Enhancing Generalisation in a dynamic Hierarchical Reinforcement Learning Agent

Alastair Ros-Davison

arosd001@gold.ac.uk

Introduction

In the field of **Reinforcement Learning (RL)**, significant challenges remain in developing agents capable of handling real-world scenarios, where environments are often diverse, dynamic, and unpredictable. One of the major obstacles in achieving this goal is achieving effective generalisation, where an agent is defined as one that can generalise well to novel, unseen situations.

This project focuses on improving an agent's generalisation capability, primarily through the use of **Procedural Content Generation (PCG)**, algorithmically generating diverse training environments, or 'levels'. By training an agent on a wide range of procedurally generated environments, the aim is to improve its ability to adapt to new, unseen scenarios, mitigating overfitting to the training environment. Techniques used for training will also include the addition of convolutional network layers, regularisation, environmental stochasticity, and simplified curriculum learning by way of modifying the PCG process to allow for difficulty scaling.

The project uses a **dynamic Hierarchical Reinforcement Learning (dHRL)** algorithm, which decomposes decision-making into subtasks executed in single timesteps. Experiments will take place in a grid-based, dungeon-crawler-like game with a discrete action and state space, providing a simple yet effective testing platform.
To analyse the impact of generalisation techniques, multiple agents will be trained upon increasing diversity of PCG levels, and a comparison will be made of the performance of agents trained with generalisation methods against a baseline agent trained without them. Performance will be measured using metrics such as average time steps taken, percentage of levels solved, and final score. This evaluation aims to determine the effectiveness of these techniques in enabling agents to generalise and perform well in new and varied environments.

1. Background

Under the umbrella of Artificial Intelligence, Machine learning and the subfield of Reinforcement Learning, have been through booms and 'winters' for over 70 years. With the development of more powerful computers in the 1990's, came a resurgence of interest. The popularity has continued to grow in the last decade with the increased accessibility of deep learning.

In 1997, Tom Mitchell defined the concept of Machine Learning with "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E ." [5] This granular definition shows that the essence of machine learning is in improving upon a particular task, through experience and evaluation of performance.

1.1 Reinforcement Learning (RL)

An RL agent interacts with an environment by making observations, taking actions, and receiving rewards or penalties. Its goal is to learn behaviours that maximise expected rewards and minimise penalties over time. [5] RL is both adaptive, and deals with long-term consequences, making it of interest for solving complex real-world problems. [3]

An RL system consists of four main components: a policy to determine actions, a reward signal, a value function, and optionally, a model. [3]

A policy can roughly be considered as a mapping from perceived states of the environment to actions to be taken within these states. This mapping defines the behaviour of our agent.

The reward signal can be thought of as the goal within a problem. At each timestep as our agent acts within its environment it will receive back from the environment the reward for its action, positive, negative, or null – and our agents seek to maximise reward.

A value function is one that assesses the agent's experiences within the environment and calculates the perceived value of a state based on an estimation of future reward to come.

Finally, a model of the environment describes something akin to a simulation of the environment, or a way to infer behaviour of the environment. It allows an agent a degree of planning within its environment rather than relying solely on trial and error.
Methods for reinforcement learning using models are known as model-based methods, while those without them are known as model-free. [3]

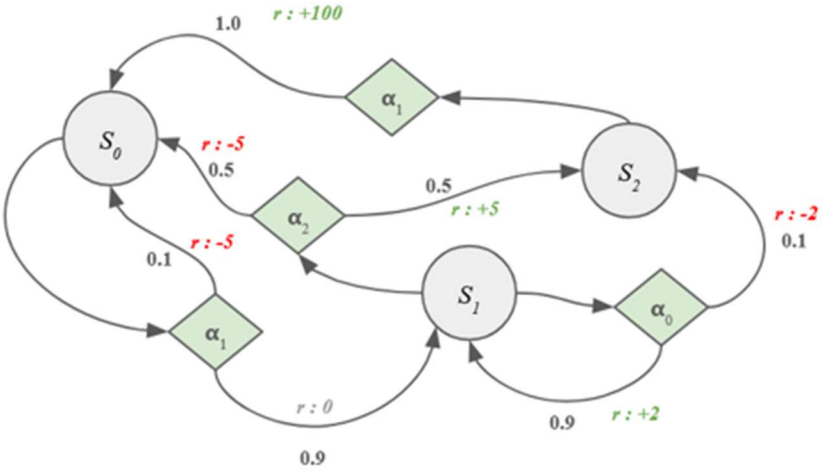
This project will be focusing on a model-free method for the purposes of this project.

1.2 Markov Decision Process (MDP)

A Markov decision process or MDP is a mathematical framework used to model a decision-making problem in which the Markov property is upheld, and outcomes are partly random, partly controlled. The Markov property is present if the next state can be predicted based on the observation of the current state, more specifically no knowledge of the previous state is needed. [3]

MDP's feature a set of states, actions, rewards dependent on state and action, and transition probabilities for states and actions. [8]

Figure 1: A Markov Decision Process



In Figures 1- 5, γ is equal to a discount factor, Q being the Q-Value for a state-action, R for the set of rewards, r for a singular reward, s for states, α for actions, and T being the transition function for states and actions. The MDP in Figure 1 features numbers from 0 to 1 for probability.

1.3 Q-Learning

Q-Learning is an important algorithm within reinforcement learning. It provides a method for converging on an optimal policy within a system defined by an MDP.
Using the Bellman's Q-Value Iteration Algorithm [5], one can estimate the optimal state-action values, otherwise known as (Quality) Q-Values. This is the sum of the discounted future rewards on average after an agent reaches state s and takes action α, assuming optimal future action and no knowledge of the outcome. [5]

Figure 2: Q-Value Iteration Algorithm

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right] \quad \text{for all } (s,a)$$

Figure taken from A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. [5]

Knowing the Q-Values, one can easily find the optimal policy, simply choose the action with the highest Q-Value.

Q-Learning uses a variation of this algorithm - shown below - based on the assumption that Q-Values are initially unknown, and seeks to update them by trial and error as an agent gains experience within the environment. Once it has trained accurate enough Q-Value estimates, the optimal policy is a greedy one, choosing the action with the maximum Q-Value. [5]

Figure 3: Q-Learning Algorithm

$$Q(s,a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} Q(s',a')$$

Figure taken from A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. [5]

Q-Learning *will* converge, but given it does not know transition probabilities or rewards, it can take much hyperparameter tuning, and many iterations to find optimal Q-Values. Q-Learning is an *off-policy* algorithm, as the policy being trained is not necessarily the one being executed – as opposed to a policy gradient algorithm, which is *on-policy*, altering the same policy that is in execution.

1.4 Exploration vs Exploitation

One can make use of the off-policy nature of Q-Learning to aid the training process by using an exploratory policy in execution, rather than one that is simply random. [5] This is the nature of exploration vs exploitation, how often does one prioritise exploring over taking the current optimal action?
An example of this would be the Boltzmann (or softmax) exploration algorithm, in which actions with high expected rewards have a higher chance to be selected.
The figure below describes the exploration strategy, finding the probability of taking an action.

Figure 4: Boltzmann Exploration

$$P(a|s) = \frac{exp(Q(s,a)/T)}{\sum_{a' \in A} exp(Q(s,a')/T)}$$

Figure taken from R. Niel and M. A. Wiering, 'Hierarchical Reinforcement Learning for Playing a Dynamic Dungeon Crawler Game' [20]

T refers to the 'temperature' coefficient, in which higher temperature reduces the difference in chance, while lower temperature favours expected reward. *P* refers to the probability of taking an action in that state, and *A* is the set of actions. [20]

1.5 Deep Q-Network (DQN)

When the possible space of states and actions becomes too large, Q-Learning starts to falter. The approach can be adapted to instead approximate Q-Values for state-actions, rather than rely on exact values (Figure 4).

One way of doing this is through the use of neural networks to find a function that approximates these values using a number of parameters (given by the parameter vector θ). [5] This can be extended to deeper networks, and there has been notable success with the added use of convolutional layers within a network, presumably helpful in determining and extracting features for use within these networks. [18]

Figure 5: Target Q-Value

$$Q_{\text{target}}\left(s,a\right) = r + \gamma \cdot \max_{a'} \quad Q_{\boldsymbol{\theta}}\left(s',a'\right)$$

Figure taken from A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. [5]

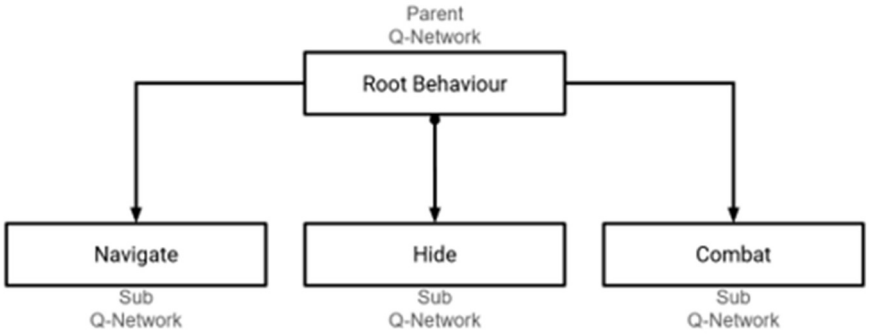
1.6 HRL and d(HRL)

HRL stands for Hierarchical Reinforcement Learning. This is a term for algorithms that seek to decompose a MDP into a hierarchy of smaller MDP's. It is primarily a divide and conquer strategy, where the bottom of the hierarchy contains simple actions, while the parents contain higher level abstractions of behaviour.

This project will be implementing dHRL, proposed by R. Niel and M. A. Wiering [20], where, similar to other HRL systems, the problem is decomposed into smaller MDP's. It consists of a root behaviour, which can then choose from other sub-behaviours. The root behaviour and sub-behaviours have their own reward functions, independent of other behaviours, and can either execute primitive actions or delegate to other sub-behaviours until an action is chosen.

Commonly in HRL systems, sub-tasks can contain arbitrarily long sequences of actions. A key difference with dHRL is that each primitive action in sub-behaviours takes only a single timestep. This allows for dynamic reaction to the environment and less chance of getting stuck in one task, additionally allowing increased focus on a single goal within training.

Figure 6: An example dHRL A.I. behaviour hierarchy



1.7 Modern RL progress

RL has seen impressive results in many settings, from besting human professional performance in complex games with continuous action spaces such as DoTA 2 [4], the board game of Go with its enormous search space and difficulty in evaluating moves and board state [6], or more real-world applications such as autonomous driving [10][11].

But RL still faces many challenges in achieving solutions to the complex and dynamic nature of real-world problems, one of which is generalisation.

2. Overview of Generalisation

Reinforcement learning agents are prone to overfitting their environments, [13] and this leads to an inability to deal with new situations. This is especially important to tackle, as the real world is dynamic and complex [11], and agents need to be able to deal with diverse situations and not fail catastrophically in the face of the unseen.

Transfer learning takes knowledge from a previously learned task to learn a new target task quicker than if there was no transfer at all [14], for example from one game to another [22].

Domain adaptation is a specific type of transfer learning, where the task may remain relatively the same, but the scope is domain shifts, such as from a simulation to the real world, and aiming to achieve meaningful generalisation to this other setting. [10][15]

Generalisation is similar in concept to transfer learning and domain adaptation, but contained within the same setting, and looking for a robust performance on contextually different, previously unseen, instances of the same task.

This task is not an easy one, and there is a great chance that a model will still overfit to a considerable extent, even when using techniques to combat this, as in the case of the CoinRun benchmark [16].

3. Key Techniques of Improving Generalisation in RL

When looking to teach an agent to adapt, the goal is to teach the underlying shapes of the problem, the behaviour that must be replicated across a range of contexts, and to that end there are a few popular techniques seen in contemporary research.

3.1 Convolutional Neural Network (CNN)

CNN's have been very successful in practical application, extracting meaningful features from data that exists in a grid-like format, notably visual input. [9]
They have seen success in game-based RL agent applications because of their usefulness within continuous domains [9][16].

The aim of the project is to generalise across environments, where the layouts of the levels feature paths, shapes, that could possibly be extracted as features for training rather than an entire grid of cells. These features could potentially allow for the training of a policy that is able to better abstract the shape of the level and avoid overfitting to specific layouts.

3.2 Data Augmentation

Data augmentation introduces artificial variance in data, commonly used with CNNs to reduce overfitting in visual tasks. For visual data, randomising colours and textures can improve robustness and performance both in settings within the same task and when adapting to new domains [15].

The same idea can be applied to good effect within environments working with physics simulations. By varying the dynamics of a task, one can better bridge the reality gap and train policies capable of adapting to unfamiliar dynamics when applied to real world situations [19].

Ultimately the concept is "if the variability in simulation is significant enough, models trained in simulation will generalise to the real world with no additional training" [15].

3.3 PCG (Procedurally Generated Content)

PCG can be utilised to randomise the setting within a domain, providing varied training environments, such as different game levels. This is shown to correlate with positive effects on the agent's capability to generalise to new settings within the same domain [16].

PPCG (Progressive Procedurally Generated Content), is an adaptation of PCG, where the difficulty of the generated level is increased as an agent learns [17]. Experiments implementing this showed that "dynamically adapting the level difficulty during training allows the agent to solve more complex levels than training it on the most difficult levels directly." [17]

There are adjacent algorithms designed to feature both 'teacher' and 'student' agents, such as the ACCEL algorithm [2], inspired by POET [21]. These pair a student agent with a dynamically edited environment to present increasing challenges, helping to teach complex behaviours. In the case of ACCEL, Kruskal's algorithm was used to create mazes of increasing complexity. [2]

While PCG is a popular and useful technique, environments should also include other factors of controllable variation. [1]

3.5 Batch Normalisation, Regularisation and Stochasticity

When working with neural networks and deep reinforcement learning, there are additional techniques employed inspired by supervised machine learning. Dropout and L2 regularisation saw some success within the CoinRun benchmark experiments [16].

However one of the findings within this paper was that while these techniques saw some success, they worked best when used together, and may still be addressing the underlying causes of poor generalisation.

Batch normalisation is seen to have a substantial regularising effect within the realm of supervised learning [12], and was also employed to great effect within the CoinRun environment, normalising between CNN layers. [16]

Using probability distribution to explore the action-space instead of taking the preferred action can lead to significantly improved performance down the line [20], both in performance of models trained in one environment or reducing generalisation error, with a notable example being the epsilon-greedy algorithm [16], or Boltzmann exploration [20][Figure 4].

4. Contribution

In R. Niel's and M. A. Wiering's paper detailing their novel architecture for a dHRL algorithm [20], they note that it would be of interest to see further work on observing if the generalisation capability of the agent could be improved through use of multiple levels to train on, chosen randomly each epoch.

This project will attempt to reimplement the dHRL algorithm and train multiple agents in a PCG environment with increasing diversity of PCG levels, as well as utilising various techniques, such as batch normalisation, regularisation, simplified curriculum learning, and the addition of CNN's.

A simplified dungeon crawler environment, and a modified Kruskal's algorithm will be used to create mazes, both perfect [2] and non-perfect [7]. Further modification will be used to alter production of levels and create sparser environments and increase difficulty artificially when developing for a simplified curriculum learning approach.

By doing this, the hope is to provide insight into the effect training upon diverse environments can have with a dHRL agent. Additionally, the effect of combining multiple common generalisation techniques will be investigated. Performance will be measured using metrics including average time steps taken, percentage of levels solved, and final score.

REFERENCES:

[1] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, 'A Survey of Zero-shot Generalisation in Deep Reinforcement Learning', *jair*, vol. 76, pp. 201–264, Jan. 2023, doi: 10.1613/jair.1.14174.

[2] A. A. J. P.-H. O. U. M. J. UCL, M. A. M. D. U. B. M. S. UCL, M. A. J. F. O. U. E. G. UCL, M. A. T. R. UCL, M. A. *Equal contribution P. March 3, and 2022 ARXIV Full paper, 'Evolving Curricula'. Accessed: Nov. 08, 2024. [Online]. Available: https://accelagent.github.io

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts London, England: The MIT Press, 2015. Accessed: Nov. 13, 2024. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[4] OpenAI *et al.*, 'Dota 2 with Large Scale Deep Reinforcement Learning', Dec. 13, 2019, *arXiv*: arXiv:1912.06680. doi: 10.48550/arXiv.1912.06680.

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, UNITED STATES: O'Reilly Media, Incorporated, 2019. Accessed: Nov. 13, 2024. [Online]. Available: http://ebookcentral.proquest.com/lib/goldsmiths/detail.action?docID=5892320

[6] D. Silver *et al.*, 'Mastering the game of Go with deep neural networks and tree search', *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[7] M. Ihsan, D. Suhaimi, M. Ramli, S. M. Yuni, and I. Maulidi, 'Non-perfect maze generation using Kruskal algorithm', *J. Nat.*, vol. 21, no. 1, Feb. 2021, doi: 10.24815/jn.v21i1.18840.

[8] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[9] Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016. Accessed: Nov. 14, 2024. [Online]. Available: http://www.deeplearningbook.org

[10] S. Wang, D. Jia, and X. Weng, 'Deep Reinforcement Learning for Autonomous Driving', May 19, 2019, *arXiv*: arXiv:1811.11329. doi: 10.48550/arXiv.1811.11329.

[11] J. Levinson et al., 'Towards fully autonomous driving: Systems and algorithms', in 2011 IEEE Intelligent Vehicles Symposium (IV), Jun. 2011, pp. 163–168. doi: 10.1109/IVS.2011.5940562.

[12] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', Mar. 02, 2015, *arXiv*: arXiv:1502.03167. doi: 10.48550/arXiv.1502.03167.

[13] C. Zhao, O. Sigaud, F. Stulp, and T. M. Hospedales, 'Investigating Generalisation in Continuous Deep Reinforcement Learning', Feb. 20, 2019, *arXiv*: arXiv:1902.07015. doi: 10.48550/arXiv.1902.07015.

[14] M. E. Taylor and P. Stone, 'Transfer Learning for Reinforcement Learning Domains: A Survey', *Journal of Machine Learning Research*, vol. 10, no. 56, pp. 1633–1685, 2009.

[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, 'Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World', Mar. 20, 2017, *arXiv*: arXiv:1703.06907. Accessed: Nov. 13, 2024. [Online]. Available: http://arxiv.org/abs/1703.06907

[16] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, 'Quantifying Generalization in Reinforcement Learning', Jul. 14, 2019, *arXiv*: arXiv:1812.02341. Accessed: Nov. 10, 2024. [Online]. Available: http://arxiv.org/abs/1812.02341

[17] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, 'Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation', Nov. 29, 2018, *arXiv*: arXiv:1806.10729. doi: 10.48550/arXiv.1806.10729.

[18] V. Mnih *et al.*, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[19] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, 'Sim-to-Real Transfer of Robotic Control with Dynamics Randomization', Mar. 03, 2018, *arXiv*: arXiv:1710.06537. doi: 10.48550/arXiv.1710.06537.

[20] R. Niel and M. A. Wiering, 'Hierarchical Reinforcement Learning for Playing a Dynamic Dungeon Crawler Game', in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, Bangalore, India: IEEE, Nov. 2018, pp. 1159–1166. doi: 10.1109/SSCI.2018.8628914.

[21] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, 'Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions', Feb. 21, 2019, *arXiv*: arXiv:1901.01753. doi: 10.48550/arXiv.1901.01753.

[22] V. Mnih *et al.*, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.