Multi-Agent Continual Deep Reinforcement Learning with Shared Experiences

Project Dissertation

by Saptarshi Nath

Student ID: F120942

Loughborough University

21COP324: AI Project (MSc Artificial Intelligence)

Program of Study: Artificial Intelligence

Supervisor: Dr Andrea Soltoggio

Submitted on 26^{th} Aug 2022

Abstract

The latest developments in deep learning show impressive results that often surpass human capabilities. These have also unlocked enhanced capabilities in the reinforcement learning setting, where the integration of deep learning techniques has allowed RL models to beat human competitors in many complex task environments. However, these systems do not possess the versatility and longevity possible by the human brain. Continual learning techniques enable existing methodologies by addressing these issues, though, agents are still limited in their ability to learn quickly.

This project looks to evaluate a system in which a number of continual deep reinforcement learning (CDRL) agents can learn continuously learn experiences over their lifetime, and share these experiences with each other to reduce the amount of training required and achieve target performance faster. The project demonstrates the performance benefits of such a system over a single agent, across the CT-Graph and MiniGrid reinforcement learning environments.

Acknowledgements I would like to thank my supervisor Dr Andrea Soltoggio and Ese Ben-Iwhiwhu for their time and support throughout this project, as well as their valuable guidance. Furthermore, I want to thank my family and friends for their support throughout my studies.

Contents

Contents 4

List of Figures 6

List of Tables 9

1 Introduction 10
- 1.1 Aims and Objectives . . . 11
- 1.2 Original Contributions . . . 12

2 Literature Review 14
- 2.1 Reinforcement Learning . . . 14
- 2.2 Multi-Agent Reinforcement Learning . . . 20
- 2.3 Continual Learning . . . 21
- 2.4 Continual Learning Desiderata . . . 22
- 2.5 Continual learning methodologies . . . 23
- 2.6 Regularisation Approaches . . . 23
- 2.7 Replay/Rehearsal Approaches . . . 25
- 2.8 Architectural Approaches . . . 26
- 2.9 Hybrid Approaches . . . 28
- 2.10 Benchmarks and Metrics . . . 29
- 2.11 Continual learning applications . . . 30

3 ShELL System Overview 31
- 3.1 System Behaviour . . . 31
- 3.2 Agent Architecture . . . 33
- 3.3 Non-Overlapping Task Curricula . . . 34

4 Software and Hardware Specifications 35
- 4.1 Project packages and dependencies . . . 35
- 4.2 Hardware specifications . . . 38

5 Experiment Methodology 41
- 5.1 CT-Graph Experiments . . . 41
- 5.2 MiniGrid Experiments . . . 42

5.3 Metrics 42
5.4 Hypothesis and Initial Experiment Results 48

6 Experiment Results **49**
6.1 CT-Graph Results 49
6.2 MiniGrid Results 56

7 Conclusions and Future Work **61**
7.1 Conclusions 61
7.2 Project Evaluation 61
7.3 Future Areas of Interest and Lessons Learned 62

Bibliography **63**

List of Figures

1 The agent-environment interaction loop in reinforcement learning. For each decision made for an action to be performed by the agent, a state change takes place in the environment. This results in a new reward and a new state and the cycle continues until the goal state is reached. In the end an optimal solution is produced. 17

2 Illustrates a brief tree diagram of popular RL algorithm classifications. 18

3 Existing popular continual learning methodologies and their classifications. 24

4 Reinforcement learning loop of the ShELL system. To achieve the agent-environment interaction, the OpenAI Gym API is used to handle the environment agent communication. This returns the state and reward from the environment to the agent for any action performed by the agent policy. Each agent has a task curriculum. Based on this curriculum the agent's action is fed to the environment. The environment returns a new state-reward pair for that task, given the agent's action. I.e., Agent 0 will feed an action to the environment for task 0, which will in turn return a new state-reward. Agent 0 will then do so for task 1 then task n. Agent 1 and agent n will behave similarly with their own unique task curriculum. . 32

5 Illustrates the process of sharing task masks from an agent to another agent when the same task is encountered. This nullifies the need to train from scratch and maintains performance. 33

6 Illustrations of CT-graph trees for branching factor 2 experiments. Left: CT-Graph with depth 2, branching factor 2. Resulting in 4-tasks. Right: CT-Graph with depth 3, branching factor 2. Resulting in 8-tasks. Bottom: CT-Graph with depth 3 and branching factor 3. This results in 27 tasks. . . . 43

7 Illustrates a visual representation of the MiniGrid environments. Top: Simple Crossing task variations (task 0, 1, 2). In these tasks, the agent must reach the green square. Walls pose as an obstruction to the agent's objective. Bottom: Lava Crossing task variations (tas. In these tasks, the agent must also reach the green square, however the internal walls have been changed to lava. Lava Crossing variation 3 is especially difficult to succeed. Images have been taken from Chevalier-Boisvert et al. (2018) 45
8 From left to right: Average, maximum and minimum rewards measured throughout training of (a) Agent 0 (b) Agent 1 of a two agent ShELL system on 4-task CT-Graph. Experiment task curriculum is as follows. Agent 0: 0, 1, 2, 3, Agent 1: 1, 2, 0, 3. Each integer indicates a task id that corresponds to a specific task end state in the CT-Graph tree. Refer to figure 6. 50
9 Plots of the instant cumulative reward (ICR) and total performance over time (TPOT) metrics for CT-Graph experiments. The top row consists of the ICR and TPOT of 4-task CT-Graph. The middle row consists of the ICR and TPOT of 8-tasks CT-Graph. The bottom row consists of the ICR and TPOT of 27 task CT-Graph. 52
10 Plots of the instant learning advantage (ILA) and total learning advantage (TLA) ShELL specific metrics. Top row: 4-task CT-Graph. Middle row: 8-tasks CT-Graph. Bottom row: 27-task CT-Graph 54
11 Time reduction advantage scores calculated at 25%, 50%, 75% and 100% of the max ICR possible in each experiment. From left to right: 2 agents, 3 agents, 10 agents. Top row: 4-task CT-Graph. Middle row: 8-task CT-Graph. Bottom row: 27-task CT-Graph. More agents are able to achieve a higher TRA score, sooner. 55

12 From left to right: Average, maximum and minimum rewards measured throughout training of Top: Agent 0, Middle: Agent 1, Bottom: Agent 2 of a three agent ShELL system on 3-task MiniGrid. Experiment task curriculum is as follows. Agent 0: 0, 1, 2, Agent 1: 1, 2, 0, Agent 2: 2, 0, 1. Each integer indicates a task id that corresponds to a specific task variation in the MiniGrid environment. Refer to figure 7. 57
13 Plots of the instant cumulative reward (ICR) and total performance over time (TPOT) metrics for MiniGrid experiments. The top row consists of the ICR and TPOT of MiniGrid Simple Crossing (3 tasks). The bottom row consists of the ICR and TPOT of MiniGrid Simple Crossing + Lava Crossing (6 tasks). 58
14 Plots of the instant learning advantage (ILA) and total learning advantage (TLA) ShELL specific metrics. Top row: 3 task MiniGrid. The bottom: 6 task MiniGrid. 59
15 Time reduction advantage scores visualised in bar plot format. Top row: 3 task MiniGrid. Bottom row: 6 task MiniGrid. More agents are able to achieve a higher TRA score, sooner. . 60

List of Tables

1 Versions of packages and modules used in the project. Includes a description of their use cases. 37
2 Versions of packages and modules used in the project. Includes a description of their use cases. 39
3 CT-graph experiment parameters. Each group of experiments have been run on ShELL systems of 2, 3 and 10 agents. 44
4 MiniGrid experiment hyper-parameters. The 3 task MiniGrid was run on ShELL systems of 2, 3 and 10 gents. The 6 task MiniGrid was run on 2, 3 and 6 agents. 45
5 ShELL system parameters. These parameters were used across all experiments to maintain consistency, with the exception of the max steps. This has been modified as required, to match the experiment parameters. 46

1 Introduction

The continual learning (CL) paradigm is the latest advancement in humanity's quest for achieving human-level capabilities in artificial intelligence. It expands on the learning capabilities that limit existing deep learning techniques, by allowing agents to continuously expand on their experiences and thereby allowing them to perform on many different tasks as opposed to just one (as is the case in the traditional deep learning paradigm). This limitation in existing neural approaches stems from the nature of the back-propagation algorithm which lies at the heart of the neural network. It is at the core of what enables them to be so effective at learning new information.

The human brain is and has been the unequivocal natural benchmark for artificial intelligence. It is capable of learning many varying tasks over its lifetime by forming new synaptic connections, a property that is known as neuroplasticity. This property prevents the brain from succumbing to the effects of catastrophic forgetting (This is explained further in section 2) and in some cases, the human brain can even exhibit the ability to selectively forget information that is no longer necessary to its performance or survival (i.e., graceful forgetting) (French, 1999; Wołczyk et al., 2021). Modern deep learning approaches to artificial intelligence are not capable of achieving this on their own and an abrupt degradation in model performance is often observed when these agents are put into a continuous learning environment. Continual learning approaches aim to build on these deep learning techniques by introducing another mechanism that controls how neural networks handle the process of learning, to reduce and even eliminate catastrophic forgetting.

While continual learning expands on the learning capabilities of neural networks, multi-agent deep reinforcement learning (MADRL) is a sub-field of reinforcement learning that focuses on studying how multiple agents can co-exist in a shared environment. An example of such studies includes Baker et al. (2019) which explores how multiple agents can collaborate to achieve a common goal. Works in this field tend to explore how independent agents can work together to achieve a common goal. However, could agents collaborate at a more fundamental i.e, at the learning level? By sharing the knowledge accumulated by each agent, a multi-agent system could reduce

the time required to learn and train across many different tasks and successfully perform across them all. Incorporating continual learning techniques would allow each agent to learn many tasks over a lifetime, improving the versatility of the agents and the overall system. Agents sharing task knowledge with each other would effectively result in a network of agents that can continuously and collaboratively learn without needing to be taken out of deployment and re-trained. There would be many limitations to such a system such as the storage and computational costs, which are even more apparent with continual learning approaches.

1.1 Aims and Objectives

The core aims of this project are to test the knowledge sharing abilities of the Shared Experience Lifelong Learning system (ShELL) and identify the benefits over traditional continual learning agents. ShELL is a multi-agent continual reinforcement learning system in which multiple agents can share the knowledge they learn for tasks. This enables agents to effectively learn a task and perform without having learn from scratch if the knowledge is already available. This has been further explained in section 3 in which an in-depth background of the ShELL system is provided.

This project will specifically look at the advantages and benefits of ShELL systems, using single continual learning systems as a baseline, using the Supermasks in Superposition algorithm in conjunction with Proximal Policy Optimisation. It is well understood in the existing literature that the supermasks in superposition algorithm (Wortsman et al., 2020) functions well in the supervised learning setting. The benefits of ShELL using SupSup+PPO will be assessed via several experiments that will be run on a selection of suitable MDP reinforcement learning environments. Randomised tasks will be constructed from each environment by changing the input distributions for each task variation. The results of these experiments will be evaluated via metrics specific to continual reinforcement learning and the ShELL system, which are detailed in 5.

Specific target performance levels will be attempted in these experiments. Given a single agent continual learner learning P tasks over T evaluation

steps in an ideal scenario where all tasks are successfully learned, the amount of time taken to learn each task would be T/P. In an ideal scenario of non-overlapping task curricula, an N agent ShELL system learning P distinct tasks at a given evaluation step t, the number of task experiences learned by the ShELL system should be NP. This means that the number of task experiences learned by ShELL would be expected to increase by a factor of N once the entire population of agents in the system have learned all tasks. Thus, the amount of time taken to learn each task would decrease from T/P for a single agent to $T/(NP)$ for a ShELL system. In other words it is expected that ShELL will be $T/(NP)$ times faster at achieved a certain amount of knowledge over the single agent. However, to account for the non-optimality of the task curriculum it is hypothesised that ShELL will be able to achieve a more modest $0.5N$. Additionally, it is hypothesised that the target for performance improvement of the ShELL system over a single agent learner could theoretically be $1.3N$ at any given evaluation step. Accounting for potential non-optimal task curricula, $1 + 0.5N$ is proposed as an alternative target.

1.2 Original Contributions

In this project, a total five unique experiments have been conducted to evaluate the performance benefits of ShELL, a multi-agent system employing CDRL agents, compared to a single continual learner, as well as identify any unique behaviours in such a system. These experiments have been run on two different benchmark environments in the Configurable Tree Graph (CT-Graph) and MiniGrid. These environments are unique in that they provide computationally inexpensive MDP environments that are suitable for the assessment of specific continual learning metrics. The CT-Graph benchmark is not natively compatible with ShELL thus, to achieve this modifications were made specifically to integrate the CT-Graph benchmark by converting the output labels into a one-hot vectors which could then be used by the ShELL agents. To evaluate the maximum potential benefits of ShELL over a single continual learner, the task curriculum used in each of these experiments must be designed in such a way that agents do not observe overlapping tasks (Where possible. This is further discussed in later sections). To achieve this an algorithm was implemented to generate randomised non-overlapping

permutations of the base curriculum. The results of these experiments have been assessed using a combination of continual learning and ShELL specific metrics that are designed to identify the key learning advantages of ShELL over a single CDRL agent, as well as identify the benefits of CDRL agent over a traditional RL agent. These metrics have also been implemented into ShELL, so that the results logged during experiments can be used to compute the metrics.

2 Literature Review

This chapter provides a foundational understanding of the reinforcement learning setting, an overview of the state-of-the-art reinforcement learning algorithms, and how deep neural networks have been used to excel in reinforcement learning capabilities. It also introduces a taxonomy of the continual learning field and notable advancements in the literature. It will explore works done in multi-agent systems, and popular benchmarks and evaluation methodologies used to assess the performance of supervised reinforcement and continual learning approaches.

2.1 Reinforcement Learning

Reinforcement learning agents learn to navigate their environments given specific information about their position in those environments. This is achieved through an iterative process of observing states and rewards/penalties and responding with actions which result in new states-rewards. At each discrete time step t where $t = 0, 1, 2, 3, ..., n$, the agent observes a representation of the state of a given environment, $S_t \in \mathcal{S}$ where $\mathcal{S}$ is a set of all the possible states. For each observed state S_t, the agent performs an action $A_t \in \mathcal{A}$ where $\mathcal{A}$ is the set of all possible actions that can be taken in response to state S_t. This action A_t then results in a reward $R_{t+1} \in \mathcal{R}$. Over n time steps, an RL agent can learn to navigate through its environment to reach a desired goal state (Sutton and Barto, 1998). As illustrated in figure 1, this RL loop is often formulated as a Markov Decision Process (MDP).

The aforementioned loop can be formulated as a Markov Decision Process defined as $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition Probability Matrix which contains the probabilities of going from a state s to another state s' given an action a, written as equation 1,

$$\mathcal{P}^a_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{1}$$

and $\mathcal{R}$ is the reward function written as equation 2.

$$\mathcal{R}^a_s = \mathbb{E}[\mathcal{R} | \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{2}$$

The above definitions establish that the reward is dependant on any action given a state transition determined by the probability transition matrix. Each state transition in an MDP satisfies the Markov Property. This states that each state transition from S_t to S' is independent of past transitions, given the present. This can be written as equation 3.

$$\mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s] = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_1 = s_1, ..., \mathcal{S}_t = s_t] \tag{3}$$

An RL agent can successfully solve an MDP using an optimal policy π^*. The policy defines a probability distribution of all the actions $a \in \mathcal{A}$ to take for each state $s \in \mathcal{S}$. This can be written in the form of equation 4,

$$\pi(a_t|s_t) = \mathbb{P}[A_t = a_t | S_t = s_t] \tag{4}$$

where $\pi(a_t|s_t)$ is the probability of an action a_t in response to a state s_t at time step t.

Not every state in an MDP is of equal value to the agent. Some states may be more beneficial to achieving the goal state than others. Therefore, a state-value function is used to learn the value of each state. Given an agent is following a policy π, the value of a state s is denoted by $\mathcal{V}_\pi(s)$. The 'value' is the expected return which can be defined as $\mathcal{V}_\pi(s) : s_t \rightarrow s_T$ where s_T is the terminal state of the MDP. The state-value function can be written as equation 5,

$$\mathcal{V}_\pi(S) \doteq \mathbb{E}_\pi \left[\mathcal{G}_t | \mathcal{S}_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s\right] \text{ , for all } s \in \mathcal{S} \tag{5}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected return from a starting state s to the next state s' and so on, given the agent follows policy π. γ is the discount rate.

Now that the value of a state is known, the value of performing an action a given a state s using policy π. The state-action value function (Q-function) can be defined as the,

$$\mathcal{Q}_\pi(s, a) \doteq \mathbb{E}_\pi \left[\mathcal{G}_t | \mathcal{S}_t = s, \mathcal{A}_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s, \mathcal{A}_t = a\right] \tag{6}$$

where the $\mathcal{Q}_\pi(s,a)$ is the expected return starting at state s, performing an action a using policy π.

Using the policy, state and action-value functions, an RL agent can manoeuvre through an MDP. To find the optimal solution, however, the optimal policy and state-value functions must be found. As the agent trains and gains experience, the policy will update to reflect the newly learned information. This will in turn effect the state and action-value function. To find the optimal functions the Bellman equation is introduced. The Bellman equation can be defined as (Bellman, 1954),

$$\mathcal{V}_\pi(S) \doteq \mathbb{E}_\pi \left[\mathcal{R}_{t+1} + \gamma \mathcal{V}_\pi(\mathcal{S}_{t+1}) | \mathcal{S}_t = s\right] = \mathcal{R}_t + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}, \mathcal{V}_\pi(s') \tag{7}$$

where by introducing π this becomes the Bellman Expectation equation. Based on this equation, it becomes apparent that the value of a state can be decomposed in to the immediate reward $\mathcal{R}_{t+1}$ and the discounted value of the next state $\gamma\mathcal{V}_\pi(\mathcal{S}_{t+1}$, given the policy π. Similarly the state-action value function can also be decomposed.

Thus, the optimal state-value and state-action value functions can be defined as,

$$\mathcal{V}^*(s) = \max_\pi \mathcal{V}_\pi(s) \tag{8}$$

$$\mathcal{Q}^*(s,a) = \max_\pi \mathcal{Q}_\pi(s,a) \tag{9}$$

By knowing the optimal state-value and state-action value functions, the optimal policy function is found by picking the actions that give the most optimal state-action value function. For an optimal policy π^*, the condition $\pi > \pi'$ if $\mathcal{V}_\pi(s) \geq \mathcal{V}_{\pi'}(s), \forall s$. Thus, the optimal policy corresponds to,

$$\pi^*(a,s) = arg \max_{a \in \mathcal{A}} \mathcal{Q}^*(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}, \mathcal{V}^*(s') \tag{10}$$

In the case the environment was to change, a traditional RL system would struggle to perform due to the inability to adapt. It would be inherently

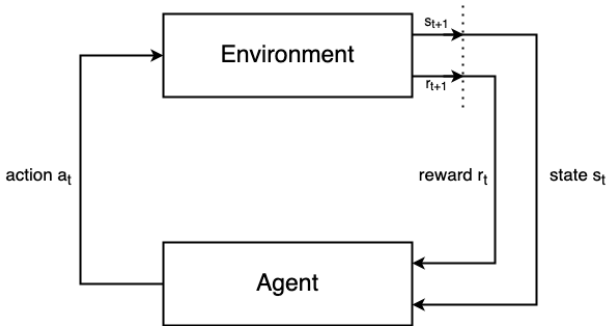

Figure 1: The agent-environment interaction loop in reinforcement learning. For each decision made for an action to be performed by the agent, a state change takes place in the environment. This results in a new reward and a new state and the cycle continues until the goal state is reached. In the end an optimal solution is produced.

beneficial to combine continual learning methodologies with RL to improve the versatility and longevity of RL agents.

An RL agent decides how to respond to any given state-reward using a set of rules known as the policy. The agent attempts to optimise the reward policy which maps state-action pairs to reach the final goal state. This is achieved by exploring states and then exploiting particularly reward states to maximise reward to the goal state. There are many such policies available today such as the notable Proximal Policy Optimisation (PPO) (Schulman et al., 2017), Deep Q-Networks (DQN) (Mnih et al., 2013), and Advantage actor Critic (A2C) (Mnih et al., 2016). These systems have generally been able to demonstrate performance that far exceeds human capabilities using benchmarks such as the Atari2600 suite of games. Later works such as Silver et al. (2016) Silver et al. (2017) and OpenAI et al. (2019) exhibited impressive results in various board games and complex game environments. A taxonomy of notable RL policies is shown in figure 2. As indicated by the taxonomy, these policies can be categorised as Model-Free and Model-Based policies. The main difference between these two categories is that Model-Based approaches utilise a map of the state transitions and rewards of a given environment. This is known as the world model. Once the world model is known, an RL agent can produce an optimal trajectory. Examples

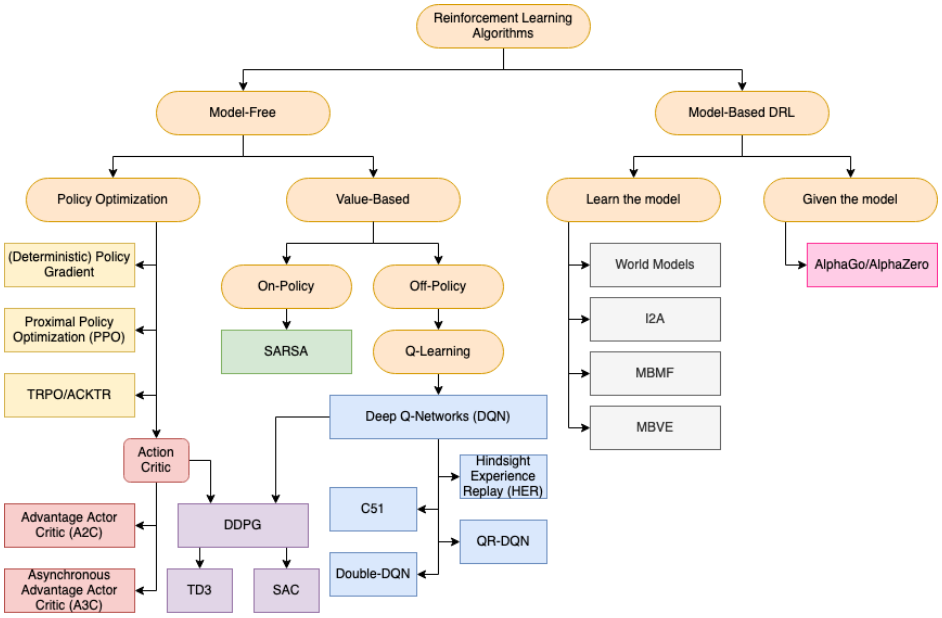

Figure 2: Illustrates a brief tree diagram of popular RL algorithm classifications.

of model-based works include World Models (Ha and Schmidhuber, 2018), Imagination-Augmented Agents (I2A) (Weber et al., 2017), Model-Based Priors for Model-Free Reinforcement Learning (MBMF) (Bansal et al., 2017), Model-Based Value Expansion (MBVE) (Feinberg et al., 2018), and AlphaZero (Silver et al., 2017).

On the other hand, model-free approaches have no prior knowledge of the environment to work with. Instead, these approaches iteratively learn about the environment by observing state-reward pairs. Model-free approaches can be further categorised as policy-based and value-based. Policy-based approaches learn the stochastic policy function, which is done on-policy. This means that each new improvement to the policy is made from actions performed based on the current policy's actions. Value-based approaches learn the policy by choosing the best action for a given state in the environment. Value-based approaches are both off-policy and on-policy. State-action-reward-state-action (SARSA) for example is an on-policy approach. Q-learning is an example of an off-policy approach, meaning actions are based

on knowledge from all stages of training rather than just the current policy.

Model-based approaches benefit from the increased efficiency in sampling (state-action pairs), however, the practicality of these approaches in unsupervised, real-world cases is limited. In reality, the ground truth world model may not always be available. In this case, prior knowledge may be used to train the agent which may lead to overfitting. Some notable model-based approaches include World Models (Ha and Schmidhuber, 2018), Imagination-Augmented Agents (I2A) (Weber et al., 2017), Model-Based Priors for Model-Free Reinforcement Learning (MBMF) (Bansal et al., 2017), Model-Based Value Expansion (MBVE) (Feinberg et al., 2018), and the aforementioned AlphaZero. Model-free approaches excel in their practicality as they do not rely on an internal model to maximise reward. These methods can also have advantages in more complex environments when constructing a sufficiently accurate world model can be challenging. Thus, many popular approaches fall under this category.

In the case of traditional RL learning about the environment is only useful given the RL agent is finding the optimal trajectory. Such an agent would not be useful when a new variation is introduced. In deep reinforcement learning (DRL) neural networks are introduced as a function approximator, which enables RL agents to generalise their knowledge to similar environments and learn from their actions similar to how a human may learn. The ability to learn different levels of abstraction from data means that DRL agents can solve much more complex tasks with less prior knowledge. The integration of deep learning into the RL setting, is the primary reason behind the advancements in the reinforcement learning paradigm, with much of the literature in recent years reporting finding impressive results in complex environments such as the Atari 2600 games.

There are many works conducted in the area of MADRL. Examples of such works include Lee and Jeong (2021), de Witt et al. (2020), Baker et al. (2019).

2.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is an area of study that describes a system of multiple RL agents that interact with each other

Stone and Veloso (2000) offers one of the earliest surveys into MARL, outlining the pros and cons of the two classifications proposed by the authors, from the perspective of machine learning implementations and propose a taxonomy as listed:

- Homogeneous non-communicating agents.
- Homogeneous communicating agents.
- Heterogeneous non-communicating agents.
- Heterogeneous communicating agents.

The authors also outline many of the benefits of multi-agent systems such as parallelism, robustness, and scalability. It is also noted that in some cases there may be increased cost-effectiveness by breaking down an entire system into smaller components (i.e., multiple smaller robots, each with their own functionalities as opposed to one larger robot with all). The geographic distribution of such a system is also noted as a possible benefit.

Gronauer and Dieopold (2022) proposes a schematic structure for the various types of multi-agent deep RL (MADRL) systems. The authors propose that such distributed systems can be classified based on the decentralised/centralised nature of the training paradigm and execution scheme, the non-stationarity of the environment, and whether the environment is an MDP or a partially observable MDP (POMDP). Some works explore MADRL in which continuous improvement is a key factor. Liang et al. (2021) explores the MADRL in which agents continuously adapt and optimise across multiple tasks, learning single policies for each task and distilling these into a unified generalisation policy. This achieves a similar effect to continual learning, though through a multi-agent system perspective.

2.3 Continual Learning

When neural networks attempt to learn the correlations between the input data and outputs, the backpropagation algorithm updates the values of weights within the network using the loss calculated after each training iteration. This continues until the loss reduces towards zero, a process known as gradient descent. If a new task is then introduced to the network, these weights are updated once again to learn the new information. In doing so, the previously learned information is effectively lost. This is known as catastrophic forgetting. The field of continual learning aims to identify methods to alleviate this problem and to enable neural networks to learn more.

Continual learning is a relatively new area of study and as such there have been many proposed taxonomies to classify the continual learning problem. One popular taxonomy is generally across many of the recent works in the field. Three incremental learning scenarios are proposed (van de Ven and Tolias, 2019), which differ on whether the identity of a given task is known. These incremental learning scenarios are listed below in order of difficulty.

- **Task Incremental Learning**. The identity of a task is provided. These models often utilise an output layer for each individual task while sharing the body of the neural network.
- **Domain Incremental Learning**. The identity is not provided to the model. Due to this limitation, these models typically attempt to solve only one given task at a time.
- **Class Incremental Learning**. In this paradigm the task identity is not given but must be inferred so that it may perform on any of the tasks that it has learned. This makes this scenario the most challenging to solve.

A recent work conducted by Wortsman et al. (2020) expands on this taxonomy by introducing additional parameters to be considered. The authors take into account whether the task space is continuous or discrete, whether or not a model has access to the task identity during the training step and/or the inference step, and whether or not the labels of tasks are shared across

all tasks or not. These learning scenarios are denoted using a three-letter system as described below in order of difficulty.

- **"GG"**: The identity is given to the mode during both the training and inference steps for both continuous and discrete task spaces.
- **"GNs"**: The identity of tasks are given to the model during training but not the inference step. The task labels are shared across all tasks. This scenario also accounts for both continuous and discrete task spaces.
- **"GNu"**: The identity is given during training but not during inference. The task labels are not shared and this scenario only accounts for a discrete task space.
- **"NNs"**: The identity not given to the model during either stage. Shared labels. Continuous or Discrete task space.

The authors of Wortsman et al. (2020) address these learning scenarios using their proposed continual learning algorithm: supermask in superposition (SupSup).

2.4 Continual Learning Desiderata

Several desiderata are often considered in many continual learning works. Due to the nature of continual learning, these desiderata can often be contradicting. This more often than not makes balancing these desiderata, very difficult. The most common desiderata that are observed across many works in this field, and thus could be considered essential, include the ability to minimise the effects of catastrophic forgetting in a model while maintaining a positive forward transfer (Wołczyk et al., 2021; Aljundi, 2019). This means that the agent must be able to learn new information, utilising previous experiences without losing said previous experiences. Some methodologies also try to balance computational and memory costs by asserting a fixed-size meaning the network does not expand to accommodate new information. Conversely, some methods opt to implement models that expand as required

at the cost of memory (Wołczyk et al., 2021; Sokar et al., 2021; Wortsman et al., 2020; Aljundi, 2019).

Some less common desiderata include:

- The ability to learn from a continuous stream of data (online learning) meaning a continual learner does not simply learn a new task in one go and move on, but rather consitently learns at every new sample of data (Aljundi, 2019)
- The ability to learn tasks without clear boundaries between tasks, and the ability to maintain plasticity and stability through graceful forgetting are also proposed desiderata.
- The authors explore the inaccessibility of previous training data when undergoing re-training steps in their model as a desiderata (Sokar et al., 2021).
- The (non)availability of task identities during inference stages as a desiderata (Wortsman et al., 2020)..

2.5 Continual learning methodologies

Three categories are generally accepted for the continual learning field. These are regularisation, replay/rehearsal and architectural methods. Many if not all continual learning approaches can be placed in one of these three categories. Some methods use a combination of concepts from two or more categories. These are known as hybrid methods. An non-exhaustive overview is shown in figure 3.

2.6 Regularisation Approaches

Regularisation CL approaches are based on the concept that changes to a network that compromise previous knowledge can be minimised to only those parameters that are not important, using a penalty. The most well-known implementation of this concept is Ridge Regularisation (L2) (Kirkpatrick et al., 2016). L2 regularisation is commonly used in deep learning ML applications. At a high-level L2 implements this property by introducing an L2

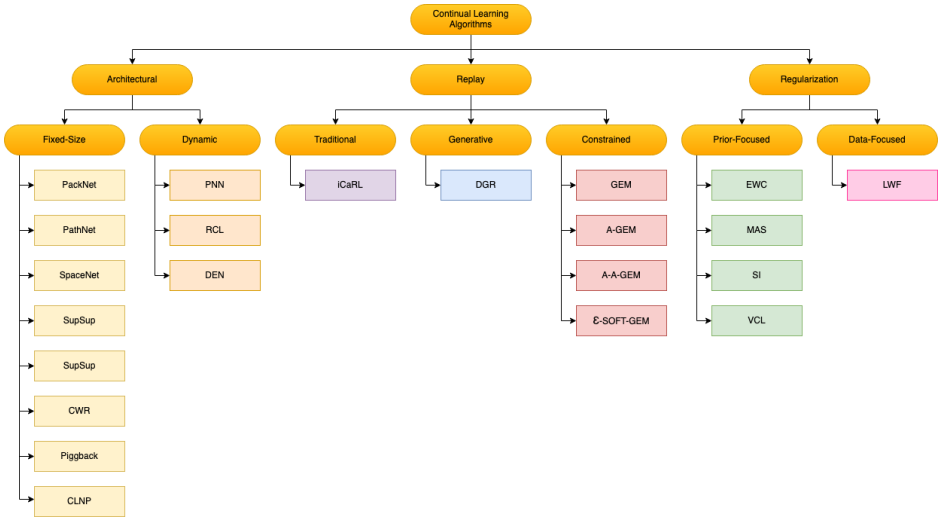

Figure 3: Existing popular continual learning methodologies and their classifications.

regularisation term to the loss function of the neural network model. This function is generally in the form of equation 11.

$$L2 = \frac{\lambda}{2N} \sum_{n=1}^{N} F_n w_n^2 \tag{11}$$

$\lambda > 0$ is the regularisation hyper-parameter that determines the strength of regularisation in the network, and w are the weights indexed by the value i. F_n denotes the importance of parameters to a task. This essentially gives every parameter in the network equal importance, thus every parameter is protected from change. This prevents the model from learning new information and thus, L2 is often used as a baseline when comparing regularisation-based CL approaches.

Many regularisation-based works typically approach from a Bayesian perspective. Thus, these methods can be sub-classified as either prior-focused or data-focused (Delange et al., 2021). The key difference between these categories is that prior-focused methods approximate the distribution of the parameters, whilst data-focused methods instead use a distillation of the knowledge from previous stages of training when learning new tasks. **Elas-**

tic Weight Consolidation (EWC) (Kirkpatrick et al., 2016) is an example of one of the most well known prior-focused methodologies. EWC works by approximating the importance of weights using the Fisher information matrix. Using these approximations, EWC attempts to protect the weights of importance and thereby, minimise the catastrophic forgetting of previous task knowledge.

Approaches such as **Variational Continual Learning (VCL** (Nguyen et al., 2017), **Synaptic Intelligence (SI)** (Zenke et al., 2017), and **Memory Aware Synapses (MAS)** (Aljundi et al., 2017) also use similar techniques to achieve the same result.

Learning Without Forgetting (LWF) (Li and Hoiem, 2016) is an example of a data-focused approach. LWF approximates the output response of the model on previous tasks, using the latest task data. The stability of the model on previous tasks can then be maintained using this approximation as a regularisation term.

Regularisation-based approaches suffer from one main limitation which is that, although they can minimise the effects of catastrophic forgetting, they cannot eliminate it. Thus, a network may still suffer from a loss of knowledge, especially as more and more tasks are introduced.

2.7 Replay/Rehearsal Approaches

These methods combine the training data of previous tasks with the data of the current task so that the network can learn all the tasks in one training cycle. This enables the network to refresh its memory of previous tasks, minimising the effects of catastrophic forgetting. Approaches that fall under this category can be further categorised under three sub-categories: Approaches that store samples of previous task data as part of the model, approaches that use a constraint, and approaches that use a secondary generative model to create training data. Methods such as **Deep Generative Replay (DGR)** (Shin et al., 2017) utilise training samples to train a generative model which is then used to re-train for the old tasks. DGR can thus, be applied in cases where old data may not necessarily be available. DGR is, however, limited

in that the generative model itself may sometimes be susceptible to catastrophic forgetting. **Gradient Episodic Memory (GEM)** (Lopez-Paz and Ranzato, 2017) falls under the constraint-based category. GEM stores a subset of samples (i.e., task descriptors) from a given task within an episodic memory which is then used to retain old task knowledge. Variations of the GEM approach have also been proposed. A-GEM (Chaudhry et al., 2019) expands on the capabilities of GEM in terms of efficiency and simplicity while A-A-GEM and ϵ-SOFT-GEM (Hu et al., 2020) implement a soft constraint to find a stable balance between learning new tasks and remembering old ones.

2.8 Architectural Approaches

Architectural methods have one major benefit over the other two categories. These approaches use various algorithms that manage the way networks allocate their parameters to tasks. This means that parameters for a specific task are not altered. By doing so these approaches generally eliminate catastrophic forgetting. These methods can be further categorised into two categories. Approaches that dynamically expand their network size to accommodate new tasks and approaches that maintain a fixed size capacity to maintain memory requirements. **Progressive Neural Networks (PNN)** (Rusu et al., 2016) is an example of a dynamically expanding approach. Each task that is learned by this model instantiates its own network which is frozen. This means that the network does not change in any way. Networks can form lateral connections between their weights, essentially re-using their parameters in an attempt to make up for the progressively increasing memory costs. Further work was done in Rusu et al. (2016) that suggests that the pruning of less active neurons in the network could reduce the memory costs, as the number of tasks learned increases. **Dynamic Expandable Network (DEN)** (Yoon et al., 2017) and **PathNet** (Fernando et al., 2017) are approaches that are similar to PNN that also use a dynamically expanding network. **Copy-Weights with Re-init (CWR)** proposes a computationally lighter counterpart to PNN. CWR implements a fixed-size model in which the output layer is expanded to accommodate new task labels. The parameters within the network are shared between these tasks. This approach tends

to result in flexibility and thus, limited performance.

PackNet (Mallya and Lazebnik, 2017) is an approach ins attempts to learn all tasks within one network, by iteratively pruning, re-training and freezing parameters. PackNet then generates a sparsity mask which is used as a filter for the network. This mask indicates active weights during inference given the task identity (Mallya and Lazebnik, 2017). This again eliminates catastrophic forgetting as the network parameters are not altered. PackNet also has the benefit of maintaining a fixed model capacity, though at a high computational cost. As it is a fixed model, the number of parameters available for learning also reduces over time, thus the scalability of smaller PackNet models is something to consider. **Continual Learning via Neural Pruning (CLNP)** (Golkar et al., 2019) provides an approach that is relatively cheaper computationally when compared to PackNet. CLNP implements a pruning method in which the average activity of neurons are used to identify which parts of the network can be re-used. **Piggyback** (Mallya et al., 2018) learns the filter masks of each task themselves, then applies this to a pre-trained dense network to select the corresponding parameters for inference given the task identity.

In the majority of these approaches, the availability of task identity is paramount. This is, however, not always guaranteed in the real world. **SpaceNet** (Sokar et al., 2021) and **Supermasks in Superposition (SupSup)** (Wortsman et al., 2020) propose novel methods to achieving continual learning without the availability of task identity during inference. **SpaceNet** Sokar et al. (2021) implements an approach in which parameters from the network are pre-allocated to observed tasks. Using sparse adaptive training, parameters of limited importance are dropped. This results in a sparse sub-network for each observed task. To increase the efficiency of parameter allocation, less important parameters in each sub-network can be shared where possible. All task outputs are shared on a single layer.

Wortsman et al. (2020) proposes Supermasks in Superposition, an alternative method in which the model forms a new sub-network or "supermask" for each task that it learns. During inference, these masks are superimposed onto the entire network and given a weighting. The identity of the task is

inferred based on the confidence of the system. This confidence is computed from the entropy of the output distribution of the superimposed tasks.

2.9 Hybrid Approaches

Hybrid methodologies use a combination of concepts of one or more of the aforementioned categories. Examples of these algorithms include **Incremental Classifier and Representation Learning (iCaRL)** and **Expandable EWC** (Jones and Sprague, 2018).

There has been some work conducted exploring the integration of continual learning with the deep reinforcement learning paradigm. Khetarpal et al. (2020a) provides a recent and thorough review of these works, while (Zhiyuan Chen, 2018) provides further insights. **Reinforced Continual Learning (RCL)** (Xu and Zhu, 2018) proposes an approach specific to reinforcement learning. The method proposes a framework of three models: "controller", "value network", and "task network". The controller is a long-short term memory (LSTM) model which is used to determine the number of nodes to expand the task network. The value network is a fully connected network (FCN) which approximates the value of the state for the reward policy (much like the critic in the actor-critic approach). Other approaches include **Policy Consolidation (PC)** (Kaplanis et al., 2019) and the Elastic Weight Consolidation (EWC). In Kirkpatrick et al. (2016) the authors provide further experiments that explore EWC in the reinforcement learning setting, on the Atari 2600 benchmark. Wołczyk et al. (2021) applies many of the aforementioned approaches in this section on their benchmark "Continual World". The authors in this work implement select continual learning methodologies with the Soft Actor-Critic RL methodology (Haarnoja et al., 2018). (Kaplanis et al., 2020) is another example of work done in the continual RL paradigm. The authors propose their method called Multi-Timescale Replay (MTR). As the name suggests the authors use a variation on the replay-based CL approach.

2.10 Benchmarks and Metrics

Benchmarks are a crucial requirement for the evaluation of any method. Most works in continual learning generally evaluate their approaches in the supervised learning setting, typically using image-based benchmarks such as MNIST (Wortsman et al., 2020; Xu and Zhu, 2018). Variations of the MNIST benchmark have also been used, such as permuted MNIST benchmark and split MNIST. In the former, the pixels of each image are permuted to generate randomized tasks of similar difficulty to the original benchmark; however, they require different solutions. A similar concept has also been applied to benchmarks such as ImageNet, CIFAR-10, and CIFAR-100 (Wołczyk et al., 2021; Xu and Zhu, 2018; Mallya and Lazebnik, 2017).

There are many benchmarks to assess continual learning models in the reinforcement learning setting. Works such as Kirkpatrick et al. (2016), Bellemare et al. (2013), and Mnih et al. (2013) have all used the Atari 2600 suite of digital games. In many cases, these models often far surpass human-level performance in these games. StarCraft2 (Raghavan et al., 2020), Minecraft (Tessler et al., 2016), and Lifelong Hanabi (Nekoei et al., 2021) also provide various game environments that have been used extensively in the reinforcement learning setting. Although these benchmarks tend to be very good for RL systems, they tend to take a very long time to use and are generally very computationally demanding. It is also unclear if these benchmarks are effective in assessing the finer desiderata of continual learning agents.

Wołczyk et al. (2021) propose the Continual World, based on the Meta-World benchmark proposed by Yu et al. (2019). The authors use a subset of Meta-World's simulated robotics manipulation tasks to construct a challenging benchmark for the continual reinforcement learning setting. Continual World has the benefits of providing a suitable way to study the forward transfer and catastrophic forgetting in CL methodologies while also being suitable to the RL paradigm. Wołczyk et al. (2021) achieves improved computational affordability relative to alternative benchmarks. Other benchmarks, such as MiniGrid (Chevalier-Boisvert et al., 2018) and CT-Graph (Ladosz et al., 2021) provide more lightweight benchmarks focused on the continual RL setting. Chevalier-Boisvert et al. (2018) provides a $N \times M$ grid world consisting

of rooms and objectives for the agent to interact with. Ladosz et al. (2021) provides a tree-like structure consisting of various states for an agent to navigate. States in the CT-Graph are mapped to randomly generated grid images that are used to represent specific states in the tree. MiniGrid and CT-Graph are both considered sparse reward environments meaning the reward is only given once the agent is near or at the goal state.

2.11 Continual learning applications

There have been many applications proposed for continual learning. For example, Khetarpal et al. (2020b) proposes the applications of continual RL in areas such as healthcare and education, among other more obvious areas such as robotics and logistics optimisation. Continual learning systems could be used to quickly adapt to the varying human biology when diagnosing symptoms or transferring experiences in identifying key identifiers of a disease to another disease (i.e., strokes and seizures). Hemati et al. (2021) proposes the applications of continual learning in anomaly and fraud detection in the financial sector while Philps et al. (2018) proposes a method called continual learning augmentation (CLA) which would be capable of identifying which experiences are the best to use, to maximise profit in financial investment. Continual learning methodologies could also be used in cases such as exploratory drones, and other autonomous systems which could benefit from the adaptability of continual learning augmented artificial intelligence models. Take for example an autonomous aircraft system capable of flying a specific model of aircraft. A continual learning-based system could then transfer that experience to another aircraft without compromising its previous knowledge. Continual learning has not been actively implemented in any particular industry, however, CL could be applied to any scenario in which personalisation and adaptability are key factors.

3 ShELL System Overview

Experiments conducted in this project will use code based on a prototype version of the Shared Experience Lifelong Learning (ShELL) system under development by Ese Ben Iwhiwhu, Dr Andrea Soltoggio of Loughborough University in collaboration with Vanderbilt University (VU), University of California Riverside (UCR) and the Defense Advanced Research Projects Agency (DARPA). In order to produce the results in this project, modifications have been made to integrate additional benchmark environments and metrics.

3.1 System Behaviour

The ShELL system, as used in this project, is a prototype system that implements an n number of individual agents that can communicate on a single host device, simulating a decentralised system. Based on the task curriculum, the environment returns a state-reward pair for each action taken by the agent. Each agent in the system has its own task curriculum, thus, for each agent, the environment will provide state-reward pairs for each task that the agent encounters. This RL loop is illustrated in figure 4.

It provides a platform to test multiple continual reinforcement learning agents using the SupSup and PPO algorithms on a specified environment on the Gym API. Each agent can continually learn tasks by learning the network masks for those tasks, which follows the concepts behind the Supermasks in Superposition continual learning approach (Wortsman et al., 2020), as explained in section 2. As the decentralised system is simulated, each agent is trained linearly. This means each agent is trained one after another, rather than in parallel. Each agent can request the best mask in the network when a task is encountered. If the mask is available then the agent can then utilise this mask. If unavailable, the agent will have the learn the mask from scratch. This mask can then be used again by the agent and shared with other agents in the network. For example, if agent 0 has learned task 0 and agent 1 then encounters the same task later in time, agent 1 can call upon agent 0's mask of task 0 to maintain consistent performance. This process of sharing masks is illustrated in figure 5. Agent 0 learns task 0 at the start of the system's lifetime. It then shares its mask to agent 1 when task 0 is

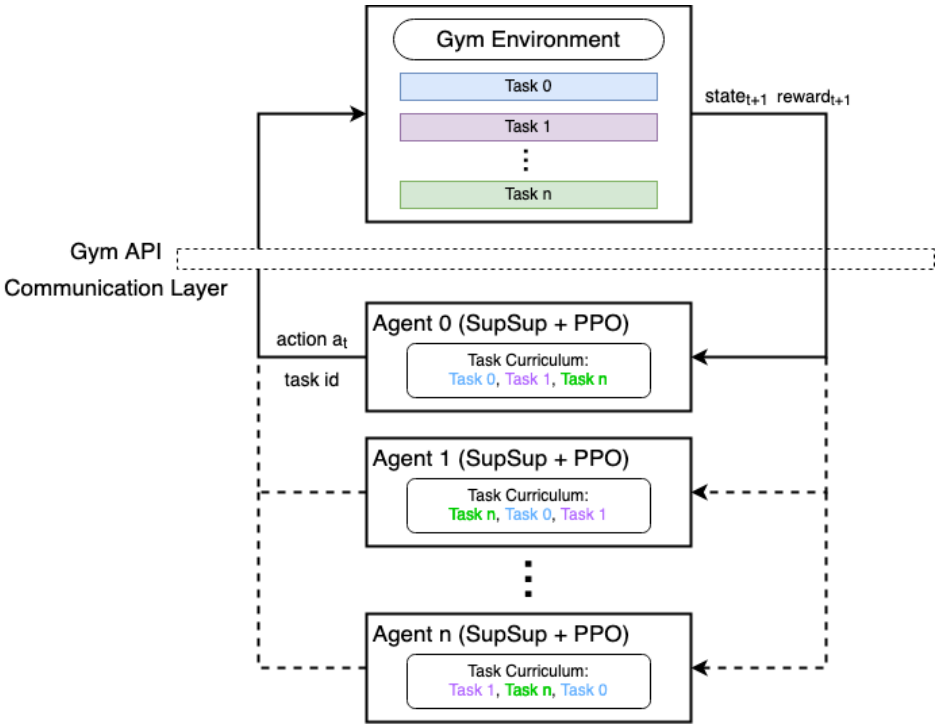

Figure 4: Reinforcement learning loop of the ShELL system. To achieve the agent-environment interaction, the OpenAI Gym API is used to handle the environment agent communication. This returns the state and reward from the environment to the agent for any action performed by the agent policy. Each agent has a task curriculum. Based on this curriculum the agent's action is fed to the environment. The environment returns a new state-reward pair for that task, given the agent's action. I.e., Agent 0 will feed an action to the environment for task 0, which will in turn return a new state-reward. Agent 0 will then do so for task 1 then task n. Agent 1 and agent n will behave similarly with their own unique task curriculum.

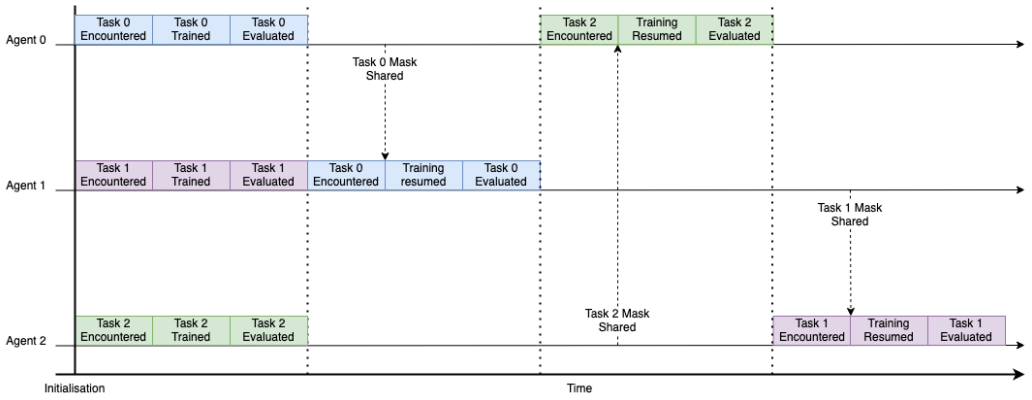

Figure 5: Illustrates the process of sharing task masks from an agent to another agent when the same task is encountered. This nullifies the need to train from scratch and maintains performance.

encountered again. Similarly, agent 2 shares its mask of task 2 with agent 0 when the task is encountered at a later time. In the case that two agents learn the same task at any given stage in the system lifetime, then the best mask is shared between the two. These rules ensure that knowledge can be shared and reused, minimising redundant training in the system.

3.2 Agent Architecture

The agent PPO architecture used in the ShELL system consists of a fully connected network (FCN) comprising of three hidden layers at 200 units per layer. The output of the FCN is fed into both the actor and critic head layer. These consist of a single layer of units equal to the number of actions, and a single layer of one unit respectively. The actor head produces the action to be taken as a result of the state and reward given to the agent. The critic head layer produces the resulting critic/value prediction. The value of weights in the network are randomly initialised as per the SupSup approach. The model then proceeds to learn the neural pathway as a floating point mask for any given task and stores this in memory to be used during inference.

3.3 Non-Overlapping Task Curricula

To identify the maximum potential benefits of a ShELL system of N agents, the task curriculum for a given experiment must be designed in such a way that it is non-overlapping. This means that no two agents in the system should observe the same task at any given time. To achieve this an algorithm has been implemented to generate a randomised non-overlapping task curriculum which is then used to run the experiments.

Algorithm 1: getMat

input : $n > 0$
$\underline{m} \leftarrow [0]_{n \times n}$
buildMatrix(n, m)
return $\underline{m}$

A zero matrix is first initialised in the getMat() function, which is then fed into the buildMatrix() method as a parameter. The variable n denotes the number of tasks in the environment (i.e., for an environment of 3 tasks, $n = 3$). The buildMatrix() method then fills in the values for both halves of matrix using the fillMatrix() method. The resulting matrix is a $n \times n$ matrix consisting of non-overlapping task IDs between $0...n-1$.

Algorithm 2: buildMatrix

input : $n > 0$
input : $\underline{m}$
$r \leftarrow n - 1$
$l \leftarrow 0$
for $i \leftarrow 0...n-1$ **do**
    if *i mod 2 is 0* **then**
        fillMatrix(i, r, n, m)
        $r \leftarrow r - 1$
    else
        fillMatrix(i, l, n, m)
        $r \leftarrow r + 1$

Algorithm 3: fillMatrix

input : $i \geq 0$
input : $j \geq 0$
input : $n > 0$
input : $\underline{m}$
$x \leftarrow 1$
for $k \leftarrow i + 1...n - 1$ **do**
    $m_{k,j} \leftarrow x$
    $x \leftarrow x + 1$
for $k \leftarrow 0...i - 1$ **do**
    $m_{k,j} \leftarrow x$
    $x \leftarrow x + 1$

4 Software and Hardware Specifications

This section provides an overview of the notable software libraries, APIs, and specifications of the hardware device that has been used in this project. It also gives a brief reminder of the continual learning and reinforcement learning algorithms in the ShELL system. Finally, this section outlines the software technologies used to conduct the experimentation process and manage version control.

4.1 Project packages and dependencies

This project has been built on the Python 3.6.8 platform, using a variety of packages that are listed in table 1. PyTorch with GPU acceleration serves as the core machine learning framework, with OpenAI's Gym API as the interfacing platform to handle the environment-agent communication loop. The core reinforcement learning system is built on top of the open source DeepRL library (Zhang, 2018). This library consists of a number of popular reinforcement learning algorithms using the PyTorch library. All packages and dependencies in the project were handled using a Conda environment on the Anaconda open source package and environment management system. The continual learning approach used in this project is based on the Supermasks in Superposition methodology proposed in Wortsman et al. (2020).

SupSup's sub-network-based methodology has shown promising results in the literature and exhibited impressive results in reinforcement learning baselines and eliminates catastrophic forgetting.

Reinforcement learning benchmarks can often be computationally expensive and training can be very time-consuming to complete. It is not uncommon for training times to take hours, days and even weeks. The continual world experiments conducted in Wołczyk et al. (2021) took an approximate 100 hours to complete successfully on an 8-core CPU machine. This benchmark saw no benefit with GPU acceleration thus, the authors proposed the use of a CPU machine instead. As a result, the benchmarks were chosen for this project needed to be lightweight and ideally GPU friendly. The CT-Graph and MiniGrid benchmarks are designed to be relatively lightweight and fast and are built to be used with the Gym API. They have very few dependencies and also benefit from GPU acceleration which makes them ideal for this project.

The MiniGrid benchmark consists of $N \times M$ grid worlds that can contain objects such as walls, floors, lava, doors, keys, balls, boxes and an objective. Each tile in the environment can contain either zero or one object and is encoded as a three-dimensional tuple consisting of the object id, colour id and state. The default reward function in this environment only rewards the agent with a value of $1-0.9*(step_t/maxsteps)$ upon successfully reaching the goal tile. The reward function of MiniGrid takes into account the number of steps taken to complete the task. Otherwise, the agent is rewarded a value of 0. Thus, the environment is sparse. There are a number of different environments available in this benchmark, each with variations as well as the ability to use seed values. For this experiment, the Simple Crossing and Lava Crossing environments have been used. The task IDs of these environments have been encoded as one-hot vectors for use in the ShELL system.

By default the ShELL system prototype had support for the MiniGrid environment, however, to use the CT-Graph environment, modifications had to be made. Specifically, these modifications were made in the environment wrapper to convert the task identifiers into one hot vectors that were suitable for the ShELL system, similar to what had been done for the Min-

Packages used in this project		
Package	Version	Description
Python	3.6.8	Python version 3.6.8 is the latest version of Python that is compatible with a number of the packages and dependencies used in this project.
PyTorch	1.10.2	Core ML framework based on the Torch library. Implements a tensor computation w/GPU acceleration, and deep learning models.
SciPy	1.5.3	Dependencies.
Matplotlib-base	3.3.4	Plotting of graphs and transforms.
JSON API	9e	Handles JSON objects within the Python programming language.
OpenAI Gym API	0.19.0	Reinforcement learning simulated environment toolkit provides a variety of environments, including 3rd party environments.
Gym-MiniGrid	1.0.3	Imports the MiniGrid environment on the Gym API.
Gym-CTgraph	0.1	Imports the CT-Graph environment on the Gym API.
Tensorboard	1.15.0	Visualization toolkit for model development and experimentation workflow. Allows for evaluation metrics to be logged and manipulated.
TensorboardX	2.5	Further support for other frameworks with Tensorboard.
Pandas	1.1.5	Data manipulation package for Python.
NumPy	1.19.2	Provides a variety of powerful data structures and mathematical array operations in Python that improve efficiency and capability.

Table 1: Versions of packages and modules used in the project. Includes a description of their use cases.

iGrid environment. CT-Graph stands for configurable tree graph. As the name suggests, this benchmark initialises dynamic tree graphs with sparse rewards. The environments can be configured as both Markov Decision Processes (MDP) or partially observable Markov decision processes (POMDP). For the experiments in this project, the environments have been set as MDP. Each CT-Graph tree consists of a number of states. These states include the home state, wait states, decision states, and the end state. The home state serves as a starting position for the agent while the end state serves as the goal. The agent must learn to navigate through the wait and decision states and discover the optimal trajectory to the end state. Each state is also mapped to randomly generated images which act as the model input. Decision and wait states are mapped to multiple variations of these images with some similarities so that the agent can differentiate between individual states without confusing them for other states. End and home states are mapped of singular images for the agent to identify. Each CT-Graph tree can be configured to have several end states, each of which serves as its own unique task, thus making this suitable for the evaluation of a continual reinforcement learning agent. To ensure that an agent does not remain in a single state, defined as a "crash", the tree can be configured to return a negative reward. This can be used to dissuade the agent from engaging in this behaviour and can make the environment easier to solve.

4.2 Hardware specifications

The hardware device used to run these experiments was a Linux based GPU server operating on Ubuntu OS 18.04.6 LTS on the 5.4.0-96-generic Linux kernel. The detailed specifications of this machine have been listed in table 2. Multiple experiments were run in parallel using the GNU Screen software for Linux. GNU Screen is a terminal multiplexer program that allows for several processes to be ran on a single physical console. This allows for the experiments to be ran in parallel using the available hardware resources and a single environment. Additionally, the GNU screen software allows for the execution of scripts on the server without being connected. This is a crucial advantage for this project as experiments took multiple hours/days to complete. Experiments were allocated their own GPUs to balance out

GPU server specifications	
Architecture	x86_64
CPU(s)	40
Thread(s) per core	2
Core(s) per socket	10
CPU Model	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
L1d cache	32K
L2i cache	32K
L2 cache	256K
L3 cache	25600K
GPU 1	NVIDIA GK210GL Tesla K80 (rev a1)
GPU 2	NVIDIA GK210GL Tesla K80 (rev a1)
GPU 3	NVIDIA GK110BGL Tesla K40c (rev a1)
GPU 4	NVIDIA GK210GL Tesla K80 (rev a1)
GPU 5	NVIDIA GK210GL Tesla K80 (rev a1)

Table 2: Versions of packages and modules used in the project. Includes a description of their use cases.

the total load on the GPU server. To interface with the GPU server, a combination of Visual Studio Code and PyCharm was used. This made interacting with the GPU server and repository to be more straightforward, which reduced in turn increased productivity. Version control and repository management was enforced by building a remote repository on the DevOps software service GitLab, using Git for Linux.

5 Experiment Methodology

A total of five experiments were conducted in this project. These consist of three experiments on the CT-Graph benchmark mentioned here (Ladosz et al., 2019) and two on the MiniGrid benchmark that can be found here (Chevalier-Boisvert et al., 2018). This section presents a breakdown of these experiments and their parameters as well as illustrations of the environments and chosen tasks.

5.1 CT-Graph Experiments

Three CT-Graph experiments were conducted using depths of 2 and 3 with a branching factor of 2, and a depth of 3 with a branching factor of 3. This results in three experiments with 4, 8 and 27 tasks respectively. These environments have been illustrated in figures 6. Experiments were initially conducted for 102400 steps with the exception of the 27 task CT-Graph experiment. This environment is relatively more complex and requires additional steps for agents to successfully solve, thus 1e6 steps were used.

The results of these experiments indicated that the single continual learning agent was not able to solve all tasks successfully. Specifically for the 27 task CT-Graph on seed 9157, the agent was unable to solve tasks 17 and 18 which led to a max ICR of 25 instead of 27. This resulted in an ICR below the expected maximum ICR. As a result certain metrics were skewed in the benefit of ShELL. An example of this can be seen in 9 with the 27 task CT-Graph ICR. This issue is likely due to there not being enough steps to learn all the tasks in the larger CT-Graph trees. To alleviate this problem, additional steps were introduced for the 8 and 27 task CT-Graph experiments, for 1.204e6 and 2.56e6 respectively. Unfortunately, due to the amount of time required to complete these experiments, the 27-task CT-Graph results were not available in time. As a result, the results shown in this report for this particular experiment reflect the original skewed results.

Additional notable parameters include a crash reward of -0.01 which was applied to make solving these environments slightly easier on the agents. Each of the three experiments in the CT-Graph environment was run on

ShELL systems of 2, 3 and 10 agents. Care was taken in experiments to ensure the task curriculum of each of the agents did not overlap. This was not possible with experiments where the number of tasks was lower than the number of agents. In this case, some of the agents would be learning the same tasks at the same time. A detailed overview of the parameters used for the three CT-Graph experiments can be found in table 5.

5.2 MiniGrid Experiments

Two MiniGrid experiments were conducted using variations of the simple crossing, and lava crossing. The first experiment consists of only three variations of simple crossing, resulting in 3 tasks. The second experiment consists of a combination of three variations of simple crossing and three variations of lava crossing, resulting in 6 tasks. These tasks have been illustrated in figure 7. MiniGrid experiments were run on 2, 3, and 10 agents for Simple Crossing 3 tasks, and 2, 3, and 6 agents for the Simple Crossing + Lava Crossing experiment. Due to the higher complexity of the SC+LC experiment, 1.024e6 steps were used. This ensured that the agents did not fail to learn any tasks. Task 6 of the SC+LC experiment (Lava Crossing variation 3) was especially complex and posed a real challenge for agents to solve. A detailed overview of the parameters used for these experiments can be found in table 4.

Similar to the 27-task CT-Graph, initial experiments showed that the single agent did not successfully learn all the tasks in the 6-task MiniGrid environment. As a result, the max ICR did not meet the expected value, which skewed the metrics in favour of ShELL. Due to the time required to rerun the experiment for an increased number of experiments and investigate the underlying issue, the results were not available for this report.

5.3 Metrics

To assess the performance of a system such as ShELL, a number of specific metrics must be used. For this project ShELL performance was assessed using a combination of continual learning metrics and ShELL-specific metrics. The ShELL system is evaluated regularly throughout training in the form of evaluation blocks. During evaluation blocks, the training process

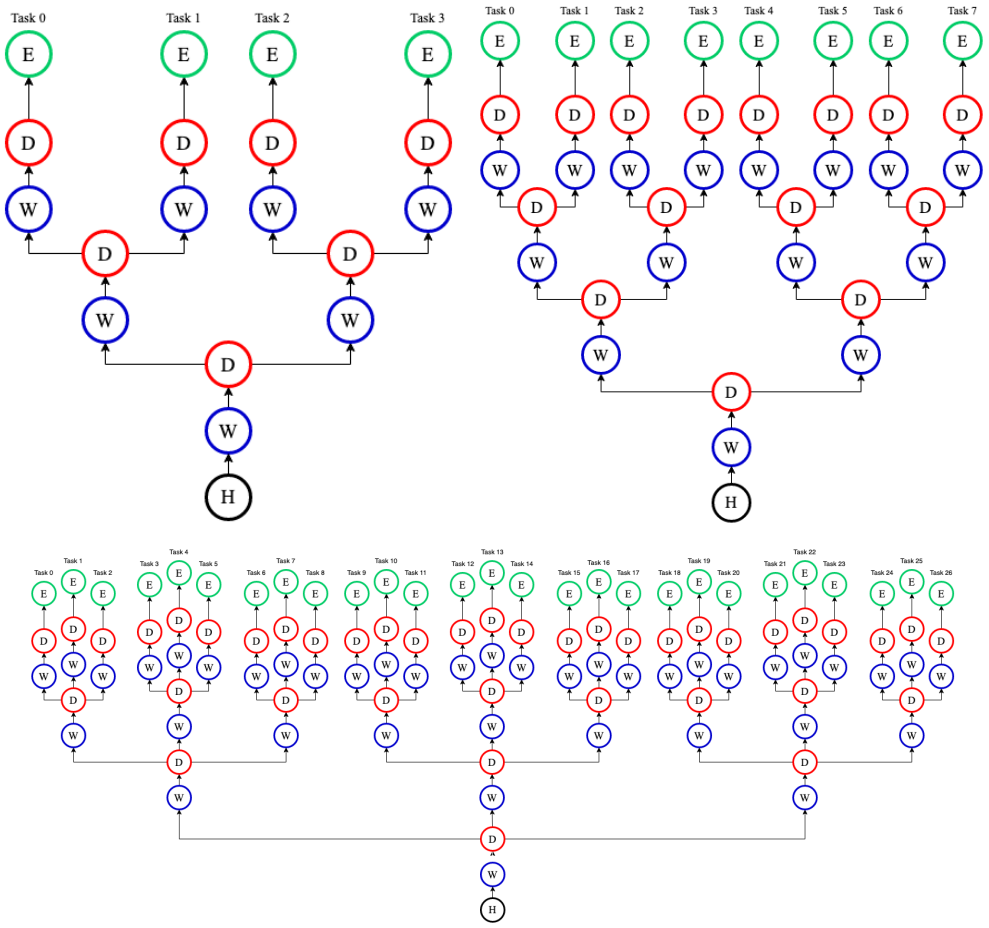

Figure 6: Illustrations of CT-graph trees for branching factor 2 experiments. Left: CT-Graph with depth 2, branching factor 2. Resulting in 4-tasks. Right: CT-Graph with depth 3, branching factor 2. Resulting in 8-tasks. Bottom: CT-Graph with depth 3 and branching factor 3. This results in 27 tasks.

hyper-parameter	Depth 2 (4-tasks)	Depth 3 (8-tasks)	Depth 3 Branching Factor 3 (27 Tasks)
general seed	3	3	3
tree depth	2	3	3
branching factor	2	2	3
wait probability	0.0	0.0	0.0
high reward value	1.0	1.0	1.0
crash reward value	-0.01	-0.01	-0.01
stochastic sampling	false	false	false
reward standard deviation	0.1	0.1	0.1
min static reward episodes	30	30	30
max static reward episodes	70	70	70
reward distribution	needle in haystack	needle in haystack	needle in haystack
MDP decision states	true	true	true
MDP wait states	true	true	true
wait states	$[2, 8]$	$[2, 16]$	$[2, 41]$
decision states	$[9, 11]$	$[17, 23]$	$[42, 54]$
graph ends	$[12, 15]$	$[24, 31]$	$[55, 81]$
image dataset seed	1	1	1
1D format	false	false	false
image dataset seed	1	1	1
number of images	16	32	82
noise on images on read	0	0	0
small rotation on read	1	1	1
max steps	102400	1.024e6	1e6

Table 3: CT-graph experiment parameters. Each group of experiments have been run on ShELL systems of 2, 3 and 10 agents.

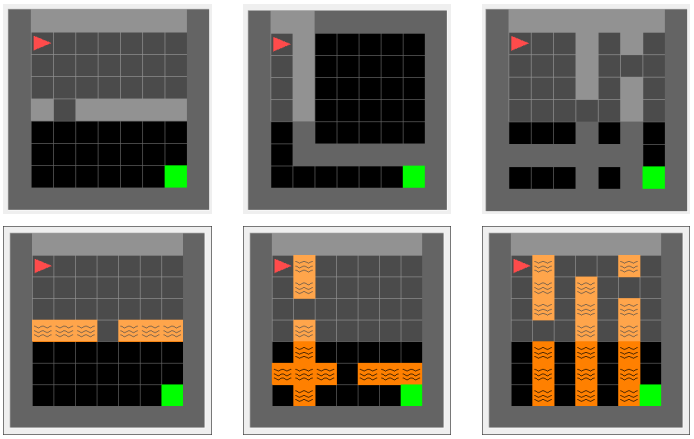

Figure 7: Illustrates a visual representation of the MiniGrid environments. Top: Simple Crossing task variations (task 0, 1, 2). In these tasks, the agent must reach the green square. Walls pose as an obstruction to the agent's objective. Bottom: Lava Crossing task variations (tas. In these tasks, the agent must also reach the green square, however the internal walls have been changed to lava. Lava Crossing variation 3 is especially difficult to succeed. Images have been taken from Chevalier-Boisvert et al. (2018)

hyper-parameter	Simple Crossing (SC) (3 Tasks)	Simple Crossing (SC) + Lava Crossing (LC) (6 Tasks)
tasks ids	Task 0: SC V1, Task 1: SC V2, Task 2: SC V3	Task 0: SC V1 , Task 1: SC V2, Task 2: SC V3, Task 3: LC V1, Task 4: LC V2, Task 5: LC V3
one hot	true	true
label dimensions	8	8
action dimensions	3	3
max steps	102400	1.024e6

Table 4: MiniGrid experiment hyper-parameters. The 3 task MiniGrid was run on ShELL systems of 2, 3 and 10 gents. The 6 task MiniGrid was run on 2, 3 and 6 agents.

hyper-parameter	Value
learning rate	0.00015
cl preservation	ss
seed	9157
number of workers	4
optimiser function	RMSprop
discount rate	0.99
use general advantage estimator	True
entropy weight	0.1
rollout length	128
optimisation epochs	8
number of mini batches	64
ppo ratio clip	0.1
iteration log interval	1
gradient clip	5
max steps	102400 or 1e6
evaluation episodes	10
require task label	True
evaluation interval	25

Table 5: ShELL system parameters. These parameters were used across all experiments to maintain consistency, with the exception of the max steps. This has been modified as required, to match the experiment parameters.

of agents are frozen so that the agent can be evaluated across all tasks in the curriculum. The continual learning-specific metrics were logged via TensorBoard. These logged values were then pulled and used to calculate the ShELL-specific metrics after the experiments were completed. To generate the results in this project, these metrics had to be implemented as part of the ShELL system so that the ICR and TPOT values could be captured at each evaluation block. The instant cumulative reward (ICR) is the sum of the rewards gained by the entire ShELL system when tested across all tasks in the curriculum. The ICR is calculated via equation 12.

$$\mathrm{ICR}(t) = \sum_{task=1}^{\mathcal{T}} A^*_{\tau,t} \tag{12}$$

Here $A^*_{\tau,t}$ is the reward gained on task τ during the evaluation block t for the agent with the most up-to-date knowledge on said task. The ICR value of ShELL can then be used to calculated the total performance over time (TPOT) at the evaluation block, T. TPOT can be calculated via the equation 13

$$\mathrm{TPOT}(T) = \sum_{t=0}^{T} \mathrm{ICR}(t) \tag{13}$$

These metrics were measured and computed during the evaluation blocks and logged onto TensorBoard. The ShELL-specific metrics were computed after the experiments were completed as they required both the ShELL experiment data as well as the single continual learning agent data as a baseline. The ShELL-specific metrics are used to assess the benefits of the ShELL system over a single continual learning agent. These metrics include the Instant Learning Advantage (ILA), the Total Learning Advantage (TLA) and the Time Reduction Advantage (TRA). The Instant Learning Advantage (ILA) can be defined as the ratio of the instant cumulative reward (ICR) of a ShELL system of n number of agents over the ICR of a single continual learning agent. This can be written as equation 14

$$\mathrm{ILA(t)} = \mathrm{ICR}(\mathrm{ShELL}, t)/\mathrm{ICR}(\mathrm{SingleLLagent}, t) \tag{14}$$

The Total Learning Advantage (TLA) indicates the benefit of ShELL by comparing the amount of reward gained over time by ShELL versus the

amount of reward gained by a single agent over time. This ratio can be written as the equation 15.

$$\text{TLA} = \text{TPOT}(\text{ShELL}, t)/\text{TPOT}(\text{SingleLLagent}, t) \tag{15}$$

Lastly, the Time Reduction Advantage (TRA) indicates the benefit of ShELL by comparing the time taken for the ShELL system to reach a certain performance level, to that of a single agent system. To indicate benefit, a ShELL system would reach this goal much before a single agent system. If $TT_p(\cdot)$ is the training time required to achieve a target predetermined ICR then the equation for TRA can be written as equation 16.

$$\text{TRA(p)} = \text{TT}_p(\text{SingleLLagent})/\text{TT}_p(\text{ShELL}) \tag{16}$$

Using a these metrics, the benefits of the ShELL system over a single continual learning agent will be assessed.

5.4 Hypothesis and Initial Experiment Results

It is hypothesised that as the number of agents, N, increases, so too will the performance of the ShELL system. As N increases, more agents will be able to learn more tasks sooner which results in more knowledge being readily available to all agents in the system. This will be made apparent by an improvement in the instant cumulative reward and by extension the total performance over time. The ShELL-specific metrics should also indicate a benefit to using a ShELL system over a single continual learning agent. As N tends towards the number of tasks in the environment, it is expected that ShELL will likely obtain the target TRA score much sooner than a single agent system.

Initial experimentation of the ShELL baselines using the SupSup and PPO algorithms indicated that the model was able to successfully learn permutations of the MiniGrid Simple Crossing variations with the accuracy across all three tasks averaging 0.9574. It is expected that the system will be able to maintain this level of performance across the other experiments.

6 Experiment Results

This section covers the experimental results from the five experiments mentioned in section 5 as well as provides an overview of the learning behaviours of select agents from the ShELL systems. Results are analysed with the original hypothesis and an evaluation is made of the findings.

6.1 CT-Graph Results

The 4-task CT-Graph environment was a fairly simple environment for the agents. As a result, they did not have much difficulty in solving the tasks. The total run time of the 2, 3 and 10 agent experiments was 44m 46s, 1h 7m 46s, and 3h 40m 20s respectively. These times increase due to the fact that ShELL system trains agents linearly meaning each agent is trained on each task in its curriculum, one after another. Therefore, the amount of time taken to train the system increases as the number of agents increases. A fully distributed ShELL system would likely exhibit a decreasing training time with agents being able to learn in parallel. This has been taken into consideration when calculating the metrics of the ShELL system for all five experiments. The 8-task CT-Graph environment introduces some additional difficulty due to the increase in the size of the search space. The run times of the 2, 3 and 10 agent experiments on this environment are 15h 50m 55s, 23h 28m 28s, and 3d 1h 11m 37s respectively. There is a further increase in the training times as the number of the tasks increases to 27. Training times for the CT-Graph depth 3 with a branching factor of 3 for 2, 3, and 10 agents were 3d 0h 54m 23s, 4d 9h 11m 20s, and 14d 15h 54m 58s.

Figure 8 shows a visualisation of learning of the 4-task CT-Graph experiment for a 2-agent ShELL system. The plots show the average, maximum and minimum rewards accumulated by the two agents in the ShELL system. For this particular experiment the task curriculum was set as Agent 0: 0, 1, 2, 3, Agent 1: 1, 2, 0, 3 (each integer value indicates a task id from the environment, i.e., 0 indicates task 0 or end state 0 in the CT-Graph tree). This task curriculum promotes knowledge transfer during the second and third learning blocks, where agent 0 can make use of the knowledge learned by agent 1 on tasks 1 and 2. Agent 1 can also make use of the task 0 knowledge

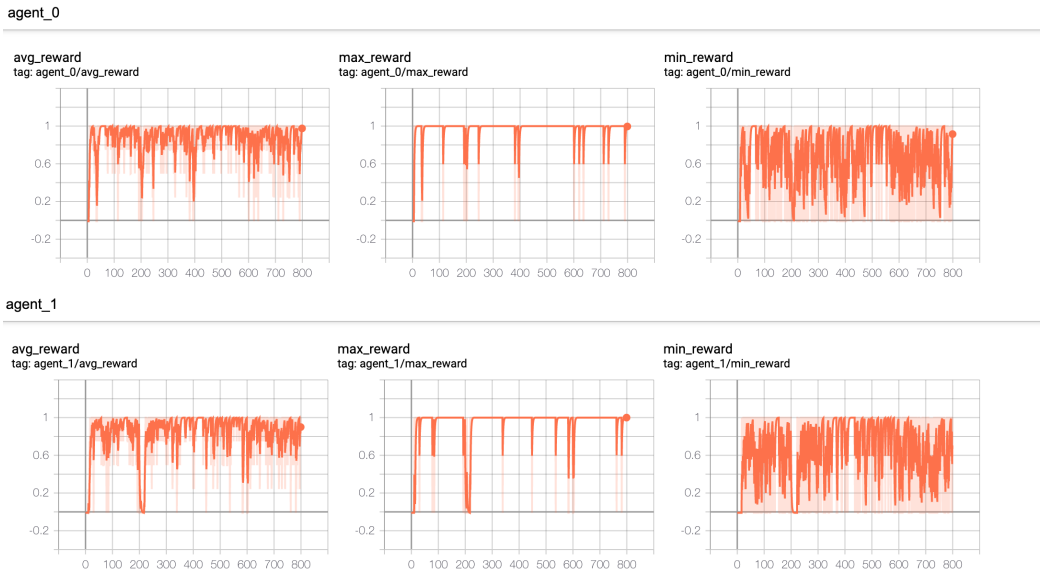

Figure 8: From left to right: Average, maximum and minimum rewards measured throughout training of (a) Agent 0 (b) Agent 1 of a two agent ShELL system on 4-task CT-Graph. Experiment task curriculum is as follows. Agent 0: 0, 1, 2, 3, Agent 1: 1, 2, 0, 3. Each integer indicates a task id that corresponds to a specific task end state in the CT-Graph tree. Refer to figure 6.

from agent 0. Task 3 is learned by both agents at the same time during the final block. Each task took a total of 200 iterations, with a total of 4-tasks in the curriculum. Thus, the complete experiment took a total of 800 iterations to complete. Based on these graphs, it is clear that the SupSup+PPO based agents are able to continuously learn multiple tasks without catastrophic forgetting, and with consistently high performance.

Each agent in the system begins by learning their first tasks from scratch. If the task masks are available for a task encountered after this, then the rewards accumulated do not decrease. For example in agent 1, during iterations 400 and 600, task 0 is encountered. As stated previously, agent 1 can use the mask for task 0 from agent 0, thus there is no visible dip in the rewards. On the other hand, agent 0 encounters task 3 at 600 iterations. This task has not been encountered before and as a result, the agent must learn this task

from scratch.

Figure 9 shows the instant cumulative reward (ICR) and total performance over time (TPOT) for the three CT-Graph experiments. It can be observed that both of these metrics show evidence that the performance of the ShELL system increases as the number of agents tends towards the number of tasks in the environment. As the number of agents passes the number of tasks i.e., 10 agents for 4-tasks, the performance gains are not as apparent. This is reflected in the TPOT graphs which show a larger gap in total performance between 3 and 10 agents when the number of tasks is increased from 4 to 8. With 27 tasks the performance gains are fairly steady and following the trend, it is possible that as the number of agents tends towards 27+ this performance gain will saturate. The ICR graphs indicate that as the number of agents increases, so does the cumulative reward accumulated by the ShELL system. This is likely because more agents can learn more tasks sooner and thus, the task masks are available to be shared sooner. In either case, there is a clear advantage to the ShELL systems over a single continual learning agent.

Figure 10 shows the ShELL-specific metrics of instant learning advantage (ILA) and total learning advantage (TLA) for each of the three CT-Graph experiments. As stated in section 5, ILA can be defined as a ratio of the ICR of ShELL over the ICR of a single continual learning agent. Similarly, TLA can be defined as a ratio of TPOT of ShELL over the TPOT of a single agent. It is apparent that with 4-tasks, the 3 and 10 agent systems appear to have 3x the amount of reward gained when compared to a single continual learner. The 2 agent system on the other hand exhibits only 2x the amount of reward, though this is still an improvement over a single agent. Similar results are shown in the 8 and 27 task CT-Graph experiments, with 10 agents reporting a maximum of 7x and 12x the reward gained over a single agent. Based on the ILA results it is clear that there is a benefit to the ShELL system compared to a single agent and this is reflected by the total learning advantage which is calculated from the total reward across all tasks from time step 0. The similar results of the 3 and 10 agent systems in the 4-task CT-Graph show that there is a plateau in the amount of reward gained from increasing the number of agents past the number of tasks (4-tasks), however,

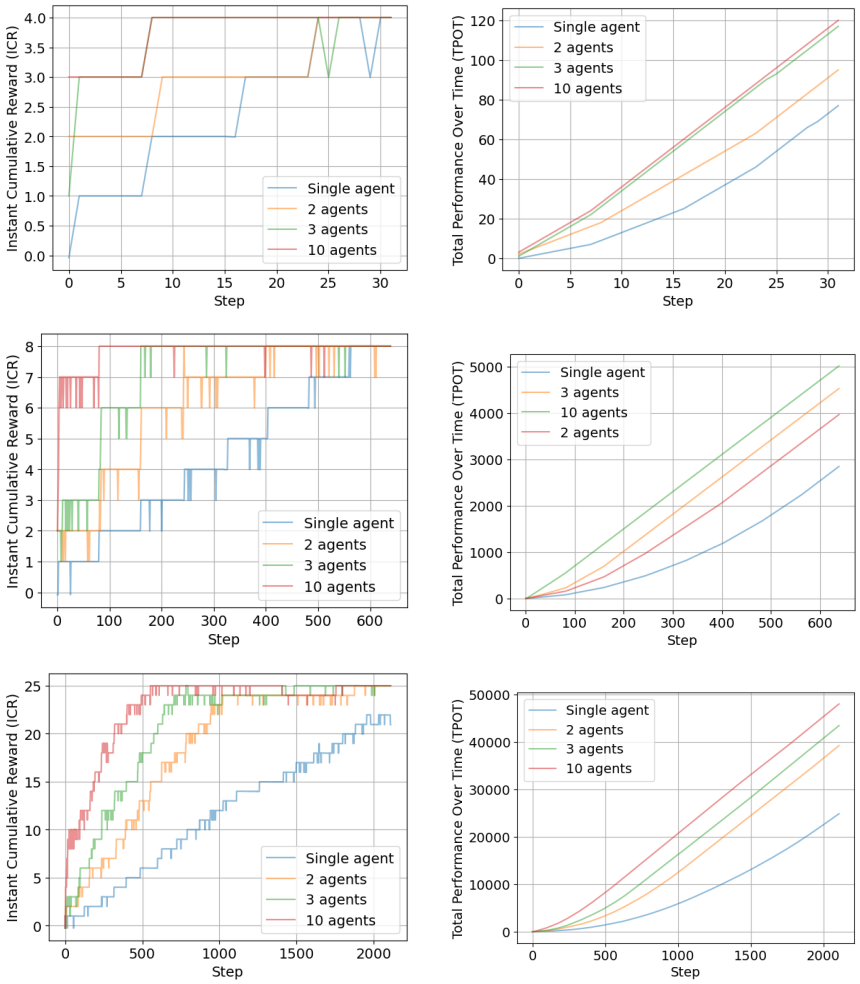

Figure 9: Plots of the instant cumulative reward (ICR) and total performance over time (TPOT) metrics for CT-Graph experiments. The top row consists of the ICR and TPOT of 4-task CT-Graph. The middle row consists of the ICR and TPOT of 8-tasks CT-Graph. The bottom row consists of the ICR and TPOT of 27 task CT-Graph.

from the TLA plot, it is apparent that the 10 agent system still posses a learning advantage when compared to the 2 and 3 agent systems.

Based on these graphs it is clear that the proposed $1.3N$ target, where N is the number of agents in the ShELL system, is not achieved. However, The alternative target of $1+0.5N$ is met with the 8 and 27 task CT-Graph experiments. With the alternative target is expected that the target performance will be 2 agents: 2, 3 agents: 2.5, 10 agents: 6. Observations show that the 2, 3 and 10 agents are able to hit these targets. An additional observation can be made based on the results of the 27-task CT-Graph. Due to the increased number of evaluation blocks in the experiment, there is a finer granularity in the results. As such, the ILA metrics are indicating performance metrics of up to 12x. It is possible that with finer granularity, a higher peak may be observed which would achieve the $1.3N$ target, though a finer granularity would increase the run times of each experiment drastically.

Figure 11 shows the individual time reduction advantage scores for each of the ShELL systems for each of the three CT-Graph experiments. The TRA ShELL metric serves as a measure of how much quicker a ShELL system is when compared to a single continual learner. As stated in section 5 it is expected that as the number of agents in the ShELL system increases, so too will the TRA score as more agents will be able to have a larger variety of tasks. This is supported by figure 11 which suggests that as the number of agents increases, so will the number of tasks observed in the ShELL system. This results in the ShELL system being able to share knowledge between agents which results in a performance improvement. Based on the results shown, the 3 and 10 agent systems in the 4-task CT-Graph experiment were 3 times faster than a single agent. 8-task CT-Graph experiments showed that the 2, 3 and 10 agent system was 81 times faster than the single agent at reaching 25% of the max ICR, though the time reduction advantage then falls off at later stages for the 2 and 3 agent systems. The 10-agent system in the 27 task environment was 48.8 times faster than the single agent at achieving 25% of the max ICR and continued to achieve target ICR values before the 2 and 3 agents. This indicates a clear benefit in the learning capabilities of the ShELL system in terms of versatility and the inherent longevity resulting from the use of continual learning methodologies. The results also indicate

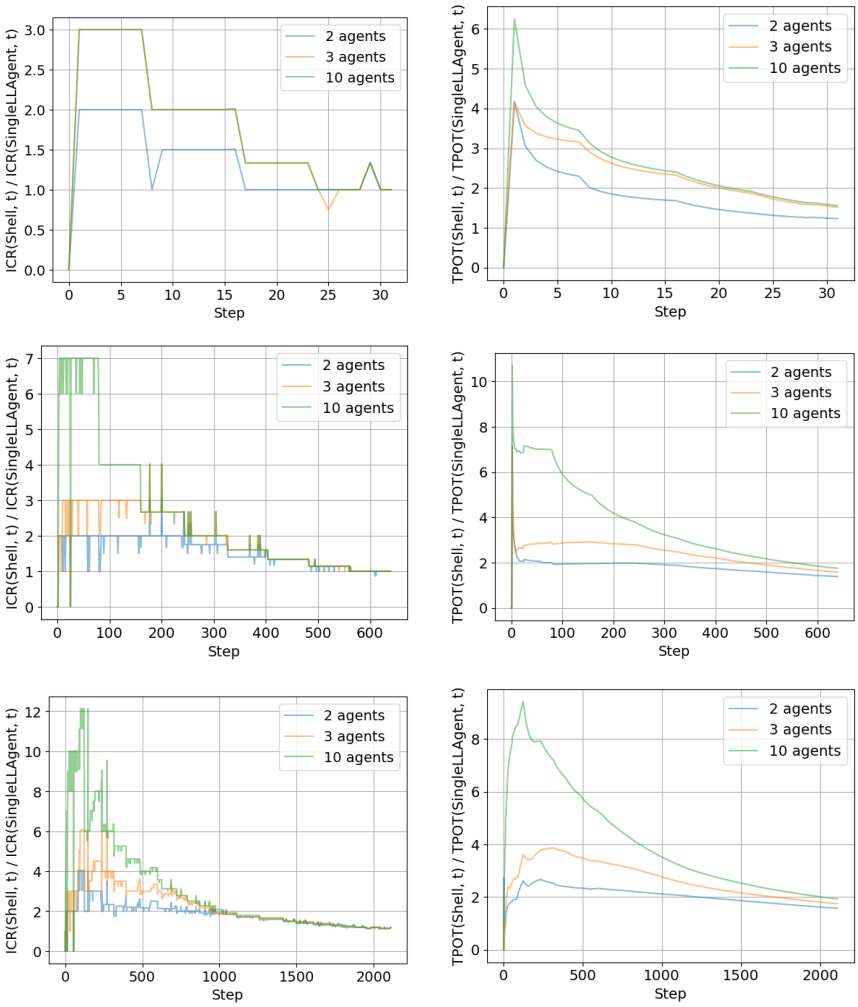

Figure 10: Plots of the instant learning advantage (ILA) and total learning advantage (TLA) ShELL specific metrics. Top row: 4-task CT-Graph. Middle row: 8-tasks CT-Graph. Bottom row: 27-task CT-Graph

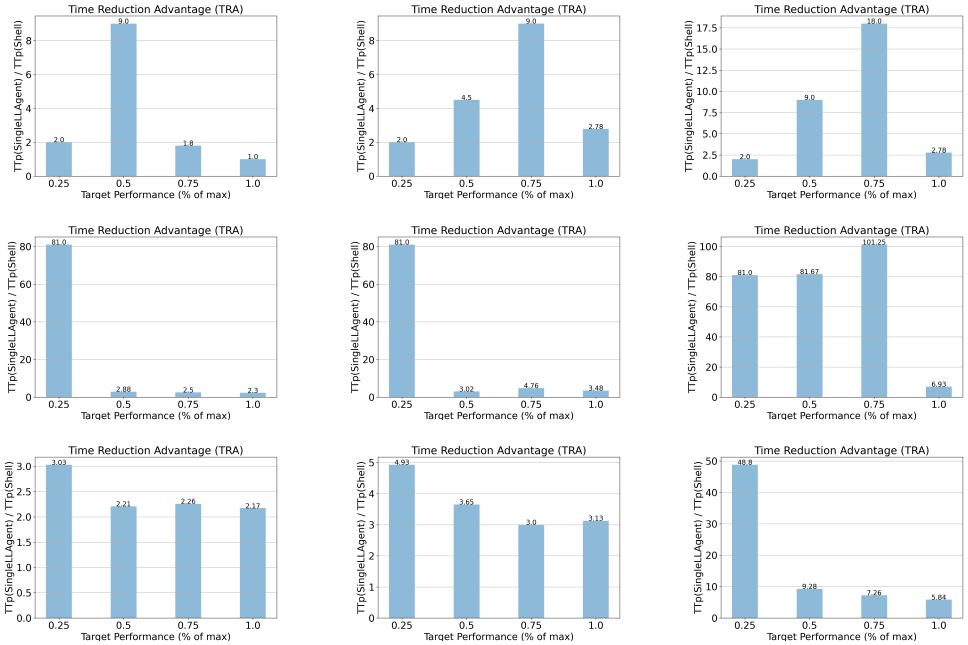

Figure 11: Time reduction advantage scores calculated at 25%, 50%, 75% and 100% of the max ICR possible in each experiment. From left to right: 2 agents, 3 agents, 10 agents. Top row: 4-task CT-Graph. Middle row: 8-task CT-Graph. Bottom row: 27-task CT-Graph. More agents are able to achieve a higher TRA score, sooner.

an increase in performance as the number of agents in the ShELL system approaches the number of tasks in the experiment. It is expected that the TRA results for the 27 task CT-Graph would in reality be slightly lower for a single agent that achieves the expected max ICR.

Given the proposed target of $0.5N$ the expected results for the TRA metrics are 2 agents: 1, 3 agents: 1.5, 10 agents: 5. The results shown in 11 indicate that the ShELL systems are able to achieve the proposed $0.5N$ target for time reduction advantage at 100% performance, with the exception of the 10 agent system on the 4-task CT-Graph environment. This is likely due to a sub-optimal set of task curriculum for the 10 agent ShELL system. If the task curriculum were further optimised, the 10 agent system would be able to meet the the target score of $0.5N$.

6.2 MiniGrid Results

MiniGrid Simple Crossing (3 tasks) environment was another example of a relatively simple environment. Agents were able to solve the Simple Crossing tasks with relative ease. The total run times of the 2, 3, and 10 agents experiments were 43m 30s, 1h 6m 30s, and 3h 14m 28s respectively. MiniGrid Simple Crossing + Lava Crossing (6 tasks) was a much more difficult environment with the addition of the three variations of the lava crossing tasks. Variation 3 of the lava crossing environment is especially difficult. The run time of the 2, 3 and 6 agent systems was 19h 38m 31s, 1d 7h 52m 39s, and 2d 8h 32m 15s respectively.

Figure 12 shows plots of the rewards at each step of training of the three agent ShELL system on the 3-task MiniGrid environment. The plots show the average, maximum and minimum rewards accumulated by the three agents in the ShELL system. For this particular experiment the task curriculum was set as Agent 0: 0, 1, 2, Agent 1: 1, 2, 0, Agent 2: 2, 0, 1 (again each integer value indicates a task id from the environment. Refer to the task IDs in table 4). This task curriculum promotes the transfer of knowledge during both the second and third learning blocks across all three agents, as there is no task overlap. All three agents begin by learning their first tasks from scratch. At this stage there is a mask available for each task in the environment, thus in later encounters, the agents can ask for the corresponding masks. This is reflected by the smooth curve with no dips indicating consistent performance. Each task took a total of 200 iterations, with a total of 3 tasks in the curriculum. Thus, the complete experiment took a total of 600 iterations to complete.

Figure 13 shows the instant cumulative reward (ICR) and total performance over time (TPOT) for the two MiniGrid experiments. The ICR plots for the 3-task MiniGrid experiment show that the 3-agent system appears to reach the peak value before the 10 agent system can do so. This is an unexpected result and appears to be an anomalous result as it immediately drops back before rising again after the 10 agent system hits the peak and maintains stability. The 6-task MiniGrid ICR plot shows that the 6-agent system outperforms the 2 and 3-agent systems, given there are 6 tasks. These

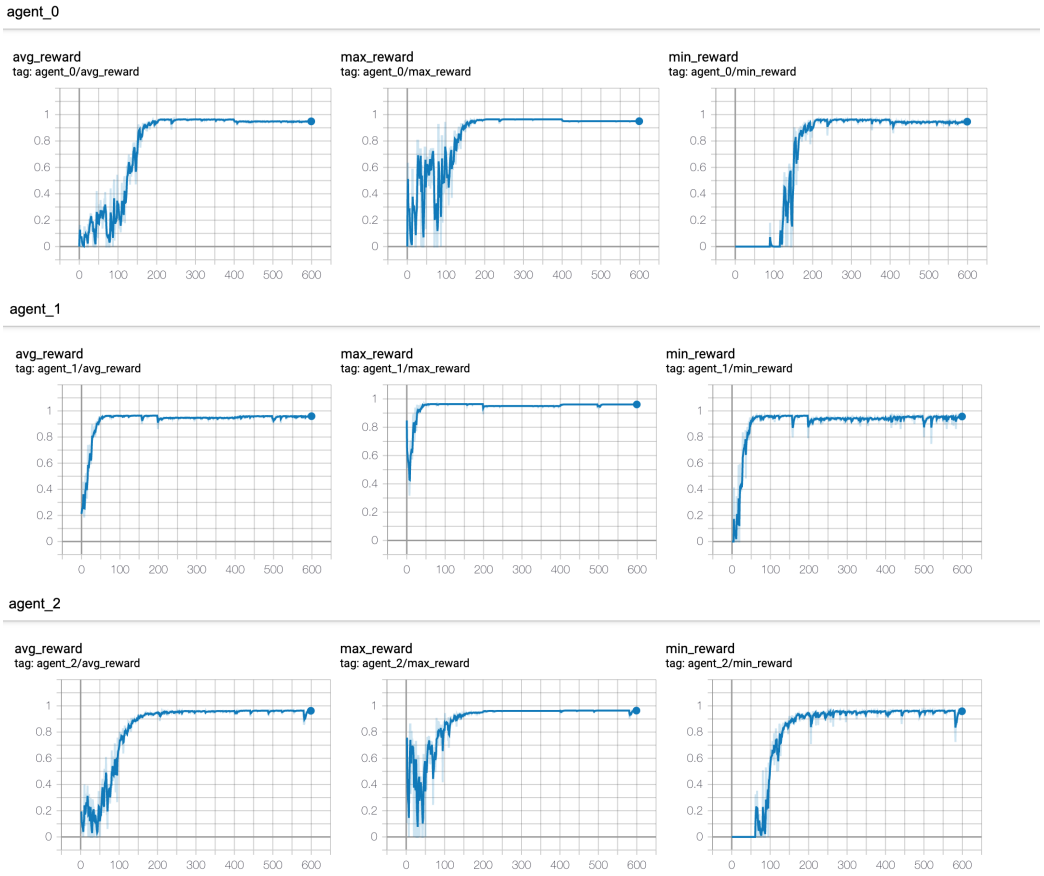

Figure 12: From left to right: Average, maximum and minimum rewards measured throughout training of Top: Agent 0, Middle: Agent 1, Bottom: Agent 2 of a three agent ShELL system on 3-task MiniGrid. Experiment task curriculum is as follows. Agent 0: 0, 1, 2, Agent 1: 1, 2, 0, Agent 2: 2, 0, 1. Each integer indicates a task id that corresponds to a specific task variation in the MiniGrid environment. Refer to figure 7.

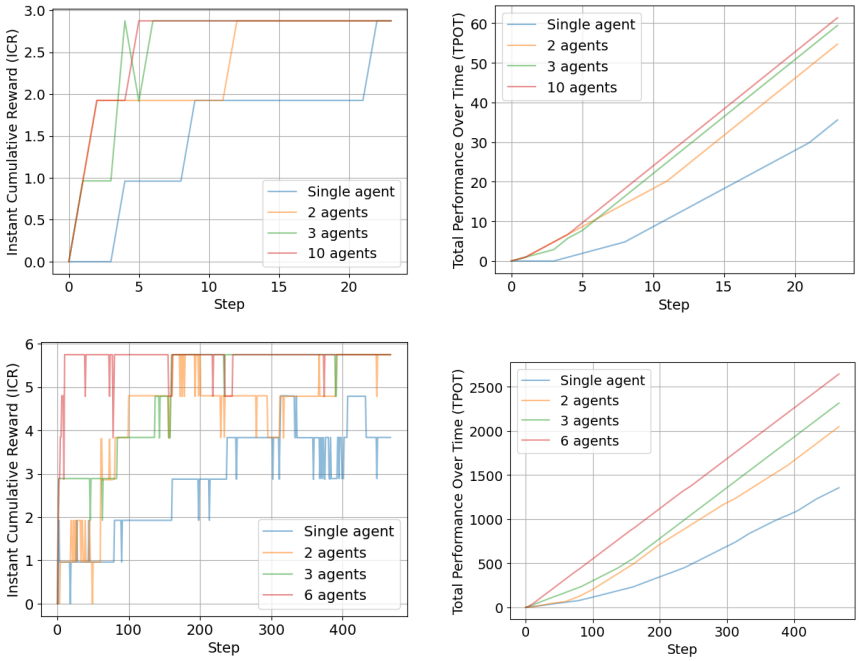

Figure 13: Plots of the instant cumulative reward (ICR) and total performance over time (TPOT) metrics for MiniGrid experiments. The top row consists of the ICR and TPOT of MiniGrid Simple Crossing (3 tasks). The bottom row consists of the ICR and TPOT of MiniGrid Simple Crossing + Lava Crossing (6 tasks).

results can be corroborated using the TPOT plots, which indicate that there is an increase in the total performance as the number of agents increases. In general, ShELL systems outperform their single agent counterparts for these experiments.

Figure 14 shows the ShELL specific metrics for the MiniGrid experiments. The ILA and TLA graphs indicate a similar trend to what is found in the CT-Graph experiments. Figure 13 shows that for the 3-task MiniGrid shows that the 3 and 10 agent achieves close to 3x the cumulative reward gained by a single agent system, with 2 agent system achieving close to 2x the reward. The 6-task MiniGrid plots show that the 6-agent system far outperforms its 3 and 2-agent counterparts, boasting close to 6x improvement in rewards gained. This is further supported by the TLA plots which show a larger

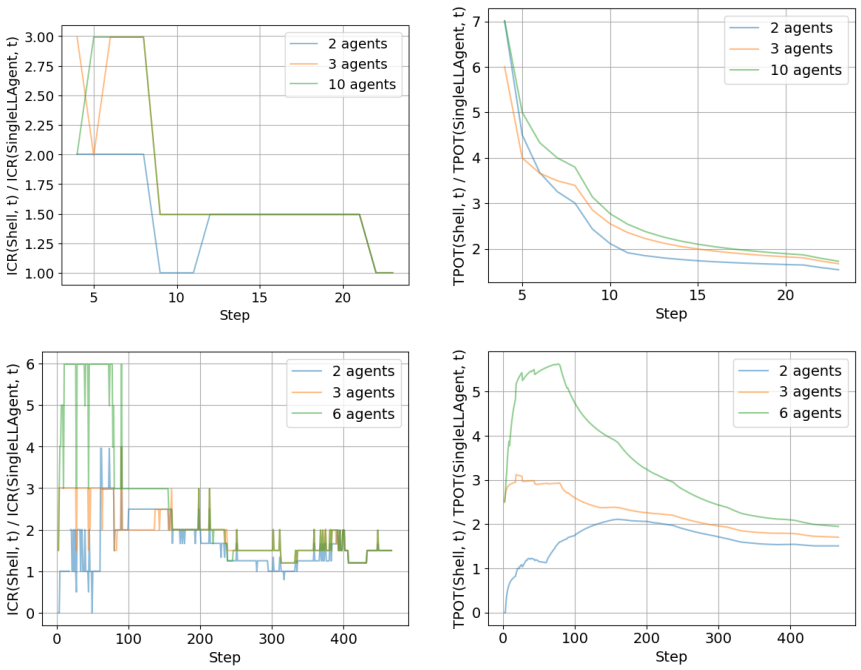

Figure 14: Plots of the instant learning advantage (ILA) and total learning advantage (TLA) ShELL specific metrics. Top row: 3 task MiniGrid. The bottom: 6 task MiniGrid.

immediate performance gain from a ShELL system with more agents. The rewards in these graphs only seem to approach the max ICR, rather than achieve it outright, due to the nature of the reward function in the MiniGrid environment, as noted previously in section 5.

Based on these results it is clear that the $1 + 0.5N$ target is generally achieved. As mentioned previously, the targets given 2, 3, 6 and 10 agents would be 2, 2.5, 4 and 6 respectively. Only in the case of the 10 agent system on 3-task MiniGrid is this not the case. This could be attributed to a sub-optimal task curriculum and should be an area to address in future runs, as it is clear based on the CT-Graph and MiniGrid results that ShELL is able to hit $1 + 0.5N$ despite the lower granularity. It is theorised that with a higher granularity across multiple seeds, ShELL could achieve the $1.3N$ target performance.

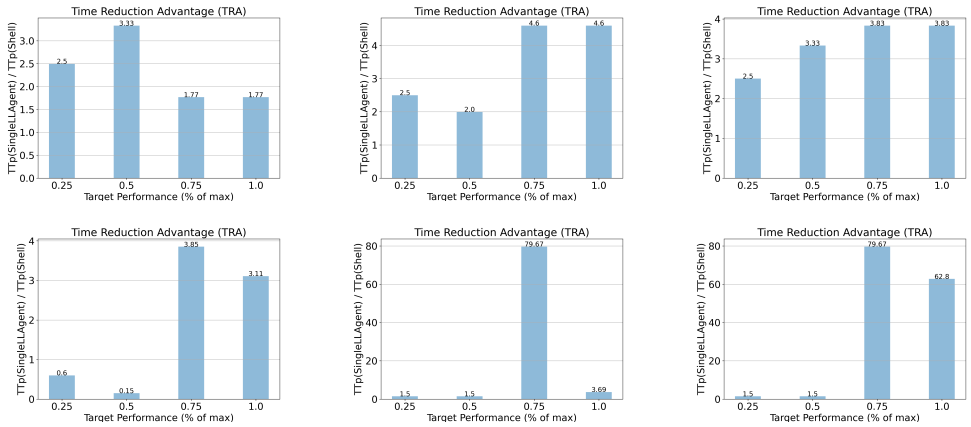

Figure 15: Time reduction advantage scores visualised in bar plot format. Top row: 3 task MiniGrid. Bottom row: 6 task MiniGrid. More agents are able to achieve a higher TRA score, sooner.

Figure 15 shows the TRA scores produced for each of the 3 ShELL systems for the two experiments. For the 3-task MiniGrid experiment, the plots show that there is a clear increase in speed with the 2, 3 and 10 agent systems when compared to a single agent across 25% to 100% of the max ICR. For the 6-task MiniGrid system, the 6-agent ShELL system is 79.67x faster at achieving 75% of the max ICR. Based on these results there is a clear advantage in ShELL in terms of the time required for ShELL to achieve a certain level of performance over a single agent.

These results indicate that the $0.5N$ target is met for 2, 3, 6 and 10 agents, where the target is 1, 1.5, 3, 5 respectively. The exception to this is the 10 agent system on 3-task MiniGrid which is suspected of sub-optimal task curriculum, and the 6-task MiniGrid which is partially skewed towards ShELL. However, taking into account this, the results still show promising results that ShELL is able to perform well.

7 Conclusions and Future Work

7.1 Conclusions

The experiments conducted in this project show clear evidence that confirms the original hypotheses in section 5. The continual learning and ShELL-specific metrics used to evaluate the performance of the ShELL system show similar trends across all experiments, indicating a clear benefit to ShELL over a single continual learning agent. The ICR/TPOT plots indicate an increase in performance as the number of agents increases. The results also show that the SupSup + PPO agent architecture is suited for performing in the continual reinforcement learning setting, and handles the CT-Graph and MiniGrid environments well, if given the right hyper-parameters. The results show that the ShELL systems can successfully learn and reuse knowledge by sharing the task masks, which in turn increases the effectiveness of ShELL at earlier points in the system's lifetime. This trend is further supported by the ILA/TLA metrics which indicate that the ShELL systems in these experiments benefit more as the number of agents tends towards the number of tasks. The ILA/TLA metrics show that as the number of agents surpasses the number of tasks in the environment, the total learning advantage remains.

Unfortunately, the single agent issues with the 27 task CT-Graph and 6 task MiniGrid experiments could not be investigated with reruns in time, thus better results were not available for this report. In future work, it should be investigated whether this is due to the single agent not having enough steps to learn all the tasks in more complex environments, or if it is caused by an underlying bug. Additionally the task curriculum optimally should be an area to consider improving for future experiments.

7.2 Project Evaluation

The project was completed successfully without disruptions or issues. This was in part due to the precautions taken via the risk analysis and project planning conducted in the early stages of the project. Software technologies were specifically used to increase productivity and reduce risks to the experimentation process and a remote GPU server was used to reduce the risk

of loss of experimentation data and increase productivity through reduced training times. A preliminary literature review of the field of continual learning and reinforcement learning was conducted ahead of the official start of the project, to attain an understanding of the various approaches and experimental methodologies. A further literature review was conducted to map the various fields relating to this project and to understand the state-of-the-art works. Meetings were held twice a week with additional research meetings held when necessary.

7.3 Future Areas of Interest and Lessons Learned

In the future, it would be interesting to explore how a fully distributed ShELL system might perform given the same experimental parameters. As the current system linearly trains tasks it is not possible to assess the real-world performance gains of the ShELL system. Additionally, the communication times of sharing the task masks between agents of a fully distributed, the decentralised system would also have to be considered. It would also be beneficial to evaluate the ShELL systems in this project with multiple different seeds to assess the statistical significance of the results produced. A more gradual increase in the number of agents would be beneficial in visualising the benefit of ShELL. Further study should also be conducted into assessing larger-scale systems such as a 27-agent system on the 27-task CT-Graph environment, though this would likely not be feasible with the linear ShELL system used in this project.

The prototype ShELL system explored in this project shares the task masks between agents. Although, in the scope of this project and the environments that were introduced to the ShELL systems, in a large-scale system with hundreds or thousands of agents, eventually the number of masks would become very large. This would result in a growing memory requirement. It would be important to explore whether there is an alternative method to share knowledge or reuse previously learned knowledge to learn new tasks.

Bibliography

Aljundi, R. (2019). Continual learning in neural networks.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2017). Memory aware synapses: Learning what (not) to forget.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula.

Bansal, S., Calandra, R., Chua, K., Levine, S., and Tomlin, C. (2017). Mbmf: Model-based priors for model-free reinforcement learning.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. E. (1954). *The Theory of Dynamic Programming*. RAND Corporation, Santa Monica, CA.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. (2019). Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*.

Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid.

de Witt, C. S., Peng, B., Kamienny, P.-A., Torr, P. H. S., Böhmer, W., and Whiteson, S. (2020). Deep multi-agent reinforcement learning for decentralized continuous cooperative control. *CoRR*, abs/2003.06709.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks.

French, R. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3:128–135.

Golkar, S., Kagan, M., and Cho, K. (2019). Continual learning via neural pruning.

Gronauer, S. and Dieopold, K. (2022). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55:1–49.

Ha, D. and Schmidhuber, J. (2018). World models.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Hemati, H., Schreyer, M., and Borth, D. (2021). Continual learning for unsupervised anomaly detection in continuous auditing of financial accounting data.

Hu, G., Zhang, W., Ding, H., and Zhu, W. (2020). Gradient episodic memory with a soft constraint for continual learning.

Jones, A. and Sprague, N. (2018). Continual learning through expandable elastic weight consolidation.

Kaplanis, C., Clopath, C., and Shanahan, M. (2020). Continual reinforcement learning with multi-timescale replay.

Kaplanis, C., Shanahan, M., and Clopath, C. (2019). Policy consolidation for continual reinforcement learning.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2020a). Towards continual reinforcement learning: A review and perspectives.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2020b). Towards continual reinforcement learning: A review and perspectives.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2016). Overcoming catastrophic forgetting in neural networks.

Ladosz, P., Ben-Iwhiwhu, E., Dick, J., Hu, Y., Ketz, N., Kolouri, S., Krichmar, J. L., Pilly, P., and Soltoggio, A. (2019). Deep reinforcement learning with modulated hebbian plus q network architecture.

Ladosz, P., Ben-Iwhiwhu, E., Dick, J., Ketz, N., Kolouri, S., Krichmar, J. L., Pilly, P. K., and Soltoggio, A. (2021). Deep reinforcement learning with modulated Hebbian plus Q-network architecture.

Lee, H. and Jeong, J. (2021). Multi-agent deep reinforcement learning (madrl) meets multi-user mimo systems.

Li, Z. and Hoiem, D. (2016). Learning without forgetting.

Liang, W., Wang, J., Bao, W., Zhu, X., Wang, Q., and Han, B. (2021). Continuous self-adaptive optimization to learn multi-task multi-agent. *Complex Intelligent Systems*.

Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning.

Mallya, A., Davis, D., and Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights.

Mallya, A. and Lazebnik, S. (2017). Packnet: Adding multiple tasks to a single network by iterative pruning.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

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

Nekoei, H., Badrinaaraayanan, A., Courville, A., and Chandar, S. (2021). Continuous coordination as a realistic scenario for lifelong learning.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning.

Philps, D., Weyde, T., Garcez, A. d., and Batchelor, R. (2018). Continual learning augmented investment decisions.

Raghavan, A., Hostetler, J., Sur, I., Rahman, A., and Divakaran, A. (2020). Lifelong learning using eigentasks: Task separation, skill acquisition, and selective transfer.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan,

K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Sokar, G., Mocanu, D. C., and Pechenizkiy, M. (2021). SpaceNet: Make free space for continual learning. *Neurocomputing*, 439:1–11.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016). A deep hierarchical approach to lifelong learning in minecraft.

van de Ven, G. M. and Tolias, A. S. (2019). Three scenarios for continual learning.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. (2020). Supermasks in superposition.

Wołczyk, M., Zajac, M., Pascanu, R., Kuciński, , and Miłoś, P. (2021). Continual world: A robotic benchmark for continual reinforcement learning.

Xu, J. and Zhu, Z. (2018). Reinforced continual learning.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2017). Lifelong learning with dynamically expandable networks.

Yu, T., Quillen, D., He, Z., Julian, R., Narayan, A., Shively, H., Bellathur, A., Hausman, K., Finn, C., and Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence.

Zhang, S. (2018). Modularized implementation of deep rl algorithms in pytorch. https://github.com/ShangtongZhang/DeepRL.

Zhiyuan Chen, B. L. (2018). *Lifelong Machine Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Claypool Publishers.