UNIVERSITY OF LEEDS

Exploring Curriculum Learning

Bakhtiyar Khan, Jack Auty, Luke McMahon, Fu Chan, Zhouyang Wang

Submitted in accordance with the requirements for the degree of MEng. Computer Science

2020/21

30 credits

The University of Leeds

Faculty of Engineering

School of Computing

April 2021

The candidate confirms that the following have been submitted:

Item	Format	Recipient(s) and Date
Report	PDF Version	Minerva 27/04/2021
Code Base	Git Repository	Gitlab 27/04/2021

The software code can be viewed at:

Type of Project: Exploratory Software

The candidate confirms that the work submitted is his/her own and that appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Signed: Bakhtiyar Khan, Jack Auty, Luke McMahon, Fu Chan, Zhouyang Wang

Acknowledgements

We would like to thank our indispensable project supervisor Dr. Matteo Leonetti of the University of Leeds for his continued provision of direction throughout the project, and for establishing a chain of communication between us and a key collaborator Andrea Bassich.
We would also like to thank Andrea Bassich of the University of York, for his excellent guidance on the implementation of the adaptive progression function used within this project.

Summary

The widely adopted approach to human education is organised in a hierarchical fashion, such that 'easier' content - generally content considered to be logically prerequisite to later content - is taught initially, prior to more 'difficult' content. Within this system, over the course of a students' education within a field they are gradually exposed to its more complex content, allowing them to apply previously learned concepts as required to build a more thorough and complete understanding of that field. This approach to learning is often formalised under the broader concept known as a 'Curriculum', and we will be adapting and applying this 'Curriculum Learning' approach for use within the Reinforcement Learning paradigm. Specifically, we will be exploring its use within a modified version of an OpenAI environment utilising the MuJoCo physics engine [1]. We found that through the application of curriculum learning techniques, the speed to convergence, and performance of the agent improved massively over traditional reinforcement learning approaches.

Contents

Chapter 1

Introduction

A key goal within the Artificial Intelligence field is developing a fully autonomous agent with the ability to learn optimal behaviour via interaction with the environment. One approach to this is reinforcement learning, a sub-field within the Artificial Intelligence domain, that has been in development over several decades, with many key advancements. The idea behind reinforcement learning is simple and is essentially a formalised trial and error, where a entity uses feedback from the environment to maximise a reward. The use of Reinforcement Learning algorithms has gained prominence for the purpose of training agents to navigate complex environments, especially those in which the desired outcome is clear but the optimal action sequence is uncertain.

A novel development in the reinforcement learning paradigm has been the application of an approach known to be effective in human education, the use of curricula; the employment of which spawned the reinforcement learning sub-field of curriculum learning. Curriculum learning at its core is about gradually and appropriately increasing the difficulty of the environment and ensuing tasks provided to the agent such that the model is optimally challenged, similarly to the way in which a human agent (student) is taught and assessed. For example, a student would be taught and assessed on arithmetic before algebra.

Our hypothesis follows that, in our exploration of this concept, we will observe a substantial increase in agent performance in comparison to that of traditional reinforcement learning - where the agent is challenged at maximum complexity throughout.

This chapter will discuss aims and objectives of the project, deliverables, planning and risk mitigation, as well as the general structure of the report.

1.1 Aims

The main aim of this project is to confirm the hypothesis above, assessing the merits in performance of its use in comparison to the standard Reinforcement Learning approach. Here, agent performance is measurable as the number of time-steps taken to learn the final task, moreover what degree of success this final model achieves in the environment at maximum complexity. Through the substantiation of this hypothesis we hope to highlight the significance of curriculum learning as a pivotal conception in the evolution of the field of reinforcement learning as a whole. This will be accomplished by running experiments within the environment we develop – quantifying the difference in speed to convergence observed, and through the subsequent evaluation of its comparative optimality.

Furthermore, we aim to provide an appraisal of various modes of curriculum generation across environments of increasing complexity. To this end, a hierarchy will be established from the

divergence in outcomes exhibited by fixed and adaptive progression functions. Where a progression function is "for calculating the complexity of the task given to the agent at any time", and therefore can be considered to be the generator of the curriculum [2].

1.2 Objectives

- Implement an environment-appropriate agent that is capable of learning a complex task utilising a curriculum learning approach.
- Implement various curriculum generation methods and evaluate the comparative performance between them.
- Demonstrate the value of curriculum learning techniques within this environment via the juxtaposition of their performance versus the traditional reinforcement learning approach.

1.3 Deliverables

The material that will be delivered by the end of the project is the following:

- A comprehensive report detailing the research undertaken, the implementation achieved and the subsequent analysis of the results generated, along with an evaluation of this.
- A git repository containing code for the environments and training.

1.4 Project Plan

The Project Plan can be split into phases, with some overlap in timing:

1. Planning
2. Experimentation
3. Background Research
4. Implementation
5. Analysis of results
6. Evaluation
7. Finalisation of Report

1.5 Project Methodology

The project will follow an Agile approach. The requirements will constantly evolve throughout the project, and by adapting an iterative approach we can ensure that we are flexible and able to meet new requirements. A backlog will be maintained, with priorities assigned to each item, such that the most important components take precedence. We aim to run the ubiquitous two-week sprints, however this is flexible and can be adjusted as necessary.

1.6 Structure of the Report

The structure of the report is roughly as follows:
Chapter 1: A brief introduction giving an overview of the project.
Chapter 2: Background Research into all related topics
Chapter 3: Implementation Details specific to the project including dependencies and environment specifics.
Chapter 4: Analysis of our results in comparison to traditional approaches
Chapter 5: Evaluation of our methodology and implementation and the conclusions we have reached

Chapter 2

Background Research

This chapter details the research we have undertaken for the project in order to make the problem more clear and formalised. It also contains state of the art methods that may be useful in this project and a brief explanation on any terminology that we used in this report.

2.1 Machine Learning

Artificial Intelligence is an exceedingly popular field in computing currently and comprises three forms: weak artificial intelligence, strong intelligence and super artificial intelligence. Machine Learning is the main representative of weak intelligence. The first definition of machine learning was cognitive computing, a field of study that allows computers to learn without explicit programming, as specified by Arthur Samuel (1959) of IBM in his research paper [3]. However this definition is somewhat vague, and the term was later re-defined more specifically by Tom Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [4]. Principally, machine learning refers to an advanced computing technique which allows computers to seek and extract knowledge from a sea of data. It could be regarded as an evolution of typical computer programming.
Broadly the purpose of this is to discover the features of the training data-set provided to the model, as to facilitate accurate predictions on future provided data - usually that of a similar structure. This is especially important when applied to large data-sets, as often such a model is able to uncover characteristics that no human agent could reasonably comprehend. In order to ensure that the features extracted from the provided data are accurate, it is essential that the appropriate machine learning methodology is applied. This, as determined by the structure of the provided training data; be that labelled examples in the case of supervised learning, unlabelled examples in unsupervised learning, or Markov decision processes for reinforcement learning. These can be considered as the three main machine learning paradigms, and are described in the following sections of this report.

2.2 Reinforcement Learning

The Reinforcement Learning (RL) paradigm, which exists between supervised and unsupervised learning, consists of learning within sequential decision making problems in which feedback is limited [5]. There are several standard terminologies within the field including Agent: the entity that takes actions, Environment: the space in which the agent performs the actions, Reward: Each action produces a reward, positive or negative, with which the agent aims to maximise the cumulative reward, known as Expected Return.

Generally, an agent is able to survey its environment and from this decide upon the course of action that it needs to take in order to reach the desired goal. "Although the designer sets the reward policy − that is, the rules of the game − he gives the model no hints or suggestions for how to solve the game" [6]. For model-free algorithms, characterised in section 2.2.3, an agent can perform actions and receive immediate reward, but they are not told which actions to take, nor their consequences - they must deduce this through trial and error. A choice is made between potential actions based on their expected reward value as perceived by the agent across prior instantiations of the environment − the goal of the agent is to maximise the total reward value accumulated by the end of the simulation.

The agent-environment interaction defines a task. There are two types of reinforcement learning task, either the task takes a finite amount of time and has a set number of terminal states - an episodic task, or it lasts forever - a continuous task [7].

2.2.1 Markov Decision Processes

A reinforcement learning task can be formalised as a Markov Decision Process (MDP). A MDP is a stochastic decision process in which the next state and reward depend only on the current state. For this report, we solely look at episodic MDPs [8].

An episodic MDP is a 6-tuple (S, A, p, r, S_0, S_f), where S is the set of states, A is the set of possible actions, p is the transition function $p(s'|s, a)$ which gives the probability of the next state s' will be transitioned into, given that action a is taken in state s. $r(s, a, s')$ is the reward function which gives out the expected reward when taking action a in state s and transitioning into state s'.(needs to paraphrase)

S_0 is the set of initial states and S_f is the set of terminal states.

Agents take actions by passing the environment its current state and action, which returns a new state and reward. The choice of action is determined by the policy. A policy is essentially the agents strategy, and can be defined formally as $\pi : S \times A \to [0, 1]$. For each state $s \in S, \pi$ is the probability distribution over $a \in A(s)$. This means that if an agent follows policy π at timestep t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. The goal of the agent is to find optimal policy $\pi^\star$ such that it maximises cumulative reward.

2.2.2 Current State of Reinforcement Learning

Many different areas of research exist within the reinforcement learning community. One such area, the playing of games, has long been a focus of research; with many notable achievements occurring in the last decade. Board games are commonly used within this domain, and their use can be traced back to 1956 when Arthur Samuel demonstrated a machines' ability to learn the game of Checkers [9]. Games are naturally suited for reinforcement learning since they embody a set of rules within which an agent can be trained, or that can be learned if starting tabula rasa, and from each game state manifest many potential series of moves which eventually lead to a reward for the player - generally either winning or losing. Furthermore, since this reward is generally not provided to the agent until the end of the game, they promote the inception of

long-term strategy. Models can be trained against human players initially in order for the agent to learn the basic mechanics of the game, and then subsequently against separate instances of themselves allowing for increasingly rapid development of strategy and technique.
In 2016 a significant milestone was achieved in that AlphaGo, a computer Go agent developed by Google DeepMind, defeated an 18-time world champion Go player, becoming the first ever such program to do so [10]. At the time, such a feat was thought by many experts to be at least a decade from fruition. Go can be considered a simple game due to having few rules, however the incomprehensible size of its search space had made it an intractable problem for traditional AI approaches - with the search space being approximately 10^{170}, a number greater than the number of atoms in the Universe [11] [12] - highlighting the significance of the achievement. Since 2016, DeepMind have developed new iterations, each better than its predecessor, with the latest spawning a general reinforcement learning algorithm, named AlphaZero, which achieves superhuman performance in many different games [13].
Deep reinforcement learning is a combination of deep learning and reinforcement learning, and is the driver behind AlphaZero and many other successful game playing programs. General algorithms show promise in many different applications. OpenAI Five, originally developed to beat human players in Dota 2, has been used to teach a robot hand dexterity. More impressively is that the training for this was performed across simulations, before being transferred to real world application successfully [14]. Future work is likely to use this approach, since reinforcement learning alone struggles to deal with the high-dimensional MDP states which are required to represent many increasingly complex real-world problems.

2.2.3 Reinforcement Learning Algorithms

Reinforcement learning algorithms can be categorised based on the algorithm being model-based or model-free, which can be seen in Fig. 2.1 [15]. Model-based algorithms have explicit models of the environment which consist of the transition and reward function. To be specific, the model can learn the transition possibility T from the current state s0 and action a to next state s1 (T(s1—s0,a)). When the model learns the transition possibility successfully, the agent knows the probability of entering a specified state from the information about the current state and action. This model can be used to provide predictions about the dynamics of the environment to the agent and the agent can use these predictions to plan ahead, learning optimal policy indirectly. This model can be explicitly programmed, such as the rules of Chess, or it can learn whilst the agent interacts with the environment. If the model is available, then algorithms outperform those without a model significantly, as seen with AlphaZero [13]. Typically, the ground-truth representation of the environment is not available, and so the agent is provided with an approximation of the model. In this case, it is possible the optimal policy may never be found. Models may also be biased, which can result in the agent performing well within the learning environment, but poorly in the real environment.
However, model-based becomes impractical as both of the state space and action space grow. Model-free algorithms learn the optimal policy by adopting a trial and error approach. The algorithms possess no information about the dynamics of the environment, and so learn the optimal policy directly through experience, which is gained by taking actions. Hence, the space

for storing all the combinations of action and state is not essential. It is better to choose model-free algorithms for reinforcement learning.

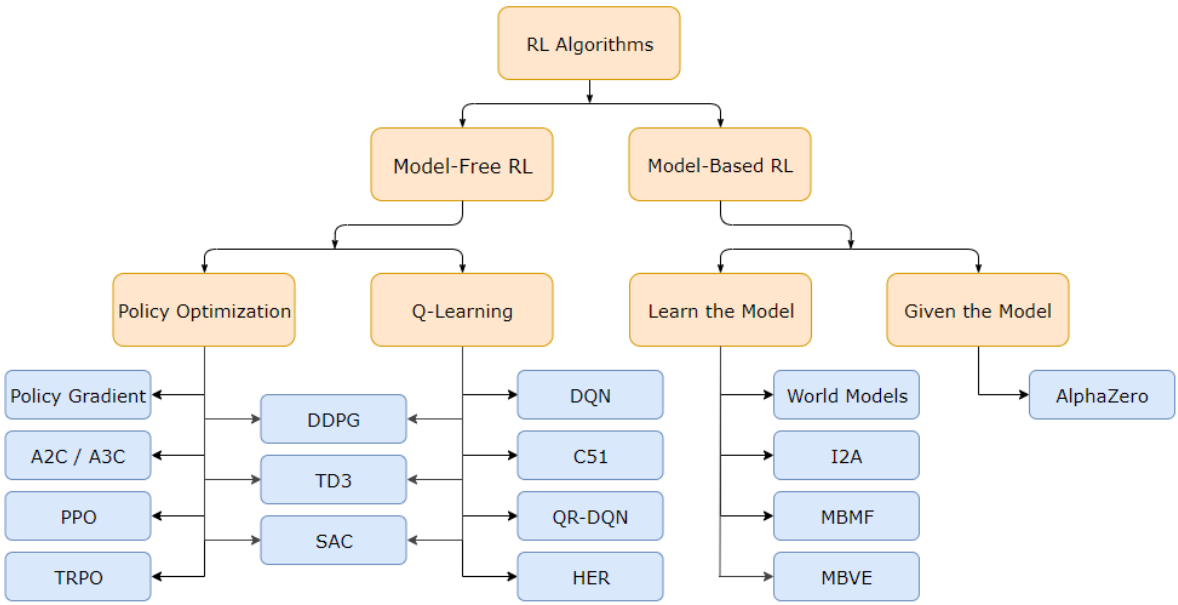

Figure 2.1: A taxonomy of reinforcement learning algorithms [15]

Reinforcement learning algorithms can also classified as on-policy and off-policy, the main difference between which is that the agent policy of on-policy algorithm updates based on the reward value of current action A, while it updates independently from the action taken, usually based on the reward value of the local optimal greedy action A, for an off-policy algorithm.

The optimisation is performed off-policy in Q-learning, which means that the training data utilised can be the data at any time during the training process [15]. Q-learning is based on Bellman Equation. The figure shows the Bellman expectation equation, where E represents expectation and is the discount factor.

$$v(s) = \in [R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \text{ [16]}$$

In Q-learning, Q is action-utility function, that is used to evaluate which action to take from a certain state. After learning, a Q-table should be generated, which is storing every state and action. The largest expected reward can be found from the optimal actions in each state throughout the Q table. The figure below shows the Bellman optimality equation to find Q value.

$$Q^*(s,a) = \in \left[r_{t+1} + \gamma \max_{a'} Q^*\left(s_{t+1}, a'\right) | s_t = s, a_t = a \right] \text{ [17]}$$

The optimisation is performed on-policy in Policy optimisation, which means the data generated by the latest policy is used when the policy is updated each time [15].

2.2.3.1 Proximal Policy Optimisation

The proximal policy optimisation algorithm is one improved from the policy gradient, an algorithm based on policy iteration. It directly collates the states, with the actions and their rewards, required in order to achieve to maximum expected cumulative reward. According to the documents of OpenAI, policy gradient algorithm is to "push up the probabilities of actions that lead to higher return, and push down the probabilities of actions that lead to lower return, until you arrive at the optimal policy" [18]. The equation of gradient policy is deduced as:

$$\nabla\overline{R_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R\left(T^n\right)\nabla logp\left(a_t|s_t,\theta\right)$$

Firstly, we define T as the sum of one single round, as $T = s_1, a_1, r_1, \ldots, s_T, a_T, r_T$.
Therefore, $R\left(T\right)$ is the sum of rewards, as $R\left(T\right) = \sum_{t=1}^{T} r_t$.
To maximize it, $\overline{R_\theta} = \sum_T R\left(T\right) P\left(T|\theta\right) \approx \frac{1}{N}\sum_{n=1}^{N} R\left(T^n\right)$ can be deduced.
Now, seeking gradient of $\overline{R_\theta}$, an equation of $\nabla\overline{R_\theta}$ can be found as:

$$\nabla\overline{R_\theta} = \sum_T R\left(T\right) P\left(T|\theta\right)\cdot\frac{\nabla P\left(T|\theta\right)}{P\left(T|\theta\right)} = \sum_T R\left(T\right) P\left(T|\theta\right)\cdot\nabla logP\left(T|\theta\right) \approx \frac{1}{N}\sum_{n=1}^{N} R\left(T^n\right)\cdot\nabla logP\left(T^n|\theta\right)$$

and $P\left(T^n|\theta\right)$ can be expanded as the following equation:

$$\mathrm{P}\left(T^n|\theta\right) = p\left(s_1\right)p\left(a_1|s_1,\theta\right)p\left(r_1,s_2|s_1,a_1\right)p\left(a_2|s_2,\theta\right)\ldots p\left(a_t|s_t,\theta\right)p\left(r_t,s_{t+1}|s_t,a_t\right)$$

$$= p\left(s_1\right)\prod_t p\left(a_t|s_t,\theta\right)p\left(r_t,s_{t+1}|s_t,a_t\right)$$

Then putting the equation into $\nabla logP\left(T^n|\theta\right)$:

$$\nabla\log P\left(T^n|\theta\right) = \nabla\log\left(P\left(s_1\right)\prod_t p\left(a_t|s_t,\theta\right)p\left(r_t,s_{t+1}|s_t,a_t\right)\right)$$

$$= \nabla\log p\left(s_1\right) + \sum_{t=1}^{T}\nabla\log p\left(a_t|s_t,\theta\right) + \sum_{t=1}^{T}\nabla lp\left(r_t,s_{t+1}|s_t,a_t\right)$$

$$= \sum_{t=1}^{T}\nabla\log p\left(a_t|s_t,\theta\right)$$

Finally,

$$\nabla\overline{R_\theta} \approx \frac{1}{N}\sum_{N}^{1} R\left(r^n\right)\cdot\nabla logP\left(t^n|\theta\right)$$

$$= \frac{1}{N}\sum_{n=1}^{1} R\left(r^n\right)\sum_{t=1}^{T_n}\nabla logp\left(a_t|s_t,\theta\right)$$

$$= \frac{1}{N}\sum_{n=1}^{1}\sum_{t=1}^{T_n} R\left(r^n\right)\nabla logp\left(a_t|s_t,\theta\right)$$

Essentially, $R(T^n)$ is multiplied with the minimized cross entropy of the actions of sampling N rounds and actions of network output, and the reward value is the gradient descent now. However, we still can improve it for each state and operation of the tuple, therefore, $R(T^n)$ should be substituted as the following:

$$R(T^n) \to \sum_{t=t'}^{T_n} \gamma^t r_t^n$$

And $\nabla\overline{R_\theta}$ changed as well as:

$$\nabla\overline{R_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{t'}^{T_n}\gamma^t r_t^n \nabla logp(a_t|s_t,\theta)$$

There still exists an overestimation problem. The state-action sampling is insufficient in practice, which means that some actions or states would not be sampled. When gradient descent training is in process, these actions or states would be zoomed in or out extremely. To improve it, we can input baseline. It would be a constant hyper-parameter to be adjusted, also known as Critic, which is a network waiting to be trained.
If critic is used, this model is called Actor-Critic Model and $\nabla\overline{R_\theta}$ changed as:

$$\nabla\overline{R_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} A^\theta(a_t|s_t)\nabla logp(a_t|s_t,\theta)$$

However, the parameter updating of policy gradient is slow because re-sampling is needed every time the parameters are updated. To improve the training speed and efficiently use the sampled data, we do importance sampling to change the on-policy training process to off-policy. Importance sampling is the transformation process from on-policy to off-policy. Assume we have a continuous random variable x , and the probability density function of it is $p(x)$, the expectation of $f(x)$ calculated by the following equation:

$$E_{x\sim p}[f(x)] = \int f(x)\,p(x)\,dx$$

If there is another probability density function $q(x)$. It has such relationship with the above equation:

$$E_{x\sim p}[f(x)] = \int f(x)\cdot p(x)\,dx = \int f(x)\frac{p(x)}{q(x)}\cdot q(x)\,dx = E_{x\sim p}\left[f(x)\frac{p(x)}{q(x)}\right]$$

$\frac{p(x)}{q(x)}$ is the importance weight. In the case of $\nabla\overline{R_\theta}$, $f(x)$ is $A^\theta(a_t|s_t)$, and $\frac{p(x)}{q(x)}$ is the ratio of the probability of the current action taken in current state between new and old policy. Therefore, in the case of sufficient sampling, the equation should be:

$$E_{x\sim p}[f(x)] = E_{x\sim p}\left[f(x)\frac{p(x)}{q(x)}\right]$$

Since importance sampling provides the transform of training process from on-policy to off-policy, it is possible to sampling sufficiently from old policy $q(x)$ and improve the new policy $p(x)$ afterward. This process can be repeated N times in one round. This greatly reduces the time of sampling state-action-reward tuple in original policy gradient algorithm. Finally, the gradient of the average reward value for N rounds is changed as well, the new $\nabla\overline{R_\theta}$ is:

$$\nabla\overline{R_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\frac{p_{\theta(a_t|s_t)}}{p_{\theta'(a_t|s_t)}}A^\theta(a_t|s_t)\nabla logp(a_t|s_t,\theta)$$

In practice, there exists a clip on $\frac{p_\theta(a_t|s_t)}{p'_\theta(a_t|s_t)}$:

$$clip\left(\frac{p_{\theta(a_t|s_t)}}{p_{\theta'(a_t|s_t)}}, 1-\epsilon, 1+\epsilon\right)$$

, where ϵ is adjustable hyperparameter.

Generally, PPO relies on "specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy" [19]. It controls the ratio of the new and old policy $r_t(\theta)$ to prevent the impact on the learning effect of agent by the changes of updating. Clip can limit the ratio $r_t(\theta)$ between $(1-\epsilon, 1+\epsilon)$.
When the advantage is positive, $\hat{A}_t > 0$, the current policy is better, optimization can be intensified. If $r_t(\theta) > 1-\epsilon$, clip will still be $1-\epsilon$ to prevent over-optimization.
When the advantage is negative, $\hat{A}_t < 0$, the current policy is worse, optimization should be reduced. If $r_t(\theta) < 1-\epsilon$, clip will still be $1-\epsilon$, but the Min function would still choose smaller value for $r_t(\theta)$.

2.3 Transfer Learning

Transfer learning refers to notion of the reuse of a model developed and trained to solve one task in order to solve another similar, yet distinct, task. To be more specific, according to the research of deep learning by Ian Goodfellow in 2016 [20], "transfer learning and domain adaptation refer to the knowledge learned in one environment being used in another domain to improve its generalisation performance". Therefore, transfer learning is not a form of classic machine learning; it is rather more related to multiple-task learning and concept drift. Its use is prevalent in solving deep learning problems, for example, in the case of training a deep model that requires a large number of data sets for pre-training.

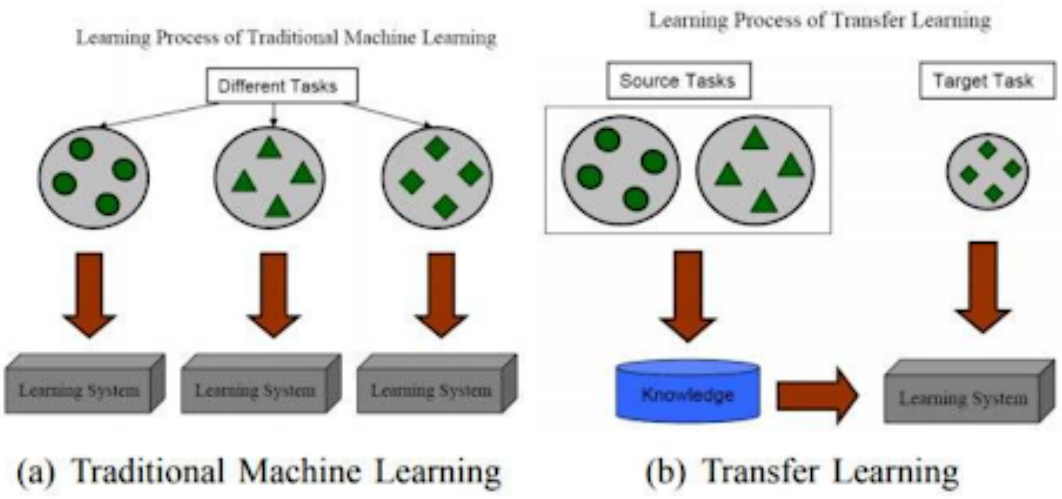

Figure 2.2: The difference between traditional machine learning and transfer learning

The Fig. 2.2 demonstrates the difference between traditional machine learning and transfer learning. For traditional machine learning, there are multiple learning systems generated during the learning process since the tasks are interpreted separately. However, the knowledge accumulated in some elemental tasks should be put into practice as a basis for the current learning task in transfer learning.

Furthermore, transfer learning can be used within reinforcement learning to improve performance. Reinforcement learning is based on environmental feedback, as delineated in section 2.2, and there are two main disadvantages to this. One is that when state-action space (SxA) of reinforcement learning is large, the searching process for seeking the optimal strategy is highly time-consuming. The other drawback is that the cost of retraining the agent is huge when failure occurs on the previous learning stage, which is often due to changes in the problem-space. To improve reinforcement learning, transferring knowledge from the source task to the target task could be invaluable, and this is a key concept used to build the idea of curriculum learning from reinforcement learning.
The improvement of learning speed is just one thing which transferring learning can achieve [21]. The complexity of a learning activity can be derived from the amount of episodes necessary for the model to achieve the desired performance. There are plenty of ways to measure the improvement of learning speed including the time to convergence to some threshold and also ratio of area. To be more specific, we can observe the effect of transfer learning within a domain by setting a threshold of policy transition and monitoring the amount of experiences needed for both a single-task and transfer learning algorithm to reach that threshold.

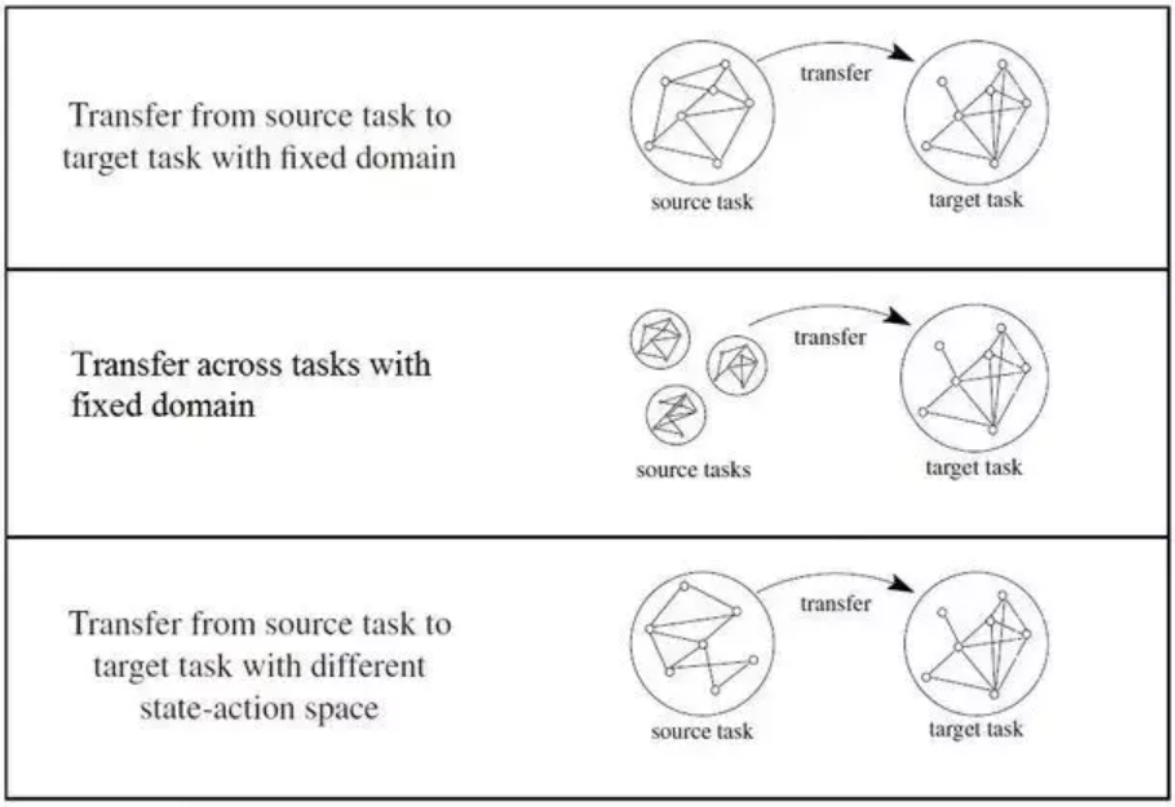

Figure 2.3: Three ways in which transfer learning is traditionally applied within reinforcement learning

The three ways in which transfer learning is traditionally applied within reinforcement learning are shown in Fig. 2.3, and explained below [22]:

- Transfer from source task to target task with fixed domain – The fixed task domain is determined by the state-action (SxA), but the specific structure of the task and the target is decided by state transition model T and reward function R.
- Transfer across tasks with fixed domain – The tasks are sharing the same domain. The algorithm is taking the knowledge collected from a source task as input and using it to improve the performance in the target task.
- Transfer from source task to target with different state-action space – Tasks contain different state-action space, transfer methods would mainly focus on defining mapping between source state-action variable and the target variable to ensure effective transfer.

2.4 Curriculum Learning

Curriculum learning is an approach to reinforcement learning inspired by human development. In a human learning domain, when learning a new skill, whether a sport or musical instrument or an academic subject, the training process is structured to present new concepts and tasks in a sequence that leverages what has previously been learned.

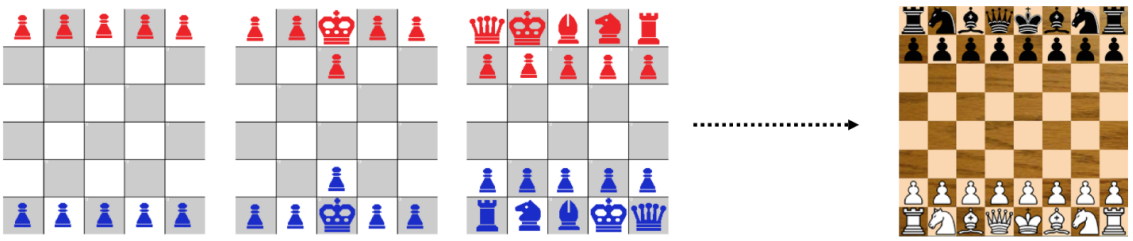

Figure 2.4: Figure 1 from paper [8] - "Different subgames in the game of Quick Chess, which are used to form a curriculum for learning the full game of Chess."

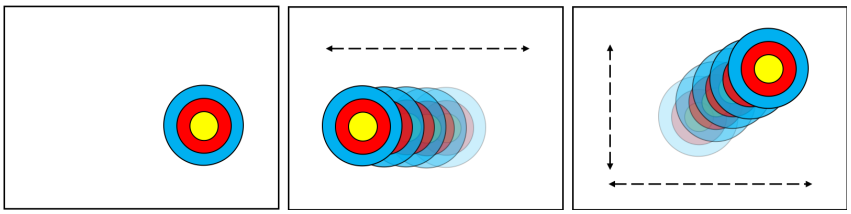

Figure 2.5: Archery targets with a stationary target, a horizontally moving target, and a horizontally and vertically moving target respectively.

Consider the example in Figure 1 from paper [8], repeated here in Fig. 2.4. Quick chess is a game designed to introduce children to the game of full chess by exposing them to increasingly difficult subgames of chess. In the first subgame, the player learns how pawns move, promotions and taking opposition pieces. The second introduces a new piece, the king, as well as a new objective, to keep the king alive. Consecutive subgames introduce new elements, a larger board or a new piece for example, which require learning new skills and building upon what has already been learned before finally tackling the full game of chess.

Consider also another example in Fig. 2.5. Initially, the archer aims to hit the bulls-eye on a stationary target. The difficulty is then increased by introducing a new element, horizontal motion and the archer now has to hit a moving target. Lastly, vertical motion is introduced, and the archer has to hit a target which can move in two planes. Like in the example of quick chess, new skills are acquired by building upon what has already been learned. However, unlike in quick chess, here the objective of the archer is the same in all cases, but the change in the behaviour of the target is what introduces differing levels of difficulty.

Using curricula to train agents in reinforcement learning has been considered from the early 1990s such as in the field of grammar learning [23] and robotics control problems [24] as well as classification problems later on in 2009 [25]. Results showed that the order of training matters and generally, learning algorithms perform better when training tasks are ordered in increasing difficulty. Overall, in curriculum learning, starting from small and simple tasks and gradually increasing the difficulty can lead to more optimal results within a task as well as faster convergence. Recent development in reinforcement learning has demonstrated how agents can utilise transfer learning [26] [27] to reuse knowledge acquired from a source task when attempting to learn a target task.

The inspiration for a curriculum learning approach has been provided. Formal explanations are

yet to be provided such as, what a curriculum is exactly and how tasks are generated. There are many ways to define what a curriculum is, with the most basic definition being that a curriculum is an ordering of tasks. Although, a curriculum does not necessarily need to have a simple linear sequence. A single task can build upon knowledge acquired from multiple source tasks similarly to how a course in an academic subject may require multiple prerequisites.

Definition: Curriculum. Let T be a set of tasks where $m_i = (S_i, A_i, p_i, r_i)$ is a task in T, the tasks are modelled as MDPs. Let D^T be the set of all possible transition tasks from tasks in T, $D^T = \{(s, a, r, s') \mid \exists m_i \in T \; s.t. \; s \in S_i, a \in A_i, s' \sim p_i(\cdot|s, a), r \leftarrow r_i(s, a, s')\}$. A curriculum $C = (V, E, g, T)$ is a directed acyclic graph, where V is the set of vertices, $E \subseteq \{(x, y) \mid (x, y) \in V \times V \wedge x \neq y\}$ is the set of directed edges and $g : V \rightarrow \mathcal{P}(D^T)$ is a mapping function which associates vertices of the graph to subsets of tasks in D^T, and $\mathcal{P}(D^T)$ is the power set of D^T. A directed edge (u, v) in C implies that tasks associated with $u \in V$ should be trained on before tasks associated with $v \in V$. All paths terminate on a single sink node $v_t \in V$ [8].

Definition: Task-level curriculum is a curriculum C in which each vertex is associated with a single task in T. Therefore, the mapping function is $g : V \rightarrow \{T\}$. In a task-level curriculum, the tasks form the nodes in the graph C [8].

Definition: Sequence curriculum is a curriculum C where the indegree and outdegree of each vertex v in the graph C is at most one with exactly one source node and one sink node [8].

Combining task-level and sequence simplification produces a task-level sequence curriculum. Such a curriculum can be represented as an ordered list of tasks $[m_1, m_2, \ldots, m_n]$. An example of a task-level sequence curriculum as an acyclic graph can be seen in Fig. 2.6. A visual example is seen in Fig. 2.7. Compare from the curriculum in Fig. 2.8, in which the final task has two source tasks from which it can learn from and transfer knowledge from these two tasks. The experimental setting in this paper uses a simplified task-level sequence curriculum explained in Section. 3.2.2.2.

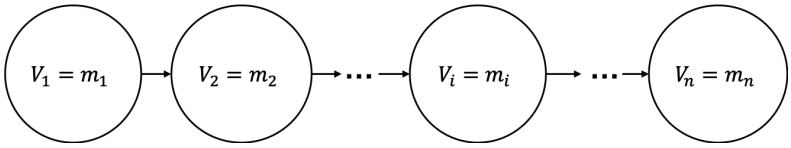

Figure 2.6: A curriculum graph of a task-level sequence curriculum.

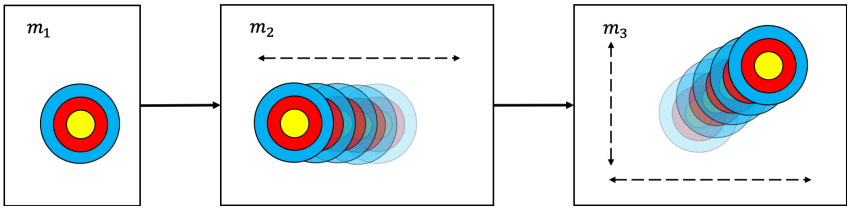

Figure 2.7: A task-level sequence curriculum graph of the bullseye example in Fig. 2.5

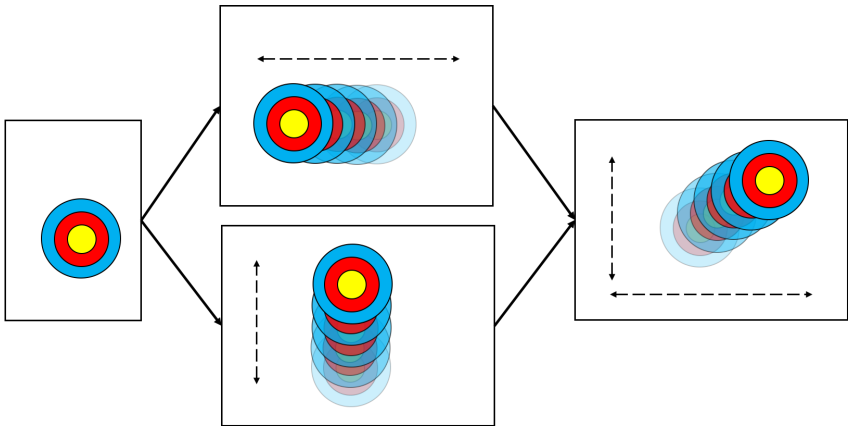

Figure 2.8: A task-level non-sequence curriculum graph of the bullseye example in Fig. 2.5, with multiple source tasks

2.4.1 Task Generation & Sequencing

Task generation is the process of creating a good set of intermediate tasks from which to obtain valuable experience. These tasks may be predetermined or dynamically generated during the curriculum construction by observing the agent's behaviour and performance. Sequencing considers how to generate the edges of the curriculum graph. In a task-level curriculum, this means that sequencing refers to the way in which the tasks are ordered. In manually designed curricula, human intervention selects the ordering of tasks. This is the traditional and more common approach although recently automated methods for sequencing are being explored. These steps are important in determining a good quality curriculum as the quality of the curriculum is important to achieve success.

Definition: Curriculum learning is a methodology to optimise the order in which experience is accumulated by the agent, so as to increase performance or training speed on a set of final tasks [8].

2.4.2 Progression Functions

The purpose of a progression function is to determine the how the complexity factor changes as progress is made throughout the curriculum, put simply how much more difficult each intermediate task should be than the previous one assigned to the agent [2]. Fixed progression functions predetermine this difficulty increase per time-step, while an adaptive progression function is one where this increase per step varies - depending on the performance of the agent on the previous step.

2.4.2.1 Fixed Progression

As stated, with a fixed progression function, the complexity increase at every time step is established prior to run-time, based on the function implemented. The two fixed progression functions outlined in [2] for which this increase is determined distinctively are linear progression functions and exponential progression functions.

The linear progression function takes only the time-step at which progression should complete (t_e) as input, was the first to be implemented. Here, the complexity factor is determined linearly [2]:

$$\Pi_l(t, t_e) = max(\frac{t}{t_e}, 1)$$

It follows from this equation that the complexity factor is at its lowest at the initial time-step, t_0 at 0, and at its highest at the last time-step t_e at 1, increasing linearly between the two.

Subsequently, an exponential progression function was devised, where the difficulty at each time step increases in an exponential fashion. In [2] this was devised to take an additional parameter, s, which influences the slope of the progression. The complexity factor is determined by an exponential progression function as:

$$\Pi_e(t, t_e, s) = 1 - max(\frac{\alpha - \beta}{1 - \beta}, 0) \tag{2.1}$$

where,

$$\alpha = e^{-\frac{t}{t_e * s}}, \quad \beta = e^{-\frac{1}{s}}$$

It follows from this equation that the complexity factor is at its lowest at the initial time-step, t_0 at 0, and at its highest at the last time-step t_e at 1, increasing exponentially per time-step between the two [2]. Fig. 2.9 shows how the complexity factor changes over time for different values of the parameter s. Smaller positive values show initially steeper gradients of increasing complexity. It is interesting to note that a value of $s = 2$ defaults to a linear progression. Thus, in the implementation, only the exponential progression function needs to be implemented to implement both the exponential progression function and the linear progression function.

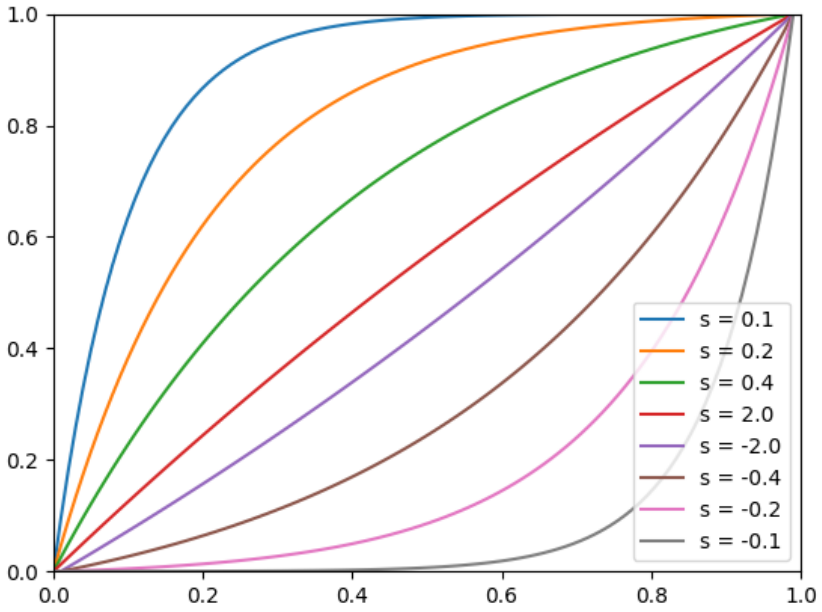

Figure 2.9: Complexity factor change over time with Exponential progression function for different parameters of s.

2.4.2.2 Adaptive Progression

Conversely, with an adaptive progression function, the increase in complexity with each successive task is determined based on the online performance of the agent, that is the performance on the previous task. The performance of the agent is calculated at each time-step throughout the task using a performance function, whose contents differ based on the context and the domain. For example, a performance function based on cumulative reward would progress complexity factor if the agent passes a provided threshold of accumulated reward.

The progression function defined in [2] was named 'Friction-Based progression' and uses a performance function that takes inspiration of physics, namely the friction between a body and the plane to work out what the speed of the object s_t, should be at any given time. Friction-Based progression determines the complexity factor based on the time step, the previous complexity, and a parameter to determine the 'friction' or resistance which considers the agent's performance [2]:

$$\Pi_f(t, c_{\text{t-1}})) = 1 - Uniform(s_t, s_{\min}) \tag{2.2}$$

"The core concept behind this progression function is to slow down the body when the agent is improving, resulting in an increase in difficulty" [2]. Generally, an adaptive progression function attempts to optimise the time taken to learn each task of increasing difficulty, by increasing the complexity in proportion to the performance of the agent on the previous task.

2.4.3 Mapping Functions

The other core part of the curriculum learning framework outlined in [2] is the mapping function. The mapping function maps a complexity factor determined by the progression function to a task from the set of all possible tasks in the domain, which the agent will train in. This mapping function Φ defines the specifics of environment using the complexity value at a given time-step [2]:

$$M_t = \Phi(c_t)$$

where M_t refers to the task at time t. By working out $\mathrm{M_t}$ at each time-step, we are effectively selecting the tasks added, and the order of these tasks, to the curriculum. This results in the curriculum as defined [2]:

$$C = \langle M_0, ..., M_e \rangle$$

.

Chapter 3

Implementation

3.1 Dependencies

3.1.1 OpenAI Gym

OpenAI Gym is an open source toolkit that provides various environments, from basic environments such as classic control theory problems to more complex environments including Atari games. Gym was developed to provide standardised benchmarks for the reinforcement learning community to use, and allows the development of reinforcement learning algorithms, as well as hosting scoreboards for algorithm benchmarks. Gym focuses on episodic reinforcement learning, where the agent repeatedly interacts with the environment for a predetermined number of episodes. Within an episode, the agent performs a series of actions until reaching a terminal state. Common interfaces are provided for the environments, but agents are left to the developer, allowing them to implement their own style of agent interface [28].

3.1.2 MuJoCo - Multi-Joint dynamics with Contact

Multi-Joint Dynamics with Contact (MuJoCo) physics engine is a commercial general purpose simulator which enables simulations of complex dynamic systems, written in C to achieve the best performance [4]. MuJoCo has been adopted by the reinforcement learning community for algorithm benchmarks, gaining traction when the OpenAI Gym interface was released. Since MuJoCo is commercial, the project required acquiring licenses which were hardware-bound, restricting the machines we could perform the simulations on. MuJoCo was selected as the physics engine over alternatives by necessity as a requirement for the other listed dependencies, as well as by convenience of being what was used in the work that inspired the project.

3.1.2.1 MuJoCo-py

MuJoCo-py is a Python interface developed by OpenAI that enables use of MuJoCo with Python3. Since the project makes use of Python dependencies such as OpenAI Gym, it requires the MuJoCo-py bindings to enable development.

3.1.2.2 MuJoCo-Worldgen

MuJoCo-Worldgen is an interface developed by OpenAI that allows users to generate complex, random environments. The interface provides a set of methods that allow seeded world generation, and can be extended to add various objects. Objects and world parameters, such as floor size, allow changing the environment to increase difficulty within a curriculum.

3.1.3 OpenAI Baselines

OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms, developed by OpenAI with the intention of creating a baseline on which the community can build upon. A vital part of research is the ability to replicate results, and by ensuring the algorithms are implemented in the same way it prevents results from being affected by unintentional bugs or untuned algorithms. This allows results to be compared fairly and progress to be met with less scepticism.

3.1.3.1 Stable Baselines

Stable Baselines is a fork of OpenAI Baselines that improves upon the latter by focusing on simplicity and consistency. A lack of comments and consistency within the code, as well as minimal documentation, made the initial version complicated to work with. Stable Baselines aimed to fix this with improvements including: a unified structure for all algorithms, additional algorithms and easy to understand documentation. More importantly, code coverage is much higher, ensuring results can be trusted more.

3.2 Experimental Setting

This section describes the environment used to test the benefits of a curriculum learning approach in reinforcement learning.

3.2.1 OpenAI Multi-Agent Emergence Environment

The environment developed and used in this paper is inspired by the complex multi-agent environment developed by OpenAI (Bowen, Baker and Markov, Todor) in [29] as described in paper [1]. This environment was chosen as a basis it possesses a large action space and mutable difficulty index. It consists of multiple self-supervised hider and seeker agents who can to use aspects of the environment to their advantage in order to fulfil their respective goals. The seekers' goal is to keep the hiders within their vision, while the hiders' is to avoid the line of sight of the seekers. It is also possible to generate different objects such as walls and move-able ramps and boxes. These can be interacted with by the agents and so the environment consists of a large action space. Furthermore, complexity is introduced through the constraint to partial observation for the agents, and long-term strategies are encouraged to the point of requirement by way of delayed rewards.

3.2.1.1 Limitations/Difficulties

OpenAI omitted to include the code used to define the architecture and train their network specific to the environment. This was an intentional choice to leave it to the developers to implement their own architecture and use reinforcement learning algorithms of their choice. The agent policies OpenAI implemented, but did not include, used two separate networks using two different reinforcement learning algorithms with decentralised execution and centralised training [1]. The difficulty of such an implementation is beyond the scope of this paper. The

computing power and resources required to simulate an environment with multiple agents and a large action space were unavailable to the authors of this paper.

3.2.2 Our Environment

Due to the reasons mentioned above regarding difficulty with implementing a complex architecture for the multi-agent environment and a lack of computing power, OpenAI's environment has been simplified to a single-agent environment with a limited action space. The simplified environment used in this paper consists of a single agent (the seeker) and a target (see Fig. **??**) – only the behaviour of the seeker will be dynamic as determined by the algorithm, the behaviour of the target will be fixed at run-time following from the difficulty index specified. There are no interactable objects within the environment. The actions available to the agent are: modifying speed in the x-y plane and rotation in the z-axis. This means that the agent is allowed to move freely in the environment.

Modifications made to OpenAI's environment were accomplished by sub-classing the Base class of the multi-agent emergence environment. Additional functionality via custom methods were added to incorporate modifiable behaviour within the environment during the training process for the purposes of implementing curriculum learning. Moreover, the generation of the environment has been overwritten so that the random generation of different environments for consecutive episodes are now deterministic, unlike in OpenAI's implementation. This allows for reproducible tests when comparing results acquired via simulation from different progression functions in curriculum learning as well as a non-curriculum learning approach.

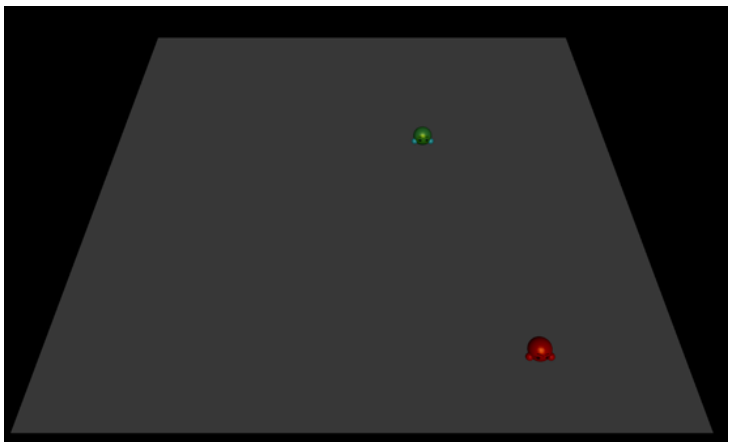

Figure 3.1: Visual representation of environment developed for this paper

3.2.2.1 Reward

The objective of the agent is to find the target or get close enough to the target. This is true for all tasks in the curriculum which is described further below. A simple reward function has been used to achieve. The agent receives a reward of 1 if and only if the agent is close enough to the target, when the Euclidean distance between the agent's and the target's position are below a threshold value. The reward is 0 otherwise. The episodes terminate upon the agent finding the target and receiving a reward of 1, if the agent is out of bounds (see Fig. 3.1) or

after 100 time-steps in the environment.

3.2.2.2 Curriculum

As described in Section. 2.4, a curriculum consists of a sequence of increasingly difficult tasks. Here we consider a task-level sequence curriculum. There are a number of ways to modify complexity. Within the environment, complexity is found into factors: the speed of the moving target and the size of the floor / environment. Considering these factors of complexity, the easiest task corresponds to a very small environment in which there is little room for the agent and target to move in and so it will be easier for the agent to find the target. Moreover, a stationary target is also an easier task for the agent. Therefore, a task of zero difficulty consists of a stationary target in the smallest possible environment. What determines the smallest possible environment has been determined by experimentation by the authors.

Conversely, the most difficult task in the curriculum consists of the target moving randomly within the environment at maximum speed, in the largest possible environment, which must be found by the agent. From the easiest task, tasks of increasing difficulty correlate to increasing speed of the randomly moving target as well as the floor size. Note that while the objective of all tasks in the curriculum are the same for the agent, to find the target, the behaviour of the target changes and this makes each task in the curriculum different and distinct. The behaviour of the environment via changing floor sizes increases the number of states the agent can explore, so this also makes each task in the curriculum different and distinct.

3.2.2.3 Mapping Function

From the degrees of complexity considered, it follows that the mapping function (see Section 2.4.3), which maps the current complexity value in the curriculum provided by the progression function to a task, will alter two parameters in the environment: the speed of the moving target and the size of the floor. In this case, the mapping function modifies each of these two parameters in a monotonically increasing manner. Here the mapping function modifies the parameters linearly. The mapping function $\Phi(c_t)$ maps to a task M_t where the domain of a task M is the Cartesian product of the two parameters in the environment, the speed of the target and the size of the floor, namely $S \times F$ where $S = [0, 0.89]$ and $F = [2, 24]$. These bounds for the domain are designed with respect to the environment. Hence, the mapping function is of the form:

$$M_t(s, f) = \Phi(c_t) = (0.89c_t,\ 22c_t + 2)$$

3.2.2.4 Other Possibilities

It is possible to consider complexity through other means. For example, introducing obstacles in the environment, such as walls or ramps, or limiting observation to only have partial observation, such as lidar vision limitation in OpenAI's multi-agent environment. It is also possible to change the objective of the agent to introduce complexity, such as moving to the target and then moving back to the starting position – this would test the agent's long-term memory. For the purpose of this paper and demonstrating the power of curriculum learning through a simple curriculum,

two degrees of complexity have been considered, the speed of the moving target and the size of the floor.

3.2.3 Parallel Execution

It is possible to train with multiple instances of an environment at the same time with a parallel implementation of PPO (see Section 2.2.3.1), the learning acquired from the multiple instances is collected before updating the policy. This allows for faster training. In this case, each instance of the domain has its own independent progression function. The interval between the instances can be changed so that agents training on different instances can have different complexities and be training on different tasks according to the mapping function. With this parallel execution, an interval range needs to be defined so that the different instances can have independent progression functions with different rates of growth [2].

For example, consider the Exponential progression function defined in (2.1) with parallel execution of training with 2 processes. It is possible to have two different parameters s_1 and s_2 for two independent Exponential progression functions for two instances of the environment which the two agents will train in. For example, a small, steep value of $s_1 = 0.1$ and a linear value of $s_2 = 2$ (see Fig. **??**), while the first environment will increase in complexity faster, the second is more stable and this can help balance training and converge to an optimal policy faster. the aforementioned points are one of the reasons the authors decided to use PPO as the learning algorithm, due to the benefits of this parallel implementation.

Chapter 4

Runtime Experiment

We carry out a runtime experiment with different progression functions and no progression functions, that is, the standard reinforcement learning approach without a curriculum. The aims of the experiment are outlined as well as the strategy of the experiment will be executed, explaining and justifying the decisions made. The results of the experiment are analysed and compared to initial expectations from our theoretical understanding of curriculum learning and whether the practical experimentation confirms or denies our hypothesis. The experiments have been run on a machine with ...

4.1 Aim

The main aim of this project is to explore the benefits of curriculum learning. Thus, the aim of this runtime experiment is to test whether curriculum learning does aid in training the agent to converge faster and more optimally than training without a curriculum.

4.2 Strategy

The agent is trained for the same number of total time-steps with an adaptive progression function, a fixed progression function and no progression function. The adaptive progression function used is the Friction-Based function, which is the same as the one described in (2.2). A parallel variant of the algorithm is used with 4 processes and an interval of 10000 time steps between processes to have a range of complexities within the 4 independent Friction-Based progression functions.

The fixed progression function used is the Exponential function, which is the same as the one described in (2.1). Similarly to the Friction-Based progression, the parallel variant is used with 4 processes and different values of the parameter s for the different processes with $s \in \{0.1, 0.73, 1.37, 2\}$ for each of the processes respectively.

No progression function refers to training without any progression function. This is training the agent on the hardest level difficulty task for the entire duration of training. This can be considered as training on a curriculum of one task with the task corresponding to the maximum complexity factor, that is the task $M_t = (0.89, 24)$ for all t (see Section 3.2.2.3). The training is executed with parallel PPO with 4 processes.

With a batch size of 10880, the agent is trained for 100 epochs. This is repeated for 15 iterations, the average performance is given for each of the three methods of training and the deviation in performance also. Repeating the training for multiple iterations increases the reliability of our results.

4.3 Results

The results of the runtime experiment can be seen in Fig. 4.1 where the performance of the agent measured as the probability of successfully completing an episode and finding the target, is plotted against the time the agent has been training.

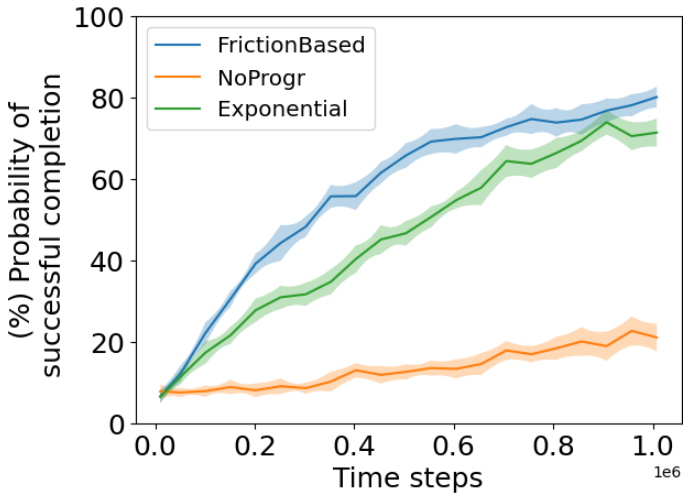

Figure 4.1: Agent performance over duration of training with Friction-Based progression function, Exponential progression function and no progression function.

The change in complexity over time is given for the two progression functions in Fig. 4.2 and Fig. 4.3. Observe that there are 4 lines present in each figure, each of the lines corresponds to an independent progression function used to train individual instances of the environment as a result of parallel execution. In the case of the Exponential function, the lines from left to right correspond to the values of $s \in \{0.1, 0.73, 1.37, 2\}$ respectively. Likewise for the Friction-Based progression, from left to right the lines represent independent progressions with different intervals in increasing order.

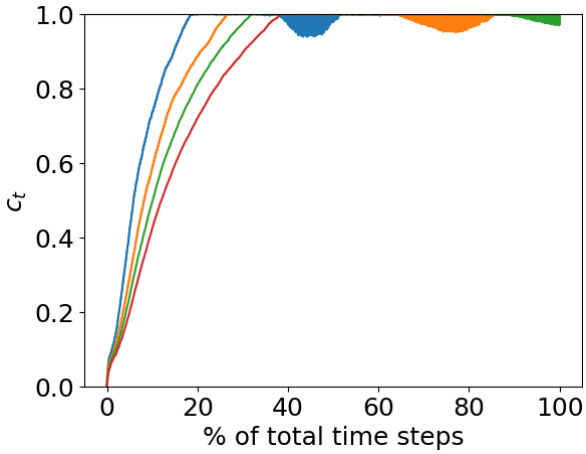

Figure 4.2: Complexity change over duration of training with Friction-Based progression

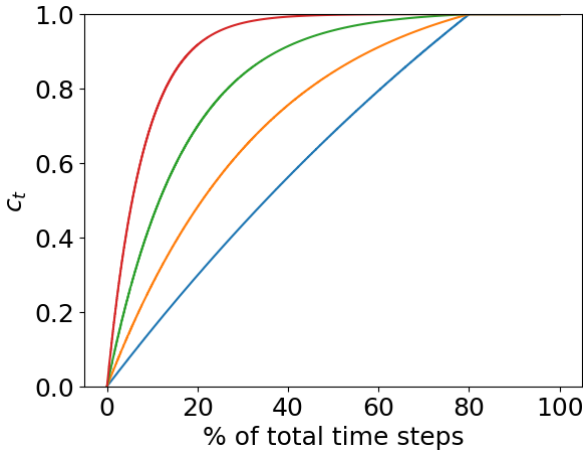

Figure 4.3: Complexity change over duration of training with Friction-Based progression

4.4 Analysis

From Fig. 4.1, it is clear to see that curriculum learning with both progression functions outperforms training without a curriculum. All methods show improvement in the agent's performance over time as expected. However, both Friction-Based progression and the Exponential progression train the agent to a more optimal policy in a much shorter time span. This is evident in the steeper gradients of the progression functions performance in comparison to training without a curriculum. The most optimal policies obtained from curriculum learning are significantly better than the policy obtained from training without a curriculum by the end of the training period. This can be seen by looking at the probability of the agent successfully completing an episode for each of the 3 methods at the final time step of training. Friction-Based progression performs the best with the agent likely to complete an episode with $\sim$80% of the time, followed by $\sim$70% for the Exponential progression and $\sim$18% without curriculum learning. This is a significant increase in performance. Considering the results of the runtime experiment, the initial hypothesis proposed has been proven to be true.

We can also compare the performance of the two progression functions from the test results in Fig. 4.1 as well as discuss the change in complexity factor over time in Fig. 4.2 and Fig. 4.3. From Fig. 4.1, it is clear to see that Friction-Based progression outperforms the Exponential progression, particularly during the middle portion of training and they begin to show signs of converging to a similar performing policy by the end of training. From our understanding of the progression functions (see Section 2.4.2), the Friction-Based progression function considers the performance of an agent when it determines the complexity the agent should be challenged with. This contrasts the Exponential progression function which is a fixed progression function and the curriculum it generates is predetermined based on its given parameters. It is because of this reason that the Friction-Based progression outperforms the fixed Exponential progression.

This is evident from the dynamic change in complexity over time with the Friction-Based progression seen in Fig. 4.2. Clearly we see less stable lines than for the Exponential progression in Fig. 4.3. More importantly, the initial gradient of the curve is much steeper showing that the Friction-Based progression has determined that the initial tasks in the curriculum are very easy and that the agent can move onto more difficult tasks much sooner than the Exponential progression allows. This explains the difference in performance evident in Fig. 4.1. While the Exponential progression does continuously increase the performance of the agent, the Friction-Based progression has also trained the agent through similar tasks in its curriculum, but much faster, and has already been training the agent on the maximum difficulty task for much longer. Thus, the policy obtained by Friction-Based progression is more optimal.

Chapter 5

Evaluation and Conclusions/Outlooks

This section includes our evaluation of the methodology followed, our implementation, the deliverables and also includes our final conclusion along with the declaration of any ethical concerns that needed to be addressed.

5.1 Evaluation of methodology

The methodology we aimed to follow was agile, following two-week sprints. Unfortunately, this plan deteriorated very early on. A key principle of Agile methodologies is frequent delivery of working software. Development was irregular, often taking place in bursts, rather than spread out evenly through a two-week sprint. Poor communication slowed development, and at times stalled it entirely. However, it should be noted that following regular two-week sprints is difficult alongside additional assignments, often taking away significant time from the project. Another issue we faced was timezone differences. This made communication much slower.

5.2 Evaluation of implementation

Our implementation fell short of the original planned scope of the project due to a variety of reasons, the most major of which was due to the missing necessary code in the initial code base that we decided to work from. This made it impossible to meet our original plan and meant that we had to reduce the complexity of the environment and thus the task assigned to the agent massively. Instead, we first implemented a simple environment with a single agent and a simple task, find the target - after which we focused mainly on the implementation of curriculum learning algorithms which was a feat of its own.
The complexity of the environment that we initially implemented through the mapping function solely controlled the speed of the target, but later we managed to tie in the floor size of the environment in with this too. Perhaps the most significant drawback that we faced was that we had to overhaul the entire code base within the last week of the project, in order to implement the friction-based progression function in the correct fashion - since our prior, naive implementation simply approximated what we were aiming to achieve rather than accomplishing it directly. Our current implementation now correctly implements regular reinforcement learning, fixed progression with both linear and exponential progression functions, and adaptive with the friction-based progression function.

5.2.1 Future Work

As discussed in section 3.2.2.4, there is potential for a diverse body of future work. On the implementation side, work introducing more complex environments and tasks would be beneficial

towards proving our hypothesis. Environment complexity can be increased to demonstrate how curriculum learning has significant improvements over regular approaches. Adding objects that agents can manipulate potentially introduces unexpected or unintended behaviours, like those seen in OpenAI's multi-agent environment [30]. Furthermore, environment complexity could be augmented through the introduction of a restricted observation space for our agent, static environment features such as walls, and multiple agents.

5.3 Deliverables

We have managed to deliver a considerably comprehensive report covering all the research that we did, our implementation and subsequent analysis of our results. While also including an evaluation of these aforementioned sections.
Furthermore, the git repository, which can be found in Appendix A contains the complete code base for the project, missing only the external dependencies mentioned in Appendix B.

5.4 Conclusion

To conclude, we have successfully accomplished demonstrating the power of curriculum learning. An initial hypothesis was made regarding the potential benefits of curriculum learning over the standard reinforcement learning approach. This was accomplished through understanding and explaining the intuition and technical theory behind curriculum learning. An implementation of a suitable environment with a simple reward function was developed to help with the proposed experiments. Curriculum learning with progression functions utilising the environment was implemented successfully. The run-time experiment was carried out successfully and the results acquired were very promising and showed the advantages of curriculum learning. Overall, curriculum learning is an intuitive approach to reinforcement learning and that intuition has been shown to be valuable and verified through this project. Curriculum learning may shape reinforcement learning in the years to come as more work is considered in this field.

5.5 Ethics

Ethical Issues: This project did not involve collecting personal or sensitive data. Many different ethical issues are discussed within machine learning, such as racial bias, however this project is unaffected by such issues since the data gathered and evaluated is purely for a theoretical project.

There are no ethical issues with the usage of code produced by others for this project. The code used to develop the environment is available for public use by OpenAI and has been adapted from [29]. Moreover, the code used to implement curriculum learning was developed from the work of Andrea Bassich of the University of York, the first author of paper [2]. He has given us access to the code he developed for the experiments in that paper and permission to use and adapt it (see Appendix C).

Legal Issues: The project was unaffected by legal issues, however we had to acquire licenses for

software. Student licenses were free provided the project was personal, excluding projects that received financial support or were used as part of employment. Dependency licenses were also considered, with all permitting use within personal projects.

References

[1] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=SkxpxJBKwS.

[2] A. Bassich, F. Foglino, M. Leonetti, and D. Kudenko, Curriculum learning with a progression function, 2020. arXiv: 2008.00511 [cs.LG].

[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, vol. 3, no. 3, pp. 210–229, 1959. DOI: 10.1147/rd.33.0210.

[4] C. Das, What is machine learning and types of machine learning –part-1, 2017. [Online]. Available: https://towardsdatascience.com/what-is-machine-learning-and-types-of-machine-learning-andrews-machine-learning-part-1-9cd9755bc647.

[5] M. van Otterlo and M. Wiering, Reinforcement Learning, First edition, ser. Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer, 2012, 638 pp., ISBN: 978-3-642-27645-3. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3.

[6] B. Osiński and K. Budek, What is reinforcement learning? the complete guide, 2018. [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

[7] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction, Second edition, ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018, 526 pp., ISBN: 978-0-262-03924-6.

[8] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," Journal of Machine Learning Research, vol. 21, no. 181, pp. 1–50, 2020.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, vol. 3, no. 3, pp. 210–229, 1959. DOI: 10.1147/rd.33.0210.

[10] The google deepmind challenge match, march 2016. [Online]. Available: https://deepmind.com/alphago-korea.

[11] D. Silver and D. Hassabis, Alphago: Mastering the ancient game of go with machine learning, Jan. 2016. [Online]. Available: https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html.

[12] M. Müller, "Computer go," Artificial Intelligence, vol. 134, no. 1-2, pp. 145–179, Jan. 2002. DOI: 10.1016/s0004-3702(01)00121-7.

[13] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, and et al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," Science, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018. DOI: 10.1126/science.aar6404.

[14] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," CoRR, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177.

[15] Part 2: Kinds of rl algorithms¶. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below.

[16] Barnabás póczos, introduction to machine learning. [Online]. Available: https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture26-ri.pdf.

[17] Blackburn, reinforcement learning: Bellman equation and optimality (part2). [Online]. Available: https://towardsdatascience.com/reinforcement-learning-markov-decision-process-part-2-96837c936ec3.

[18] Openai spining up, vanilla policy gradient. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/vpg.html.

[19] Openai spinning up, proximal policy optimization. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[20] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning (adaptive computation and machine learning se 2016.

[21] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey.," Journal of Machine Learning Research, vol. 10, no. 7, pp. 1633–1685, 2009.

[22] A. Lazaric, "Transfer in reinforcement learning: A framework and a survey," in Reinforcement Learning, Springer, 2012, pp. 143–173.

[23] J. L. Elman, "Learning and development in neural networks: The importance of starting small," Cognition, vol. 48, no. 1, pp. 71–99, 1993.

[24] T. D. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," IEEE transactions on Robotics and Automation, vol. 10, no. 3, pp. 323–333, 1994.

[25] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in Proceedings of the 26th annual international conference on machine learning, 2009, pp. 41–48.

[26] A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning," in Proceedings of the 25th international conference on Machine learning, 2008, pp. 544–551.

[27] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning.," Journal of Machine Learning Research, vol. 8, no. 9, 2007.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, Openai gym, 2016. arXiv: 1606.01540 [cs.LG].

[29] B. Baker and T. Markov, Multi-agent-emergence-environments, 2020. [Online]. Available: https://github.com/openai/multi-agent-emergence-environments.

[30] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, Emergent tool use from multi-agent autocurricula, 2019. arXiv: 1909.07528 [cs.LG].

Appendix A

Repository

This repository within which this project was developed can be found at:
https://gitlab.com/JackAuty/comp5530m-exploring-curriculum-learning.

Appendix B

External Resources

The aforementioned external resources we used as a basis for, and to inspire, our project are:

- MuJoCo - http://www.mujoco.org/
- OpenAI Multi-Agent Emergence Environments - https://github.com/openai/multi-agent-emergence-environments
- TensorFlow https://www.tensorflow.org/
- Python interface for MuJoCo - https://github.com/openai/mujoco-py/tree/1.50.1.0
- MuJoCo Worldgen - https://github.com/openai/mujoco-worldgen
- OpenAI Baselines - https://github.com/openai/baselines
- Stable Baselines - https://github.com/hill-a/stable-baselines

Appendix C

Evidence of Permission

Below are screenshots showing email correspondence with Andrea Bassich of the University of York and Dr Leonetti of the University of Leeds. The screenshots demonstrate that the authors of this paper were given permission to use the code we worked with in this paper.

From: **Matteo Leonetti** m.leonetti@leeds.ac.uk
Subject: help with mujoco
Date: 28 January 2021 at 15:38
To: Andrea Bassich ab1770@york.ac.uk
Cc: Fu Chan [sc17fsc] sc17fsc@leeds.ac.uk, Jack Auty [sc17j4a] sc17j4a@leeds.ac.uk, Luke McMahon [ll16l4m] ll16l4m@leeds.ac.uk, Bakhtiyar Khan [sc17bhrk] sc17bhrk@leeds.ac.uk

ML

Ciao Andrea,

a group of students (CCed) are working on a project for their MSc, and I suggested that they implemented the curriculum with a progression function in this environment https://openai.com/blog/emergent-tool-use/ . They are at the early stages, and the environment is based on mujoco, so they thought it would be helpful to see how your interface with the simulator works in the maze environment (I seem to remember that's the one).

Would you be able to have a quick chat with them and possibly share your code so that they have something to start from? The outcome of the project might also be useful to us for further experiments in curriculum learning.

Thanks!
Matteo

From: **Andrea Bassich** ab1770@york.ac.uk
Subject: Re: help with mujoco
Date: 28 January 2021 at 17:00
To: Matteo Leonetti M.Leonetti@leeds.ac.uk
Cc: Fu Chan [sc17fsc] sc17fsc@leeds.ac.uk, Jack Auty [sc17j4a] sc17j4a@leeds.ac.uk, Luke McMahon [ll16l4m] ll16l4m@leeds.ac.uk, Bakhtiyar Khan [sc17bhrk] sc17bhrk@leeds.ac.uk

AB

Hi all,

It definitely looks like an interesting environment where Progression Functions could be useful.

I attached the code I am currently using to run some experiments to this email. I do plan to upload it to GitHub in the future but before then I'll need to add some stuff (like a requirements.txt, etc.) - and improve the documentation. In the meanwhile all you need to get up and running should be already in the curr_pf folder.

If you have any questions or want to discuss anything please let me know.

Cheers,

Andrea

CL_with_PF.tar.gz