

XmodRL: Explainable modular reinforcement learning

Master thesis by Dwarak Adugodi Vittal
Date of submission: 12. November 2021

1. Review: Prof. Dr. Kristian Kersting
2. Review: Quentin Delfosse and Johannes Czech
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department
AIML

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt


Hiermit versichere ich, Dwarak Adugodi Vittal, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 12. November 2021



D. Vittal

Abstract

Most of the existing approaches for extracting explanations from deep neural networks, frequently used in reinforcement learning algorithms, are categorized as Post-hoc explainability approaches, even though transparent algorithms deliver better explanations. In addition, humans tend to over-interpret the results of these approaches for formulating partly false behavior explanations. This thesis presents XmodRL: A reinforcement learning pipeline designed to understand the behavior of a reinforcement learning agent by being modular and having human-understandable intermediate results. The reinforcement learning task is split into subtasks. For each subtask, exchangeable solvers are used. Later, the thesis presents experiments representing implementations of this pipeline. The experiments show the benefits of using the presented pipeline for understanding the trained agent's behavior. The framework¹ implementing the pipeline and used in the experiments is publicly available.

Keywords: Reinforcement learning, XAI, XRL, Explainability, Modular

¹<https://github.com/dwabii95/XmodRL> (visited 10.11.2021)

Zusammenfassung

Viele der existierenden Ansätze, Erklärbarkeit aus oft durch Reinforcement Learning-Algorithmen verwendeten tiefen, neuronalen Netzen zu extrahieren, lassen sich als Post-Hoc-Erklärbarkeitsansätze klassifizieren. Dies ist der Fall, obwohl transparente Algorithmen bessere Erklärungen liefern. Dazu tendieren Menschen dazu, Ergebnisse aus Post-Hoc-Erklärbarkeitsansätze zu über-interpretieren und dabei teilweise falsche Verhaltensklärungen aufstellen. Diese Thesis präsentiert XmodRL: Eine Reinforcement Learning-Pipeline, welche designed wurde, um leichter das Verhalten von RL-Agenten zu verstehen. Dies wird durch die eigene Modularität und verständliche Zwischenergebnisse ermöglicht. Die RL-Aufgabe wird in Teilaufgaben aufgeteilt, bei der jede Teilaufgabe einen austauschbaren Aufgabenlöser besitzt. Danach werden Experimente vorgestellt, bei der verschiedene Implementierungen der Pipeline evaluiert werden. Es werden die Vorteile bezüglich des Verständnisses vom Verhalten eines Agenten gezeigt, die das Implementieren der Pipeline mit sich bringt. Das Framework², welches die Pipeline implementiert und für die Experimente verwendet wird, ist frei verfügbar.

Stichworte: Reinforcement learning, XAI, XRL, Explainability, Modular

²<https://github.com/dwabii95/XmodRL> (visited 10.11.2021)

Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Formulation	2
1.3 Outline	4
2 Background	5
2.1 Reinforcement learning algorithms	5
2.1.1 Q-Learning and DQN	6
2.1.2 REINFORCE	7
2.1.3 Deep GA	7
2.1.4 DreamerV2	8
2.2 Technical Background	9
2.2.1 OpenAI Gym	9
2.2.2 The Atari Annotated RAM Interface (AtariARI)	10
3 Related Work	11
3.1 Post-hoc explainability based related work	11
3.1.1 Integrated Gradients	11
3.1.2 Saliency maps for interpreting the agent's behavior	12
3.2 Transparent algorithm design based related work	12
3.2.1 SPACE	12
3.2.2 Explainable Hierarchical Reinforcement Learning for Robotic Ma- nipulation	14
4 Designing an explainable reinforcement learning pipeline	15

4.1	Extracted primary goals from previous work	15
4.2	Overview of the hierarchical task splitting	16
4.3	Approaches for $T1$	18
4.3.1	Supervised vs. unsupervised learning	18
4.3.2	Applying Post-Hoc explainability methods	19
4.4	Approaches for $T2$	20
4.4.1	Meaningful features for position coordinates	20
4.5	Approaches for $T3$	21
4.5.1	Training a model based on given features	22
4.6	Comparing the presented pipeline with the major goals	22
4.6.1	Major goal 1: Subgoals of minimal size	22
4.6.2	Major goal 2: Understandable intermediate results	23
4.6.3	Major goal 3: Interchangeable solver	23
5	Experiments	24
5.1	Experiment setup	24
5.2	RAM Annotations as raw features for reinforcement learning algorithms . .	25
5.2.1	Experiment results of using raw features for Pong	26
5.3	SPACE for $T1$	26
5.3.1	Evaluation of SPACE for Pong	27
5.4	Handcrafted meaningful features for reinforcement learning algorithms . .	29
5.4.1	Experiment results of using handcrafted meaningful features for Pong	31
5.5	First generalized approach for meaningful features	32
5.5.1	Experiment results with generalized meaningful features	33
5.6	Automatic feature selection with pruning	33
5.6.1	Feature weight-based pruning	35
5.6.2	Integrated gradient-based pruning	36
5.6.3	Correlation-based pruning prior training	36
5.6.4	Comparing the presented pruning methods	37
5.7	Minimizing the policy model	38
5.8	Model-based approach of explainable reinforcement learning	39
5.8.1	Minidreamer: A simplified version of DreamerV2	40
5.8.2	Evaluating Minidreamer's first implementation	40
6	Exemplary investigation for a polices behavior	42
6.1	Using integrated gradient values to interpret behavior	42
6.2	Interpretation approach of a linear model policy	43

7	Conclusion and Outlook	45
7.1	Conclusion	45
7.2	Outlook	46
A	Appendix	xiii
A.1	Training hyperparameter and configuration	xiii
A.1.1	General algorithm default hyperparameter	xiii
A.1.2	Specific hyperparameters changes for the experiments	xiv
A.2	Training results	xv

List of Figures

1.1	Representation of the performance-explainability tradeoff	2
1.2	An example of hierarchical learning for playing a game	3
2.1	The principle of reinforcement learning	5
2.2	One iteration of DeepGA visually shown.	8
3.1	The principle of SPACE	13
4.1	Split subtasks for the RL task to solve an Atari2600 game.	16
5.1	DeepGA reward graph with raw features	27
5.2	Reward graph with SPACE output as input	28
5.3	Visual evaluation of SPACE	29
5.4	Illustration of calculating movement trajectory distances	30
5.5	Entropy histogram and correlation map for meaningful features.	34
5.6	Reward graphs for threshold pruning	35

List of Tables

2.1	AtariARI features for Tennis, Pong, and Boxing	10
5.1	Experiment results for using raw features	26
5.2	Results for handcrafted meaningful features	31
5.3	Experiment results with the generalization of meaningful features	33
5.4	Pruning experiment results	38
5.5	Experiment results for using linear models	39
5.6	Statistical values about the predicted state	41
6.1	The highest and lowest IG-Values, showing feature importance	42
6.2	IG-Values for different game situations	43
6.3	Weights for each input feature showing their influence	44
A.1	Default DeepGA hyperparameters	xiii
A.2	Default REINFORCE hyperparameters	xiii
A.3	Default DQN hyperparameters	xiv
A.4	Default Minidreamer hyperparameters	xiv
A.5	Specific hyperparameter changes for experiment 5.4	xiv
A.6	Specific hyperparameter changes for experiment 5.5	xiv
A.7	Pruning hyperparameter for experiment 5.6	xv
A.8	Absolute training scores from experiment 5.2	xv
A.9	Absolute training scores from experiment 5.3	xv
A.10	Absolute training scores from experiment 5.4	xv
A.11	Absolute training scores from experiment 5.5	xv
A.12	Absolute training scores from experiment 5.6	xvi
A.13	Absolute training scores from experiment 5.7	xvi

1. Introduction

1.1. Motivation

Machine learning models, especially reinforcement learning models, have gotten more powerful and flexible over the last decades [25, p. 1]. Together with deep learning for learning (Deep RL) feature representations, the performances of these models tend to be impressive in many areas [18, p. 1]. Due to the performances, Deep RL has arrived in the global industry and is used in various areas [18, p. 2]. Unfortunately, the understanding of these models is decreasing correlatively (see 1.1) [25, p. 1]). Since Deep RL algorithms must be explainable in many use cases, this trend of having less understandable and explainable models needs to be challenged [18, p. 2]. The reason why a model must be explainable can differ depending on the use case and therefore could be trustworthiness, causality, transferability, informativeness, fairness, confidence, accessibility, interactivity and privacy awareness [18, p. 2]. For example, a medical diagnosis system using deep learning to help the diagnostic must be transparent, understandable and explainable. Otherwise, the physicians, regulators and the patients cannot fully trust the result of the medical diagnostic system [27, p. 1]. Consequently, they cannot use the decision made by the medical diagnosis system. Since due to the critical application area of this system, it is mandatory to get the explanation of the given decision the system has made [27, p. 1-2].

Getting explainability into a Deep RL algorithm can be achieved through different approaches. These approaches can be categorized in two different categories: **Transparent algorithms** and **Post-hoc explainability** [18, p. 6]. Transparent algorithms get their explainability by the design of themselves. These include representation learning, where the algorithm learns abstract features from the given data, which could help to explain better the decision of the resulting model, simultaneous learning, where the algorithm learns the policy and the explanation simultaneously, and hierarchical learning, where the algorithm learns to divide its main goal into subgoals, which helps to explain some decisions the resulting model takes [18, p. 6-11]. Post-hoc explainability describes the

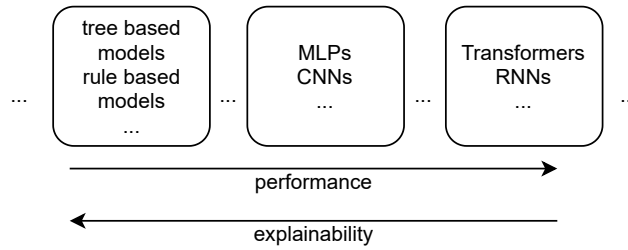


Figure 1.1.: Simplified illustrative representation of the performance-explainability trade-off. Adapted from [25, p. 2]

approaches that are applied after the origin RL algorithm has already trained the model. It extends the trained model with parts that are trying to explain the model. The types of extensions include approaches of using saliency maps, which describes the highlighting of salient elements on given data to create the saliency maps, and interaction data, which describes the statistical analysis of the given interaction data between the RL Model and the given environment [18, p. 13-14].

However, most of the existing works towards explainable Deep RL fall into the category of Post-hoc explainability [25, p. 15]. Since most of the RL-Tasks are highly complex tasks compared to areas with un-/supervised models, it is simpler to add an explainable component to an existing Deep RL model than to redesign the model to be more explainable. As a consequence, their approach category will change to the transparent algorithm approach [25, p. 15]. Using Post-hoc explainability approaches also comes with the costs of accuracy/performance compared to the unmodified Deep RL model [25, p. 15].

1.2. Problem Formulation

Considering the current existing work towards explainable Deep RL mentioned in the last section, it would be more practical to use an approach of transparent algorithms with Post-hoc explainability to parts of the transparent algorithm, which are still not explainable enough. The most prominent issue about the explainability of Deep RL algorithms is seeing them as black boxes [25, p. 1] for the whole given task. Therefore, it should make sense to prioritize applying hierarchical learning by splitting the task into subtasks. It would create smaller, separate algorithms which typically need less effort to explain their behavior. For example, the task is playing an Atari game with the game screen as input. Here, the Deep

RL algorithm has first to extract raw features from the given image (e.g. position of the objects), process them to meaningful features (f.e. distance between objects, speed of objects), and finally learn a policy with the given processed meaningful features. Applying hierarchical learning would mean using different algorithms for each of these tasks instead of one Deep RL algorithm. Theoretically, it should be easier to get explainability into an algorithm, which only extracts raw features from an image or an algorithm that processes meaningful features from raw features instead of getting explainability into the algorithm that does everything.

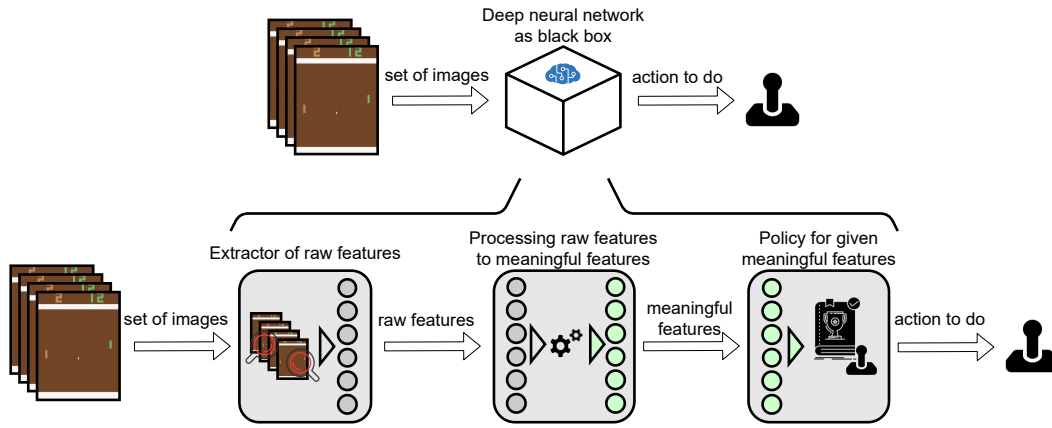


Figure 1.2.: An example application of using hierarchical learning for the RL Task to play a game.

This thesis presents approaches that apply the proposed idea by either choosing existing methods or designing new ideas for each subtask. The different solvers for the subtasks intend to have an entire pipeline of small, human-understandable parts, creating an explainable and interpretable solution for the whole Deep RL task. The focus of the presented pipeline and its performance evaluation is between already having the raw features, converting them to meaningful features, and generating a policy. The experiments to evaluate the chosen and designed solutions for the subtasks gets various Atari2600 games assigned as the Deep RL task. The different approaches presented in this thesis will be evaluated, compared, and discussed.

1.3. Outline

In the beginning, this thesis presents and explains the foundations and background information needed. This includes the reinforcement learning algorithms used and the technical background for the implementation made for this thesis. It is followed by reviewing the existing approaches for getting explainability into reinforcement learning models. The shown approaches are categorized into their kind of approach. In addition, the chapter highlights their strong points and weaknesses. The subsequent chapter describes the design of the reinforcement learning pipeline addressing the shortcomings highlighted in the previous chapter. This includes defining every part of the pipeline, its possible implementation, and its advantages and possible weaknesses.

After the theoretical part of this thesis, the following chapter presents the experiments and their results. The experiments represent different attempts to implement the pipeline shown in the previous chapter. The first experiments evaluate ideas, how the pipeline could split the reinforcement task and what kind of solver for the subtasks could be used while still achieving an acceptable performance for the given task. The following experiments evaluate attempts to either have intermediate or final results with higher quality or simplify the solver for the subtasks. The next chapter describes exemplary attempts to understand the behavior of a trained reinforcement learning model by taking some of the implementations from the experiments. It highlights the benefits of implementing the pipeline. Finally, the last chapter summarizes the main points of this thesis with an outlook on how future researchers could extend the presented pipeline or implementation attempts.

2. Background

2.1. Reinforcement learning algorithms

A typical reinforcement learning algorithm is defined inside a decision-making environment played by an agent by its state space, a set of actions, and the outcome of applying an action in the current state to get the next state combined with the reward of using the last action [8, p. 1][24, p. 2]. How the agent learns to decide which action to take at which state differs the reinforcement learning algorithms to each other.

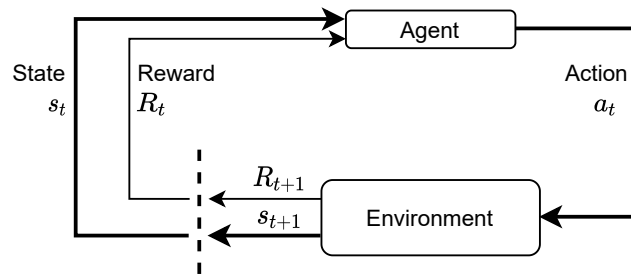


Figure 2.1.: The principle of reinforcement learning described by Sutton [30]

There exist two categories of reinforcement learning algorithms: *Model-free* and *model-based* approaches. Model-based approaches create an internal model of the transitions and outcomes of the environment. The agent chooses its actions for the existing environment by planning it with its internal model. Model-free approaches directly learn the state/action values, a policy, or both with the given experience. In other words, the agent learns how to behave in the environment without having an internal model. With a policy, a state has a *value*, representing the expected future from the current state applying the action from the policy. With correct values applied to the given states, the agent knows which action to take at each state to have the best possible future outcome [8, p. 2].

This thesis will mention model-free approaches (q-Learning/DQN, REINFORCE, Deep GA) and a model-based approach (DreamerV2) to compare the impact of different feature extraction approaches and their effects on different kinds of reinforcement learning algorithms. Therefore, these algorithms will be shortly introduced.

2.1.1. Q-Learning and DQN

Q-Learning is a model-free reinforcement learning algorithm in which the actor learns to act optimally inside the markovian domain by experiencing the outputs of its actions. The agent tries an action at a given state. It evaluates the given consequences in the form of reward and evaluates the value of the given state/action pair. It repeatedly tries every action at every state to learn the optimal action inside the markovian domain in every given state [33, p. 1]. The values optimized by q-Learning are called Q Values. A Q value represents the expected discounted reward for using action a at state s while following policy π afterward. The discounted reward describes rewards that are received s steps away are less important than rewards that are received now [33, p. 2]. After each step i , the Q Value with the current state as s_i and the selected action as a_i is updated with the *Bellman Equation*:

$$Q(s_i, a_i) = (1 - \alpha)Q(s_i, a_i) + \alpha(r_i + \gamma * \max_b Q(s_{i+1}, b)) \quad (2.1)$$

with the received reward as r_i , the learning rate as α , the discount factor as γ , the next possible action as b and the next state as s_{i+1} [33, p. 3][10, p. 4].

Since the state space can be quite large in specific reinforcement learning tasks, as for example Atari2600 games, a function approximator θ can be used to estimate the Q Value, $Q(s, a, \theta) \approx Q^*(s, a)$, with Q^* as the optimal Q Value. One kind of non-linear function approximator θ could be a neural network. One kind of non-linear function approximator θ could be a neural network [24, p. 3]. Setting the neural network to learn over an experience replay memory has shown to give some benefits compared to direct interaction with the environment, such as data efficiency and breaking possible correlations when only learning with consecutive inputs [24, p. 4-p. 5]. After applying all the changes to Q-Learning, the resulted algorithm is called *deep Q-Learning* [24, p. 4] with the neural network as *Deep Q-Network* (DQN) [24, p. 6].

2.1.2. REINFORCE

A way to solve a reinforcement learning problem is to train the decision-making policy directly. These algorithms fall under the Model-Free reinforcement learning algorithms. REINFORCE, which stands for "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility", describes the class of algorithms which tries to learn the policy directly by performing gradient descent on the expected reward and backpropagation [20, p. 25][34, p. 5].

2.1.3. Deep GA

Instead of training deep neural networks via backpropagation, Deep GA trains the neural network by evolving its weights with a simple, gradient-free population-based genetic algorithm. By developing this algorithm, the developer tested its performance on deep reinforcement learning benchmarks, including Atari 2600 and Humanoid Locomotion in the MuJoCo Simulator. Deep GA performs competitive even with its simplicity compared to DQN, A3C, and evolution strategies(ES). [28, p. 1]

A typical genetic algorithm creates in each generation population \mathcal{P} with \mathcal{N} individuals [28, p. 2]. In the algorithm presented here, an individual is a neural network parameter vector θ . At every generation, the fitness score of each θ_i is the final reward of the given reinforcement learning task running k times by the neural network with θ_i . After calculating the fitness scores of all θ , *truncation selection* is performed: The top T individuals θ_T are selected to be the parent of the next generation. To create an individual θ_i for the next generation, a single parent from the given set of parents θ_T is selected uniformly at random and mutated by applying additive Gaussian noise to its parameter. This process is repeated $\mathcal{N} - 1$ times. The \mathcal{N}^{th} individual of each generation is the best parent from the set of parents θ_T without any modification. This principle of adding the best of the best is called *elitism*. To find the true elite in the given set of parents θ_T , the fitness score of each $\theta_j, j \subseteq T$ is again evaluated e times with $e > k$ to be sure to select the correct individual as the elite. Finally, the algorithm will evaluate the new generation. This procedure is repeated G times while every iteration represents a single generation getting evaluated and supplying the next generation's parents [28, p. 3].

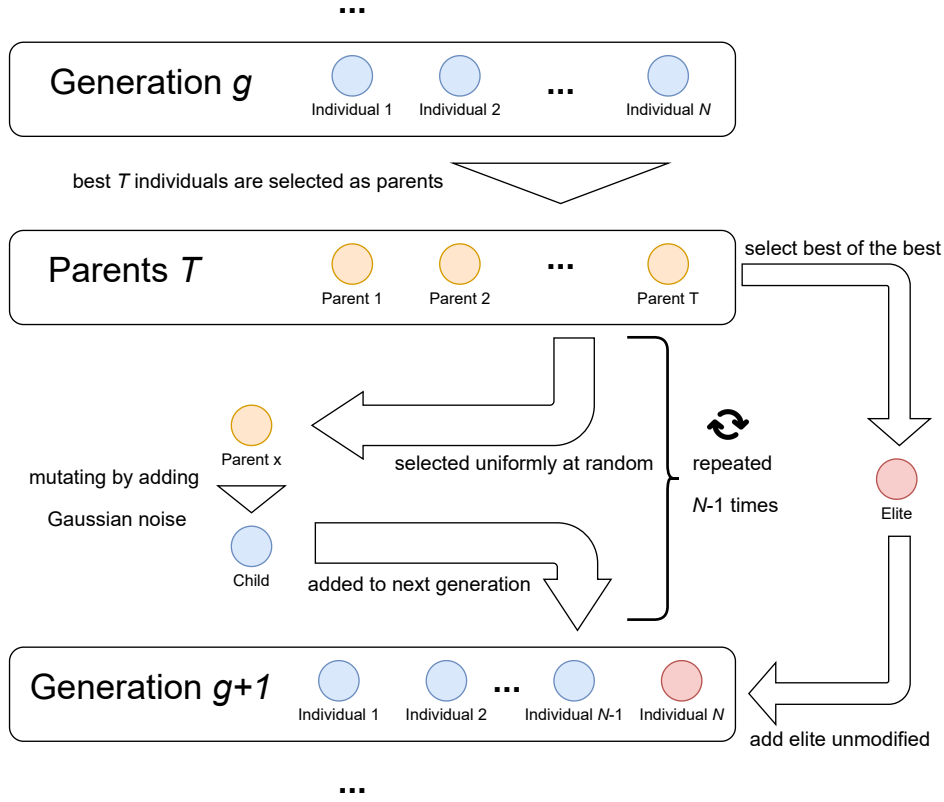


Figure 2.2.: One iteration of DeepGA is visually displayed. From a generation, the best T parents will be selected and mutated to generate the next generation. The algorithm adds the best of the parents to the next generation as the elite.

2.1.4. DreamerV2

DreamerV2 is a model-based reinforcement learning algorithm, which trains an agent from predictions in the compact latent space from its world model [15, p. 1]. DreamerV2 is an update of the original DreamerV1 algorithm from Hafner [14]. Like other model-based RL algorithms, DreamerV2 consists of three components: A world model trained by past experiences, an actor and critic trained by imagined sequences of compact model states, and using the actor to generate new experiences by letting it interact with the environment [15, p. 2].

The world model summarizes the experience in sets of images from the environment into a compact latent space. The world model can predict the next state in the compact latent space to replace the actual environment for training an agent to solve the given environment. The main advantage of using the compact latent space is the ability to predict multiple states in parallel in a single batch without interacting with many environments to generate the images for the batch. The model inputs the last state and an action and predicts the next state and the corresponding reward [15, p. 2-3]. The agent of DreamerV2 trains itself by using an actor and a critic. The actor gets the predicted current state from the world model and chooses an appropriate action for the current state. The current state and the selected action predict the next state and the corresponding reward with the world model. The critic gets the future predicted reward and lets it influence the currently predicted rewards in the planning horizon of the actor to benefit its learning progress [15, p. 4]. Finally, like mentioned above, the trained actor is used to generate new experiences by interacting with the actual environment. Since the algorithm trains the actor while generating the experience with the actor, the kind of experience generated may differ from using an untrained actor. Therefore the separated trained world model gets fed with new types of experience. At the same time, the actor's performance rises, which benefits the predictions made by the world model for training actor and critic.

The modifications made in DreamerV2 to achieve better performance compared to DreamerV1 are listed here [15, p. 18]. They are mainly influencing the kind of state, model design, and loss calculation of the algorithm. DreamerV2 achieves human-level performance across Atari2600 games (capped by 200Mio steps). Since it uses a compact latent space, the training can run it on a single GPU. Compared to other model-free RL algorithms for a single GPU, it also outperforms Rainbow, IQN, and DQN [15, p. 1]. MuZero[26], another model-based RL algorithm, shows the potential of planning with a given world model but with the cost of using more resources.

2.2. Technical Background

2.2.1. OpenAI Gym

OpenAI Gym is a toolkit for reinforcement learning research that includes a collection of benchmarks problems and a website¹ to compare results [6, p. 1]. Since there exists a variety of different benchmarks for reinforcement learning, for example, the Arcade

¹<http://gym.openai.com/> (visited 06.11.2021)

Learning Environment (ALE) for Atari2600 games or RLLab for continuous control problems, the toolkit of OpenAI Gym bundles most of the common reinforcement learning benchmarks into an easy to use software package [6, p. 1].

2.2.2. The Atari Annotated RAM Interface (AtariARI)

Atari2600 games used in the Atari Learning Environment (ALE) [4, p. 1] are using the RAM state (128 bytes per timestep) to store state information from the currently running game. For example, the RAM state stores the sprites' location or the game's current score. The Atari Annotated RAM Interface (AtariARI) describes the mapping of possible valuable variables from the RAM state for 22 games [1, p. 4]. Interested parties can retrieve the list of the variables for each game from Ankesh Anand [1, p. 5]. Table 2.1 shows exemplary the variables extracted by AtariARI for the selected games, used with the prototypes mentioned in this thesis.

Game	Agent localization	Other localization	Score live display	Overall
Pong	player-x, player-y	enemy-x, enemy-y, ball-x, ball-y	player-score, enemy-score	8
Tennis	player-x, player-y	enemy-x, enemy-y, ball-x, ball-y	player-score, enemy-score	8
Boxing	player-x, player-y	enemy-x, enemy-y	player-score, enemy-score	6

Table 2.1.: The mapped features from AtariARI for Tennis, Pong, and Boxing.

3. Related Work

As mentioned in the introduction, related work about getting explainability into reinforcement algorithms are either one of the following two categories: Transparent algorithms or Post-hoc explainability [18, p. 6]. This section describes specific approaches used by the prototypes described later in the thesis and categorizes them in one of the two shown categories.

3.1. Post-hoc explainability based related work

3.1.1. Integrated Gradients

Attributing the output of a neural network to its input features helps to understand the neural network's decision to create the output by showing the importance of each given input feature. Unfortunately, many methods to attribute the output to its input features lack either of the two following axioms: *Sensitivity* or *Implementation Invariance* [29, p. 1]. The first axiom, *Sensitivity*, is satisfied by the attribution method if, for every input and baseline that differ in one feature but have different predictions, the differing feature should be given a non-zero attribution [29, p. 2]. The attribution method satisfies the second axiom *Implementation Invariance* when the attribution of two networks with different architecture, but still the same output, are identical [29, p. 2]. Integrated Gradients describe an attribution method that satisfies both presented axioms [29, p. 3]. Integrated gradients are defined as follows:

"Integrated gradients are defined as the path integral of the gradients along the straight-line path from the baseline x' to the input x ." [29, p. 3]

Possible application areas of using integrated gradients are using it for detecting the critical pixels of the input image with an object recognition network, for diabetic retinopathy prediction, question classification, machine translations, or chemistry models [29, p. 6-8].

3.1.2. Saliency maps for interpreting the agent's behavior

Saliency maps, like Grad-CAM (Gradient-weighted Class Activation Map), can be used for trying to explain the behavior of the agents trained by reinforcement learning algorithms [19, p. 1][12, p. 1]. Grad-CAM is a visualization technique, which considers the backward and forward pass. By doing this, it can show better visual results regarding the local points of interest on the input image, which are influencing the predicted class [19, p. 1]. In Joo's work [19], Grad-CAM is used with E-A3C, an extended version of A3C, a variant of the model-free policy-gradient reinforcement learning algorithm. It is used by taking a trained policy from E-A3C and inserting the needed steps to generate the image directly after the last CNN layer of the policy network [19, p. 1-2]. Consequently, this method falls into the category of Post-hoc explainability-based approach. Greydanus [12] shows examples, how saliency maps like Grad-CAM can be interpreted and used to either try to understand the resulting policy from the training or to debug not converging agents [12, p. 4-7].

However, Atrey et al. [2] showed that hypotheses made alone with saliency maps are often falsifiable. These maps should only use them to explore the behavior of an agent instead of creating these falsifiable hypotheses. Human observers tend to bring in the possible missing of challenging associations between the observed and its possible reasoning, which typically is evident for the observer but not guaranteed since the observer has pre-knowledge and is often biased. The author suggests using additional tools to create hypotheses that describe and explains the agent's behavior [2, p. 9].

3.2. Transparent algorithm design based related work

3.2.1. SPACE

Spatially Parallel Attention and Component Extraction (SPACE) is a generative latent variable model for *unsupervised object-oriented scene representation learning*. It combines

approaches of spatial-attention and scene-mixture to provide factorized object representations for foreground objects while also decomposing background segments of complex morphology [23, p. 1]. For SPACE a scene x is decomposed into two independent latents, which are z^{fg} as the foreground and z^{bg} as the background. While the foreground z^{fg} is furthermore decomposed into separate foreground objects $z^{fg} = \{z_i^{fg}\}$, the background z^{bg} is also decomposed into a sequence of background segments $z^{bg} = z_{1:K}^{bg}$ [23, p. 2].

The foreground is divided into $H \times W$ cells while each cell models the nearest object in the scene. Furthermore each cell i is associated with a set of latents $(z_i^{pres}, z_i^{where}, z_i^{depth}, z_i^{what})$, which represents whether the cell models an object (z_i^{pres}), the object position (z_i^{where}), the depth of the object (z_i^{depth}) and the object appearance with its mask (z_i^{what}) [23, p. 3]. The background z^{bg} is modeled with background segments z_k^{bg} which are separated by their mixing probabilities and their RGB-Distribution [23, p. 3-4]. Figure 3.1 shows the collaboration of the foreground and background modules and how the separated component of each module is used to reconstruct the origin image [23, p. 9-10].

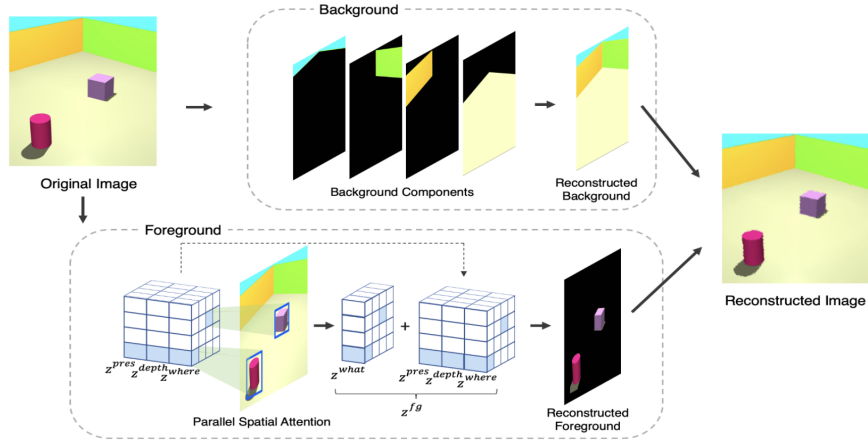


Figure 3.1.: The principle of SPACE and its foreground and background module [23, p. 3].

Due to its character of being an unsupervised method to decompose given images, it returns more understandable results than examining the result of the CNN layers in typical black-box characteristic neural networks used in various reinforcement learning algorithms. The results and its usage from SPACE could be compared to the results from applying saliency maps to the given CNN-Layers. Therefore, SPACE could be a usable solution regarding hierarchical reinforcement learning for analyzing and extracting relevant information from the provided images.

3.2.2. Explainable Hierarchical Reinforcement Learning for Robotic Manipulation

Beyret et al. [5] presents an approach of applying hierarchical reinforcement learning to solve RL tasks in the field of robotic systems [5, p. 1]. The presented hierarchical reinforcement learning architecture, called Dot-to-Dot, consists of a high-level agent, creating subgoals for low-level agents. Solving all subgoals together should solve the whole main goal of the given task. Therefore, the main goal of the high-level agent is to design these subgoals to get the main goal solved with them [5, p. 2-3]. The explanatory part of this approach comes with the idea to have the high-level agent trying to explain the agent's behavior through the created subgoals. Post-hoc explainability approaches could be applied to the high-level agent to get an understanding of the created subgoals and their conjunction concerning solving the main goal [5, p. 5]. It should be mentioned, the grade of explainability the high-level agent can provide could be exaggerated from the observer/applier, similar to Atrey et al. [2] description about saliency maps since Beyret et al. [5] only shows one example of retrieving some explanation about the behavior of the high-level agent without briefly discussing the possibilities their approach could bring regarding explaining and interpreting the agent's behavior [5, p. 5].

4. Designing an explainable reinforcement learning pipeline

4.1. Extracted primary goals from previous work

Looking at the already existing work presented in chapter 3, this chapter formulates the main goals for the concept it is introducing:

1. **Subgoals of minimal size:** Beyret et al. [5] tried to show, splitting the whole task into smaller tasks and trying to get the reasoning behind the smaller tasks separately should be easier than having one big solver for the entire task. Therefore, one goal should be spitting the given task into subtasks. These subtasks should have their smallest size possible to remove complexity for solving this subtask and from the solver.
2. **Understandable intermediate results:** Saliency maps (3.1.2), as an example of a post-hoc explainability approach, are trying to convert the interim results of a solver for the whole RL task into a from a human perspective (hopefully) interpretable state. To carry on with the goal of these approaches, having the solver of the subtasks designed to already output interpretable intermediate results would make it easier to extract the reasoning of the final decision after using all subtasks.
3. **Interchangeable solver of subgoals:** SPACE (3.2.1) as an example of a solver for one subtask hints the potential of designing the subtasks in a way that the solvers for the subtasks are exchangeable at any point with a newer or better solver for this specific subtask.

4.2. Overview of the hierarchical task splitting

As hinted in the previous section, splitting the RL task into subtasks and having separate modules to solve each task for the basis of the presented pipeline helps to bring explainability into the solution for solving the RL task. Therefore, showing how to design the subtasks should be the first step. The main task to solve is bounded to a handful of selected Atari2600 games. Typical for this kind of task is having only sets of images of the last frames from the game and deciding which action to take to solve the game. Consequently, the main task to solve the game can be split into subtasks, as shown in figure 4.1. In the following, the subtasks are described as separate tasks.

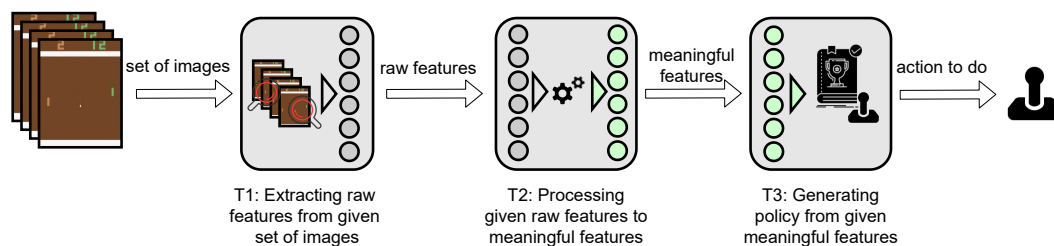


Figure 4.1.: Split subtasks for the RL task to solve an Atari2600 game.

- T1* The first task analyzes the given set of images and extracts raw features from it. The given images typically contain sprites of the game objects and the current score of the current game (depending there is any score in the selected game). The solver for this subtask must have some image detection and classification method to locate the position of the sprites, which typically represents objects in the game. Therefore the extracted raw features are axis-oriented coordinates and the plain score values of the current game state.
- T2* The second task involves receiving the extracted raw features and processing them to possible meaningful features. Since raw features typically consist of just position coordinates and score values, they can usually be processed to better understand what information about the game's current state can be extracted from the raw features. For example, just having the coordinates of two game objects for the last n frames does not tell much about the situation. Having the distance calculated between the two objects with the given coordinates or the speed and direction the objects are moving, which this solver can also calculate with the given raw features, helps much more to understand what is currently happening in the given state

compared to just having the unprocessed coordinates of the object. The problem of this task is to define what kind of meaningful feature has to be processed. On the one side, it depends on the given kind of raw features which are available. For calculating specific types of meaningful features, this solver must provide specific raw features. On the other hand, it also depends on the task to solve. Not every kind of meaningful feature calculated from the given raw features must be helpful for the task. Therefore it is not trivial to decide which meaningful features should be calculated with the given raw features.

T3 The third and final task involves generating and applying a policy with the given meaningful features to solve the RL task. Having already processed features into meaningful features should theoretically help to apply a policy without much processing of the given features. In addition, after having trained a policy, it should be possible to see which of the meaningful features are necessary and used to apply the policy and which feature is unnecessary. For example, having two objects where the task is to move one object as close as possible to the other object while the second object is randomly moving, the policy can be set as a straightforward rule: Minimize the distance between the object by moving the controllable object in the direction of the other object. Applying this rule as the policy should be much easier when already having the distance of the objects as meaningful features instead of first processing the raw features to the distance. In addition, applying the correct explainability method can show, the policy uses the distance between the two objects for solving the task instead of just assuming the policy is processing the raw features to the distance of the two objects. The policy may calculate some other meaningful feature when given only the raw features to generate its policy. This other self-processed meaningful feature could be way less interpretable than the processed meaningful feature from the previous task. The following chapter will discuss the degree of the truthiness of both here presented theses.

The three tasks *T1*, *T2* and *T3* consist of different kinds of tasks. The solver of the first task *T1* has to use image detection and classification techniques, while the solver of the second task *T2* has to use feature engineering techniques. Classic reinforcement learning algorithms can solve the third task *T3* with the benefit of simplifying the model because of the already processed features.

4.3. Approaches for $T1$

Typically when the task is to detect objects on given images, the common practice is to use convolutional layers inside the first layers of the solving deep neural network [21, p. 1]. The reinforcement learning task to play an Atari2600 game is no exception. One of the earliest approaches by using a deep neural network to solve the task has convolutional layers in its first layers of the network architecture [24, p. 5]. Since this task consists of just detecting the objects inside the image, it would be logical to use some convolutional layers, like taking the first part of the deep neural network, which solves the game in [24]. One problem with this approach is getting the correct feedback on how well the convolutional layers detect anything. Having the layers as a part of the whole network, which gets trained by playing the game, the agent can directly check whether the information from the convolutional layers forwarded to the subsequent layers is good enough or has to optimize them. Having the task to detect objects on the given image as a separate task, its solver has not had the option to get direct feedback from the solver of the subsequent tasks about itself without breaking the concept of having a separate, trainable solver for each task.

4.3.1. Supervised vs. unsupervised learning

Taking the work of Krizhevsky et al. [21] as an example, one way to individually solve this task is by transforming the task into supervised learning. Like training the network to detect objects in the ImageNet Dataset, the solver of this task gets unlabeled and labeled images to train itself for detecting the objects like the labeled images. An advantage of this solution would be the implementation difficulty since meanwhile lot of research has already been done in this section [22, p. 1]. Another advantage would be the result of the solver: Exact locations of each classified object on the given image. A disadvantage of this approach would be the effort to prepare the data for training. The training data has to be collected and be labeled. The quality of the trained result depends on the labeling quality. Furthermore, it takes flexibility by already deciding which objects have to be detected on the given set of training images since all detectable classes must be labeled before training.

To overcome the disadvantages of the first presented approach would be to apply an unsupervised learning approach. For example, using SPACE (3.2.1) or any similar unsupervised model [7] which can detect objects without using labeled data removes the effort

to take for creating the training data. It also removes the constraints about the flexibility for the detectable objects since it now depends on the model, its hyperparameters and just the unlabeled images as the training data set. It could be argued, this is actually a disadvantage compared to have the control of the classes to detect regarding the loss of control of it. Regarding the aspect of having a generalized approach, giving the responsibility of choosing the detectable classes to the model by itself should be more valuable compared to manually decide of the detectable classes. A possible real disadvantage for the unsupervised approach is the increased model complexity of the current available unsupervised approaches [7][23] compared to typical supervised approaches [16]. Consequently, more computational power is needed to train a model from an unsupervised approach than a supervised approach.

Instead of analyzing the given input image, it is possible in particular RL tasks, like playing an Atari2600 game, to retrieve the environment informations by accessing the RAM with f.e. AtariARI (2.2.2). This is more accurate than extracting the information from the given images since the game writes its exact game state variables inside the ram. In addition, it is also less computational heavy. Since it is only possible when the state variables of the current RL task are accessible, and this is typically not often the case, this approach represents the optimal solution for the task, which can nearly never be applied.

4.3.2. Applying Post-Hoc explainability methods

This section could discuss the quality of the result of the different approaches. Since having the best performance is not the main focus of the presented concept, let us assume that all presented approaches deliver results with the needed quality level of similarity to having no dependency to the chosen approach for the availability of the methods presented in this section. As mentioned earlier, it is easier to understand the respective solver of smaller subtasks instead of trying to understand the single solver for the whole task. Therefore applying post-hoc explainability methods to these subtasks should help to raise the interpretability and explainability for the solver of this subtask and its decision and results. This also includes creating various types of saliency maps [12] including GradCAMs [19] to the trained model since it is one of the most popular visualisation technique to get interpretability into models with image based inputs [2, p. 1]. Atrey et al. [2, p. 9.10] has shown saliency maps cannot be fully trusted to link the results of the map and the behavior of an RL agent. Since this is only the first part of a collection of possible interpretable and explainable arguments based on all the subtasks, this could build a valid argumentation base for the following arguments in the collection.

4.4. Approaches for $T2$

Having the raw features as the input for this subtask invites us to process them into meaningful features, generating a policy for the whole RL task. The methods of processing the raw features depend strongly on the type of incoming raw features. For example, the previous subtasks extracted the written scores for both players of a running Atari game. The only way to process them is to calculate the difference between both scores. Since the position of the scores on the screen is usually irrelevant, it is not necessary to calculate its possible velocity or its moving trajectory.

4.4.1. Meaningful features for position coordinates

However, having the coordinates of the different game objects in the Atari game, like the player or its enemies, applying more processing methods returns meaningful features, which helps generate the policy in the next subtask. The most apparent meaningful feature to calculate is the distance between two objects. Choosing the kind of distance to calculate, whether it is the manhattan distance, euclidean distance, or separate distances on each axis (x and y-axis, since the coordinates are from a 2d-image), is not trivial. It depends on the given task and the type of objects from the source of the given coordinates. Another obvious meaningful feature would be the velocity of each game object. When the coordinates of the objects feed the solver of this task, and the steps between the given frames are constant, the processing method can easily calculate the exact velocity of the objects by temporarily saving the coordinates of the current frame and using them in the next frame. A bit more complicated meaningful feature would be having the moving trajectory of each object. With the requirement that the objects are moving, calculating and having its moving trajectory could help to see whether an object is going into a direction or space relevant to the task. A simple example of this would be a game where the enemy shoots missiles at the player. Combining with the missile's velocity, seeing the moving trajectory helps the player to detect a possible collision.

The current examples show some of the possibilities for meaningful features. Like mentioned earlier, they are heavily dependent on the given raw features from $T1$ and the goal of the whole task. The processing method could also apply more complicated feature engineering approaches [11][31] instead of simple processing steps. Developer of the processing method should consider two other aspects as well for choosing the feature engineering steps for this task:

Understandable features Having an understandable and interpretable output from $T2$ should be targeted. Otherwise, it is less likely to understand which type of feature the policy from $T3$ is more using. It is easier to argue why the policy has been decided in one way when its input features are more understandable.

Number of features The more feature engineering approaches are applied to the given raw features, the more meaningful features are calculated and given for the policy for $T3$. At some point, it should be balanced whether enough (types of) meaningful features are already available or other types of meaningful features still need to be calculated. It should be considered that the increasing amount of given meaningful features could lead to increasing difficulty in understanding the reasoning behind the policy using all of these meaningful features.

An obvious approach to tackling too many features would be applying pruning techniques to the given features. By this time, it is common to use pruning techniques for optimizing a model/policy or its inference [3, p. 5]. Therefore, applying one or multiple pruning techniques should help to differentiate these meaningful features, which are more important for the policy of $T3$, from the features, which are either correlated to other features or just unimportant. The pruning technique can either take the given features and select from this the important features, or it could also take input from the policy from $T3$, for example, its integrated gradient value for each feature [29], to help the pruning technique for its selection process.

4.5. Approaches for $T3$

Since the given features are already processed (and optional pruned), the solver of the third task $T3$ can directly use them to generate a policy to solve the whole task finally. One way to generate the policy would be applying typical reinforcement learning algorithms (see section 2.1). A benefit of using a model-free reinforcement learning algorithm is, the trained model does not have to learn anything about extracting and processing features since it is already done in the previous tasks. Furthermore, the parts of the network architecture about feature processing, which are typically the convolutional layers (CNN-layers) at the beginning of the network [24, p. 5-6], can be removed. This reduces the model's complexity to train and has the side effect of reducing the needed computational resources to train the model. To continue the idea of model optimization, the fully connected layers, which generally are set behind the CNN-layers [24, p. 5-6], can also be reduced, as they usually process the internal extracted features from the given images

by the CNN-layers into processed internal features for the final policy decision. In the best case, the model has no hidden layer and is, therefore, a linear model, which directly connects the meaningful features to the specific actions the policy has to decide. Thus, why the policy has chosen a specific action with the given features can be directly seen from the weights of the connections inside the linear model. However, even when the model has one hidden layer, it should still be easier to extract its reasoning of a decision than having multiple fully connected layers or even CNN-layers in the model. The degree of model reduction, which still results in a task-solving policy and its performance, will be shown in the next chapter (5).

4.5.1. Training a model based on given features

Instead of using a model-free reinforcement learning algorithm, a model based reinforcement learning algorithm can also be used. Instead of training a world model based on the given images, the basis of the model could be provided by either the raw features already extracted in $T1$ or the meaningful features from $T2$. Similar to reducing the model for the model-free reinforcement learning algorithms, the model architecture for training a world model can also be reduced due to having linear inputs instead of images. Consequently, it should take way fewer resources to train the world model (and the whole algorithm) compared to existing model-based reinforcement learning algorithms, which all train with given sets of images [15, p. 10]. A world model with the possible efficiency advantage described here could also help to develop new model-based approaches by using the world model more heavily than the existing ones.

4.6. Comparing the presented pipeline with the major goals

This section compares the presented pipeline with its subgoals and its intermediate results to the major goals to evaluate whether they are achieved.

4.6.1. Major goal 1: Subgoals of minimal size

To evaluate the first major goal of having subgoals of minimal sizes in the subtasks, the available intermediate results of the presented pipeline must be compared to its previous result. Looking at the first intermediate result in position coordinates on the input images

extracted on them, the goal is achieved because there is no useful intermediate goal inside the goal of extracting raw features in the form of position coordinates from an image. The following intermediate result in the form of meaningful features processed from raw features also has no useful subgoal inside the goal of processing raw features to meaningful features. The solver for T_3 can directly derive the last result as a policy from the given meaningful features, which shows the same property as the previous goals of having no useful subgoal inside generating a policy from the raw features. Summarized, all intermediate results of the given tasks show the achievement of the first major goal of having subgoals of minimal size for the split tasks.

4.6.2. Major goal 2: Understandable intermediate results

The next major goal of understanding intermediate results can be easily evaluated by examining the intermediate results and their type. The first two intermediate results, the raw features in the form of position coordinates on the given images and the meaningful features processed from the raw features, show by themselves their understandability as they were actively chosen to be a type of understandable features. However, the understandability of the policy generated from the meaningful features depends heavily on the chosen complexity of the policy and its generation. When this complexity is kept as simple as possible, the major goal of having understandable intermediate results is achieved.

4.6.3. Major goal 3: Interchangeable solver

This section can easily show the achievement of the third major goal of having interchangeable solver for each task. Every subtask only depends on the result of the previous task, not on how the previous task is solved. Therefore, changing the solver of a subtask does not influence the result of the next subtask when the results of the changed solver are the same as the replaced solver.

5. Experiments

5.1. Experiment setup

The experiments presented in this chapter try to show possibilities, advantages, and drawbacks for implementing the presented pipeline in the previous chapter. This chapter describes the setup for the experiments and how the experiments differ from each other inside the sections. As mentioned in the last chapters, the task to solve will be bound to Atari2600 games. The sections in this chapter will likewise present the games used in the experiments in the sections. The implementation of all experiments is done with PyTorch¹. To evaluate the final score of the experiments and compare it with the human scores, a human normalized score ($hns(s)$) will be introduced for all experiments:

$$\frac{s - s_{min}}{s_{human} - s_{min}} = hns(s) \quad (5.1)$$

With s as the achieved score, s_{min} as the minimum achievable score in the game, and s_{human} as the human score. The human normalized score will show the relative difference between the human score and the achieved score. The human scores are extracted from Wang et al. [32, p. 12] for all experiments. A trained agent has solved the game when it has beaten the opponent. When no opponent is available, an agent has solved the game by achieving a minimum of 50% human normalized score.

¹<https://pytorch.org/> (visited 06.11.2021)

5.2. RAM Annotations as raw features for reinforcement learning algorithms

This experiment shows that the position coordinates of all relevant game objects contain enough information to solve the given RL task. The position coordinates represent the result from the first task $T1$. When the goal is shown, the subsequent tasks can have enough information to solve the task, despite only having the extracted raw features given instead of the whole image. The implementation will use AtariARI (2.2.2) to remove the lack of quality in the results of $T1$. AtariARI extracts the raw features directly from the RAM. Thus the extracted coordinates of all game objects will be accurate at every frame. The selected RL task will be Pong, an Atari2600 game, for two reasons: The first reason for using Pong is its relatively bad score for random actions. Without any learning, the agent has nearly the worst score, which is comparable to doing nothing. However, it is not the most challenging task to learn when comparing the scores of other reinforcement learning algorithms [17, p. 11]. To sum it up, the agent has to learn something to solve this task. In this case, the agent has to learn it with the given output from AtariARI. The second reason affects the next experiment, which is also using Pong. Therefore the next section will mention it.

Pong has three relevant game objects: the player's paddle, the enemy paddle, and the ball. Therefore, the given raw features from AtariARI are the x and y coordinates from the three relevant game objects. To simulate the frame stacking, which is typically done for solving Atari2600 games [24, p. 5], the raw features from the current frame and the previous frame are given per step. The agent to solve the game consists of a multi-layer perceptron (MLP) with one hidden layer. There is no need to have CNN-Layers since the given input is the raw features instead of images. The hidden layer represents the step of processing the raw features to meaningful features ($T2$). This experiment uses DeepGA (2.2) as the reinforcement learning algorithm due to its simplicity and computational efficiency for training agents to solve Atari2600 games. If DeepGA can solve Pong by using raw features, it can hint why some games have not worked by using the direct input for DeepGA in the experiments made by Such et al. [28]. Pong may be one of those games since it is not mentioned in the results [28, p. 5.6]. As a comparison, REINFORCE(2.1.2) is also applied with the same network architecture for the agent. REINFORCE is selected due to being one of the first policy-gradient-based approaches to train a policy directly. Therefore, it stands as the counterpart for DeepGA to train a policy with backpropagation since DeepGA does not use backpropagation. The comparison between DeepGA and REINFORCE is selected because both algorithms apply different ideas to train the policy but are one of the most

basic approaches from its idea. The appendix lists the exact training hyperparameters (A).

5.2.1. Experiment results of using raw features for Pong

Game	Random	DeepGA	REINFORCE
Pong	0.8%	118.0%	0.0%

Table 5.1.: The final experiment results in human normalized scores. For choosing the agent for DeepGA, the algorithm selected the elite agent from the last generation. DeepGA can solve Pong with given raw features. The elite agent performs better than an average human player. However, REINFORCE has not solved Pong and even performed worse compared to doing random actions.

The experiment results (Table 5.1) show that Pong can be solved with the raw features in position coordinates from the relevant game objects. Consequently, the position coordinates of the relevant game objects contain enough information to solve the task. Figure 5.1 even shows, the elite agent from DeepGA even achieves the maximum score. Nevertheless, REINFORCE has not solved Pong. Trying multiple seeds have always resulted in only scoring the worst score possible in Pong. Thus it can be said, the raw features may contain enough information to solve Pong, but not every reinforcement learning algorithm can solve it with the given information.

5.3. SPACE for $T1$

This experiment aims to evaluate whether a ready-to-use solver for $T1$ delivers enough information from the given images in the form of raw features to apply a reinforcement learning algorithm successfully. The solver selected for this experiment is SPACE (3.2.1). When successfully trained with a game, it delivers the position of the detected game objects in the form of a list. To evaluate the quality of the output from SPACE, DQN (2.1.1) is applied to it. This experiment will compare the result to the benchmark, which applies DQN to the whole RL task with the images as the input. The Duel-DQN architecture [32] is used due to its applicability for Atari2600 games [32, p. 8-9], having the possibility to directly create a benchmark to compare based on the given images and its simplicity compared to more advanced model-free approaches like Rainbow [17]. The exact training parameters and configurations are listed in the appendix (A). The Atari2600 game used

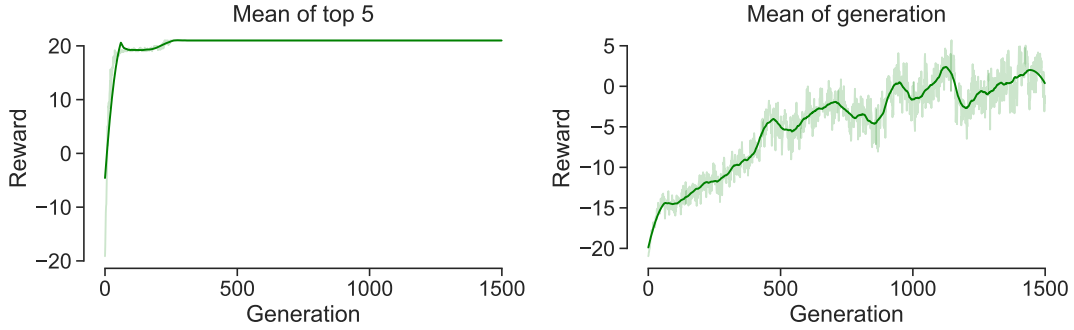


Figure 5.1.: The reward graph for using DeepGA with raw features as input. The left plot shows the mean reward score of the top 5 of each generation, while the right plot shows the mean reward score of the whole generation.

in this experiment is exclusively Pong since it was not trained for the source paper [23, p. 12]. It removes the argument of biased selecting the Atari2600 game and represents a new RL task to challenge for the solver.

To train SPACE for Pong, the exact parameters [23, p. 18] and preprocessing steps for the given data [23, p. 22] proposed for Atari2600 games were used. It directly converges, but only after minor additional, computer vision-based preprocessing steps, including setting the background to black, the game objects were detected as foreground objects. It may come from the color similarity between the game objects and the background since the already tested games for SPACE all had black background while having colored game objects. This experiment will ignore the mentioned problems for now to test the quality of the output SPACE delivers.

5.3.1. Evaluation of SPACE for Pong

As mentioned in 4.5 having already extracted information from the given images invites to use just a MLP instead of a network architecture with CNN-Layers. Thus the first experiments were done with a DuelDQN architecture, where the CNN-Layers were removed to keep the benefits of applying DuelDQN but still using a MLP. The input was the z_i^{where} masked by z_i^{pres} (now $z_{z^{pres}}^{where}$) to get only the position coordinates from the foreground objects. z_i^{what} was also attached to give the $z_{z^{pres}}^{where}$'s its description. When the $z_{z^{pres}}^{where}$ were

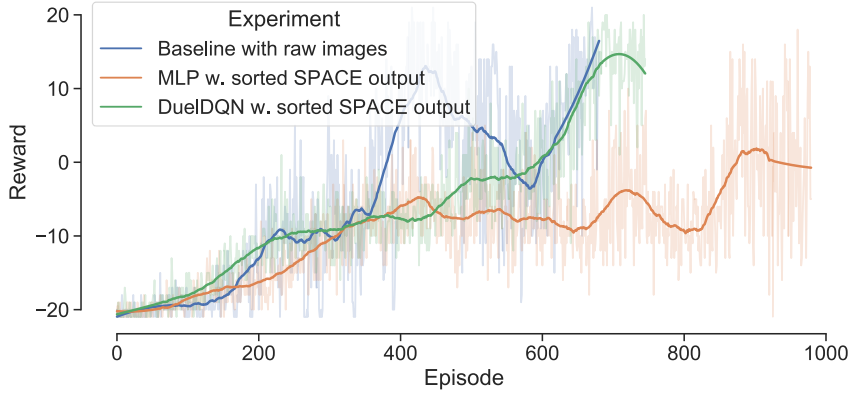


Figure 5.2.: The reward graph for using the unprocessed SPACE output is showing its lack of data quality. Replacing the CNN-Layers of the DuelDQN architecture (blue) with SPACE for preprocessing the images and then using an MLP (orange) resulted in an inferior performance. The architecture had to be the same as the original DuelDQN architecture and keep the SPACE output in the image-like shape to achieve similar performance (green) as the baseline (orange).

manually sorted as a processing step, the agent converged (see figure 5.2), otherwise the agent did not train at all, which might be attributed to z_i^{what} not helping enough to categorize the unsorted input. Nevertheless, the performance was worse compared to the baseline. Only using the original DuelDQN architecture with the CNN-Layers and providing the output from SPACE in an image-like structure as provided by SPACE with z_i^{pres} as the top layer resulted in similar performance compared to the baseline. This could be attributed to seeing the input provided by SPACE as the original image without the noise of the non-relevant objects on the images.

Regardless of the performance for the different approaches, the sum of adjustments needed shows the difficulty of using SPACE as the solver for $T1$. Figure 5.3 shows additional visual hints of the missing quality for the output from SPACE. Since the previous experiment has shown, the position coordinates of the relevant game objects are enough to train a MLP for Pong, AtariARI will be used as the solver for $T1$ for future experiments. Thereby it is guaranteed that the solver's missing quality will not influence the results of the future experiments for $T1$. In addition, it should be mentioned: SPACE can be a valuable

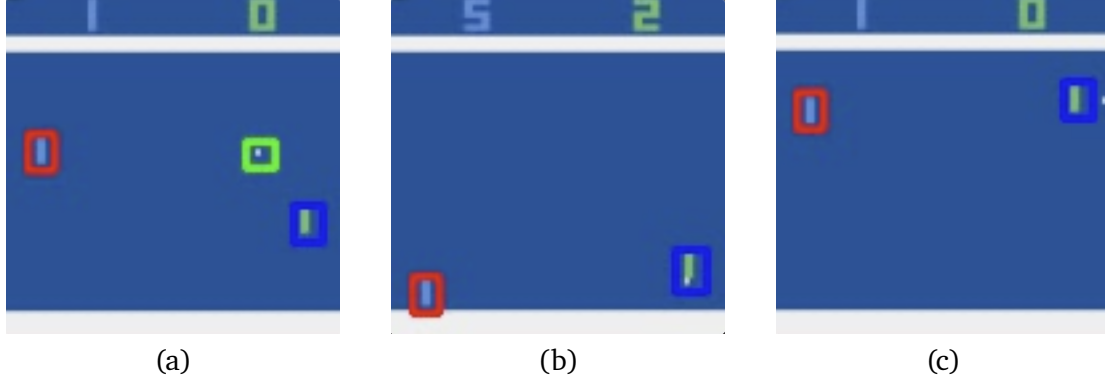


Figure 5.3.: Visual evaluation of the (missing) quality for the output from SPACE. The rectangles on the images show the given position coordinates received from SPACE. The color shows the assertion of the position coordinates to its game object type, done after receiving the output from SPACE. While (a) shows the accuracy of SPACE’s output, (b) and (c) show the problem for exceptional cases, where the ball is near another game object and SPACE cannot keep both game objects apart.

qualitative solver for $T1$. Some adjustments have to be made to achieve the quality of results needed for solving RL tasks. Whereas the effort required to make these adjustments can go beyond the scope of this thesis, future experiments will not examine it longer.

5.4. Handcrafted meaningful features for reinforcement learning algorithms

This experiment aims to evaluate having three separate solvers for the three given subtasks $T1$, $T2$, and $T3$. As mentioned in the previous experiment, this experiment will now use AtariARI for the first task. For calculating meaningful features out of the given raw features, it will manually select and calculate specific types of meaningful features and have the prototype for $T2$. As mentioned in the possible approaches for $T3$ (4.5), a policy should be directly trainable with given meaningful features. DeepGA and REINFORCE are training the policy. The architecture for the policy network is the same as in the first experiment (5.2) to only test the influence of processing the raw features to meaningful features. At

a later experiment, A subsequent experiment will minimize the model architecture by removing the hidden layer. Pong is still the game to solve.

The three selected categories of meaningful features are the velocity of game objects, their axis oriented position difference, and the position difference between the movement trajectory of one game object to the axis difference of the other game object. The velocity is selected to get information about the movement of a selected game object. For the same reason, the difference between the moving trajectory of one game object to the axis difference of the other game object is selected. This feature tells the distance between a possible collision between the moving object and its movement trajectory and the other object. Figure 5.4 shows the derivation of this feature type visually. Axis oriented position difference is selected because it tells the actual position difference between two game objects. Since some objects in Atari2600 games have limited moving possibilities, like the paddles in Pong, which only move around the y axis, the calculated distance differences are calculated separately per axis.

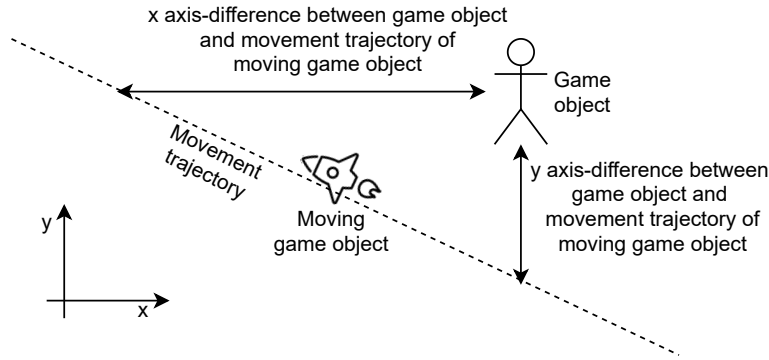


Figure 5.4.: A visual illustration of calculating the axis oriented difference between the movement trajectory of a moving game object and another game object. This feature helps to understand the movement the game object is doing.

To help the description of the experiments, notations for the three presented feature types will be introduced:

- $F_v(a)$ describing the velocity of game object a .
- $F_d^{x/y}(a, b)$ describing the axis oriented distance between game object a and b .
- $F_m^{x/y}(a, b)$ describing the axis oriented distance between the movement trajectory of the game object a and the selected axis of b .

As mentioned earlier, Pong consists of three relevant game objects: The player paddle (p), the enemy paddle (e), and the ball (b). The manually selected features for this experiment are listed here:

- $F_m^y(b, p)$ as the distance between the player's y-axis position and the possible intersection of the ball with it. The player's goal should be to minimize this distance as the player can only move around the y axis. To hit the ball, the player has to move its paddle into the way of the ball.
- $F_d^x(p, b)$ as the x-axis distance between the player and the ball. This feature helps to understand whether the ball is near the own paddle or the enemy paddle.
- 0 as a feature without information value.
- $F_d^x(b, e)$ as the correlated feature with $F_d^x(p, b)$.
- $F_d^y(b, e)$ as the y axis distance between the ball and the enemy. This feature should be normally as unimportant as feature 0 for the player.
- $F_v(b)$ as the ball's velocity, since it is the primary game object in Pong.

5.4.1. Experiment results of using handcrafted meaningful features for Pong

Game	Random	DeepGA	REINFORCE
Pong	0.8%	118.0%	77.4%

Table 5.2.: The final experiment results in human normalized scores. For choosing the agent for DeepGA, the algorithm selected the elite agent from the last generation. The results show the manual desired meaningful features based on the presented feature categories are enough to train a policy.

The experiment results (Table 5.2) show, DeepGA can solve Pong with the selected meaningful features. This comes with no surprise since DeepGA already solved Pong in the first experiment with just raw features, and now we should have features with more information in them. However, now REINFORCE also solves the game and achieves nearly average human performance. It can be said, the three independent solvers for the subtasks $T1$, $T2$, and $T3$ can solve the game. Compared to the first experiment (5.2), having a solver for $T2$ even helps $T3$ solve the whole task.

5.5. First generalized approach for meaningful features

This experiment aims to evaluate a first approach of generalizing the calculation of the meaningful features introduced in the previous experiment. Thereby, this section presents an algorithm to calculate all types of meaningful features for every game object. A more generalized approach for the task $T2$ represented by this algorithm simulates the change of the solver for $T2$ and its impact on the whole task. The network architecture is kept the same, except for the number of input nodes. These now depend on the number of features the algorithm outputs. Tennis and Boxing will be added aside from Pong to get solved. Compared to Pong, the player can move on both axes in both of the added games. From a human perspective, it seems harder to win against the enemy in Tennis as in Pong [32, p. 12]. Otherwise, the game principle is similar in both games. For this very reason, Tennis is added to represent a more challenging variation of Pong. Boxing is added to have a game as a comparison that is not based on hitting a ball. Otherwise, it could be said, the given meaningful features only work for ball based games.

Algorithm 1 Creating meaningful features

Require: f_r

```
 $f_m = []$ 
for  $i$  in  $\text{range}(0, \text{len}(f_r))$  do
     $f_m.\text{add}(F_v(f_r[i]))$ 
    for  $j$  in  $\text{range}(0, \text{len}(f_r))$  do
        if  $j > i$  then
             $f_m.\text{add}(F_d^x(f_r[i], f_r[j]), F_d^y(f_r[i], f_r[j]))$ 
        end if
    end for
    for  $j$  in  $\text{range}(0, \text{len}(f_r))$  do
        if  $i \neq j$  then
             $f_m.\text{add}(F_m^y(f_r[i], f_r[j]), F_m^x(f_r[i], f_r[j]))$ 
        end if
    end for
end for
return  $f_m$ 
```

Algorithm 1 shows the procedure to calculate and collect all presented types of meaningful features in a more generalized way. f_r is the list of game objects with their raw features. The algorithm fills f_m as the list of meaningful features. The algorithm loops over every

game object. At first, the algorithm calculates the velocity of the current game object and adds it to f_m . After that, it calculates all the axis-oriented distances to the current game object and all the other game objects with the condition, the distance is not already added in inverted form to f_m . Finally, it calculates the axis-oriented distance between the movement trajectory and the axes of all the other game objects.

5.5.1. Experiment results with generalized meaningful features

Game	Random	DeepGA	REINFORCE
Pong	0.8%	115.2%	51.5%
Tennis	1.3%	293.0%	87.3%
Boxing	89.3%	175.6%	138.7%

Table 5.3.: The final experiment results with the generalization of meaningful features in human normalized scores. For choosing the agent for DeepGA, the algorithm has selected the elite agent from the last generation.

The results of this experiment (Table 5.3) show that the agents can learn to play the games with the given meaningful features received from algorithm 1. The agents trained by DeepGA still perform better than the agents trained by REINFORCE. With the additional features, it should be mentioned that the agents converged faster than in the previous experiment. Nevertheless, the performance dropped a bit, which may be attributed to the same architecture used in both experiments, even when the feature count has been raised from six to 21 features. Since it is not the main goal to achieve the maximum performance, future experiments will not further evaluate the mentioned performance reduction and its reasoning.

5.6. Automatic feature selection with pruning

After showing that a generalized approach of creating meaningful features still works for the given tasks, optimizing the generated meaningful features is the next target. Figure 5.5 shows, not all features hold the same information value. In addition, some given meaningful features are also correlated. Therefore, feeding the agent with all meaningful features would mean giving the agent some theoretically unimportant information or even duplicate information. As mentioned in section 4.4, applying pruning techniques to the

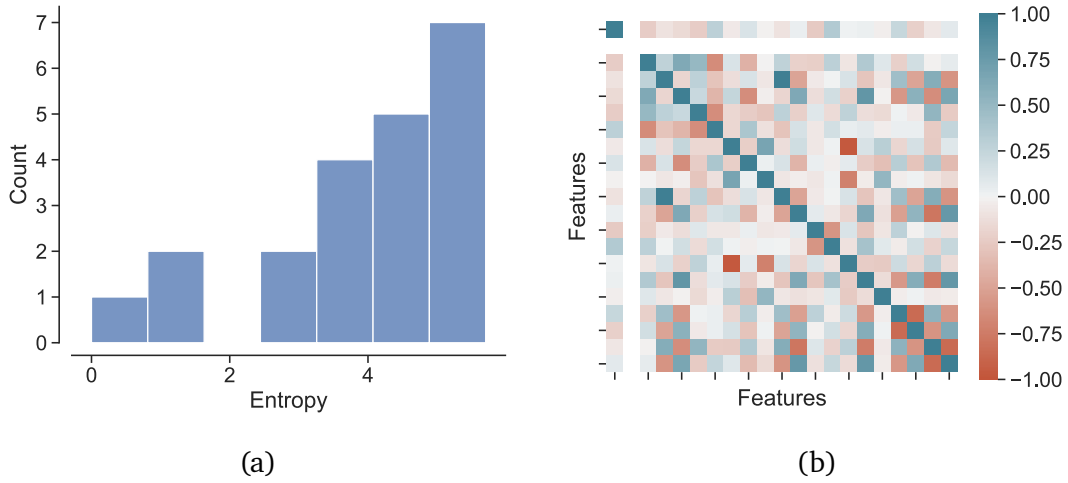


Figure 5.5.: Both plots show the different information values for the given meaningful features for Pong. While (a) shows a histogram of the entropy values for the meaningful features, (b) shows the correlation between all features as a correlation map. On the histogram, it can be seen, there exists a difference between meaningful features since their entropy values differ considerably. The correlation map shows that some meaningful features only deliver redundant pieces of information already available through other meaningful features. The data is for both plots is generated by playing multiple episodes of Pong with random actions and logging the meaningful features at every step.

given meaningful features could be a solution for automatically selecting the most suitable features for the agent trained in task $T3$. This experiment evaluates different pruning approaches and compares them to each other regarding applicability. This experiment will also evaluate how well each pruning method can explain why a specific feature is pruned. This experiment aims to show exemplary methods to select the best features in the given pipeline. Due to resource limitations, it will only evaluate the different pruning methods with Pong and REINFORCE as the selected algorithm.

5.6.1. Feature weight-based pruning

The first pruning method to evaluate is based on the average weights inside the neural network used for the training. It will calculate the average weight of all connections from an input feature to its next layer inside the neural network. If the absolute value of this average weight is below a specific threshold value, the pruning procedure will deactivate the input feature and all of its weights. This experiment applies the pruning procedure every 10.000 episodes.

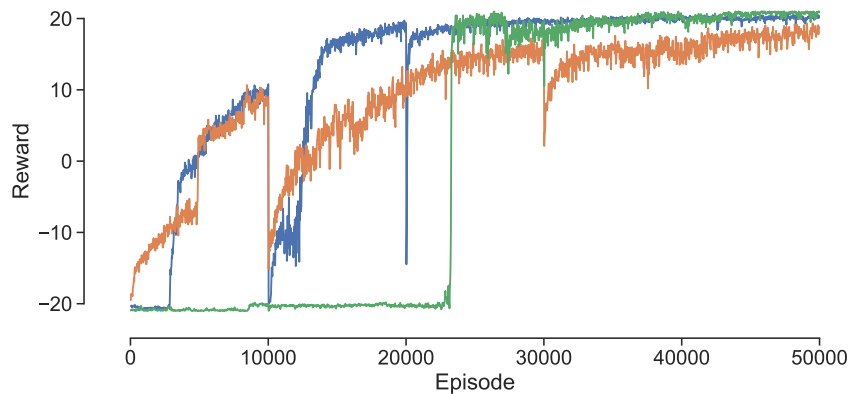


Figure 5.6.: Raw reward graphs for applying threshold pruning for all three runs with different seeds. No smoothing is applied here. The sudden performance drop and its reorientation of each training at every 10k steps can be seen. No reorientation phase can be seen at some point, which could hint that all remaining features are valuable enough regarding the given hyperparameters for pruning. The green run hints, pruning and its following reorientation can help to break a local optimum.

Running multiple seeds with the described pruning technique has shown, it will not always prune the same features. However, one run is pruning some features correlated to the features pruned by another run. On average, 25.4% of the input feature are pruned. This value should be viewed with caution since it depends on all of the given training hyperparameters. Figure 5.6 is also showing the dynamics of pruning while training. When the agent has already learned something, pruning features reduce the current performance drastically. However, the agent can typically reorient itself and retrain to its performance before the pruning step, or better. It is also showing, pruning can help an

agent stuck at a local optimum to reorient itself for breaking the local optima, e.g. the green run in figure 5.6. Evaluating the trained models, the weights of the pruned features are still zero, even after the last pruning step was 9.999 episodes away. The reason why the specific features are pruned is self-explanatory with the weights of the trained models.

5.6.2. Integrated gradient-based pruning

The pruning based on the integrated gradient (3.1.1) value applied in this experiment is similar to the pruning method based on the weight threshold value described in the last section. The only difference is, instead of calculating the average weight of the connection from a feature input to the next layer, the integrated gradient value of each feature is calculated. If the absolute value of the features integrated gradient value is below a specific threshold, the feature gets pruned and its weights are set to zero. Similar to the previous pruning method, integrated gradient-based pruning is also applied every 10.000 episodes. The results of using this pruning method are similar to the results of applying the last method pruning:

- Most of the time at the pruning episodes (after every 10.000 steps), especially after the first few pruning episodes, the performance drops and the agent has to reorient itself. The reorientation happens faster than the agent has taken to train itself to the performance level before the pruning episode.
- Similar to the previous pruning method, runs on different seeds tend to prune different features. However, some of these pruned features are correlated to pruned features from other runs. Accordingly, there exists a pool of correlated features that the pruning method will always prune.

5.6.3. Correlation-based pruning prior training

The pruning method evaluated now is based on the correlation values between all given meaningful features. Inspired by the correlation map in figure 5.5, the second feature of each feature pair with an absolute correlation value over a specific threshold will be pruned. This results in only having features whose absolute correlation values to each other are below the specified threshold. This pruning method is applied before the actual training of the model by using random actions and collecting all feature values. The selection of the features to prune does not directly depend on the model to train and its understanding of the features. Nevertheless, it should be mentioned, the correlation

values between features can change when using a trained model for collecting all feature values per episode. Since the actual training does not differ methodically from training without pruning, there is no notable difference in applying this pruning method, except the trained model uses fewer features for similar results. To measure the possible efficiency gain, the following section introduces a new metric.

5.6.4. Comparing the presented pruning methods

As the primary target of this experiment is not to maximize the performance. However, it should still be shown why pruning also makes sense regarding the efficiency of training a model. To value a model using fewer features while achieving similar performance as a model using more features, the following metric represents the update to the equation of the human normalized score:

$$\frac{s \cdot f_m}{f_u} = s_{fn} \quad (5.2)$$

$$\frac{s_{fn} - s_{min}}{s_{human} - s_{min}} = hns(s) \quad (5.3)$$

With s as the achieved score, f_m as the maximum number of available features, and f_u as the number of features used after the pruning process. All of the other variables stay the same as introduced with the human normalized equation. An advantage of this equation is that the non-pruned models' score does not change, while the pruned models get their benefit for using fewer features.

Even when the introduced metric seems not perfect and The experiments did not optimize the pruning methods to achieve the best performance, the results in table 5.4 indicate the potential of pruning the given input features. An obvious approach to optimize the performance would be finding the correct hyperparameter regarding the pruning method, e.g. the pruning interval or threshold value. Multiple pruning methods could also be combined, e.g. the correlation-based pruning together with the integrated gradient-based pruning.

Moving on to the explanatory part, why each pruning method has pruned a specific feature, each presented pruning method is self-explanatory about its reasoning for pruning a particular feature. For the feature weight-based pruning method, a researcher can directly extract it from the weights of the trained model, why a specific feature is pruned. In

pruning method	pruned	$hns(s)$	s	s_{fn}
-	0%	51.5%	-2.7	-2.7
p_t	25.4%	134.0%	19.9	26.7
p_{ig}	39.7%	83.2%	5.2	8.6
p_c	33.4%	55.4%	-0.8	-1.2

Table 5.4.: Experiment results for applying different pruning methods, showing the potential of pruning the given features. The score is calculated by using the updated human normalized score equation. With the pruning method p_t as the feature weight-based pruning, p_{ig} as the integrated gradient-based pruning and p_c as the correlation-based pruning. The second column shows the number of features pruned.

contrast, the integrated gradient value of the feature delivers the reason for the integrated gradient-based pruning method. The correlation-based pruning method can be validated directly by the correlation values and map generated before the training. Consequently, choosing the pruning method depends on the user, for what reason they want to prune a feature. Either they prefer the reason to be the integrated gradient value of the feature, its average weight, or its correlation to other features.

5.7. Minimizing the policy model

This experiment aims to evaluate the reduction of the model size that is trained with the reinforcement learning algorithms for solving task $T3$. Given the architecture of having one hidden layer between the input and output nodes, this experiment will remove this hidden layer. It will result in a linear model since the input nodes are now directly connected to the output nodes. In other words, the given meaningful features as the input are directly connected to the actions as the output nodes. Theoretically, when having a model with no hidden layers, a researcher can now extract the behavior of a model from the weights of the selected action. Each connection from the input features to the action chosen by the agent has its weight which is directly telling how much the input feature is influencing to select this specific action [9, p. 3]. Thus having a working model architecture with no hidden layer as the policy would reduce the effort into getting the reasoning and understanding of the behavior applied from the policy. The selected games are Pong, Tennis, and Boxing. It allows comparing the result to the experiment, where

algorithm 1 has been introduced since the model’s input in both experiments is the same.

Game	h_n	Random	DeepGA	REINFORCE
Pong	1	0.8%	115.2%	51.5%
	0	0.8%	118.0%	0.0%
Tennis	1	1.3%	293.0%	87.3%
	0	1.3%	273.9%	0.0%
Boxing	1	89.3%	175.6%	138.7%
	0	89.3%	154.6%	114.8%

Table 5.5.: The final experiment results in human normalized scores. For choosing the agent for DeepGA, the algorithm selected the elite agent from the last generation. h_n is the number of hidden layers used. The results of using the hidden layer from the previous experiment are listed as well. REINFORCE has some problems with training a linear network. However, when the linear model is learning, its performance of using no hidden layer is nearly similar to its performance of using one hidden layer.

The experiment results in table 5.5 show, REINFORCE has problems learning the linear model. Depending on the game, it either achieves similar performance compared to the model with one hidden layer or does not learn anything with the given model. Besides that, the achieved scores are nearly the same as the scores by using one hidden layer. This shows, the meaningful features given to the model are processed enough, the model does not need an extra hidden layer to create an internal state from it. It can directly map the features to the specific actions while achieving usable performance.

5.8. Model-based approach of explainable reinforcement learning

This experiment aims to evaluate the possibility of creating a world model based on the intermediate results. The used intermediate results could be either the extracted raw features or the processed meaningful features. Typical world models are trained by using images from the world to predict [13, p. 2]. Compared to recent world model-based reinforcement algorithms [26][14][15], creating a world model with raw features or meaningful features instead of given images should simplify the model and its training due to now having a regression problem. The main advantage of having a world model based on one of the intermediate results would be the possibility of using one of the recent

model based reinforcement learning algorithms (or even developing a new one) while keeping the pipeline presented in chapter 4. The approaches for getting explainability and understandability made with the designed pipeline will be held by keeping its design. In this experiment, the world to model is one of the selected Atari2600 games. Before testing a simplified world model based on an existing model-based reinforcement learning algorithm, the following section presents the design of the simplified world model and its reinforcement learning algorithm.

5.8.1. Minidreamer: A simplified version of DreamerV2

Taking DreamerV2 (2.1.4) as a template due to its ratio between performance and computational need as a model-based reinforcement learning algorithm, the complexity of its main models are reduced. From here, the minimized DreamerV2 will be called Minidreamer. As a first approach, the state to predict by the world model consists of raw features extracted by the solver of task $T1$. Therefore, the components of the world model architecture can be stripped down into only two MLPs. The first MLP is to predict the next raw features, e.g. the coordinates of the next frame for all game objects. The second MLP is to predict the possible reward returned for the given state. Both MLPs take the current state and action as the input. Otherwise, the core principle to train the two components, the world model and the policy, is kept roughly the same as in DreamerV2 [15, p. 2]:

1. The policy interacts in the given environment and creates a history consisting of states.
2. The world model learns from the saved history of states to predict the next state by any given state.
3. The world model is used to predict state after state for training the policy. The policy learns only with the predictions by the world model.

As the world model can predict the next state by any state, the policy is not bound at learning by interacting with the environment. Consequently, future developers could theoretically use planning algorithms.

5.8.2. Evaluating Minidreamer's first implementation

As mentioned in the previous section, the world model of the algorithm will only learn to correctly predict the raw features from the selected world besides the reward. The policy

is kept as a random action chooser since the actual difference between the presented algorithm and the already existing algorithms lies in the training of the world model. Due to technical limitations, the world model is only trained with Pong and Tennis as the worlds to learn.

Game	D	mean	std	min	25%	75%	max
Pong	Identity	-0.11	1.47	-5.26	-1.00	0.76	5.02
	Next state	-0.02	1.00	-2.96	-0.59	0.54	5.41
Tennis	Identity	-0.02	0.58	-3.57	-0.41	0.32	2.29
	Next state	-0.02	0.46	-2.07	-0.3	0.26	2.12

Table 5.6.: Statistical values about the difference between the predicted state to the identity and the actual next state. They are showing, the prediction is more often closer to the next state than to the identity. Column 'D' describes the state to subtract from the prediction. The data taken are limited to the last 10% steps of the training.

Table 5.6 shows that the predicted next state is closer to the actual next state than to the given state as the identity most of the time. Manually evaluating the predicted states by comparing their values reveals that the prediction lies most of the time between the actual next state as the correct values to predict and the identity while being closer to the correct values to predict. In addition, specific game scenarios, e.g. when the ball bounces off from the side of the pitch or game objects are disappearing and appearing again, the state prediction of the world model is way off the identity and the correct values to predict. In addition, even when the reward gets predicted accurately, it should be observed with caution. The actual reward of the history used for training the world model has kept near the maximum negative score since the policy to create the history consists of randomly selected actions. The list of possible changes, which may raise the performance, seems large after the first evaluation. It includes using bigger sequences to train the world model, training a policy with various reinforcement learning algorithms, using more resources to find better hyperparameters, or trying different model architectures. Whether the quality of its prediction can achieve a level that its quality can train an actual policy must still be evaluated. As a result, this thesis will not further evaluate this approach. Otherwise, further implementing and optimizing the algorithm will go beyond the scope of this thesis.

6. Exemplary investigation for a polices behavior

6.1. Using integrated gradient values to interpret behavior

After showing different approaches to how developers could implement the presented pipeline from chapter 4, this chapter now shows first approaches how applying specific techniques can explain the final decision made by a policy. A quickly usable method is extracting the integrated gradient values (3.1.1) for each feature over a whole episode. The integrated gradient value (IG-Value) shows the importance of the specific feature from the policy's perspective for the selected action in the current step. The extracted IG-Values are from the model trained for Boxing by DeepGA from section 5.5. Theoretically, a model with pruned features could also be selected. The only difference between extracting the IG-Values from the models with pruned features compared to the other models is that the IG-Values of the pruned features are always constant zero. A shallow model could also be chosen, but they offer another approach for understanding its decision. It will be explained in the next section.

Feature	$F_d^x(e, p)$	$F_d^y(e, p)$	$F_v(p)$	$F_v(e)$
norm.IG-Value	100.0%	88.7%	9.5%	4.2%

Table 6.1.: Showing the two highest and lowest normalized IG-Values. This model rates the distance on the x- and y-axis as the essential feature for its decisions, while the velocities of both game objects are influencing the decision at least. The IG-Values are normalized by taking their absolute mean and putting them in relation to the highest absolute mean IG-Value.

Table 6.1 shows the two highest and lowest normalized mean IG-Values over a whole episode for the selected actions. The trained policy takes the distance between the two

game objects on both axes as the essential feature for its decision with the received results. Meanwhile, the velocity of both game objects has the weakest influence on the policy's decision. Depending on the game, it is now interesting to see whether the importance of each feature changes over specific game situations. In boxing, a separation of game situations would be having both players close to each other, and far away from each other.

Distance > 20	Feature	$F_d^x(e, p)$	$F_d^y(e, p)$	$F_v(p)$	$F_v(e)$
	norm.IG-Value	100.0%	53.7%	2.7%	0.3%
Distance ≤ 20	Feature	$F_d^x(e, p)$	$\mathbf{F_t^y(p, e)}$	$F_v(p)$	$F_v(e)$
	norm.IG-Value	100.0%	82.3%	7.1%	1.7%

Table 6.2.: The features with the highest and lowest IG-Value split over two different game situations. The first game situation is when the distance between the player and the enemy is bigger than 20, representing not being close together. The second situation is therefore having the player and the enemy close to each other. The IG-Values show, the policy weights feature importance differently when having different game situations. The value 20 is manually selected as the threshold value after visually examining the game.

Table 6.2 shows an example, how the feature importance can change depending on the game situation. From a logical perspective, the policy has different strategies depending on the distance of the own controllable player and the enemy.

6.2. Interpretation approach of a linear model policy

As hinted in the previous section, having a network architecture without hidden layers as the policy allows to directly extract how each feature influences any decision from its trained weights. Table 6.3 shows exemplary the weights of each input feature. Consequently, there is no need to apply Post-hoc explainability approaches to understand the behavior of the policy.

A	$W_{F_v(p)}$	$W_{F_d^x(e,p)}$	$W_{F_d^y(e,p)}$	$W_{F_m^y(e,p)}$	$W_{F_m^x(e,p)}$	$W_{F_v(e)}$	$W_{F_m^y(p,e)}$	$W_{F_m^x(p,e)}$
N.	0.14	-0.14	0.06	0.08	-0.25	0.15	-0.13	0.25
F.	0.04	-0.05	0.08	-0.01	0.16	-0.16	-0.01	-0.03
U.	-0.07	-0.12	0.15	0.06	-0.08	-0.30	-0.19	-0.04
R.	-0.12	0.12	0.14	0.05	-0.30	0.42	0.05	0.38
L.	0.28	-0.29	-0.49	-0.16	0.32	-0.56	0.23	-0.20
D.	0.30	-0.08	0.59	0.11	-0.30	0.16	-0.19	0.15
U.R.	-0.22	0.05	-0.36	-0.08	0.30	0.51	0.45	0.16
U.L.	-0.06	-0.33	0.10	0.07	-0.18	0.21	0.13	0.36
D.R.	-0.28	-0.14	0.18	-0.07	-0.14	-0.44	0.03	0.07
D.L.	0.13	0.15	0.14	0.04	0.13	0.09	-0.35	0.19
U.F.	-0.25	-0.23	0.02	-0.07	0.22	-0.22	-0.29	0.23
R.F.	-0.18	0.04	0.05	-0.20	0.03	-0.02	-0.22	-0.12
L.F.	0.07	0.01	-0.14	-0.10	0.29	-0.10	0.17	0.19
D.F.	-0.11	-0.36	0.48	0.02	0.28	0.03	0.28	0.13
U.R.F.	0.53	-0.15	0.18	0.30	0.34	0.41	-0.28	-0.35
U.L.F.	-0.18	0.05	0.36	-0.29	0.21	-0.04	-0.26	-0.10
D.R.F.	0.32	0.30	-0.33	0.18	0.06	0.11	0.18	0.27
D.L.F.	0.29	0.13	-0.48	-0.15	-0.34	0.01	-0.25	0.11

Table 6.3.: Weights for each input feature showing their impact on the decisions the policy makes. The weights of the feature with the most significant influence on the actions are highlighted. The weights were extracted from the model trained in section 5.7 for Boxing by DeepGA. (N: NOOP, F: FIRE, U: UP, R: RIGHT, L: LEFT, D: DOWN)

7. Conclusion and Outlook

7.1. Conclusion

This thesis has presented an approach to design a reinforcement learning pipeline to easier understand its (intermediate) results and behavior. Compared to other strategies for explaining reinforcement learning models, designing a transparent algorithm with understandable intermediate results helps more than applying Post-hoc explainability approaches to understand a model's behavior. Having a separate solver for the given subtasks can help to evaluate better and understand each solver. In addition, the smaller solver for the subtasks allows applying more advanced techniques for these subtasks since each solver can be developed and trained individually.

The experiments made for this thesis have approved the advantages of having the designed pipeline for a reinforcement learning tasks. The intermediate results from the subtasks have a level of understandability that a human can evaluate the performance of the subtasks solver. For example, comparing the performance of SPACE with the ground-truth in the form of AtariARI has resulted in the decision, the quality of SPACE's results is not good enough for the subsequent subtasks. Additionally, the experiments have shown, applying Post-hoc explainability approaches, calculating the integrated gradient values as an example, to a transparent algorithm helps to develop conclusions about the behavior and reasoning of the trained model with more information value than only applying Post-hoc explainability approaches to already existing reinforcement learning algorithms used for solving the whole given task.

To sum it up, this thesis has shown the potential of designing a reinforcement learning pipeline with the goal of better understanding the whole process from the input to the output. Further developing this approach should return an even better understandable solver for the subtasks and the entire task while still achieving viable performance.

7.2. Outlook

Researchers and developers should further evaluate the experiments' implementation with more games or even more different reinforcement learning tasks. This includes investigating existing approaches (and maybe designing new approaches) for extracting raw features from the input since AtariARI only supports a handful of games. This includes examining different approaches for processing raw features to meaningful features since different tasks could have different raw features than position coordinates. In addition, they can evaluate more model-free reinforcement learning algorithms for generating the policy to compare their performance. However, since every model-free reinforcement learning algorithm to create a policy result in just a policy, applying more model-free algorithms should only help find the algorithm that delivers the policy with the best performance.

As already hinted with the first implementation of Minidreamer, they should further examine it. Since examining it with the needed resources would go beyond the scope of this thesis, the required effort until usable results are achieved should not be underestimated. At first, they must evaluate the proposed world model, whether its current approach has enough potential to model the world with the needed accuracy for generating the policy or choose a different approach. After having a usable world model, they should evaluate the potential of using a planning algorithm. It should be focused, how using a world model and a planning algorithm could benefit the task solver compared to using a model-free algorithm without a model.

They should also examine to apply more Post-hoc explainability to the existing implementations except just extracting the integrated gradient values. The resulting interpretations about the behavior from using different approaches can be compared. Typically, all interpretations and explanations extracted are expected to be relatively close. Therefore, its concatenation could be a behaviors explanation with a deeper foundation of truth since it validates itself with the multiple Post-hoc explainability approaches.

Acknowledgement

The author thanks Quentin Delfosse and Johannes Czech from the Machine Learning Lab of TU Darmstadt for helping him with all kinds of questions regarding the thesis and its implementation. They spend much more time than usual helping him. The author also thanks the Machine Learning Lab for providing access to the DGX2-Server used for some experiments. He thanks Paras Chopra for providing an elegant implementation basis for DeepGA¹. Finally, he thanks for the help and support of his family and friends, especially his brother Bipesh Adugodi Vittal who supported him on many aspects regarding how to write this thesis.

¹<https://github.com/paraschopra/deepneuroevolution> (visited 10.11.2021)

Bibliography

- [1] Ankesh Anand et al. “Unsupervised State Representation Learning in Atari”. In: *CoRR* abs/1906.08226 (2019). arXiv: 1906.08226. URL: <http://arxiv.org/abs/1906.08226>.
- [2] Akanksha Atrey, Kaleigh Clary, and David Jensen. *Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning*. 2020. arXiv: 1912.05743 [cs.LG].
- [3] Babajide O Ayinde, Tamer Inanc, and Jacek M Zurada. “Redundant feature pruning for accelerated inference in deep neural networks”. In: *Neural Networks* 118 (2019), pp. 148–158.
- [4] M. G. Bellemare et al. “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279. ISSN: 1076-9757. DOI: 10.1613/jair.3912. URL: <http://dx.doi.org/10.1613/jair.3912>.
- [5] Benjamin Beyret, Ali Shafte, and A. Aldo Faisal. “Dot-to-Dot: Explainable Hierarchical Reinforcement Learning for Robotic Manipulation”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Nov. 2019). DOI: 10.1109/iros40897.2019.8968488. URL: <http://dx.doi.org/10.1109/IROS40897.2019.8968488>.
- [6] Greg Brockman et al. “OpenAI Gym”. In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: <http://arxiv.org/abs/1606.01540>.
- [7] Eric Crawford and Joelle Pineau. “Spatially invariant unsupervised object detection with convolutional neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3412–3420.
- [8] Peter Dayan and Yael Niv. “Reinforcement learning: the good, the bad and the ugly”. In: *Current opinion in neurobiology* 18.2 (2008), pp. 185–196.

-
-
- [9] AD Dongare, RR Kharde, Amit D Kachare, et al. “Introduction to artificial neural network”. In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.
- [10] Eyal Even-Dar, Yishay Mansour, and Peter Bartlett. “Learning Rates for Q-learning.” In: *Journal of machine learning Research* 5.1 (2003).
- [11] Cheng Fan et al. “Deep learning-based feature engineering methods for improved building energy prediction”. In: *Applied energy* 240 (2019), pp. 35–45.
- [12] Sam Greydanus et al. *Visualizing and Understanding Atari Agents*. 2018. arXiv: 1711.00138 [cs.AI].
- [13] David Ha and Jürgen Schmidhuber. “World models”. In: *arXiv preprint arXiv:1803.10122* (2018).
- [14] Danijar Hafner et al. *Dream to Control: Learning Behaviors by Latent Imagination*. 2020. arXiv: 1912.01603 [cs.LG].
- [15] Danijar Hafner et al. “Mastering Atari with Discrete World Models”. In: *CoRR* abs/2010.02193 (2020). arXiv: 2010.02193. URL: <https://arxiv.org/abs/2010.02193>.
- [16] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [17] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: 1710.02298 [cs.AI].
- [18] Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. “Explainability in deep reinforcement learning”. In: *Knowledge-Based Systems* 214 (2021), p. 106685.
- [19] Ho-Taek Joo and Kyung-Joong Kim. “Visualization of deep reinforcement learning using grad-CAM: how AI plays atari games?” In: *2019 IEEE Conference on Games (CoG)*. IEEE. 2019, pp. 1–2.
- [20] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.
- [21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

-
-
- [22] Han S Lee et al. “Can Deep Neural Networks Match the Related Objects?: A Survey on ImageNet-trained Classification Models”. In: *arXiv preprint arXiv:1709.03806* (2017).
- [23] Zhixuan Lin et al. *SPACE: Unsupervised Object-Oriented Scene Representation via Spatial Attention and Decomposition*. 2020. arXiv: 2001.02407 [cs.LG].
- [24] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).
- [25] Erika Puiutta and Eric MSP Veith. *Explainable Reinforcement Learning: A Survey*. 2020. arXiv: 2005.06247 [cs.LG].
- [26] Julian Schrittwieser et al. “Mastering Atari, Go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (Dec. 2020), pp. 604–609. ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4. URL: <http://dx.doi.org/10.1038/s41586-020-03051-4>.
- [27] Amitojdeep Singh, Sourya Sengupta, and Vasudevan Lakshminarayanan. “Explainable deep learning models in medical image analysis”. In: *Journal of Imaging* 6.6 (2020), p. 52.
- [28] Felipe Petroski Such et al. “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”. In: *CoRR* abs/1712.06567 (2017). arXiv: 1712.06567. URL: <http://arxiv.org/abs/1712.06567>.
- [29] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic attribution for deep networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3319–3328.
- [30] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [31] Dino Vlahcek and Domen Mongus. “An Efficient Iterative Approach to Explainable Feature Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).
- [32] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: 1511.06581 [cs.LG].
- [33] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.
- [34] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992), pp. 229–256.

A. Appendix

A.1. Training hyperparameter and configuration

A.1.1. General algorithm default hyperparameter

The default hyperparameters for all algorithms used in the experiments are listed here. When not mentioned in the specific experiment hyperparameter section, the hyperparameters are exactly as listed here. This means, not mentioned experiments in this chapter have used only the default parameters.

Hyperparameter	Value
Generations	1500
Generation size	500
Parent list size	20
n runs per generation	3
n runs for elite	5
Top n for elite	10
Mutation power	0.02

Table A.1.: Default DeepGA hyperparameters.

Hyperparameter	Value
Episodes	50000
Max. episode size	50000
Gamma	0.97
Learning rate	0.0001

Table A.2.: Default REINFORCE hyperparameters

Hyperparameter	Value
Episodes	1000
Max. episode size	50000
Batch size	128
Gamma	0.97
Eps start	1.0
Eps end	0.01
Eps decay	100000
Learning rate	0.0002
Memory size	50000
Memory min size	25000
Stacked frames	4

Table A.3.: Default DQN hyperparameters

Hyperparameter	Value
Max steps	1000000
Batch	10
History size	50
Min history size	10
Learning rate predictor	0.001
Alpha	0.1

Table A.4.: Default Minidreamer hyperparameters

A.1.2. Specific hyperparameters changes for the experiments

Game	Algorithm	Hyperparameter	Value
Pong	REINFORCE	Learning rate	0.001

Table A.5.: Specific hyperparameter changes for experiment 5.4.

Game	Algorithm	Hyperparameter	Value
Tennis	REINFORCE	Episodes	18000

Table A.6.: Specific hyperparameter changes for experiment 5.5.

Pruning method	Hyperparameter	Value
Feature weight based	Threshold value	0.005
Integrated gradient value based	Threshold value	0.01
Correlation value based	Episodes	5
Correlation value based	Threshold value	0.75

Table A.7.: Pruning hyperparameter for experiment 5.6.

A.2. Training results

The absolute training scores from the experiments are listed here. The scores from Rainbow [17, p. 12] are also listed to have a comparison.

Game	Random	Human	DeepGA	REINFORCE	Rainbow
Pong	-20.7	14.6	21.0	-21.0	20.9

Table A.8.: Absolute training scores from experiment 5.2.

Game	Random	Human	Baseline	MLP	DuelDQN	Rainbow
Pong	-20.7	14.6	19.1	-0.3	16.8	20.9

Table A.9.: Absolute training scores from experiment 5.3.

Game	Random	Human	DeepGA	REINFORCE	Rainbow
Pong	-20.7	14.6	21.0	11.5	20.9

Table A.10.: Absolute training scores from experiment 5.4

Game	Random	Human	DeepGA	REINFORCE	Rainbow
Pong	-20.7	14.6	20.0	-2.68	20.9
Tennis	-23.8	-8.3	22.0	-10.3	0.0
Boxing	0.1	12.1	96.8	55.43	99.6

Table A.11.: Absolute training scores from experiment 5.5

Game	Random	Human	p_t	p_{ig}	p_c	Rainbow
Pong	-20.7	14.6	19.9	4.3	-0.5	20.9

Table A.12.: Absolute training scores from experiment 5.6

Game	Random	Human	DeepGA	REINFORCE	Rainbow
Pong	-20.7	14.6	21.0	-21.0	20.9
Tennis	-23.8	-8.3	18.9	-24.0	0
Boxing	0.1	12.1	73.4	28.7	99.6

Table A.13.: Absolute training scores from experiment 5.7