



ulm university universität
uulm

Ulm University | 89069 Ulm | Germany

**Faculty of Engineering,
Computer Science and Psychology**
Institute of Media Informatics
Visual Computing Group

Visualizing Deep Reinforcement Learning

Bachelor Thesis at Ulm University

Presented by:

Marios Sirtmatsis
marios.sirtmatsis@uni-ulm.de

Examiner:

Prof. Dr. Timo Ropinski

Advisor:

Alex Bäuerle
Heinke Hihn

2021

Last updated August 28, 2021

© 2021 Marios Sirtmatsis

This work is licensed under the Creative Commons **Attribution-NonCommercial-ShareAlike 3.0 Unported** License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-sa/3.0/>.

Typesetting: PDF- \LaTeX 2_ε

Abstract

Over the past years Machine Learning and Deep Learning have gained interest across multiple domains in research and industry. A rising sub-field that emerged as a combination between Reinforcement Learning and Deep Learning is Deep Reinforcement Learning. On multiple occasions Deep Reinforcement Learning was able to proof its success for multiple tasks in multiple domains and has reached wide acceptance throughout different research communities. However, similar to Deep Learning, Deep Reinforcement Learning is still considered a black box and needs further investigation for a deeper understanding. Different to Deep Learning where multiple visualisation techniques and applications exist to explain Deep Learning methods, Deep Reinforcement Learning has not had this much attention in the visualisation field yet. Even though, there exist some approaches on visualising Deep Reinforcement Learning and visualisation systems for Deep Reinforcement Learning, they most often either require a lot of user knowledge in Deep Reinforcement Learning, are not generally usable for many different Deep Reinforcement Learning settings or do not come with an easy-to-use framework to guide users from logging data up to its visualisation. For alleviating these problems, this work introduces a generally applicable visualisation tool for Deep Reinforcement Learning, DRLVis. DRLVis is easy to use for developers and experts in the field, generates easily interpretable yet sensible visualisations and is independent of the underlying Deep Reinforcement Learning setting or algorithm. This work also includes a case study to showcase the applicability of DRLVis.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Structure of this work	2
2	Background	3
2.1	Reinforcement Learning Overview	3
2.2	Deep Learning	3
2.3	Methods in Deep Reinforcement Learning: Deep Q Learning	5
2.4	Reinforcement Learning: A formal Introduction	6
3	Related Work	9
3.1	Visualisation Techniques in Deep Reinforcement Learning	9
3.2	Visualisation Systems for Deep Reinforcement Learning	11
3.2.1	DQNViz	11
3.2.2	ReLVis	12
3.2.3	DRLViz	13
3.2.4	PolicyExplainer	14
3.3	Approach	15
4	Requirement Analysis	17
4.1	Development Process	17
4.2	Requirements	18
4.2.1	Analysis Specific Requirements	18
4.2.2	Tool Specific Requirements	19
5	Visualisation	21
5.1	Core Visualisation Concept	21
5.2	Overview	22
5.2.1	Episode Information Panel	23
5.2.2	Episode Charts	24
5.2.3	Timestep Charts	26
5.2.4	Video View	27
5.2.5	Modular Chart Inclusion	28
5.3	Confidence View	30
5.3.1	The Random State Experiment	30

CONTENTS

5.3.2 Confidence Chart	31
6 Technical Realisation	33
6.1 Architecture	33
6.2 Frameworks	35
6.2.1 Vue.js/Vuex	35
6.2.2 D3.js	35
6.2.3 Flask	35
6.2.4 Tensorflow and Tensorboard	35
6.3 Logging	36
6.4 Backend Preprocessing	38
7 Case Study	41
7.1 Learning Environment	41
7.2 Algorithm	41
7.3 Use Case: Training a Deep Q Network	43
8 Conclusion	49
8.1 Limitations	49
8.2 Future Work	50

1 Introduction

In the past decade multiple approaches in the Deep Learning (DL) subfield of Machine learning have proven to be promising. Combining these approaches of Deep Learning with the research domain of Reinforcement Learning (RL), has not shown to be successful for a long time. Still, there have been multiple approaches, which eventually further improved upon this idea and at least after Deepmind's successes with their Deep Q Network (DQN) [23], the Deep Reinforcement Learning (DRL) field has experienced rising interest and attention. Multiple approaches for different tasks in the domain have, since then, proven to be successful and even yielded astonishing results. For instance, DRL was the heart of Deepmind's AlphaGo, which was able to win against the world's best player in the game of Go [32]. Also, DRL is increasingly applied in real world scenarios and one of the hottest topics in Machine Learning research. However, DRL methods still require lots of prior knowledge in different domains across math or computer science and DRL algorithms are still black boxes to most people. Although there exist multiple approaches on unveiling the black box of Deep Learning through visual analysis, DRL has not had this much attention in the visualisation field [40]. This work's goal is to create another visualisation tool for DRL training processes and to help increase interpretability of DRL processes.

1.1 Motivation

To alleviate this lack of methods in visual analytics methods in the DRL fields, some approaches have been developed over the past few years and with them came interesting insights about known algorithms in DRL as well as better explainability about agent behaviours. To only name a few, DQNViz focused on increasing interpretability for DQNs, DRLViz analyzed DRL methods with memory and PolicyExplainer was able to retrieve visual explanations on agent behaviour [37, 11, 22]. Even though, these approaches brought new insights, they also came with some drawbacks. First, most of the visualisation tools in DRL were developed for experts who already have plenty of experience in the field and are able to analyze meaningful, but complex visualisations. Second, most tools answer specific research questions for specific fields in the DRL domain and therefore can not be applied generally for any DRL setting. And third, most of existent DRL visualisation tools do not come with an easy-to-use framework and thus, include major overhead for integrating the visualisation tool into ones DRL training pipeline. This work focuses on alleviating these issues by introducing DRLVis, a generally applicable, easy-to-use and yet sensible visualisation tool. DRLVis is capable of visualising DRL training processes for any DRL setting and is not bound to specific DRL algorithms. Also, DRLVis is easy to integrate in ones training pipeline, as it comes with a logging framework for saving data

throughout running a DRL algorithm and a user-friendly frontend to present the saved data properly. Thanks to its simplicity, DRLVis can be used by researchers in the DRL domain as well as by developers in DRL. The visualisation system is published on github¹ and free to use.

1.2 Structure of this work

- 5 In the following, the structure of this work will be explained. First, chapter² will focus on providing the necessary theoretical framework for understanding RL and most importantly DRL. Chapter² also introduces the Deep Q Learning algorithm, which is the used DRL algorithm for training and example visualisations in this work. Chapter³ will summarize related work in the field of DRL visualisation. It will first introduce different techniques, which are commonly used for
10 visualising DRL and second, sum up different visualisation tools in the DRL field. In chapter⁴ the requirements for DRLVis will be demonstrated. Chapter⁵ will present the visualisations of DRLVis and therefore marks the core part of this work. Following, in chapter⁶ further details about the inner workings of the tool will be explained. In chapter⁷ a case study will be conducted on the task of training a DQN to perform reasonably on the cart-pole problem. Chapter⁸ will
15 summarize the approaches that were introduced in this work, bring up limitations of this work and also motivate future research directions in visualising DRL.

¹<https://github.com/masirt/drlvis>

2 Background

For a sense on why visualisations are needed for guiding DRL, one has to first understand the concepts of (Deep) RL. In the following sections, the main problem of RL will be stated and concepts around RL will be introduced. Furthermore, the use of Deep Methods for RL applications will be motivated and explained on the prominent example of DQNs.

2.1 Reinforcement Learning Overview

Reinforcement Learning is a sub-field of Machine Learning and combines different approaches among disciplines like statistics, learning, and computation. It is often wrongly classified as a part of supervised or unsupervised learning. However, it does not have the characteristics for being treated as an approach of these fields [34]. While supervised learning, on the one hand, focuses on independent and identically distributed (i.i.d) datasets, which are available at training time, in RL, datasets are created via interaction with an environment and can not be assumed as i.i.d by any means. On the other hand, unsupervised learning includes methods of detecting patterns and clustering techniques, which are not in the scope of RL approaches. Figure 2.1 from the work of Li accurately places RL and DRL into the whole field of Machine Learning and even broader, Artificial Intelligence [16].

Overall, Reinforcement Learning is about achieving a *goal* with an *agent* through interaction with an *environment*. This is done in a way where the agent learns how to correctly behave for reaching the goal. Specifically, by explicitly or implicitly creating a mapping from situations to *actions* which result in the highest probability for success. For giving an agent directions on which steps on its way to the goal are preferable, feedback through numerical *rewards* is used. The last section of this chapter will focus on formally embedding this intuition into a mathematical framework.

2.2 Deep Learning

Deep Learning is a field placed in the broad area of machine learning as depicted in Figure 2.1. At its core, Deep Learning is about applying machine learning techniques using *Neural Networks*, which in their essence, are graphs consisting of different layers which contain a certain amount of nodes. For demonstration purposes, Figure 2.2 depicts a standard feedforward neural network.

2 BACKGROUND

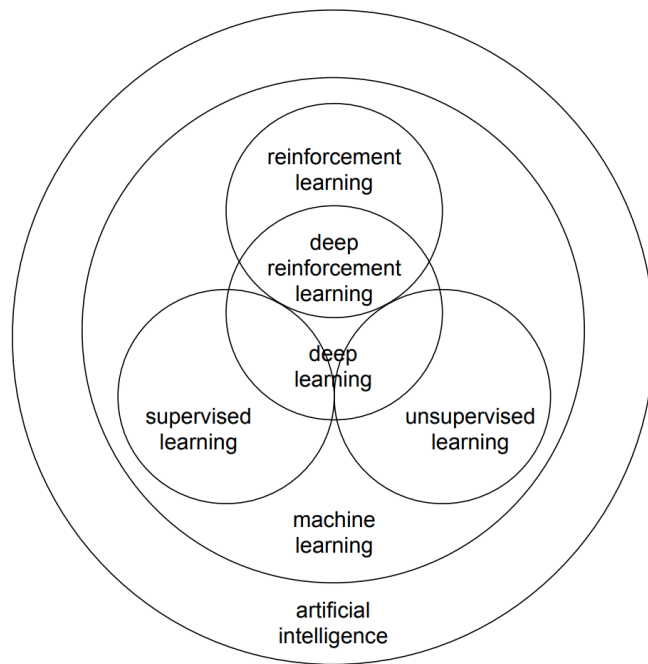


Figure 2.1: Intersections and relationships between different types of learning in the Machine Learning landscape from Li's work [16].

The circles are the nodes of the networks which are vertically stacked into multiple layers, which are then placed side by side, horizontally.

Using *deeper* Neural Network Architectures resulted in some of the most promising approaches across many different tasks, like image classification [14] or data mining on graphs [13]. Overall, deep learning is capable of learning high quality representations of highly complex datasets through different *optimization techniques* and proofed its *raison d'être* multiple times over the last decade.

The big emergence of Neural Networks began in the last decade, but still, one of the first works in RL using deep learning techniques was from Tesauro in 1994, where a neural network was trained to play the game of backgammon astonishingly well, by training a Neural Network through self play [35]. Even though, DRL was not as highly researched from then on, the increased computational capabilities, well-established methods in RL and intriguing approaches in solving tasks through Deep Learning resulted in more recent approaches in DRL, making it one of the most researched fields in computer science. These approaches, and specifically one of the most prominent (Deep Q Learning [24]), will be discussed in the next section.

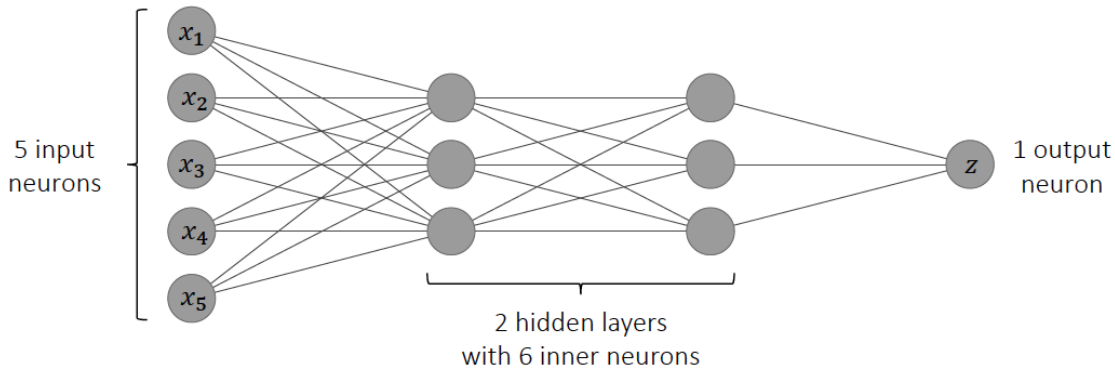


Figure 2.2: An example illustration of a deep neural network with 5 input neurons/nodes, 2 hidden layers with three inner neurons each, and 1 output neuron. The network is densely connected, meaning every node from a layer $n - 1$ is connected to every node of layer n . Figure extracted from lecture notes from Timo Ropinski [27].

2.3 Methods in Deep Reinforcement Learning: Deep Q Learning

In the last decade, lots of approaches emerged on solving the RL problem through neural networks. Most of these approaches focus on using neural networks as *function approximators* for optimal (action) value functions and variations of just these. (e.g. advantage functions) Using approximations is necessary, as calculating optimal values for states as defined in equation (2.4) through traditional methods, like Dynamic Programming, is highly inefficient in means of memory and not feasible for environments with very large state spaces, like e.g. Atari Games. This problem of RL is also often referred to as the *curse of dimensionality*.

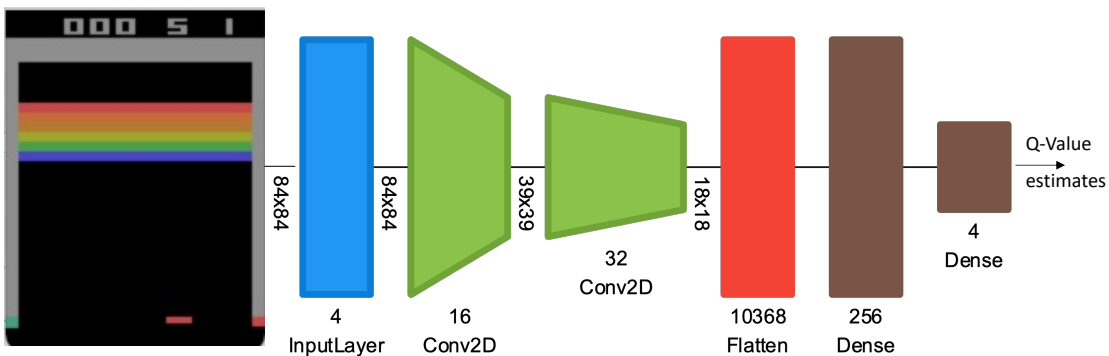


Figure 2.3: A picture of the original Deep Q Network architecture [24]. The network architecture graph was generated with the work of Bäuerle et al. [6]. The last layer has 4 output units in this example figure, but this can be arbitrary depending on the amount of actions per action space.

One of the most prominent approaches in solving this problem, and the first one to do so by creating a direct mapping from raw pixel data to a probability distribution over possible actions, is Deep Q Learning by Mnih et al. [24]. In their work, Mnih et al. introduced a complex neural network architecture, which can be found in Figure 2.3. It was exploiting recent techniques of the deep learning field, being a *convolutional neural network*. This kind of neural network was first successfully applied into a deep architecture by Krizhevsky et al. [14] and its idea was motivated by the human visual cortex. The biggest advancement of this kind of network is its ability to extract meaningful features based on the spatial structures in images and also doing so while being shift, scale and distortion invariant to some extent. However, combining this method with a variant of Q-Learning [39] using *experience replay* [18] and a *fixed target network* gave the method enough stability to provide human superior results in many Atari games. The detailed workings of the algorithm and the aforementioned terms will be described in chapter 7 within the case study, as Deep Q Learning will be used for showcasing the applicability of DRLVis.

2.4 Reinforcement Learning: A formal Introduction

In this work, we will concentrate on *episodic tasks* in the RL domain, which are tasks that terminate after a given amount of *steps* t , like e.g. games. Formally, most RL problems are formulated as Markov Decision Processes (MDP) and exploit the advantages of this mathematical framework. MDPs are defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where \mathcal{S} , the so called *state space*, is a set of states $S_t \in \mathcal{S}$ which are observed through interaction of an *agent* at time step t with an *environment* \mathcal{E} . For interacting with the environment, an agent selects actions A_t from the *action space* \mathcal{A} , as specified in the *policy* $\pi(A_t|S_t)$, which essentially is a probability distribution over all possible actions, and therefore a mapping from states to actions. While interacting, the agent also receives rewards R_t , apart from observing environment states, which are generated by an underlying *reward function* $\mathcal{R}(A_t, S_t)$, and reaches a successor state S_{t+1} with a *transition probability* $\mathcal{P}(S_{t+1}|S, A)$. This loop of interaction is illustrated in Figure 2.4.

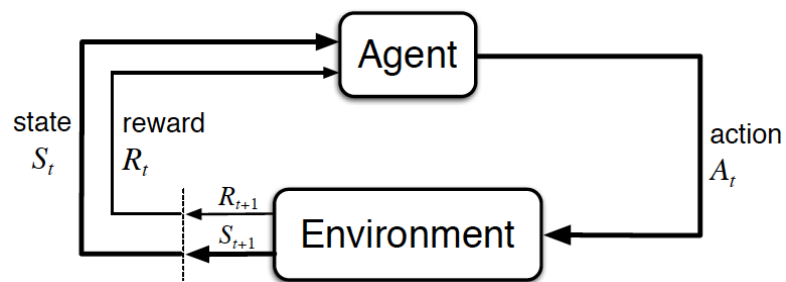


Figure 2.4: A depiction of the interaction cycle between agent and environment. Per time step t an agent receives a reward and a state from the environment, upon which it selects a new action. This graphic is from Sutton et al. [34].

The formal episodic goal of an agent is to maximize the *expected return* G_t , which is defined in equation (2.1). γ is the *discount factor*, weighing future rewards against immediate rewards and T is the final time step.

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (2.1)$$

For reaching the aforementioned goal, most algorithms in RL use *value functions* $v_\pi(s)$ or *action value functions* $q_\pi(s, a)$. Value functions give expectations on the return that an agent is going to earn for being in state s and following policy $\pi(a|s)$ thereafter. Analogously, action value functions provide an expected return for being in a state s , taking action a from there and following policy $\pi(a|s)$ after that. Intuitively, like stated in the work of Sutton et al. [34], value functions provide an awareness on how desirable it is to be in a certain state and action value functions create a sense on how desirable it is to be in a certain state and select a certain action after that. Equations (2.2) and (2.3) lay out the mathematical definition of both value functions and action value functions [34]. One can see that equation (2.2) defines the value function as the expected return for starting from state s and following policy π thereafter. Analogously, starting from state s , selecting action a and following policy π from there on, describes the action value function.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^T \gamma^k R_{t+k+1} | S_t = s\right] \quad (2.2)$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^T \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \quad (2.3)$$

For the sake of simplicity, it is only focused on finite MDPs in this work, where $T < \infty$. By doing so, the optimal value and action value functions can be defined through the so called *Bellman optimality equation*. This work will mostly focus on Q-learning methods and thus, only the optimal *action value function* is defined in equation (2.4), which describes it as the action value function for following an (optimal) policy π and therefore reaching a maximal value q_* for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$.

$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \quad (2.4)$$

The Bellman optimality equation for q_* is defined in equation (2.5). It intuitively defines an optimal action value function recursively as the current reward in addition to the optimal action value function of the successor state, which is reached by selecting a maximizing action a' (i.e. following an optimal policy). This recursive equation comes handy for approaches for solving the optimality equation (2.4) and therefore getting an optimal action value function, which is desirable for solving the whole problem of RL.

$$q_*(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \quad (2.5)$$

This section gave an introduction into the mathematical background, used in almost any RL algorithm and is therefore not covering every detail, exhaustively. It shall only provide a foundation

2 BACKGROUND

for the following chapters.

This chapter gave a brief introduction into the field of Reinforcement Learning, related math and motivated the use of Deep Learning Techniques for solving the Reinforcement Learning
5 problem.

3 Related Work

In the past years many different methods of visualising deep learning models have been studied and implemented. Most of them were developed to support classical supervised or unsupervised machine learning tasks and have helped to overcome major difficulties, as well as improve the understanding of Neural Nets in general. In a recent survey, Yuan et al. summarize main visualisation techniques and categorize them into methods that are used before, while or after training a model [40]. Some of them include CNNVis, GANVis and RNNVis to name the more prominent examples [19] [38] [21]. Even though, these visualisations helped to improve the understanding across different tasks in the machine learning field, (deep) RL mostly lacked rich, well-founded and easy-to-use visual analytics tools. The next sections shall provide an overview of recent visualisation approaches in (deep) RL and embed this work's approach in the landscape of DRL visualisations.

3.1 Visualisation Techniques in Deep Reinforcement Learning

Within RL, the most common and traditional approach is, to visualise the reward an agent receives or estimated Q-values predicted by an agent [24]. These metrics can be used for analysis and interpretation of an agent's behaviour and evolution, but they do not give insights on why the agent selects certain actions and dismisses others. Thus, they treat agents as black boxes. Therefore, different approaches were used to first, cluster observations with similar activations in the last layer of a neural networks and second, use saliency techniques for observing main characteristics which lead to an agent's decision. For doing so, Mnih. et al visualised a dimensionally reduced (2D) representation of the last hidden layer in their DQN [24]. The dimensionality reduction was done with t-SNE [36], which is one of the most commonly used dimensionality reduction techniques in the visualisation landscape.

This was then even taken further by Zahavy et al. [41], who implemented a t-SNE visualisation system with the additional ability of clustering the state space with hand crafted features by color and using a new aggregation of MDPs, the so called Semi Aggregated Markov Decision Process. Doing so, they were able to discover that DQNs learn features that have hierarchical structures and therefore achieve main goals by first learning and achieving sub-goals. In their work, Zahavy et al. also visualised saliency maps of observed frames in a gradient-based manner, similar to an approach by Symonians et al. [33] to uncover which parts of an observation influence an

3 RELATED WORK

agent the most in making its decision. Greydanus et al. build upon this approach and used perturbation-based saliency techniques to overcome drawbacks of earlier saliency methods [12] and better understand an agent's behaviour and strategies [9]. They experimented with different Atari environments and were able to discover why an agent chooses certain actions in certain situations. They also evaluated their visualisations with non-experts, which were able to classify well-performing against overfitting agents, where overfitting agents were agents which learned the game specifications by heart. In Figure 3.1 one can see an example of this agent behaviour, where the agent controls the paddle on the right. This example was artificially generated through variations in the observed state, which can be found in the work of Greydanus et al. [9]. The part on which the agent focuses is colored in purple and one can see that the overfitting agent "hacks" the specifications of the game of pong and therefore knows, out of experience, which parts of the observed frame will be relevant in the future, making it more prone to badly generalizing to new situations, where its assumptions are wrong.

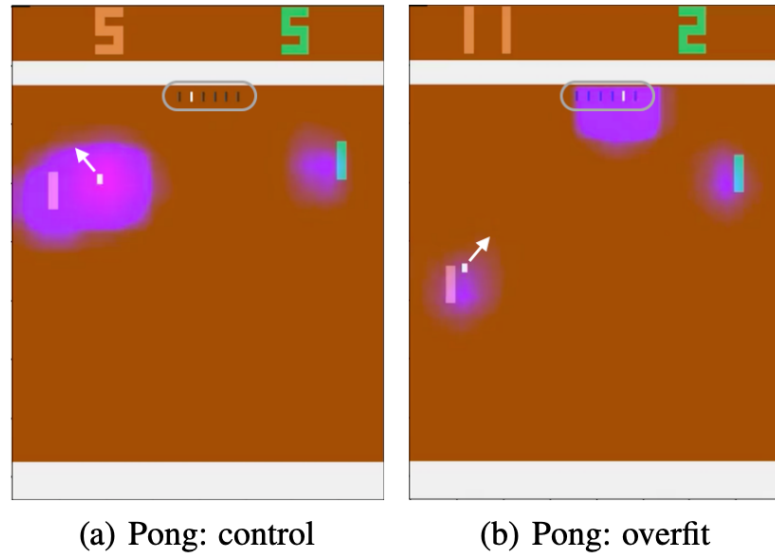


Figure 3.1: This figure from Greydanus et al. [9] shows an example for a well performing agent on the left and a overfitting agent on the right playing the Atari game of Pong, where the purple marks indicate what the agent focuses on, observed through saliency techniques.

The aforementioned approaches uncovered different behavioural patterns of agents by applying visualisation techniques without changing network architectures of underlying neural nets and also without inherently modifying RL algorithms, that were used. Annasamy et al. tackled RL interpretability with a different approach [5]. By introducing key-value stores inside a DQN-Architecture, they achieved to create global explanations of an agents behaviour, meaning an input-independent mapping from input to output of the neural net. They did this by constraining the keys in the key-value stores to be reconstructable and thus, enable a direct inversion of these

keys, which resulted in attended versions of input images. These can be used for interpreting the agent's decisions. By introducing this new interpretable DQN (*i-DQN*) architecture, they brought a more interpretable but nonetheless well-performing new approach into the landscape of DRL. Following a similar approach, Mott et al. [25] integrated attention into a RL agent. Their proposed architecture not only creates attention maps which can be used for reasoning about an agent's decisions, but also creates a bottleneck in an agent, which makes it focus on important parts of observations. They validated their method by comparing it with traditional saliency methods [9] and found different behavioural patterns in the Atari environment, through analysing generated attention maps.

3.2 Visualisation Systems for Deep Reinforcement Learning

In contrast to the aforementioned visual analytics approaches, this section gives a short introduction and description of related visual analytics *systems* in the field of DRL. These systems were developed for being used by developers and researches in the field of DRL to improve the understanding of algorithms in DRL. This section also reflects different benefits and drawbacks of these visualisation systems.

3.2.1 DQNViz

One of the first and most exhaustive visualisation tools was DQNViz, which was proposed by Wang et al. to further improve performance of Deep Q Learning Methods and shed light on the black box of DQNs [37]. They achieved this goal, by developing a four-level visualisation tool, starting from a general overall training level of visualisations down to using visualisations to inspect small blocks within the learning process, so called *segments*. Through the application of their tool, they discovered different action-reward patterns, which gave a broader understanding of the agent's behaviour, leading to interesting insights on how to choose hyperparameters for the exploration-exploitation trade-off, which is one of the major problems to solve in RL. A screenshot of DQNViz can be found in Figure 3.2. There, one can see the core visualisation concept behind DQNViz, being the four level top-down visualisation approach. Through different handles and buttons the user can further filter and select different points of interest. This guides users through exploration of training data collected during the agent-environment-interaction cycle.

However, one major drawback of DQNViz is its high complexity, making it solely applicable for domain experts in DRL or at least require an extensive introduction. Additionally, DQNViz was developed for environments with small action spaces (i.e. small number of different actions for an agent to select) only and is thus not applicable for environments with large action spaces. In addition, DQNViz is designed only for DQNs, making it constrained to a specific RL algorithm (Q-learning). Therefore DQNViz is not directly useful for analysing other RL algorithms that are not related to the Q learning setting. At last, there is no easy way for developers to use DQNViz

3 RELATED WORK

for analysing their DRL implementation. As DQNViz is not openly available and there is no data collection mechanism which guides a user from logging to visualising relevant data.



Figure 3.2: A screenshot of the main view of DQNViz from the paper’s figures [37]. The view is divided into three main parts and a smaller window. These parts are a the Statistics View, b the Epoch View, c the Trajectory View and d the Segment View, which depict the four different levels of abstraction.

3.2.2 ReLVis

Another tool on increasing RL interpretability through visual analytics is ReLVis [28]. In their work, Saldanha et al. focus on creating a system, which better suits the needs of data scientists rather than RL experts or researchers. With the goal of increasing different levels of Situation Awareness, described by Endsley [8], they introduce ReLVis, a visual analytics system for data scientists. Similar to DQNViz, ReLVis allows users to obtain a better understanding of an agent’s different behavioural patterns and learned strategies over time, but is also applicable to larger and continuous action spaces. In contrast to DQNViz, it enables users to compare the agent’s performance under different hyperparameter settings, through which it achieves an increase of a user’s hyperparameter awareness and leads to better decision making in selecting these.

Even though ReLVis seems to be a promising system to improve the performance of one’s RL implementation, it is not openly available for the use of developers in the field of RL. It also does

not support analysis specifically for deep methods, like weights or salience analysis [9], in the RL domain due to its generality.

3.2.3 DRLViz

With DRLViz, Jaunet et al. introduce another tool for visualising DRL methods [11]. The visual analytics system was created for domain experts in the field of DRL. It is specifically dedicated to DRL with memory. As depicted in Figure 3.3, the center of the user interface is a matrix, which consists of the agent's memory vectors over time, where one column contains the different dimensions of a memory vector at timestep t , the rows show the agent's memory evolution over time. Through interaction, like filtering or selecting, a user is able to identify an agent's behaviours, see the agent's internal memory, identify redundant parts of it, and connect decisions of the agent to its memory.

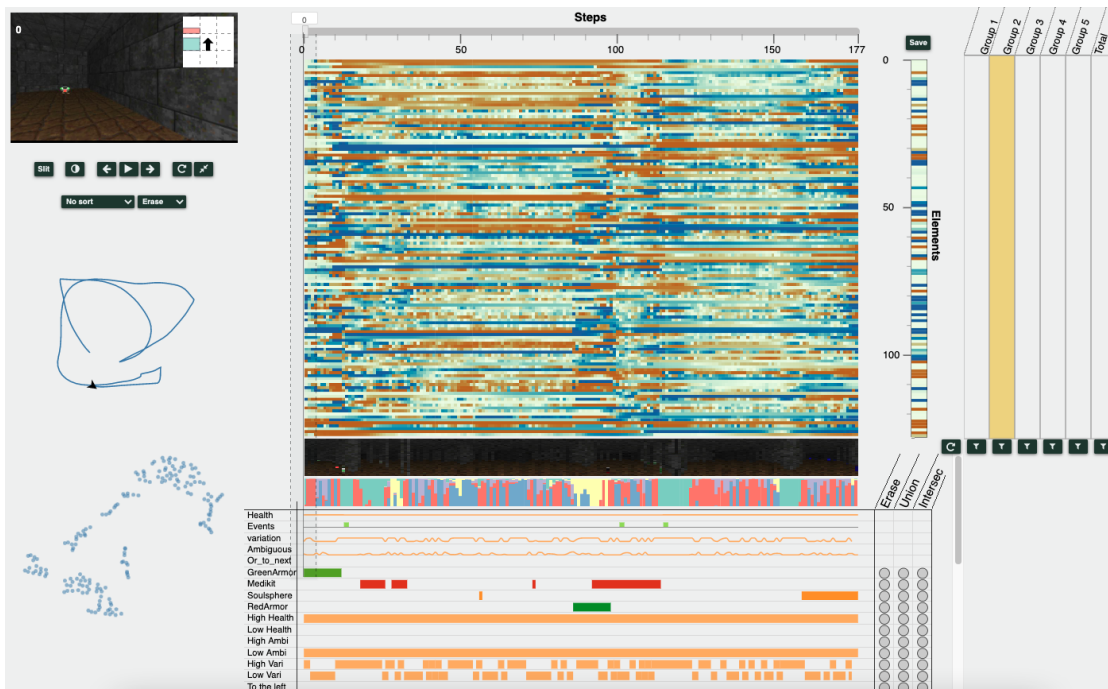


Figure 3.3: The interface of Jaunet et al.'s DRLViz [11]. It can be accessed at [1] and depicts the analytics system with scenario 3, which was also used in the paper.

Through the use of DRLViz, experts in the field of DRL with memory were able to identify interesting patterns in agent's behaviours. Nevertheless, because of its specialisation on DRL with memory, DRLViz is not generally applicable to any DRL algorithm and implementation. Furthermore, the visual analytics tool comes with a complex interface, leading to difficulties for non-experts to use the system. Also, for a more fine-grained analysis, the authors designed

3 RELATED WORK

hand-made metrics. They used them for a scenario-based analysis, making DRLViz, once again, not generally applicable to other scenarios, at least not without major integration overhead.

3.2.4 PolicyExplainer

PolicyExplainer by Mishra et al. is one of the most recent visual analytics system for DRL and the first to support querying an RL agent about certain behaviours [22]. In return, it provides visual explanations about these behaviours. The goal of this work is to provide visual explanations about RL agent behaviour, especially for non-experts in RL and especially for safety-critical domain.

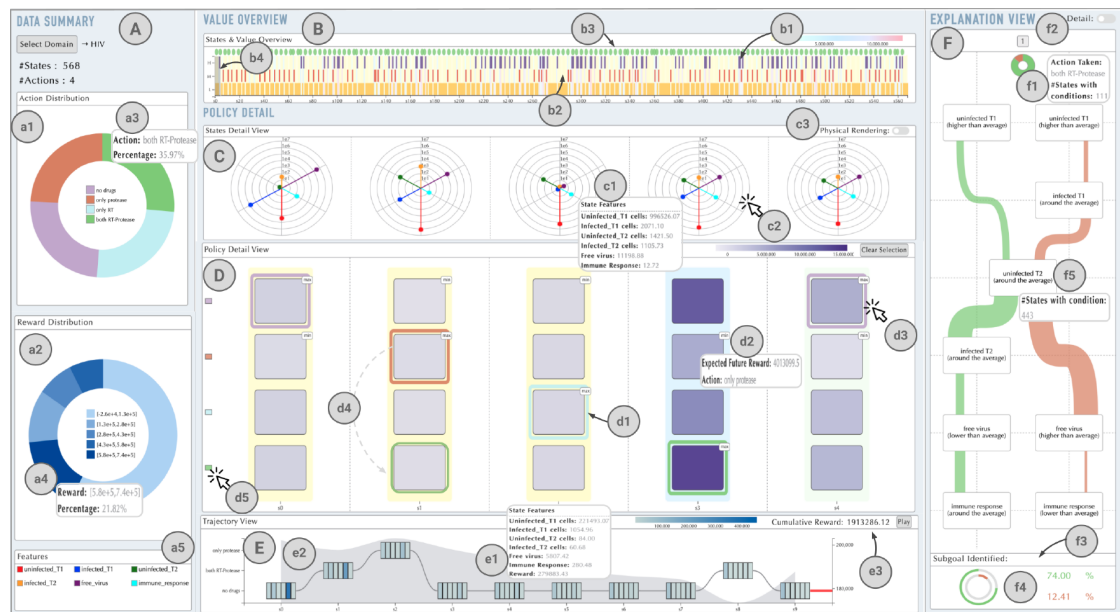


Figure 3.4: The visual interface of *PolicyExplainer* [22]. The tool consists of six different parts, (A) the Data Summary providing RL-related statistics, (B) the Value Overview depicting expected rewards at different states (C - D) States Detail and Policy Detail view, with more information on states and possible actions, (E) the Trajectory View depicting the agent's action selection over time, (F) the Explanation View, which shows the main idea of the paper, which is to answer the questions "Why? Why not? When?".

They try to tackle this problem by answering the most asked questions about RL policies: "Why this action?", "Why not this other action?" and "When is this action taken?", as has been found in previous work [10, 17]. Through their visual analytics tool, they achieved this goal by first, providing users with an overview of common statistics in RL (like action and reward distributions) and successively introducing more details up to explaining different agent behaviours. A screenshot can be found in Figure 3.4. The different parts of the tool are explained in the caption of the figure. However, the most novel part of this visualisation system lies in the *Explanation View*. Through the use of decision trees [26], the system covers users with information about why actions were

selected over others in certain situations. This gives users the possibility to open the black box of a learning agent and better reason about why an agent behaves in a certain way.

The approach followed by Mishra et al. [22] is especially interesting and will most certainly be of major importance in future work. In the context of DRL visualisation though, *PolicyExplainer* cannot be applied to environments with large state spaces, as it tries to give a state overview, which will lose in meaningfulness with huge state spaces like in Video Games [24] or many real-world scenarios. Also, the tool was designed for 3 main simple scenarios, making it not generally applicable to any DRL environment, at least not without major time expenses for integration.

3.3 Approach

In the following paragraphs, an intuition on differences between existing work in the field of visual analytics systems in DRL and this work's approach will be given. The concrete requirements for this work's visualisation system, will be described in chapter 5.

In this work, instead of focusing on specific DRL domains or DRL settings, the goal was to develop a tool, that can generally be used across the entire DRL field. Therefore, it has to be generally applicable to any DRL algorithm and not focus on specifics like e.g. Deep-Q-Learning in DQNViz [37] or DRL with memory as in DRLViz [11]. Also, in this work the focus lies on creating meaningful and expressive, but also pragmatically usable and easily interpretable visualisations. The main goal of restricting the scope to simpler but easier understandable visualisations is to expand the range of possible users for using the presented visualisation tool, DRLVis. Different to most related work, this work wants to introduce a tool that is usable for researchers and DRL experts, but also DRL developers in the field.

Furthermore, the system developed in this work is capable of performing at least some salience or weight analysis for supporting visual analytics of deep methods. The presented visual analytics system is able to generalize to any DRL scenario, having large state and action spaces, as well. The workflow of DRLVis should not start with the visualisations themselves but with appropriately logging data. This is done in a way, where users do not have much time-overhead for logging the data suitable to their problem sets. At last, DRLVis is openly available to support a broad range of researchers and developers in the field.

4 Requirement Analysis

This chapter will first show the development process behind creating the DRL visualisation system presented in this work. For this, it is especially important to understand how the requirements that practitioners have for DRL visualisation systems were abstracted. These requirements will be described in the second section of this chapter and their implementation will be annotated through the rest of this work.

4.1 Development Process

Through regular meetings with two experts, relevant techniques and requirements were elaborated for visualizations of DRL training processes. The first expert (E1), has more than 4 years of experience in the field of DRL, while the second expert (E2) has the same number of years of experience in visualisation, specifically visualisation in the machine learning domain.

Different than some of the other existing work on visualising DRL training processes, the focus of this work lies on simple but pragmatic visualisations. These should either already have been used by E1 in his daily routine and further be improved by means of quality and expressiveness, or be simply interpretable. For achieving this goal, the development process started with an elaboration of currently used visualisation techniques by E1. These included line charts created with Tensorboard - a visualisation library provided by Google [3]. The visualised data consisted of scalar values which were logged per episode, like e.g. episode returns. An outlier analysis was performed by E1 through looking into visual frames of the interaction between agent and environment in just these episodes. The first approach in this work's development process thus was, to combine these two analysis techniques into one system, where users could visually analyse charts and "zoom" into videos of the agent interacting with an environment in interesting episodes. This basic concept was the starting point for multiple iterative cycles where findings with E1 and E2 would be discussed and the visual analytics system would be improved based on these discussions.

This iterative loop consisted of three stages. First, the current state of DRLVis would be discussed with E1 and E2 in meetings and further requirements for DRLVis would be extracted. Second, new features and more expressive visualisations would be developed, resulting in a new state of the system. Third, a meeting only with E2 would be held to discuss structural and technical improvements for the system. From there, one goes back to stage 1 and repeats this cycle multiple times, having a running system at every stage in this loop. In the following sections,

the core requirements and visualisation concepts extracted from the iterative development process, will be shown and explained in detail.

4.2 Requirements

Over these multiple iterations different requirements that should be fulfilled by a visualisation system in the DRL domain were extracted. They are separated into two main categories. The first stack of requirements includes *analysis specific* requirements and describes different analyses that should be possible within a visual analytics system for DRL. The second batch of requirements consists of *tool specific* items, being about technical and implementation-specific requirements.

4.2.1 Analysis Specific Requirements

The following paragraphs include requirements that should be fulfilled by DRLVis to enable users to perform a sensible analysis. It focuses on different types of analyses users can perform on DRL processes.

AS1 Metrics Analysis. As a first analysis technique, users should be able to analyze general metrics which evaluate the agent behaviour or give an intuition on the agent's evolution over time. These shall include e.g. episode returns and other episodic metrics but should not be limited to the episodic level. The first goal of using metrics for analysis is to have an overall starting point to evaluate which parts of the DRL training process are worth analyzing in further detail. The second goal is to concretely evaluate an agent's interactions with the environment. This can be done through plotting e.g. the rewards collected per time step or the estimated Q-Values predicted by an agent per time step.

AS2 Confidence Analysis. Multiple of the visualisations used by E1 were explicitly or implicitly about analysing the confidence of an agent in selecting certain actions. Therefore, a visual analytics system for DRL should provide the possibility of analysing an agent's confidence in action selection. Also, users should be able to assess the sensibility of the magnitude of an agent's confidence in certain situations. Furthermore, the evolution of an agent's confidence in selecting actions, should be comparable over time, so that users can elaborate whether the agent is learning desired behaviours appropriately.

AS3 Video Analysis. Analyzing images or videos on agent-environment-interaction was one of the main steps in the visualisation pipeline of E1. It only seems natural to wonder what the interaction between agent and environment looks like visually and thus, this should be a main part of DRLVis. Users should have the opportunity to see how an agent behaves in every time step of an episode. Also, combining image based analysis with confidence based analysis is

something that users should be enabled to do, so that a direct mapping from confidence in selecting an action and the interaction with the environment itself can be recognized.

AS4 User Guidance. Following a top-down approach, DRLVis should guide users from an overall top-level perspective to a more fine-grained view. Therefore, users should get the handles to start a visual analysis on an overviewing level, where parts of interest should be identified. From there on, users should be able to individually select just these parts for a further, more detailed analysis. As episodes, and time steps within episodes, define RL specific kinds of abstraction of the RL framework, DRLVis should use an episode overview and a time step detail-view within episodes to guide users from an overviewing to a detailed level. Also, users should not only be guided through visual analysis, but provided with the necessary tooling to log data in an easy and effective way. Throughout, guidance should be a main focus of a visual analytics tool for DRL, for enabling users to visualise their learning processes in a fast and uncomplicated way.

4.2.2 Tool Specific Requirements

In the following requirements, the technical and tool specific requirements that DRLVis should meet, will be described.

TS1 Independence of DRL specifics. Different to earlier approaches, DRLVis should not rely on specifics within the DRL domain. In explicit, it should not be dependant of specific algorithms in DRL, like e.g. DQNs in DQNViz, but rather be applicable in for any DRL algorithm [37]. Also, DRLVis should not be dependant of specific DRL problems or DRL settings, like e.g. DRLViz is relying on DRL with memory [11]. Rather, it should be able to adapt to any DRL setting. Overall, DRLVis should, to some extent, be usable for any problem in the field of DRL.

TS2 Modality. The technical realisation of visualisations within DRLVis should happen in a modular fashion. Therefore, users should be able to include their own metrics in the logging process and let them be visualised. Also, users should not be required to use all of the default visualisation presented in this work, but able to exclude visualisations within the logging process by simply not logging the corresponding values. By doing so, users have the freedom to only visualise items that are of interest to them, which should make DRLVis individually applicable.

TS3 Easy Integration and Use. DRLVis should be implemented in a way, where it is easy to integrate it in DRL implementations. This requirement should be met, so that users do not have a major overhead for system integration but rather can focus on analysing results and improving their DRL implementations. This should also remove, or at least lower the barrier for users of including a visual analytics system in their implementation process of DRL algorithms. In addition to an easy integration, DRLVis should also be easy to use for developers and researchers across the DRL field. Especially for users new to the DRL domain, the learning curve can be quite

4 REQUIREMENT ANALYSIS

steep in the beginning. Therefore DRLVis shall help to further improve the understanding of DRL algorithms and to find possible space for improvement within DRL algorithm implementations.

TS4 Scalability. For being generally applicable, it is important for a visualisation tool in DRL to scale to arbitrary state and action space dimensions. When playing Atari games [24] or applying RL to real world scenarios, the encountered state spaces can be enormously big. Therefore, DRLVis has to be independent of this size. This means that it should not depend on small state spaces. Also, the behaviour of an agent can get very complex in certain scenarios, where action spaces can have an arbitrary size. Thus, DRLVis should once again, not rely on smaller action spaces but rather provide visualisations which are independent of the number of possible actions. However, the focus will lie on only visualising tasks with discrete action spaces.

In the beginning of this chapter the development process of DRLVis was summarized and briefly explained. Following, requirements for a visualisation tool in DRL categorized into different requirement types. The different requirements inside these categories were then further explained.

5 Visualisation

As the core part of this work, this chapter will describe the visual interface of DRLVis. First, the key aspects of the *Overview* window, which was introduced earlier will be summarized. After that, the different charts within the Overview that can be used for analysis whilst using DRLVis will be described. Following, the *Confidence View* and its core structure will be summarized. At last, a so called Random State Experiment, which is used to generate necessary data points for the Confidence Visualisation, will be explained. All example figures shown below visualise the interaction of a DQN and an Atari Breakout Environment from OpenAI Gym [7].

5.1 Core Visualisation Concept

The core idea behind how the visual interface that users interact with looks like is depicted in Figure 5.1. The figure shows a screenshot of the visual analytics system developed in this work. On top, one can see a navigation bar, which is used for switching between the *Overview* and the *Confidence View*. The Overview is divided into two parts as DRLVis offers two levels of abstraction in its visualisations. This aligns with the principle of giving an overview first and providing details on demand [31]. These two levels are the *Episode View* and the *Timestep View*. In the Episode View users can analyse data on an overall training level summarizing entire episode with certain metrics. Inside this view, users also can select episodes that are interesting to them because of some visual findings on this top level of abstraction. By selecting an episode, users can then analyse visualisations that depict information about the episode in the Timestep View. Users therefore start on an overviewing level and only further analyse episodes, which interest them. This is especially important, as large amounts of data can be overwhelming. Having an overview as a starting point of analysis, therefore, gives users better guidance.

The left part of the Overview is dedicated to analytics on an *episode level*, the Episode View, which is the highest level of abstraction for DRLVis [AS1, AS4]. In this part of the visualisation, different charts for data, collected on an episode level, are shown. The episode for which this data is depicted can be selected in the Episode View. On the right hand side, one can see visualisations on a *time step level*, assembled to the Timestep View. The Timestep View provides visualisations for data collected throughout an episode and during each interaction between an agent and an environment [AS1]. Also, the Timestep View includes a video component, which depicts the agent-environment interaction at a certain time step inside an episode [AS3]. The Confidence View, depicted in Figure 5.2, includes a *Confidence Chart* and a corresponding legend, which will be further explained in the following sections. The chart visually captures data that is collected through a guided experiment that determines the confidence of an agent in

5 VISUALISATION

selecting actions for certain, randomly selected states **AS2**. This experiment will also be further elaborated in the following sections, as well as concrete details about the visualisation concepts that were introduced in this section.

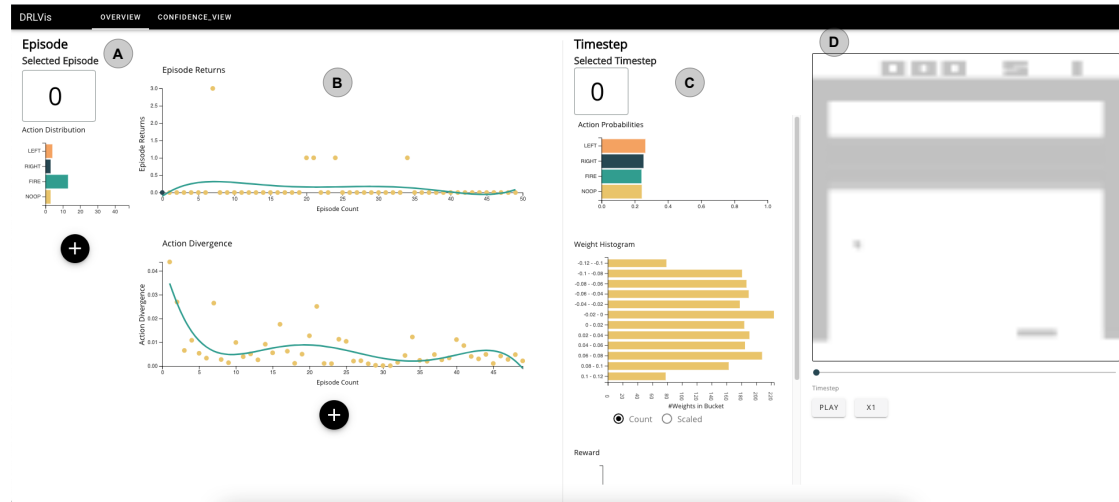


Figure 5.1: A screenshot of the visualisation tool presented in this work. The navigation bar enables users to switch between Overview and Confidence View. Section **A** and **B** show the Episode View, which includes an Episode Information Panel and Episode Charts. The sections **C** and **D** show the Timestep View including multiple timestep-level charts and a video component.

5.2 Overview

Figure 5.1 depicts the Overview window of DRLVis. On the top of the Figure, one can see the navigation bar which is used to switch between the Overview and the Confidence View. Overall, the Overview is divided into an Episode View, which captures information about all episodes that were logged and the Timestep View, which visualises data that was collected throughout an episode. Users therefore can start by analysing episode-level data on the left-hand side (the Episode View) and get themselves an overview over the underlying training information. They can then dive into a detailed analysis of interesting episodes by looking at visualisations on the right-hand side, being the Timestep View. Section **A** includes the *Episode Information Panel*. It consists of an input field, where one can select the episode which is the focus for the analysis and a bar chart that shows the distribution over all actions over the entire currently selected episode. In section **B** one can see *Episode Charts* that summarize a metric for an entire episode per data point in their scatterplot format. Section **C** covers timestep related charts, which depict information about collected data inside the currently selected episode. These *Timestep Charts* can be different kinds of data and come in different kinds of charts, which will



Figure 5.2: A screenshot of the Confidence View. Confidence Chart on the left, its legend on the right and a slider to change the episode for which the confidence chart is depicted on the bottom.

be explained in the following sections. Finally, section **D** inside the Overview mainly consists of a video component, which depicts the observed state by the agent or the visually captured agent-environment-interaction at the currently selected timestep if the agent does not observe visual but other formats of states. Both the video and the Timestep Charts can either be winded back and forth through a slider under the video component or let play through pressing the Play Button. Users can also select different playback speeds through toggling the Playback Speed button next to the Play Button.

5.2.1 Episode Information Panel

The Episode Information Panel includes two different components. The first is an input form where users can select the episode that they want to inspect in the Timestep View. This episode is then set as a global variable, which is accessed throughout the whole application. The second component is a bar chart that shows a distribution of all possible actions in the action space for the currently selected episode. In [Figure 5.3](#) one can see a screenshot of this chart. The x-axis measures the number of times the analysed agent chose an action over the entire episode. The y-axis is divided into the different actions. This chart generalizes to any number of actions and is not bound to a small action space **TS4**. To enable scaling up to arbitrary action space sizes, the bars were stacked next to one another vertically. This makes it easy to have more actions as they simply appear as additional bars. By simply logging values for more actions, the chart will visualise a distribution over all actions that were included. Hovering over a bar results in the appearance of a Tooltip, which contains the actual action count.

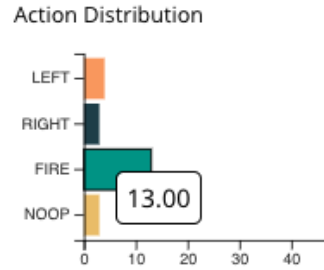


Figure 5.3: A screenshot of the action distribution chart inside the Episode Information Panel. The y-axis is a band of different actions while the x-axis measures the count of an action being selected by the agent throughout the entire current episode.

5.2.2 Episode Charts

Episode Charts are scatter plots where each data point summarizes information about an episode. The two episode charts that come per default with our visualisation tool can be found in [Figure 5.4](#). They both have the same principal structure. The x-axis depicts the episode number for the data point. The y-axis depicts a selected metric. Data for each data point is aggregated over the entire episode. A metric could be the accumulated reward throughout an episode. Scatter plots are especially well suited when it comes to visualising data points in a discrete way. Also, through them, users can easily select the episode they want to further analyse by simply clicking on a circle inside the scatter plot. When having a huge amount of episodes though, it can be hard to get a sense of the overall trend direction, so it is important to have another measure that supports a trend visualisation. That is why in addition to the scatter plot structure, an episode chart comes with a trend line, that fits all collected data points as illustrated by code in listing [6.4](#). The data point that corresponds with the currently selected episode is colored differently than all the other data points. By clicking on a data point one can change the currently selected episode to the episode count on the x-axis corresponding to the clicked data point. Hovering over points shows the exact y-axis value inside a Tooltip.

The first episode chart that comes per default can be seen in of [Figure 5.4 \(a\)](#). It shows the *episode returns* that an agent received for all episodes that were logged. One can use this chart to analyse whether the agent is increasing the received rewards over time. If this is not the case, they can look further into Timestep Charts to debug the agent. The second chart, which can be found in [Figure 5.4 \(b\)](#), depicts the *action divergence* for every episode that was logged. An agent predicts probabilities for every action that can be selected in most DRL algorithms. The action divergence is a measure to capture the divergence between these probabilities comparing episode n to episode $n - 1$. This is done by calculating the KL-Divergence between the collected probabilities of a certain episode pair [\[15\]](#). The idea behind this divergence was motivated by Schulman et al.'s work, which implies that too high divergence between the agent's predictions is not desirable [\[29\]](#). Therefore, the action divergence chart can be used to further analyse episodes where the action divergence is high and measures can be applied to avoid this anomaly,

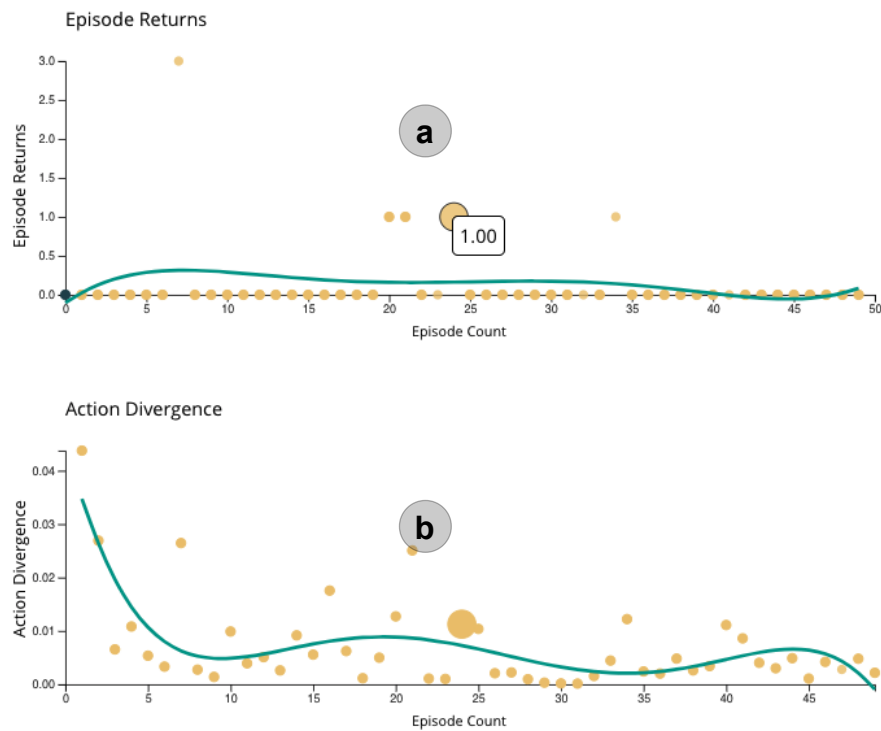


Figure 5.4: A screenshot of the default episode charts of DRLVis. They are scatter plots with a trend line and the possibility of interaction through clicking (selecting episodes) and hovering (see the actual value of the y-axis).

like e.g. reducing the learning rate. Episode charts give users the possibility to start their analysis on an overviewing level and dive further into parts of interest **AS1, AS4**.

5.2.3 Timestep Charts

In contrast to episode charts, timestep charts come in three different formats that show three different kinds of data. The first timestep chart is the *Action Probabilities Chart*. It depicts the probabilities predicted by an agent for choosing each action of the action space for each time step inside an episode. The x-axis therefore measures the probabilities from 0 to 1. The y-axis shows the actions. This chart is dynamic and changes with every time step change. Hovering over a bar makes a Tooltip visible that shows the exact underlying predicted probability. A screenshot can be found in **Figure 5.5**. Users can view this chart to analyze the confidence of an agent in selecting certain actions and combine this with the corresponding image in the video component to evaluate an agent's sensibility **AS2**. Same as the Action Distribution chart, the Action Probabilities chart scales to any amount of different actions, as more actions simply result in more bars inside the bar chart **TS4**.

15

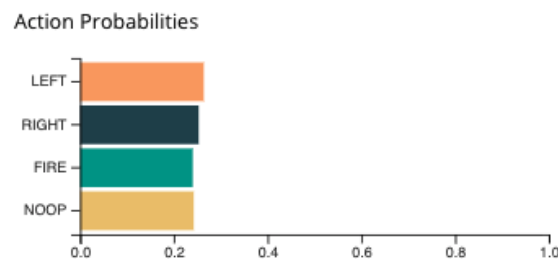


Figure 5.5: The Action Probabilities chart. A bar chart with every action being one bar and the height of the bar determining the predicted probability for selecting this action by an agent. (similar to an agent's confidence in selecting an action)

The second kind of timestep charts is the *Weight Histogram*. A screenshot is depicted in **Figure 5.6**. It shows a histogram over weights in an agent's underlying neural network. For doing so, the weights which lie between the last and the penultimate layer inside a neural network architecture were used for example purposes. All of these weights get divided into buckets based on their values. The histogram's bars show either the number or the fraction of weights inside of a bucket depending on whether *count* or *scaled* is selected in the radio button group, as it sometimes is more interesting to see the fraction of weights inside a bucket instead of the count and vice versa. By hovering over a bar, one can see the exact number or fraction of weights. The histogram always shows the logged weights for the current time step in the currently selected episode. It changes dynamically with changing time steps. With this chart, users can analyze the agent's neural network parameters (weights) and therefore look inside it. One could, for instance, imply that drastic changes in the distribution over a short period of time could indicate that the

agent is learning suboptimally because of e.g. a high learning rate.

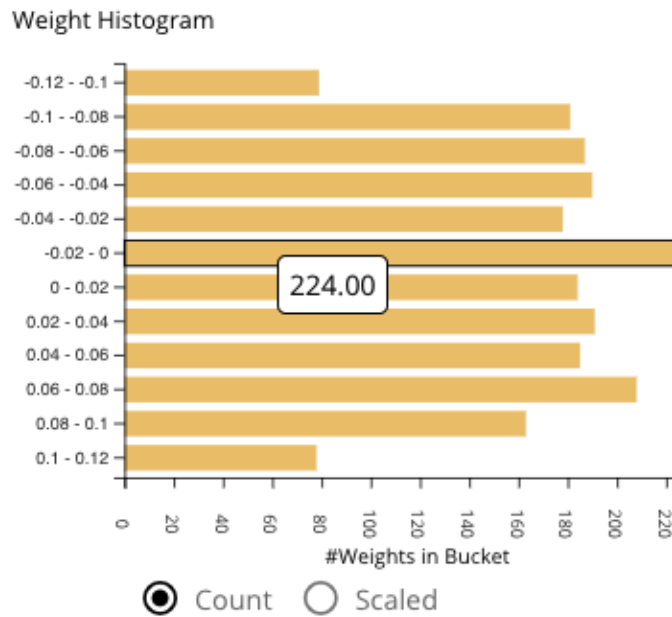


Figure 5.6: A picture of the weight histogram timestep chart. The x-axis measures the number or fraction of weights inside a bar. The y-axis measures the size of a bar bucket by showing its bounds.

The third timestep chart that comes per default is the *Reward Plot*. [Figure 5.7](#) includes a screenshot of it. The reward plot is a bar chart with a trendline. Different to the rest of the timestep charts, it captures all timesteps for the selected episode on the x-axis at once. The dynamic part of the plot is that the bar inside the plot changes its x-position to always be the current time step. The bar's height indicates the received reward at the current timestep. Similar to the episode charts, it is important to give users an intuition about the trend of the underlying data points. Therefore, a trendline is used, which fits all rewards over an entire episode and thus for each time step. When hovering over the bar, it shows a Tooltip containing the actual value of the reward and the current timestep. Same as the action probabilities chart, this plot gets most of its meaning when combined with the video view. Users can analyze which actions in which states lead to high rewards. Of course, due to the credit assignment problem in RL, immediate actions do not necessarily take credit for immediate rewards, but the plot is a starting point to analyze if the agent performs reasonably well.

5.2.4 Video View

Another component of the Timestep View is the *Video View*. It simply contains a video consisting of images that capture the agent interacting with the environment. A screenshot can be seen

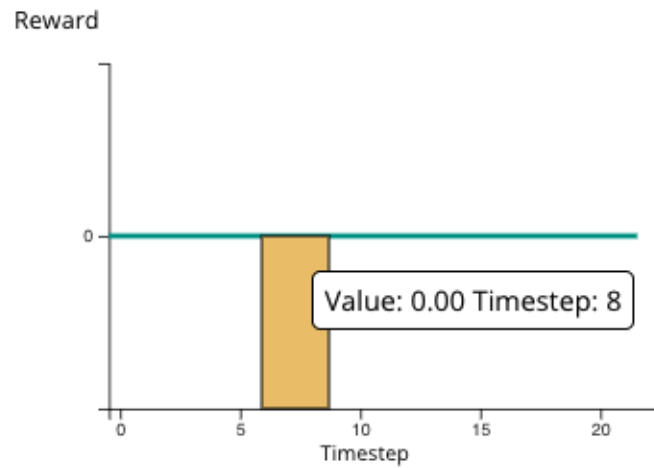


Figure 5.7: A screenshot of the Reward Plot. The trendline fits all rewards throughout the entire currently selected episode. The bar always is at the current timestep horizontally. The height of a bar indicates the magnitude of reward an agent received at the timestep the bar is currently at.

in [Figure 5.8](#). Through the slider under the video frame, users can change the current time step as they like. This not only changes the depicted image frame to the one corresponding to the current time step, but also changes every other timestep chart to depict data of the current time step in the currently selected episode. By clicking on the play button, users can start a loop where the current time step is incremented until reaching the maximal time step inside the current episode. After that, the loop starts off again from the first time step. Through toggling the playback speed button one can either speed this loop up or slow it down. The video view gives users the possibility to connect timestep charts with the actual visual behind the interaction between agent and environment [AS3](#). This shall help users to better understand why agent's behave as they do and in some cases also find patterns in an agent's behaviour corresponding to some states within the environment.

5.2.5 Modular Chart Inclusion

With the goal of being modular [TS2](#), DRLVis also comes with an option to add further plots for data that is interesting for users. Therefore, there exist three *Add-Buttons*, which are illustrated through a white plus sign outlined by a black circle. These exist for three different underlying chart formats: The action distribution chart, the episode chart and the reward plot. By simply logging data in the same format as for these charts and by giving them custom tags in the logging process, users can visualise other data than the aforementioned.

For doing so, they first have to log the data correctly by using the provided logging framework. By then clicking on one of the Add-Buttons, they can plot their data in the specified format. This



Figure 5.8: A screenshot of the Video View. It is in grayscale as in this case it depicts the observation of a DQN playing Breakout like in the work of Mnih et al. [24], where colored frames were preprocessed to grayscale ones before passing them to the DQN. One can interact with the view through the play button, the playback speed toggle button and the slider.

5 VISUALISATION

will be further illustrated in the case study by a hands-on example. However, when clicking on an Add-Button, a dialog appears that shows other logged tags with the same format as the chart format one wants to use for visualisation of custom data. Figure 5.9 shows just this dialog. Users can select the custom logged data they want to visualise and which they logged under this custom name tag before, by selecting the correct radio button. They can also give the chart a title by typing it into the dialog's text box. By clicking on Save the chart gets appended into the right spot in the Overview. If one does not want to include it anyhow, the close button closes the dialog without including a custom chart.

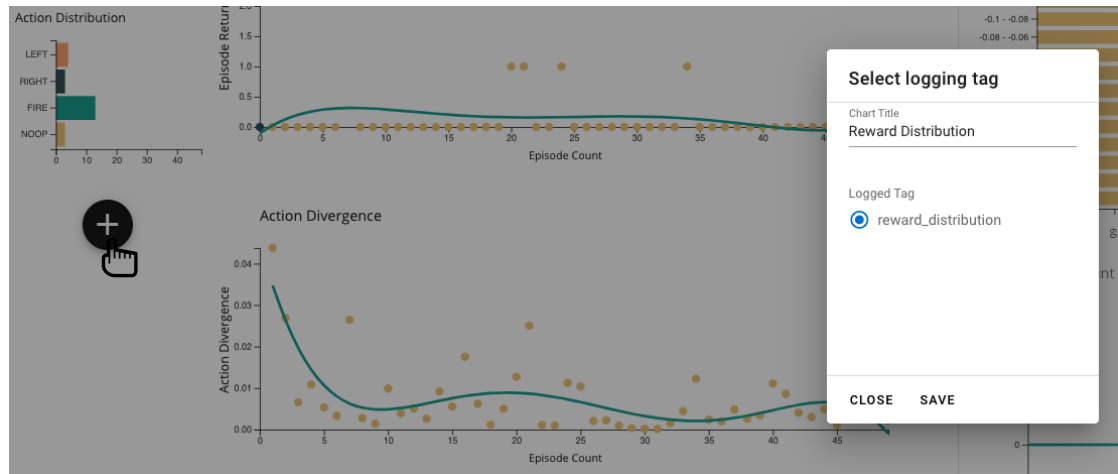


Figure 5.9: A visual illustration of the workings of additional chart inclusion. By clicking on the button a dialog appears. In this example we want to add a Reward Distribution Chart under our Action Distribution Chart. The reward distributions were included in the logging process before. Otherwise they do not appear in the dialog.

5.3 Confidence View

By clicking on the *Confidence View* tab in Figure 5.1, users can switch from the Overview to the Confidence View. A screenshot of the Confidence View is depicted in Figure 5.2. Overall, it consists of a scatter plot on the left and a legend on the right. The main goal of the Confidence View is to depict how the confidence of an agent in selecting actions evolves over time and to cluster similar states. This will be further explained in the Confidence Chart section. However, the slider on the bottom can be used to navigate through different episodes and thus see how the confidence scatter plot changes over time AS2.

5.3.1 The Random State Experiment

To generate data points that will be then drawn to the Confidence Chart, one has to go through a logging routine. This routine is the so called random state experiment. In general, its goal is

to evaluate how the agent decides in selecting actions for random states inside the state space of the environment the agent interacts with. Therefore, an arbitrary amount of random state examples are passed to the agent and it has to decide which action to select for every example it receives. This is done by the agent through predicting values on how probable it is that it would
 5 select an action for the received state for every action inside the action space. The selected action is the one corresponding to the highest predicted value.

For logging purposes, one has to differentiate between working with image or non-image data. When working with non-image data, the random state samples are generated completely at
 10 random, uniformly before passing them to the agent. When working with image data, the agent is required to have some kind of buffer, like e.g. the replay memory in DQNs [24], where it stores already encountered observed states, as randomly sampling pixels would not result in sensible images. Example states are then sampled from this buffer and passed as input to the agent's underlying neural network. To facilitate a two-dimensional visualisation of the random states,
 15 UMAP is applied on the random samples as a dimensionality reduction technique [20]. Following, the selected action, predicted probabilities and the reduction of the random state samples are saved to file. When handling lower-dimensional state data (non-image data) this data is also logged. Otherwise, images get logged separately to enable later visualisation of random state sample image frames. To measure the confidence of an agent in selecting one action over the
 20 rest, the entropy on every action prediction per state sample is calculated and subtracted from one [30].

5.3.2 Confidence Chart

The *Confidence Chart* and its legend come in two different formats depending on the underlying type of data they are processing. When handling image data as input random states for the
 25 agent, the legend consists of n bars where $n = |\mathcal{A}|$ and \mathcal{A} is the action space. The bar's colors indicate the action an agent selected and their color's opacity indicates the confidence with which the agent selected this action. Within the scatter plot of the confidence chart, one can see the dimensional reduced data points for each random state in the random state experiment. A screenshot of the scatter plot can be seen in [Figure 5.10]. Annotated by the legend bars, each
 30 data point has a color which corresponds to the action that was selected for this particular state and a varying opacity, which indicates the agent's confidence for selecting this action for this state. High opacity, therefore, means high confidence and low opacity means low confidence. Hovering over a circle in the scatter plot shows a Tooltip, which depicts the actual magnitude of confidence for selecting the action, indicated by the circle's color. When working with image
 35 data, the observation for which the agent selected this action is depicted inside the Tooltip, as well.

In case users train agents that are not working with image data and want to use the confidence chart to analyze them, the confidence chart differs in some minor ways. First, the legend on the right gets extended to also show the maximum and minimum values that are possible inside the

5 VISUALISATION

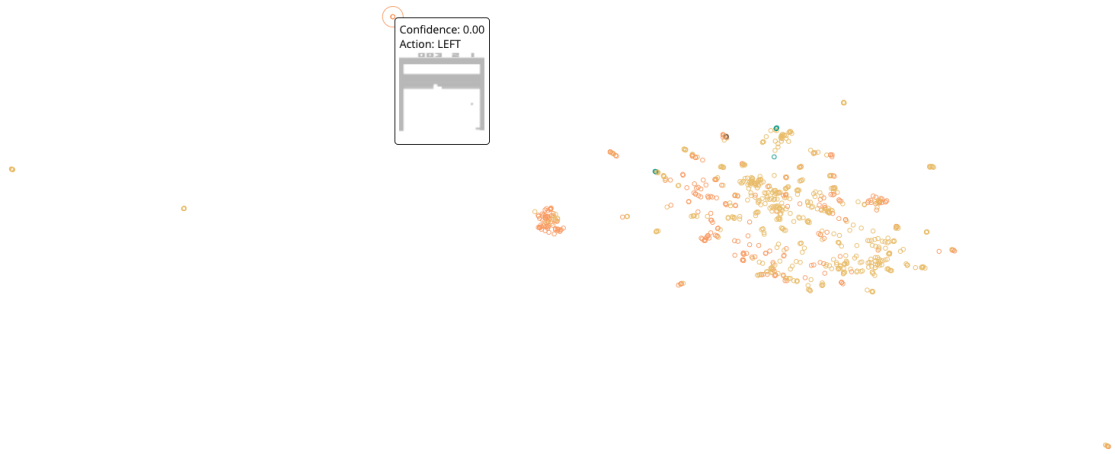


Figure 5.10: The confidence chart inside the confidence view. Each data point corresponds to one random state sample. Through hovering over a sample, one can get information about the sample, the agent's action and its confidence for selecting this action for this sample. The color also indicates the action.

state space. This is important to better be able to interpret the random state values, the agent had to reason actions for. Second, these state values are enlisted within the Tooltip from before instead of an image being depicted.

- 5 Overall, users can use the confidence chart to analyse an agent's confidence in selecting actions for some randomly sampled states. By also giving users state values or the agent's observed images, they can classify whether the agent is right with its assumptions on selecting certain actions in certain states. Also, by giving users the handles to navigate through multiple episodes, users can see how the agent evolves in means of confidently selecting actions for
- 10 states over time **AS2**.

In this chapter, the visualisation concept was introduced and frontend components were explained in detail. The main guideline of visualisations in DRLVis is to provide users with an overview and on demand detail information. For this, episodes and time steps inside episodes

15 were used as two levels of abstraction. To also further be able to analyze the confidence of an agent, DRLVis comes with a Confidence View where users can observe the evolution of an agent's confidence over time. For all visualisations, a possible interpretation was included to further guide users in their visual analysis.

6 Technical Realisation

As defined earlier, DRLVis should be easy-to-use and also easy to integrate within a users DRL pipeline **TS3** and guide users throughout the logging and visualisation process **AS4**. Therefore, it is important to use a technical realisation, which is easy to integrate within ones implementation but also familiar to users. Because of this, in this work, it was decided to develop a web application, which works with a frontend and a backend. Logging is also included into the process of using DRLVis to provide users with a cohesive pipeline when using DRLVis. Logging is also used for enabling users to include data that is interesting to them and exclude data that is not **TS2**.

6.1 Architecture

With the decision to develop a web application, an architecture consisting of a frontend and a backend was developed for DRLVis. The frontend handles the visualisations while the backend is necessary for processing log files and retrieving data for the frontend. **Figure 6.1** summarizes the developed architecture and shows the different frameworks that were used. As one can see, the frontend is implemented with Vue.js. The different parts of the frontend were divided into multiple components, where every component fulfills a different task. As an example, one can look into code snippet **6.1** which illustrates the main structure of a component built with Vue.js. This structure was adapted and used across the whole web application. For a consistent state management, Vuex was used, which will be further described in later sections. Vuex enables consistent sharing and persisting of data through the entire frontend, like e.g. committing the currently *selected episode* to other components, relying on this information.

The communication between backend and frontend is performed with HTTP requests between Vue.js components in the frontend and a flask server in the backend. From there, the data is processed by a *data preprocessor*, which returns requested data. Following this data preprocessing, the flask server handles the reply back to the frontend.

```
1  @Component
2  export default class CustomComponent extends Vue {
3    name: "CustomComponent"
4
5    @stateDecorator.State
6    public globalVariable: type;
7
```

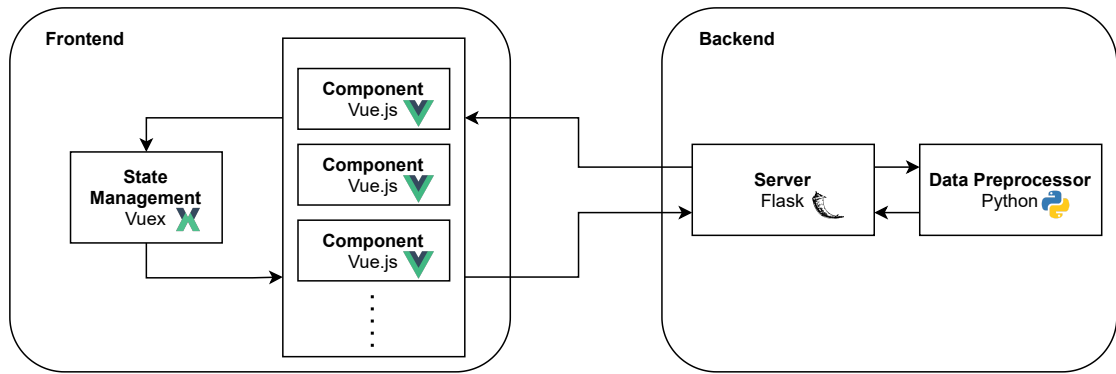


Figure 6.1: A diagram containing the architecture of DRLVis. It is mainly divided into frontend and backend. The frontend is separated into different components which interact with the backend. The backend consists of a server and a data preprocessor, which rearranges logged data before passing it to the server and finally to the frontend. Every part of the visualisation system in this figure contains an annotation about which programming language/framework was used for its development.

```

8     @stateDecorator.Mutation
9     public globalMutation: (parameter: type) => returnType;
10
11     @Watch("variableToWatch")
5 12     public handleVariableToWatchChanged(newVariableToWatch:
        typeOfVariableToWatch): void {
13         ...
14     }
15
16     public requestDataFromBackend() {
17         ...
18     }
19
20     beforeMount() {...}
15 21     mounted() {...}
22     computed() {...}
23     created() {...}
24     .
25     .
20 26     .
27
28 }

```

Listing 6.1: The code structure of a Component created with Vue.js. This can essentially be adapted to any component and extended individually. In lines 20-23, example Vue.js-specific life cycle hooks are enlisted [2].

6.2 Frameworks

For an easy transition between concept creation and implementation, multiple frameworks to develop the aforementioned architecture were used. These will be briefly summarized in the following sections.

5 6.2.1 Vue.js/Vuex

Vue.js is an open-source JavaScript framework, which was first released in February of 2014. Since then, it was completely maintained on an open-source basis and evolved over multiple iterations. It is suitable for creating any kind of web application and due to its component-based structure it can handle highly complex systems. For state management across an entire web application Vue.js comes with a state management pattern and library, called Vuex. Its main advantage is that it allows the consistent use of data on a global level through the entire web application. For routing purposes, the *vue-router* was used.

6.2.2 D3.js

D3.js is one of the most used libraries for data visualisation across the field. At its core it is a JavaScript library for directly manipulating DOM-elements and it was open-sourced under the BSD license with the main contributor being Mike Bostock. In this work, it is used for generating data visualisations. For doing so, it provides the possibility to create SVG-elements and generate any kind of visualisation inside just these. D3.js should not be understood as a boilerplate framework for generating data visualisations but rather a toolbox for doing so.

20 6.2.3 Flask

Flask is an open-source Python framework for generating WSGI web applications and has grown to be one of the most used frameworks for doing so. It was first released in April 2010 by Armin Ronacher. It was steadily improved since then and currently is in its second version. For the application, proposed in this work, it is used as an interface between frontend and backend. It can handle simple to complex HTTP requests, which suffices for the use in this work.

6.2.4 Tensorflow and Tensorboard

Tensorflow is one of the best-known and most used libraries for generating deep learning models, but also handling many other tasks in the machine learning field [4]. It was released by the Google Brain Team in November 2015 and is currently in its second version. In this work, Tensorflow's logging functions, which are used for visualisation within Tensorboard, a dashboard-like visualisation tool [3], are being exploited. Tensorflow is also used in this work for processing the logged data and for doing so, it is highly relied on the inner workings of the Tensorboard backend.

6.3 Logging

To handle data logging in an easy way, DRLVis comes with a logging functionality [\(AS4\)](#). This not only gives users the opportunity to save time for establishing data logging from scratch, but also ensures a clean and structured way of storing data, which is visualised later. For efficiency and simplicity reasons, this work relied on Tensorflow's summary logging library [\[4\]](#). It provides easy ways for saving data in default formats, like scalars, images or graphs. Also, it comes with the option of logging data in arbitrary data formats through its `tensorflow.summary.write` function. Thanks to this possibility, any data relevant for DRL visualisation could be logged and users could be given the possibility to log custom data for different use cases.

Functions inside the logging framework of DRLVis can essentially be categorized into two kinds of logging. The first kind includes logging functions for data points that were found interesting in the development process. These include values, which were mentioned earlier in the requirements analysis. Specifically, users are given functions for logging *episode returns*, *action divergences*, *action distributions* and *action probabilities*. As already mentioned, these kinds of data each have their own logging function inside of the logging framework of DRLVis and have to be logged in the logging process. To provide an example, listing 6.2 shows an example implementation of one function used in the first category of logging functions. It logs the return that is collected by an agent through an entire episode and treats this value as a scalar with the episode count as step. This will later be used by the backend to retrieve the correct data points and return them to the frontend. Additional default values to the ones before, which are highly recommended to be used but not required are *frame images*, *weights*, *confidence experiment data*, *action meanings* and *rewards*. All of these values also have functions for logging them. An exception are the rewards, which have to be logged with functions from category 2 that will be introduced in the next paragraph. If any of these values are not interesting for users, they can exclude them by simply not logging them in the first place, which makes the visualisation system individual and modular [\(TS2\)](#).

```

1  import tensorflow as tf
2
3  def log_episode_return(episode_return, episode_count):
4      """log the return/score/accumulated reward per episode
5      Params:
6          episode_return: int
7          A scalar value --> the return/score/accumulated reward
8          episode_count: int
9          This should regularly be the episode in which the return is
              logged
10     """
11     tf.summary.scalar(name="episode-rewards", data=episode_return,
12                      step=episode_count, description=None)

```

Listing 6.2: This listing shows one example of possible logging functions for default logging values. It is held simple, as it mainly is a wrapper around the *tf.summary.scalar* functionality.

The second kind of logging functions are ones that support modality in means of adding information that suit users' needs **TS2**. These are functions that allow to log custom values in three different formats. The first format is an episodic scalar value. This is complementing the *episode returns* and *action divergences* from earlier. An example value, which could be logged this way is the *loss* of a model, used for learning to predict e.g. Q-Values in the Q-learning setting, averaged over an entire episode. The second format is a time step scalar collected throughout an episode at each step of interaction between an agent and an environment. This could e.g. be the received *reward* or the taken *action* at every time step. The third format is a custom distribution. Complementing the *action distribution* from before, distributions for other values were found to be a use case of specific interest. One could e.g. log the distribution of *rewards* over an episode. An implementation of the custom distribution logging function can be found in listing 6.3. First, the function generates value counts for each unique value of a logged list of scalars, e.g. the rewards collected in an episode. Then, it saves the data as a numpy-array with the *tf.summary.write* function.

```

1  import tensorflow as tf
2
3  def log_custom_distribution(distribution_data, custom_tag, episode_count):
4
5      """log the distribution of a custom value (e.g. selected actions,
6          earned rewards)
7          Params:
8              distribution_data: numpy.ndarray
9                  An array or a list of counts for the specified value per episode.
10                 Could for
11                 example be like: Rewards: [0, 5, 3, 5, 7] in an episode with 5
12                 timesteps, where the
13                 agent receives 0 reward in the first timestep, 5 in the second,
14                 ...
15                 episode_num: int
16                     the current episode count/number
17             """
18         distribution_data = np.array(distribution_data)
19         # returns unique values and corresponding value counts
20         values, value_counts = np.unique(distribution_data, return_counts=True)
21
22         metadata = summary_pb2.SummaryMetadata()
23         tensor_dict = {}
24         for val, val_c in zip(values, value_counts):
25             tensor_dict[val] = val_c

```

```

20     metadata.plugin_data.plugin_name = str(custom_tag)
21     metadata.data_class = summary_pb2.DATA_CLASS_TENSOR
22     tf.summary.write(tag=str(custom_tag), step=episode_count, metadata=
        metadata,
5  23         tensor=np.array(list(tensor_dict.items()))))

```

Listing 6.3: This listing shows a function for logging custom distributions over a list of values passed to the functions, e.g. rewards.

This section gave an overview of the logging functionality provided by DRLVis. Even though simple, the logging functions retrieve data in a formatted way and also guide users into possible directions. They also enable users to choose new directions in which kinds of data shall be visualised. The logging functions are very easy to use and integrate into users' implementations, as they are simple hooks, which can just be inserted without further workload **TS3**. They also allow logging of any statistics in the mentioned data formats and can therefore be applied to multiple different DRL settings **TS1**. The example listings from above are not representative for the whole logging framework. Further details can be found in the implementation itself¹

6.4 Backend Preprocessing

After data has been logged, it has to be processed for further use in visualisations. Therefore, the backend comes with a data preprocessing framework, which retrieves and formats logged data and a server, which functions as an interface between the data preprocessor and the frontend.

The basic concept of the backend is the following. Every request made by the frontend is directly sent to the data preprocessor. By retrieving demanded data and processing it into a form of use for the frontend, the data preprocessor creates a Python dictionary including the requested data. If requested data does not exist, because it was not logged or because of an error, the data preprocessor returns an empty dictionary and prints an error on the console. For retrieving data, saved throughout the logging process, the data preprocessor relies on tensorboard's backend event processing library. Further information on that can be found on tensorboard's github page [\[3\]](#). Listing [6.4](#) shows the implementation of a processing function in the data preprocessing framework. In this particular case, this function is used for retrieving scalar values and relevant information to them. First, the values are extracted from the log files. Following that, through *numpy.polyfit*, trendline data points are generated for the beforehand extracted scalar values. Trendline data points and scalar values are assembled into one item and collected inside a dictionary. The dictionary is then returned and sent back to the frontend for further visualisation purposes. This function could e.g. be used to retrieve logged episode returns, which were mentioned before.

```

35  1 def get_scalar_values_by_tag(self, tag):

```

¹The implementation DRLVis was published on <https://github.com/masirt/drlvis>.

```

2      """A method to get scalar values by one single tag.
3      Params:
4          tag: string
5          A simple string containing the tag of which the values (step,
6          value) shall be returned
7      Returns:
8          scalar_listing:
9              The values {step: [val, polyfittrend]} filtered by the tag.
10     """
11     scalars = self._get_scalars_by_tag(tag)
12     if scalars == {}:
13         print("Scalar value queried by"+str(tag)+" does not exist.")
14         return {}
15
16     scalar_listing = {}
17     for scalar in scalars:
18         scalar_listing[scalar.step] = scalar.value
19     scalar_dict_vals = np.array(list(scalar_listing.values()))
20
21     polynomial_fit = np.polyfit(
22         np.arange(len(scalar_dict_vals)), scalar_dict_vals, 5)
23     poly_handler = np.poly1d(polynomial_fit)
24
25     for index, scalar in enumerate(scalars):
26         scalar_listing[scalar.step] = [
27             scalar.value, poly_handler(index)]
28
29     return scalar_listing

```

Listing 6.4: A listing that includes a sample data preprocessing function. This function returns scalar values for a specific log tag. This could e.g. be "episode-rewards" for the aforementioned episode returns.

30 This section gave an overview of the inner workings of the backend. It also provided an example listing to give an intuition about the code structure inside the data preprocessing framework.

Summing up, the architecture of DRLVis consists of a frontend and a backend. Furthermore, DRLVis comes with a framework to support data logging. Through the use of Tensorflow's summary API and tensorboard's functions to process data logged in this format, DRLVis processes
35 logged data in the backend and provides expressive visualisations in the frontend.

7 Case Study

To showcase DRLVis on real DRL task, this chapter includes a case study. The goal of the case study is to first, elaborate the workflow of this work's visualisation tool and to second, evaluate the interpretability of generated visualisations. For understanding the environment on which the agent in the case study is trained on and the algorithm which is used for training it, sections 7.1 and 7.2 of this chapter will cover details of the learning environment and the algorithm which navigates the agent through it. After that, it follows the case study for the use case of training a Deep Q Network to solve the CartPole problem [34]. It will focus on using DRLVis for evaluating training progress and reasoning about generated visualisations.

7.1 Learning Environment

For training the agent in the following sections, CartPole will be used as the environment the agent interacts with. CartPole is a common benchmark environment to test new DRL algorithms and implementations. The task of CartPole for an agent is to balance a pendulum on a cart. Figure 7.1 shows an example frame, which depicts the cart, which the agent can either push left or right and the pole which the agent needs to balance. The pendulum starts of in a vertical position and the goal is to keep it as straight as possible for the longest possible period of time without moving too much away from the center. An episode ends either when the pendulum bends more than 15 degrees from its vertical position or when the cart leaves the center by more than 2.4 units. The *states* that the agent receives at each time step consist of four different values, giving the agent information about cart and pole: *Cart Position*, *Cart Velocity*, *Pole Angle*, *Pole Velocity at Tip*. The only *actions* the agent can use for interacting with the environment are either moving the cart *left* or *right*, by applying force -1 or +1 to it. The *reward* an agent receives is 1 for every time step that it survives in the environment. As a simulator to interact with the environment, OpenAI Gym was used [7]. OpenAI Gym is a toolbox which contains a plethora of environments that can are used for research or simply implementing algorithms in agent-environment-interaction learning.

7.2 Algorithm

The original DQN from Mnih et al. [24] was trained on different Atari environments. The agent therefore observed image frames as its input and thus the underlying architecture was the Convolutional Neural Network depicted in Figure 2.3. As the case study will focus on a simpler

7 CASE STUDY

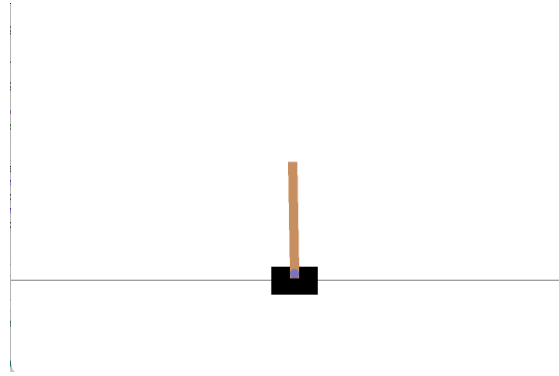


Figure 7.1: An illustrative example on how the environment that the agent will be interacting with in the following case study looks like. The agent interacts with the environment by pushing the black bottom cart either with force +1 or -1.

environment, CartPole, which does not process image data but scalar state values, the architecture in the case study is a simple feedforward Neural Network similar to the one shown in [Figure 2.2](#).

5 Algorithm [1](#) shows the original DQN algorithm that was introduced by Mnih et al. [\[24\]](#). This algorithm will be used for the case study. First, the algorithm initialises a replay memory, an action-value function and a target action-value function. The replay memory is used for storing experiences that the agent collects over time. It was introduced as one of the major contributions of the DQN algorithm, because it improves training in multiple ways. The most important advantage of using a replay memory is that when training the agent, one samples the training examples randomly from the replay memory. Different to e.g. supervised learning, underlying training data distributions may change over time in RL. Therefore, learning directly from successively collected data examples can be inefficient. That is why randomly sampling from experienced ones can alleviate the issue of correlations between training samples and thus lead to more stable training.

10 The action-value function is a Neural Network, the DQN, that is used as a function approximator for the action-value function. The target action-value function is a copy of the main DQN that is generated after a fixed number of steps. It is used for generating target values that are needed in the learning step of the algorithm. By using a separate DQN for doing so instead of the main DQN itself, one can again stabilize training and avoid divergence.

15

20

Second, the algorithm preprocesses the input images, which is not of greater importance, as the case study will not rely on image inputs. Third, the agent selects an action based on an epsilon-greedy policy [\[23\]](#) and interacts through it with the environment. It then receives a new state and a reward as a result and stores observed experience in the replay memory.

25

At last, the algorithm gets into the learning phase, where the agent randomly samples collected experience from the replay memory, generates target values based upon these and performs a

gradient descent update step with respect to the parameters θ of the main DQN. This routine of interacting, storing and learning is repeated for multiple steps t throughout multiple *episodes*.

Algorithm 1. The original Deep Q Learning algorithm from Mnih et al. [24].

```

1 Initialize replay memory  $\mathcal{D}$  to capacity  $N$ ;
2 Initialize action-value function  $Q$  with random weights  $\theta$ ;
3 Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$ ;
4 for  $episode=1, M$  do
5   Initialise sequence  $s_1 = x_1$  and preprocessed sequence  $\phi = \phi(s_1)$ ;
6   for  $t=1, T$  do
7     With probability  $\epsilon$  select random action  $a_t$ ;
8     otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$ ;
9     Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ ;
10    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ ;
11    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ ;
12    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ ;
13    if episode terminates at step  $j+1$  then
14      | Set  $y_j = r_j$ ;
15    else
16      | Set  $y_j = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$ ;
17    end
18    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network
      parameters  $\theta$ ;
19    Every  $C$  steps reset  $\hat{Q} = Q$ 
20  end
21 end

```

7.3 Use Case: Training a Deep Q Network

Susan is a Data Scientist and new to the field of DRL. She is familiar with deep learning frame-
 5 works like Tensorflow and also has knowledge in the Deep Learning Field. She is trying to
 explore possibilities of using DRL for in her every-day job. For doing so, she wants to first
 focus on studying different DRL algorithms. Therefore, Susan implements the aforementioned
 Deep Q Learning algorithm. With it, she wants to solve the cart-pole task through training the
 algorithm's DQN for approximately 1000 episodes. After that Susan decides to use DRLVis to
 10 further investigate what her agent did during training and to hopefully reason about different
 behaviours.

To use visualisations inside DRLVis for interpretation, Susan already included logging hooks
 into her implementation. She logged every value that either comes as default with DRLVis or is
 15 recommended to be logged. Also, she was interested in the evolution of the loss over multiple
 episodes and thus she logged the loss as a custom episodic value, which can be found in line 18

7 CASE STUDY

from Algorithm 1.

Susan trains her DQN for 1000 episodes. As soon as training is complete, she starts with her visual analysis using DRLVis. She begins by studying the overview window, where she first analyses the episode info panel. There she sees the distribution over actions the agent selected throughout an episode. As the cart-pole problem is a balancing task, the goal is to keep the pole in the middle of the cart. As Susan believes that the best way doing so would be to toggle between the left and right actions, she hopes to mostly find balanced distributions over the two actions. After plotting the action distribution plot for multiple episodes, Susan finds that in episodes in the beginning of the training process, most often the two action bars are slightly unbalanced, meaning that one bar is a little wider than the other. Figure 7.2 shows an example screenshot of the action distribution plot quite early in training.

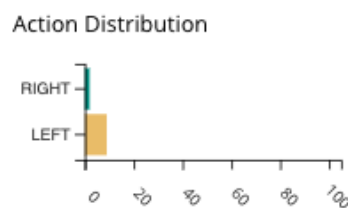


Figure 7.2: The action distribution chart observed early in training. The bars have a slightly different height horizontally which indicates a missing balance between selected actions throughout the episode.

One can see that the bar heights are not differing largely but are not balanced perfectly either. Susan also observes, that bars in later episodes quite often show better balanced distributions over the two actions. She now wants to bring this insight into context with the episode charts. At first glance, Susan can already see that her agent performed decently over time. One can see the two default episode charts that Susan observed in Figure 7.3. She now wants to prove her hypothesis that the cart-pole task's success partly relies on toggling between the two actions and thus having a balanced action distribution. After investigating the action distribution plot over multiple episodes and connecting this investigation with looking at the episode return chart, Susan finds that the agent had a very balanced distribution over the possible actions when retrieving high episode returns. This makes Susan confident, that her hypothesis about balancing tasks and toggling actions is correct.

While observing the two episode charts, Susan also finds that the data points in the action divergence chart in some situations negatively correlate with the episode return data points in the episode return chart. She wants to exploit this observation for finding outlying data points in the action divergence chart and further analysing the corresponding episodes in the Timestep View. Susan finds multiple outliers in the action divergence chart, where the data point action divergence value in episode n was much higher than the one in episode $n - 1$. She further investigated multiple outliers in the timestep view by looking at the video frame and similarly

7.3 USE CASE: TRAINING A DEEP Q NETWORK

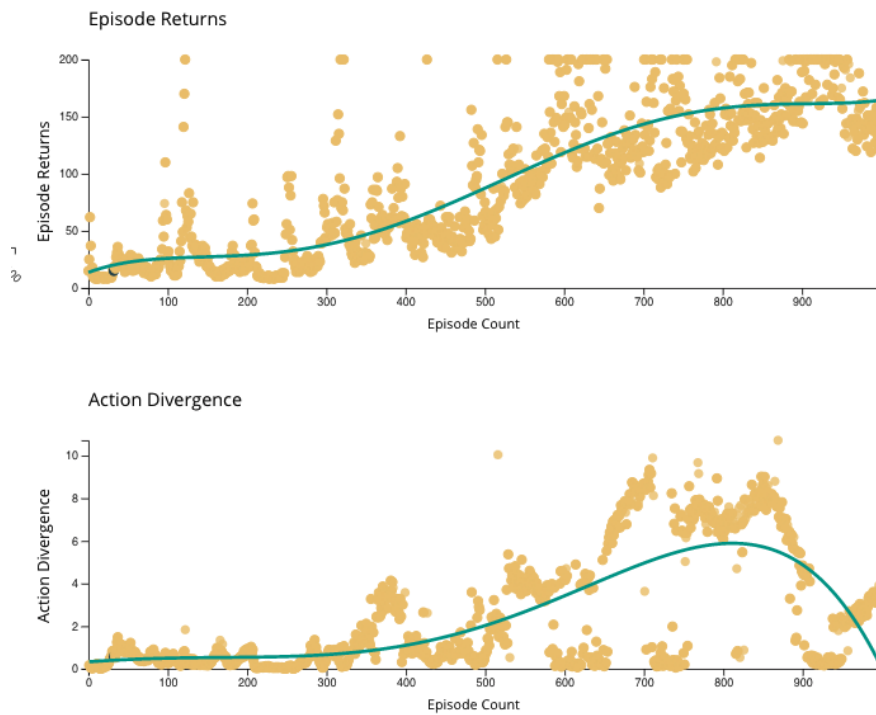


Figure 7.3: The two default episode charts, namely the episode return chart and the action divergence chart. One can see for some episodes that falling returns from one episode to another negatively correlate with rising action divergences.

7 CASE STUDY

observing the action probabilities chart. Susan finds that the agent seems to be biased in selecting to go right instead of left in many situations where going left would be preferable in these episodes. This behaviour leads to a suboptimal situation where it is difficult to balance the cart. She validates her thought through a look at the action distribution chart and sees that the action bars differ largely, with the one corresponding to the action *RIGHT* being larger than the bar for the action *LEFT*.

As a last analysis step in the Overview window, Susan wants to look into the customly logged loss over multiple episodes and connect this with the evolution of weights over time. She therefore includes the logged *loss per episode* by interacting with the Add-Button. A figure of the episode loss chart, which plots the loss for each episode can be seen in [Figure 7.4](#). By looking at it, Susan observes that the loss does not converge while training, which is good as early convergence would be suboptimal.

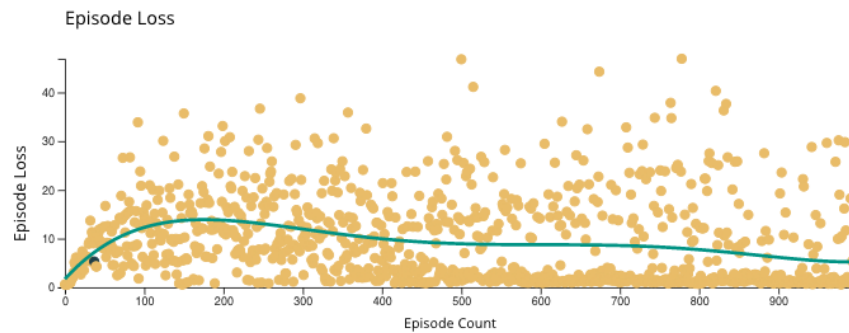


Figure 7.4: A modular Episode Loss Chart. One can see that the loss does not converge to an optimum, which is due to the nature of a DQN's loss function, one can find in [algorithm 1](#). However, the loss trend is going down as required.

To some point, an oscillating loss is expected for the DQN, as the target values it compares it predictions to change over time and shall avoid disoptimal convergence. Overall Susan still finds that the loss trend is going downwards, thanks to the trend line, which is desirable and shows that the agent is learning to some extent. Susan now wants to look at the weight evolution over time. [Figure 7.5](#) shows the evolution of the weight histogram of weights connecting nodes from the second last layer to the last layer in the neural network. One can see that the buckets of weights largely change over time and end up in two big buckets placing most of the weights values in the range -1.5 to -1 and 1 to 1.5. Susan sees that even though the weight distribution changes multiple times over the whole weight evolution, it still remains in a moderate range. She, therefore, implies that the learning rate that she used for training was neither too high nor too low. However, in some episodes she finds that the weight distribution changes more significantly, especially in the beginning of training. She thus considers using a lower learning rate in the beginning and applying some learning rate scheduling strategy.

7.3 USE CASE: TRAINING A DEEP Q NETWORK

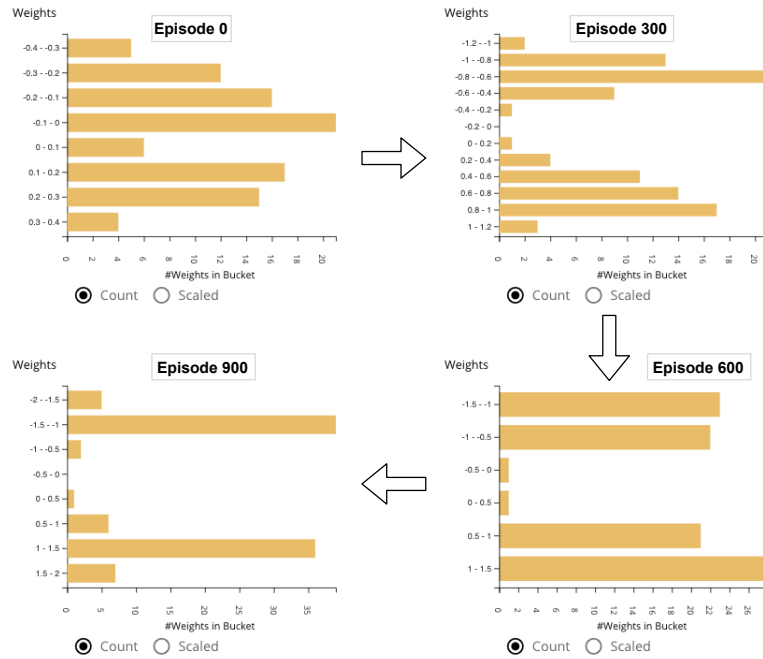


Figure 7.5: This figure shows four different snapshots of the weight histogram chart and therefore depicts the evolution of weights over time. One can see that the weights stay in a certain range and eventually accumulate in two buckets.

At last, Susan wants to see how the agent's confidence in selecting actions changes over time. She swaps over to the Confidence View to start her confidence analysis. Susan observes the confidence chart over time and finds that the agent's confidence evolves as expected. One can see an image illustration of this evolution in [Figure 7.6](#). In the beginning the differently colored data points in the confidence plot overlap quite often and the opacity of the data points is mostly quite low. This shows that the agent is not yet confident in selecting actions. It also shows that the agent has not yet learned a clear representation of the state space as the Confidence Chart places similar states next to one another and an agent should be able to separate dissimilar states in its action selection. Over time, the agent learns a better representation of the state space and also gets more confident in selecting its actions. Susan observes this evolution in the Confidence Chart and is quite sure that her agent learned well throughout training.

In this chapter, first a learning environment was introduced for this case study, namely OpenAI Gym's CartPole environment. Then, the Deep Q Learning algorithm was explained in detail and differences to the DQN implementation used in the case study were annotated. In the last section DRLVis was used to analyse a DRL training process visually. There, several observations were made and further interpreted.

7 CASE STUDY

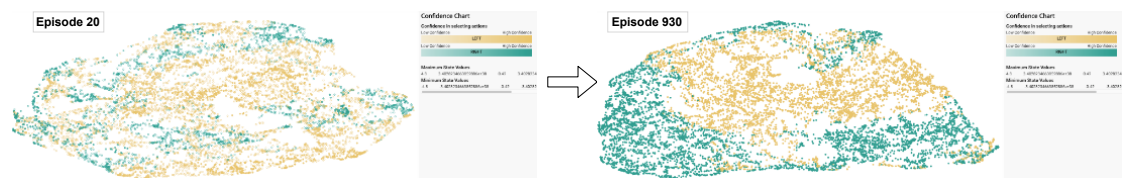


Figure 7.6: One can see the Confidence Chart in two different episodes. The first, on the left-hand side is quite early in training and shows an agent unconfident in selecting action. In the second chart (right-hand side), late in training, one can assume a confident underlying agent.

8 Conclusion

In this work, a new visualisation system for DRL, DRLVis, was presented. This work's focus is on creating meaningful charts that visualise different metrics about the DRL training process and give users handles to further investigate agent behaviours. Visualised metrics were connected to visual images of the agent performing in the environment. Also, this work presented a novel approach on analyzing an agent's confidence in selecting actions over time. In the following, major limitations of the presented system will be discussed and brought into context with future research directions in the field of DRL visualisation.

8.1 Limitations

The first limitation of DRLVis is its dependency on the Tensorboard backend framework [3]. As Tensorboard was not designed to handle large amounts of image data, tests showed, that it does not scale well when the number of saved images increases drastically. This is especially problematic when using DRLVis for visualising DRL algorithms that take images as input data. This is first, because the aforementioned random state experiment produces a large overhead of image data when random states are images. Second, DRL algorithms working with image data most commonly take more time for training and therefore save more images throughout training, which then is not handled efficiently by the backend. Even though salience analysis gave great support in understanding agent behaviour, the second limitation of this work is that it does not include any visual salience analysis in its visualisations. Although this work focused on generating generally applicable charts, this comes with major drawbacks in the ability of creating expressive visualisations for specific problems. For instance, DRLViz was able to show results on how memory reduction influences an agent's learning behaviour [11]. Specific findings like this are setting-specific and thus require on setting-specific visualisations like a matrix of stacked memory vectors in DRLViz. This, however, is not possible with DRLVis due to its generality. At last, DRLVis came with the restriction of only handling discrete action spaces. This reduces possibilities of being generally applicable, as environments with continuous action spaces cannot be visualised with DRLVis. Nevertheless, continuous action spaces can be converted to discrete ones. This could then enable DRL training processes to be visualised with DRLVis even though relying on environments with a continuous action space.

8.2 Future Work

Technically, the first step to improve DRLVis would be to overcome the bottleneck of the Tensorboard backend by means of processing time efficiency. This is especially important for DRL algorithms that handle image data as input. Another tool-specific improvement would be to extend the current state of the confidence view. Currently, the data points in the confidence charts are colored in correspondence to selected actions for a state and their opacity is determined by the confidence in selecting just these actions. One could extend this framework for multiple other metrics in DRL. For example, one could colorize data points corresponding to the earned reward in the underlying state and determine the opacity by the magnitude of the expected reward for the state. Of course, there always is room for possible improvements, but the two mentioned ones should give an intuition on possible directions for further improving DRLVis.

With DRLVis, this work presented a visualisation tool for DRL training processes. DRLVis differs from other related work by means of easier use and easier understandable visualisation. It also supports users from the logging process up to visualisation of interesting metrics for the DRL field. Furthermore, DRLVis not only enables users to visualise different metrics but also comes with a video view for the agent's observations and the option to analyze an agent's confidence for certain situations over a period of time. However, there still is a limited amount of DRL visualisation systems, specifically for non-experts. As DRL steps up to an important role in real world scenarios as well as in society, approaches that can shed light on the black box of DRL are crucial for transparency and the future of DRL itself. PolicyExplainer, for instance, gives a good approach for possible future directions of explanatory systems for DRL agents [22]. Looking inside a DRL agent will become more important in the future and thus visual analytics system that support this idea, as well. With the same goal, but by following a different approach, more transparent models can also be a great part of the future of DRL interpretability. Like in the i-DQN introduced by Annasamy et al., including interpretability in the development process of new DRL algorithms and underlying Deep Learning architectures is worth further discovery in the future [5]. The advantage of this approach is that it forces researchers to let thoughts flow into the interpretability direction, already during DRL algorithm development.

List of Figures

	2.1	Categorization of RL into the landscape of Artificial Intelligence	4
	2.2	An example Feedforward Neural Network	5
	2.3	The DQN architecture	5
5	2.4	The agent-environment-interaction cycle	6
	3.1	Overfit vs Control Agent in the game of Pong	10
	3.2	The DQNViz Interface	12
	3.3	The DRLViz Interface	13
	3.4	The PolicyExplainer Interface	14
10	5.1	A screenshot of the entry-point DRLVis Interface	22
	5.2	A screenshot of the Confidence View	23
	5.3	A screenshot of the Action Distribution Chart	24
	5.4	A screenshot of the default Episode Charts	25
	5.5	A screenshot of the Action Probabilities Chart	26
15	5.6	A screenshot of the Weight Histogram	27
	5.7	A screenshot of the Reward Plot	28
	5.8	A screenshot of the Video View	29
	5.9	An illustration of modular chart inclusion options	30
	5.10	A screenshot of the Confidence Chart	32
20	6.1	A conceptual illustration of the DRLVis architecture	34
	7.1	A screenshot of the Cart Pole environment	42
	7.2	The Action Distribution Chart inside the case study	44
	7.3	The default Episode Charts inside the case study	45
	7.4	An Episode Loss Chart inside the case study	46
25	7.5	An illustration of weight evolution over time	47
	7.6	The Confidence Chart inside the case study	48

Listings

6.1	The Vue.js template for structuring a Component	33
6.2	An example logging function for default logging values	36
6.3	The logging function for saving custom distributions	37
5 6.4	A sample data preprocessing function	38

List of Algorithms

1	The original Deep Q Learning algorithm from Mnih et al. [24].	43
---	---	----

Bibliography

- [1] "Drlviz demo (vizdoom scenario as in the paper), openly available," <https://sical.github.io/drlviz/main.html?scenario=3>, Accessed: 2021-05-16.
- 5 [2] "Lifecycle hooks | vue.js," <https://v3.vuejs.org/api/options-lifecycle-hooks.html>, Accessed: 2021-07-25.
- [3] "Tensorboard on github," <https://github.com/tensorflow/tensorboard>, Accessed: 2021-07-28.
- [4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, 10 C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- 15 [5] R. M. Annasamy and K. Sycara, "Towards better interpretability in deep q-networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4561–4569.
- [6] A. Bäuerle, C. Van Onzenoodt, and T. Ropinski, "Net2vis—a visual grammar for automatically generating publication-tailored cnn architecture visualizations," *IEEE transactions on visualization and computer graphics*, vol. 27, no. 6, pp. 2980–2991, 2021.
- 20 [7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
- [8] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," in *Situational awareness*. Routledge, 2017, pp. 9–42.
- 25 [9] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1792–1801.
- [10] B. Hayes and J. A. Shah, "Improving robot controller transparency through autonomous policy explanation," in *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2017, pp. 303–312.

BIBLIOGRAPHY

- [11] T. Jaunet, R. Vuillemot, and C. Wolf, "Drviz: Understanding decisions and memory in deep reinforcement learning," in *Computer Graphics Forum*, vol. 39, no. 3. Wiley Online Library, 2020, pp. 49–61.
- [12] P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim, "The (un)reliability of saliency methods," 2017.
- [13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
- [14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
- [15] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.
- [16] Y. Li, "Deep reinforcement learning," 2018.
- [17] B. Y. Lim, A. K. Dey, and D. Avrahami, "Why and why not explanations improve the intelligibility of context-aware intelligent systems," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 2119–2128.
- [18] L.-J. Lin, *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.
- [19] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 91–100, 2016.
- [20] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.
- [21] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding hidden memories of recurrent neural networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2017, pp. 13–24.
- [22] A. Mishra, U. Soni, J. Huang, and C. Bryan, "Why? why not? when? visual explanations of agent behavior in reinforcement learning," *arXiv preprint arXiv:2104.02818*, 2021.
- [23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

- [25] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, and D. J. Rezende, "Towards interpretable reinforcement learning using attention augmented agents," *arXiv preprint arXiv:1906.02500*, 2019.
- [26] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.
- [27] T. Ropinski, "Lecture notes in deep learning for graphics from winter term 2019," October 2019.
- [28] E. Saldanha, B. Praggastis, T. Billow, and D. L. Arendt, "ReLVis: Visual Analytics for Situational Awareness During Reinforcement Learning Experimentation," in *EuroVis 2019 - Short Papers*, J. Johansson, F. Sadlo, and G. E. Marai, Eds. The Eurographics Association, 2019.
- [29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.
- [30] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.
- [31] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*, ser. Interactive Technologies, B. B. BEDERSON and B. SHNEIDERMAN, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 364–371. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9781558609150500469>
- [32] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [33] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2014.
- [34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [35] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.
- [36] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [37] J. Wang, L. Gou, H.-W. Shen, and H. Yang, "Dqnviz: A visual analytics approach to understand deep q-networks," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 288–298, 2018.

BIBLIOGRAPHY


- [38] J. Wang, L. Gou, H. Yang, and H.-W. Shen, "Ganviz: A visual analytics approach to understand the adversarial game," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 6, pp. 1905–1917, 2018.
- [39] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.
- 5 [40] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, and S. Liu, "A survey of visual analytics techniques for machine learning," *Computational Visual Media*, pp. 1–34, 2020.
- [41] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1899–1908.

Declaration

I, Marios Sirtmatsis, matriculation number 1020254, hereby declare that I created this work on my own. Except where referenced and credited to the original author, I have not used the material of others in my work.

5

Ulm,



Marios Sirtmatsis