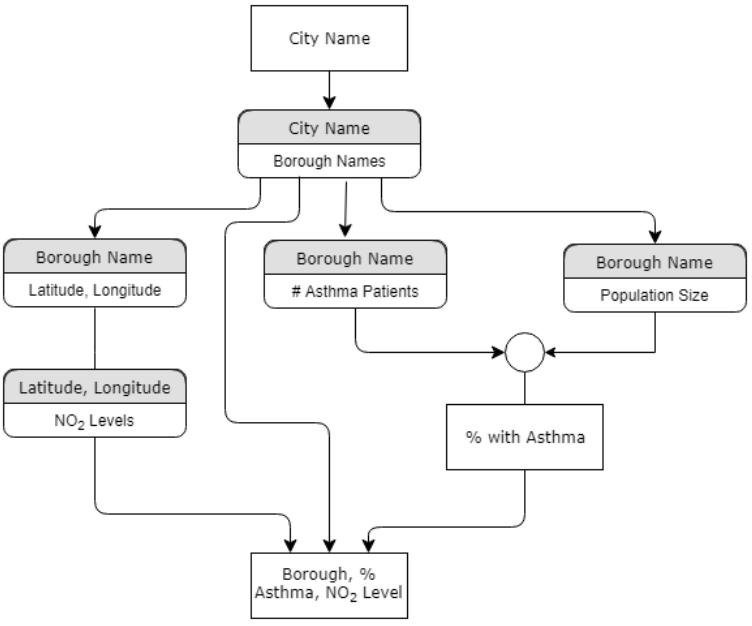


Figure 1.1: diagram of workflow for example distributed system

1. A component that returns the names of boroughs given a city name
2. A service to return the number of asthma sufferers in a borough given a borough name
3. A service to return the total population given a borough name
4. A service that finds the geographical coordinates of the centre of each borough
5. A service that returns the nitrogen dioxide levels given a set of coordinates

Accessing these components in the correct order is vital: using the population component without first breaking down the city into its separate boroughs will return the population for all of London, which would then calculate an incorrect percentage in each borough. Similarly, accessing the number of asthma sufferers with a set of geographical coordinates will return no results if the syntax of that component requires a borough name.

A system of this size can be managed by setting the workflow manually, but what if a system has hundreds of different components? Or the system has multiple different goals? What if the syntax and semantics of certain components are not fully clear? Finding a way to automate the construction of workflows allows for the creation of much more complex systems, and circumvents the difficulty of using components that are not perfectly described.

Careful consideration of the examples given so far shows in fact that there are two basic classes of distributed system.

Distributed Systems Under Common Management

The first case to arise historically is the set of distributed systems under the control of a single management organization. Very large systems are often built using this paradigm as it provides for easy scaling to support large numbers of users or workloads, it enforces a clear task breakdown so that one extremely difficult task becomes many

easy or moderately difficult tasks, and it allows for greater resilience if multiple copies of key components are hosted on different sets of physical infrastructure.

These can still be extremely complex but there is an increased possibility of maintaining a set of interoperability rules and inter system contracts that allows the components to communicate in a standard fashion and allows the managing organization to hand construct a directory of services. In this case management and documentation of the workflow is possible but automated tools speed development and deployment and potentially save costs.

Distributed Systems Using Third Party Services

The second case to arise is where some of the distributed components are developed and managed by a third party. This is the case where an overall solution needs the services of multiple systems controlled by other organizations. There are examples in the military where there can be critical interdependencies between allies sharing key intelligence, surveillance, attack and defense capabilities and where systems need to be fast, quick to react to changing circumstances, and robust in the case of a loss of physical facility. A more accessible and still complex area is that of online shopping. Many small online stores have access to a large number of independently provided data and services which they can easily combine to provide their overall service. As indicated a store may use various services to provide it overall product e.g. product price adjustment could be provided by EcomPricer, foreign exchange conversion could be from JPMorgan, fulfillment could be from Shopify.

1.1.2 Deep Reinforcement Learning

Reinforcement learning is defined by the following:

> A *learning agent* interacts with its *environment* by choosing *actions* that

> affect the *state* of the environment, and the eventual *reward*. The agent seeks to develop a *policy* that maximises reward by mapping states to the optimal actions.

This definition has a few basic terms that need to be accurately described.

The *learning agent* is the problem solver or operator itself, and the *environment* is what this agent interacts with. In a game of chess, the player would be the agent, and the chess board would be the environment. An agent can have complete or incomplete control over its environment. For example, in chess the player can move its own pieces wherever it likes (within the constraints of the game - a bishop can only move diagonally and the king can only move one square at a time), but it cannot move the opponent's pieces, giving the agent incomplete control over the environment.

The *actions* are exactly that, the actions the agent can make at a given timestep. These actions affect the *state* of the environment (positioning of pieces on a chess board). Actions and states can be either discrete or continuous, depending on the environment; this project uses the tabular case, where there is a finite set of discrete states and a finite set of discrete actions. Additionally, as in many reinforcement learning problems, this project works with a special case where the action space is the same throughout the state space - in other words, at every state the same set of actions can be performed.

Actions affect the state of the environment as well as the reward, both immediate and eventual. The *reward* is a scalar value given at each timestep. In a game of chess, this could be 0 for each non-terminal move, 1 if the game results in a win, -1 if a loss and 0 for a stalemate. For a solution that requires reaching a goal state in the smallest amount of timesteps (such as golf), each timestep could provide a reward of -1 until the goal state is reached. A faster solution would then have a higher cumulative reward at the terminal state.

Problems that don't have a terminal state are called continuous cases, and require slightly different treatment than cases with terminal states. In these cases a balance

between valuing long term and short term reward must be found, and often require different mathematical treatment than cases with terminating states. Environments that terminate after a finite number of timesteps are called episodic cases, and are the type of cases that will be examined throughout this dissertation.

A *policy* is a function, either deterministic or stochastic, that maps states to actions in order to maximise cumulative reward. In tabular cases, the policy can often be represented as a matrix or dictionary, where each action within each state is given a probability for its selection. Often this policy is deterministic: the probabilities of all actions are 0 for a given state except for the action that will bring the most reward, which would have a probability of 1 - this is called a $greedy$ policy. Other policies are stochastic: even the actions considered less valuable have a non-zero chance of being chosen. These policies are often used because a learning algorithm needs to explore multiple paths to see if there is a possible reward resulting from an action. This is often termed 'exploration vs exploitation': policies need to balance *exploration*, which helps to determine which actions are best, with *exploitation*, which is taking the actions that return the best possible reward. The most common type of stochastic policy is the ε-greedy function. With a probability of $1 - \varepsilon$, the ε-greedy function acts greedy and chooses the most valuable action. For the remaining ε, all actions are chosen from with equal probability regardless of value.

Different policy types are used in different learning algorithms, often depending on whether the algorithm is an *on-policy* or *off-policy* algorithm. An on-policy algorithm has only one policy that it uses to learn and collect reward. An agent learning chess with an on-policy algorithm might make use of an ε-greedy policy, by selecting the best action most of the time, and with probability ε choosing some random action to learn its value, so these on-policy algorithms can learn and collect reward simultaneously. An off-policy algorithm has two policies: a behaviour policy and a target policy. The behaviour policy is the policy used to generate different actions to learn from, which updates the target policy, which is the policy used to collect reward. An agent using an

off policy algorithm for chess might use an ε-greedy behaviour policy with a large ε for practice games to test new moves, and then use a greedy target policy for competition games in order to increase chances of winning.

Markov Decision Processes

Markov decision processes (MDPs) are the mathematical framework used to describe and standardize all reinforcement learning problems. By defining a problem using this framework, algorithms can be applied that are used throughout the reinforcement learning field.

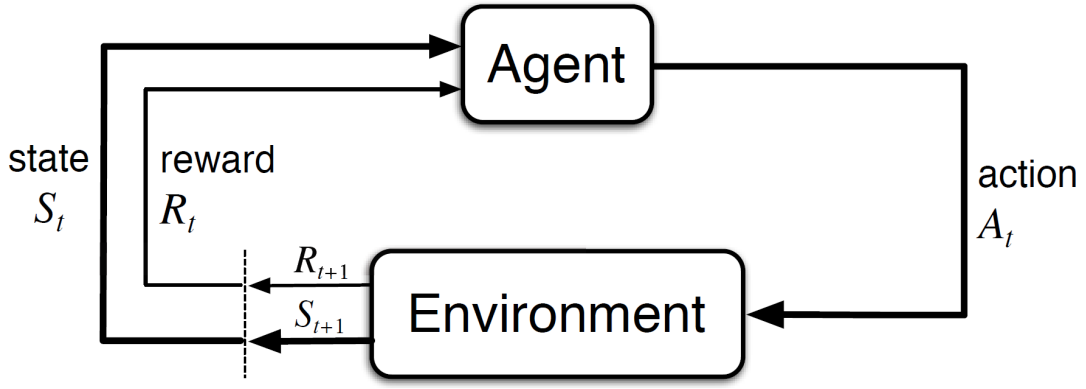


Figure 1.2: graph displaying MDP taken from Sutton book

MDPs frame an RL problem as seen in figure 1.2. At each time step $t = 0,1,2,3...$ the agent receives information about the environment's state $S_t \in S$, and selects an action $A_t \in A$ based on the policy $\pi(A_t \mid S_t)$. At the next timestep, the agent receives a reward $R_{t+1} \in R \subset \mathbb{R}$ and is in a new state S_{t+1}. This trajectory then continues: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2...$ until the episode terminates.

The dynamics of a finite MDP (which has a finite set of states, actions, and rewards) is

defined as a probability:

$$p(s', r \mid s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{1.1}$$

$$\text{where } p : S \times R \times S \times A \to [0, 1]$$

$$\text{and } \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \text{ for all } s \in S, a \in A(s)$$

These probabilities should completely characterise the dynamics of the environment in an ideal MDP. This means that states within the environment should have the *Markov property*, which means the state must include all of the information needed to determine the next state and reward given the action selected - the history of all previous states can be thrown away and all important information must remain in the current state. An example of this is a toy helicopter. The state must include the current position and velocity of the helicopter in order to correctly determine the position of the helicopter in the next state. If the state included only position, it would either have to look to the previous state to calculate velocity, and therefore not be Markov (it cannot use history), or will not have enough information to determine the position of the next timestep. While useful from a theoretical standpoint to prove convergence of learning algorithms, in application of reinforcement learning, the Markov property can make solutions too computationally expensive, as will be discussed later on.

For episodic cases, the return G_t is the sum of all rewards that have been given during the episode:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{1.2}$$

In many cases, immediate reward is considered more valuable than potential reward later on, so we can introduce a discount factor $\gamma \in (0, 1]$:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.3}$$

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{1.4}$$

A unified notation for returns for both episodic and continuing cases is:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{1.5}$$

The expected return starting from a state s and following a given policy π is found using the *value function*:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{for all } s \in S \tag{1.6}$$

and similarly, the expected return starting from a state s, choosing an action a and then following policy π is found using the *action-value function*:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{1.7}$$

1.1.3 Representation Learning

In the recent years, there has been an exponential increase in data that is being generated from various sources such as text, images, graphs, and videos. The availability of such massive data has led to a necessity for finding accurate representations of data to perform analysis, classification, and other tasks. Embeddings are the mathematical representations that help in summarizing and capturing the essential characteristics of data.

Representation Learning can be defined as the process of representing high-dimensional data in a lower dimensional space while preserving the essential characteristics of the original data. In other words, it is a mathematical technique that transforms complex data into a simpler form, normally referred to as an embedding, allowing it to be analyzed and understood more effectively. Embeddings are usually created using machine learning algorithms that find lower-dimensional coordinates for high-dimensional data.

There are different types of embeddings that are used for various purposes. some of these include:

1. Text Embedding: Text embedding techniques are used to represent words or phrases as a vector of numbers in a lower-dimensional space. Techniques such as word2vec, GloVe, and Fasttext are commonly used for text embedding [33][89][52].

2. Image Embedding: Image embedding techniques are used to represent images as a vector of numbers in a lower-dimensional space. Techniques such as convolutional neural networks (CNN) and deep auto-encoders are commonly used for image embedding.

3. Graph Embedding: Graph embedding techniques are used to represent graphs as a vector of numbers in a lower-dimensional space. Techniques such as Graph Convolutional Networks (GCN) and DeepWalk are commonly used for graph embedding [57][90].

Embedding techniques are widely used in various domains such as natural language processing, image recognition, recommendation engines, and social network analysis. These embeddings have a wide array of use cases. Text embedding techniques are used for sentiment analysis to understand the emotional state of a large number of people by analyzing social media posts, reviews, and other text data. Image embedding techniques are used in image recognition applications such as facial recognition, object detection, and image search. Embedding techniques are also used in recommendation engines to suggest products, movies, or music based on user preferences.

There are mutliple benefits of embeddings. Embedding reduces the dimensionality of the data, which simplifies the analysis and reduces the computational complexity. These embeddings can represent the essential characteristics of the data that help to improve the accuracy of machine learning models. Embedding also reduces the computational overhead of high-dimensional data, which leads to faster processing and analysis.

Embedding is a critical technique used for representing high-dimensional data in a lower-dimensional space. Text, image, and graph embeddings are commonly used in different domains for tasks such as sentiment analysis, image recognition, and recommendation engines. The benefits of embeddings are reduced dimensionality, improved accuracy, and faster processing.

1.2 Motivation and Research Questions

We hypothesise that *the use of reinforcement learning in combination with other deep learning techniques can be used to construct and represent workflows. These workflows can be used for applications within large scale distributed systems, and their representations can be analysed to obtain further useful information.* This hypothesis can be summarised into the following research question:

How can reinforcement learning be used to construct workflows and represent them in an efficient manner?

This research question can be further broken down into the following questions:

RQ1 What are the best representations for encoding workflows for later analysis of distributed systems and to use as a platform for automatically constructing further workflows?

The research question of what are the best representations for encoding workflows is essential for achieving the broader hypothesis of utilising reinforcement learning and other deep learning techniques to construct workflows for distributed systems. The challenge lies in investigating and identifying the optimal methods of representing these workflows accurately, efficiently and in a way that is amenable to automation. A critical aspect of this research question is determining the level of granularity required to capture the necessary information and the trade-offs between complexity and use-

fulness in analysing these workflows later on. The objective of this research question is to identify and propose new encoding schemes that leverage the inherent properties of the distributed system and the workflow, combining it with deep learning techniques, thereby enabling efficient and automated analysis of these representations.

RQ2 How can reinforcement learning be used to produce vector representations of workflows?

The primary objective of this research question is to explore the use of reinforcement learning techniques for constructing and representing workflows in vector form. Given that workflows in large scale distributed systems can be complex and involve numerous steps, it is imperative to represent them in a compact and efficient manner. By utilizing reinforcement learning algorithms, it is possible to encapsulate the sequential nature of workflows and learn to navigate the space of possible actions, adapting as necessary based on the current state and desired outcomes. The resulting vector representation can then be used for further analysis and optimization, providing valuable insights into the inner workings of complex distributed systems. Ultimately, this research question aims to establish the feasibility and effectiveness of using reinforcement learning as a tool for producing vector representations of workflows in large scale distributed systems.

RQ3 When represented as vectors, how can reinforcement learning be used to construct workflows?

Reinforcement learning algorithms rely on the creation of a state space that can be represented as vectors in order to make decisions based on a reward system. In the case of constructing workflows, the potential exists to create a reinforcement learning state space that incorporates the various steps and decision points within a workflow. By representing each step and decision point as a vector within the state space, the reinforcement learning algorithm can learn to identify the most efficient series of actions

to achieve a desired outcome. The use of deep learning techniques in conjunction with reinforcement learning can enable the system to identify patterns and make predictions about future steps within a workflow, further increasing the efficiency and effectiveness of the system.

RQ4 When treated as graphs, how can reinforcement learning be used to construct workflows?

With this research question, we aim to investigate how reinforcement learning, when employed in conjunction with geometric deep learning techniques, can aid in the construction of workflows as graphs. Graphs are a natural representation method for workflows within distributed systems, so by utilizing geometric deep learning techniques, we can utilise reinforcement learning to generate workflows that can be used in large scale distributed systems.

RQ5 What methods can be used to increase learning efficiency with less labelled data?

The proposed research question aims to explore the methods that can be used to increase the efficiency of learning with less labelled data. Many areas of machine learning are highly dependent on labelled data, which can impose significant challenges in real-world large-scale distributed systems, where well labelled workflows created by human experts are minimal. Therefore, it is crucial to investigate efficient methods that could potentially reduce the amount of labelled data required for training these reinforcement learning algorithms.

1.3 Thesis Structure and Contributions

The remaining chapters, and the contributions therein, are as follows:

Chapter Two - Literature Review: This chapter introduces the research topic and provides a background for the study of geometric deep learning, specifically in the area of graph generation. The literature review explores the current academic work in the areas of graph convolutional networks, reinforcement learning for graphs, and reinforcement learning for complex environments, highlighting the gaps in the existing literature.

Chapter Three - Representation Learning for Knowledge Graphs: This chapter explores the combination of knowledge graphs (KGs) with semantic vector spaces (SVSs) via knowledge graph embedding (KGE) and reports on the state-of-the-art in KGE. It describes the operational benefits that can be gained from this approach and the considerations for observational ontologies that describe complex and rapidly-evolving environments.

C1 This examines the benefits of various graph embedding techniques that can be used to represent workflows. These embedding techniques can be used with semantic vector spaces to provide better analysis of workflows in distributed systems. This answers **RQ1**.

Chapter Four - Compositional Plan Vectors for Multitask Learning: The third chapter proposes a novel algorithm for deep reinforcement learning called CPV-TER. This method uses compositional plan vectors (CPVs) to efficiently learn multiple tasks simultaneously, by representing the subtasks as vectors and sequences of subtasks as the sum of those vectors. The chapter demonstrates that the approach allows for more efficient learning and outperforms other standard multi-task RL algorithms.

C2 This chapter demonstrates a new method to construct multi-step tasks with reinforcement learning. This method produces hierarchical solutions to tasks without expert demonstrations. This answers **RQ3**.

C3 This method also produces embeddings which can be used to represent workflows in a vector representation. These vectors can then be used to perform analysis of the distributed system that contains the workflows. this answers **RQ2**.

C4 This chapter also provides a new replay method that allows for more efficient learning with data in reinforcement learning. By using self produced data as expert demonstration imitation learning techniques can be used on top of reinforcement learning techniques. This answers **RQ5**.

C5 In this chapter we also released a benchmark that involves tasks of a sequential nature and additionally requires minimal compute power in comparison to other benchmark environments in the MTRL space. This is a new focus area of green AI and aims to make the academic field both greener and more inclusive for academics with minimal access to additional compute power [109]. This is not a direct result of one of the main research questions but is nonetheless an important contribution to the reinforcement learning space.

Chapter Five - Deep Geometric Learning for Directed Acyclic Graphs: This chapter presents a novel method for generating directed acyclic graphs (DAGs) using deep reinforcement learning. DAGs with specified structures and highly sparse reward environments are challenging to generate. This method demonstrates generating DAGs with node types and topology satisfying criteria.

C6 This chapter demonstrates a new method to construct directed acyclic graphs using geometric deep learning combined with reinforcement learning. This demonstrates an ability to produce workflows with minimal data outside of workflow application success. This answers **RQ4**.

Chapter Six - Conclusion: In this final conclusion chapter we provide a final review of the research questions and how we answered them, and how these answers contributed to the fields of reinforcement learning and representation learning. We delineate potential areas for future work stemming from each of the chapters, and we make our final remarks.

Chapter 2

Literature Review

2.1 Overview

Applications that run on large distributed systems can require access to multiple microservices creating complex workflows. With large scale systems, these workflows can become too large to efficiently design by hand, particularly when the system is dynamic and prone to failure and reorganization. Designing workflows on large distributed systems is a highly complex task. Some automation or approximation is needed to develop policies for previously unseen tasks or redesign policies when the distributed system's structure has changed significantly.

Machine learning can be applied to handle the complexities inherent in the data by treating workflows and schedules on distributed systems as structured graphs. By using a combination of machine learning and reinforcement learning, new workflows can be constructed even on large systems that are dynamic, interpretable, and require minimal training data.

In this literature review, we take into consideration current academic work, particularly in the areas of graph convolutional networks, reinforcement learning for graphs, and multitask reinforcement learning. In doing so, we highlight the existing gaps in the literature. Considering the aims of the thesis, this literature review will proceed through five major topics: graph neural networks, deep reinforcement learning for graphs and schedules, reinforcement learning methods for large and complex environments, multi-

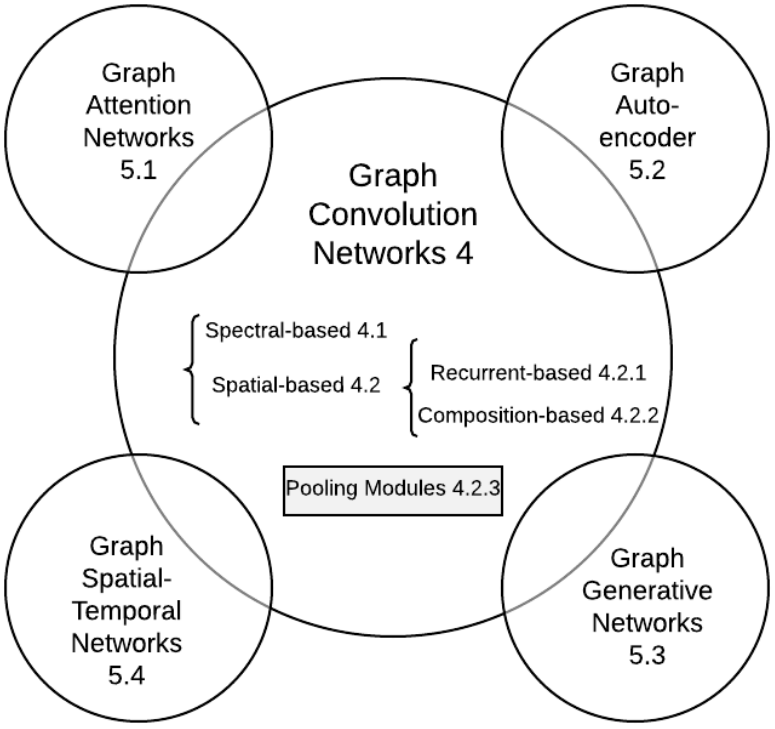


Figure 2.1: Figure reproduced from [129] showing the major areas of graph based neural networks.

task reinforcement learning, and replay methods for reinforcement learning.

2.2 Graph Neural Networks

Machine Learning, typically known for working with images, text and audio, has recently expanded into working with graph based structures. Geometric deep learning, a field that has expanded rapidly within the last decade, has a wide range of applications, including drug discovery, link prediction for social networks, and applications for task scheduling. Geometric deep learning differs significantly from typical machine learning methods due to the non-euclidean nature of graphs [140].

There have been a number of comprehensive literature reviews in this area. Notably, [138] discusses a wide range of techniques in the field, including supervised techniques

such as Graph Recurrent Neural Networks (GRNNs), unsupervised methods such as Graph Auto Encoders (GAEs), and even briefly touching on graph-based reinforcement learning, a small but growing area of the geometric deep learning field. They also lay out the major open issues in the field, as well as suggest some potential future directions. [140] focuses more specifically on supervised learning methods, and categorises methods by type of training data, propagation step, and even by graph type. They also discuss frameworks used in the field, including MPNN, an important framework described in section 2.2.2. Finally, [129] provides a valuable taxonomy to categorise graph neural network methods, as well as discussing important benchmarks and mentioning valuable open source data within the community. Figure 2.1, reproduced from [129], groups the major components of graph neural networks into 5 distinct, yet overlapping areas. In this section of the literature survey we focus on graph convolutional networks, with a particular attention paid to spatial graph convolutional networks.

2.2.1 Graph Convolutional Networks

Graph convolutional networks (GCNs) are the basis for the majority of graph neural networks. They are similar to standard convolutional neural networks (CNNs) in that they convolve over localities: for images this would be a group of neighbouring pixels [59], for natural language processing this could be nearby words or individual letters [56]. For graphs, standard CNNs cannot be used, as these methods rely on the strict euclidean nature of image and sentence structure, which graphs do not have. GCNs use same underlying principle of convolving over localities, but with different methods to achieve this. There are two major types of GCNs: spectral and spatial.

Spectral GCNs

The first GCNs developed were spectral, as proposed initially by [13]. Here the term spectral refers to spectral graph theory, which works primarily with graph laplacians,

$\mathcal{L} = I - D^{-1/2}AD^{-1/2}$ (where I is the identity matrix, D is the degree matrix and A is the adjacency matrix), which hold most useful properties of graphs in a matrix. By performing an eigendecompostition on the laplacian, we can then convolve over these eigenvectors. This method has a strong theoretical basis, however the time complexity of finding the eigenvectors of the laplacians of graphs is very high ($O(N^2)$), and is generally non-scalable [138]. Other spectral methods have lowered this complexity by using chebyshev polynomials, which in effect are truncations of the laplacian eigenvectors, reducing the time complexity significantly [19]. Despite these reductions in time complexity, they still require high amounts of memory, and generally do not generalise to graphs with different structures [129]. For this reason we focus mainly on spatial GCNs.

Spatial GCNs

Spatial GCNs work very similarly to standard CNNs by combining information at each point from neighbouring areas. For spatial techniques this generally involves using the graph's adjacency matrix and aggregating information from each node's nearest neighbours. Depending on how the information is aggregated, how many neighbours are sampled, and whether or not pooling functions are used, spatial GCNs can encode a wide variety of features, and perform a number of different tasks at different scales. In this survey we focus mainly on spatial methods.

2.2.2 MPNN Framework for Spatial GCNs

[32] reformulated multiple important existing spatial models into a single cohesive framework, allowing for easier descriptions of variations of the basic model. This framework is called the Message Passing Neural Network (MPNN), and works by splitting each method into two parts: a 'message passing' phase and a 'readout' phase.

During each timestep t of the message passing phase, the hidden states h_v^t at each node

v are updated based on an update function U which acts on the current state of the node, and the states of the neighbouring nodes (thus a message is passed from node to node). Formally this is written as:

$$h_v^{t+1} = U_t(h_v^t, \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})) \tag{2.1}$$

Here the message passing function takes as input the state of the current node, the state of neighbouring nodes, and the state of connecting edges e_{vw}.

The readout phase takes these hidden states and computes a final feature vector to describe the whole graph, formally written as:

$$\hat{y} = R(\{h_v^T | v \in G\}) \tag{2.2}$$

and is effectively a pooling function, often described separately from GCN methods.

By using this message passing framework, we can describe all spatial GCNs with a message passing function and an update function, and can describe pooling methods as readout functions.

2.2.3 Major GCN Variants

There is a large number of spatial GCN variants, which take advantage of different types of features and produce varying outputs. One of the earlier variants, node2vec [34] outputs a feature representation for each individual node as opposed to the entire graph. The main message passing function is in effect a concatenation, which is a standard message passing function. The key feature of node2vec is that its message passing function only takes information from a subset of a nodes neighbours via a random walk method. This increases efficiency significantly, which is very important for larger graph structures. This random walk is designed to act somewhere in between a depth first search method and a breadth first search method. This allows for a flexible representation that can account both for variations in substructures and the graph as a whole.

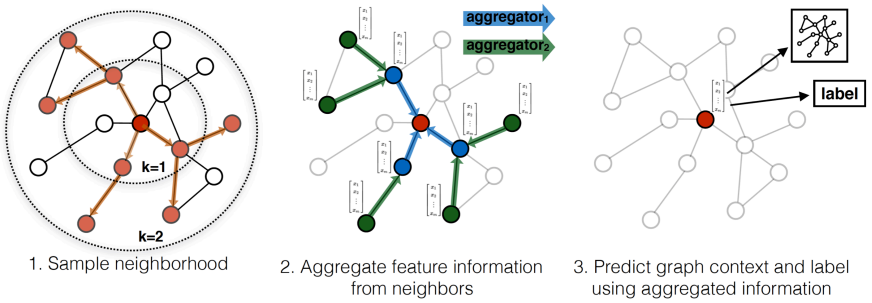


Figure 2.2: Figure reproduced from [37] showing the underlying principle of the GraphSAGE algorithm.

GraphSAGE [37] provides a feature representation of the whole graph, and uses a message passing function that involves both aggregation and concatenation. It first aggregates the current node and all neighbouring nodes, generally by performing an element-wise mean, and then concatenates this aggregation with the current node state. This concatenation is then manipulated with a weight function, which is learned over training.

struc2vec [97] aims to produce node representations that emphasis inherent structural similarities in nodes, as opposed to location in the node, via a skipgram-style model. For example, two nodes that have the same degree and have neighbours with the same degrees should produce similar feature vectors, even if they aren't near each other locally.

Most of these methods work primarily with undirected edges with no edge weightings. Recently struc2vec++ [112] was proposed to handle directed and weighted edges, which is particularly important for workflow applications. In a message passing formulation, the message passing function would be similar to that of struc2vec, however the neighbouring nodes would be split into two different neighbourhoods: nodes with incoming edges from the current node, and nodes with outgoing edges to the current node.

Recently, Facebook published a method to embed vectors for massive scale graphs

[63]. This is the first embedding method for graphs to scale to billions of nodes and edges. This works by performing a block decomposition of the adjacency matrix (used to determine neighbouring nodes) as well as by distributing computation and by performing negative sampling, all of which are techniques which can be implemented alongside other GCN methods to bring them to scale.

As in the NLP field where self-attention mechanisms, notably Transformer models [122], have shown significant success, recent approaches have introduced self-attention mechanisms into GNNs. Graph Transformer Networks (GTN) [135] is a model inspired by the work on non-local neural networks. The authors propose a novel localized self-attention mechanism to capitalize on the locality principle, a common underlying property of natural graphs. This type of self-attention operation enhances the model's capability to capture long-range dependencies in graph-structured data, which can be critical for many downstream tasks.

Finally, [93] published a method to bring interpretability to graph embeddings, which is particularly important for applications to distributed workflows, particularly when the microservices have a set workload or are of a sensitive nature, as is often the case in military contexts.

2.3 Reinforcement Learning on Graphs

Supervised learning methods generally work best in instances where there is a large amount of training data available. For generation tasks, this is often not the case. Supervised learning is often used for generation when the aim is to generate examples that have similar properties to previously seen examples, in which case GANs can be used, as in [66]. For workflow construction however, where the task may never have been performed before, this is clearly not the case.

In these instances, reinforcement learning can be used, as it does not require labelled examples, but instead only requires a numerical reward depending on whether or not

the policy produces desired results. While reinforcement learning on the whole is a very large field, reinforcement learning on graphs is a particularly small and new area. This is again largely due to the complexity and scalability issues of graphs, as reinforcement learning typically struggles with the so-called 'curse of dimensionality' [115].

Reinforcement learning has already been applied for scheduling problems, namely in [87]. This paper uses a tabular Q-learning technique. By *tabular* we mean that each possible state (in this case, a graph) is calculated separately, and a policy for each action is calculated individually as if in a table. In a game of chess for example this would be akin to learning a set of actions for every possible state of play, which is unfeasible for even a relatively small graph. A supervised or unsupervised learning method to produce vectors to represent states is almost always used in current reinforcement learning techniques. This allows for states to be approximated, and for policies to be used even if a specific state hasn't yet been explored, but a similar state has. Therefore by applying a GCN to a workflow graph it should decrease the exploration levels needed significantly.

Reinforcement Learning has been combined with graph embeddings, specifically for molecule generation as in [133]. In this paper, the learning agent added a single atom or bond in each timestep, and the reward was based on a GAN determining if the developed molecule was similar to a base set of known molecules. The states provided to the learning agents were a graph embedding produced by a GCN. This method provided a significant increase on chemical property optimization than previous baselines.

Similar methods have also been implemented for combinatorial optimisation using graphs, a potential use case to approximate workflows [16]. This paper uses structure2vec for its embedding, and trains using Q-learning. Notably this method can be implemented on graphs of up to 1200 nodes, which while still relatively small, is much larger than standard molecular structures, which is the major focus for current work

in graph based reinforcement learning. Another recent paper compares multiple graph embedding schemes for reinforcement learning, including node2vec, GraphWave and variational graph auto-encoders [73].

2.4 Reinforcement Learning for Complex Environments

As graphs are so complex, particularly at scales as needed for workflows, additional optimization methods need to be implemented. In order to fully implement graph based deep reinforcement learning for constructing workflows in distributed systems, a few different goals need to be achieved.

First, an agent learning to construct workflows in large distributed systems will need to be able to choose to send information from a wide variety of microservices to another. This means the agent should be able to learn policies for environments with large action spaces, and potentially even continuous action spaces. Typical reinforcement learning generally only deals with relatively small action spaces, such as cardinal directions in a video game, but for graph construction this action space could potentially scale linearly with the size of the graph in the worst case. [22] provides a concise overview of the state of the art methods for continuous control in reinforcement learning, and concludes that methods such as TRPO and TNPG are optimal algorithms for large action spaces.

Another issue with large graphs is that the region that produces a positive reward becomes a significantly smaller part of the total state space as graphs scale. In order to handle this, intelligent exploration and use of that exploration needs to be implemented. Rainbow DQN [42] implements all state of the art additions to Deep Q networks, a very popular deep reinforcement method. The paper also performs an ablation study to determine which variations are the most important in improving the end result. In this paper they determine that both prioritized experience replay and multi step reward calculations most improve the result. prioritized experience replay involves replaying actions that have already been taken and stored in a memory. The memories chosen to

replay are prioritized if they produced an unusual reward, either high or low. This allows for a single action to affect the overall policy significantly if it produces a positive result, even if the probability of reaching that state by chance are low. Multi-step rewards bootstrap rewards from the greedy actions from future states as opposed to using the single state reward. This increases values of states that eventually produce rewards significantly, particularly in environments where a reward may not be seen until multiple steps have been achieved, as in a workflow. Another method, Ape-X DQN [45], implements prioritized replay in a distributed manner, resulting in even faster learning in a significantly faster wall-clock time.

Another important issue in constructing workflows in large distributed systems is a method to distribute learning over multiple learning agents. As these systems are by definition distributed, it is therefore inherent that a distributed solution for constructing workflows is desired. [23] first introduced the concept of multi-agent reinforcement learning. [62] provides a concise overview of how current reinforcement learning methods can be implemented in a multi-agent fashion. [51] utilizes GCNs to implement multi-agent learning specifically, and finds that representing agents as nodes in a graph to plan out cooperation significantly assists in learning speeds. Learning with Learning Opponent Awareness [31] accounts for interacting in an environment that has adversarial learning agents - this could be an area to consider when designing workflows on systems that struggle with overloading, particularly when the learning agent does not have ownership over the scheduling decisions. [30] focuses on communication difficulties, and information sharing protocols when handling multi agent learning on distributed environments, a particular issue when constructing workflows for systems on the edge, or when dealing with systems that handle sensitive information.

Finally, hierarchical reinforcement learning can be implemented to learn sub tasks in workflows, and increase learning in highly complex environments. Certain pooling operations, such as DiffPool[132], account for substructures in graphs, allowing for vector embeddings that retain information on these important substructures. Then,

multilevel hierarchies can be implemented from the reinforcement learning end to learn particular subtasks that make up a final workflow [65]. This has already been implemented as a graph based heirarchical reinforcement learning method as in [1]. Hierarchical reinforcment learning is also used in multi-task reinforcement learning, and is discussed further in section 2.5.1.

2.5 Multi-Task Reinforcement Learning

Workflows by definition complete multiple tasks, making multi-task reinforcement learning a logical area of focus for workflow construction. Multi-task reinforcement learning (MTRL) has emerged as a promising approach to enable agents to learn multiple tasks simultaneously, thereby improving their overall performance and efficiency. MTRL frameworks and algorithms have been developed to address the challenges of learning multiple tasks, such as task interference and the need for efficient exploration. In this literature review, we focus on four key sub-topics of MTRL: methods for MTRL, application areas, and benchmarks used for performance assessment. We provide an overview of the state-of-the-art papers in each sub-topic, highlighting their strengths and limitations. This section of the literature review aims to provide a comprehensive understanding of the current landscape of MTRL.

2.5.1 Algorithms and Techniques for MTRL

There are several algorithms and methods used for multi-task reinforcement learning, which can be categorized into different types based on their approach to handling multiple tasks simultaneously: Meta-Learning, Joint Training Approaches, Hierarchical RL, and Transfer Learning. Some of these methods overlap with each other and can be used in conjunction. Overall, these different types of algorithms and methods offer different advantages and disadvantages for multi-task reinforcement learning. Joint

training is useful when the tasks have a lot of common features, while meta-learning is useful when the agent needs to quickly adapt to new tasks. Hierarchical approaches are useful when the tasks have a natural hierarchy, and curriculum learning is useful for speeding up the learning process by exploiting similarities between tasks.

In the following subsections, we give a cursory overview of these areas while outlining key papers from the area. While not all of these techniques are utilised in the thesis it is nonetheless important to demonstrate the different approaches within the MTRL space.

Meta-Learning

Meta-Learning is at times considered to be a separate domain from MTRL, however it has considerable overlap so still merits examination here. Meta-learning and multi-task reinforcement learning are related, as both involve learning across multiple tasks. However, the key difference is that meta-learning focuses on learning how to learn across tasks, while multi-task reinforcement learning focuses on learning how to perform multiple tasks simultaneously.

Multi-task reinforcement learning involves learning to perform multiple tasks at the same time, with the goal of improving performance on all tasks. It can be thought of as a way to transfer knowledge and skills from one task to another, with the idea that performance on each task can help improve performance on the others. Meta-learning, on the other hand, focuses on learning how to learn across tasks with the goal of improving overall learning and adaptation speed. By learning how to learn quickly and efficiently, meta-learning can help improve performance on a wide range of tasks, even those that have not been explicitly encountered before.

the most popular method in this space is Model-Agnostic Meta-Learning (MAML) [28]. The agent performs negative adaptation by explicitly trying to forget the information it previously learned on a given task, which helps it to adapt to new tasks more

effectively. The Reptile agent is an adaptation of MAML that proposes a first-order algorithm for meta-learning that is model-agnostic [84]. The agent uses gradient-based optimization to learn a better initialization for the learning process, which can be applied to different tasks.

The POET agent can generate complex and diverse learning environments and their solutions by using a self-created curriculum approach [124]. The agent uses a co-evolutionary algorithm to generate increasingly complex environments while simultaneously improving the agent's ability to solve those environments. PEARL proposes an agent that performs efficient off-policy meta-reinforcement learning using probabilistic context variables [95]. This approach allows the agent to learn from different policies and apply that knowledge to new tasks, improving the performance of the agent on new tasks. The DREAM agent can efficiently decouple exploration and exploitation for meta-reinforcement learning without impacting the agent's performance [71]. The agent uses a two-policy approach that helps the agent to balance exploration and exploitation while still learning effectively.

Meta-Learning is often a key consideration for performance benchmarks for MTRL, and the environments are created to test both MTRL and Meta-Learning solutions. These benchmarks are considered further in subsection 2.5.3.

Joint Training

Joint training approaches for multi-task reinforcement learning refer to the process of simultaneously learning multiple tasks using a single reinforcement learning model. This approach involves training the model to optimize multiple objectives and achieve optimal performance across multiple tasks. The goal of joint training is to improve the efficiency and scalability of the learning process by sharing information and leveraging the similarities and differences across the learned tasks.

One common approach for joint training in multi-task reinforcement learning is multi-

objective reinforcement learning, which involves optimizing multiple objectives or goals simultaneously. This uses a set of reward functions that correspond to different tasks, and the model learns to optimize the overall reward by balancing the different task objectives. This approach can be useful when there are tasks that have conflicting objectives, and the model needs to optimize across all of them.

Overall, joint training approaches for multi-task reinforcement learning can enable agents to learn efficiently and effectively across multiple tasks by sharing information and leveraging knowledge from related tasks. These approaches can be particularly useful in real-world applications, where agents need to learn multiple tasks simultaneously in a complex and dynamic environment. Joint training approaches typically use adapted versions of reinforcement learning agents used for single task RL, such as Rainbow DQN or APE-X (discussed previously in section 2.4. PPO, TRPO, and SAC are common MTRL choices and are often used in benchmark demonstrations.

Proximal Policy Optimization (PPO) is a model-free, on-policy algorithm that updates the policy parameters using a trust region optimization step [108]. It uses a clipped surrogate objective function to prevent the policy from changing too much between updates, ensuring stability and reducing the likelihood of catastrophic forgetting. PPO is especially effective for continuous control problems and has been shown to achieve state-of-the-art performance on a variety of benchmark tasks. Trust Region Policy Optimization (TRPO) is a model-free, on-policy algorithm that updates the policy parameters using a trust region optimization step similar to that used in PPO [107]. However, it uses a natural policy gradient to update the parameter vector, which takes into account the geometry of the policy parameter space to ensure that updates are well-behaved. TRPO has been shown to be effective on a variety of tasks, especially those with high-dimensional state spaces.

Soft Actor-Critic (SAC) is an off-policy, model-free algorithm that uses a maximum entropy objective function to encourage exploration and robustness to stochastic environments [36]. It uses a differentiable policy critic network to estimate the value of

the current policy and regularizes the policy update with an entropy term to prevent premature convergence. SAC has been shown to achieve state-of-the-art performance on a variety of continuous control tasks and has been used for multi-task learning in combination with hierarchical reinforcement learning.

Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) is a subfield of Reinforcement Learning that is concerned with learning and decision-making in complex environments with multiple levels of abstraction. In HRL, the learning agent decomposes a complex task into a series of smaller tasks, which can be solved independently, and then integrates them to achieve the overall goal of the task. HRL is motivated by the idea that complex tasks can be more efficiently learned if they are broken down into smaller, simpler tasks that can be learned independently.

The main advantage of HRL is that it can significantly reduce the learning time and computational resources required to learn complex tasks. However, HRL also poses some challenges, such as how to design the hierarchy, how to handle the imperfect decomposition of tasks, and how to integrate the learning of different levels.

"Learning Multi-Level Hierarchies with Hindsight", proposes a hierarchical RL agent that uses the hindsight experience replay (HER) technique to improve sample efficiency [64]. The agent consists of multiple levels of controllers, each responsible for a different timescale of decision-making. At the lowest level, the agent takes individual actions, while at higher levels, it takes longer-term decisions based on the outputs of the lower-level controllers. HER is used to augment the agent's experience with successful trajectories that may not have been optimal for the original task.

The H-DQN agent integrates both temporal abstraction (i.e., the ability to learn actions that span multiple time steps) and intrinsic motivation (i.e., the motivation to explore) in a hierarchical manner [60]. The agent consists of multiple levels of controllers that

use deep neural networks to learn representations of the state and action spaces. The higher-level controllers use these representations to guide the lower-level controllers in their decision-making.

The third paper, "Data-Efficient Hierarchical Reinforcement Learning", proposes a hierarchical RL agent that uses a tree-structured policy to efficiently learn multiple tasks [79]. The agent consists of multiple levels of controllers that operate at different levels of abstraction. The lower-level controllers use tree-structured policies to learn local behaviors, while the higher-level controllers use them to learn global behaviors. The agent also uses a novel method for transfer learning, where learning from one task can be used to speed up learning on another task.

"FeUdal Networks for Hierarchical Reinforcement Learning", proposes an agent that uses a feudal network architecture to learn hierarchical policies [123]. The agent consists of two networks, a manager network that learns high-level policies and a worker network that learns low-level policies. The worker network receives goals from the manager network and generates actions to achieve those goals. The manager network receives rewards based on the worker's actions and updates its policies accordingly.

"The Option-Critic Architecture" proposes an agent that uses options to learn hierarchical policies [3]. Options are defined as temporally extended actions that are composed of lower-level actions. The agent consists of two networks, an option network that learns high-level policies and a critic network that learns to evaluate the options. The option network receives rewards based on the critic's evaluations and updates its policies accordingly.

Transfer Learning

Transfer learning involves using knowledge gained from learning one task to accelerate learning on another related task. This can be done by reusing parts of a previously trained agent, such as the hidden layers of a neural network, to initialize the learning of

a new agent. This approach aims to leverage previously acquired knowledge to reduce the sample complexity and improve the convergence speed of the learning process. Some of these approaches can be adapted into a joint training approach, but tend to work best when treated in a transfer learning manner.

The key feature of DISTRAL is its ability to leverage the shared structure that exists across multiple tasks [119]. This agent is able to identify common patterns and structures that are shared between different tasks, and it can use this information to learn more efficiently. DISTRAL is able to do this through a process called distillation, which involves compressing the knowledge gained from training on multiple tasks. DISTRAL consists of two main components: a backbone network and a task-specific head network. The backbone network is responsible for processing the input data and extracting important features that are relevant to all the tasks. The task-specific head network is responsible for processing the output of the backbone network and generating task-specific outputs. In addition to its ability to learn multiple tasks simultaneously, DISTRAL also has the ability to transfer knowledge between tasks. This means that it can learn a new task more quickly by leveraging the knowledge that it has gained from training on other tasks.

The IMPALA (Importance-weighted Actor-Learner Architecture) agent is a type of multi-task reinforcement learning agent that breaks down the reinforcement learning problem into two separate components: learning a policy and learning a value function. This approach allows the agent to perform multiple tasks simultaneously while also scaling well with large state and action spaces [24]. The agent consists of multiple actors and one central learner. Each actor interacts independently with the environment, generating episodes of experience. These episodes are then sent to the central learner, which updates the policy and value function parameters based on the collected data.

The key innovation of the IMPALA agent is the use of importance weights. Importance weights adjust the contribution of each actor's experience in the learning process based

on the performance of the policy at the time the experience was generated. This allows the agent to use experience from all tasks to improve the policy and value function, even if some tasks are experiencing slower progress or have lower rewards. Additionally, the IMPALA agent uses a prioritized experience replay buffer, which stores and prioritizes the most useful experiences for learning. This helps to improve learning efficiency and generalize experience across different tasks.

The PopART agent uses a population of neural networks, which each specialize in a different task [43]. The networks are trained separately but are allowed to share information with each other through the use of a shared memory. This allows them to learn from each other and to transfer knowledge between tasks, which can greatly improve learning efficiency.

Additionally, the PopART agent uses reward shaping to guide the learning process. This involves modifying the reward signal given to the agent to incentivize desirable behavior. The reward shaping function is also learned by the agent, which allows it to adapt to changing environments and tasks.

The agent proposed in the ICLR 2018 Paper "Learning an Embedding Space for Transferable Robot Skills" is a multi-task reinforcement learning framework that learns a shared embedding space for transferring skills across different tasks [40]. The model comprises of three major components: a policy network, a value network, and an embedding network. The policy network produces a distribution over actions given the current state of the environment, while the value network predicts the expected cumulative reward obtained from the current state. These two networks are trained using the standard deep reinforcement learning algorithms like actor-critic, which maximizes the expected reward of a set of tasks simultaneously.

The embedding network maps the state-action pairs to a lower-dimensional embedding space, which captures the underlying structure of the tasks and their similarities. This embedding space facilitates transfer between different tasks as it aligns their representations. The embedding network is trained by minimizing a contrastive loss, which

aims to bring together embedding vectors of states that belong to the same task while pushing apart those that belong to different tasks. Overall, the proposed agent seeks to learn a shared embedding space that captures the common structure of multiple tasks, allowing for transfer of skills and knowledge between them. By doing so, it enables more efficient and effective multi-task reinforcement learning, reducing the amount of task-specific training required.

Curriculum Learning

Curriculum Learning is a technique used in machine learning and reinforcement learning that involves gradually increasing the complexity of learning tasks presented to a model, with the aim of facilitating faster training and improved performance on the target task. In Curriculum Learning, simpler tasks are presented initially to the model, which then gradually progress to more complex tasks.

Curriculum Learning is particularly relevant in the context of multi-task reinforcement learning, where the objective is to learn multiple tasks simultaneously. In multi-task reinforcement learning, a model is trained to learn multiple tasks at the same time, which can be more challenging than learning a single task in isolation. Curriculum Learning can be used to structure the learning process [92], making it easier for the model to learn multiple tasks simultaneously by presenting simpler tasks first, before moving on to more complex tasks later. This can also be used to auto-generate goals, which can go hand in hand with experience replay methods [61][29][94].

Overall, Curriculum Learning is a powerful technique that has the potential to accelerate the learning process and improve the performance of models in a range of settings, including multi-task reinforcement learning. By providing a structured learning process, Curriculum Learning can help models to generalize better and learn more effectively, leading to more robust and adaptable AI systems.

2.5.2 Applications

In recent years, multi-task RL has been applied to a wide range of domains, including robotics and manipulation, multi-task recommendations, and energy and communication efficiency. In this section, we will review some applications of MTRL, and examine a few key papers from each application area.

Robotics and Manipulation

Robotics has become an increasingly important application context for the field of artificial intelligence, particularly in the context of MTRL. The ability to manipulate objects with dexterity and precision is crucial for many real-world applications, from manufacturing to healthcare. In recent years, several papers have made significant contributions to the academic literature on robotics and manipulation. These papers have advanced our understanding of how robots can learn to manipulate objects in complex and dynamic environments, and have paved the way for future research in this exciting and rapidly evolving field.

MT-Opt is a new MTRL technique for robotics proposed by the Google Robotics Lab [54]. The authors note that while existing deep RL methods are effective at learning skills, they require considerable on-robot training time and engineering effort for each task. MT-Opt aims to amortize the cost of learning a large repertoire of behaviors over multiple related tasks, making the learning process more data-efficient and faster. The authors demonstrate the effectiveness of MT-Opt on a variety of robotic tasks and show that it outperforms single-task RL in terms of data efficiency and complexity of tasks that can be performed.

The paper presents several novel contributions to multi-task RL for robotics. First, the authors address the challenge of defining rewards for multiple tasks through a scalable, success-classifier-based approach. Second, they demonstrate how a multi-task system can acquire new tasks quickly through shared representations and learned policies.

Finally, they show that learning multiple related tasks simultaneously can increase data efficiency and enable more complex tasks to be performed. Overall, this paper demonstrates the potential of multi-task RL for robotics and provides a promising new approach for developing general-purpose robotic systems with a large repertoire of skills.

The paper entitled "Learning to Grasp the Ungraspable with Emergent Extrinsic Dexterity" has extended the Robotics and Manipulation literature by presenting a system that addressed the limitations of previous work in an area called extrinsic dexterity [141]. The system was designed to study the task of "Occluded Grasping", which is the ability to grasp objects that could not be accessed directly by first moving the object, e.g., by pushing it against a wall so that it can then become graspable.

The system used a simple gripper and its ability to use the environment to increase that grippers capabilities was referred to as "extrinsic dexterity". Interestingly, the policy learned to push the object against the wall to achieve its goal without any additional reward design to explicitly encourage extrinsic dexterity.

Another significant contribution described in the paper was that the policy trained in simulation was able to be zero-shot transferred to a physical robot. This robot could then take advantage of this extrinsic dexterity technique across a range of object sizes and surface types.

Multi-task Recommendations

Multi-task Recommendations is a crucial aspect of modern-day recommender systems that aims to provide personalized recommendations to users across multiple tasks. With the advent of Multi-task Reinforcement Learning, the ability to learn and optimize recommendations across multiple tasks has become increasingly important.

Multi-task Learning has seen moderate success in Recommender System applications. However, previous MTRL-based recommendation models did not account for the inter-

session patterns and behaviours of user-item interactions, as they were constructed based on only item-wise datasets. Additionally, balancing multiple objectives has been a challenge in this field, which was typically avoided through linear estimations.

Some of these issues are addressed in "Multi-Task Recommendations with Reinforcement Learning" [72]. The method's structure addressed the two aforementioned issues by constructing environment from session-wise interactions and training a multi-task actor-critic network structure that is still compatible with prior MTRL-based recommendation models.

The effectiveness of the paper's agent was demonstrated through experiments on two real-world public datasets, which showed a higher statistical relevance of recommendations when compared against current state-of-the-art models. Overall, the paper extended the Multi-task Recommendations literature by proposing a novel RL-enhanced MTL framework that addressed the issues of disregarding session-wise patterns and balancing multiple objectives. The proposed framework showed promising results and could be applied to various RS applications.

Energy and Communication Efficiency

In recent years, multi-task reinforcement learning (RL) has emerged as a promising approach for solving complex problems that require multiple objectives to be optimized simultaneously. However, as the number of tasks and agents involved in the learning process increases, so does the demand for energy and communication resources. This has led to a growing interest in understanding the tradeoffs between energy and communication efficiency in multi-task RL applications. One recent paper using MTRL for this application is "On the Energy and Communication Efficiency Tradeoffs in Federated and Multi-Task Learning" [103]. These papers have shed light on the challenges and opportunities associated with optimizing energy and communication efficiency in multi-task RL, and have provided valuable insights for future research in this area.

The paper explored the energy costs associated with MTRL processes in distributed wireless networks. The authors noted that the use of Federated Learning (FL) allows for the development of new strategies for solving multiple learning tasks concurrently by using cooperation among networked devices. MTRL is one such strategy that leverages relevant similarities across tasks to increase learning efficiency compared with standard transfer learning approaches.

The method is examined on a clustered multi-task network environment, where autonomous agents learned different but related tasks. The MTRL process was carried out in two phases: the optimization of the meta-model that is designed to learn new tasks at speed, and a task-specific model adaptation stage where the learned meta-model was transferred to agents for each task and optimized further.

The results of the study showed that the Model-Agnostic Meta-Learning (MAML) method reduces the energy bill by over twice that of previous methods. The findings suggested that MTRL could significantly reduce energy footprints and improve efficiency in wireless networks, which had important implications for the design of future wireless systems.

2.5.3 Performance Benchmarks

In the context of reinforcement learning, benchmarks refer to standardized problems or tasks that are used to evaluate and compare the performance of different learning algorithms or models. These benchmarks may involve simulated environments, games, robotics tasks, or real-world applications, and they are designed to capture specific aspects of the learning problem, such as exploration, exploitation, generalization, transfer learning, or safety.

Benchmarks play a key role in advancing the field of reinforcement learning, as they enable researchers to measure progress, identify strengths and weaknesses of different approaches, and guide the development of new algorithms and techniques. To be

effective, benchmarks should be challenging, well-defined, and representative of the problem domain, and they should be shared openly to enable fair comparisons and reproducibility of results.

Several widely used benchmarks in reinforcement learning include the Atari games, the MuJoCo physics simulator, the OpenAI Gym library, and the DeepMind Control Suite [77][118][12]. These benchmarks have spurred significant progress in the field over the past decade, leading to breakthroughs in deep reinforcement learning, hierarchical learning, meta-learning, and other areas.

The OpenAI Gym Benchmark is a widely used platform for developing and testing reinforcement learning algorithms. It is designed to provide a standard interface for researchers and developers to evaluate and compare their algorithms in a controlled environment. The Gym Benchmark consists of a set of virtual environments, each of which represents a different task or problem that an agent must learn to solve. These environments range from classic control problems, such as balancing a pole, to more complex tasks such as playing video games.

The MuJoCo Benchmark for reinforcement learning is a widely used evaluation framework for benchmarking and comparing the performance of reinforcement learning algorithms across a range of robotic control tasks. The benchmark is based on the MuJoCo physics engine, which provides a highly realistic simulation environment for a variety of robotic systems. The benchmark consists of a set of 20 continuous control tasks, each of which requires agents to learn to control a robotic system to achieve a desired goal. The tasks vary in complexity, ranging from simple tasks such as balancing a pendulum to more complex tasks such as navigating a maze or climbing a pole.

The performance of reinforcement learning algorithms on these tasks is measured using metrics such as the final reward achieved, the number of steps required to achieve the goal, and the stability of the learned policies. The benchmark is designed to encourage the development of algorithms that are both sample-efficient and robust to changes in the environment. Several popular reinforcement learning algorithms, such as deep

deterministic policy gradients (DDPG) and proximal policy optimization (PPO), have been evaluated using the MuJoCo Benchmark, and the results of these evaluations have been used to guide further research and development in the field.

The DexArt benchmark proposed the use of a virtual robot hand to enable better approximation to human behavior and enable the robot to operate on various objects [4]. The benchmark defined multiple complex manipulation tasks within each task. The main focus of the DexArt benchmark is to evaluate the generalizability of learned policies on what are referred to as articulated objects, which are objects consisting of multiple structures pinned of otherwise joined together, such as scissors, buckets with handles, or laptops. This is an extremely challenging given the high degrees of freedom of both hands and articulated objects. This benchmark allows for further investigation into how 3D representation learning affects decision making in RL.

The MineRL Benchmark is a comprehensive evaluation framework and dataset for reinforcement learning agents in Minecraft, which is a popular game often used for artificial intelligence research [35]. It is designed to provide a rigorous and standardized evaluation protocol for comparing different reinforcement learning methods, and for measuring progress in the field over time. The MineRL dataset includes over 60 million frames of gameplay, covering a wide range of tasks such as navigation, tool use, tree chopping, and more. Each frame includes high-resolution visual observations and a rich array of sensory information such as the player's inventory, health, and position. The dataset is accompanied by a set of challenging benchmark tasks that test the agent's ability to perform a variety of Minecraft tasks in different environments, such as building structures or collecting resources. The MineRL Benchmark is unique in that it incorporates several important features to ensure a fair and meaningful evaluation of reinforcement learning agents. For example, the dataset intentionally includes natural variation in the gameplay environment, such as natural hazards or unpredictable behavior from non-player characters. Additionally, the benchmark tasks are designed to be complex and difficult, requiring agents to demonstrate flexible and adaptive behavior,

rather than simply memorizing a set of pre-defined rules.

The ALE Benchmark is a popular platform for evaluating the performance of reinforcement learning algorithms in the domain of video games [7]. The platform consists of a collection of Atari 2600 games that have been modified to allow for programmatic control and data collection. Each game in the benchmark is treated as a separate task, and the performance of a given algorithm is evaluated based on its average score across all tasks. To evaluate performance, the ALE Benchmark uses a variety of metrics, including average score per game, percent of games solved, and median normalized score. These metrics provide a comprehensive view of an algorithm's performance and can help researchers identify its strengths and weaknesses.

One of the major advantages of the ALE Benchmark is its flexibility. Researchers can easily modify the platform to test their own algorithms or add new games to the benchmark. Additionally, the platform provides a rich set of tools for data analysis and visualization, making it easy to compare results across different algorithms and games.

The StarCraft II Benchmark is a standard test suite used to evaluate and assess the performance of multi-agent reinforcement learning in the real-time strategy game StarCraft II [101]. It was developed by researchers at DeepMind and Blizzard Entertainment as part of the StarCraft II AI research environment.

The benchmark consists of a set of 10 mini-games, each with its own specific challenges and objectives. These mini-games are designed to test different aspects of an agent's abilities, such as resource management, unit control, and decision making under uncertainty. In each mini-game, the agent is tasked with achieving a specific goal or objective, such as collecting resources, building structures, or defeating an opponent. The agent must learn to navigate the game environment, gather information, make decisions, and execute actions in a timely and efficient manner.

The StarCraft II Benchmark is considered one of the most challenging and complex environments for reinforcement learning due to the high dimensional nature of the game

state, the large action space, and the high level of uncertainty and partial observability. Therefore, the benchmark provides a rigorous test of an agent's ability to learn and adapt in complex and dynamic environments.

The MetaWorld benchmark is a suite of tasks designed to evaluate the performance and generalization capabilities of reinforcement learning algorithms [134]. It comprises 50 diverse tasks that cover a wide range of scenarios, such as object manipulation, navigation, tool use, and puzzle solving. The tasks are designed with a meta-learning framework, which allows for efficient adaptation to unseen tasks by leveraging the knowledge acquired from solving related tasks.

In addition, the MetaWorld benchmark incorporates a hierarchical representation of tasks, where the tasks are divided into three levels: skill, navigation, and strategy levels. The skill level comprises low-level actions that can be combined to complete more complex tasks. The navigation level involves finding a sequence of skill-level actions that achieve a more abstract goal. The strategy level involves developing a high-level plan to solve a task that requires reasoning and decision-making abilities.

2.6 Replay Methods

Experience Replay is a popular technique in Reinforcement Learning that allows the agent to learn from past experiences stored in a replay buffer. The basic idea behind Experience Replay is to store the agent's experiences during interaction with the environment and sample these experiences mini-batch uniformly at random.

The motivation behind Experience Replay is to break the correlation between subsequent states that are generated during an agent's interaction with the environment. Experience Replay methods have been shown to improve convergence, stabilize learning, and reduce the number of interactions required for an agent to learn a good policy.

Some common Experience Replay methods are:

Basic Experience Replay: This is the simplest form of experience replay, where the agent stores all past experiences in a buffer and samples uniformly from them during learning.

Prioritized Experience Replay (PER): In this method, experiences with high TD error or prediction error are given a higher priority for sampling [105].

N-Step Experience Replay: Instead of updating the Q-value based on a single-state transition, this method takes multiple consecutive transitions and updates the Q-value using Bootstrapped Returns [25].

Hindsight Experience Replay (HER): This allows an agent to learn from failed experiences by adjusting the goal state from the one originally intended to the one that was achieved during the interaction [2].

Generative Adversarial Imitation Learning (GAIL): GAIL is an Experience Replay technique that uses a GAN to learn a policy from expert demonstrations [44].

There are some other important, but less popular, replay methods to note. DPER is a distributed and prioritized version of experience replay (PER) for deep reinforcement learning agents [45]. This method implements a distributed network of agents that can learn from past experiences in parallel. The proposed method prioritizes the experiences that are more informative or have a higher impact on performance, allowing for a more efficient and improved learning process.

ERO aims to improve the computational efficiency of deep reinforcement learning agents [136]. To achieve this, the authors propose a novel experience replay method that samples experiences based on their uncertainty. The method uses an uncertainty estimator that determines how informative each experience is and prioritizes those with higher uncertainty over others. The result is a more efficient learning process that requires fewer samples and can be applied to large-scale networks.

S4RL introduces a simple self-supervised replay method that can learn from unannotated data [111]. The approach uses a deep generative model that can create and evaluate synthetic experiences to improve the agent's policy. The proposed method is particularly useful in offline reinforcement learning where an agent has a fixed dataset and cannot obtain new data. MixUp, for continuous control, generates new experiences by linearly interpolating past ones, creating a continuous transition between them [68]. The authors show that MixUp can improve sample efficiency and help agents learn better control policies, particularly in challenging continuous control environments. Lastly, NMER proposes a neighborhood mixup method that aims to achieve improved sample efficiency in continuous control tasks [102]. The proposed method linearly interpolates between neighboring samples in the replay buffer to generate new experiences. The approach's primary benefit is its ability to constrain the generated samples within local convex regions, which helps the agent generalize better and learn faster.

2.7 Conclusion

In this chapter we have examined the literature relating to geometric deep learning and reinforcement learning that encompass the contributions made within the thesis. We first explored geometric deep learning with a focus on its use with reinforcement learning. The main gap in this research field is mainly in implementing reinforcement learning to generate graphs. Very few reinforcement based graph generation works have been published, in large part due to the fact that the field as a whole is so new. In addition, most graph based work deals with instances where there is already a large amount of data, in which case variational auto encoders tend to be more useful. For the specific use case where we intend to construct graphs with structures not yet seen before, however, reinforcement learning is the ideal solution. This motivates our method for reinforcement learning for graph construction (*Contribution C4*) in chapter five. Additionally, of the graph generation works involving reinforcement learning, most work with the applications for molecules and drug discovery, which tends toward

graphs of smaller size. There is a gap in the literature for geometric learning for the application of distributed systems, motivating our analysis of representation methods for workflows (*Contribution C1*) in chapter three.

We also examined the field of multi-task reinforcement learning. While there are MTRL methods that have used task embeddings as a representation [40], they are for continuous tasks for robotics, not for discrete tasks which better represent workflows, and additionally these embeddings are not composable in nature. This motivates our method presented in chapter 4 which produces plan vector representations (*Contribution C2*) that can then be used for production of workflows (*Contribution C3*). We also investigated benchmarks in the reinforcement learning field, noting a lack of benchmarks with sequential tasks and low computational cost, motivating our production of a benchmark (*Contribution C6*).

Finally, we examine the use of Replay methods in Reinforcement learning, where we find a gap in using full trajectories to increase the efficiency of learning on produced data (*Contribution C5*).

Chapter 3

Representation Learning for Knowledge Graphs

3.1 Overview

Knowledge graphs (KGs) capture complex information about a specific domain using rich logical semantics to encode relationships between entities and allow detailed and specific data querying. Semantic vector spaces (SVSs) encode the relative meanings of terms based on their co-occurrence in data and support useful operations for computing the relationships between these terms.

In this chapter, we report on contemporary methods for SVS representation of KGs, otherwise known as knowledge graph embedding (KGE), particularly as it relates to graph databases based on observational ontologies. These embeddings enable additional analysis tasks to leverage learned semantic vectors to gain additional insights. These techniques learn this representation directly from the graph topology and may be used alternatively—or complementary—to traditional KG queries and natural language processing (NLP) of any associated text corpus.

The aim of this chapter is to survey the general techniques for KGE and its downstream applications, as well as to perform experimental analyses using the Distributed Analytics and Information Sciences (DAIS) International Technology Alliance (ITA) Science Library (SL) co-authorship database to examine how we might produce embeddings

more usefully representative of phenomenological databases [11].

In Section 3.2, we survey methods for knowledge graph embedding, outline the differences between the key approaches and highlight cases where particular approaches may be preferable. In Section 3.3, we outline our experiments using DAIS-ITA SL data, before concluding the chapter in Section 3.4.

3.2 Knowledge graph embedding methods

The formal definition of a knowledge graph is a directed heterogeneous multigraph whose node (entity) and edge (relation) types have domain-specific semantics (often conveyed through an ontology). Knowledge graph embedding (KGE) represents the entities and relations in a continuous vector space. This embedding is a distributional representation, meaning that all properties of a single entity or relation are learned together and the resultant description is spread across every dimension in the vector space, such that no dimension corresponds to a specific attribute. A well-designed embedding method should ensure that the resulting space captures the semantic properties of the data, such that similar entities and relations are grouped together in the space and unrelated concepts are orthogonal to each other (see Section 3.3.1 for more on vector similarity). These embeddings allow additional analysis and machine learning techniques to be applied, such as entity classification, fuzzy matching, link prediction and recommendation systems.

Other research has combined a natural language SVS with queries on knowledge bases (KBs), to leverage analogical reasoning [113]. This is achieved by book-ending the query with semantic expansion. This has been shown to improve KB responses with greater coverage using approximate knowledge, i.e., by querying the KB with the original and analogous tasks, using closely-related examples drawn from the SVS. This enables assessment of the confidence of correctness in the query response.

There are numerous approaches to graph embedding and the field is expanding rapidly.

In this chapter, we mainly consider two different types of framework for learning embeddings of entities: energy-based methods and random-walk-based methods. In the interest of focus and brevity, we don't specifically report on Bayesian clustering and matrix (tensor) factorisation techniques (e.g., DistMult, HolE, ComplEx) [39, 85, 86, 110].

3.2.1 Energy-Based Embedding

In much of the literature, so-called energy-based methods are the only type of models directly surveyed or included in KGE libraries [17, 139, 38]. Essentially, these frameworks learn embeddings by associating pairs of entities with some scalar scoring function or "energy" metric [10, 50, 8]. This is generally also a function of the relationship, such that there is some energy associated with every triple of source node (head h), edge (relation r) and target node (tail t). All existing triples in the graph form a set S. A selection of such models and the corresponding scoring functions is shown in Table 3.1.

Model	Scoring function
SE	$\|M_{rh}\boldsymbol{h} - M_{rt}\boldsymbol{t}\|_1$
Unstructured	$\|\boldsymbol{h} - \boldsymbol{t}\|_{1,2}^2$
LFM	$-\boldsymbol{h}^T M_r \boldsymbol{t}$
TransE	$\|\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\|_{1,2}^2$
TransH	$\|\boldsymbol{h}_\perp + \boldsymbol{r} - \boldsymbol{t}_\perp\|_{1,2}^2$
TransR	$\|M_r\boldsymbol{h} + \boldsymbol{r} - M_r\boldsymbol{t}\|_{1,2}^2$

Table 3.1: A selection of energy-based models and their corresponding scoring functions.

The terms in table 3.1 can be explained as follows:

- $\boldsymbol{h}$, $\boldsymbol{t}$, $\boldsymbol{r}$: Entity embeddings in a vector space, where $\boldsymbol{h}$ is the head entity, $\boldsymbol{t}$ is the tail entity, and $\boldsymbol{r}$ is the relationship (edge) embedding.

- M_{rh}, M_{rt}, M_r: Transformation matrices or projection matrices specific to relationships (r).
- $\|\cdot\|_{1,2}$: Mixed norm or a combination of L1 and L2 norms.
- $\|\cdot\|_1$: L1 norm (Manhattan distance) of a vector.
- $\|\cdot\|_2$: L2 norm (Euclidean distance) of a vector.
- $\boldsymbol{h}_\perp$, $\boldsymbol{t}_\perp$: Entities projected onto hyperplanes specific to TransH model.
- $M_r\boldsymbol{h}$, $M_r\boldsymbol{t}$: Projection of entity embeddings onto relationship-specific spaces using TransR model's projection matrices (M_r).

The energy of each triple $s = (h, r, t) \in S$ is equal to $f(s)$ for some scoring function f. For every genuine triple in the graph, we generate a collection of corrupted triples where either the head or tail entity (but not both) has been replaced by another random entity $s' = (h', r, t') \in S'$.

The embeddings are then learned using a triplet loss function,

$$\mathcal{L} = \sum_{s} \sum_{s'} \max\left(0, \gamma + f(s) - f(s')\right), \tag{3.1}$$

where $\gamma > 0$ is the margin between positive (triple) and negative (corrupted triple) energies. The optimization is performed using stochastic gradient descent.

A simple geometrically intuitive implementation is TransE (translational energy). In this model each entity is learned as a single embedding in the same k-dimensional vector space. The energy of each triple $s = (h, r, t) \in S$ is equal to the similarity measure $\|\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\|_{1,2}^2$ (either the L1 or L2-norm). For every triple in the graph, we sample from the set of corrupted triples. The intention of this is to encourage $\boldsymbol{h} + \boldsymbol{r} \approx \boldsymbol{t}$ when (h, r, t) is a genuine triple and for $\boldsymbol{h}' + \boldsymbol{r}$ to be far from $\boldsymbol{t}$ when it is not. Results show that this model outperforms the previous energy-based approaches (RESCAL, SE, SME, LFM) at link prediction for $hits@10$ and mean rank [9, 10, 50, 8].

A simple extension of this uses a composite model of TransE and a textual-context model for relationship prediction [128]. It is shown to perform better than either model in isolation. SENSE also creates single node embeddings by jointly learning graph structure and the textual information associated with each node [96]. This method is shown to be very effective when each node can be associated with a rich text corpus.

TransH extends TransE by projecting $\boldsymbol{h}$ and $\boldsymbol{t}$ onto a hyperplane for each relationship, which allows entities to participate differently in different relationships [125]. Thus, it better encodes 1-to-N and N-to-1 relationships.

Another extension of TransE is TransR [70]. In this model, entities are embedded in different vector spaces for each relationship, as opposed to TransE and TransH where they share the same space. A projection matrix is learned for each relationship that projects the head and tail entities into the appropriate relationship space before computing the distance. This approach enforces $M_r\boldsymbol{h} + \boldsymbol{r} \approx M_r\boldsymbol{t}$ for positive triples, where M_r is a projection matrix between the entity and relationship vector spaces. This is shown to achieve a higher accuracy on a binary relation classification task than either TransE, TransH, or other pre-existing energy-based methods.

There are several open-source packages that enable one to apply energy-based structural approaches to learn Knowledge Graph Embedding. Two that we have identified are DGL-KE and OpenKE [17, 139]. Both of these packages broadly implement the same range of energy-based triplet-loss models—TransE, TransE (L1), TransE (L2), TransR, RESCAL, DistMult, ComplEx, RotatE. Each package also incorporates a number of efficiency improvements for parallel computing and optimization in multi-CPU and multi-GPU environments.

3.2.2 Random-Walk-Based Embedding

Another approach to learning node embeddings is to use a random-walk in combination with a skip-gram model, which was introduced for Word2Vec [76, 75, 33]. In this

natural language processing context, each word in a sentence is used to predict the words in a surrounding window to learn word embeddings. The efficiency of the skip-gram model is vastly improved by using negative sampling to approximate the full softmax function.

DeepWalk is designed to leverage this skip-gram model in a graph context. A node in the graph is randomly selected, and then a random step is repeatedly taken to a neighboring node building up a sequence of such nodes with some maximum length. This effectively forms a "sentence" of node "words". This process is repeated to construct a corpus of node sequences that can then be used as input to the skip-gram model to build up node embeddings.

LINE extends DeepWalk by using an objective function that incorporates both the first and second-order proximity of connected nodes—DeepWalk accounts for only second-order proximity [117]. It combines this with an optimized edge sampling approach to maintain efficiency. Note that first-order proximity is only applicable to undirected graphs. It has been shown to outperform DeepWalk on a number of applicable datasets.

Node2Vec extends DeepWalk by introducing a biased random walk that provides some control over the random steps taken [34]. It does this by introducing two weight parameters, p and q. Parameter p determines the probability of revisiting a previously visited node, while q influences how far from the source node the walk will go. This provides a lever between breadth-first sampling (targeting structural equivalence) and depth-first sampling (targeting homophily). Setting $p = q = 1$ recovers the DeepWalk model. Node2Vec has been shown to outperform DeepWalk and LINE in multilabel classification and link prediction tasks. However, it is still limited to homogeneous networks with singular types of entities and relationships.

Subgraph2vec learns embeddings for rooted subgraphs in large graphs, rather than individual node embeddings [80]. Its foundation is based on graph kernels. These are kernel functions that calculate the inner products of graphs. They measure the similarity of pairs of graphs by breaking them down into their atomic substructures.

Subgraph2vec extends Deep Graph Kernel to the learning of subgraphs of different degrees occurring in the same local neighborhoods via a skip-gram adapted to allow variable-length radial contexts [131]. Graph2Vec extends this principle to large complete graphs [81].

MetaPath2Vec extends random-walk embeddings to heterogeneous networks [21]. These graphs are defined by $G = (V, E, \phi)$ where each vertex $v \in V$ and edge $e \in E$ is mapped to its respective type by $\phi(v) : V \rightarrow T_V$ and $\phi(e) : E \rightarrow T_E$. This is achieved using meta-path-based random walks, were the probability of a step between nodes is conditioned on the type of the adjoining relationship. These are then used in a heterogeneous skip-gram optimization functions to learn word embeddings. All entities, regardless of type, are embedded in the same low-dimensional vector space. MetaPath2Vec++ additionally employs heterogeneous negative sampling, in which the softmax function is normalized with respect to the context node type. This model have been shown to outperform Node2Vec and LINE in multiclassification, node clustering, similarity search, and visualization tasks on a heterogeneous scholar network.

Heterogenous graphs are also addressed in MetaGraph2Vec [137]. MetaGraph2Vec purports to learn more formative embeddings of sparse graphs than MetaPath2Vec by providing richer structural contexts to measure semantic similarity between distant nodes. It achieves this by proposing a new meta-path guided random walk using meta-graphs, which comprise unions of multiple meta-paths. When generating random walks, this can provide a superset of the random walks generated by individual meta-paths, i.e., it permits more varied walks between any two nodes. This has been shown to outperform MetaPath2Vec at node classification and clustering tasks on given datasets.

Continuous-time dynamic networks graphs are defined by $G = (V, E_T, T)$, where E_T is the set of temporal edges that are mapped to a corresponding timestamp $T : E_T \rightarrow \mathbb{R}^+$. Nguyen et al. propose a random-walk generalization designed to accommodate such graphs [83]. A temporal walk is one that respects the flow time, such that nodes

at earlier time stamps cannot follow those at later ones. I.e., for subsequent edges e_i, e_{i+1} we require $t_i \leqslant t_{i+1}$. Nguyen et al. also propose methods for biased selection of initial temporal edge selection and subsequent edges in the walk. The resultant time-preserving embeddings have been shown to be effective at temporal link prediction, achieving gains over non-temporal models on a number of datasets. This suggests that temporal random walks can model time dependency in networks to learn appropriate and meaningful representations.

GraphSAGE differs from the other approaches listed here because it is an inductive approach [37]. All other models discussed so far can only learn embeddings on fixed graphs: they are transductive. They do not generalize to nodes that were not seen during training and the full graph must be known at training time. However, GraphSAGE can generate node embeddings for new nodes or input graphs "on-the-fly", so long as they have the same ontology as the original training data. This means that the semantic representation can adapt as the graph dynamically evolves over time with the addition or removal of entities and relationships. It achieves this by selecting a single node (the centre of a sub-graph) and then a number of neighboring nodes in each layer. It then aggregates the node embeddings in each layer and passes their embeddings to a neural network that aims to predict the central node. Four different aggregation methods have been proposed: mean node aggregation, GCN aggregation, LSTM aggregation, and pooling aggregation. This model could be used effectively for learning node embeddings for knowledge graphs where new information is added frequently, such as is expected with ontology-based graphs.

PyTorch Geometric (PyG) is a library based on PyTorch for deep learning on graphs and similar structures [27]. It provides a highly optimized framework for Graph Neural Networks (GNNs) with support for multi-CPU and GPU environments. Included are multiple methods random-walk-based embedding learning methods, including Node2Vec and Metapath2Vec. For our experiments, we chose to implement the random-walk-based approaches using this library.

GraphVite is a comprehensive, high-level package that implements both random-walk node-embedding (DeepWalk, LINE, Node2Vec) and energy-based knowledge-graph embeddings (TransE, RotatE) with a focus on high-speed implementation [142]. As a high-level package it may be useful for rapid comparison of multiple models on a given dataset, as well as data visualization; however, its complexity may hinder extension.

3.3 Analysis

In the previous section, we have outlined a wide range of models that can be used to create node and relationship embeddings based on different algorithms. For the purposes of our analysis and demonstration, we have selected a subset to demonstrate the overall utility of graph embeddings and to examine the strengths and weaknesses of different methods.

The models we have selected are TransE (L2) and TransR (using the DGL-KE package—which are examples of energy-based methods), and Node2Vec (using the PyTorch-Geometric library). We have chosen to highlight TransE (L2) and TransR as the literature would suggest that these are, in general, the best performing energy-based techniques in a range of downstream tasks. Node2Vec has been selected as the archetypal example of the random-walk-based methods. We train the models for 30 epochs using a 100-dimensional embedding space. It is probable that different models operating on different datasets would yield better results using appropriately-tuned embedding sizes—in addition to all other relevant hyperparameters. The decision not to explore additional dimensions was primarily guided by the scope and intent of this preliminary investigation. The choice of a fixed 100-dimensional space aimed to establish a baseline comparison across the selected models rather than exhaustively exploring various embedding dimensions. Given the breadth of parameters and hyperparameters affecting model performance, such as embedding size, learning rates, and architectural nuances, our emphasis was on demonstrating the utility of graph embeddings and

comparing specific methodologies. Exploring a broader spectrum of dimensions would indeed offer a more comprehensive comparison; however, due to resource constraints and the primary focus on methodological comparison within this confined environment, the investigation was delimited to this specific dimensionality.

These experiments were conducted for the purpose of proof-of-concept exploration on a private phenomenological dataset for which the DAIS-ITA Science Library (SL) is a publicly-available—yet limited—proxy; the results should be treated as preliminary and purely for the purpose of illustration.

The selection of the DAIS-ITA Science Library (SL) as the basis for experimentation within this study was a strategic decision driven by multiple factors crucial to the research's integrity and feasibility. The DAIS-ITA SL stands as a repository of invaluable scientific literature, encompassing a spectrum of subdisciplines pertinent to our investigative domain. Its composition includes scholarly articles, research papers, and datasets spanning several decades, providing a collection of interconnected information that mirrors the complexities inherent in real-world data.

Moreover, the unique characteristics of the DAIS-ITA SL offer a testbed that replicates the challenges encountered in handling proprietary or sensitive datasets frequently encountered in scientific research. While publicly accessible, the SL operates under controlled constraints, simulating the limitations one might face when working with restricted data sources due to privacy concerns or proprietary restrictions. This quality positions it as an ideal proxy for preliminary investigations and proof-of-concept endeavors, allowing for methodological trials and algorithmic assessments within a confined yet representative environment.

The decision to leverage the DAIS-ITA SL aligns with the ethos of this research endeavor, aiming not only to evaluate graph embedding techniques but also to address the practical constraints often encountered when dealing with real-world, confidential datasets. By utilizing this publicly-available yet restricted repository, the study endeavors to simulate scenarios where data accessibility is limited, urging the develop-

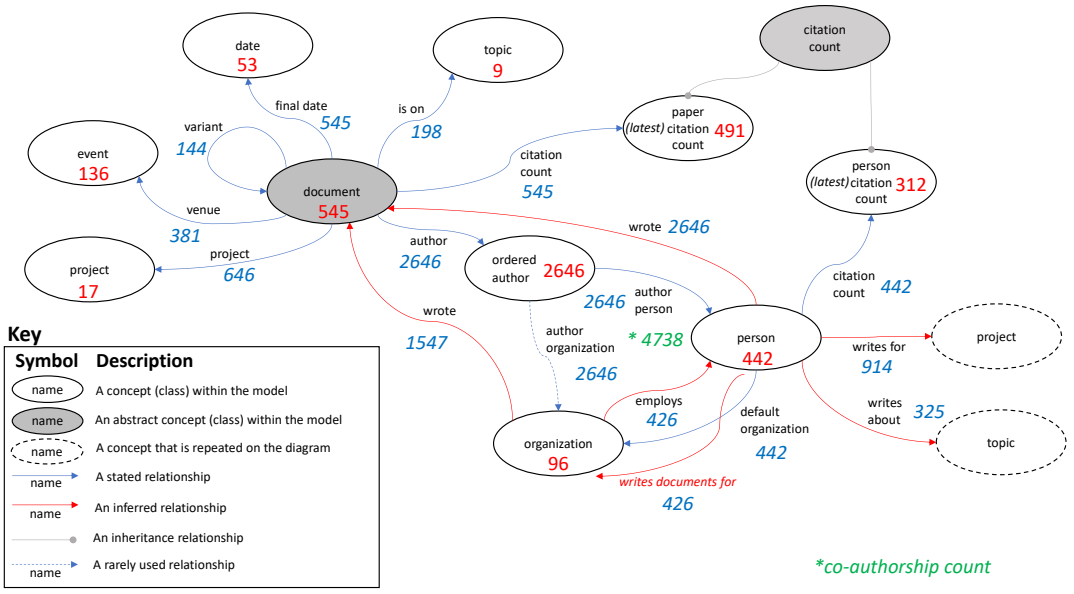


Figure 3.1: A diagram showing the DAIS-ITA SL ontology and the corresponding counts at time of analysis.

ment and assessment of methods that demonstrate robustness and efficacy even under such stringent conditions. As such, the outcomes of these experiments are presented not as definitive conclusions but rather as indicative insights into the adaptability and performance of these graph embedding techniques under controlled constraints, serving as a springboard for further exploration and validation in future studies. The official website for the DAIS-ITA SL may be found at `http://sl.dais-ita.org/science-library`. A diagram illustrating the underlying ontology is shown in Figure 3.1.

3.3.1 Vector Similarity and Concept Matching

The similarity of any two nodes, x and y, can be evaluated using the cosine similarity of their real-valued vector embeddings, $\boldsymbol{x}$ and $\boldsymbol{y}$,

$$\text{similarity} = \cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}. \tag{3.2}$$

Note that in binary vector spaces, the Hamming distance may be used instead. This provides the capability of assessing the similarity of nodes or clusters in a KG via semantic matching, i.e., via a distributed representation of all attributes rather than selecting specific, limited attributes to compare.

This capacity for finding semantically similar nodes or clusters of nodes—representing events in an observational ontology—is a key feature of interest in the application of semantic vector spaces to knowledge graphs. However, this is difficult to evaluate directly as it requires a subset of labelled data—likely provided by a human subject matter expert—that identifies similar entities or events and provides a metric for an embedding to be measured against.

Thus, whilst the semantic vector space itself is trained unsupervised—the model is not told what features are important or which examples of nodes are considered similar—any similarity assessment requires labelled data. I.e., evaluation of embedding quality is necessarily supervised. This knowledge is required in order to quantitatively analyse the success of the embedding and this, in turn, is used during model selection and validation.

Even without this data, quantitative metrics for estimating embedding quality are available. The preliminary results of analyses using a selection of these metrics are presented for the DAIS-ITA SL dataset in the remainder of Section 3.3. However, it should be noted that these are not the most direct metric for the desired feature, which presently can only be provided by human expertise via labelled assessment data.

3.3.2 Energy-Based Link Prediction

For energy-based models, we can determine the mean or $hits@10$ link prediction—without training a downstream classifier model as we do below. $hits@10$ link prediction is the count of how many positive triples are ranked in the top 10 positions when tested in the model alongside a set of synthetic negatives.

First, the full dataset of triples is split into a training set and test set, whilst ensuring that the two resultant graphs remain connected. Every entity in the test set must be present in the training set; these techniques only have embeddings for previously seen data. We then train the embeddings on the dataset for the relevant model (see Section 3.2.1 for a list of methods).

Next, for each test triple, the tail is replaced by every other known entity in the dataset to create corrupted triplets. The energies of each of these corrupted triplets is then determined according the particular model, and they are sorted in ascending order. For $hits@10$, if the correct tail entity is within the top ten lowest energies, then the prediction is considered successful. We can also calculate the mean rank.

While this metric may be indicative of a particular energy-model's improvement over another, it is not useful as a metric across different model types, e.g., random-walk-based models. Therefore, we do not make use of it in this analysis. It is described here for clarification when consulting other literature.

3.3.3 Node Multiclassification

The primary evaluation metric we will use to assess the quality of the embeddings is node classification. Here we evaluate the performance of the embeddings in a downstream (post-training) task aimed at classifying the type of node, given the node's embedded vector as input.

For classifier training, we are careful not to bias the result by selecting 20 vector examples for each class, as the number of nodes for each class is not regular. For example, there are many more nodes corresponding to authors than venues (see Figure 3.1). We then train a limited-memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) logistic regression model, with the node's class as the target variable. We then test on 1000 of the remaining entities and assess the performance using the F_1-score. In the case of the Science Library, there are nine possible types of node, including author, venue,

organization etc. Preliminary results are shown in Table 3.2.

Dataset	TransE	TransR	Node2Vec ($p = 1, q = 1$)
DAIS-ITA SL	0.999	0.492	0.610

Table 3.2: The results of node-multiclassification on the science library dataset

We can see from the results that TransE performs significantly better than the other two models at this task, with an F_1-score of 0.999. This is explained by the fact that the TransE model is inherently sensitive to graph heterogeneity; the type of the edge is embedded in the vector space by the model, as explained in Section 3.2.1.

TransR is also sensitive to heterogeneity, however, it performs more poorly, with an F_1-score of 0.492. This may be due to the increased size of the parameter space, as each relation has its own vector space. The size of the Science Library dataset is relatively small, potentially leading to undertraining at the embedding stage.

Node2Vec does not perform as well as TransE, but, with an F_1-score of 0.610, performs better than TransR. Basic Node2Vec is insensitive to heterogeneity but can leverage homophily and structural similarity in a way that energy-based methods cannot. Thus, while it is not directly aware of different node types, the relatively high F_1-score suggests that it can leverage this to infer the class of the node from the graph topology.

3.3.4 Data Visualization

The typical embedding space that we used for our models is 100-dimensional. In order to visualize this in two dimensions, we use a technique called t-distributed stochastic neighbor embedding (t-SNE) [120]. This is a non-linear dimensionality reduction technique. I.e., it can separate data that cannot be partitioned by a straight line. It does this using the local relationships between points to learn a low-dimensional mapping.

As an overview, t-SNE constructs a Gaussian probability distribution over pairs of objects in the full high-dimensional space using their Euclidean distance as a simil-

(a) Node2Vec ($p = 1, q = 1$)

(b) TransE (L2).

Figure 3.2: Two-dimensional visualization of the DAIS-ITA Science Library (SL) dataset embedded with Node2Vec and TransE, respectively, in 100-dimensions and reduced using t-SNE. Different colors represent different SL node types.

arity metric. It then uses the t-distribution to recreate this probability in the lower-dimensional space. This technique overcomes the crowding of points—where points in high-dimensional space tend to overlap when projected to a lower dimension. It uses gradient descent with a convex loss function to learn this mapping.

While there is no way to truly capture the subtleties of node embedding separation in the high-dimensional space, it does act as a heuristic metric that serves to show the emergence of structure, which indicates that the embeddings are learning to distinguish underlying features in the graph.

Figure 3.2 shows visualizations of the embeddings of science library nodes trained using the Node2Vec (Figure 3.2a) and TransE (L2) models (Figure 3.2b). Note that the clusters do not correlate with entity type (distinguished by color). This indicates that the clusters in two-dimensions capture features of the latent space. It can be observed that the TransE (L2) model appears to better separate the data into distinct clusters than Node2Vec. This is consistent with the results of the node multiclassification task

above.

3.3.5 Potential Limitations

Despite the promising outcomes and performance insights gained from the application of various graph embedding techniques to the DAIS-ITA Science Library dataset, it's imperative to acknowledge the limitations and potential negative aspects inherent in this endeavor.

One of the primary limitations lies in the dataset size and its representativeness. The Science Library dataset, while serving as a valuable resource, might be relatively small in comparison to more extensive and diverse datasets available in other domains. The size constraint can lead to potential underfitting issues, limiting the ability of complex models to generalize effectively. This constraint becomes more pronounced in the context of energy-based models like TransR, where the increased parameter space might exacerbate undertraining concerns, as observed in the multiclassification task.

Furthermore, the evaluation metrics utilized, such as node multiclassification and energy-based link prediction, while informative, possess inherent limitations. For instance, node multiclassification, while demonstrating the strengths of models like TransE in capturing graph heterogeneity, might not fully encapsulate the nuances of real-world knowledge graphs due to the dataset's limited representation of node types. Additionally, energy-based link prediction, although informative for specific model comparisons, might not be universally applicable across different model types, limiting its scope in comprehensive model evaluation.

Moreover, the use of t-SNE for data visualization, while providing valuable heuristic insights into the learned embedding spaces, inherently suffers from the curse of dimensionality. The reduction of high-dimensional embeddings into two dimensions might oversimplify or distort the true distribution of embeddings, potentially masking crucial information in the visualization process.

In the pursuit of evaluating embedding quality and model performance, the reliance on human-labeled data to assess semantic similarity remains a significant challenge. While various quantitative metrics and downstream tasks serve as proxies for evaluation, the absence of explicit human expert validation might limit the holistic assessment of semantic representation in the embedding space.

Another aspect worth considering pertains to the intrinsic biases or limitations of the chosen embedding techniques. Each method, while showcasing distinctive strengths, might also exhibit biases or limitations concerning specific graph structures or types of relationships, which could impact the overall generalizability and applicability of the learned embeddings to broader knowledge domains.

Overall, these limitations and potential negative aspects underscore the necessity for a nuanced interpretation of the obtained results and emphasize the need for future research to address these constraints, potentially through larger and more diverse datasets, enhanced evaluation methodologies, and exploration of robustness across various graph structures and domains. Some of these issues are addressed in section 6.2.

3.4 Summary

In this chapter we have provided a broad overview of relatively recent knowledge graph embedding methods. Many of these techniques are in active development and have found broad applicability in a wide range of industries. We have highlighted a selection of landmark models and discussed their advantages and disadvantages, particularly with regards to heterogeneous, observational knowledge graphs describing dynamic environments. We have presented some illustrative analysis using a co-authorship graph database and outlined several metrics used to evaluate the quality of semantic embeddings.

This chapter addresses the first research question:

RQ1 What are the best representations for encoding workflows for later analysis of distributed systems and to use as a platform for automatically constructing further workflows?

We do this by examining energy based and random walk based embedding methods for graphs and performing experimental analysis using the DAIS-ITA co-authorship database as an example dataset, resulting in the first contribution:

C1 *This examines the benefits of various graph embedding techniques that can be used to represent workflows. These embedding techniques can be used with semantic vector spaces to provide better analysis of workflows in distributed systems. This answers* **RQ1**.

Chapter 4

Compositional Plan Vectors for Multitask Learning

4.1 Overview

Multi-task reinforcement learning has demonstrated success in domains like Atari World and in robotic control benchmarks such as MT-10. However, these benchmarks are lacking in their applicability to real world problems. The Atari suite, for example, could be considered too diverse between individual game types, with no similarity in the tasks being learned. MT-10, a benchmark based on the MuJoCo sawyer arm [134], does have similarity within tasks (while not being too narrow for sufficient evaluation), however, the domain lacks compositionality, in which individual tasks could be combined together in a particular sequence to achieve a larger goal.

Most real-world tasks are compositional in nature whereby they can be divided into a sequence of subtasks where each subtask has clear semantics. For example, the task of making a cup of tea can be divided into a sequence of subtasks which begins with the subtask of boiling water. These tasks and subtasks also have semantic similarity, for example making instant coffee would be more similar to making tea than baking bread. Both of these properties can be leveraged in vector representations, such as in Mikolov et al. [76], which produced compositional representations in a language model with arithmetic properties, such as $\text{“king”} - \text{“man”} + \text{“woman”} = \text{“queen”}$. Current

state-of-the-art multi-task algorithms do not account for the semantic and compositional nature of most real-world tasks. This precludes efficient generalization to wider domains, while simultaneously losing the ability to represent these tasks in a manner that can be useful for representation learning.

Devin et al. [20] proposed compositional plan vectors (CPV) to produce semantic and composable representations for sequential tasks when provided with expert examples under an imitation learning schema. We posit that a multi-task reinforcement learning method that leverages this representation technique will train faster than standard methods that do not account for latent structure.

A well known limitation of imitation learning is the need for expert demonstrations for each task learned. We aim to move the CPV approach into a reinforcement learning paradigm that forgoes this requirement. However, one of the loss functions introduced with the CPV method explicitly depends on the presence of these demonstrations. To solve this, we propose trajectory experience replay (TER), a replay method that allows for the re-use of prior experiences as expert demonstrations, even within environments where successful episodes are highly sparse. By using TER alongside the task representation methods, we produce a reinforcement learning agent that can learn efficiently in compositional environments, in a method we call CPV-TER.

The contributions of our work are two-fold. First, the transfer of compositional plan vectors from an imitation learning paradigm into a multi-task reinforcement learning schema. Secondly, we contribute the TER method, which not only enhances the CPV representation learning, but also provides an effective replay method for multi-task RL on its own. We evaluate CPV-TER on a discrete-action environment in line with prior research, with ablation studies to understand the contributions of each component and performance studies on different environment variants.

4.2 Background

This work is a combination of three separate paradigms: compositional plan vectors, multi-task reinforcement learning, and replay methods for reinforcement learning. In this section we aim to both introduce the formalism used throughout the rest of the chapter and give an overview of each area to show how they can be combined to produce the CPV-TER method.

4.2.1 Compositional Plan Vectors with Imitation Learning

Imitation learning is used to efficiently learn a desired set of behaviors for an agent interacting within an environment. Imitation learning develops these behaviors, or policies π by imitating an expert's behavior. While most popularly used in self-driving cars [14], it has also been used for robotics [74], and in video games [98]. Although training is largely much faster and more efficient than reinforcement learning, these methods require labelled data in the form of trajectories, where reinforcement learning does not. Additionally, imitation learning can be particularly brittle, in that once an agent experiences behavior out of distribution, it significantly struggles to recover and return to useful behavior within a trajectory [46].

Behavioral cloning, a type of imitation learning, relies on an agent producing a policy that can reproduce the same behaviors as an expert demonstration when placed in the same environment. It takes the form of a log loss, which aims to adjust the parameters θ of a policy π, which produces an action $\boldsymbol{a}_t$ given an observation $\boldsymbol{o}_t$ at timestep t:

$$\mathcal{L}_{\mathrm{IL}}(\mathcal{D}, \theta) = \sum_{i=0}^{N} \sum_{t=0}^{H^i} -\log(\pi_\theta^i(\boldsymbol{a}_t^i | \boldsymbol{o}_t^i)), \tag{4.1}$$

using dataset,

$$\mathcal{D} = \left\{ \left(\boldsymbol{O}^{\mathrm{ref}^i}_{[0,T^i]}, \boldsymbol{O}^i_{[0:H^i]}, \boldsymbol{A}^i_{[0:H^i-1]} \right) \right\}_{i=1}^{N},$$

Where:

- $\mathcal{D}$ is the dataset containing N trajectories, each denoted by $(\boldsymbol{O}^{\text{ref}i}_{[0,T^i]}, \boldsymbol{O}^i_{[0:H^i]}, \boldsymbol{A}^i_{[0:H^i-1]})$.
- T^i represents the length of the i^{th} reference trajectory.
- H^i is the length of the i^{th} demonstration trajectory, indicating the number of observation-action pairs in that trajectory.
- The dataset $\mathcal{D}$ comprises expert reference trajectories, observations $\boldsymbol{O}$, and corresponding actions $\boldsymbol{A}$. Each trajectory $(\boldsymbol{O}^{\text{ref}i}_{[0,T^i]}, \boldsymbol{O}^i_{[0:H^i]}, \boldsymbol{A}^i_{[0:H^i-1]})$ is indexed by i, containing the observed states $\boldsymbol{O}$ and actions $\boldsymbol{A}$ along the trajectory.

The loss function quantifies the dissimilarity between the actions produced by the policy and the expert actions at each timestep across all trajectories in the dataset.

Compositional plan vector representations for sequential tasks are introduced in [20]. This method produces a plan vector $g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l)$, parameterized by ϕ, where $\boldsymbol{o}_k$ and $\boldsymbol{o}_l$ are initial and final observations of a trajectory, respectively. The plan vector g is trained to be used as an embedded representation of the entire set of actions required to complete a given task.

These vectors are produced in an imitation learning paradigm alongside two key structural constraints. With these two loss functions, the produced representations are composable; a second plan vector $g_\phi(\boldsymbol{o}_l, \boldsymbol{o}_j)$ can be summed with $g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l)$, to produce a plan vector $g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_j)$ that can perform both tasks effectively without ever having seen that scenario before. The representations are also semantic: a plan vector $g_\phi(\boldsymbol{o}_a, \boldsymbol{o}_b)$ will be more similar to a plan vector $g_\phi(\boldsymbol{o}_c, \boldsymbol{o}_d)$ that represents similar tasks, than it will be to $g_\phi(\boldsymbol{o}_x, \boldsymbol{o}_z)$, a plan vector that represents tasks that are highly dissimilar.

Both loss functions take the form of triplet margin losses as introduced in [106],

$$l_{\text{tri}}(a, p, n) = \max\{\|a - p\|_2 - \|a - n\|_2 + 1.0, 0\}. \tag{4.2}$$

This form enforces the similarity between a and p and the differences between a and n, where a, p, and n are vectors produced by the embedding network.

The Homomorphic Loss Function

The first constraint introduced in [20] is the homomorphic loss function, which enforces compositionality of plan vectors. $\mathcal{L}_{\text{Hom}}$ increases similarity between the representations of the sum of two parts of a trajectory and the representation of the full trajectory,

$$\mathcal{L}_{\text{Hom}}(\mathcal{D}, \phi) = \sum_{i=0}^{N} \sum_{t=0}^{H^i} l_{\text{tri}}(g_\phi(\boldsymbol{o}_0^i, \boldsymbol{o}_t^i) + g_\phi(\boldsymbol{o}_t^i, \boldsymbol{o}_T^i), \\ g_\phi(\boldsymbol{o}_0^i, \boldsymbol{o}_T^i), g_\phi(\boldsymbol{o}_0^j, \boldsymbol{o}_T^j)). \tag{4.3}$$

Compositionality may be defined as the notion that complex structures can be understood through the characteristics of their simpler constituents and the rules used to combine them.

Within the context of reinforcement learning (RL) and plan vectors, compositionality implies that a plan vector encoding the entire sequence of tasks required to achieve a larger task can be "composed" of the plan vectors for each smaller sub-task. This would mean that the plan vector representing a sequence of tasks A -> B -> C could be reconstructed by summing the plan vectors for subtasks A -> B and B -> C.

Therefore, stating it formally, given tasks A -> B (represented by plan vector $g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l)$), and B -> C (represented by plan vector $g_\phi(\boldsymbol{o}_l, \boldsymbol{o}_m)$), the composite task A -> B -> C is represented by the sum of the two preceding plan vectors, i.e.,

$g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_m) = g_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l) + g_\phi(\boldsymbol{o}_l, \boldsymbol{o}_m)$.

This induces a property where vectors that encode similar tasks will be closer together in the embedding space, thereby simplifying the reinforcement learning process.

The homomorphic loss function explicitly enforces this compositional property. It aims to reduce the difference between the plan vector representing the whole task and the sum of the plan vectors representing the sub-tasks. Stated as an objective function, it forms a so-called triplet loss:

$$\mathcal{L}_{\text{hom}} = \max\{\|\mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l) + \mathbf{g}_\phi(\boldsymbol{o}_l, \boldsymbol{o}_m) - \mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_m)\|_2 - \|\mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_j)\|_2 + 1, 0\}.$$

The first term inside the maximum function, $\|\mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_l) + \mathbf{g}_\phi(\boldsymbol{o}_l, \boldsymbol{o}_m) - \mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_m)\|_2$, computes the Euclidean distance (or L_2 norm) between the sum of plan vectors for individual tasks and the plan vector for the composite task.

The second term, $\|\mathbf{g}_\phi(\boldsymbol{o}_k, \boldsymbol{o}_j)\|_2$, measures the Euclidean distance to a non-relevant reference vector. This ensures that the optimization balances the similarity of relevant vectors against the distance to irrelevant vectors, thus fostering the semantic and compositional properties of plan vectors.

By minimizing this maximized distance, the learning algorithm effectively seeks to ensure that the summed plan vectors (representing subtasks) align as closely as possible with the plan vector of the whole task, thereby reinforcing the compositional property of plan vectors.

The Pairwise Loss Function

The second constraint, called the pairwise loss $\mathcal{L}_{\text{Pair}}$ is first introduced in James et al. [49]. This loss function regularizes the embeddings to improve their semantic similarity,

$$\begin{aligned}\mathcal{L}_{\text{Pair}}(\mathcal{D}, \phi) = \sum_{i=0}^{N} \sum_{t=0}^{H^i} l_{\text{tri}}(g_\phi(\boldsymbol{o}_0^i, \boldsymbol{o}_T^i), g_\phi(\boldsymbol{o}^{\text{ref}^i}{}_0, \boldsymbol{o}^{\text{ref}^i}{}_T), \\ g_\phi(\boldsymbol{o}^{\text{ref}^j}{}_0, \boldsymbol{o}^{\text{ref}^j}{}_T)),\end{aligned} \tag{4.4}$$

for any $j \neq i$.

Encouraging semantic similarity among plan vectors is of utmost importance in reinforcement learning (RL), especially when working with environments that have compositional or sequential tasks. What we mean by semantic similarity here is that plan vectors representing similar tasks should be similar, while plan vectors of dissimilar tasks should markedly differ. For instance, the plan vectors representing making tea and making coffee should bear more resemblance to each other than making tea and,

say, baking bread. This semantic similarity in plan vectors arises from the fact that the subtasks involved in making tea and coffee have a significant overlap, such as boiling water or adding a beverage base.

The rationale behind fostering semantic similarity within plan vectors has both practical and theoretical implications. From a practical standpoint, this allows for meaningful transfer learning, which is a cornerstone of multi-task RL. Agents can effectively leverage experiences from related tasks rather than learning each task in isolation. In the process, the agent harnesses shared sub-tasks across different tasks, leading to better performance and optimized learning.

From a theoretical standpoint, semantic similarity in plan vectors establishes relevance within a latent task space. This modeled task space, described by the plan vectors, in turn encapsulates the underlying structure of the task domain. As a result, an RL agent learns to recognize and associate specific tasks with corresponding parts of the task space, leading to enhanced structure learning.

The pairwise loss regularizes the embedding space by comparing the "distance" between plan vectors representing similar tasks and those representing dissimilar tasks. The function attempts to minimize the difference between similar tasks while maximizing the dissimilarity of unrelated tasks.

The pairwise loss function, as defined in Eq. 4.4, is a form of the 'Triplet Loss', initially introduced in deep metric learning for training embeddings [106]. Here, the input usually involves three instances: an anchor point, a positive point (similar to the anchor), and a negative point (dissimilar from the anchor). The loss is then computed by ensuring that anchor and positive points remain close, while the anchor and the negative point are separated by a certain margin in the embedding space [106].

For CPV-TER, the anchor point is the current trajectory's plan vector, and the similar point is a reference trajectory's plan vector. The dissimilar point is another random reference trajectory. By minimizing this loss, we ensure that plan vectors represent-

ing similar tasks are closer together (semantic), while dissimilar tasks diverge in the embedding space. This way, the pairwise loss function achieves regularization and encouraging semantic similarity in plan vectors.

These two loss functions, combined with the imitation learning architecture presented in the Devin et al. paper, can learn task sequences that involve twice as many skills as the trajectories seen during training. While $\mathcal{L}_{\text{Hom}}$ is self-supervised, $\mathcal{L}_{\text{Pair}}$, and the standard imitation learning loss function used in the paper, require reference trajectories. Although learning with reference trajectories is faster and can be more effective than other methods, reference trajectories are often not available in real-world scenarios, or there are not enough to produce effective policies. We posit that this method can be extended to a reinforcement learning space when combined with an additional trajectory storage method.

4.2.2 Multi-Task Reinforcement Learning

Reinforcement learning, like imitation learning, is used to develop policies π for agents to interact within a particular environment. Unlike imitation learning, however, it does not rely on expert demonstrations. Instead, the agent explores the environment by performing different actions and observing the resulting environment state, and receives numerical rewards from the environment itself. This reward is then used to better inform the policy to allow the agent to make better decisions.

For the purposes of this chapter, we have focused on the rainbow method [42], which is a variant of the value-based Q-learning algorithm [127]. Q-learning algorithms train a policy network to select an action that will bring the most future value for the agent, or the highest Q-value. The Q-value is updated after each timestep,

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a'), \tag{4.5}$$

where $\gamma \in [0, 1]$ is a discount for long term rewards.

Multi-task RL is a new development in the reinforcement learning space, with the first OpenAI environments released in 2018 [91]. Agents in these environments must learn policies that determine value from both the observation $\boldsymbol{o}$ and goal g, as first presented in [104]. Note goal g is separate from the function that produces goal vectors g_ϕ. The reward in the environment is then dependent on the goal specified, i.e., $r(s, a, g)$.

There are multiple methods to develop these policies that have been explored within the literature. Many of these methods focus on policy distillation of task-specific agents into a single model that can perform all tasks, such as DISTRAL [100, 119]. Others, including PopArt, learn multiple tasks in parallel with asynchronous actors [43, 88]. Methods including UNREAL [48] focus on off-policy learning on a single course of experiences, as opposed to acting in parallel. We focus on this latter type of multi-task methods, and aim to develop an agent that can learn a policy by interacting with all tasks simultaneously.

4.2.3 Replay Methods for Reinforcement Learning

Experience replay methods are used in conjunction with reinforcement learning and are found in nearly all of the current state-of-the-art RL methods [42, 126, 67]. There is a wide array of experience replay methods, but here we focus on two in particular: prioritized experience replay (PER) and hindsight experience replay (HER).

Prioritized experience replay, first introduced for DQN [105], is possibly the most well known replay method. In prior methods, experiences were uniformly sampled from a memory buffer to be used in training. PER introduces a criterion to assess how important each timestep stored in the memory is for training, and samples transitions with an emphasis on more important transitions. For Q-learning methods, this criterion is the TD-error δ_i produced with the most recent model (as opposed to the model used to produce the transition),

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\theta^-}\left(s_{t+1}, a\right) - Q_\theta\left(s_t, a_t\right), \tag{4.6}$$

where Q_{θ^-} is a target network. Transitions with the largest δ_i are replayed more often, thereby increasing training frequency in areas that produce the largest differential between expected and actual reward.

Hindsight experience replay, initially used for deep deterministic policy gradients (DDPG), is of particular use to multi-task reinforcement learning [2]. When rewards are dependent on a specific goal, as described Section 4.2.2, reward spaces can become highly sparse. To increase sample efficiency, HER introduced the concept of sampling the entire set of goals available within the environment for storing transitions. Although a particular transition may not have achieved the goal provided to the agent, and thus earn zero or negative rewards, that transition may have been considered successful if working towards another task. By storing the transition as a successful transition for another goal, we greatly increase the exploration efficiency of an agent.

4.3 Sequential Multi-Task Environments

Although multi-task RL has become an increasingly popular area of research over recent years, there are still few standardized multi-task environments. Many papers have used adapted versions of robotic arm environments developed with the MuJoCo physics engine [41]. While a popular choice, this is a continuous domain as opposed to discrete, and the goals have only a small sequential range, which makes this domain less than ideal for testing our method. The current standard benchmark for multi-task and meta-RL is meta-world, a series of MuJoCo-based tasks of varying difficulty. The most tested benchmark within this is MT-10, which evaluates ten tasks: reach, push, pick and place, open door, open drawer, close drawer, press button top-down, insert peg side, open window, and open box. These tasks have fixed object and goal locations. Although this is a well developed benchmark, the tasks don't have a sequential nature, which is where our method is expected to perform significantly better than other multi-task methods. For testing of the CPV-TER method we require an environment with a

sequential structure for the various tasks.

For the purposes of this chapter we opted to use the crafting-world environment first presented in the original Compositional Plan Vector paper [20], but adapted it into a standardized OpenAI Gym API structure [12]. Its adaptation into the standardized OpenAI Gym API structure further reinforces its compatibility with contemporary RL research methodologies, positioning it as an accessible and widely accepted benchmarking environment. This is the standard structure for nearly all current RL papers. By using this environment, we could more directly test the efficacy of our RL method (although by design it will not work as fast as the original imitation learning method since we will not provide the system any expert examples). Additionally, by bringing it in to the standard gym API structure, we could make some changes to the representations of the environment, while keeping the mechanics of the grid world true to the original paper. Our environment is packaged for general use at:
`pypi.org/project/gym-craftingworld/`.

Unlike prevalent benchmarks like MuJoCo-based tasks or Meta-World's MT-10, the crafting-world environment caters explicitly to sequential task complexities, aligning closely with the inherent nature of our proposed RL methodology. Within this environment, tasks exhibit a connected, step-by-step nature, requiring agents to execute actions in a coherent sequence, thereby fostering an ideal setting for assessing the efficacy of our proposed CPV-TER method within a sequential context.

This environment can then be used as a benchmark environment for other multi-task and meta-learning agents, particularly for those aiming to solve tasks of a sequential nature. Additionally, the lightweight nature of the environment, as opposed to more intensive environments such as MuJoCo, Starcraft, Atari and MineRL, allow for testing of agents while using less computational power, which is a particular focus of green AI [109]. This attribute not only expedites experimentation but also democratizes access for researchers with varying computational resources, facilitating broader engagement and benchmarking opportunities within the RL community.

Moreover, the availability of the crafting-world environment as a packaged and standardized API via the PyPI repository (`pypi.org/project/gym-craftingworld/`) contributes to its utility as a benchmarking tool. Its integration into the OpenAI Gym framework fosters ease of adoption and allows for potential modifications in environment representations while preserving the fundamental grid-world mechanics, empowering researchers to explore diverse experimental paradigms within a familiar and adaptable framework.

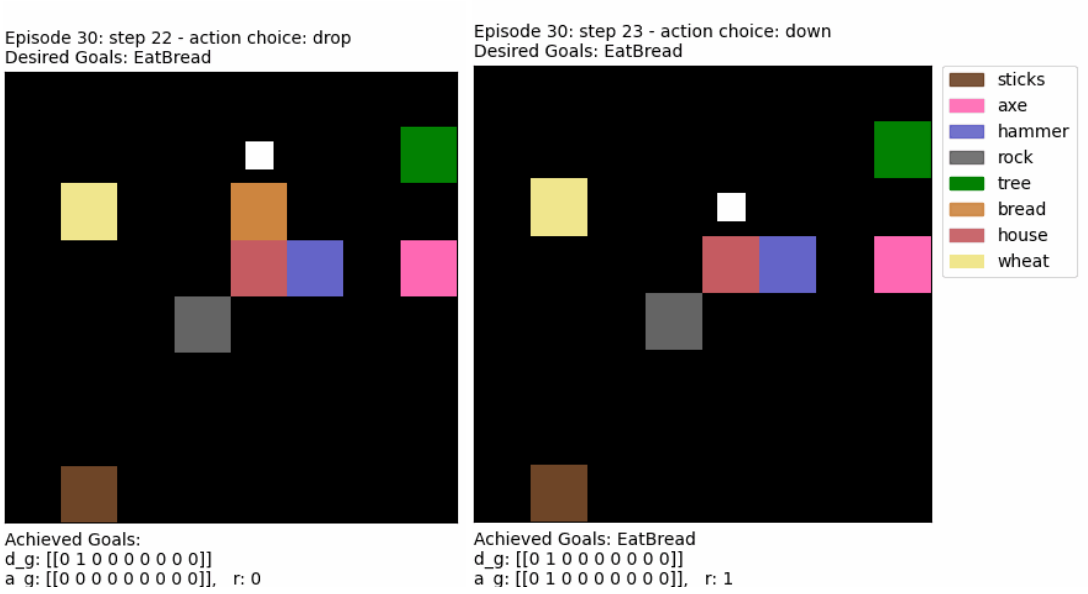
In summary, the crafting-world environment's selection stems from its alignment with our research objectives, catering specifically to sequential task structures, offering computational efficiency, and providing a standardized and accessible platform for evaluating and advancing multi-task and meta-learning agents within the RL domain. We discuss the more pertinent mechanics of the environment in Section 4.3.1, but a more detailed description of the environment can be explored in the documentation at `https://gym-craftingworld.readthedocs.io/`.

4.3.1 Crafting Environment

The crafting environment has three general task types, with increasing difficulty:

1. *Locating*. This task type requires the agent to locate a specified item and move to that location. Two specific tasks that fall under this category are `EatBread` and `GoToHouse`. While these can be straightforward in most scenarios, there are certain environment states that require additional planning, such as if the bread or the house need to be constructed, or if there are rocks in the way that either need to be navigated around or destroyed.

2. *Moving*. These tasks, including `MoveHammer` and `MoveSticks`, involve locating the item required, and using the `pickup` and `drop` actions in the correct order. Perhaps more involved is when the agent is explicitly *not* provided this task: if the agent uses one of these items for an additional task, the agent needs



(a) (b)

Figure 4.1: Two timesteps from the `craftingworld` environment shown side-by-side—the agent moves over the bread, and the achieved goal a_g and reward are updated.

to learn to return the item to the original location at the end of the episode to achieve the highest reward.

3. *Manipulating*. This task type requires locating the appropriate tool and navigating to the correct item in order to turn it into a final specified item. For example, `MakeHouse` requires the agent to navigate to the hammer, pick it up, and navigate to the sticks in order to achieve the specified goal. Many of these tasks can build up on each other in a specified order; e.g., in an environment without sticks available, the agent can perform the `ChopTree` task first to gain the materials required.

States: The state of the environment contains all the information about the current contents of the grid world, including the location of the agent and what items it may or may not be holding. The gym environment can provide this to the agent in two

formats: RBG and one-hot. RGB is the standard method used in reinforcement learning applications, such as those used to solve various Atari games [77]. For the basic `craftingworld-v2`, which is an eight-by-eight grid world, the observation is an RGB image of the screen, with an array of shape $(32, 32, 3)$. Each area of the environment is represented by a 4x4 pixel square resulting in the $8 * 4 = 32$ length for the x and y dimensions. The observation has a z dimension of 3 to hold each of the 3 RGB values. The environment can also return the state as a one-hot encoding of the grid world contents, i.e., the observation is an array of shape $(8, 8, 12)$—the z dimension is a 12 length vector which contains a slot representing each of the 8 possible objects present at that location, one for the agent, and one for each of the three items that the agent can pick up—the axe, the hammer, and the stick.

Actions: The actions space for the crafting environment consists of six discrete actions. Four of these actions are for movement across the grid world—the agent can move one space per timestep in any of the four cardinal directions. If the agent attempts to move off the grid (e.g., by selecting `Up` when already at the top of the grid), no action is performed for the timestep. The agent can move over spaces that contain items, with the exception of spaces that contain `Tree` or `Rock` items. In this instance, the item presents as an obstacle the agent must navigate around, unless an agent is holding an `Axe` or `Hammer` item, respectively. If the agent is holding the appropriate tool, the `Tree` is converted to `Sticks`, the `Rock` item is destroyed, and the agent can successfully navigate to that space in the grid. The other two actions are `PickUp` and `Drop`, which must occur when the agent is on the same space as the item. The agent can pick up and hold only one item at a time.

Goals: Standard multi-goal reinforcement learning environments represent the goal as an exact representation in a similar format to the observation. In Atari methods, for example this could be an exact image of the screen, for robotic tasks this could be an exact location of the object the robot needs to move. For our purposes we are

attempting to have the method not only learn plan vectors, but how to generalize the goals themselves. This means instead of giving an RGB image of the stick item moved two spaces, we want to provide the agent with a more general `MoveStick` goal. To do this we represent the list of all possible tasks as a one hot encoded vector, as shown in Fig. 4.1.

Rewards: Within this environment the rewards structure is a binary and sparse reward scenario, where the reward is one if the achieved goals vector is the same as the desired goals vector, and 0 otherwise,

$$r(s, g^d) = \begin{cases} 1 & \text{if } g^d - g^a = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

where the goal achieved g^a is calculated in the environment and depends on the state.

State-goal distributions: For all tasks the initial number of items, and positions in the grid-world are fixed, as are items within the MT-10 benchmark. For later experiments, we aim to randomize the positions of the objects to determine if the policy can effectively learn to adapt to these scenarios.

4.4 Methodology and Implementation

The CPV-TER is a value-based method adapted from DQN, with variations in three key areas: the network structure, the optimization step, and the experience replay structure. Pseudocode for the CPV-TER method is shown in Alg. 4.2.

The CPV-TER framework is based on two main components: the Compositional Plan Vectors (CPV) and Reinforcement Learning theory. Both have a strong theoretical foundation in the literature.

Algorithm 1 Compositional Plan Vectors with Trajectory Experience Replay (CPV-TER)

Given:
- an off-policy RL algorithm $\mathbb{A}$ # e.g. DQN, Rainbow
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. e.g. $r(s, a, g) = 0$ (fail), 1 (success)

Initialize neural networks for $\mathbb{A}$
Initialize replay buffer R, sorted inner buffer R_s
for epoch $= 1, M$ **do**
  for episode $= 1, N$ **do**
    Sample a goal g^d (desired goal) and an initial state s_0.
    Set goal achieved g^a_0 to a zero vector for t_0
    for t $= 0, T-1$ **do**
      Sample an action a_t using the behavioral policy from $\mathbb{A}$:
        $a_t \leftarrow \pi(s_t||g^d||g^a_t)$ # || denotes concatenation
      Execute the action a_t and observe a new state s_{t+1} and achieved goals g^a_t
      $r_t := r(g^d, g^a_t)$ # calculate reward (g^a_t calculated with s_t and s_0)
      Store the transition $(s_t||g^d||g^a_t, a_t, r_t, s_{t+1}||g^d||g^a_{t+1})$ in R
      Store the goal state from the transition in the sorted buffer $R_s(g^d)$ # TER
      Sample a set of additional desired goals for replay $G := \mathbb{S}$ (current episode)
      for $g'^d \in G$ **do**
        $r'_t := r(g'^d, g^a_t)$
        Store the transition $(s_t||g'^d||g^a_t, a_t, r'_t, s_{t+1}||g'^d||g^a_{t+1})$ in R # HER
        Store the imagined goal state in the sorted buffer $R_s(g'^d)$ # TER
      end for
    end for
    Sample a mini-batch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ with $\mathcal{L}_Q$ and $\mathcal{L}_{\text{hom}}$ and minibatch B
  end for
  Train $\mathbb{A}$ from the collected trajectories in R_s with equivalent g^d vectors using $\mathcal{L}_{pair}$
  Update the priorities in R using the trained model
end for

Figure 4.2: Pseudocode for CPV-TER

The reinforcement learning component of CPV-TER is heavily dependent on the fundamentals of Q-learning [127]. Q-learning is a method for estimating the value of taking different actions from a state in a Markov Decision Process (MDP), within the broader realm of Temporal Difference (TD) Learning. The theoretical grounding of Q-learning and TD Learning is provided by the concept of MDPs and Bellman's Equation. An MDP is defined by the tuple (S, A, P, R) [115], where S denotes the state space, A the action space, P the transition probabilities of reaching state s' when action a is taken in state s, and R is the reward function.

The evolution of Q-learning from a single-agent algorithm to a multi-task algorithm

used an extension of the reward function beyond single task contexts to that catered to multiple tasks [104]. As a result, the reward function, originally denoted by $r(s, a)$ in the single task context, evolved to a multi-task context, as denoted by $r(s, a, g)$ in the multi-task context.

Compositional Plan Vectors (CPV) have their theoretical grounding in imitation learning. Imitation learning, first introduced by [116], is a learning framework whereby an agent learns by observing demonstrations by an expert, and then learning a policy that mimics these expert behaviors. In [20], Devin et. al presented the CPV method which builds on the imitation learning schema by encapsulating the characteristics of the tasks using a feature vector, serving as a compact representation of the entire set of action sequences needed to accomplish a task/subtask. CPVs are semantic and composable, meaning that they can represent a combination of subtasks even if the model has never seen this combination during training. The essence of plan vectors is drawn from the word embedding field [76], where words that represent similar meanings are closer in the embedded space, allowing for operations like vector addition and subtraction to result in semantically meaningful outcomes.

As for the Trajectory Experience Replay (TER), its theoretical grounding is rooted in the methodology of Experience Replay [69] which was designed to alleviate problems in RL such as data inefficiency and harmful correlations between successive samples in learning. The TER method, introduced as part of the CPV-TER, improves on this by focusing on successful trajectories rather than solely focused on individual experiences.

While our approach is a novel combination of existing methods, each part's theoretical grounding is individually robust due to an extensive history of research and application. In combining these theories, we have established a strong foundation that ensures the effectiveness of CPV-TER across various applications.

It is often challenging to provide mathematical proofs in reinforcement learning due to the empirical nature of these methods. Instead, effectiveness is typically demon-

strated through rigorous empirical evaluation, such as performance studies on different environment variants. Future work on CPV-TER can involve theoretical analysis on its convergence properties inspired by convergence proofs in traditional Q Learning [47] or the theoretical validation for performance improvement by experience replay [26]. Such theoretical analysis, however, is out of the scope of this thesis.

4.4.1 Network Architecture

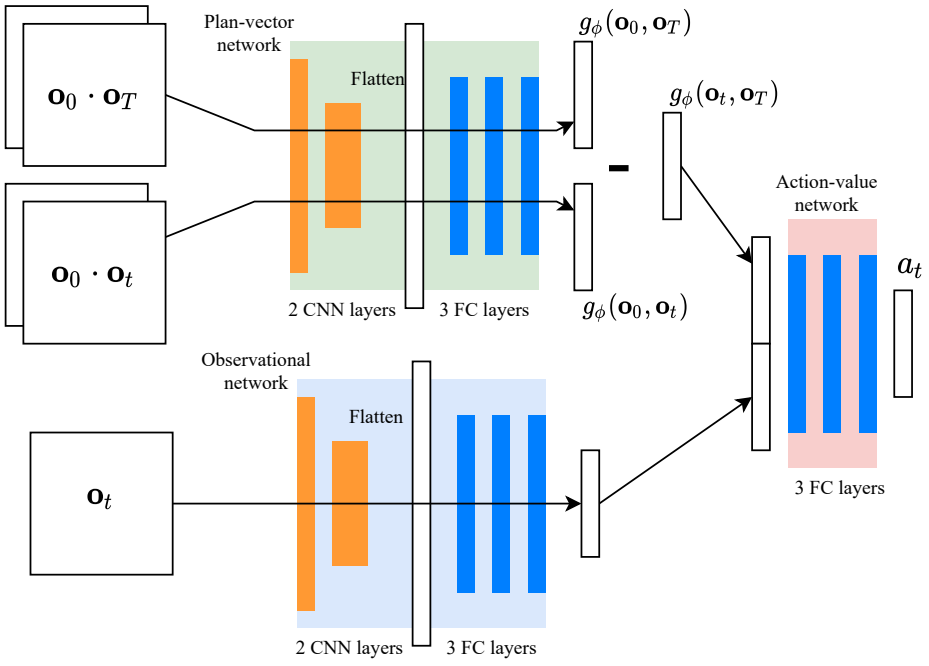


Figure 4.3: Policy network architecture of the CPV-TER method, consisting of an Observation, Plan Vector, and Action Value subnetwork. The green network produces plan vectors from the concatenation of two images, and the red policy network takes the plan vector and the convolution of the current observation to return a six unit vector representing the values of each action. The agent selects the highest value action under a greedy policy.

The neural network used to produce behavioral policies in CPV-TER consists of three networks working in conjunction. The network illustrated in blue in Fig. 4.6, which

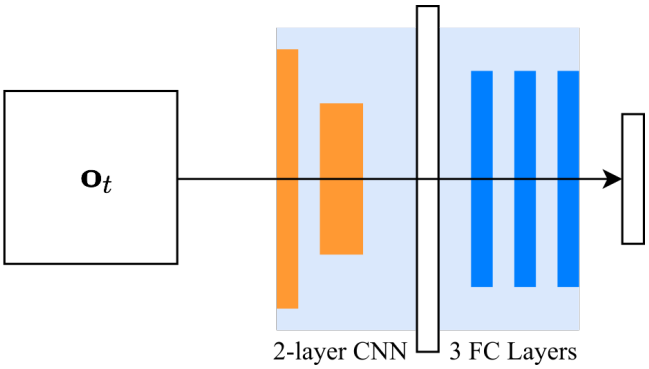


Figure 4.4: Observation network architecture of the CPV-TER method. The network takes in the observation of the current timestep o_t, in the form of an image. The output is a vector representation of the observation. The network is composed of 2 convolutional layers, a flattening layer, and 3 fully connected layers.

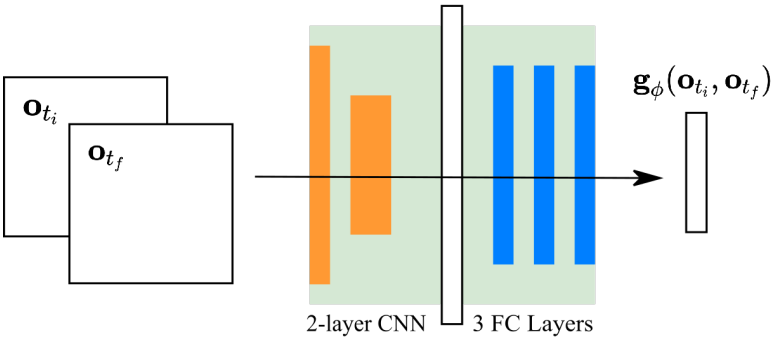


Figure 4.5: Plan vector network architecture of the CPV-TER method. The network takes in the observations of two timesteps o_{ti} and o_{tf}, each in the form of images, that are concatenated together. The output is a vector representation of the 'plan' needed to go from o_{ti} to o_{tf}. o_{ti} is typically the current timestep in the environment and o_{tf} is the goal state. The network is composed of 2 convolutional layers, a flattening layer, and 3 fully connected layers, identical to the observation network.

we will refer to as the 'observation network' is a CNN that takes the observation of the current state o_t, represented as an RGB image as described in Section 4.3.1. This CNN contains two convolutional layers, two pooling layers, and three fully connected layers

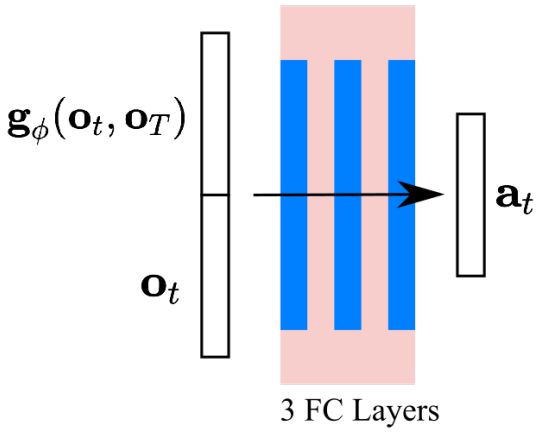


Figure 4.6: Action value network architecture of the CPV-TER method. This network takes in the outputs of the plan vector network and the observation network, the plan vector $\mathbf{g}_\phi(o_t, o_T)$ and the vector representation of the current timestep's observation $\mathbf{o}_t$, respectively. These two vectors are concatenated into a single vector and then used as an input into the action value network. The output is a vector representing the potential reward values for each of the possible actions, which is a standard output for policy based RL models. The network is composed of 3 fully connected layers.

with 64 hidden units and ReLU activations. Instead of returning an output of a six unit vector to represent each action-value, this sub-network instead outputs a representation of the state in the form of a 256 length feature vector, to be used as an input for the action-value network.

The output network, which we will refer to as the 'action-value network', is illustrated in red in Fig. 4.6. This is an MLP network that contains three fully connected layers with ReLU activations, which outputs the six unit vector to represent the value of the six possible actions. This action-value network is the most similar to standard policy networks that operate on Q-values. A standard network would take the current state observation as input and produce the six action values within a single network. Here, our network instead takes the processed 256-length feature vector produced with

the observation network, concatenated with another 256-length vector called the plan vector $g_\phi(\boldsymbol{o}_t, \boldsymbol{o}_T)$.

The plan vectors, $g_\phi(\boldsymbol{o}_a, \boldsymbol{o}_b)$, represent the sequence of subtasks required to get from observation $\boldsymbol{o}_a$ to observation $\boldsymbol{o}_b$. These plan vectors are semantic and composable, as described previously in Section 4.2.1. The plan vector network, displayed in green in the figure, has the same structure as the observation network, with two convolutional layers, two pooling layers, and three fully connected layers with 64 hidden units and ReLU activations. Instead of taking in a single image as an input as in the observation network, the plan vector network takes two images concatenated together. To produce the plan vector $g_\phi(\boldsymbol{o}_t, \boldsymbol{o}_T)$, it takes the current timestep image $\boldsymbol{o}_t$, and an image $\boldsymbol{o}_T$ 'imagined' by the environment to represent the desired goal state g^d. These are concatenated together as $\boldsymbol{o}_t \cdot \boldsymbol{o}_T$. This produces the plan vector $g_\phi(\boldsymbol{o}_t, \boldsymbol{o}_T)$, which is concatenated with the feature vector from the observation network and used as an input for the action-value network.

In addition to the plan vector $g_\phi(\boldsymbol{o}_t, \boldsymbol{o}_T)$, also called the 'forward' vector, the plan vector network is also used to produce two additional vectors. The first is the 'backwards' vector, $g_\phi(\boldsymbol{o}_0, \boldsymbol{o}_t)$, which represents the task sequence required to get from the initial state to the current state. This is made when the initial and current observations $\boldsymbol{o}_0 \cdot \boldsymbol{o}_t$ are provided to the plan network. The second vector is the 'full' vector, $g_\phi(\boldsymbol{o}_0, \boldsymbol{o}_T)$, which represents the plan to get from the initial state to the desired goal state. These are not sent to the action-value network, instead they are used in the optimization stage, to ensure composability: when the network is fully optimized, the addition of the forward and backward plan vectors should produce the full plan vector.

4.4.2 Optimization

The CPV-TER method leverages the Markov Decision Process (MDP) structure, which is defined as a tuple $(S, A, \mathcal{P}, R, \gamma, \rho)$ where S is the state space, A is the action space,

$\mathcal{P}$ is the transition probability distribution, $R : S \times A \times S \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and $\rho : S \to [0, 1]$ is the initial state distribution. The performance of an RL agent in an MDP is evaluated by the expected cumulative reward, and the agent's goal is to find an optimal policy $\pi^* : S \to A$ that maximizes this quantity.

Multitask learning is fundamentally motivated by the observation that humans leverage their knowledge acquired from one task while learning another related one [99]. The key idea in MTL is that with appropriate task design, the learner can utilize the common structures among tasks to improve the generalization performance [6].

Taking these principles into account, at the optimization stage, the CPV-TER method uses two additional loss functions alongside the standard Q-value loss function. The homomorphic triplet loss as stated in Eq. 4.3 aims to minimise the difference between the full plan vector $g_\phi(\boldsymbol{o}_0^i, \boldsymbol{o}_T^i)$ and the sum of the forward and backward plan vectors $g_\phi(\boldsymbol{o}_0^i, \boldsymbol{o}_t^i) + g_\phi(\boldsymbol{o}_t^i, \boldsymbol{o}_T^i)$. These vectors can be split at any timestep during the episode to create subvectors, however by splitting at a timestep when a subtask has been achieved (as described by the reward and tasks completed vectors) they can be split into subtasks. This homomorphic triplet loss function reinforces the composability of the plan vectors that prime the policy network, which improves planning ability for the agent when the subtasks are hierarchical. This loss function does not require reference trajectories, so can be used directly alongside the Q-value loss function after each timestep, as in line 22 of Alg. 4.2.

The pairwise loss function, shown in Eq. 4.4, enforces the semantic properties of the plan vectors, i.e., vectors that represent similar tasks are close together in the embedding space. In the imitation learning space in [20], this is implemented by using this loss function with a reference trajectory that is produced by an expert, and comparing it to the trajectory and embedding produced by the policy itself. In reinforcement learning, however, there are by design no expert demonstrations to reference from. Instead of utilizing expert demonstrations, we instead store prior episodes in our tra-

jectory experience replay (TER) storage buffer as described in Section 4.4.3. With this buffer we can obtain reference images $\boldsymbol{o}^{\text{ref}^{i_0}}$ and $\boldsymbol{o}^{\text{ref}^{i_T}}$ to produce a reference plan vector $g_\phi(\boldsymbol{o}^{\text{ref}^{i_0}}, \boldsymbol{o}^{\text{ref}^{i_T}})$. The pairwise loss function then aims to minimise the difference between this plan vector and the full plan vector from the current episode, in order to to enforce the semantic similarity of plans that represent similar tasks. When a prior episode with the same goal is present in the TER buffer, the network is optimised with $\mathcal{L}_{pair}$, as in line 24 of Algo. 4.2.

4.4.3 Experience Replay

Another significant principle CPV-TER leverages is Experience Replay (ER) [78]. ER is a data-efficient learning method that stores past experiences and randomly samples mini-batches from these memories to update the learning model. This random sampling breaks the correlations in the observation sequence and stabilises learning, reducing the amount of data we waste. Theoretically, ER manifests the Dyna architecture [114], which interleaves direct RL updates with model learning and planning steps.

PER [105], an ER variant, is another key concept in CPV-TER. PER is an RL approach which stochastically samples experiences from the replay buffer based on their priority value. Transition priorities can be defined by different metrics such as the absolute TD-error, and experiences with high priority (large TD-error) have a higher chance of being sampled for learning. PER is inspired by the idea that experiences may not be equally useful for learning; therefore, prioritizing the replay can speed up the learning process and increase performance of RL algorithms.

In order to obtain reference trajectories without pre-provided expert demonstrations, we utilise a specialized replay storage method, which we denote trajectory experience replay (TER).

In addition to a standard buffer prioritized by TD-error, TER has a secondary buffer set that is sorted by the desired goal g^d for each transition. This is sorted with the one-

hot encoding of the goal as opposed to the image representation of the goal—as there are nine possible tasks within the crafting environment this requires $2^9 = 512$ inner buffers. As we want to simulate expert trajectories, we aim to store only transitions where the agent has been successful in achieving the desired goal, or the transitions that lead to success for multi-step reward storage. Storing these transitions only for the successful steps in a sparse environment would mean the buffers would only contain minimal examples. To increase the data to these buffers, we adapt the technique used in hindsight experience replay (HER) [2]. To do this, we also recalculate the rewards for a given transition t for an imagined goal $g^{d'}$ which is equal to the current achieved goal g_t^a. This drastically increases the number of successful trajectories within the sorted buffers. To calculate the pairwise loss, the inner buffer with the same desired goal g^d can then be sampled for a reference.

An added advantage of the TER method is it can easily be adapted to store entire trajectories as opposed to singular transitions, as in the CURIOUS method [15]. We can then use these full trajectories as self-produced expert trajectories with behavior replicating loss functions to create a hybrid reinforcement-imitation learning model.

4.5 Results

In this section we analyse the main experimental results. First we show that CPV-TER method performs better than standard reinforcement learning methods for multi-task learning by performing an ablation study of its components. Then we study the performance of the CPV-TER agent on different environments with varying numbers of tasks. Training takes place over 500 to 5×10^3 episodes, depending on the environment being tested. Each episode has an upper limit of 100 timesteps for the agent to achieve the provided goal. If the desired goal is reached the episode is terminated early. The performance of each agent is measured by the score the agent achieves per episode.

The score $S = 100 - \Delta t$ is the upper limit of timesteps subtracted by the amount of

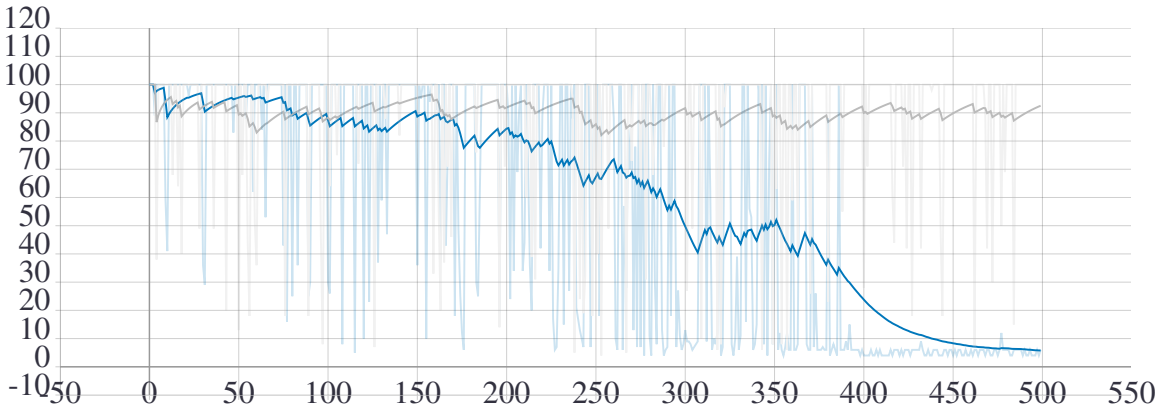


Figure 4.7: Average number of steps required to complete an episode, with a maximum episode length of 100, over a training time of 500 episodes, comparing a typical ϵ-greedy method using linear layers (blue) and a softmax action exploration method using a boltzmann temperature gradient (grey). while softmax can be a useful exploration alternative, it requires extensive tuning and may stuggle with changing objectives as required for multitask learning. This method is a strong avenue for future work.

timesteps taken to achieve the goal (capped at 100). A score of 90 means the agent required ten steps to complete the goal. While the theoretical maximum score is 100, this would mean the agent completed the goal in zero timesteps. The true maximum score of a goal can be anywhere from 99 to 30 depending on the number of subtasks provided and the location of the items required. We compare our methods against two types of baselines. The first is a DQN agent provided with the same network architecture so it can process both the current observation and the goal observation, as opposed to a flat network architecture that would only process one image. This baseline does not use the compositional or semantic architectures, and does not make use of the TER memory, however it does have a replay buffer with n-step rewards and utilizes hindsight experience replay. The second baseline we compare against is a true random agent, that uniformly randomly selects an action at each timestep. This agent is used to provide an insight into the sparsity of the environment and its reward structure.

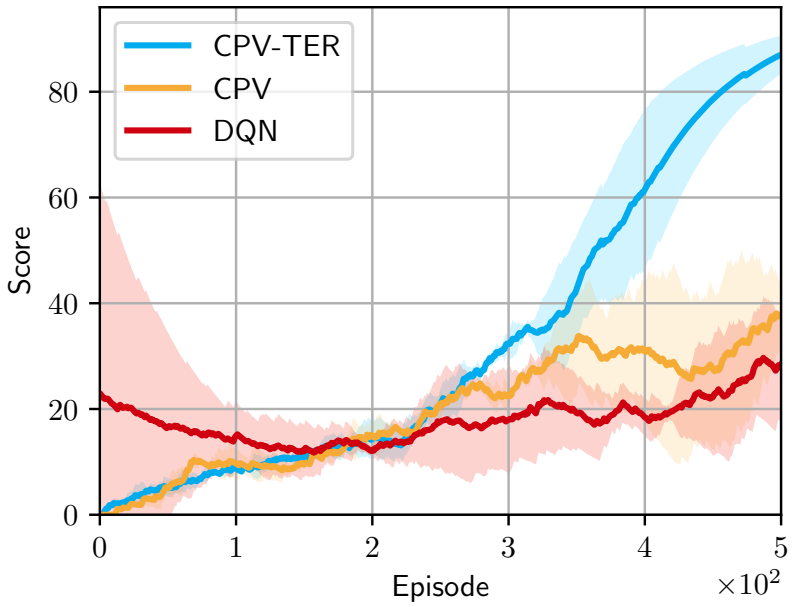


Figure 4.8: Ablation study. Score achieved by an agent per episode, ± 1 standard deviation, with a maximum score of 100, over a training time of 500 episodes, averaged over three runs. Performance is compared between goal-adapted DQN, DQN with CPV losses, and CPV-TER.

Ablation study. In order to fully assess the CPV-TER contribution to the multi-task reinforcement learning space, we performed experiments on the two-task environment with different components of the method removed. We also compared to a baseline DQN approach, which contains none of the additional components from this paper. Fig. 4.8 shows the result of this study. Each agent was trained over 500 episodes, each with a maximum number of 100 steps, resulting in training with an upper limit of 5×10^4 frames.

We can see from Fig. 4.8 that the CPV-TER method has a significantly better performance than any ablation method or standard DQN, with the time taken to achieve the specified goal brought to under ten steps on average per episode by the end of the training run. This results in a performance score that is more than triple the score achieved by DQN with a goal-conditioned network. We can also see from the figure

that while DQN with CPVs does perform better than the baseline DQN, the implementation of trajectory experience replay drastically increases the performance. This is because while the pairwise loss function can still be implemented without TER, the range of 'expert trajectories' is limited to the same episode in training. This prevents the pairwise loss function from being adequately used to reinforce semantic similarity between similar plan vectors.

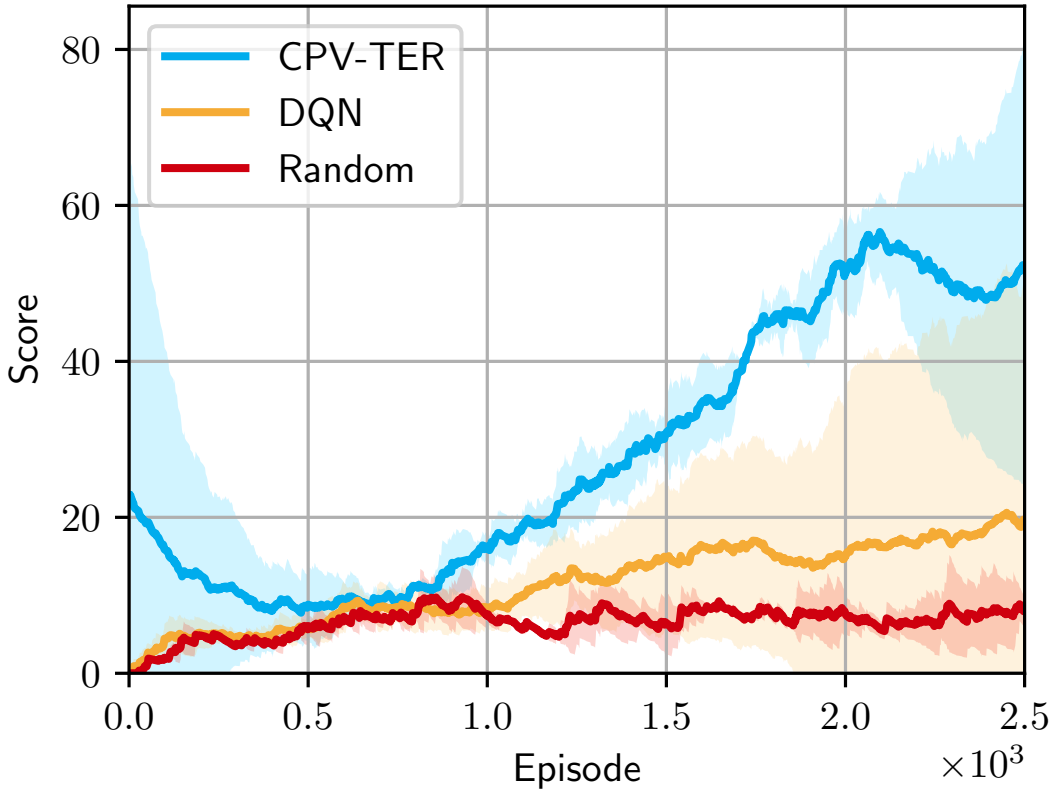
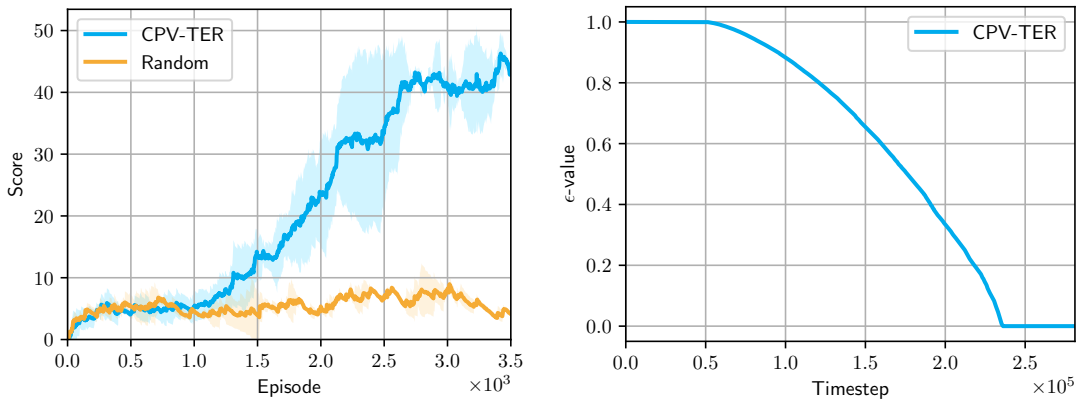


Figure 4.9: Three-task environment. Score achieved by an agent per episode, with a maximum score of 100, with three possible tasks provided as a goal, over a training period of 2.5×10^3 episodes. CPV-TER is compared to standard goal-adapted DQN and a random agent.

Performance Study. The CPV-TER agent was tested on the crafting environment with different numbers of possible tasks provided to them. Fig. 4.9 shows the perform-



(a) (b)

Figure 4.10: Four-task environment. Fig. 4.10a displays the score achieved by an agent per episode, with a maximum score of 100, with four tasks provided as a goal, over a training time of 500 episodes. CPV-TER is compared to a random agent. Fig. 4.10b shows the ϵ-value over the training period for the CPV-TER agent, which decays over time leading from random (exploratory) policies to greedy policies.

ance of CPV-TER in an environment with three possible tasks, one of each general task type as described in Section 4.3.1. This took longer to train, with 2.5×10^3 episodes (up to 2.5×10^5 training steps, but this is a much smaller training time when compared to other baselines within reinforcement learning, which can often use millions of transition images [42]).

Fig. 4.10b shows the shape of the ϵ-value used for ϵ-greedy exploration in all agents. With an exploration heavy start, this provided a wider range of transitions images for storage in the transition experience replay buffers. This means that none of the agents show any learning within the first $15\,\%$ of episodes, as ϵ is set to 1. The ϵ parameter only reaches 0.5 after approximately $55\,\%$ of training is complete, which is where the performance of each agent begins to diverge.

Fig. 4.11 shows the performance of the CPV-TER method on an environment with five possible tasks. We can see that the agent can perform well even in as little as

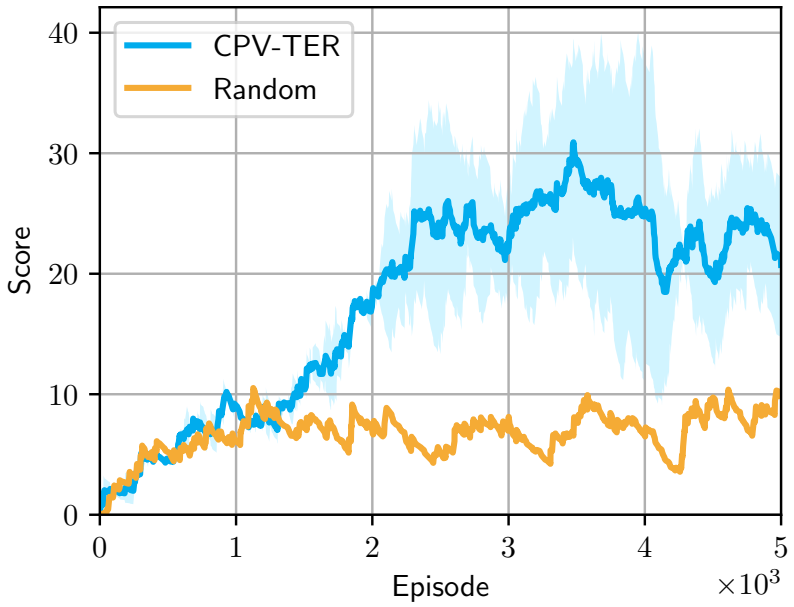


Figure 4.11: Five-task environment. Score archived with five possible tasks provided as a goal, over a training period of 5×10^3 episodes.

1×10^3 episodes, reducing the task completion time by approximately $25\,\%$. The current benchmark for multi-task learning is MT-10, which consists of ten tasks for an agent to complete. While this is only five tasks, we believe the sequential nature of the tasks in the crafting environment adds to the complexity enough to consider them to be similar benchmarks.

4.5.1 Potential Limitations

Despite the promising outcomes and contributions highlighted in the experimentation, the multi-task reinforcement learning methodology also exhibits certain limitations and potential challenges that warrant consideration.

One primary limitation is the sensitivity of the proposed CPV-TER method to the environment structure and complexity. While the approach demonstrates notable efficiency in handling sequential tasks within the crafting-world environment, its performance

might vary significantly in environments with different characteristics or varying levels of task interdependency. The observed success in crafting-world tasks might not generalize seamlessly to more diverse and complex environments, which could limit the broader applicability of the proposed method.

Furthermore, the reliance on a specific formulation of task compositionality might introduce biases or limitations in addressing tasks that deviate significantly from the assumed structure. The method's efficacy heavily relies on the availability of structured and hierarchical tasks, potentially restricting its adaptability to more chaotic or less organized task landscapes.

Another noteworthy limitation pertains to the trade-off between exploration and exploitation in the reinforcement learning process. The utilization of an epsilon-greedy policy for exploration introduces a delicate balance that could influence the agent's ability to explore the state space thoroughly versus exploiting known information. The exploration policy's efficacy might vary across different environments, impacting the agent's learning efficiency and generalization.

Additionally, the method's reliance on trajectory experience replay for storing and reusing prior experiences as self-produced expert demonstrations could lead to challenges related to sample efficiency. While this approach allows for leveraging previously encountered trajectories, it might not efficiently generalize to novel or unseen trajectories, potentially hindering the agent's ability to adapt to new tasks or environments effectively.

Moreover, the computational overhead or resource requirements for implementing CPVTER, albeit lower compared to some existing benchmarks, might still pose challenges for adoption in resource-constrained settings or less equipped academic environments.

Lastly, while the crafting-world environment provides a valuable benchmark for sequential tasks in the multi-task reinforcement learning space, its suitability as a comprehensive and representative benchmark for all sequential task scenarios across dif-

ferent domains remains a subject of consideration. The environment's characteristics might not encapsulate the diverse range of complexities and dynamics present in real-world sequential task settings, limiting its representativeness.

These limitations underscore the need for further exploration, potential modifications, and robustness evaluations of the CPV-TER method across diverse environments and task structures. Addressing these limitations could enhance the method's adaptability, generalization capabilities, and applicability across a broader spectrum of multi-task learning scenarios. These issues are addressed in section 6.2.

4.6 Summary

Most environments have some level of inherent structure to them, where subtasks can be completed in a certain order to complete a larger goal. We showed that if we harness the underlying structure of an environment by optimizing for compositionality, then we can improve the training time of multi-task reinforcement learning. We did this by producing a methodology that utilises compositional plan vectors in a DQN style agent to achieve multiple tasks in sequence in one environment.

This addresses the third research question:

RQ3 When represented as vectors, how can reinforcement learning be used to construct workflows?

This results in the second contribution:

C2 *This chapter demonstrates a new method to construct multi-step tasks with reinforcement learning. This method produces hierarchical solutions to tasks without expert demonstrations. This answers* **RQ3**.

The multi-task reinforcement learning methodology presented in this work introduces a novel algorithm for efficient multi-task learning using a composable inductive bias. This bias is distinct from other types previously considered, such as the hierarchical inductive bias used in hierarchical reinforcement learning. We believe this particular approach has the potential to develop into an active subfield of reinforcement learning research, in much the same way that hierarchical reinforcement learning has. We did this by creating a DQN-styled agent with a neural network that produces compositional plan vectors using a plan-vector sub-network.

This addresses the second research question:

RQ2 How can reinforcement learning be used to produce vector representations of workflows?

resulting in the third contribution:

C3 *This method also produces embeddings which can be used to represent workflows in a vector representation. These vectors can then be used to perform analysis of the distributed system that contains the workflows. this answers* **RQ2**.

We further demonstrated that if we store prior experiences in a manner that allows for reuse as self-produced expert demonstrations, then we can optimize for semantic similarity of tasks, further improving efficiency of training. We did this by producing a replay buffer that stores entire trajectories allowing for the use of imitation learning loss functions in addition to standard reinforcement learning losses.

This addresses the fifth research question:

RQ5 What methods can be used to increase learning efficiency with less labelled data?

This results in our fourth contribution:

C4 *This chapter also provides a new replay method that allows for more efficient learning with data in reinforcement learning. By using self produced data as expert demonstration imitation learning techniques can be used on top of reinforcement learning techniques. This answers* **RQ5**.

Additionally, we released our adaptation of the crafting world environment in an OpenAI gym format that allows for use of this environment as a performance benchmark for sequential tasks. This environment also requires less compute power than robotics or gaming benchmarks, which encompass the majority of benchmarks for RL.

C5 *In this chapter we also released a benchmark that involves tasks of a sequential nature and additionally requires minimal compute power in comparison to other benchmark environments in the MTRL space. This is a new focus area of green AI and aims to make the academic field both greener and more inclusive for academics with minimal access to additional compute power [109].* This is not a direct result of one of the main research questions but is nonetheless an important contribution to the reinforcement learning space.

Chapter 5

Deep Geometric Learning for Directed Acyclic Graphs

5.1 Overview

Graph generation is a quickly growing area of study with applications in a wide range of problem domains, such as drug discovery and task scheduling problems [133, 87]. Most existing graph generation methods use some form of supervised or semi-supervised learning requiring large amounts of training data. For example, [18] use a data set of 133,885 molecules as a prior distribution for graph generation. However, in certain application fields, such as distributed systems composition, prior examples are either sparse or nonexistent. We aim to create a method that can generate graphs with no prior data.

A few recent works have used reinforcement learning to generate undirected graphs where these methods require less training data or supervision. These works employ generative adversarial networks alongside proximate policy optimization [133]. Q-learning has also been implemented for graph construction, however it has been implemented with tabular methods and is therefore unusable at scale due to the exponential size of the state space that needs to be explored[87].

We propose a novel deep Q-learning approach to construct directed acyclic graphs (DAGs). Deep Q-learning is a model-free reinforcement learning algorithm that is

known to perform well with large action and state spaces that require function approximation. In this implementation, we combine Q-learning with a feed-forward graph convolutional neural network where actions correspond to the addition of a set of nodes and edges. This method can account for large scale directed DAGs, with multiple node types, and potentially continuously valued node features.

The rest of this chapter is organized as follows. In section 5.2, we discuss various properties of workflows in distributed systems, and the benefits of representing those workflows in Directed Acyclic Graphs. In section 5.3, we discuss the benefits of using Q-learning as an RL approach for DAG construction. In Section 5.4, we provide an overview of the DAG generation environment and the characteristics of the DAGs relevant to the rest of this chapter. Section 5.5 describes the architecture of the graph neural network that is used to map state action pairs to Q-values. Section 5.6 describes how the Q-learning algorithm is applied. Section 5.7 provides preliminary results, before the conclusion in Section 5.8.

5.2 Workflows Represented as Directed Acyclic Graphs

In this section we discuss various properties of workflows in distributed systems, and the benefits of representing those workflows in Directed Acyclic Graphs (DAGs).

A Directed Acyclic Graph (DAG) is a pair $\mathcal{G} = (V, E)$, where V is the set of vertices (sometimes referred to as nodes) and E is the set of edges connecting those vertices. The set of edges E consist of ordered pairs to account for directionality, e.g. the pair (X, Y) denotes an edge from node X to node Y, which is distinct from (Y, X), which is an edge from node Y to node X. A graph is acyclic when it contains no cycles, which is a path from a node back to itself. For example, a graph $\mathcal{G} = (V, E)$, where $V = \{A, B, C, D\}$ and $E = \{(A, B), (A, D), (B, C), (C, A)\}$ is not acyclic, as it contains the path $A \rightarrow B \rightarrow C \rightarrow A$ [82]. These sets of edges can be considered partially ordered sets. DAGs are commonly employed to depict workflows in distrib-

uted systems. They establish node connections that help define intricate systems and illustrate a sequence of events, their likelihoods, and interactions among them. The direction within these connections signifies the flow of data, dependencies between data elements, or simply the order in which tasks should be executed.

In multiprocessor systems, tasks are divided into several subtasks for parallel execution. DAGs are used to represent data dependencies and communication time between tasks. Proper representation of tasks is necessary for the efficient execution of scheduling algorithms.

DAGs can also be used to represent code in compiler design, describing the inputs and outputs of each arithmetic operation performed within the code. This representation allows the compiler to perform common subexpression elimination efficiently.

Directed Acyclic Graphs (DAGs) are a powerful tool for representing causal relationships and encoding assumptions about the relationships between variables. The first advantage of DAGs over cyclic graphs is that they simplify task scheduling. In cyclic graphs, a task may depend on itself or its downstream tasks, making it difficult to determine the proper order of task execution. This problem is particularly challenging in parallel computing environments, where multiple threads or processes can execute tasks concurrently. In contrast, DAGs guarantee that each task is executed only once and in the correct order, simplifying the scheduling process.

The second advantage of DAGs is that they enable efficient parallelization. In DAGs, tasks with no dependencies can be executed concurrently, enabling efficient use of computing resources. In contrast, cyclic graphs may require the execution of tasks in a specific order, limiting the amount of parallelism that can be achieved.

The third advantage of DAGs is that they support incremental execution and fault tolerance. DAGs enable the re-execution of individual tasks that fail or need to be recomputed due to changes in input data. By preserving the partial results of executed tasks, DAGs support incremental execution, reducing the time required to complete

workflows. Additionally, DAGs can tolerate failures of individual tasks or computing resources by re-executing failed tasks or switching to alternative computing resources.

Overall, using DAGs to represent distributed system workflows can help improve reliability and predict anomalous behaviors in complicated distributed systems.

5.3 Q-Learning as a Method for DAG Construction

One of the key advantages of Q-learning is its ability to handle the exploration-exploitation tradeoff. Exploration refers to the agent's ability to try out new actions and observe their outcomes, which is critical for finding optimal policies. Exploitation refers to the agent's ability to choose actions that have produced promising outcomes in the past, which is critical for maximizing rewards. Q-learning achieves this balance by using an ϵ-greedy policy, where the agent takes the action with the highest Q-value with probability $(1 - \epsilon)$ and a random action with probability ϵ. This approach allows the agent to explore new actions while also exploiting its current knowledge.

Another benefit of Q-learning is its ability to handle non-linear and non-differentiable reward functions. RL tasks often involve complex and non-linear reward functions, such as in the game of Atari. Q-learning does not require any assumptions about the reward function and can handle both continuous and discrete state and action spaces. Moreover, Q-learning does not require any gradient computations, making it a suitable method for non-differentiable reward functions.

Q-learning is also computationally efficient and scales well to large scale RL problems. The update rule in Q-learning is simple and can be easily implemented in parallel. This makes it suitable for online learning scenarios where data is collected iteratively. In addition, Q-learning can be extended to deep RL, where the Q-value function is approximated using deep neural networks. This approach, known as deep Q-learning, has achieved state-of-the-art performance in a range of tasks, including Atari games and robotic control.

Q-learning strikes a balance between exploration and exploitation, can handle non-linear and non-differentiable reward functions, and is computationally efficient. Its versatility has made it a popular method for a wide range of RL tasks, and its extensions to deep RL have pushed the boundaries of what is possible in this field.

5.4 Problem Definition

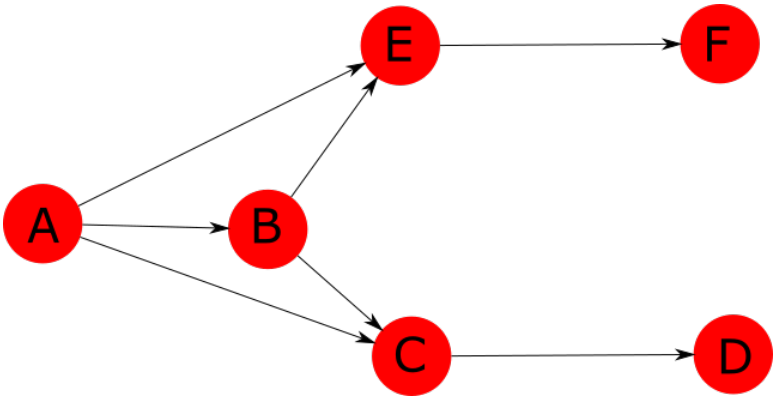
In this section we formulate the problem of DAG construction as a reinforcement learning problem. The goal of our learning agent is to construct a directed graph $\mathcal{G} = (V, E)$, where each node is one of b node types. The reinforcement learning problem is posed as an agent environment structure, where an agent interacts with an environment and receives numerical rewards. Representation of states, actions and rewards are defined in matrices.

5.4.1 Matrix Representation of DAGs

There are two main data structures for DAGs in computer science, adjacency lists and adjacency matrices. For the purposes of this method, adjacency matrices are the representation of choice, the reasons for which are explained in this section. First we define both representations, then explain the benefits and drawbacks of each.

Adjacency Lists

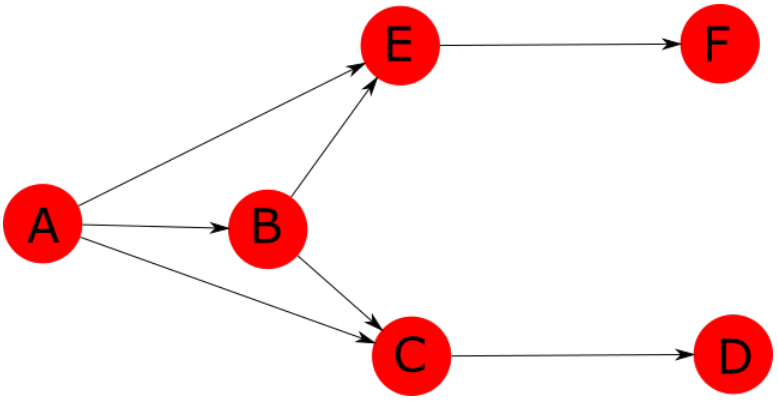
An adjacency list is a list of all nodes and lists containing the nodes each node is connected to. Each vertex of the graph is represented by a node of the list, and the edges are represented as links between the nodes. An example DAG and adjacency list is shown below:




```
graph = {'A': ['B', 'C', 'E'], 'B': ['C', 'E'], 'C': ['D'],
         'D': [], 'E': ['F'], 'F': []}
```

Adjacency lists use less memory than adjacency matrices. Space complexity of an adjacency list: $O(|V|+|E|)$. They are also more efficient than adjacency matrices when adding or deleting edges, taking $O(1)$ time for these operations. However, finding the weight of an edge in an adjacency list takes $O(d)$ time, where d is the degree of the vertex. Traversing an adjacency list to visit all the vertices of the graph takes $O(|V| + |E|)$ time.

Adjacency Matrices



$$\text{graph} = \begin{vmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

An adjacency matrix represents a graph as a 2D array of size $|V| \times |V|$. If there is an outgoing edge between vertices i and j, then the value at cell (i, j) is 1, else it is 0.

The space complexity of an adjacency matrix is $O(|V|^2)$. It excels in finding the weight of an edge ($O(1)$ time) and traversing all vertices ($O(|V|^2)$ time). However, adjacency matrices use more memory than adjacency lists and are unsuitable for sparse graphs.

Pros and Cons

Both adjacency lists and adjacency matrices have their advantages and disadvantages. The key points for this methodology are as follows:

Adjacency Lists

- Use less memory ($O(|V| + |E|)$ space complexity).

- Efficient for adding or deleting edges ($O(1)$ time).
- Suitable for sparse graphs.
- Inefficient in finding edge weights ($O(d)$ time) and traversing all vertices ($O(|V|+|E|)$ time).

Adjacency Matrices

- Higher memory usage ($O(|V|^2)$ space complexity).
- Efficient in finding edge weights ($O(1)$ time) and traversing all vertices ($O(|V|^2)$ time).
- Unsuitable for sparse graphs.
- Easier to pass to a neural network due to fixed dimensions

The choice of data structure depends on the specific requirements of the application. If memory is a concern, or the graph is sparse, an adjacency list is a better choice. If the graph is dense or finding edge weights is a frequent operation, an adjacency matrix is more suitable. In this case, while memory is always an important concern, the graphs for this method are not sparse, so an adjacency matrix is a better choice for DAG generation.

5.4.2 State

The state of the environment at time t in a given episode corresponds to a DAG and is denoted $\mathcal{G}_t$. The topology of $\mathcal{G}_t$ is represented using a binary adjacency matrix $\boldsymbol{A}$ where $\boldsymbol{A}[i, j] = 1$ indicates that a directed edge exists from the node i to the node j. Note that a 1 at the location $[i, j]$ in $\boldsymbol{A}^\top$ indicates that a directed edge exists from the node j to the node i.

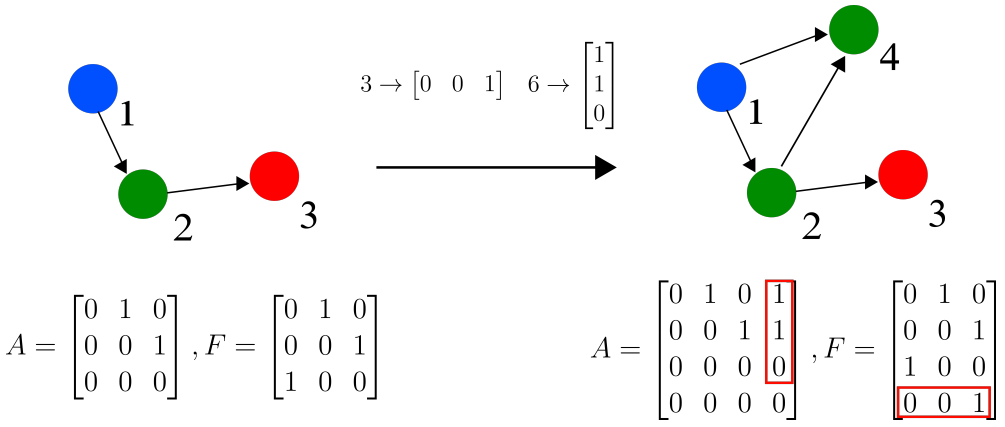


Figure 5.1: An example of a single action in the DAG generation environment, along with the state descriptions before and after the action is taken. Here, the 4^{th} node is added, of type 3, with incoming edges from nodes 1 and 2. This action is described further in Section 5.4.3.

Individual node types are represented using a feature matrix $\mathbf{F} \in \{0,1\}^{n \times b}$ containing a one hot encoding of the node types for each node, where b is the number of possible node types and n is the number of nodes in $\mathcal{G}_t$. The initial state for each episode is the null graph, which corresponds to no services being called within a distributed system, i.e. the workflow has not yet begun.

5.4.3 Action

In this environment, each action consists of two parts: adding a new node, and adding the set of incoming edges to that node. The new node n_t can be any of the b node types within the environment. This node type is added to the matrix of node features $\boldsymbol{F}$ by appending the one hot encoding of this feature as an additional row. In Figure 5.1, a node of type 3 (represented as a green node) is added to the DAG, so a row of $[0, 0, 1]$ is added to the feature matrix of the DAG $\mathcal{G}_t$.

Next, the set of incoming edges N_k^{in} for the new node is added. For example, in Figure 5.1, the 4^{th} node has incoming edges from nodes 1 and 2. By only allowing for

the addition of incoming edges, we create a topologically sorted graph. Topological sorting for DAGs is a linear ordering of nodes such that for every directed edge A B, node A comes before node B in the ordering. Topological sorting is only possible if the graph has no directed cycles. This ensures that all graphs created are acyclic - if outgoing edges were added, this would allow for the edges to be added from the current node to preceding nodes, potentially creating a closed path making the graph cyclic. This is in effect the reverse of Khan's algorithm, which is a topological sorting method for DAGs. Khan's algorithm is based on the observation that any DAG must have at least one node that has no incoming edges. This is because if there were no such nodes, then every node in the DAG would be part of a cycle. The algorithm works by repeatedly finding nodes with no incoming edges, adding them to the sorted list, and removing them from the graph along with their outgoing edges. This process continues until all nodes have been added to the sorted list. [53]

In order to select the set of incoming edges, all feasible extensions are encoded as a binary vector, where 1 on the n^{th} position of the vector represents an incoming edge from the n^{th} node to the newly created node. In Figure 5.1, the possible vectors would range between 001, connecting only the 3^{rd} and 4^{th} nodes, and 111, connecting all previous nodes to the 4^{th} node. These vectors can then be converted from their binary format to an integer $i \in [1, 2^{n-1}]$. This allows for enumeration of all feasible extensions when required to determine the action with the maximum Q-value. This also allows for selection of a random extension where performing exploration (as necessary in ϵ-greedy Q-learning).

In Figure 5.1, the 4^{th} node is added, so the possible set of edges is in the range $i \in [1, 7]$. In the figure, $i = 6$, which converted to binary format is 110, so edges are added from the first and second nodes. This binary number is appended to the adjacency matrix of $\mathcal{G}$ as a new column. Note that the null set of edges is excluded as a possible choice here: this is in order to prevent floating nodes, although this doesn't affect the theory of the method significantly.

This method of adding a new node and multiple edges in a single action is in contrast to many graph generation methods which add a single node or edge per action [133]. This new method has the disadvantage of creating a much larger action space of $b \times (2^{n-1} - 1)$ possible actions, as opposed to $b + n - 1$ possible actions with a single node or edge addition. However, this method reaches a final DAG in fewer timesteps and accounts for symmetries in edge additions: when adding edges one at a time, a policy could develop which favours a particular order of adding edges to the set. As order of edge additions does not affect the final structure of the DAG, this asymmetrical policy is prevented, which is why adding an entire set of edges is preferable overall (see figure 5.3).

5.4.4 Reward

Reward design is an important feature in reinforcement learning. in this section we discuss the principles of reward design in reinforcement learning and then discuss reward structure and reasoning behind the choices made.

Reward Design for Reinforcement Learning

Reward design for implementation of this method depends largely on the application's problem domain. For scheduling problems, for instance, the reward could be a numerical value based on the speed and quality of service of the resulting task schedule. These rewards can consist of both intermediate rewards and a singular reward once a generation episode terminates.

Reward Structure for DAG generation

For the purposes of evaluating the proposed graph generation method, in this work we performed simulations in which a positive reward was returned if the agent produced a

DAG isomorphic to a ground truth DAG, and a reward of 0 otherwise. This is clearly an extreme case, as even for a 10 node graph with only 1 node type, there would be $1.018\text{e}{+13}$ possible final states, of which approximately 1-5 produce a non-zero reward. However, in order to prove the generality of this method the isomorphic reward case was used for the results in this chapter, and tested on smaller graphs. Potential work for extension to larger graphs is discussed in Section 6.2.

5.4.5 Environment Description and Justification

The environment employed in this chapter serves as a platform for investigating the generation of Directed Acyclic Graphs (DAGs) through reinforcement learning paradigms. Each episode within this environment unfolds as the agent sequentially constructs a DAG, denoted as $\mathcal{G}_t$, by iteratively adding nodes and their respective incoming edges. The state representation of $\mathcal{G}_t$ is encapsulated by a binary adjacency matrix $\boldsymbol{A}$, encoding the directed edges between nodes. Complementing this, a feature matrix $\mathbf{F} \in 0, 1^{n \times b}$ captures individual node types via one-hot encoding, facilitating a comprehensive depiction of the evolving DAG's topology.

The environment's action space is characterized by a dual-step action: first, the addition of a new node along with its associated node type encoded in the feature matrix $\mathbf{F}$, and subsequently, the inclusion of incoming edges to the newly added node. By structuring actions in this manner, the environment ensures the creation of topologically sorted graphs, adhering to Directed Acyclic Graph constraints. This action design aligns with the essence of topological sorting algorithms, preventing cyclic structures and fostering the creation of acyclic graphs essential for various real-world applications, such as workflow scheduling and dependency modeling.

The choice of this environment over conventional graph generation paradigms stems from its unique action space design, which enables simultaneous addition of nodes and multiple edges in a single action. This departure from the norm in graph generation

methods allows for expedited convergence towards the final DAG within fewer time steps. Despite enlarging the action space substantially, this approach mitigates biases inherent in sequential addition methods, effectively nullifying asymmetrical policies favoring specific edge addition sequences, thus enhancing the model's generality and efficiency.

Moreover, the reward structure adopted for evaluating this environment emphasizes the production of a DAG isomorphic to a predefined ground truth DAG. This deliberate choice of reward, while resulting in a vast state space, serves as a rigorous evaluation metric, accentuating the method's ability to generate precise graph structures. The extremity of this reward scenario, where only a fraction of possible states yields positive rewards, challenges the model to optimize its decisions towards the exact replication of the target DAG, thereby showcasing the robustness and adaptability of the proposed method.

The selection of this environment is justified by its unique attributes aligning with our research focus on generating precise DAG structures through RL. Its distinct action space design, coupled with the reward structure emphasizing isomorphism to a target DAG, provides a rigorous platform to test and validate the efficacy of our proposed approach. The environment's alignment with real-world scenarios requiring acyclic graph structures and its capacity to challenge the model in reproducing specific graph patterns reinforce its significance in advancing graph generation methodologies within the realm of reinforcement learning.

5.5 Model

The Q function is a mapping from a state action pair to a real value indicating the predicted future reward for the pair in question. We learn this mapping using a policy network, which takes the state of the environment and outputs a single scalar value which can be treated as the Q-value. Our policy network is in effect a feed-forward

graph convolutional network, similar to graphSAGE [37]. A major difference here is this network accounts for direction of the edges, in a manner similar to that of the struc2vec++ method [112].

The network will take the adjacency matrix and one hot encoded features of a given state $\mathcal{G}_t$ as the input. The feed-forward architecture consists of two convolutional layers, followed by a non-linearity, then a pooling layer. The network propagates this one hot representation through two graph convolutional layers, each consisting of two steps. The first step is a concatenation of the feature representation of that node, the sum of features of outgoing neighbours of that node, and the sum of features of incoming neighbours of that node. Incoming and outgoing neighbour feature sums are found by performing matrix multiplication with the one hot feature representation and the adjacency matrix containing in-degrees, and the adjacency matrix containing out-degrees (which is simply the transpose here), respectively. The second step of the convolutional layer is to pass the concatenation through a simple non-linearity.

After the two convolutional layers, the representation is passed through a linear layer and is then passed to a pooling layer, which aggregates the individual node representations into a graph representation using a sum function. A sum aggregator is chosen here as opposed to a mean or max aggregator as it preserves both the ratio of node types, unlike the max aggregator, as well as differentiating between different scales of graphs, unlike the mean aggregator [130]. This representation is then passed through a final linear layer and ReLU unit, and then a fully connected linear layer, producing a single scalar which is used as the Q-value for the input state, as discussed further in section 5.6.

Formally, this network is described as follows. Given a DAG $\mathcal{G} = (V, E)$ with adjacency matrix $\boldsymbol{A}$ containing in degrees and $\boldsymbol{A}^\top$ containing out degrees. Each node v is initially represented by a vector $\boldsymbol{h}_v^0$ which is a one hot encoding of the node type. Node v has a set of neighbouring nodes N_k^{in} connected with incoming edges, and a set of neighbouring nodes N_k^{out} connected with outgoing edges. In every convolutional

layer l the representations of the current node, and sums of incoming and outgoing neighbour node representations $\boldsymbol{h}_v^l$, $\boldsymbol{h}_w^l$, $\boldsymbol{h}_u^l$ are horizontally concatenated:

$$\mathbf{H}^{l+1} = \text{CONCAT}(\mathbf{H}^l, \boldsymbol{A}\mathbf{H}^l, \boldsymbol{A}^\top \mathbf{H}^l) \tag{5.1}$$

This concatenated representation is then passed through a non-linearity (eq. 5.2) where $\mathbf{W}$ and $\mathbf{B}$ are trainable matrices.

$$\mathbf{H}^{l+1} = \text{ReLU}(\boldsymbol{H}^l \mathbf{W}^l + \mathbf{B}^l) \tag{5.2}$$

After two convolutions with this method, the node representations are pooled as in eq. 5.3, resulting in a scalar value.

$$\mathbf{H}^{l+1} = \text{SUM}(\boldsymbol{H}^l \mathbf{W}^l + \mathbf{B}^l) \tag{5.3}$$

This value is passed through a final non-linearity and then a fully connected linear layer to produce the final result.

5.6 Learning and Inference

This method generates training data for the policy network using an ϵ-greedy implementation of the DQN method. With probability ϵ a random action is selected from the range of possible actions, otherwise the approximate Q_{max} value is selected. This Q_{max} value is found using a sample based method - instead of evaluating each possible action, only a fixed number of random actions are evaluated, and the action from these with the highest value is chosen. This is because the action space is in the order of $b \times 2^{n-1}$, so evaluating each possible action would be too costly. After performing this action, the current Q-value for the state is evaluated using the policy network. Additionally, the Q-value is calculated by adding the reward to the Q-value of following the optimal policy thereafter, as shown in eq. 5.4.

$$Q^\pi(S_t, A_t) = r + \gamma \max_a Q(S_{t+1}, a) \tag{5.4}$$

Note that $\max_a Q(S_{t+1}, a)$ in eq. 5.4 is calculated using a separate target network, which is not trained every step, but instead has network weights from the policy network copied over after a fixed number of episodes.

Taking the difference of these Q-values produces our training error $\delta = Q(S_t, A_t) - Q^{\pi}(S_t, A_t)$, and the loss function is then simply the L^2 norm $\mathcal{L} = ||\delta||$. This is back-propagated through the network, and the Q-values produced by the network for each state are then closer to the true Q-values. This update occurs online, or after every time step, as opposed to in batches. While producing noisier results, this tends to result in a higher convergence rate [55]. After training is complete, a greedy method is used to generate the DAGs, whereby the action with the largest Q-value is selected at each step.

A second learning agent utilizing prioritized experience replay [105] was also implemented; this agent updated on the current timestep and additionally on 30 timesteps that have previously been selected that are held in a memory buffer. The actions replayed are selected at random, with a higher probability for selection for actions with high rewards, and actions more recently added to the memory buffer.

5.7 Results

Figure 5.2 shows selected results of an implementation of this method. All data is averaged over 20 runs and training takes place over 10,000 episodes. For each run, a new “ground truth" DAG is created for the given graph size and number of types, chosen at random. This is in order to prevent any bias from the selection of the graph. Figure 5.3 shows that a binary action selection results in a faster convergence rate, because it accounts for symmetries in the addition of edges.

Changes in total accumulated reward due to increasing the scale and node types of the DAGs is shown in Table 5.1 — this decrease in accumulated reward is to be expected due to the exponentially-increasing size of the action space. For a DAG of only 6

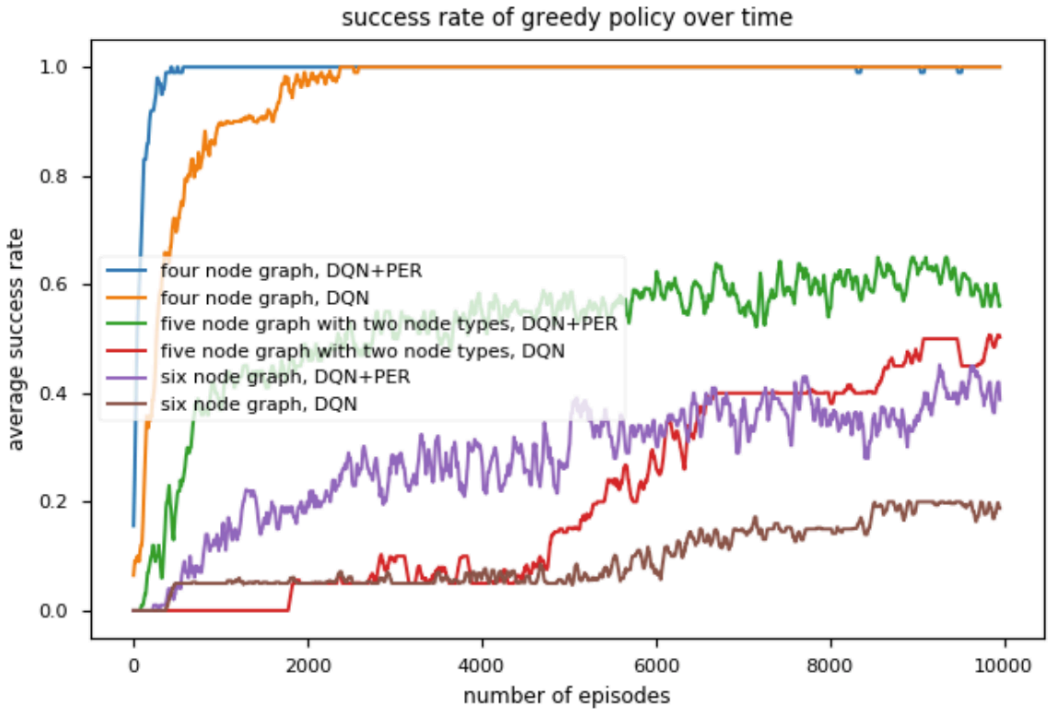


Figure 5.2: The average success rate of the greedy policy over time, for DAGs of different sizes and different numbers of node types. For each graph type, a learning agent trains using an ϵ-greedy policy over 10,000 episodes. The agent then runs an episode on the greedy policy after each ϵ-greedy episode. Each learning agent is reset and trained 20 times, and the success rate of the greedy policy is the average of these 20 runs. For readability, the moving average over 50 episodes is plotted. Two types of agents are displayed: the standard DQN, and DQN with prioritized experience replay (DQN+PER).

nodes with only a single node type, there are 9,765 possible terminating states, of which approximately 5 are isomorphic to the 'true' DAG that the agent is attempting to learn. Only these 5 states provide any non-zero reward, creating a highly sparse reward environment. For a DAG with 7 nodes, there are 615,195 possible graphs, again with the same amount of true graphs that provide any reward. It is clear then that learning over a span of 10,000 episodes is unlikely to learn anything, as it is highly unlikely any of these episodes will return any positive reward. The exploding nature of

Table 5.1: A comparison of average total reward for DAGs of varying size and number of unique node types, with basic DQN, DQN with prioritised experience replay, and an agent that selects actions at random. The rewards are totaled over 10,000 attempts, and averaged over 20 runs.

	DAG Isomorphisms				
DAG Sizes	4 nodes	5 nodes			6 nodes
Node Types	1	1	2	3	1
Random	882	75	6	2	0
DQN	9524	5242	2083	1374	955
DQN+PER	9902	7380	5169	796	2802

the method shows a preliminary result that the method can work for overall structures, i.e. each node could be used to represent a substructure. However, the data shows that this method could not be used as is for a reward structure this minimal for methods much larger. However, given a proper example scenario one would most likely have a broader reward scheme than pure isomorphism, allowing for larger values. This is a good avenue for further research and is elaborated on further in the discussion and future work sections.

Contrast this to the work of [18], who were able to generate molecular structures of up to 9 nodes with 5 node types. however, these results are not directly comparable, as their method involves using a dataset of 133,885 compounds in order to produce a prior distribution. Our method is exploring graph generation under the context that there is little to no previous examples or data to use. Additionally, our method has a highly sparse reward space that relies entirely on achieving an exact isomorphic reproduction of the ground truth. [18] instead designs reward on chemical properties which creates a more populated reward space.

In assessing the practical applications of deep reinforcement learning for DAG generation, it is instructive to look at areas where directed graphs are commonly used.

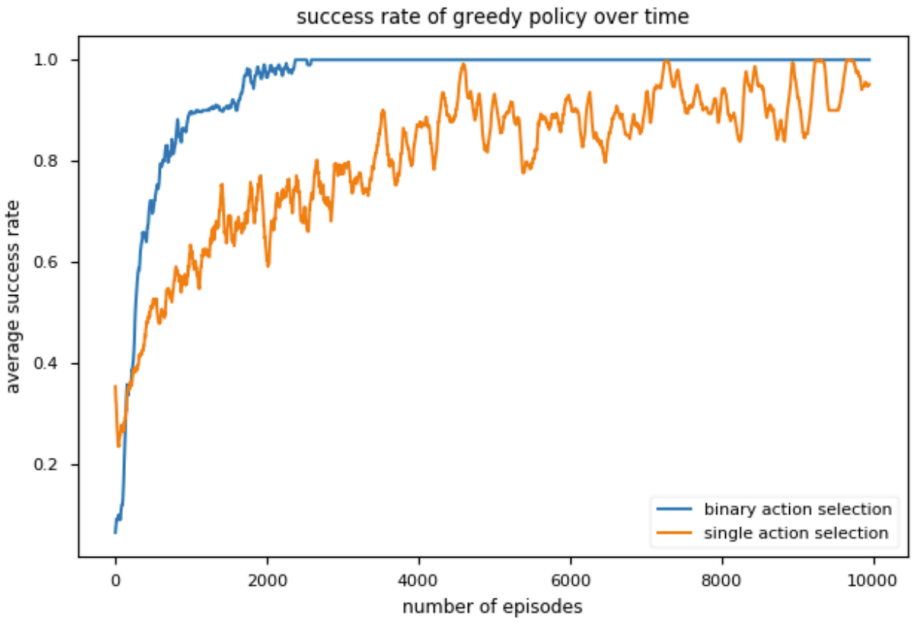


Figure 5.3: A comparison of learning rate for the standard DQN with a binary action selection, and an action selection that involves adding only a single node or edge per timestep.

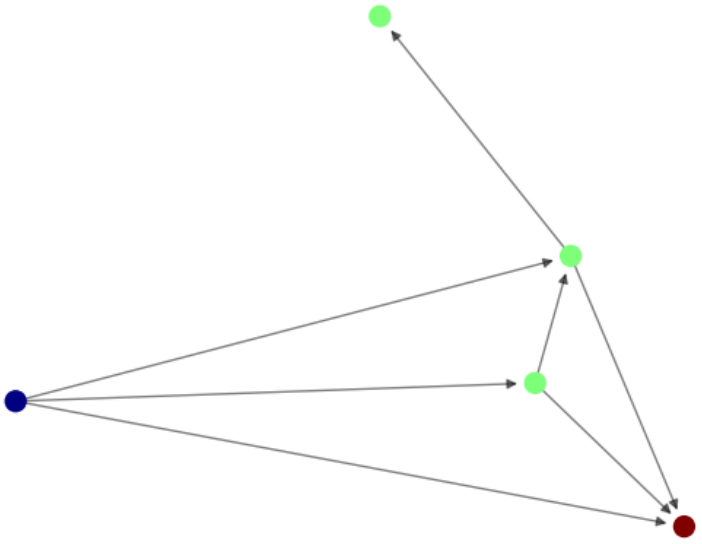


Figure 5.4: An example DAG with 5 nodes of 3 types. For the DAG to be considered isomorphic the nodes of each type need to be connected in the same manner. For this DAG there are 25,515 possible terminating states, of which only 3 return a non-zero reward.

Some sectors that could benefit from this approach include telecommunication networks, power grid networks, and supply chain networks [58]. For example, in a telecommunications network, the nodes could be stations, while the edges across stations could represent potential communication links. The objective of graph generation in this context might be to design a network topology that optimizes for cost efficiency, coverage, and robustness. A deep Q-learning based approach can use this criterion as a reward function, learn from historical topologies, and explore new topologies to improve network design.

Deep reinforcement learning is poised to have significant impacts in these sectors, among others, mainly because of the advantages it offers over other methods used in graph generation. GraphRNN and GraphAF, for example, are deep generative models for graph generation. GraphRNN uses a hierarchical method to generate graphs sequentially, while GraphAF uses an autoregressive flow-based model. Both these methods have demonstrated effectiveness in generating diverse and complex graph structures. However, unlike deep reinforcement learning and specifically Q-learning, they heavily rely on large-scale training data to sufficiently capture graph structures and lack the ability to explore novel graphs independent of prior data. Thus, the advantages of our approach over these models lie in their capacity to explore and exploit the space of possible graphs even with no prior data, making it particularly suitable for tasks where data is scarce or non-existent.

Evolutionary algorithms and Bayesian optimization are popular choices in hyperparameter tuning and optimization problems, including graph generation. These methods perform a sequential exploration-exploitation tradeoff strategy and can handle large search spaces and non-convexity. However, both methods usually require time-consuming process and large computational resources, especially when the dimension of the search space is large. Moreover, they do not readily provide a way to incorporate prior knowledge into the search process, unlike deep Q-learning methods that naturally integrate this through the Q-value function.

5.7.1 Potential Limitations

One of the primary limitations of the introduced method is associated with the scalability and computational complexity stemming from the exponential growth of the action space. As the size of the DAGs increases, the number of possible configurations escalates exponentially, resulting in a vastly expanded action space. This exponential explosion considerably challenges the ability of reinforcement learning algorithms to efficiently explore and learn optimal policies, leading to diminished learning rates and increased sparsity in reward signals.

Moreover, the sparsity of the reward space poses a significant challenge in the learning process. The reward function primarily relies on achieving an exact isomorphic reproduction of the 'ground truth' DAG, resulting in an extremely sparse reward landscape. In scenarios with larger DAGs or increased node types, the probability of encountering rewarding states decreases substantially. This sparsity can impede the learning process, causing difficulties in convergence and hindering the agent's ability to generalize well to unseen or larger DAGs.

The method's dependency on a 'ground truth' DAG for each run introduces potential bias due to the randomness in generating these ground truth graphs. Variability in the ground truth graphs across different runs might impact the learning process, causing variations in performance and affecting the generalizability of the trained model.

Additionally, the comparison to existing methods such as MolGAN, which leverages prior distributions from a dataset of compounds, highlights a key limitation. While MolGAN demonstrates success in generating molecular structures, it does so using a dataset, providing a richer reward landscape based on chemical properties. In contrast, the proposed method operates in a scenario with little to no prior examples or data, resulting in a highly sparse and specific reward space focused solely on isomorphic reproduction.

The limitation of not exploring larger DAGs due to computational constraints also

restricts the method's practical applicability to scenarios demanding complex or larger-scale DAG generation. Real-world applications often involve workflows or systems represented by considerably larger graphs, rendering the current method impractical for such scenarios due to computational limitations.

Furthermore, the reliance on a simple graph convolutional network based on spatial approaches might limit the model's expressiveness and capacity to capture complex graph structures. More sophisticated graph neural network architectures might be required to effectively capture intricate relationships and patterns within larger and more diverse DAGs.

These limitations underscore the need for further exploration into scalable approaches, enhanced reward structures, and more sophisticated neural network architectures to address the challenges of scalability, sparsity in rewards, and model expressiveness. Overcoming these limitations could potentially broaden the method's applicability and efficacy in generating directed acyclic graphs across diverse domains and larger scales. These issues are discussed further in section 6.2.

5.8 Summary

This chapter introduced a deep Q-learning method for directed acyclic graph generation. By using a simple graph convolutional network based on spatial approaches, we can produce Q-values for various DAG states, with the transpose of the adjacency matrix used to account for edge directions. Only smaller graphs were tested in this implementation, due to the exponential size of the action space and the increased sparsity of the rewards.

This chapter addresses the fourth research question:

RQ4 When treated as graphs, how can reinforcement learning be used to construct workflows?

C6 This chapter demonstrates a new method to construct directed acyclic graphs using geometric deep learning combined with reinforcement learning. This demonstrates an ability to produce workflows with minimal data outside of workflow application success. This answers **RQ4**.

Chapter 6

Conclusion

This chapter encompasses an overview of the contributions made in each previous chapter and their relevance to the research questions initially presented in Chapter One. Additionally, we explore the questions that have emerged from the analysis conducted in Chapters Three, Four, and Five, acknowledging them as potential future research avenues with corresponding insights. Our final task is to present our overall observations in order to bring the thesis to a close.

6.1 Research Questions and Contributions

In chapter One we presented our hypothesis: *the use of reinforcement learning in combination with other deep learning techniques can be used to construct and representing workflows. These workflows can be used for applications within large scale distributed systems, and their representations can be analysed to obtain further useful information.*

This hypothesis motivated our investigation of reinforcement learning and representation learning methods for the construction of workflows. It is useful to recall the five research questions **(RQ1-RQ5)** originally introduced in chapter One, presented as a basis for exploration:

RQ1 What are the best representations for encoding workflows for later analysis of distributed systems and to use as a platform for automatically constructing further

workflows?

RQ2 How can reinforcement learning be used to produce vector representations of workflows?

RQ3 When represented as vectors, how can reinforcement learning be used to construct workflows?

RQ4 When treated as graphs, how can reinforcement learning be used to construct workflows?

RQ5 What methods can be used to increase learning efficiency with less labelled data?

This section examines our approach for answering **RQ1** - **RQ5**, while outlining the degree to which they were accomplished. Additionally, we draw attention to the noteworthy findings in every chapter and specify any contributions that they provide.

RQ1 is addressed first in chapter Three, where we examine different methods of embedding techniques for knowledge graphs. We analysed two key methods types for graph representation, energy-based methods (Section 3.2.1) and random-walk methods (Section 3.2.2). Energy-based embedding methods aim to minimize an energy function that maps the graph vertices into a low-dimensional space while preserving pairwise distances. Random-walk-based embedding methods generate a random walk on the graph and use the resultant probability distribution to represent the graph structure in a low-dimensional space. Energy-based methods focus on pairwise distances between vertices, while random-walk-based methods focus on global graph structure captured by the random walk distribution. We ran our experimental analysis by producing 100-dimensional embeddings of the DAIS-ITA Science Library dataset. We produced these embeddings using Trans E and Trans R, which are energy based methods,

and Node2Vec, which is a random-walk method. We then compare selected vectors from each embedding with a logistic regression model of the dataset. The F_1 score allows us to statistically compare the node classification abilities of each method when given an embedding vector as input.

We found that the Trans-E method had the best classification ability with this dataset, Trans-R the worst, and Node2Vec functioning in between these two methods. We also examined Trans E and Node2Vec by visualising the embeddings using a t-distributed neighbour embedding technique. These allowed us to determine that while certain energy-based embedding methods allowed for the most accurate classification, random walk methods can still classify to a moderate degree while also leveraging homophily and structural similarity. As a result of this examination, chapter three provides *insight into the benefits of various graph embedding techniques that can be used to represent workflows. These embedding techniques can be used with semantic vector spaces to provide better analysis of workflows in distributed systems.* This supports research question **RQ1** and represents contribution **C1**.

In Chapter Four, we produce a method that uses reinforcement learning to transform a range of workflows into a vector space **RQ2** in the style of compositional plan vectors (Section 4.2.1). The policy network architecture of the CPV method contains a sub-network called the Plan-vector network, which intakes concatenations of the initial state of the environment and the goal state of then environment, as well as a concatenation of the initial state and the current state of the environment, and produces 256-dimensional vector embeddings called 'plan vectors'. These vectors can represent workflows of tasks within a given system, and are inherently composable; the vectors can be added to each other to create new workflows that describe completing the tasks described by both individual plan vectors.

This method simultaneously uses the vector representations to construct further workflows within the space of the distributed system **RQ3**. The plan vectors are used as inputs for a Deep Q-Network which produces values for each possible subsequent action,

which informs a learning agent to perform subsequent tasks within a desired workflow. By priming the network with these plan vectors, the learning agent can learn similar tasks (and thus workflows) with increasing speed and accuracy.

Further, we produced a replay method (**RQ5**) that assists in efficiently reusing the produced data to create a more efficient learning method. Most replay methods for reinforcement learning take singular successful timesteps and replay them to the learning agent to increase the amount of useful data in the system. We store full trajectories and treat them as expert trajectories, as is done in imitation learning. These trajectories can be used with behavioral cloning loss functions to increase the learning rate.

Chapter Four contains four main contributions:

C2: *A new method to construct multi-step tasks with reinforcement learning. This method produces hierarchical solutions to tasks without expert demonstrations.*

C3: *This method also produces embeddings which can be used to represent workflows in a vector representation. These vectors can then be used to perform analysis of the distributed system that contains the workflows.*

C4: *Provides a new replay method that allows for more efficient learning with data in reinforcement learning. By using self produced data as expert demonstration imitation learning techniques can be used on top of reinforcement learning techniques.*

C5: *We also released a benchmark that involves tasks of a sequential nature and additionally requires minimal compute power in comparison to other benchmark environments in the MTRL space.*

Chapter Five uses a graph representation to produce workflows with reinforcement learning (**RQ4**). In chapter Five, we treat workflows within a system as directed acyclic graphs, and produce these graphs using a deep Q network. By adding all incoming

edges to a node in one action, represented as a binary action, we find we can decrease the size of the action space, thus increasing the learning efficiency. This chapter provides the final contribution of the thesis:

C6: *This chapter demonstrates a new method to construct directed acyclic graphs using geometric deep learning combined with reinforcement learning. This demonstrates an ability to produce workflows with minimal data outside of workflow application success.*

6.2 Future Work

In this section we discuss possible future research that has been motivated by work conducted over the course of this thesis.

Future Direction 1 - *Analysis of representation methods with labelled data sets.*

First-and-foremost, in order to provide a truly representative and quantitative evaluation of similarity matching, it is necessary to have labelled data of nodes/events/fragments that are deemed similar by a human subject-matter expert. While semantic vector embedding itself is unsupervised, its evaluation requires labelled data. In the absence of this, however, we can proceed with proxy evaluations such as node classification and link prediction as described in Section 3.3.

An avenue for future work is to incorporate work by Summers-Stay et al. on natural language semantic vector spaces to enrich knowledge base queries and responses [113]. This would require ground truth in the form of example queries and expected responses for the ontology of interest.

Another important step would be to carry out a thorough fine-tuned analysis of all available models on a rich knowledge graph dataset, ideally a well known example

knowledge graph, but with consideration of data preprocessing to determine if different graph structures yield better performance with different approaches. As a separate—but related—task these could be wrapped in the experimental user interface that we built to rapidly explore different models and graphs in our experimental work so far.

For knowledge graphs that are comprised of disconnected fragments (e.g., a series of disconnected events or reports) it is likely that kernel methods such as Subgraph2Vec and Graph2Vec may be more appropriate than the individual node/edge-based approaches that have been highlighted in this chapter. A further study based on just these methods could be conducted with a suitable knowledge graph.

Another promising direction is to explore time-respecting random walks for specific phenomenological ontologies. This would require the knowledge graph to have edges mapped to timestamps to create a continuous-time dynamic network on which the performance of temporal link prediction could be assessed.

The capability for online learning may be very valuable with regards to graphs based on observational ontologies, as these are constantly and dynamically updated in real-time. Therefore, an inductive approach—such as GraphSAGE—could be highly advantageous. This technique learns embeddings that generalize to previously unseen nodes. Research to explore heterogeneous extensions to these models that better capture ontology-specific semantics is highly recommended.

Finally, it could be useful to incorporate a notion of trust into graph databases based on observational data. This would make use of DAIS-ITA research by Barclay et al. on decentralized identifiers (DIDs) and verifiable credentials (VCs) to ratify claims and qualities [5]. Potentially nodes and edges from observers and organizations could be mapped to some scalar-level of trust derived from cryptographically signed VCs and knowledge of DIDs of a group or network of trusted peers, in a similar way to continuous-time models (see Section 3.2.2). This would mean that when an event is observed and ratified by an increasing number of trusted observers, or by observers who are deemed to be more trustworthy, it accumulates a higher trust level. Random-

walk-based methods that act on such a graph could be biased towards edges with a higher-level of trust, causing embedded representations of the graph to be give preference to more highly trusted observations.

Future Direction 2 - *Test the CPV-TER method with an enterprise framework for increased testing.* Testing of this method using an enterprise RL framework will increase testing speed and ability. While RLLib and others were examined, the minimal customisation ability in the trajectory rollout functions of these frameworks makes implementation difficult.

Future Direction 3 - *Investigate methods to expand this method for different graph features including edge weightings and node features. Additionally further optimization of the RL agent can increase learning efficiency.*

For future work, a graph network that accounts for edge weights and continuous node features needs to be developed.

A method that utilizes hierarchical reinforcement learning could significantly increase the convergence rate for DAGs at scale by learning structures of subgraphs found in generated graphs.

Further improvements to the base DQN could also be implemented, in particular multi-step return calculations and dueling Q-networks as discussed in [42].

Finally an application to a specific problem domain would allow for a more detailed reward design, and a more accurate test of this method.

6.3 Final Remarks

The motivation behind this thesis was to explore the use of machine learning techniques, particularly reinforcement learning, for the construction and representation of

workflows in large-scale distributed systems. As technology continues to advance, the complexity and evolution of these workflows are increasing every day. We need automated and adaptable solutions to keep up with these changes and maintain optimal performance, while also reducing operational costs.

Through a series of experiments and research, this thesis has demonstrated the efficacy of reinforcement learning for automating the construction of workflows. The use of graph and vector-based representation learning techniques has also been explored, resulting in better analysis of workflows within distributed systems. Additionally, new replay methods for efficient learning with data and novel methods for constructing directed acyclic graphs using geometric deep learning combined with reinforcement learning have been proposed.

The contributions of this thesis provide valuable insights into the benefits of using reinforcement learning and representation learning for constructing and analyzing workflows in distributed systems. The use of these techniques has the potential to greatly improve system performance and reduce operational costs by automating decision-making processes, optimizing workflows, and reducing the need for manual interventions.

The primary contributions of this thesis lie in the successful exploration and application of reinforcement learning techniques to automate the construction of workflows in distributed systems. Specifically, the investigation into graph and vector-based representation learning has illuminated a path toward enhanced workflow analysis within complex distributed environments. By devising novel methods like the Compositional Plan Vectors (CPV) approach and employing reinforcement learning for constructing directed acyclic graphs (DAGs), this thesis establishes a framework for efficient workflow automation and representation. These advancements represent not only technological breakthroughs but also practical solutions that could significantly impact industries relying on efficient workflow management.

However, this thesis does acknowledge certain limitations that offer avenues for future

exploration. One notable limitation pertains to the scalability of the developed methods when handling vast and diverse datasets commonly encountered in real-world scenarios. Extending the research to address challenges related to large-scale distributed systems and handling complex workflows in such environments represents a critical next step. Additionally, while the introduced methods show promise, further refinement is needed to account for intricate graph features, edge weightings, and continuous node attributes in the construction and representation of workflows. Furthermore, exploring hierarchical reinforcement learning techniques and enhancing the base Deep Q-Network (DQN) models could substantially improve convergence rates and learning efficiency for directed acyclic graphs (DAGs) at scale.

Among the most significant achievements of this thesis is the successful demonstration of reinforcement learning's potential in automating workflow construction, offering a glimpse into the future of optimized decision-making in distributed systems. The proposed methods for graph representation and vector embedding have showcased the transformative impact on workflow analysis, setting the stage for streamlined operations and cost reduction in industrial settings. Notably, the innovative CPV approach and the utilization of geometric deep learning combined with reinforcement learning to construct DAGs stand out as pioneering methodologies with far-reaching implications for workflow automation. These achievements underscore the potential for machine learning to revolutionize workflow management in complex distributed systems.

In conclusion, this thesis has made significant contributions to the field of machine learning for distributed systems. It has demonstrated the potential of using reinforcement learning and representation learning for the construction and analysis of workflows, and provided new methods for achieving these goals. There is still much work to be done in this area, but the findings of this thesis provide a solid foundation for future research and development.

Bibliography

[1] Prithviraj Ammanabrolu and Mark O. Riedl. Playing text-adventure games with graph-based deep reinforcement learning, 2018.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017.

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb 2017.

[4] Chen Bao, Helin Xu, Yuzhe Qin, and Xiaolong Wang. Dexart: Benchmarking generalizable dexterous manipulation with articulated objects, 2023.

[5] Iain Barclay, Swapna Radha, Alun Preece, Ian Taylor, and Jarek Nabrzyski. Certifying provenance of scientific datasets with self-sovereign identity and verifiable credentials. *arXiv preprint arXiv:2004.02796*, 2020.

[6] Jonathan Baxter. A model of inductive bias learning. *CoRR*, abs/1106.0245, 2011.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253279, Jun 2013.

[8] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[9] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[10] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.

[11] Dave Braines, Jane Stockdill-Mander, and Eunjin Lee. The science library: Curation and visualization of a science gateway repository. *Concurrency and Computation: Practice and Experience*, 33(19):e6100, 2021.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2013.

[14] Felipe Codevilla, Matthias Muller, Antonio Lopez, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[15] Cédric Colas, Pierre Fournier, Olivier Sigaud, Mohamed Chetouani, and Pierre-Yves Oudeyer. Curious: Intrinsically motivated modular multi-goal reinforcement learning, 2018.

[16] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs, 2017.

[17] Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

[18] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv e-prints*, page arXiv:1805.11973, May 2018.

[19] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[20] Coline Devin, Daniel Geng, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Compositional plan vectors. 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.

[21] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[22] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv:1604.06778 [cs]*, Apr 2016. arXiv: 1604.06778.

[23] Maxim Egorov. Multi-agent deep reinforcement learning. page 8.

[24] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

[25] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay, 2020.

[26] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. *CoRR*, abs/2007.06700, 2020.

[27] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[28] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[29] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents, 2017.

[30] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. May 2016.

[31] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv:1709.04326 [cs]*, Sep 2017. arXiv: 1709.04326.

[32] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[33] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[34] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[35] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Aug 2019.

[36] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[37] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[38] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*, pages 139–144, 2018.

[39] Richard A Harshman and Margaret E Lundy. Parafac: Parallel factor analysis. *Computational Statistics & Data Analysis*, 18(1):39–72, 1994.

[40] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[41] Peter Henderson, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. Benchmark environments for multitask learning in continuous domains, 2017.

[42] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

[43] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:37963803, Jul 2019.

[44] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016.

[45] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018.

[46] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50:21:1–21:35, 2017.

[47] T. Jaakkola, Michael I. Jordan, and Satinder Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[48] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks, 2016.

[49] Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. *Conference on Robot Learning (CoRL)*, 2018.

[50] Rodolphe Jenatton, Nicolas Le Roux, Antoine Bordes, and Guillaume Obozinski. A latent factor model for highly multi-relational data. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 3176–3184, 2012.

[51] Jiechuan Jiang, Chen Dun, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *CoRR*, abs/1810.09202, 2018.

[52] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2017.

[53] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558562, nov 1962.

[54] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

[55] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[56] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[57] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[58] Edward Elson Kosasih and Alexandra Brintrup. A machine learning approach for predicting hidden links in supply chain with graph neural networks. *International Journal of Production Research*, 60(17):5380–5393, 2022.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[60] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, 2016.

[61] M. Pawan Kumar, Ben Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, 2010.

[62] David L. Leottau, Javier Ruiz-del Solar, and Robert Babuka. Decentralized reinforcement learning of robot behaviors. *Artificial Intelligence*, 256:130159, Mar 2018.

[63] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system, 2019.

[64] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight, 2017.

[65] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019.

[66] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[67] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[68] Junfan Lin, Zhongzhan Huang, Keze Wang, Xiaodan Liang, Weiwei Chen, and Liang Lin. Continuous transition: Improving sample efficiency for continuous control problems via mixup. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021.

[69] Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[70] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[71] Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices, 2020.

[72] Ziru Liu, Jiejie Tian, Qingpeng Cai, Xiangyu Zhao, Jingtong Gao, Shuchang Liu, Dayou Chen, Tonghao He, Dong Zheng, Peng Jiang, and Kun Gai. Multi-task recommendations with reinforcement learning. *Proceedings of the ACM Web Conference 2023*, Apr 2023.

[73] Sephora Madjiheurem and Laura Toni. Representation learning on graphs: A reinforcement learning application, 2019.

[74] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 543–548, 2006.

[75] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[76] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[78] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[79] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018.

[80] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.

[81] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[82] Richard E. Neapolitan and Xia Jiang. Chapter 4 - learning bayesian networks. In Richard E. Neapolitan and Xia Jiang, editors, *Probabilistic Methods for Financial and Marketing Informatics*, pages 111–175. Morgan Kaufmann, Burlington, 2007.

[83] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976, 2018.

[84] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018.

[85] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[86] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280, 2012.

[87] Alexandru Iulian Orhean, Florin Pop, and Ioan Raicu. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *J. Parallel Distrib. Comput.*, 117:292–302, 2018.

[88] Emilio Parisotto, Jimmy Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2016.

[89] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

[90] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2014.

[91] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[92] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments, 2019.

[93] Kristina Preuer, Günter Klambauer, Friedrich Rippmann, Sepp Hochreiter, and Thomas Unterthiner. Interpretable deep learning in drug discovery, 2019.

[94] Sebastien Racaniere, Andrew K. Lampinen, Adam Santoro, David P. Reichert, Vlad Firoiu, and Timothy P. Lillicrap. Automated curricula through setter-solver interactions, 2019.

[95] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables, 2019.

[96] Swati Rallapalli, Liang Ma, Mudhakar Srivatsa, Ananthram Swami, Heesung Kwon, Graham Bent, and Christopher Simpkin. Sense: Semantically enhanced node sequence embedding. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 665–670. IEEE, 2019.

[97] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. struc2vec. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 17*, 2017.

[98] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2010.

[99] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[100] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

[101] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019.

[102] Ryan Sander, Wilko Schwarting, Tim Seyde, Igor Gilitschenski, Sertac Karaman, and Daniela Rus. Neighborhood mixup experience replay: Local convex interpolation for improved sample efficiency in continuous control tasks, 2022.

[103] Stefano Savazzi, Vittorio Rampa, Sanaz Kianoush, and Mehdi Bennis. On the energy and communication efficiency tradeoffs in federated and multi-task learning. *2022 IEEE 33rd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep 2022.

[104] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David 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 2015. PMLR.

[105] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015.

[106] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[107] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.

[108] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[109] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):5463, Nov 2020.

[110] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 650–658, 2008.

[111] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning, 2021.

[112] Niklas Steenfatt, Giannis Nikolentzos, Michalis Vazirgiannis, and Qiang Zhao. Learning structural node representations on directed graphs. In Luca Maria Aiello, Chantal Cherifi, Hocine Cherifi, Renaud Lambiotte, Pietro Lió, and Luis M. Rocha, editors, *Complex Networks and Their Applications VII*, pages 132–144, Cham, 2019. Springer International Publishing.

[113] Douglas Summers-Stay, Clare Voss, and Taylor Cassidy. Using a distributional semantic vector space with a knowledge base for reasoning in uncertain conditions. *Biologically Inspired Cognitive Architectures*, 16:34–44, 2016.

[114] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2:160–163, 1990.

[115] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[116] Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *International Conference on Machine Learning*, 2008.

[117] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[118] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

[119] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning, 2017.

[120] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[121] Maarten van Steen and Andrew S. Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, Oct 2016.

[122] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[123] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017.

[124] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019.

[125] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[126] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay, 2016.

[127] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[128] Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*, 2013.

[129] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[130] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[131] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[132] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *CoRR*, abs/1806.08804, 2018.

[133] Jiaxuan You, Bowen Liu, Rex Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *CoRR*, abs/1806.02473, 2018.

[134] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2019.

[135] Seongjun Yun, Minbyul Jeong, Sungdong Yoo, Seunghun Lee, Sean S. Yi, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks: Learning meta-path graphs to improve gnns, 2021.

[136] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. Experience replay optimization. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Aug 2019.

[137] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Metagraph2vec: Complex semantic path augmented heterogeneous network embedding. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 196–208. Springer, 2018.

[138] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *CoRR*, abs/1812.04202, 2018.

[139] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020.

[140] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2018.

[141] Wenxuan Zhou and David Held. Learning to grasp the ungraspable with emergent extrinsic dexterity, 2022.

[142] Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504, 2019.