Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Jonathan Jørgensen

Quantifying Environmental Diversity in Reinforcement Learning

Master's thesis in Artificial Intelligence
Supervisor: Keith Downing
June 2020

NTNU
Kunnskap for en bedre verden

Jonathan Jørgensen

Quantifying Environmental Diversity in Reinforcement Learning

Master's thesis in Artificial Intelligence
Supervisor: Keith Downing
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Preface & Acknowledgements

Before settling on the exact topic for this thesis, I visited both meta reinforcement learning and benchmarking for multi-task learning. During these explorations, the concept of diversity caught my interest, as it seemed important, yet never properly defined in literature (to the extent of my limited knowledge). The final direction turned out to be a challenging project, but at the same time a satisfying pursuit, as it contributes slightly to opening the often frustrating black box of machine learning. In the end, I enjoyed both the journey and the destination, and hope to pick up the thread at some point in future research.

I would like to thank my supervisor Prof. Keith L. Downing, as well as my co-supervisor Dr. Arjun Chandra, for providing excellent guidance and feedback during the whole process. Additionally I thank Ole Christian Eidheim and Johannes Austbø Grande for their input, as well as my fellow students, family and colleagues for comments and relevant conversations.

Note A: Despite having a single author, the pronoun "we" is used, this is to make the language familiar and conformed with similar documents.

Note B: The cover image is inspired by a comment that pointed out how the title of this thesis can seem related to biology at a first glance. Painted by Melchior d' Hondecoeter. Circa 1680

Abstract

Solving multiple task with the same general agent is a wide open problem within reinforcement learning. In this project we seek to explore this by taking a closer look at the diversity in sets of environments. To do this, a novel algorithm for quantifying diversity is proposed, where the value functions or policy approximators produced by expert agents trained for each individual environment are compared numerically over a set of states. A class of environments is developed to demonstrate the usage of this method, and the results are promising and used as early indicators on the nature of diversity. A central backdrop through the whole project is the potential for scaling this system beyond the proof of concept stage.

Sammendrag

Å løse flere oppgaver med den samme agenten er en viktig problemstilling i reinforcement learning. I dette prosjektet utforsker vi konseptet *variasjon* i problem-sett. For å måle dette presenteres en algoritme for å kvantifisere denne variasjonen. Denne algoritmen trener opp en ekspert-agent for hvert problem og gjør en numerisk sammenlikning av verdi-funksjonene deres. Utviklingen og bruken av denne metoden er demonstrert på enkle egenutviklede illustrasjons-problemer, og resultatene er lovende og tolkes som tidlige indikatorer på rollen til problem-variasjon under læring. Potensialet for å skalere systemet til å passe reelle problemstillinger er et sentralt tema.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This is a set of five randomly generated gold-collecting puzzles. In this project we attempt to assign a numerical score that reflects the *diversity* between the tasks in sets like these.

Reinforcement Learning (RL) is a paradigm of machine learning that optimizes the performance of an agent according to a numerical reward signal. In recent years it has grown immensely in popularity, largely due to the many impressive results achieved by these techniques (Mnih et al., 2015; Silver et al., 2016; Vinyals et al., 2019). While the state-of-the-art agents consistently display superhuman mastery of certain singular tasks, they struggle where our own brains excel: *at solving multiple tasks*.

Multi-task reinforcement learning is a more recent niche, and it includes research towards solving multiple tasks using the same agent (Hessel et al., 2018), retaining the solutions, and adapting to newly introduced tasks (Platanios et al., 2020; Finn et al., 2017). A majority of the work towards these goals consists of designing agents and environment sets for evaluating these agents. This project focuses on the latter, namely the environment sets.

One of the current challenges of reinforcement learning is to design an industry standard benchmark for evaluating the multi-task capabilities of an agent. These efforts include using existing games (Nichol et al., 2018; Bellemare et al., 2012), leveraging procedural generation (Cobbe et al., 2019) and sets of modified copies of classical environments (Duan et al., 2016).

In this project we will not design yet another benchmark, but instead focus on what we believe to be an essential property of an environment set: diversity. Although the term has been used in different contexts, to our knowledge there is no proper definition available, nor any attempts at quantifying it.

1.1 Research Questions and Goal

1.1.1 Research Goal

The research goal of this project is to establish the idea of diversity within the context of reinforcement learning. To elaborate, we want to *define* diversity, both as a concept, as well as a numerical property of an environment set. Additionally, we want to demonstrate the usage of diversity analysis as a tool, and briefly observe what kind of effect this quantity has on training general agents.

This thesis seeks to address two main research questions, where the latter is dependent on the former. Each question will be elaborated, and sub-questions and constraints will follow. As an exploratory project, it is hard to draw strict conclusions, but the goal is to provide a meaningful coverage within the scope of a master's thesis.

1. **How can we compare different reinforcement learning environments numerically?**
2. **In what ways does environment diversity affect generalization?**

1.1.2 Research Question 1: Environment Comparison

Certain types of data, such as integers or strings, have well-defined methods for comparison. Environments are essentially programs, with highly complex structures that can have infinite capabilities. Although there are ways to formalize and approach all possible environments (Hernandez-Orallo, 2010), this project is constrained to simple illustrative environments.

While toy environments might not be representative for every possible environment, they should be sufficient for "proof of concept" experiments. This sufficiency is based on the assumption that the methods used for comparing are as general and domain-independent as possible. Given the fundamental nature of this work, it is also reasonable to assume that if a pattern is consistently observed in several toy environments, it is likely to be present to some degree in other environments.

The phrasing of RQ1 entails both comparing environments pairwise, as well as calculating diversity, as we consider the diversity function to be a generalized version of a comparison metric.

1.1.3 Research Question 2: Diversity and Generalization

If there were already established methods available for comparing environments, the second half of this project could work as a stand-alone thesis. The objective here is rather to explore an overlooked perspective in multi-task reinforcement learning: *the conditions for generalization*. An obvious precondition is a proper learning algorithm and hyper-parameters, but that only makes up one half of the agent-environment cycle. The other half, *the environment*, is also a crucial component to determine whether training will converge. We want to explore how a specific property of the learning environment, *diversity*, affects performance.

1.2 Motivation

In broad strokes, the contribution of this project can be divided into a theoretical and a practical perspective. The theoretical contribution is an attempt at formalizing the concept of environment diversity, and the motivation behind this is that such a formalization should exist, but does not. The practical contribution is that of the algorithmic metrics for comparing tasks and measuring diversity. The implementations can serve as tools for *diversity analysis*, whose usage is outlined in the utility subsection (1.2.2).

1.2.1 Theoretical Implications

A fundamental question within general AI goes as follows:

> *Under what circumstances does an agent generalize well to multiple tasks?*

This is a complex question, and if a strict answer exists, that is likely to be complex as well. The question can be decomposed into two keywords, *circumstances* and *generalize (well)*, each demanding further elaboration. The latter part is linked to measuring performance and intelligence, and related efforts can be traced all the way back to the Turing test (Turing, 1950). While we do address multi-task performance metrics in this project, it is primarily for practical reasons. Our main contribution is instead a step towards formalizing the *circumstances* of multi-task learning, and we propose that a central component to this is the titular *environment diversity*.

1.2.2 The Utility of a Diversity Function

While this project explores the possible internal mechanics of a diversity function, discussing the utility of such a function should provide a greater context to the problem and the motivation. The following list assumes a diversity function is existing and implementable, and that it accepts any set of environments.

- If one or more benchmark environments are proposed for measuring the generalization capabilities of an agent, the diversity of different configurations of said benchmarks could serve as an important property. Both as a descriptive property, as well as a basis for comparing and ranking different benchmarks.
- While training an agent in a multi-task setting, the diversity of the training set and the testing set, as well as the inter-set diversity between the two could serve as useful information both for debugging poor performance, as well as informing agent design prior to training. For example, if training and agent for solving multiple ATARI games, it could be useful to know if some of the games are more distinct from the others.
- When an agent is trained with robustness and adaptability in mind, the diversity in the simulated training environment could play an important role when deploying a trained agent into a real-world setting. For example, if training automobile agents in a simulator such as CARLA (Dosovitskiy et al., 2017), it is desirable for the agent to encounter a wide range of scenarios inside the simulator instead of the real world, both with safety and resources in mind. Another aspect is that the simulator and real environment are very likely to have many subtle differences, such as color and lighting in image observations, and a robust model could be less sensitive to this.

Chapter 2

Background

Diversity is an established term in the English language, but in order to define it in a technical setting, the surrounding context is essential. In this project, the context is reinforcement learning, and the language includes agents, environments, states and actions. In this chapter we describe the basic essentials of this paradigm, as well as specific topics that are used thorough this project. Finally, a terminology section is provided to clearly define some of the key terms for the remaining chapters.

2.1 Reinforcement Learning

Reinforcement learning is a sub-paradigm of machine learning that deals with training agents in sequential decision-making problems. The shared objective of all algorithms defined under this paradigm is to maximize a reward signal from a task environment, by acting as a response to observations.

The main structure during both training and execution is the agent-environment interface. Through this interface, the agent is prompted for an action at each time step, and after submitting one, the environment returns the successor state and a reward. Each time step in this process corresponds to a (state s, action a, reward r) tuple (Sutton and Barto, 2018)

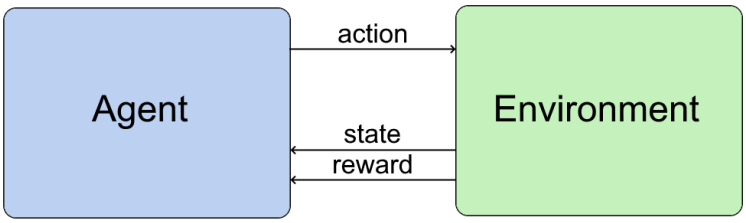


Figure 2.1: The agent-environment interaction cycle

2.1.1 Value functions

A value function returns the expected future discounted sum of rewards for a given state or state-action pair. The two main value functions is the state-value function $V(s)$ and the action-state function $Q(s, a)$. For all environments, there exists a *true value function* that is equal to the actual expectation, usually denoted by an asterisk ($V^*(s)$ or $Q^*(s, a)$). A class of learning algorithms known as *value-based* RL, a value function is approximated, usually as a regression problem.

2.1.2 Policies

Where a value function might inform an actor on the best action at the moment, a policy function will instead guide the actor directly. A policy, usually denoted as $\pi(s)$ returns either a specific action (a deterministic policy), or a probability distribution over all actions (a stochastic policy). Where value-based methods

approximate value functions, *policy − based* instead approximate the optimal policy. Some algorithms do both, and they are called *actor-critic* methods, where the actor is the policy, and the critic is the value function.

A class of algorithms in reinforcement learning called policy gradient methods seek to approximate the optimal policy directly. Some of the main advantages of this over value-based methods is that environments with a stochastic optimal policy can be solved, and continuous action spaces can be handled more easily.

2.2 Function Approximators

The optimal value functions and optimal policies in reinforcement learning are unknown, and if they were known, there would be little reason to perform learning. Instead, learning algorithms usually *approximate* one or both of these function by acting and observation. The actual representation of the approximation is embedded into the parameters of a model, and in this section, some of these models are introduced.

2.2.1 Linear Regression

Linear regression is a machine learning algorithm that learns a vector of parameters w to predict an output $\hat{y}$ from the input x, $\hat{y} = w^T x$

2.2.2 Artificial Neural Network

Artificial Neural Networks, or ANNs, are mathematical models composed of artifical neurons, inspired by biology. By minimizing a *loss function* while training on a *data set*, the network approximates the patterns in the data. Linear regression is a special case of a neural network, where the input is mapped directly to the output, with no intermediate (or "hidden") neurons. (Goodfellow et al., 2016)

2.3 Learning Algorithms

Reinforcement learning provides a framework for formalizing agents, environments and their interaction, but in order to train an actual agent towards optimal behaviour, a learning algorithm is required.

2.3.1 Q-learning

Q-learning is an algorithm that directly approximates the function $Q(s, a)$ through the Bellman equation. The classical implementation represents the function as a table, where the rows are states, and the columns are actions.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \text{max}_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.1}$$

The bellman equation (Eq. 2.1) is applied at every step during training, to update one cell of the Q-table. s_t is the state before acting, s_{t+1} is the state after acting, a_t is the action taken, and r_t is the reward received. α is the learning rate, and γ is the discount factor used to determine how far into the future rewards matter. The $r + \gamma maxQ$ term is the target value in the update, and it shows that rewards are bootstrapped from discounted future states.

2.3.2 DQN

The vanilla implementation of DQN extends classical Q-learning by introducing its three main components: a function approximator, a target network, and a replay buffer. The function approximator is usually a deep neural network (hence the name DQN) which is trained to approximate the Q function. Unlike tabular Q-learning, which computes each action-value pair individually, a Q-network outputs Q values for all actions simultaneously. The target network is a regularly cached copy of the Q-network that provides more stable value estimations during training. Finally, the replay buffer contains agent memories which is used as a dataset for supervised learning. (Mnih et al., 2015)

2.3.3 REINFORCE

REINFORCE, also known as "vanilla policy gradient", is an algorithm that directly estimates the parameters of a policy function approximation by gradient ascent. As the policy function outputs a probability distribution over actions, the model has a softmax output. These probabilities reflect how likely it is that the respective action is optimal behaviour.

$$\theta \leftarrow \theta + \alpha\gamma^t \Delta \ln \pi(a_t|s_t, \theta) \tag{2.2}$$

At every time step, the model is updated by applying the rule in equation 2.2. θ is the model parameters (eg. the weights of a neural network) and π is the policy function, with respect to the parameters. The rest of the terms are the same as in section 2.3.1.

2.4 Environments

The task to be solved by an agent is represented by the environment interface. On certain transitions, a numerical rewards signal is returned to the agent, and this serves as the basis for learning. Russell and Norvig (2002) presents seven properties to classify task environments in artificial intelligence. These properties are very high-level and descriptive, and their main purpose is to categorize environments in a way that is useful when selecting what methods to design an agent by in advance. Most of these properties correspond to technical properties in the environment implementation, but some, such as observability or multi-agency are more debatable in nature.

- **Fully vs. partially observable**
  Observability refers to the information exposed to the agent through its sensors. This definition is restricts information to what is relevant to solving the task, so if excess information, such as the weather during a chess match is excluded from the observation, the environment is still considered fully observable.
- **Single agent vs. multiagent**
  If more than one intelligent agent acts simultaneously in an environment, it is considered multiagent. Within multiagent environments, further distinction is made on whether the environment is competetive or cooperative.
- **Deterministic vs. Stochastic**
  Determinism refers here to the dynamics of the environment, and whether every state-action pair consistently determines the next state.
- **Episodic vs. Sequential**
  In an episodic environment, there is no persistence in the states, so every episode is independent on the previous. In sequential environments, the outcome of an action may depend on previous actions taken.
- **Static vs. Dynamic**
  Dynamic environments have a timeline that moves independently of the agent, while static environments "wait" for every action.
- **Discrete vs. Continuous**
  This refers to the state, the actions and time itself. The real world is considered continuous in all three regards, while simulators can have any combination of cases for these three properties.
- **Known vs. Unknown**
  If a complete model of the environment dynamics is available, it is considered known. A known environment can be both deterministic and stochastic, as the model can output a probability distribution to reflect the dynamics of a stochastic environment.

They classify real-world problem solving, such as taxi driving as a *partially observable, multi-agent, sequential, dynamic, continuous and unknown* domain. The determinism is still up for a more philosophical debate, but from the perspective of agents such as humans, it appears to be *stochastic* as well.

Using these definitions, the environments in reinforcement learning are usually sequential, discrete (in terms of time), and in most cases, static. Although time in the environment is classified as discrete, both the action space and state space can be continuous, and while many environments deal in episodes, multiple actions are taken within one episode (except for bandit environments). Lastly, a live environment to a deployed agent might not "wait" for actions to be taken, but during training it is usually meaningless to put any time constraints on action selection, as it will only halt any learning. The remaining properties can

vary across domains, but the typical toy problem operates with fully observable and deterministic single agent environments with a known and available environment model.

2.4.1 Markov Decision Processes

A Markov Decision Process, or MDP, is a formalization of sequential decision making. At the heart of such a process lies an agent-environment interaction cycle, where an agent acts, and the environment reacts. An MDP can be represented by a graph, where the nodes correspond to states, and the transitions to actions and rewards. The outcome of an action applied to a state is determined by the *dynamics* of the environment, a function that determines the probability of a transition. Deterministic environments, where the outcome of a state-action pair is always the same, can be considered a special case MDP where the dynamic function returns strictly one and zero.

2.4.2 Multi-Task Learning Setup

Traditional reinforcement learning matches the agent-environment cycle in figure 2.1 both in theory and practice. When handling multiple environments, however, a few additional considerations must be made. From the perspective of the agent, a multi-task setup can be identical to a single-task one, as the environment manages which task to present at every step. A common approach is to sample a random environment whenever the current one terminates. (Hessel et al., 2018)

2.5 Terminology

As this project is covering an under-explored niche of machine learning, parts of the terminology is not well-established. This subsection seeks to define a selection of the most central terms used across this thesis. While most of them are familiar and/or self-explanatory, they are still included to avoid ambiguity and potential confusion from different interpretations.

Environment Set While the underlying implementation in this project usually groups environments into list structures, the most appropriate mathematical term is a set. If an environment should occur more frequently than others, this can be implemented into the sampling process, instead of having duplicates.

Generalization In supervised learning, generalization refers specifically to a model finding patterns in the training data and applying them to the testing data. Supervised methods are often used as a part of reinforcement learning algorithms, and therefore this definition applies to both sub-fields. However, in RL there is also the concept of generalizing behaviour over multiple tasks (multi-task learning), and in this context there is a more "high-level" generalization at play.

Environment vs. Task In this project, the terms environment and task often refer to the same concept, as the included environments contain exactly one very specific task. In reality a single environment can contain multiple tasks, but when illustrating diversity, this will only complicate matters.

Task Domain Environments that have the same dynamics are considered to be of the same domain. This term is usually found in more theoretical discussions, and can include unimplementable environments, such as the real world or those with infinitely complex dynamics.

Environment Class An environment class is the implementation of a task domain. In a technical context, such as instructions to reproduce results, this terminology is more appropriate.

Model The term model has several distinct definitions in reinforcement learning, but one of the most established uses lies in whether an algorithm is model-free or not, which refers to a model of the environment. In this document, this is **not** the case, unless it's explicitly named any variant of "environment model". The primary use will rather be in reference to the model that represents/estimates the value function or a policy. This includes tables, linear regression and neural network models, which are all used at different points through the project.

Expert Agent An agent that is fitted to a specific task, and not expected to perform well on other tasks.

General Agent An agent that is capable of solving multiple tasks well, but might not solve each optimally.

Chapter 3

Related work

When presenting work related to this project, different perspectives to diversity and environments are featured. The first sections consist of different approaches to quantification of task environments and generalization. The remaining sections address projects that refer to diversity, grouped into solution-based diversity and environmental diversity.

3.1 Structured Literature Review

Early on in this project, a structured literature review (SLR) was conducted to find relevant materials to build upon. Seeing as this research is a little different in nature than many other publications in the field, this was not trivial. The first step involved keyword search in various academic databases. The keywords used were **Generalization**, **Multi-Task Learning**, **Environments**, **Diversity** and **Reinforcement learning**, in different combinations. The most useful results were from the combination *Generalization + Reinforcement Learning + Multi-Task Learning*.

The abstract and introduction of the most promising papers found were read properly, and the references found while doing this were also considered. Roughly fifty documents were handled during this process, where about half were discarded, and about ten additional papers were included in the related literature outside of the SLR draft.

3.2 Universal Intelligence Measure

Legg and Hutter (2007) introduces the Universal Intelligence Measure (UIM), an attempt at formalizing intelligence within a mathematical and algorithmic framework. They use the structure and terminology of reinforcement learning and propose the set of all Turing computable environments with a finitely bounded return of rewards as a benchmark for measuring the true general intelligence of an agent. Even if this proves to be an accurate metric, it is theoretical in nature and computationally infeasible. Unlike the well-known Turing test, which can be vague in nature and is constantly debated, the UIM is stripped of any association with human intelligence and behaviour, and is thus more useful for an algorithmic approach to artificial intelligence.

Legg and Veness (2011) attempts to approximate the computationally infeasable UIM by introducing AIQ, the Algorithmic Intelligence Quotient. As no canonical Turing machine is available for use as the reference machine in this setup, the modified variant of the BF programming language is chosen as an alternative. Random programs are then generated, and things such as redundant code segments and programs without input or output are discarded.

In the context of this project, we assume environment sets produced by both UIM and AIQ to approach a theoretical ceiling for diversity. This assumption is based on the idea that in the set of all possible environments, the most diverse pair should also be present. Another interesting consideration is that these sets also include the *least* diverse environment pairs.

3.3 Quantifying Generalization in Reinforcement Learning

While metrics such as AIQ serve as a useful guide for generalization capabilities, it is rather abstract and hard to tie to practical problem solving. Other attempts at a similar benchmark abandon the notion of universal intelligence and all possible Turing computable environments, and instead focus on one or a few domains of traditional problem solving. The advantages of this approach include easier interpretability, ease of development and agents that can reasonably be expected to perform well in similar real-world environments. In theory, any set of environments can be considered a subset of the set presented in UIM, and this also applies to both AIQ and CoinRun (Cobbe et al., 2018). The difference between these two approaches is that AIQ is attempting to approximate all environments, while CoinRun is a hand-crafted subset with a multitude of "aesthetic constraints" meant to anchor the environments in realistic logic.

Cobbe et al. (2018) investigate overfitting in RL and perform multi-task reinforcement learning by splitting the environments into a test set and an evaluation set, a practice common in supervised learning. Through experiments featuring their CoinRun environment, they show that agents need to be exposed to a vast selection of levels before successfully generalizing to unseen ones. In their example, the test performance didn't match training performance before the training set size surpassed 10000 unique levels. In the context of diversity, it is interesting to ask whether a smaller set of a higher diversity could achieve the same results.

3.4 Diversity in Solutions

In order to properly illustrate the diversity in this project, namely *environmental diversity*, it is imporant to outline other interpretations of the term. To do this, we roughly group the other interpretations under the term "solution diversity". When referring to diversity in solutions, this includes methods were multiple solutions are considered simultaneously (eg. evolutionary algorithms), but also those where a singular solution changes over time (eg. most RL algorithms).

When searching for the solution to a problem, a narrow approach can potentially halt the progress of a learning algorithm completely. In RL, this is embodied in the "exploration vs. exploitation" dilemma. From a high-level perspective, exploitation can be described as a lack of diversity among the solutions considered during training. Diversity in solutions is relevant both in a single-task and a multi-task setting.

DIAYN, short for "Diversity is all you need" (Eysenbach et al., 2018), is an algorithm for unsupervised pretraining for RL agents. In this context, the objective is to train forth a *diverse* set of skills. Their method is based around training a maximum entropy policy, without receiving a reward signal during

training. Not only is this method a good pretraining setup for traditional RL training, but it can even solve certain tasks by itself, hence the "all you need" phrasing of the name. An improved version of the algorithm was introduced in Sharma et al. (2019).

Although the exact definitions may vary, the concept of diversity is present in various other sub-fields of computer science as well. One of the most notable examples is evolutionary algorithms (EA). Methods within this discipline operate with a population in one shape or another, and this population consists of distinct specimen entities. Bhattacharya (2014) emphasizes the importance of diversity in EA to prevent premature convergence of the system.

Although their work focuses on diversity in their population, which is most comparable to the agent-side of the learning problem, while this project in concerned with environments, the relationship between diversity and performance is similar. An important distinction is that in EA, diversity is controllable during training, while we define it as a static property of the environment set that can only be changed by modifying the set.

A different example of solution diversity in machine learning is ensemble methods, where multiple models are combined to act as one (Dietterich, 2000). The advantages of this approach include robustness, as the weaknesses of the individual learner can be compensated for by other parts of the ensemble. The diversity among the structures and the parameters of the individual learners in an ensemble is the key reason to why it is a solid technique overall.

3.5 Environmental Diversity

Although the exact definition of environmental diversity is not established in the literature, its significance and value is indirectly emphasized in multiple ways.

Randomizing or augmenting environment properties and agent observations can be used as a method for training more robust agents (Lee et al., 2019; Slaoui et al., 2019). For visual environments, this can be done by for example changing colors or textures in the observation image, or by full transformations, such as rotation. From our perspective, this can be seen as injecting *artificial diversity* into the training, with the intention of improving the agent. One interesting question is to ask how much of this injection the system can handle, and whether there is a "golden ratio" where the advantages are maximized without breaking the training.

A more natural source of diversity is the agency of other agents in a multi-agent setting. Instead of having a set of different tasks, confrontational scenarios with other agents can provide a seemingly endless supply of unique tasks. Al-Shedivat et al. (2017) approach multi-agent environments as if they were multi-task environments, and apply their proposed meta-learning algorithm to adapt to this ever-changing environment. Unlike classical multi-task settings, where

tasks are typically sampled from a fixed set, the nonstationary nature of multi-agent environments could present a task once, and then never again. This pace breaks many of the more "steady" approaches to learning, and forces an emphasis on adaptation, hence the use of meta-learning techniques.

3.5.1 Benchmarks

In the recent years, several different benchmarks for multi-task reinforcement learning have been proposed. Some focus on specific aspects such as Never-Ending Learning (Platanios et al., 2020), while others provide a more general set of environments, suitable for both single and multi-task learning. An already established suite of tasks, MuJoCo, has been used by many to test various types of multi-task RL, such as meta-learning (ref). The famous ALE (Arcade Learning Environment, ref), featured in deepmind's DQN demonstration (Mnih et al., 2013), is also a suitable candidate, as the state and action spaces are identical for all games.

Cobbe et al. (2019) presents a environment suite leveraging procedural generation to create virtually infinite variations of six different tasks. When introducing the environment, they list diversity as one of the central desirable features in a proper benchmark. Generating content procedurally is their solution to providing this diversity.

Chapter 4

Method

This chapter introduces the methods developed for this project, as well as the main ideas behind their design. First we establish the core ideas behind diversity in RL, as well as its possible implications. Secondly we introduce the diversity algorithm and the different configurations of it. Finally we outline the experiments that have been carried out to test the different hypotheses about the nature of a diversity function. More details about specific implementations can be found in appendix A.

4.1 Defining Diversity

A policy or value function that suggests optimal behaviour for all states can be considered the "solution" to the task represented in an environment. If these functions turn out identical for two environments, we can consider the environments equal, at least from the perspective of an agent. If an agent is to solve more than one distinct environment, however, and still behave optimally, it needs to somehow embed the value function for each task into its underlying model.

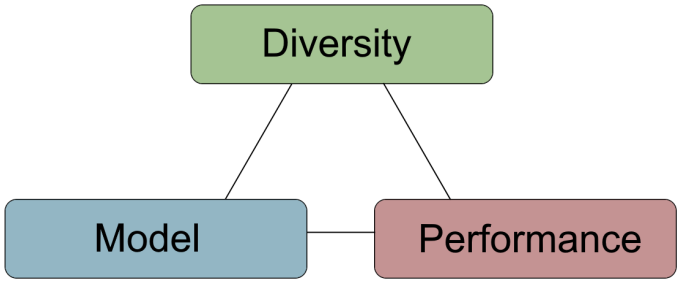


Figure 4.1: An overview of three central components of a multi-task learning problem. The diversity is a property of the environment set, the learning model is designed by the user, and the performance is a consequence of the two.

Figure 4.1 illustrates the idea that the agent model, the learning performance and the environment diversity are all linked. In this context, the model represents the architecture, parameters and hyper-parameters of the agent, and performance represents metrics such as return, convergence and stability. An important note is that this figure does by no means suggest that these three are the *only* components at play during multi-task learning, but in the context of this project, they are the most relevant. One of our key hypotheses is that there is a meaningful symbiosis between these three concepts. To further elaborate, the hypothesis implies the following relations:

- The performance is determined by both the model and the diversity.
- The diversity is a property of the environment set, and cannot be changed without changing the set.

In other words, diversity is static, performance is only observed, and neither are directly controlled by the agent designer. This means that a diversity function should be consistent in the score it assigns to an environment set. The performance on both training and evaluation sets could be affected by diversity.

We provide the following definition of diversity:

> **Environmental diversity** is the extent of how the individual tasks in a set differ from each other.

To further emphasize this, we illustrate the extreme cases of zero and maximal diversity. If there is zero diversity, the exact solution to one task can be successfully applied to all other tasks to achieve optimal behaviour. This does not necessarily mean that the tasks are identical in presentation. If the diversity is maximized, no tasks will have any shared properties, and explicit solutions for each must be embedded into the agent to behave optimally.

4.2 Designing the Metrics

While the preceding definition of diversity illustrates its significance and behaviour, the internal mechanics are still largely unknown. In this section we move towards a technical implementation of a diversity function approximator. We discuss different representations of task environments, as this is a precondition to comparing them. The central ideas behind the resulting implementation presented in section 4.3 are gradually outlined, and alternative approaches are briefly explained.

When designing a metric there are a number of factors to consider:

- Does the metric properly reflect the concept that is being measured?
- What are the limitations of the metric?
- Do these limitations constitute an acceptable compromise?
- What is the computational complexity?
- How does the metric scale within the range of expected usage?

Naturally, the first factor is essential, while the others are more implementation-oriented.

Firstly, and most importantly, we want to outline a pool of concepts that are expected to have some relation to diversity. In this context, diversity is an aggregated extension of similarity, and to measure similarity, we need to represent the compared objects in a format where their features align.

In the literature, MDPs are established as a theoretical representation of an environment, and this makes them an ideal candidate for a comparable representation. In the early stages of this project, MDPs were generated from toy environments and compared as graph structures. The two main issues with this approach were that graph comparison is very hard (Wills and Meyer, 2019), and that the methods do not scale well, both in time and memory usage.

The approach that our proposed metrics are built on has a different perspective. Instead of comparing environments directly, we compare their approximated

value functions or policies. This methodology is based on the theory that generalization (or intelligence) is akin to the compression of data. Dowe and Hajek (1998) states that a proper intelligence test does not only require a passing of the classical Turing Test, but also that the agent should have a compression of the subject matter.

We tie this idea into reinforcement learning, specifically algorithms built around neural networks (deep RL). If the optimal value function or policy is successfully represented by a neural network, it can be used by an agent to behave optimally. This is the theoretical outline for value-based and policy-based methods in RL, where these functions are approximated through acting in the environment.

Extending into multi-task learning, we state that the optimal value function for a multi-task setting is likely to be related to the optimal value functions of the individual tasks. Rusu et al. (2015) present a method called policy distillation, where agents are trained by mimicking expert agents, with the goal of transferring the knowledge into a smaller model, or combining several experts into one general model.

The techniques applied in policy distillation serve as the main inspiration for the final metrics developed for this project. Other methods were considered, but dismissed for various reasons:

Dynamic Programming Dynamic programming (Sutton and Barto, 2018) can be used to solve RL tasks optimally, but require a model of the environment, and doesn't scale well.

Environment Model Approximation Approximating the environment dynamics model (Kuvayev and Sutton, 1996) can be used to embed the environment into a neural network. This neural network can then be used as a basis for comparison. While interesting, this approach is problematic, because different tasks can have the same dynamics.

Graph Comparison of MDPs As mentioned previously, a graph based comparison is hard to define properly in a way that ties it to diversity. Additionally, MDPs as data structures in memory can be very large or infinite in size. This direction might be revisited in the future, however, as early developments are being made towards approximating MDPs (van der Pol et al., 2020).

4.3 Comparison Metrics

The core component of the diversity algorithm is the comparison metric. This is a function that takes two task environments as an input, and returns a score based on how different they are. All variations of the comparison metric introduced in this project have the same general structure, but the following aspects are different:

- Which learning algorithm that is used to train the expert agents

- Whether a softmax function is applied during comparison
- How the state distribution is produced

4.3.1 Comparison Algorithm

The general algorithm for comparing a pair of environments has a precondition, which is that an expert agent is trained for each environment. If this precondition is satisfied, comparison consists of following three steps:

1. Create a list of states (*all_states* in Figure 4.2)
2. Iterate over every state in the list, and compare the expert agent responses
3. Return the averaged result of comparing across *all_states*

The states in *all_states* is a concatenation of relevant states from both environments. Two interpretations of *relevant states* are implemented in this project: **full state distribution**, and **agent memory**. The full distribution is simply a list of all reachable states, generated by the environment itself. This is of course dependent on whether such a function is implemented in the environment. The agent memory solution is rather based on which states the expert agent visited during training.

Using the full state distribution should be less prone to a false score, as both the environment and the agent model is given full coverage in the comparison. This solution has two main drawbacks, the first is that the environment implementation might not provide this kind of information, or it might be practically impossible to do so (such as a very large amount, or even infinite number of states). The second drawback is that all states are weighed equally, when in reality, some might be more significant for the comparison than others.

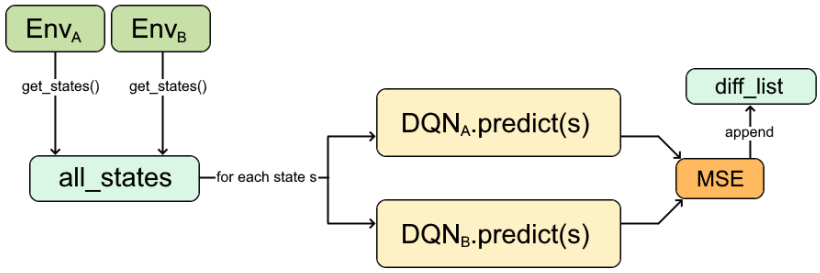


Figure 4.2: Value-based comparison, DQN version, using full state distribution

The memory-based solution, however, does not share any of the drawbacks of the full distribution. Implementation wise, it is a modification of the learning

algorithm, rather than the environment, and some methods even have such a memory implemented by default, such as the replay buffer in DQN. In terms of state significance, states that are visited more frequently are featured more in the memory. An issue with this approach is that the memory is highly dependent on the training and exploration, and it is unlikely to contain the exact same distribution across multiple independent runs, even with the same initial configuration. This contributes to instability in the metric, which is already an issue as the expert agents are only *approximations* of optimal behaviour.

When the list of states is assembled, the next step of the algorithm is to iterate over the list and compare the expert agent behaviour for each state. The exact anatomy of this step is dictated by which learning algorithm is used for the agents, but the final step in each iteration is the same: to compute the *mean squared error* (MSE) between two vectors. In this project, these vectors are either state-action-values (DQN implementation), or action probabilities (REINFORCE implementation).

4.3.2 Expert Agents

An expert agent is an agent that is trained to solve one specific task, and they are the most important components of this system. The main idea is that these agents approximate either the optimal policy, an optimal value function, or both. These functions are optimal with respect to the task environment in which they are trained, and serve as a link between the dynamics and the rewards. We propose that environmental diversity relates to optimal behaviour, and use the expert agents as representatives for this.

This approach introduces one of the major limitations to this system: the individual tasks of the environment set have to be solved in order to do a diversity analysis of the set in a multi-task setting. This builds on the assumption that solving tasks individually is *typically* easier than solving the combined, and we acknowledge that this might not apply to all task domains.

For this project, the value-based algorithm DQN, and the policy-based REINFORCE were selected for training expert agents. Both solve the toy environments we use well, and they provide an action-value function and a policy approximation, respectively, which are interesting to compare in this context. The conceptual simplicity and lightweight implementations of these also contribute to a less convoluted system. Additionally, because the agents are trained independently, a simple and self-contained training setup can be duplicated and distributed to reduce the duration of the analysis.

4.4 The Diversity Algorithm

A comparison metric alone is an operator that returns a numerical representation of the difference between *two* environments. This does not equal a full diversity function, as diversity is a property of a population of objects, rather

than only a pair. Note that the implementation of this function is and should be as independent of the underlying comparison metric as possible.

The general algorithm for the diversity function is based on the naive approach of comparing every unique pair and using the mean difference as the estimated diversity. As illustrated in table 4.1, identity pairs (along the diagonal), and reflected pairs (in the lower triangle), are both omitted from the calculation. Because all low-level difference calculations in this system are either absolute or squared, it is commutative (Diff(A, B) is equal to Diff(B, A)), and thus, having both would contribute nothing but increased execution time for the algorithm.

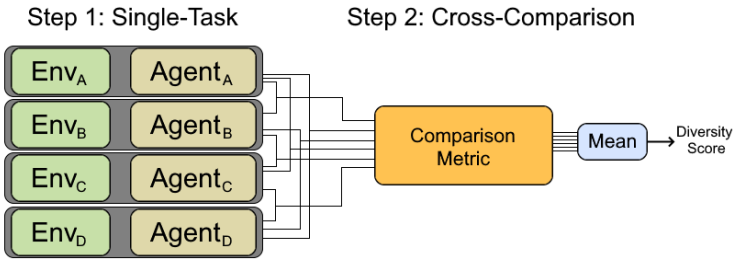


Figure 4.3: The general algorithm structure for computing the diversity score.

Figure 4.3 illustrates the diversity algorithm, where expert agents for all tasks are compared pairwise, and the mean difference is returned as the final diversity score. Algorithm 1 provides a more detailed description, and the term Diff(expert$_i$, expert$_j$) represents the previously defined comparison scheme.

Algorithm 1: Diversity Function Approximation

Input: list of environments
Output: diversity score
initialize list of expert agents
for *i in range(0, n_envs)* **do**
  expert$_i$.train(env$_i$)
end
diff_list = []
for *i in range(0, n_envs)* **do**
  for *j in range(i, n_envs)* **do**
    diff_list.append(Diff(expert$_i$, expert$_j$))
  end
end
return mean(diff_list)

	Env_A	Env_B	Env_C	Env_D
Env_A	-	X	X	X
Env_B	-	-	X	X
Env_C	-	-	-	X
Env_D	-	-	-	-

Table 4.1: A matrix showing the pairs that will be compared when handling a set of four environments (crosses indicate comparison)

4.5 Environments

Exactly what constitutes *one* environment is highly dependent upon the context of the discussion. In our project, we define an environment to be the unique tuple of an initial state, the terminal goal state(s), and the transition dynamics. If any of these are changed, it is considered a different environment. Other projects might take a whole domain and consider it a singular environment, but for our purposes, this approach removes much of the necessary task-space granularity for properly demonstrating the concept of diversity.

When designing the environments used for this project, a general framework named *kvad* was developed. It is inspired by the various *gridworld* presented in Sutton and Barto (2018), but with an emphasis on multi-task settings. The following properties were central to the development:

- **Scaling** To test environments of different sizes
- **Normalized rewards** To avoid some common multi-task issues caused by reward of different scales (Hessel et al., 2018)
- **Fast execution** For running numerous experiments
- **Interpretable** Visual and intuitive
- **Expressibility** Different dynamics

Scaling happens through changing the world size, and in theory the dimensions can be as large as possible, but the input layer in the agent model needs to handle it. Normalizing the rewards is mainly a convention when using the framework, where most tasks operate exclusively with the rewards -1 and 1. Fast execution is possible through lightweight dynamics and no mandatory rendering. Tasks that involve navigation and simple game-like interactions are typically relatable and interpretable. A wide range of tasks can be realized in a 2D grid. Figure 4.4 shows the shape of observations in kvad, and by having dedicated roles for different layers of the grid, we can in theory implement many different types of games within this framework.

Early experiments show that even small and simple environments are meaningful in terms of diversity. Because of this, we choose to continue in this direction, as

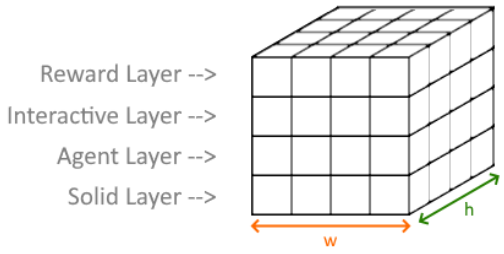


Figure 4.4: The shape of observations in kvad, where two dimensions are determined by the width w and height h, while the third dimension consists of four layers, each representing a different type of object.

it brings some major advantages. The obvious advantage is fast execution, as it allows for a greater number of experiments to be run, which is important for this project. Another advantage is that these environments are implemented with a method for extracting the full state space, which is used to produce important reference values for the diversity analysis.

Sprites[1] are used instead of colored squares, for two main reasons, the first being that they are less prone to information loss when converted to gray-scale, and secondly because the symbology can often communicate the task with little to no explanation. Performance wise the agent is never exposed to this rendering, so the execution time is not affected unless render mode is turned on for eg. debugging or demonstration purposes.

Some of the design choices behind these environments are more long-term, such as the interactive layer in the state tensor, or the grid-like structure. The environment class used for this project, called *gridworld:collect* features a player (person shaped), and gold (yellow stacks) and the action space consists of four discrete actions, one for each direction the agent can move in. If it walks into the same cell as a stack of gold, a reward of 1 is given, and if this was the last stack of gold, the game terminates.

4.6 Measuring Performance

In this project we conduct a wide range of different experiments. This serves as a field test for diversity analysis, to observe whether these methods work as expected in a practical setting. Because the environments used have a termi-

[1]The sprites are created by JoeCreates (https://twitter.com/JoeCreates) and distributed on OpenGameArt under the CC BY-SA 3.0 license. https://creativecommons.org/licenses/by-sa/3.0/

Figure 4.5: A graphical render of one environment (4 x 4, seed 33) of the gridworld:collect class, where the agent is rewarded 1 point for picking up stacks of gold until none are left.

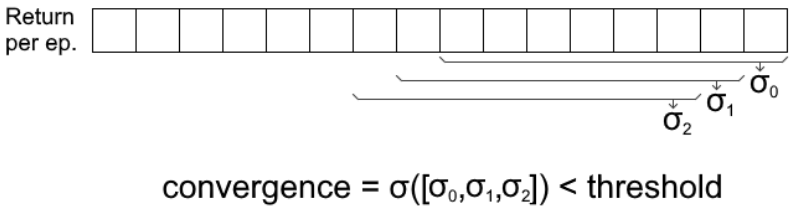


Figure 4.6: An example of a 3-step convergence condition with window size 8

nating goal, we measure the time to complete the task as the main performance metric.

4.6.1 Convergence

When training the expert agents, it is important to detect convergence, to save time. When training on multiple environments, a new one is sampled randomly whenever the current one reaches a terminal state. This is to prevent a predictable pattern in the order of environments that the agent could potentially fit to, as well as to provide more exploration. Because the different tasks can have arbitrarily different returns from optimal play in an episode, it is important to define convergence as a *stable* variance in return, instead of just a low variance. To do this, we use a sliding window approach to compute the standard deviation for a constant number of episodes prior to the latest one, and then use the standard deviation of those values again to check for convergence (std of previous stds). Because they are only based on unchanged historical data, these values can be cached for faster execution. The window size and threshold are parameters that determine how confident the convergence check should be before stopping the training session early.

4.6.2 Stability

Reinforcement learning methods are often unstable and dependent on initial conditions. Because of randomization, seemingly identical experiments can have completely different outcomes. Despite this, there can still be consistent trends across multiple training sessions. To capture this, most experiments that involve the training of agents will be averaged over many repeated sessions, and only the mean performance will be considered as a result. The standard deviation can be included as a stability metric, to further validate (or invalidate) the results.

In order to test the stability of a diversity function, we will investigate how it changes over time when the underlying agents are trained. To do this, we initialize the agents for all environments, and then proceed to train them for a limited number of episodes. After training, the diversity of the set is measured with respect to the current parameters of the agents. This process is repeated multiple times, without resetting the agents for each iterations, but instead continue the training from where it left off.

We want to measure two kinds of stability: first, that the metric converges with minimal noise and that it doesn't "unlearn" its value after a while. Second, that several runs with different initial conditions (such as the seed to the random number generator) produce roughly the same results. To test the first, we train for an extended period of time, and to test the second, we run multiple sessions and compare them to each other.

As the diversity is based on several combined approximations, it also inherits the instability of every approximation. To emphasis this, we will train the expert agents in intervals and record how the diversity changes over time.

4.6.3 Neural Network Architectures

The expert agents in this project use linear models, implemented as neural networks with no hidden layers. This is because most, if not all of the environments in kvad can be solved easily by such a model. Generally, smaller models are also less prone to overfitting, and this model has the minimal number of parameters possible for a fully connected neural network. For the multi-task training, we introduce hidden layers where necessary, typically one dense layer of 32 neurons.

4.7 Experiment Overview

Through this project, a wide range of experiments are carried out in order to cover as much as possible of diversity analysis. To test stability, most experiments are repeated multiple times, and the agents are trained from scratch every time.

4.7.1 Environment Baselines

Before the diversity-related experiments, we begin by solving a selection of environments to get a sense of their difficulty. Since all the environments base their reward systems on a win condition, a fitting performance measure is how many steps the agent uses to reach this condition on average (where a lower value is better)

4.7.2 The Identity Test

The first test directly related to comparison and diversity is the identity test. Among all the tests, this is likely the one with the strongest "ground truth", as an environment should be equal to itself. This test consists of doing a diversity analysis on a set of size 2, where both environments contained are the same. We expect the value to approach zero, but allow for a slight deviation, as the method is built on approximations.

4.7.3 Untrained Expert Agents

To provide further context for both the scores from the identity test and diversity analysis, we run the algorithm with zero training steps. In this experiment, only full state distributions are used, as the agent memory is empty.

4.7.4 Diversity

Moving from environment pairs to sets, we extend select comparison metrics to compute diversity. Both hand-picked and randomly sampled sets are measured, and multiple task domains are represented.

There are two main interpretations of stability to be tested:

1. How the diversity score converges with respect to training iterations in the expert agents.
2. Whether multiple repetitions of the whole algorithm with different random seeds estimate roughly the same diversity score.

The following diversity function configurations will be tested:

- **dqn-full** DQN[2] agents with a full state distribution
- **dqn-mem** DQN agents with a memory-based state distribution
- **sm-dqn-full** DQN agents with a full state distribution and softmax values
- **rein-full** REINFORCE agents with a full state distribution

[2]Technically, since linear models are used for these experiments, the "Deep" in DQN is misleading. The learning algorithm is the same, and for more complex environments deep network would be necessary. But in this particular case, the setup is more akin to the original experience replay scheme (Lin, 1992)

- **rein-mem** REINFORCE agents with a memory-based state distribution

The first two are occasionally grouped as *value-based metrics*, and the final two as *policy-based metrics*, while the softmaxed variant is considered as a hybrid.

4.7.5 Multi-Task RL

The final suite of experiments aims to test the central hypothesis on how diversity affects performance, outlined in figure 4.1. The environment sets used are the same as those who will be featured during a mass diversity analysis.

Chapter 5

Results

In the previous chapter we presented an algorithm for diversity analysis and its configurations. In this chapter we showcase its usage in practical experiments. The experimental part of this project serve as a field test of both the implementation as well as the theoretical concept of diversity.

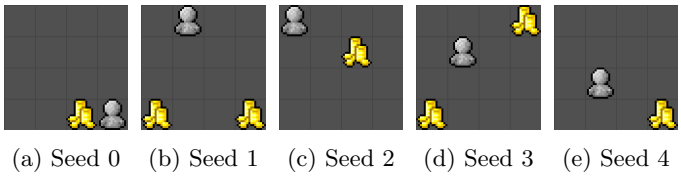
5.1 Environment Baselines

A selection of environments were sampled, and different agents were deployed to record the number of time steps until termination. The algorithms are not configured beyond default hyper-parameter values, as the intention is not to showcase optimal behaviour, or even compare agents, but rather to establish a context for the experiments to follow.

Seed →	0	1	2	3	4	5	6	7	8	9
RandomAgent	15	83	23	79	50	67	58	63	52	45
DQN	3	34	11	40	11	38	29	20	14	18
REINFORCE	8	46	9	67	9	44	36	32	16	33

Table 5.1: Mean solution times for three different agents. The environment class is 4 x 4 gridworld:collect, with seeds 0 through 9.

One key observation from this experiment is that while random acting is overall the worst policy, it does solve the environments in reasonable time. This suggests that the training is not dependent on extensive exploration. Secondly, the distribution of "difficulty" is clearly visible, as the relative solution times between ten seeds vary quite a bit.



(a) Seed 0 (b) Seed 1 (c) Seed 2 (d) Seed 3 (e) Seed 4

(f) Seed 5 (g) Seed 6 (h) Seed 7 (i) Seed 8 (j) Seed 9

Figure 5.1: Renders of the initial state of the first 10 environments in *gridworld:collect*

5.2 Diversity

Before showing results from diversity analysis, we establish a frame of reference by presenting some general observations about the values:

- For value-based metrics (**dqn-full** and **dqn-mem**), all recorded diversity scores lie roughly within the range [4, 14]. This excludes sets containing only identical environments.

- If the agents are not trained, or trained very little, the diversity score is typically low, as the agent models output random values. More on this is featured in section 5.2.2

5.2.1 The Identity Test

The first test is designed to test the only known ground truth of our comparisons: that something is equal to itself. Because the comparison metrics all return the *difference* between two environments, this test expects the value zero when applied to a set of two identical copies. Because the metrics are based on approximations, the value is expected to approach zero within an acceptable margin.

Seed →	0	1	2	3	4	5	6	7	8	9
Run 1	1.28	0.00	0.05	0.00	0.12	0.00	0.04	0.11	0.42	0.33
Run 2	0.94	0.00	0.07	0.00	0.10	0.12	0.03	0.00	0.00	0.06
Run 3	0.09	0.07	0.45	0.00	0.03	0.00	0.04	0.21	0.01	0.06
Run 4	0.73	0.00	0.42	0.00	0.03	0.01	0.04	0.00	0.01	0.17
Run 5	0.32	0.01	0.85	0.00	0.03	0.01	0.12	0.01	0.04	0.03
Avg.	0.67	0.02	0.37	0.00	0.06	0.03	0.06	0.07	0.10	0.13

Table 5.2: The identity test performed on the ten first environments in 4 x 4 gridworld:collect using **dqn-full**. This is done over five independent runs.

By running the identity test on ten random environments, we observe that for all but two (seed 0 and 2, visualized in Figure 5.1 (a) and (c)), the score is low. Upon investigating the deviant cases, we see that the environment has a goal state neighboring the initial state. In a traditional RL setup, this simply makes the training converge really fast, but in our case it can lead to a bad value-function approximation, because most reachable states are unlikely to be visited before termination.

To test whether the score continues to decrease with more training, we run an extended session on each environment, and show the results in table 5.3. For every environment, except for seed 2, this leads to a significantly lower value, which further confirms that the diversity approximation passes the identity test under the right conditions. The main reason why we are confident in these results is that the observed values are far below the scores produced by sets of different environments, which means that the algorithm clearly distinguishes between sets that are with and without diversity.

0	1	2	3	4	5	6	7	8	9
0.090	0.016	0.556	0.009	0.036	0.045	0.005	0.002	0.002	0.005

Table 5.3: Extended training of value functions for identity test

Seed →	0	1	2	3	4	5	6	7	8	9
Run 1	1.51	0.04	0.08	0.03	0.04	0.00	0.00	0.02	0.01	0.14
Run 2	0.07	0.07	0.01	0.01	0.38	0.02	0.00	0.05	0.01	0.01
Run 3	1.09	0.03	0.57	0.01	0.10	0.06	0.01	0.10	0.36	0.16
Run 4	0.44	0.01	0.11	0.03	0.29	0.16	0.02	0.01	0.09	0.06
Run 5	0.53	0.06	0.01	0.00	0.05	0.01	0.02	0.00	0.09	0.11
Avg.	0.73	0.04	0.16	0.02	0.17	0.02	0.01	0.04	0.11	0.09

Table 5.4: The identity test performed on the ten first environments in 4 x 4 gridworld:collect using **dqn-mem**. This is done over five independent runs.

Interestingly, using the replay buffer does not solve this. Unlike the full state distribution, which is uniform, the replay buffer only contains visited states, and duplicates of states that are visited multiple times.

5.2.2 Untrained Diversity Analysis

Table 5.5 shows the results from performing diversity analysis with no training. The values are relatively consistent, and they seem to be independent of the environment set. An interesting observation is that **rein-full** and **sm-dqn-full** report the same values, this is expected, since they both have softmax applied to their outputs, but as training progresses (table 5.7), their value ranges are completely different. Another observation is that the previous cases where the identity test failed report a diversity score that is higher than the values in this experiment.

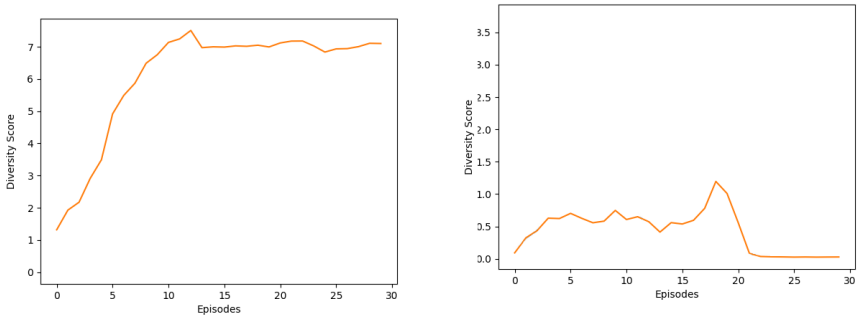
	dqn-full	**rein-full**	**sm-dqn-full**
Set 1	0.132	0.0064	0.0068
Set 2	0.134	0.0059	0.0064
Set 3	0.134	0.0066	0.0063
Set 4	0.138	0.0063	0.0059
Set 5	0.144	0.0069	0.0066

Table 5.5: Averaged results from an untrained diversity analysis of 5 different environment sets

5.2.3 Diversity Analysis of Hand-picked Sets

Unlike the identity test, the next set of experiments do not have a specific expected value. For these tests we are more interested in observing the *range* of values that are produced when comparing environments of different sizes. Figure 5.2 shows the difference between two pairs of hand-picked environments. For (a) the player and gold are in opposing corners of the grid, and their positions

are switched in the second environment in the set. In (b) we are observing environments where the gold position is the same, but the agent starts at different positions.



(a) Diagonally mirrored environments

(b) The same environment but with different initial states

Figure 5.2: Comparing hand-picked environments using a value-based metric

To provide an insight to what one set could look like, figure 5.3 shows a set has a value-based diversity score of 7.5. Compared to other results, this is a relatively low diversity.

(a) Seed 0 (b) Seed 2 (c) Seed 4 (d) Seed 7 (e) Seed 8

Figure 5.3: This set of 5 environments (4 x 4, seed 0, 2, 4, 7 and 8) has an estimated value-based diversity of about **7.5** (dqn-full)

5.3 Stability

To measure stability, we are looking for two different patterns:

- That the metric converges as the expert agents are trained more
- That the produced values are roughly the same

We observe these patterns in two different ways: first by plotting how diversity changes over training, and later by including the standard deviation of the measurements.

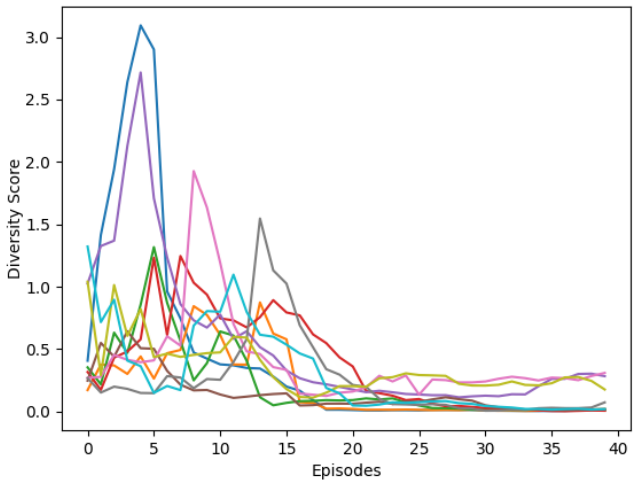
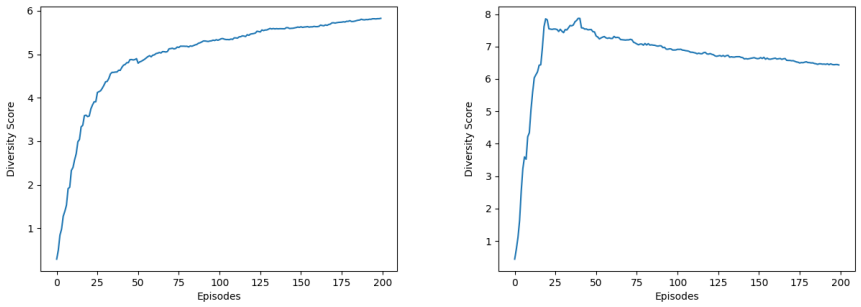


Figure 5.4: Ten independent runs of the identity test to showcase how the score changes with expert agent training. Every point on the x-axis corresponds to 100 episodes of training, and the y-axis represents the estimated difference.

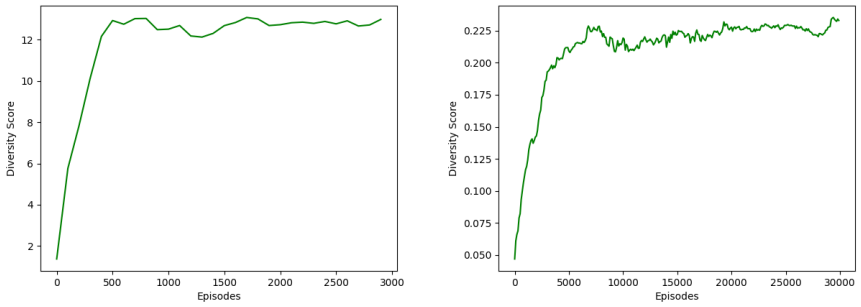
Figure 5.4 showcases the identity test repeated ten times. This is done both to show how the value-based metric converges as the expert agents are trained, as well as to see if they converge towards the same value. A majority of the runs converge near zero after about 3000 episodes, while the rest have a notable offset.



(a) Underestimation (b) Overestimation

Figure 5.5: Two different runs of a diversity analysis (dqn-full) on the same environment set. This example is meant to illustrate that the metric is converging towards a specific value, instead of just continuously increasing/decreasing.

To showcase that the diversity score is quite consistent, and that further training is likely to stabilize the value further, figure 5.5 shows that the score during training appears to settle towards circa 6. This is supported by the observation that the score is approached both from below (subfigure (a)) and from above (subfigure (b)). A similar pattern of overshooting and undershooting the score during early training was seen in other experiments throughout the project.



(a) Value-based diversity, score: 12.98 (b) Policy-based diversity, score: 0.23

Figure 5.6: Measuring diversity during training for both policy- and value-based metrics. The environment set is 4 x 4 gridworld:collect, seeds [115, 116, 117, 118, 119]

To show how value-based (**dqn-full**) and policy-based (**rein-full**) converge during training, we show them side by side in figure 5.6. As the underlying training algorithms are fundamentally different, they have a different

5.4 Scaling

To test scaling, we run a diversity analysis on sets of increasing sizes, and measure the time usage. A secondary observation is that the diversity scores produced during this appears to be within the same range.

Set Size	Diversity Score	Total Time
5	8.01	723s
10	9.64	2067s
15	9.53	4081s
20	10.53	6714s
25	9.20	9290s
30	9.11	11028s

Table 5.6: Diversity analysis of different set sizes (dqn-full)

As seen in Figure 5.7, the time usage of the diversity function appears to scale

linearly with the number of environments (at least until size 30). This suggests that the agent training is the bottleneck.

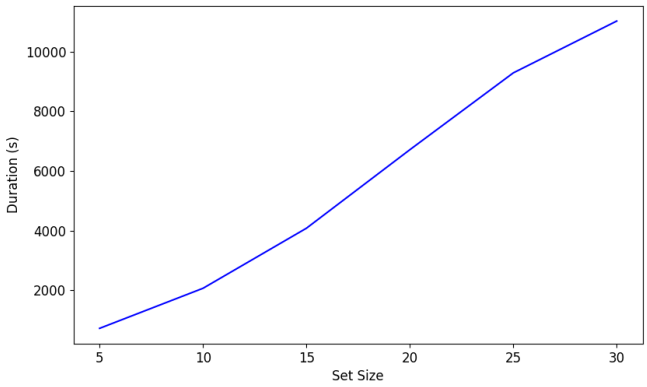


Figure 5.7: Wall clock duration (seconds) of a diversity analysis of different environment sizes. Generated from table 5.6

5.5 Diversity Function Configurations

Table 5.7 shows the results from a mass diversity test, where 5 environment sets are analyzed by all metrics several times. We are interested in multiple relationships in these results:

- The difference between the state distributions (column 1 vs 2, and 4 vs 5)
- The stability of each measurement (standard deviations, in parentheses)
- How the diversity scores rank against each other, and whether this ranking is similar across metric configurations.

These results will be discussed further in the next chapter, but some immediate observations to note are:

- **sm-dqn-full** reports diversities that are *below* the initial values from no training.
- **rein-mem** scores above 1.0 on set 4, which is unexpected, as the method is supposed to be normalized.
- Set 1 has the lowest diversity, according to all configurations
- Set 4 has the highest diversity, according to the metrics using a full state distribution

	dqn-full		dqn-mem		sm-dqn-full		rein-full		rein-mem	
	div	std	div	std	div	std	div	std	div	std
Set 1	5.29	(0.040)	5.03	(0.101)	0.0012	(0.00016)	0.64	(0.004)	0.50	(0.029)
Set 2	8.31	(0.054)	7.06	(0.307)	0.0019	(0.00045)	0.70	(0.021)	0.67	(0.012)
Set 3	9.03	(0.370)	11.82	(1.05)	0.0020	(0.00035)	0.70	(0.030)	0.68	(0.039)
Set 4	9.35	(0.591)	7.68	(0.532)	0.0029	(0.00018)	0.92	(0.048)	1.05?	(0.039)
Set 5	9.14	(0.135)	6.54	(0.370)	0.0022	(0.00016)	0.89	(0.049)	0.74	(0.069)

Table 5.7: Results from a diversity analysis of 5 different environment sets, using every configuration of the diversity function

5.6 Diversity & Multi-Task Training

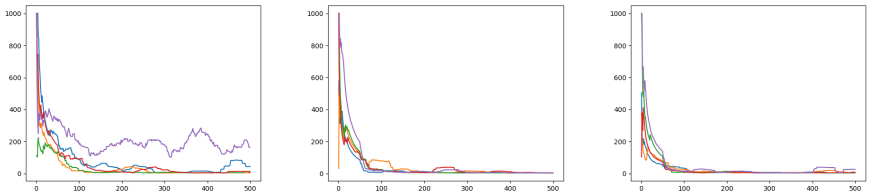
Finally, we present a mass multi-task training session where three different models are trained on each of the environment sets featured in the diversity analysis of table 5.7. All sessions are run with a DQN of the same hyper-parameter configuration, the only difference is the environment set and the neural network model.

Model 1 8 neurons in the hidden layer
Model 2 16 neurons in the hidden layer
Model 3 32 neurons in the hidden layer

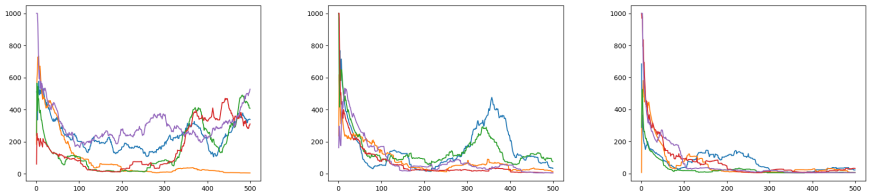
The horizontal axis represents training time, from 0 to 500 episodes. The vertical axis shows how fast the agent solves an environment on each episode. The values are smoothed by representing each step as an average of the previous 50 steps of raw values. Each session is repeated for five independent runs, and the runs are plotted together with different colors. For each episode, an environment is randomly sampled from the set.

Some of the most important observations in this experiments are the following:

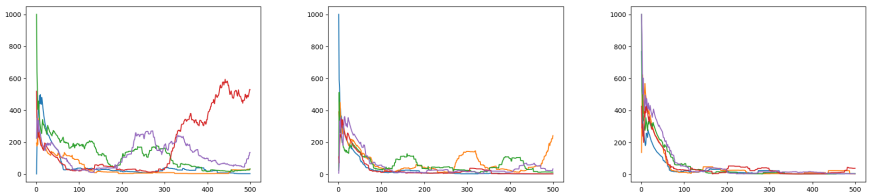
- The smallest model (Model 1), performs reasonably well on set 1, which has the lowest diversity, but it struggles with the other sets.
- The larger models (Model 2 and 3) generally perform better, with near-perfect convergence in figure 5.8 (b), (c) and (i)
- Among the settings that struggle, we observe both instability (e and g), as well as failure to converge at all (d and m)



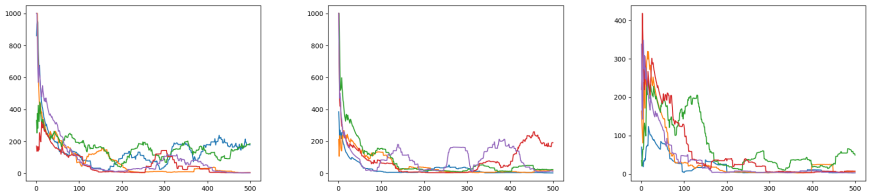
(a) Set 1, Model 1 (b) Set 1, Model 2 (c) Set 1, Model 3



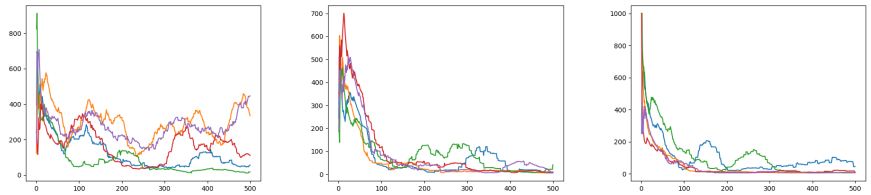
(d) Set 2, Model 1 (e) Set 2, Model 2 (f) Set 2, Model 3



(g) Set 3, Model 1 (h) Set 3, Model 2 (i) Set 3, Model 3



(j) Set 4, Model 1 (k) Set 4, Model 2 (l) Set 4, Model 3



(m) Set 5, Model 1 (n) Set 5, Model 2 (o) Set 5, Model 3

Figure 5.8: Result from training sets of different diversities with different models.

Chapter 6

Discussion & Conclusion

Most of the results presented in the previous chapter are interesting from multiple perspectives. In this final chapter we will discuss the general patterns observed, and carefully interpret their significance beyond the experiments within this thesis. Finally we conclude by summarizing the main takeaways of this project, and look to the future both in terms of development and research.

6.1 Diversity Analysis

We are testing the diversity function approximator in three different ways. First we run the identity test, which is the only test with an absolute ground truth. This is the only test that can be concluded as a pass or a fail. The second way of testing is to apply it to different environment sets, and in this situation the score itself is less meaningful, and instead we focus on performance metrics such as stability and how different configurations relate to one another. Finally, we focus on the actual diversity scores, and attempt to map out their significance.

We recall that the following configurations of the diversity function are tested:

- **dqn-full** DQN agents with a full state distribution
- **dqn-mem** DQN agents with a memory-based state distribution
- **sm-dqn-full** DQN agents with a full state distribution and softmax values
- **rein-full** REINFORCE agents with a full state distribution
- **rein-mem** REINFORCE agents with a memory-based state distribution

6.1.1 The Identity Test

Through the identity test, we observe pairs of identical environments, expecting the diversity score to approach zero. Generally, the test is passed in our experiments, as the scores are clearly moving towards zero within a reasonable margin, but some environments fail. The common denominator of the failing environments is that they are easy to solve, and that a significant portion of the state space is critically under-explored. This is an important observation, as it emphasizes the role of exploration in diversity analysis. For certain very simple environments, less exploration can in fact be a good strategy for maximizing rewards, but it can lead to poor approximations of value functions. In a traditional RL setup, this might not be a concern, but in our case it is critical, as the value/policy approximations are central to our diversity analysis.

Larger, more complex environments are also prone to unexplored states, but we believe that it will be less of an issue when compared to the cases from the diversity test. This belief is based on the idea of *relative coverage* of the state space, where examples such as seed 0 (figure 5.1 (a)) fail to cover a large percentage of states during training. Despite this, a surprising observation was made when performing an identity test using a memory-based state representation. The faulty environment (seed 0) did not perform any better, which is surprising because unexplored states are omitted from the memory-based comparison. Seed 2, however, saw a significant improvement from this modification, which was expected.

Outside of the context of diversity, the identity test is interesting because it explores whether training sessions of different initial conditions converge towards the same value function.

6.1.2 Value Ranges & Normalization

The value range of the different configurations of the diversity function is unknown. For the normalized methods (**sm-dqn-full**, **rein-full** and **rein-mem**), the bounds are known, but that does not mean the full range [0, 1] is used. One of the results scored a value greater than 1.0, which could be an issue with the implementation. This result has not been reproduced yet, and is left as an investigation for the future. Normalization has not been a major focus in this project, as the value-based methods were developed first and showed interesting results early on, and because normalization is more important when scaling the system to handle comparison across domains with different reward scales.

One of the major weaknesses of value-based comparison is that the diversity score is heavily affected by the reward scaling in the environments. For example, if we take two identical *gridworld:collect* environments and double the rewards in one of them, a value-based metric would not pass the identity test. This is not an issue in our experiments, as all rewards are the same, but for future work it serves as a constraint that could be avoided. Possible directions for solving this is either to focus more on policy-based methods, or to look into methods of normalizing values, such as reward clipping (van Hasselt et al., 2016)

6.1.3 Training Expert Agents

When deciding when to stop training of the expert agents in order to compare them, a dilemma emerges: *further training could improve the approximations.* This is analogous to the classical *quantity versus quality* dilemma, where a more accurate diversity score will take longer to estimate. One possible approach to this is to stop training based on model loss instead of returns, as this should better reflect convergence in the approximation model, instead of the agent behaviour. This change should have a minimal effect on the results, as the expert agents are currently overtrained, but it would enable for a better condition for early stopping.

For this domain, the REINFORCE algorithm has a tendency to converge faster, in terms of wall-clock time. Whether this is related to hyper-parameters or the environment dynamics is not explored in this project, but fast convergence is generally desired, as diversity analysis is currently a slow process.

6.1.4 State Distributions

Two key observations can be made about memory-based state distributions:

1. Both distribution schemes perform similarly on the identity test.
2. Diversity scores are typically a bit below the score assigned by using full state distribution on the same set.

The first observation is surprising, as the error in the diversity test is believed to be related to under-explored areas in the state space. Using agent memory

to represent the state space weights states by recent visitations, and a high visitation count should in theory lead to a more confident value approximation. Another reason why this is surprising is outlined in the second observation, that memory-based analysis typically reports a lower score overall.

There are two main features that separate memory-based from the full distribution: the memory is not guaranteed to contain all unique states, and one state can appear more than once. These features, combined with the fact that memory changes across sessions, make up the three possible causes for instability in memory-based analysis. Other related configuration properties, such as agent exploration and memory length, are interesting candidates for future experiments.

6.1.5 Stability

When evaluating the stability of these methods, there are two aspects to look for: *convergence* and *consistency*. To consider a measurement converged, the value is expected to settle within a small range, and the variance is expected to decrease with further iterations. Several experiments show this behaviour clearly, with a value that quickly narrows down the bounds of the observed diversity. Another property of a stable convergence is that the model doesn't "forget" after a while, which is a general problem with neural networks (Kemker et al., 2017).

The factors that contribute to stability are the expert agents and the state distribution, as they are the only components of a diversity analysis that change during training. A full state distribution is static, but when using agent memory, the stability relies on whether the memory content is a good representation of the relevant areas of the state space. Agent stability is a an open topic in reinforcement learning (Nikishin et al., 2018), but in this system, the main concern is that the value and policy approximations are stable, not necessarily agent behaviour.

The second aspect of stability is consistency, and the main measure of this is the standard deviation across multiple independent runs. All the proposed configurations score reasonably well in this regard, and several experiments (most notably figure 5.5) suggest that this can improve with further training of the expert agents.

6.2 Environments

Both the individual environments, as well as the sets, are small in size for this project. This allows us to perform an extensive amount of experiments, which is essential to explorations of this nature. A major concern, however, is that the work becomes too abstract or artificial and doesn't apply to "real" environments. The results from the various tests outlined in chapter 5 definitely showcase some meaningful patterns, and it would be very surprising if these patterns

are exclusive to the specific domain and scale we have tackled. Claiming that the values measured are indeed representative of diversity would of course be premature, but according to our previously proposed definitions and hypotheses of environmental diversity the results are looking promising so far.

The experiments conducted in Cobbe et al. (2018) (see section 3.3) all concern environment sets where every task is sampled from the same domain. So far, our discussion of diversity has been under the same constraint. An interesting proposal for further experimentation is diversity across multiple (compatible) domains.

6.3 Scaling

One of the most important conditions for scaling diversity analysis beyond toy problems is that the full state distribution can be successfully replaced by the memory-based solution. This is because the full distribution is typically huge, possibly infinite, and likely to be unavailable. The agent memory, on the other hand, is a versatile solution, implemented as a fixed-size buffer where the only limiting factor is the shape and size of observation samples. Even the most complex environments, such as a real-world robot with a high-definition RGB observation feed would be possible to handle using a memory-based solution.

As observed in our experiments, memory-based comparison has a generally higher standard deviation. This is expected, as the content of the memory is very likely to vary between sessions. As briefly discussed in section 6.1.4, the scores deviate significantly from the ones produced with a full state distribution, which could be problematic for scaling. Potential solutions is to try different memory sizes, or implement concept similar to prioritized experience replay (Schaul et al., 2015). One interesting hypothesis is that this issue might be less significant with image-based observations, as they tend to change relatively few pixels between states, which could lead to less variance in the data contained in the memory.

6.4 Diversity & Multi-Task Training

The multi-task training experiments produced the main results linked to our hypothesis on the role of diversity (section 4.1). To recap, we want to investigate the relationship between model complexity, diversity and performance. Some of the central observations in this experiment include:

- The smallest model performs well on the least diverse set, and struggles on the more diverse sets.
- The larger models generally perform better
- Among the settings that struggle, we observe both instability, as well as failure to converge at all

All of these observation align well with our hypothesis, as the general pattern is that all high-diversity sets (2-5) require a larger model to be solved. This is of course by no means conclusive, both because we are operating within a restricted domain, and because there are other potential sources to poor performance than diversity. Some environments are simply harder to solve than other, and difficulty is not necessary linked to diversity, as multiple difficult environment can still be very similar to each other, according to our metrics. However, all environments in *gridworld:collect* should be relatively similar in terms of difficulty.

6.5 Conclusion

In conclusion of this thesis, it is tempting to make one or more definitive statements about diversity. Both this project and related works *suggest* that a diverse set of tasks is advantageous to a learning agent, but to our knowledge there is no definite proof available. However, in the restricted domain of our toy environments, the proposed diversity function approximator satisfies the requirements of producing consistent diversity scores, and it passes the identity test. This shows that we do approximate some characteristic of environment sets, but whether that characteristic is actually diversity, as it is defined in section 4.1, remains as a final black box.

6.5.1 Diversity Metrics

In section 4.7.4 we presented five configurations of the diversity function. Each has its own strengths and weaknesses, and there is no obvious "victor" emerging from the results. This does not mean that the results are inconclusive, but rather that the exact desired properties of the diversity function, such as normalized scores, is still an open discussion.

6.5.2 The Nature of Diversity

A core idea of this work is that diversity is a static property of any unique set of environments. Because the metrics are built on approximations, the estimated score is not expected to be strictly identical across multiple runs, but it should be evident that it is converging towards a "true" value. Even if the scores produced seem consistent and meaningful, they would be of little value if they had no connection to training and performance. Although this connection requires *extensive* testing to confirm, we begin the effort by recording the performance of agents that are trained on sets of a notably different diversity, according to our metrics. Our initial efforts suggest that there is a significant pattern present between diversity and multi-task agent performance, and we are excited to explore this further in the future.

6.5.3 Future Work

A natural next step in this project would be to explore *scaling*, both in terms of covering all possible environments, as well as performance when handling large environments. Several known environment sets, such as the ATARI 2600 suite could in theory be diversity measured at this point. The main condition for a computationally reasonable analysis is that the individual environments in the set can be solved significantly more easily than the multi-task setting. This is because the expert agent for each environment needs to converge in order to have a good value/policy approximation.

On the implementation side it would be natural to restructure the project as a proper analysis tool, instead of an experimental script collection. Some im-

portant properties of such a tool include compatibility with common machine learning libraries and OpenAI Gym (Brockman et al., 2016). A great selection of configuration parameters should be provided, as the optimal setup for a stable diversity analysis is far from established through this work, and likely to be dependent on factors such as the task domain and reward scale. While the system is currently limited to environments with the same observation shape, this could be bypassed for image-based environments by for example scaling observed images to be of the same size, and then compare them.

Optimization is also a major topic for future development, both because the current implementation is rather slow, and that the environments that are interesting to analyze are significantly larger. By using replay buffers as state distributions for comparison, the computational time for the comparison step is largely predictable. As the expert agents are trained in complete independence from one another, this step can be heavily optimized across parallel processing units. This requires that a proper configurations for the memory-based diversity analysis are explored, as they are currently quite unstable, as seen both from the standard deviation, as well as the mismatch with diversity scores produced by full state distributions.

In terms of further experiments it would be very interesting to dig deeper into the anatomy of sets of different diversities. A specific experiment towards this is to analyze the saliency maps (Greydanus et al., 2017) in a multi-task trained neural network for a visual environment, and compare it to that of a single-task one. This informs how different regions in a the observation images are used in both networks, which could be an important insight into how multiple tasks are managed by the agent.

As the field of reinforcement learning is continuously growing in several interesting directions, extending the coverage of diversity analysis is exciting. In meta-learning, where agents are trained to adapt efficiently when a new task is introduced, we can try to relate this efficiency to the diversity between the new task and the previously seen tasks. For multi-agent settings, we can analyze whether agents trained in the exact same environment instance perceive their task to be the same as the other agents. This is not only interesting for a pure diversity perspective, but also to tie multi-agent and single-agent RL together. Finally, we would like to extend into "true" multi-task learning, where the tasks are sampled from truly different domains. Extensive use should mature the methods introduced in this thesis, and after this initial presentation, we believe that these techniques, or at least the ideas behind them, could have the potential to enrich the development of AI.

Bibliography

M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents, 2012.

M. Bhattacharya. Diversity handling in evolutionary landscape, 2014.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning, 2018.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2019.

T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–??, 2000. URL `citeseer.nj.nec.com/dietterich00ensemble.html`. It is a good classic article reviewing ensemble methods. He shows intutively why ensembles is a good idea: Statistical, computational, representational.

A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator, 2017.

D. Dowe and A. Hajek. A non-behavioural, computational extension to the turing test. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications*, pages 101 – 106, Singapore, 1998. World Scientific Publishing. ISBN 981 02 3352 3. International Conference on Computational Intelligence and Multimedia Application, ICCIMA 98 ; Conference date: 07-02-1998 Through 10-02-1998.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control, 2016.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function, 2018.

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

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding atari agents, 2017.

J. Hernandez-Orallo. A (hopefully) non-biased universal environment class for measuring intelligence of biological and artificial systems. *Artificial General Intelligence - Proceedings of the Third Conference on Artificial General Intelligence, AGI 2010*, 06 2010. doi: 10.2991/agi.2010.18.

M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task deep reinforcement learning with popart, 2018.

R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan. Measuring catastrophic forgetting in neural networks, 2017.

L. Kuvayev and R. S. Sutton. Model-based reinforcement learning with an approximate, learned model. In *in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pages 101–105, 1996.

K. Lee, K. Lee, J. Shin, and H. Lee. Network randomization: A simple technique for generalization in deep reinforcement learning, 2019.

S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence, 2007.

S. Legg and J. Veness. An approximation of the universal intelligence measure, 2011.

L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, USA, 1992.

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

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in rl, 2018.

E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. P. Vetrov, and A. G. Wilson. Improving stability in deep reinforcement learning with weight averaging. 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

E. A. Platanios, A. Saparov, and T. Mitchell. Jelly bean world: A testbed for never-ending learning, 2020.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 12 2002. ISBN 0137903952.

A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation, 2015.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015.

A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills, 2019.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 1 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

R. B. Slaoui, W. R. Clements, J. N. Foerster, and S. Toth. Robust visual domain randomization for reinforcement learning, 2019.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423. URL `http://www.jstor.org/stable/2251299`.

E. van der Pol, T. Kipf, F. A. Oliehoek, and M. Welling. Plannable approximations to mdp homomorphisms: Equivariance under actions, 2020.

H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude, 2016.

O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019. doi: 10.1038/s41586-019-1724-z.

P. Wills and F. G. Meyer. Metrics for graph comparison: A practitioner's guide, 2019.

Appendix A: Implementation Details

Kvad: Gridworld Framework

Through this project, the environment framework *kvad* was developed. Kvad is short for "kvadrat", which is the Norwegian word for square, and the shortened form "kvad" is also the word for ancient Norse poems, which fits well, because these environments all consist of a square grid, and they are simplistic and short, and designed to prove specific points. The design philosophy behind the framework is to arrange environments into a hierarchical structure, using inheritance for functionality such as movement and collisions. This aligns with the underlying theme of this thesis, which is to compare environments, as they have shared features on an implementation level, which serves as a good base for comparability.

Repositories

Diversity Analysis (NumEDAL=Numerical Environment Diversity Analysis Library):
https://github.com/Jontahan/numedal
Author is the only contributor

Environment:
https://github.com/Jontahan/kvad
Author is the only contributor

RL Framework:
https://github.com/CogitoNTNU/vicero
Author is a main contributor

Machine Learning Back-end:
https://github.com/pytorch/pytorch
Paszke et al. (2019)

Experiment Manager

For convenience, an experiment manager was implemented for this project. The main advantages and responsibilities of such a system include:

- To handle all file IO, making sure no data is lost, either to not saving, or to overriding. One sub-directory for each experiment is created in the ouput folder, and checks are made to prevent filename conflicts.
- To handle multiple repetitions of the same experiment. When dealing with potentially unstable systems, it's important to verify everything by running it multiple times. The experiment manager takes care of this, including writing the results of each iteration to separate files, with suffix indexing.
- To contain most (if not all) relevant data required to reproduce the results of an experiment. This is implemented by writing all parameters to a separate file, as well as information such as start time and duration.

Implementation wise, the manager is supplied with a function object that returns a list of outputs, and a dictionary containing the parameters to this function. This allows for the main experimentation scripts to be short and clean.

While not a part of the manager, a similar tool was created to deal with plots, and the various labels and configurations involved in creating those. It cooperates with the manager by using the data files as input for the actual plot content.

Appendix B: Experiment Details

Hardware

Most experiments were executed on a personal laptop.

CPU: Quad-core Intel Core i7-5500U 2.4GHz
RAM: 8GB

Statistics

 # expert agents trained: ~ 10000
 # diversity scores computed: ~ 1000
 # unique environments tested: ~ 200
total accumulated time spent training: 1-2 months

NTNU

Kunnskap for en bedre verden