AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

INSTYTUT INFORMATYKI

Praca dyplomowa

System for explaining actions of cloud resource management policies trained with use of Deep Reinforcement Learning

System do objaśniania akcji polityk systemu zarządzania zasobami chmurowymi z wykorzystaniem trenowania przez głębokie uczenie ze wzmocnieniem

Autor:	*Andrzej Małota*
Kierunek studiów:	Informatyka
Opiekun pracy:	*dr. inż. Włodzimierz Funika*

Kraków, 2022

Abstract

Cloud computing resources management is a critical component of the modern cloud computing platform, which is one of the most important IT services developed in recent years as most online applications are dependent upon its resources. The objective of cloud resource management is to optimally manage computing resources for the application in a way that minimizes the cost of the infrastructure for the user while maintaining the high Quality-of-Service (QoS). This task is often solved using rule-based policies but more robust and complex solutions such as Deep Reinforcement Learning (DRL) agents are being heavily researched or already used in production. As the agent's decisions have a huge financial impact on both the user and the Cloud Services Provider (CSP) it is very important to understand the reasoning behind the policy decisions to achieve higher trust, transparency, and dependability of the system. This thesis provides global and local interpretability by applying the eXplainable Artificial Intelligence (XAI) algorithm - Integrated Gradients (IG) to the deep learning model upon which the DRL agent is built to produce feature s that show the magnitude and the direction of the feature's influence on the agent's decision. Thanks to the statistical analysis of the DRL agents, we were able to differentiate between the proactive and reactive agents. A reactive agent reacted with delay to the increase in the workload. On the other hand, proactive agents were able to predict the spikes and provide the necessary computing resources ahead of time. The main analysis based on the feature attributions was very fruitful, firstly, we were able to create agent policy summarization, by calculating the absolute mean attributions or positive/negative mean attributions we could showcase the most important features for each agent. Secondly, feature attributions empowered us to analyze the learning process of the agents by looking at the feature attributions changes between the consecutive iterations of the agent. Local interpretability meant that we could debug the agent's prediction, analyze why the agent made the wrong decisions, or even find examples where the agent made the right decision but based it on the wrong reasons. In the last step of the analysis, we successfully performed feature selection by removing from the environment the feature, that had a small impact on the agent's decision and showed that there was no performance drop in the agent trained without that feature while making the training faster and the agent simpler and easier to interpret.

Streszczenie

Zarządzanie zasobami chmury obliczeniowej jest krytycznym elementem nowoczesnej platformy obliczeniowej, która jest jedną z najważniejszych usług IT stworzonych w ostatnich latach, ponieważ większość aplikacji internetowych działa dzięki jej zasobom. Celem zarządzania zasobami chmury jest optymalne zarządzanie zasobami obliczeniowymi aplikacji w taki sposób, aby zminimalizować koszt infrastruktury dla użytkownika przy zachowaniu wysokiej jakości usług. To zadanie jest często rozwiązywane za pomocą algorytmów opartych na regułach, ale niezawodne i złożone rozwiązania, takie jak agenci głębokiego uczenia ze wzmocnieniem, są intensywnie badane lub już używane w produkcji. Ponieważ decyzje agenta mają ogromny wpływ finansowy zarówno na użytkownika, jak i dostawcę usług w chmurze, bardzo ważne jest zrozumienie decyzji polityki agenta w celu osiągnięcia wyższego zaufania, przejrzystości i niezawodności systemu. Ta praca przedstawia globalną i lokalną interpretację poprzez zastosowanie algorytmu interpretującego sieci neuronowe — Integrated Gradients (IG) do agenta w celu wygenerowania atrybucji cech, które pokazują wielkość i kierunek wpływu cechy na decyzje agenta. Dzięki analizie statystycznej byliśmy w stanie rozróżnić pomiędzy agentami proaktywnymi i reaktywnymi. Agent reaktywny reagował z opóźnieniem na wzrost nakładu pracy. Z drugiej strony, proaktywni agenci byli w stanie przewidzieć skoki w nakładzie pracy i zapewnić niezbędne zasoby obliczeniowe z wyprzedzeniem. Główna analiza oparta na atrybucjach cech była bardzo owocna, po pierwsze, byliśmy w stanie stworzyć podsumowanie polityki agentów, obliczając bezwzględne średnie atrybucje lub dodatnie/ujemne średnie atrybucje, dzięki czemu mogliśmy przedstawić najważniejsze cechy dla każdego agenta. Po drugie, atrybucje cech umożliwiły nam analizę procesu uczenia się agentów poprzez obserwację zmian atrybucji cech pomiędzy kolejnymi iteracjami agenta. Lokalna interpretacja oznaczała, że mogliśmy analizować predykcje agenta, na przykład, dlaczego agent podjął złą decyzję, a nawet znaleźć przykłady, w których agent podjął właściwą decyzję, ale z niewłaściwych powodów. W ostatnim kroku analizy pomyślnie przeprowadziliśmy selekcję cech, usuwając ze środowiska cechę, która miała niewielki wpływ na decyzję agenta i wykazaliśmy, że jej brak nie spowodował spadku wydajności agenta, jednocześnie przyspieszając jego trening, oraz czyniąc go prostszym i łatwiejszym do zinterpretowania.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cloud computing provides access to computing resources when they are needed via the internet. The resources are managed by a Cloud Service Provider (CSP). These resources can be available for a subscription fee or billed according to their usage [56]. Virtualization is a fundamental technology of cloud computing, that allows multiple operating systems to be executed on the same physical instance and structures them into Virtual Machine (VM) [42]. VMs are used by CSP in order to provide: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS). There are numerous advantages of cloud computing, the most basic ones being lower costs, re-provisioning of resources and remote accessibility. Cloud computing is very flexible, we can quickly access more resources from cloud providers if we have a need to broaden our business. Remote accessibility gives us access anywhere in the world at any time [3]. On top of that, cloud computing removes the huge cost of buying complete software and hardware at the start.

Resource management in the cloud computing field is the technique that assigns available resources to the cloud applications that need them - over the internet. Resource management can, unfortunately, provide too much or too few resources for the services if the allocation is not managed precisely [3]. CSP provide an on-demand dynamic resource allocation model - auto-scaling. It can add or remove resources (e.g. Central Processing Unit (CPU), Random-

Access Memory (RAM)) to adapt to workload changes and ensure stable performance and resource cost. There are two types of auto-scaling techniques, vertical and horizontal scaling. The vertical scaling adds or removes resources within the VM - increasing its size. Horizontal scaling means adding or releasing instances of VMs [11, 16]. There are two auto-scaling approaches, reactive and proactive. Reactive methods auto-scale compute resources after detecting that there are not enough resources for an application, such as the rule-based method proposed in [52, 53]. Proactive methods can predict the workload of the application and scale the resources based on the predictions. Older proactive methods such as SmartScale [11] used polynomial regression to predict resource requirements. Nowadays, the best proactive methods are based on ML algorithms - deep learning and reinforcement learning.

Learning from the interaction is a foundational idea underlying nearly all theories of learning and intelligence [51]. Learning from the interaction is inspired by human and animal behavior. RL is a formalized approach to learning from interaction through computation where a learning agent (e.g., human, animal, robot) tries to achieve a goal by interacting with its surrounding environment, receiving positive or negative rewards during the process [33]. During a training process, RL agents learn to map situations-observations into actions. Upon executing an action, the agent receives a reward signal. The objective of the agent is to achieve a target through receiving the highest rewards for its actions which indirectly leads to the end goal. What is interesting, the agent must discover which actions to take on its own by trying them out and finding out which return the biggest rewards [51].

DRL is one of the subfields of Machine Learning (ML) that combines RL and Deep Learning (DL) - with Deep Neural Network (DNN) as function approximators. In recent years it has attracted much attention and has been successfully applied to play Atari games based on raw pixels [34], [35], in history-making achievements in Games such as Go [49], Starcraft [57] and Robotics such as solving Rubik's cube [38] or walking/running [59]. The more practical, real-world applications are in Natural Language Processing (NLP) in human-machine dialogue [8], traffic signal control [18] or cloud resource management [60] - presented in this Thesis.

Many ML algorithms and especially DNN suffer from a lack of explainability. The effort to explain these networks and many other ML algorithms, before using them to solve real-world problems where dependability and trust are essential has become known as XAI. XAI provides tools that allow humans to understand the decision-making reasoning of Artificial Intelligence (AI) systems. Meantime there is a growing trend in government regulations that force companies to provide an explanation for their algorithm's decisions. The European Union introduced a right to explanation in the General Data Protection Right (GDPR) but it covers only the local aspect of interpretability. In the United States, insurance companies are required to be able to explain their rate and coverage decisions.[26]

While explainability starts being well developed for standard ML models and neural networks [41], [46], RL has still many issues that need to be resolved. It is still unclear as to how these complicated algorithms function and also how to propagate the reasoning of the RL model to the users. Without that, RL models will not be applied to the fields such as defense, finance, or medicine where it is imperative for the human system maintainers and end users to understand its decisions [23].

Explainable Reinforcement Learning (XRL) is a recently new subfield of XAI which has received a lot of attention. The goal of XRL is to clear up the reasoning behind DRL agents [32]. DRL models are very complex models which makes them difficult to debug for potential developers. An explainable model should help fix problems with: the design of the environment - especially the reward function, and the design of the model architecture and speed up the whole development [23]. To the author's best knowledge, no academic paper has used XAI techniques to interpret DRL agents in the cloud resource management setting.

1.2 Problem Definition

The success of DRL in research opens the way for its usage in the industrial world but with that, there will be a lot of new responsibilities attached. Cloud resource management is a real-world problem, with big financial consequences for both the cloud owner as well as the customers. Here are the few things that will need to be addressed before society can possibly accept that the decision-making of the cloud resource management should be delegated to the DRL agent:

- fairness

  Both the owner and the customer must trust that the system is fair. The owner can sleep in peace knowing that the system is just which saves him from probably future legal issues and the customer knows he is not being cheated and for example, his applications are not being allocated unnecessary resources for the financial gain of the owner.

- dependability

  Probably the most important thing is that the system behaves optimally. There are a lot of metrics that can score the agent but by adding the XAI component we can even better evaluate its actions. For example, the system might be making the right decisions for the wrong reasons and we wouldn't even know.

- transparency

  The system has to be transparent in how it works. It will increase its credibility and its general decision-making can be monitored. For example, if a model's performance starts to diminish, by looking at the feature relevance we can quite quickly spot that the data shift happened and perform data engineering or retrain the agent.

This work will focus on explaining the actions of the cloud computing resource management policies trained with the use of Deep Reinforcement Learning, to be more precise, the DRL agent will dynamically manage virtualized computing resources as an IaaS provider using horizontal auto-scaling, which is used to create or delete VMs based on application requirement and resource utilization. Cloud computing resource management policy will concentrate on minimizing resource usage costs while maintaining a high Quality-of-Service (QoS). This work is an extension of the works done by our promoter in [13] and [14], by adding the agent's action interpretability to the already existing system.

1.3 Research Objective

This work will present and compare different DRL learning algorithms such as DQN [34] or PPO [45] with different policy architectures: MLP, CNN, Recurrent Neural Network (RNN) as well as different combinations between them used to manage cloud resources in a simulated environment. After the models are trained, they will be evaluated in a couple of experiment runs. Based on the data from experiments, we will perform post hoc explainability, that is calculate feature attributions (how the feature's value contributed to the agent's prediction) after the training and then perform a robust statistical analysis of the agent's predictions furthermore, using XRL techniques, precise analysis about the reasoning behind the agent decisions.

1.4 Content of this work

The chapter 2 provides information about related research papers. Chapter 3 presents theoretical background of this Thesis. It introduces DNN and explains its different components, providing an introduction to RL followed by DRL and XAI. In the chapter 4 we introduce the simulation environment and its training system architecture. Chapter 5 presents the proposed interpretability system design and explains the reasons behind the steps taken in the analysis. We present the experiment results and analysis in the chapter 6 and also present a summary and conclusion of the work done in the Thesis and provide directions for future work in chapter 7.

Chapter 2

Related Work

2.1 DRL in Cloud Resource Management

In recent years, the usage of DRL in cloud resource provisioning gained significant attention. In [58], authors explore the use of standard RL and DRL for cloud provisioning, where users specify rewards based on cost and performance to express goals, and the RL algorithm figures out how to achieve them [58]. As a baseline of their RL approaches, they used a tabular-based Q-learning algorithm [51] but due to its tabular nature, it is limited when deployed with continuous states. They also used DQN [34], with CNN with 1D convolutions as a function approximator, which is said to be good for the temporal type of input data. They used two different simulation frameworks for performing tests: a simplified Python environment for running very fast tests, and a more realistic environment based on CloudSim [48], which is a Java-based package that simulates IaaS [58]. Their policy based on DQN achieved better results than the threshold-based one.

Another great work - [6], focused on minimizing power consumption for CSP. They used Double Deep Q-Networks (DDQN) [22] which has faster and more stable training than vanilla DQN. The proposed DRL-based cloud resource provisioning and task scheduling consist of two stages, the first stage allocates the task to one of the F server farms. After allocating the task into server farm F_f, the second stage of the processor takes responsibility for choosing the exact server to run the task in. The energy cost of the performed action is used to calculate the reward function. The proposed DRL-Cloud achieves up to 320% energy cost efficiency improvement while maintaining a lower reject rate on average. For example CSP setup with 5, 000 servers and 200, 000 tasks, compared to a fast round-robin baseline, the proposed DRL-Cloud achieves up to 144% runtime reduction [6].

The authors of [54], used Asynchronous-Advantage-Actor-Critic (A3C) learning built with the use of Residual Recurrent Neural Network (R2N2) to manage the Edge-Cloud computing environment. They conducted experiments on real-world data set which showed a significant improvement in terms of energy consumption, response time, Service-Level-Agreement (SLA) and running cost by 14.4%, 7.74%, 31.9%, and 4.64%, respectively when compared to the state-of-the-art algorithms [54].

The starting point of this work is the two papers authored by our thesis supervisor and his colleagues - [13, 14]. In the first one [13], they compared three DRL policy gradient optimization methods: Vanilla Policy Gradient (VPG), PPO, and Trust Region Policy Optimization (TRPO), to create a policy used to control the behavior of an autonomous management agent. The agent is interacting with a simulated cloud computing environment, which is processing a stream of computing jobs. The environment state is represented by a set of metrics that are calculated in each step of the simulation. These metrics include: the number of running virtual machines, 90-th percentile of task queue wait time, 99-th percentile of task queue wait time, average CPU utilization, 90-th percentile of CPU utilization, total task queue wait time, wait queue size. The reward function is set up as the negative cost of running the infrastructure including the SLA penalties - 0.00001$ for every second of delay in task execution. The policy which achieved the lowest cost was created using the PPO algorithm [13]. The newer paper - [14], which is based on the findings of the previous one, compared the PPO-based autonomous management agent with the traditional autonomous management approach available in Amazon Web Services (AWS). As a workload to which automatic scaling policies were applied, they have chosen a simple evolutionary experiment (pytorch-dnn-evolution) which improves the architecture of a network that recognizes handwritten numbers (the Modified National Institute of Standards and Technology (MNIST) dataset [9]). They performed policy training inside the simulated cloud environment and then deployed the policy to real cloud infrastructure - the AWS Elastic Compute Cloud. Under the management of the PPO-trained policy, the overall experiment's total cost of managed resources was slightly lower (0.7%) than the threshold-based approach [14]. This approach has one very important advantage other than high performance and cost reduction of the infrastructure - there is no need to manually set threshold levels for each application. The trained policy can generalize well enough to be re-used across multiple similar workloads.

2.2 Explainable AI in Deep Reinforcement Learning

DRL has shown great success in solving various sequential decision-making problems, such as playing complicated games or controlling simulated and real-life robots. However, existing DRL agents make decisions in an opaque fashion, taking actions without accompanying explanations. This lack of transparency creates key barriers to establishing trust in an agent's policy and scrutinizing policy weakness. This issue significantly limits the applicability of DRL techniques in critical application fields such as finance, self-driving cars, or cloud resource management. [20]

So far, most of the research on the usage of XAI in DRL has focused on finding the relationship between the agent's action and the input observation at the specific time step - detecting the features that contributed the most to the agent's action at that specific time. It is possible to classify all recent studies into two main categories: transparent methods and post hoc explainability according to the XAI taxonomies in Barredo Arrieta et al. [4]. On the one hand, inherently transparent algorithms include by definition every algorithm which is understandable by itself, such as decision trees or linear regression. On the other hand, post hoc explainability includes all methods that provide explanations of an RL algorithm after its training, such as SHapley Additive exPlanations (SHAP) [29] or Integrated Gradients (IG) [50] [23]. This work will perform post hoc explainability, so that will be the focus for the rest of this section.

When dealing with images as input data, one can provide explanations through saliency maps. A saliency map is a heat map that highlights pixels that hold the most relevant information and the value for each pixel shows the magnitude of its contribution to the CNN's output. Saliency maps are very easily understandable by humans - it's their major advantage. However, they are not perfect - they are sensitive to input variations. The authors of [19] aim to fix that with their perturbation-based saliency method which they used to play in a few of OpenAI Gym's [5] Atari 2600 environment games. In their work, they focused on agents trained via the A3C algorithm. They conducted a series of investigative explorations into explaining Atari agents. First, they identified the key strategies of the three agents that exceed human baselines in their environments. Second, they visualized agents throughout training to see how their policies evolved. Third, they explored the use of saliency for detecting when an agent is earning high rewards for the "wrong reasons". This includes a demonstration that the saliency approach allows non-experts to detect such situations. Fourth, they found Atari games where trained agents performed poorly and used saliency to "debug" these agents by identifying the basis of their low-quality decisions [19].

Another excellent work [25], applied post hoc interpretability to the DRL agent which learned to play the video game CoinRun [7]. They used PPO [45] algorithm, with the agent's model being CNN. Attribution shows how the neurons affect each other - usually how the input of the network influences its output.

They apply dimensionality reduction techniques [37] to the input frames from the game and calculate attributions using IG. They built an interface for exploring the detected objects which shows the original image and also positive and negative attributions - which explain how the objects influence the agent's value function and policy. Their analysis consists of three main parts:

- Dissecting failure - Thorough analysis of the agent's behavior in situations where the achieved reward was not optimal in order to understand what did the agent do wrong and the reasons behind it.
- Hallucinations - They explored cases where the model "saw" a feature that actually wasn't there.
- Model editing - They manually edited model weights so that the agent couldn't see some threats. They verified the changes by exploring numerous examples and finding which threats made the agent fail.

They also formulated the diversity hypothesis - RL models become more interpretable as the environments they are trained on becoming more diverse, which they provide evidence for by measuring the relationship between interpretability and generalization. They concluded that the diversity may lead to interpretable features via generalization but in the end, they still considered the hypothesis to be highly unproven [25].

In [36], a team of researchers from DeepMind performed an interpretability analysis of the DRL agent with a recurrent attention model on the games from the Atari 2600 environment. Firstly, they observed basic attention patterns using Attention map produced by the attention model and found out that the model attends to task-relevant things in the frame - player, enemies, and score. The Atari environment is often quite predictable: enemies appear at regular times and in regular configurations. It is, therefore, important to ensure that the model truly learns to attend to the objects of interest and act upon the information, rather than memorize or react to certain patterns in the game. In order to test it, they injected an enemy object into the observation at an unexpected time and in an unexpected location. They observed that the agent correctly attends to and reacts to the new object. Another interesting thing that they discovered was that the model performs forward planning/scanning - it learns to scan through available paths starting from the player character, making sure there are no obstacles or enemies in the way. When it does see the enemy, another path is produced in order to avoid it. In the end, they compared their method to the other saliency analysis method introduced in [19] and showed that their method allows a more comprehensive analysis of the information the agent is using to inform its policy [36].

2.3 Summary

This chapter presented academic papers that are related to the topic of this Thesis. The starting point of this work is the two papers authored by our thesis supervisor and his colleagues - [13, 14] in which they experiment with multiple DRL algorithms to create a policy used to control the behavior of an autonomous cloud resource management agent in the simulated environment. The proposed papers however lacked any kind of explainability of the agent's decisions, which would be necessary for it to be used not only in the academic setting but also in practical applications. We found three research analyses where explainability techniques were successfully applied to the DRL agents playing a two-dimensional (2D) games, thus producing 2D saliency maps showing contributions of each game frame pixel to the agent's prediction. They provided foundations for our approach to explaining the actions of our DRL agents and explainability-based analysis.

Chapter 3

Theoretical background

This chapter will introduce theoretical aspects of the ML algorithms used in this work in order for the reader to be able to follow the path of thought and understand the presented solution. It will present concepts from the field of classical RL, introduce Artificial Neural Networks and a few of the architectures such as Feedforward Neural Network (FNN), CNN, and RNN) as they are used as function approximators in order to achieve DRL. The last section will introduce the field of Explainable AI.

3.1 Reinforcement Learning

3.1.1 RL Introduction

Reinforcement Learning aims to create intelligent agents which learn by performing a certain set of actions in the special environment, which in return provides positive or negative feedback - reward, to show the agent whether the action taken was good or bad in terms of achieving a specified goal. During training, the agent explores different actions and observes the rewards, it tries to maximize cumulative reward over the episode and changes its behavior to do so [51].

3.1.2 RL concepts

This subsection presents basic RL concepts. Building blocks such as environment, agent, episode, or RL problem's methodology and algorithms. Figure 3.1 presents the base RL workflow.

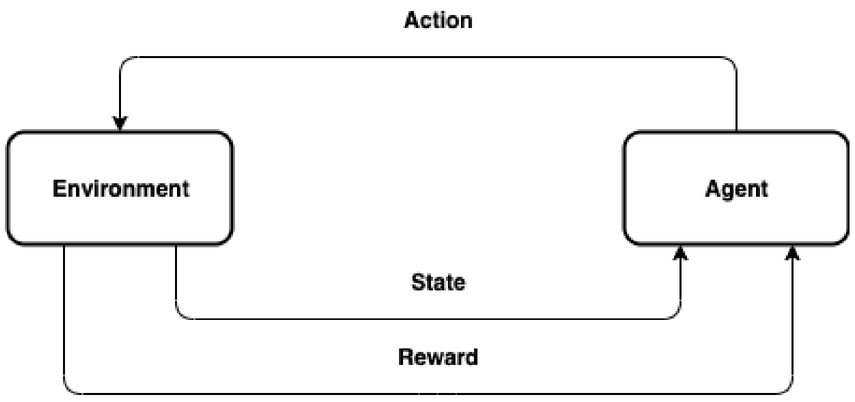

Figure 3.1: RL diagram. The environment provides the agent with the current state and the reward and based on that the agent performs the action.

Environment

The environment is the world with which an agent lives and interacts. It is usually some task that we try to solve, a game or a simulation. At each step of interaction, the environment receives the action to take from the agent and provides the agent with the observation (possibly only partial observation) of the world's state and the reward associated with the taken action. The environment usually changes when the agent performs the action but it also might change on its own. The environment can be fully observable when the agent can observe the complete state of the environment, or it can be partially observable when the agent only sees a partial observation [51].

Agent

An agent is a decision-maker, based on the observation received from the state of the environment, the agent decided which action to perform. After the action, the environment gets updated and returns a reward signal. The reward signal contributes to the calculation of new policy and/or value function [51].

Episode

A sequence of steps (state, reward, and action) can be joined together to create an episode. Usually, an episode starts from some initial state and ends when an agent reaches a terminal state, after which the environment is reset back to the initial state. In the grid-world example in Figure 3.2, the sequence of time steps from starting point to the finish point would be one episode [51].

Policy

A policy defines the behavior of the agent. It is a mapping between state and action, it shows which action to take when in a specific state. There are a lot of available solutions on how to encapsulate the policy. Starting from the simple lookup table to the more complex parameterized policies, their output is a computable function that depends on a set of parameters, that can be modified thanks to some optimization algorithm. There are two types of policies, deterministic policy always returns the same action given some observation - there is no uncertainty. Stochastic policy outputs a probability distribution over actions - the agent can perform different actions from the same state. As a consequence, the exploration/exploitation trade-off gets handled without the need for some hard coding [51].

Reward Signal and Return

At time step t, after taking the action the agent receives the reward signal $\mathrm{R_t}$. The goal of the agent is to maximize its cumulative reward, called **return $\mathbf{G_t}$**.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T = \sum_{k=0}^{T-t-1} R_{t+1+k} \tag{3.1}$$

where T is the final time step. Usually, the discounted reward is being used, which lowers the impact of future rewards for the benefit of immediate reward [51].

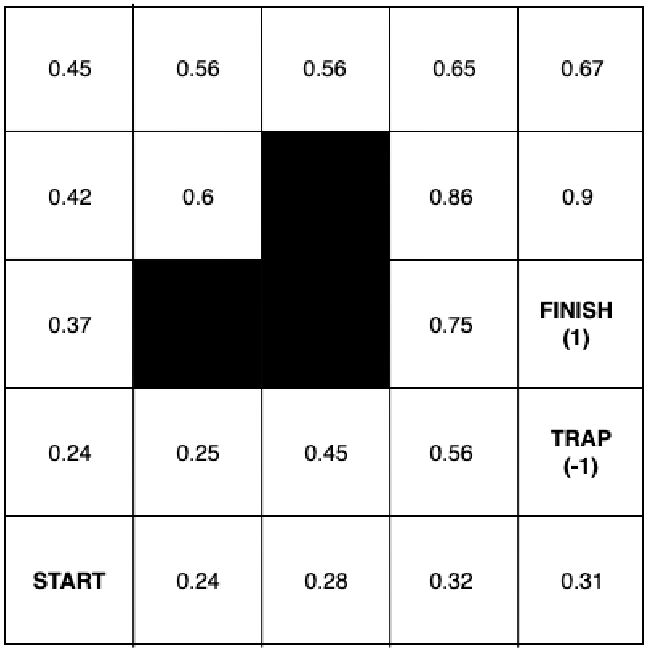

Figure 3.2: A grid-world environment. Each cell is a state and the value represents the value of the state-value function V$_\pi$(s). The agent of the agent is to reach the flags and avoid the trap.

$$G_{t,discounted} = R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \ldots + R_T = \sum_{k=0}^{T-t-1} \lambda^k R_{t+1+k} \tag{3.2}$$

where λ is a parameter, $0 \leq \lambda \leq 1$, called the discount rate [51].

Value-Function based algorithms

Value-function-based algorithms involve estimating value functions, which are functions of states that estimate how good it is for the agent to be in a given state. The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Informally, the value of a state s under a policy π, denoted V$_\pi$(s), is the expected return when starting in s and following π thereafter [51].

$$V_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \lambda^k R_{t+1+k}|S_t = s] \tag{3.3}$$

where $E_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of the terminal state, if any, is always zero. We call the function v the state-value function for policy π [51].

In Figure 3.2 a grid-world environment is shown where the agent has the task of getting to the finish by moving left, right, up, and down. The agent is rewarded for getting to the goal but must avoid the trap, for entering which he receives a penalty. Each cell displays a value that is a value from the state-value function for that state. As expected, values are higher closer to the finish and lower closer to the trap.

Policy-based algorithms

Policy-based algorithms work more directly than value-function-based algorithms. Policy-based methods search directly for a policy π that maximizes the expected return instead of finding the value for each possible state and based on that deriving optimal policy. The main idea behind policy-based algorithms is to iteratively update the policy parameters θ, so that the expected return increases. The optimization process can be formalized as follows: [30]

$$\theta_{i+1} = \theta_i + \delta\theta_i \tag{3.4}$$

Exploration vs Exploitation

While interacting with the environment, the agent can use these two strategies:

- Exploration: choose a random action - when following this approach agent can visit states never seen before.
- Exploitation: act greedily – choose actions that return the highest reward according to the accumulated knowledge.

One of the most used exploring policies is the ε-greedy policy, in which there is a ε

chance for the agent to take a random action and a 1ε chance for the agent to pick an action based on its policy. Usually, the value of the ε parameter is a function of time, where the ε value decrease over time so that the agent acts more greedily over time [30].

3.1.3 RL algorithms

Q-learning

Q-learning, known as an off-policy Temporal Difference (TD) control algorithm was one of the most important breakthroughs in RL. Q-learning uses action-value function Q$_\pi$(s,a) which shows how good it it to take an action a in state s. The action-value function is similar to the already introduced state-value function V$_\pi$(s) but in addition to the state it also considers the action [31]. Q-learning uses optimal ε-greedy policy. At the start Q$_\pi$(s, a) is initialized with random values and these values are updated during the training process [51]. Q-learning update procedure is defined by:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \lambda \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{3.5}$$

where α is the learning rate. The learning rate shows how much the old value should be updated. The steps for the algorithm for Q-learning are presented below [51].

Algorithm 1 Q-learning: An off-policy Temporal Difference (TD) control algorithm.

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $epsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathscr{S}^+, a \in \mathscr{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
while $S$ is not Terminal do
    Initialize $S$
    for $episodestep \leftarrow 1$ to $N$ do
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
```

3.2 Artificial Neural Networks as Function Approximators

The tabular version of Q-learning where the action-value function is a two-dimensional table over all possible states and actions works well for a small number of states but most tasks have very large state space which cannot possibly fit into the table. The solution for this problem is to approximate the action-value function with the parametrized function such as a neural network.

3.2.1 Feedforward Neural Networks

Feedforward neural networks, also often called MLP, are the quintessential deep learning models. The objective of the MLP is to approximate a function. For example, function $y = f(x)$ takes x as input and outputs y in response. MLP specifies a mapping $y = f(x;\theta)$ and during the training process that aims to find the best function approximation learns the value of the parameters θ. The model is called Feedforward because given x and when calculating y the information flows only in one direction - forward, from x to y [17].

The main and also smallest component of the network is a neuron. As shown in Figure 3.3, the neuron takes x as input, multiplies it with the weights θ, and sums it up. The sum is the input of the activation function which produces the output. The activation function is responsible for adding non-linearity to the network which enables it to approximate arbitrary functions. There are a lot of activation functions to choose from but the one used the most is Rectified Linear Unit (ReLU) [17]. The formula is as follows:

$$f(x) = \max(0, x) \tag{3.6}$$

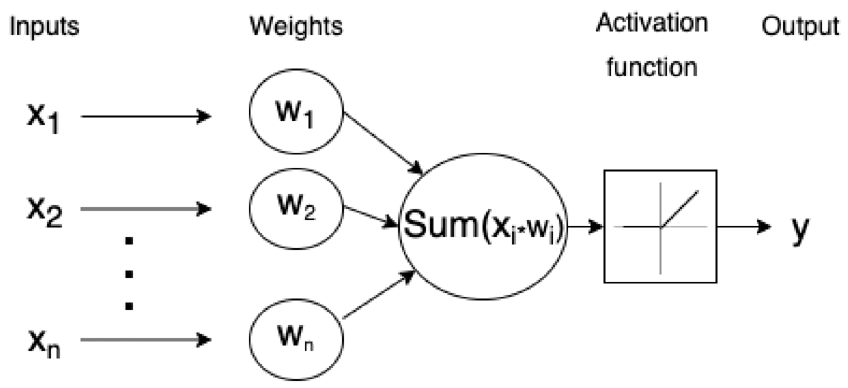

Figure 3.3: Mathematical formula for an artificial neuron. The inputs are multiplied by their corresponding weights and summed up. The sum is used as an input of the activation function which produces the neuron's output.

Feedforward neural networks are called networks because they are typically represented by chaining together many different networks - functions. In the example Figure 3.4, we have three columns of neurons. The column of neurons is called a layer, each neuron in the layer has the same inputs and the same activation function but separate weights. If we represent three consecutive layers connected in a chain as functions: $f^{(1)}, f^{(2)}$ and $f^{(3)}$, then the FNN would form a function: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. The $f^{(1)}$ is called the first layer, $f^{(2)}$ second layer, and so on. The "deep" in the "deep neural networks" and "deep learning" arises from the length of the chain which represents the depth of the model [17].

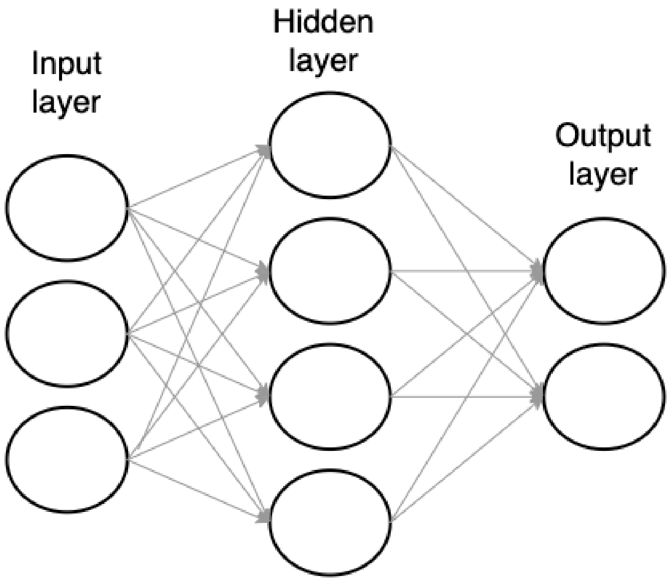

Figure 3.4: A feedforward neural network consisting of three layers: input, hidden, and output. The circles represent neurons and the connections between them and their weights.

Learning process

The learning process of the network, also called training aims to find values of the weights that result in the best possible approximation of the function by FNN. In a supervised learning setting, each input training example x is accompanied by a label y (also called true value) and the objective is to tune the weights so that given x as the input of the FNN the calculated output is as close as possible to the true value (label y) [17]. The whole training process is iterative and consists of the following steps:

1. **Forward pass**
   The input X is propagated forward through the network and the network outputs predicted value of y like: $y_{predicted} = f(x, \theta)$.

2. **Back-propagation**
   The just calculated output Y_{pred} is matched with the true output Y in order to compute the loss $L(\theta)$ which shows how close the predicted values are to the true values - how good is the performance of the network. There are a lot of loss functions to choose

from, depending on the task that is being solved but the most common one is the Mean Square Error (MSE) [47] with the formula:

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{n}(Y_i - Y_{pred,i})^2 \tag{3.7}$$

Back-propagation [43] is an algorithm that allows propagating the global gradient of loss $\nabla(\theta)$ backward (from the output layer to the input layer) so that there is a gradient of the error with respect to each neural network weight. The chain rule is used in order to backpropagate the gradient.

3. **Update**
   The last step is to update the weights of all neurons. The most common optimizer is the Stochastic Gradient Descent (SGD). The gradient is the direction in which the function increases the most, in this case, the direction and magnitude that increases the loss function the most but since we are trying to minimize the error, SGD updates the weights in the negative direction of the gradient of loss [17]. The update rule of gradient descent goes as follows:

$$\theta_i \leftarrow \theta_i - \alpha\nabla L(\theta) \tag{3.8}$$

where α is the learning rate parameter that defines how much the weight should be changed at each update.

3.2.2 Convolutional Neural Networks

Convolutional networks [28], also known as CNN, were made to process data with a grid-like topology. For example one-dimensional (1D) grid-like time-series data or 2D rid of pixels like image data. They have been extremely successful in practical applications. The "convolutional" in the name of CNN came from the mathematical operation - convolution [17]. The general formula for convolution is stated below:

$$s(t) = (x * w)(t) \tag{3.9}$$

where asterisk $*$ denotes convolution operation. The first argument x is referred to as input and the second argument w as a kernel. The output is called the feature map [17]. For a 2D image we need a 2D kernel and the formula is as follows:

$$S(t) = (K * I) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.10}$$

Which application to the one patch of the input image is shown in Figure 3.5.

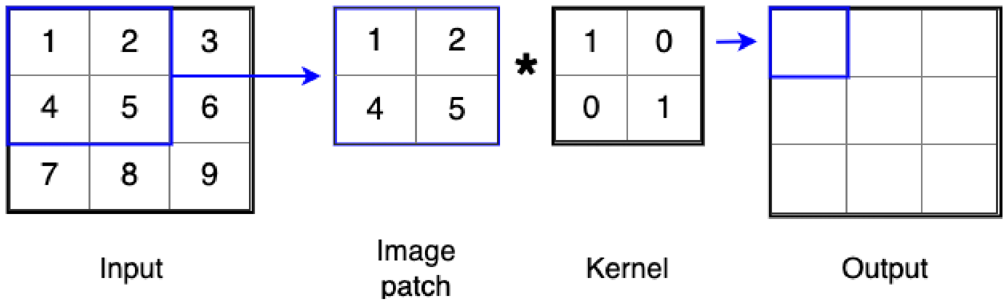

Figure 3.5: Example convolution operation applied on the one patch on the input image.

One neuron in a Convolutional Layer is defined by a single kernel and the values inside a kernel matrix are his weights. The approximate CNN with the fully-connected layer as the one to the last layer is shown in Figure 3.6.

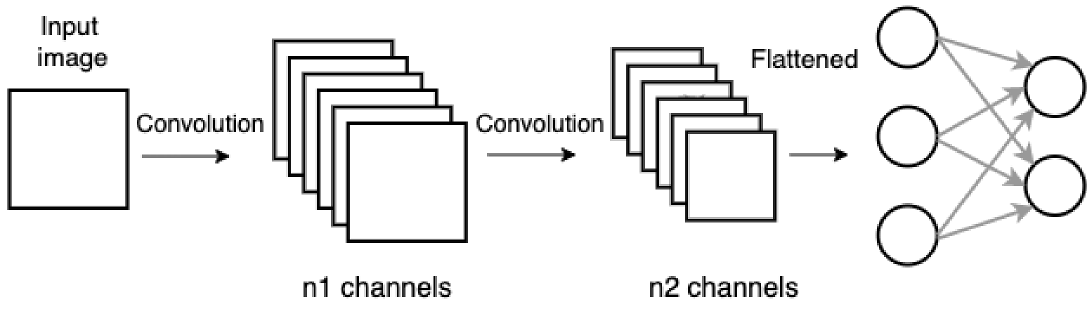

Figure 3.6: Example CNN with the fully-connected layer as the one to the last layer.

3.2.3 Recurrent Neural Networks

RNN [44] are a family of neural networks for processing sequential data. Much as a convolutional network is a neural network that is specialized for processing a grid of values such as an image, a recurrent neural network is a network that is specialized for the sequential type of data such as language translations or time series [17].

RNNs have "memory", they can influence current output based on not only current input but also prior inputs. They are able to achieve that thanks to the hidden state of the recurrent unit (neuron) [17]. The formula for calculation of the recurrent state is shown below:

$$h^{(t)} = f(h^{(t-1)}, x^(t); \theta \tag{3.11}$$

To better explain the hidden state, Figure 3.7 presents unrolling of the recurrent unit through time. The output of the recurrent unit from the previous time step is fed back into the unit in the next time step.

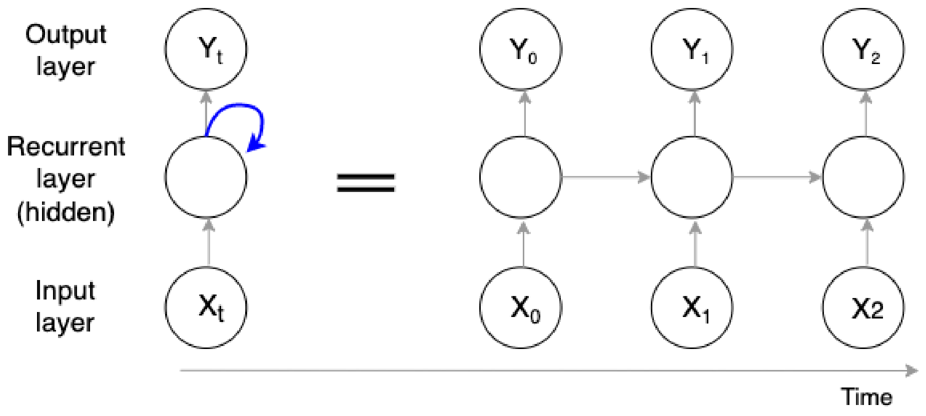

Figure 3.7: Unrolling of the recurrent unit through time.

Figure 3.8 below, shows an example of the RNN with the feedback connections in the recurrent layer.

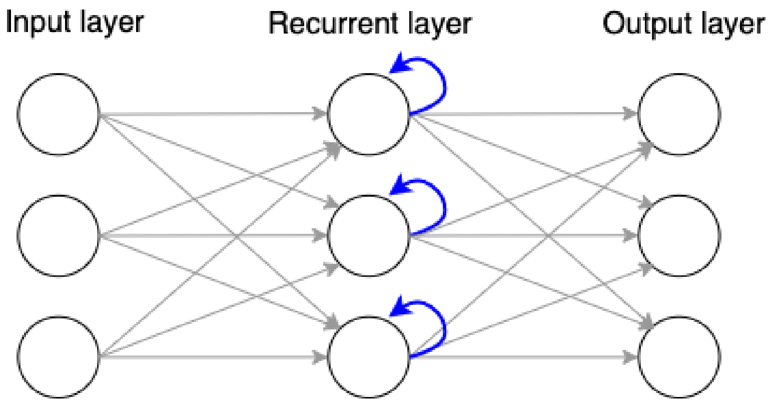

Figure 3.8: Example of the Recurrent Neural Network.

3.3 Deep Reinforcement Learning

As already mentioned in section 3.2, classical RL algorithms are based on a tabular learning scheme - they accumulate the knowledge in a table. Using a table strongly limits these algorithms to problems with a low number of states and actions. Real-world tasks usually consist of large state spaces. It is not possible to visit all possible states or state-action pairs but also due to memory constraints size of the table is limited. Using DNN as a function approximator instead of the table frees it from limitations and also has many other advantages such as: sharing knowledge about alike states or state-action pairs, faster training time, and a possibility to deal well with previously unseen states.

3.3.1 Deep Q-Learning

Deep Q-Learning was introduced in [35], where the researchers among other things used the Q-Learning algorithm with CNN as a function approximator to play Atari 2600 games and achieved human-level performance on them. The model takes pre-processed game frames, using CNN to extract features out of them, and outputs Q-values for each possible action. The DQN algorithm and its difference from Q-Learning are presented visually in Figure 3.9.

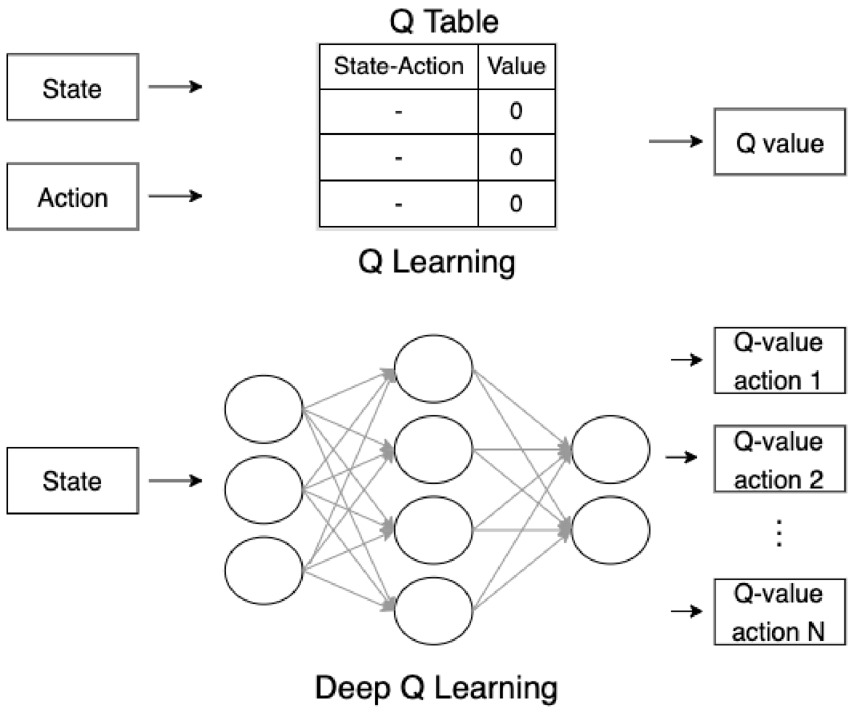

Figure 3.9: Difference between Q-Learning and Deep Q Learning (DQN) algorithms.

Vanilla DQN has suffered from unstable learning so the researchers came up with two excellent mechanisms to improve it: experience replay and the frozen target network.

The objective of the experience replay is to store the agent's experience at each time step: (s_t, a_t, r_t, s_{t+1}) in a replay buffer, where s_t, a_t, r_t is a state, action and reward at time step t respectively and s_{t+1} is a new, following state. At each time step, a batch of experiences is sampled from the experience buffer at random. During training, the trajectories being returned by the environment are correlated and can cause the model to overfit. Experience replay breaks temporal correlations but also repeats old experiences.

Frozen target network, presented in DDQN algorithm [22], introduces another neural network, called target network in order to reduce overestimation of Q-values and smooth out the learning process. The "frozen" word comes from the process in which weights of the target network are frozen most of the time and the network is updated less often than the main network by copying weight between them.

The steps of the DQN algorithm with experience replay are presented below:

Algorithm 2 Deep Q-learning with Experience Replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights
for $episode \leftarrow 1$ to $M$ do
    Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi 1 = \phi(s_1)$
    for $t \leftarrow 1$ to $T$ do
        With probability $\varepsilon$ select a random action $a_t$ otherwise select $a_t = \max_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{t+j})$ from $D$
        Set $y_j = \begin{cases} \mathrm{r}_j & \text{if terminal } \phi_{j+1} \\ \mathrm{r}_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal} \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j Q(\phi_j, a_j; \theta))^2$
```

3.3.2 Proximal Policy Optimization

This section will introduce actor-critic architecture, the policy gradient algorithm and at the end the PPO algorithm.

Policy Gradient

As already explained in section 3.1.2, the policy is a function that maps states into actions, notifying the agent about which action to take from that state. Policy gradient methods are a set of algorithms that use parametrized types of policy $\pi(a|s, \theta)$. Given that a neural network is used to approximate policy function the parameters θ are the weights of the network [51].

Policy returns the probability of taking an action a while being in the state s at the time step t, given θ as its parameters:

$$\pi(a|s,\theta) = PrA_t = a|S_t = s, \theta_t = \theta \tag{3.12}$$

The objective of the policy gradient method is to maximize a performance measure $J(\theta)$ [51]. Parameters θ are being updated via gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{3.13}$$

Methods that learn approximations to both policy and value functions are often called actor-critic methods, where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function, usually, a state-value function [51].

Algorithm

The objective of the PPO algorithm is to optimize performance measure $J(\theta)$. At each time step, the parameters θ get updated in a way that ensures that the difference between the previous policy and the updated policy is small [51]. The formula for the objective, called the Clipped Surrogate Objective is defined by:

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \tag{3.14}$$

where ε is a hyperparameter, $r_t(\theta)$ denotes the probability ratio with formula 3.15, A_t is an estimator of the advantage function and $clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$ function keeps the value of $r_t(\theta)$ within some range [45].

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.15}$$

The steps of the PPO algorithm with Actor-Critic architecture are presented below:

Algorithm 3 PPO, Actor-Critic Style.

```
for $iteration \leftarrow 1$ to ... do
    Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi 1 = \phi(s_1)$
    for $actor \leftarrow 1$ to $N$ do
        Run policy $\pi_{\theta_{old}}$ in environment for $T$ time steps
        Compute advantage estimates $A_1, ..., A_T$
        Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
        $\theta_{old} \leftarrow \theta$
```

3.4 Explainable AI

is a research field that aims to make the results of AI systems more understandable to humans. It has been present in AI research since the 1970s but has risen to importance in the last decade as ML has become more prominent and applied in many real-world problems in the fields such as healthcare, finance, military, transportation, or legal [24].

3.4.1 Introduction

In the quest to make AI systems explainable, several explanation methods and strategies have been proposed. The XAI taxonomy is presented in Figure 3.10. The interpretability methods are classified according to three criteria [2]:

- the complexity of interpretability,
- the scoop of interpretability,
- the level of dependency from the used ML model [2].

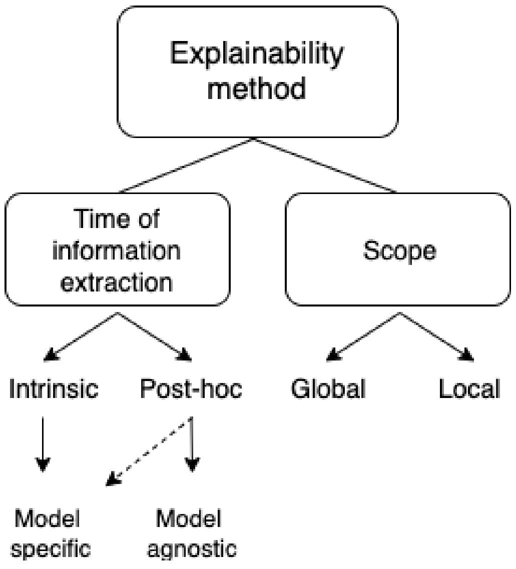

Figure 3.10: A pseudo ontology of XAI methods taxonomy.

Scoop Related Methods

Scoop-related methods can be distinguished between two subclasses:

- **Global interpretability**

  Global interpretability allows us to understand the whole logic behind the model over some data population [2].

- **Local interpretability**

  Local interpretability explains the reasons behind single prediction [2].

Model Related Methods

Another important way to classify model interpretability techniques are whether they can be applied to any ML algorithm or just a single type of algorithm [2]:

- **Model-Specific interpretability**

  Model-specific interpretability methods are limited to specific model classes. Intrinsic methods are by definition model-specific [2].

- **Model-Agnostic interpretability**

  Model-agnostic methods are not limited to a particular type of ML model, they separate prediction from explanation. Model-agnostic interpretations are usually post hoc and can be used to interpret DNN in local and global scope [2].

3.4.2 Integrated Gradients

IG [50] is a post hoc, model-agnostic interpretability method. It is used to attribute the prediction of a DNN to its input features - called feature attribution or feature importance. IG is usually recommended for interpretation of DNNs and just differentiable models in general.

We have a function F that represents DNN, an attribution of the prediction at input $x = (x_1, ..., x_n)$, relative to a baseline input x' is a vector:

$$A_F(x,x') = (a_1, ..., a_n) \tag{3.16}$$

where a_i is the contribution of x_i to the prediction $F(x)$.

Why the need for a baseline? Humans usually perform attribution in a counterfactual fashion - when assigning blame to something, we implicitly consider its absence as a baseline. Baselines for IG should be a long distance from the average input and also be “neutral” - in a sense where DNN output for the baseline is close to zero. A good example would be a black image when dealing with CNN [50].

Formally, the integrated gradient along the i-th dimension for an input x and baseline x' is defined as follows:

$$IntegratedGrads_i(x) = (x - x') \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} d\alpha \tag{3.17}$$

where $\frac{\partial F(x)}{\partial x_i}$ is the gradient of F along the i-th dimension at x [50].

3.5 Summary

This chapter introduced theoretical knowledge of the ML algorithms used in this work in order for the reader to be able to follow the path of thought and understand the presented solution methodology in Chapter 5 and analysis in Chapter 6.

Chapter 4

Environment Setup

4.1 Architecture of the training environment

4.1.1 Architecture

The simulation environment that has been used was implemented by the authors of papers [13, 14] and provided to us for usage in this Thesis. A fundamental component of the simulation process is implemented with CloudSim Plus [48] simulation environment. The environment is wrapped with the interface provisioned by the Open AI Gym framework [5]. Open AI Gym is the most used RL environment, that provides a simple and functional interface that meshes really well with existing RL algorithms frameworks. RL algorithms used in this work were provided by the Stable Baselines3 library [40] and the XAI algorithms were provided by the Captum [27] library. Both of these libraries are based on the well-known DNN framework PyTorch [39]. The below Figure 4.1 presents the system architecture.

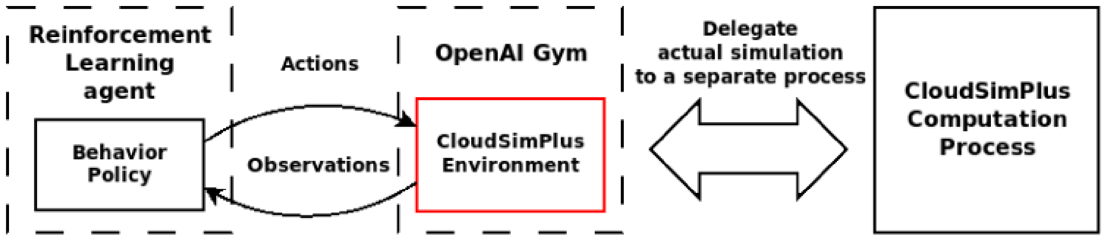
Reinforcement Learning agent
Behavior Policy
Actions
Observations
OpenAI Gym
CloudSimPlus Environment
Delegate actual simulation to a separate process
CloudSimPlus Computation Process

Figure 4.1: Components of the system [13].

4.1.2 Workload

The workload used in this Thesis is a simple evolutionary experiment (pytorch-dnn-evolution) that improves the architecture of a network that recognizes handwritten numbers (the MNIST dataset [9]). Below, in Figure 4.2 is presented the example workload run.

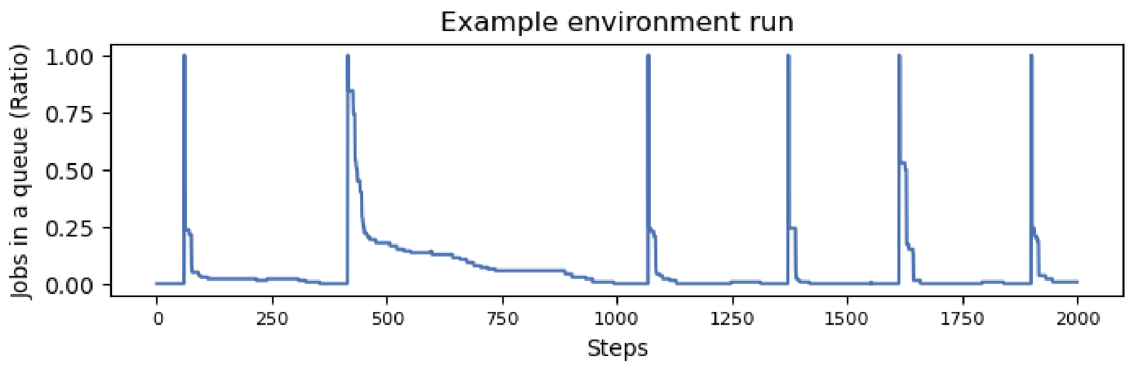

Figure 4.2: Example environment run which shows workload dynamics.

4.2 Environment description

The main objective of the agent was to allocate cloud infrastructure's resources in an optimal way to the running workload. In order to finish the experiment in a reasonable time, simulation time was sped up and the number of steps per episode was limited.

4.2.1 Objective function

A reward is supposed to give the agent feedback information on how good his previous action was with regard to the environment state and the overall goal of the experiment. The reward function tries to keep the cost of the running infrastructure as low as possible, while still maintaining the running application so that the waiting queue for its tasks is minimized. The reward function equals the negative cost of running the infrastructure plus the SLA penalty. The SLA penalty adds some cost for every second delay in task execution and was calculated by multiplying the number of seconds of delay of task execution by the penalty value. The cost of running the infrastructure was calculated by multiplying the number of VMs by the hourly cost of using the VM.

$$RewardFunction = \max -(CostOfInfrastructure + SLApenalty) \tag{4.1}$$

where:
$CostOfInfrastructure = NumberOfVMs * HourlyCostOfVM$, $HourlyCostOfVM = 0.2\$$,
$SLApenalty = NumberOfSecondsOfDelay * Penalty$, $Penalty = 0.00001\$$

4.2.2 Environment state

A state is represented by a vector/matrix of cloud infrastructure metrics, that are being calculated at each time step of the simulation. These metrics include:

- the number of running virtual machines - vmAllocatedRatio,
- average RAM utilization - avgMemoryUtilization,
- 90-th percentile of RAM utilization - p90MemoryUtilization,
- average CPU utilization - avgCPUUtilization,
- 90-th percentile of CPU utilization - p90CPUUtilization,
- total task queue wait time - waitingJobsRatioGlobal,

- recent task queue wait time - waitingJobsRatioRecent.

4.2.3 Actions

The set of available actions is limited to:

- do nothing,
- add/remove small VM,
- add/remove medium VM,
- add/remove large VM.

The decisions of the agent are not always immediately executed in the environment. Taking an action is subject to environmental restrictions [14].

4.2.4 Training

To reduce the training time, the time in the simulation was speeded up 60 times [14] and also the maximum number of steps in the episode was limited to 500 steps.

4.3 Summary

This chapter described in detail the simulated environment that allows us to understand the problem more precisely. Presented environment metrics and available actions will be repeatedly used in the subsequent Chapter 5 and following it Chapter 6.

Chapter 5

Agent Decision Interpretation System Design

This chapter will present the proposed methodology on how to interpret the decisions of the agent that manages cloud computing resources. It will show different approaches to the agent's interpretability based on the type of the DRL algorithm used and the underlying type of the DNN model's architecture.

5.1 Explainability in the cloud resources management system

The proposed explainability component will work as an "offline" tool that provides necessary analysis and debugging tools for the computer scientist responsible for building and managing the DRL-based cloud resources management system. By "offline", it means that it will not provide explanations for the predictions made by the agent in real-time (although it is proposed to be implemented in future works) but can be used for the analysis of the model in the different stages of its lifetime:

- **Design and Development**
  The proposed explainability component can be an invaluable tool during the design and

development process of the system.

  – **Performance**
    Besides all the well-known performance metrics, explainability can be really helpful to debug and improve the performance of the models.

  – **DRL algorithm and model design**
    Interpretation can also be very helpful in the combined design plus creation/training phase by providing feedback to the design phase on how to modify the model's architecture or inform about the training algorithm for the best adaption to the problem.

  – **System understanding**
    Explainability can vastly contribute to the understating of the agent making it more trustworthy for potential clients.

- **Utilization, Monitoring and optional retraining**
  It is very important to monitor the model during usage in production in order to make sure that its performance doesn't diminish and if it does - act on it immediately. Interpretability can have a positive impact on the maintenance of the system thanks to constant monitoring of the model:

  – **Examples causing poor performance**
    Find examples that perform poorly and then run on them the analysis using interpretability in order to find the roots of the problem.

  – **Proactive monitoring**
    Systematic checks of the data shifts with the additional support of the global and local interpretability analysis resulting in model debugging and optional retraining if deemed necessary.

  – **"Online" decision support**
    It is also possible to use the explainability component in the human-in-the-loop setting, where the human system's supervisor can monitor and make decisions instead of the agent by using its prediction and explainability for support. Infor-

mation to the supervisor could be given in the form of text or graphics as shown in Figure 5.1.

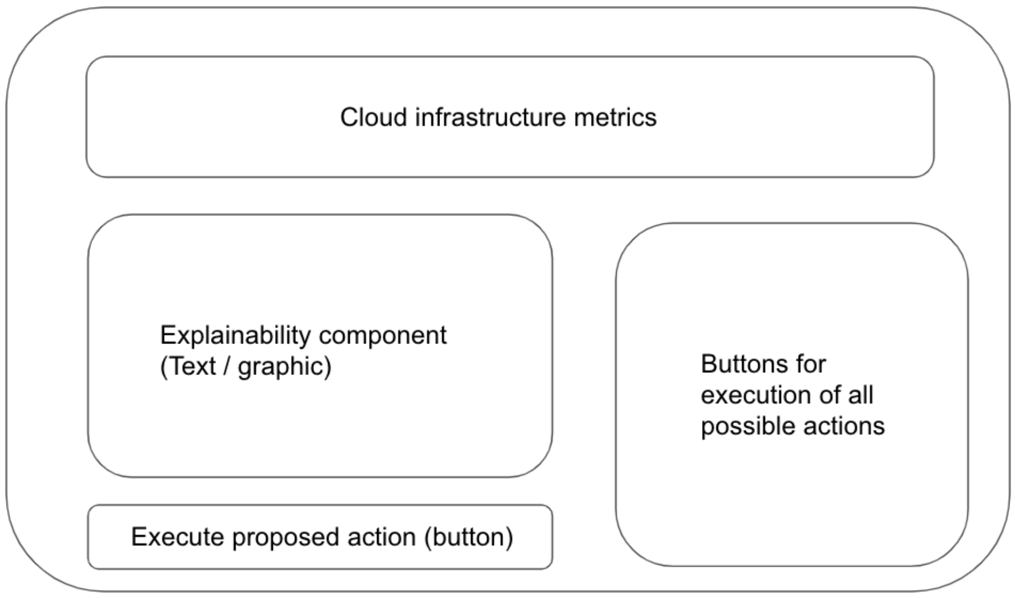
Cloud infrastructure metrics
Explainability component (Text / graphic)
Buttons for execution of all possible actions
Execute proposed action (button)

Figure 5.1: Decision support system's UI

Text information could be as simple as presenting feature value-attribution relation with respect to the action. It could consist only of the most positive attributions that explain why this action was chosen, for instance:
"Proposed action is to remove large VM, because:

- average CPU utilization is low,
- wait queue size is low."

Or provide both the most negative and positive attributions:
"Proposed action is to remove large VM, reasons for the decision:

- average CPU utilization is low,
- wait queue size is low.

Reasons against the decision:

    * total task queue wait time is high,
    * 90-th percentile of CPU utilization is low."

The graphic approach provides more information but it would be up to the supervisor to interpret them, which could take some time. An example would be the plot of the attributions.

5.2 Explainable Reinforcement Learning

Explaining the DRL agent that runs autonomous resource management of cloud computing infrastructure means finding the relation between input data feeding the agent and his predictions. In order to find this relation, this work will explain the workings of DNN upon which this agent is built, as done very successfully in the three works presented before [19, 25, 36]. We will use the XAI technique - the XAI algorithm, to explain the DNN's predictions. XAI is a simple, yet powerful axiomatic attribution method that requires almost no modification of the original network. XAI will show how much each input feature contributed to the output. In terms of the presented problem, it would inform on which cloud infrastructure's metrics was the decision made and exactly how. The positive attribution value for the feature means that this feature contributed positively towards a prediction making it more likely to be chosen and negative attribution is just the opposite. The proposed system would show a plot of the actual state of the infrastructure - infrastructure metrics and attributions for the chosen action or each possible action.

5.3 XAI applied to agents

This section compares how different DRL algorithms and underlying DNN architectures can be interpreted using XAI and also their advantages and disadvantages. There are two DRL algorithms used in this work: DQN and PPO. Both algorithms are considerably different but they have one main thing in common, they both use DNN as the backbone for their training and prediction processes.

5.3.1 DQN

DQN algorithm outputs q-values that represent the probability of taking the action *a* in the state *s*. There are q-values for each possible action and the final prediction is an action with the highest probability. The XAI can calculate feature attributions for each possible action. Behind each action is a q-value which tells us how "good" is that action. The XAI calculates feature attribution in relation to each action by taking their respective q-value. Figure 5.2 shows DQN.

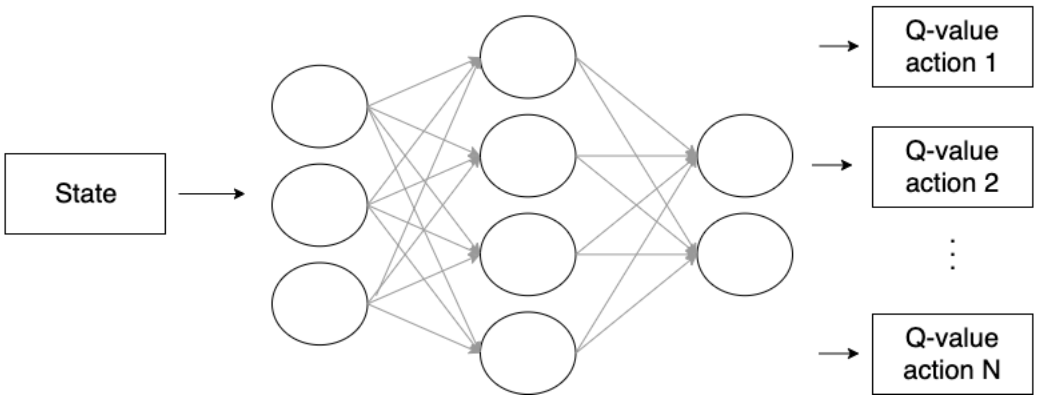

Figure 5.2: Simple draft of the DQN model's network.

MLP Policy

MLP is the simplest and the oldest DNN architecture. Due to its simplicity, it carries less information than other architectures but it's easier to understand and faster to calculate attributions on. MLP accepts as input, 1D vector of feature observations. So at time step-t, it is limited in its attribution to just that one 1D vector from the same time step.

Figure 5.3 presents the attributions for the best action which are in the form of the two-figure plot:

- the first one shows the environment state, the y-axis presents metric value (0-1)

- the second one, below it, shows the attributions value (positive in green, negative in red) for the best action

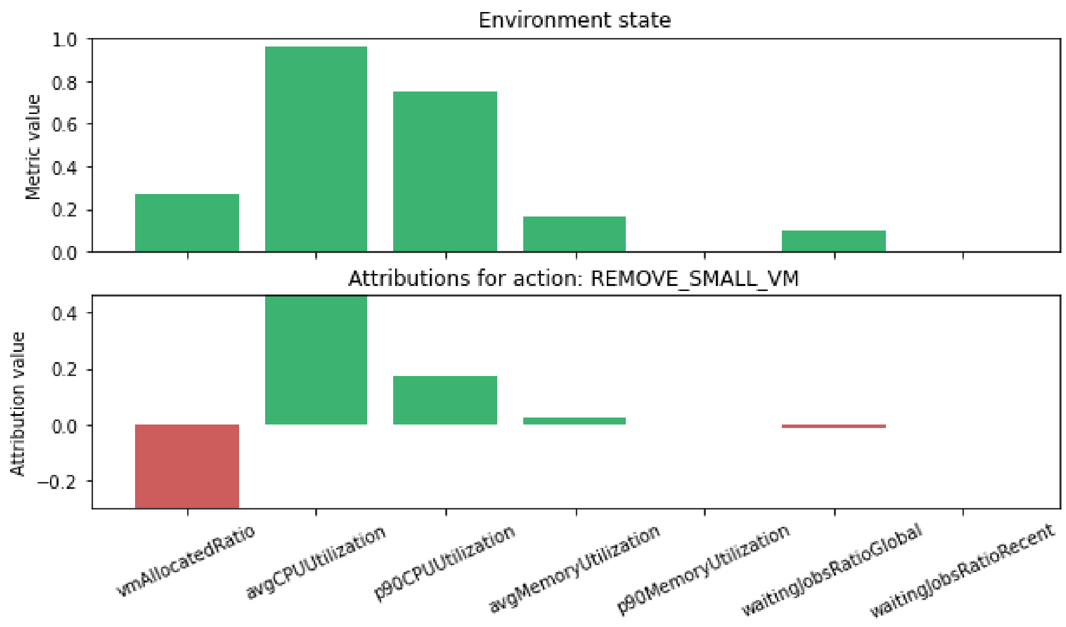

Figure 5.3: An example agent's decision explanation is presented in form of the visual bar plot for the chosen action.

Figure 5.4 presents attributions for each possible action so that we can also understand the reasons behind the prediction for the actions that were not chosen.

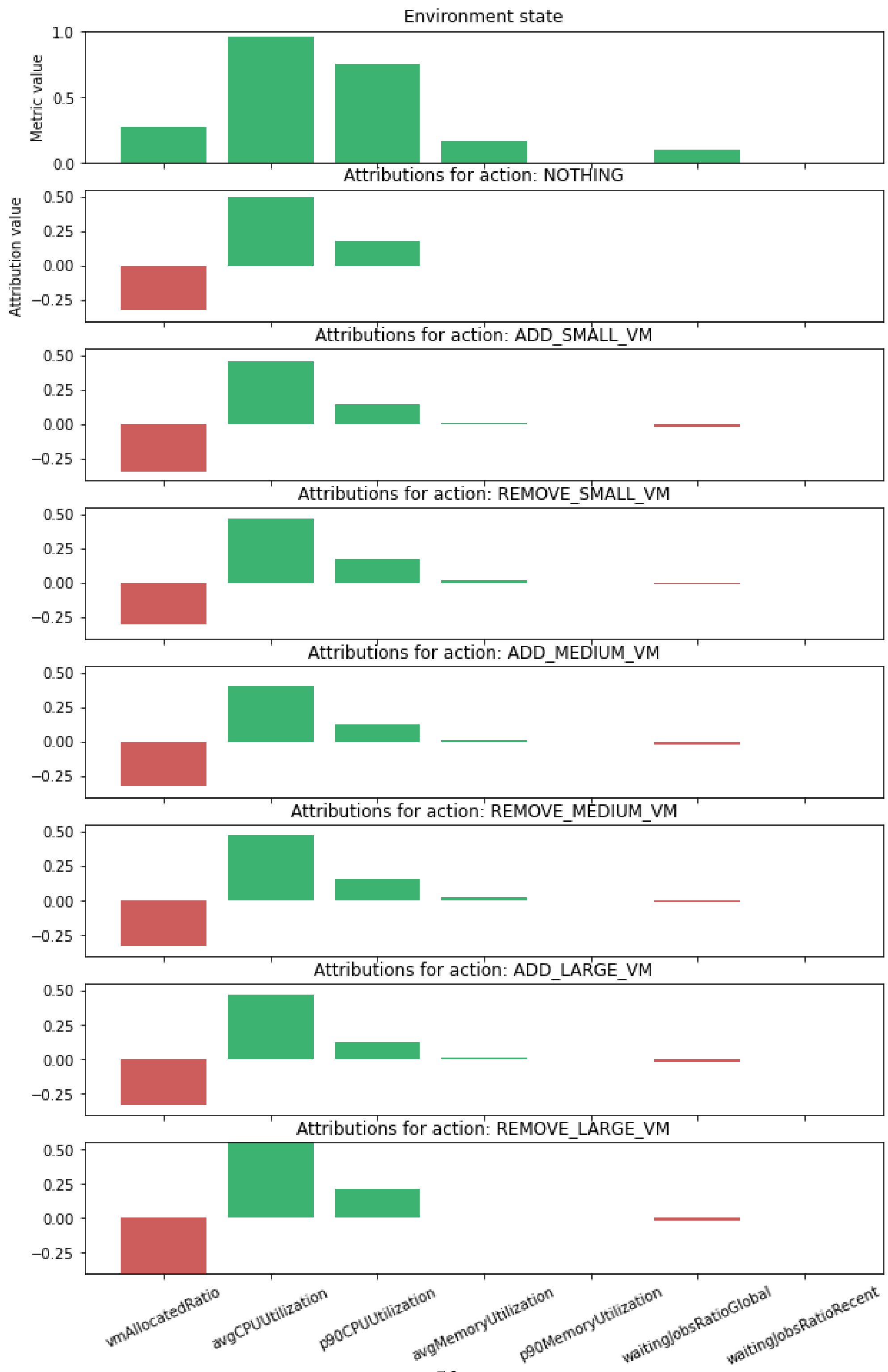

Figure 5.4: An example agent's decision explanation is presented in form of the visual bar plot for each possible action.

CNN Policy

CNN is mostly used in image recognition as well as in time series prediction. The main advantage of CNN over MLP in the task of attribution is that CNN can present attributions for action in time step-t for input features from previous n-time steps. It is important because autonomous systems such as this one have some lag but also most of the time agent's decision is based upon changes in the infrastructure that took place sometime in the past. The idea behind including environment state from the past works on taking 2D frames from the environment state's history, shown in Figure 5.5.

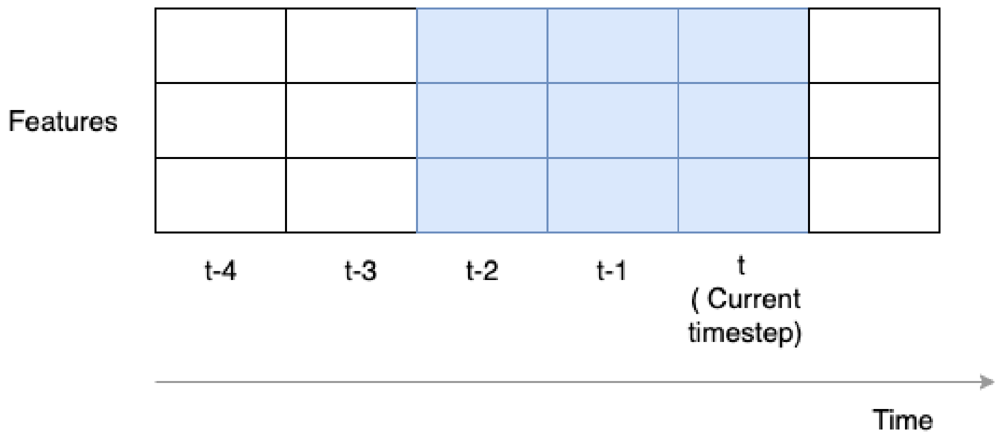

Figure 5.5: Shown in blue, it is the example frame from the feature's history (it includes 3 time steps, the current one, and two frames before that).

Figure 5.6 presents the attributions for the best action which are in the form of the three-figure plot:

- the first one shows the frame of the environment state, x-axis shows time steps, from 0 - the current one down to the -14 time step (15-th time step from the past). the y-axis presents a value (0-1) for each environment metric
- the second one shows the positive attributions value (the higher the value the greener the color) for the best action

- the third one, shows the negative attributions value (the higher the value the redder the color) for the best action

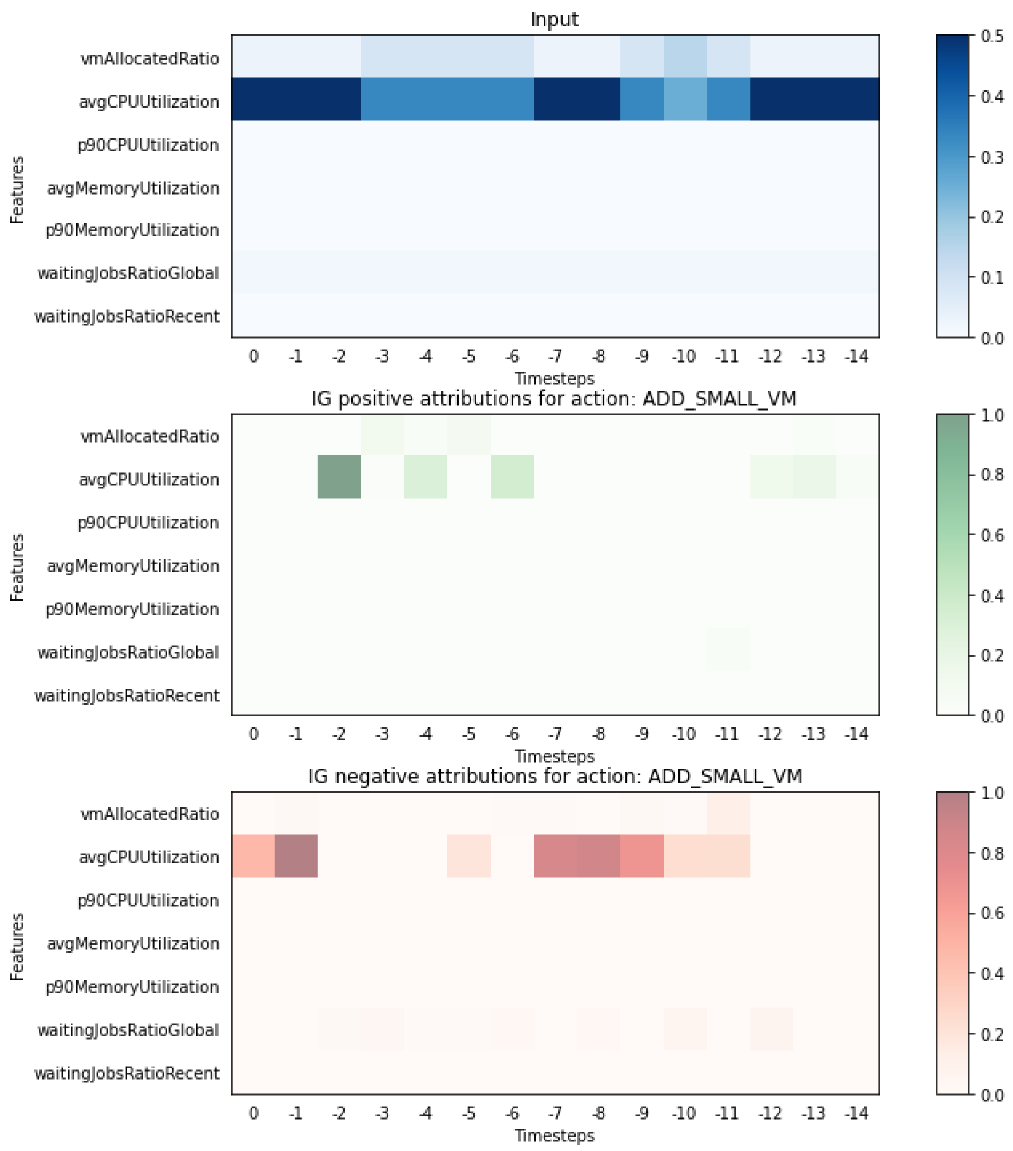

Figure 5.6: Example agent's decision explanation presented in form of the visual heat map for both positive and negative attributions.

5.3.2 PPO

PPO algorithm, having actor-critic architecture outputs two values: critic network returns a value - predicted discounter reward and actor-network returns action distribution. We are interested in the behavior of the actor because he is directly responsible for selecting an action so we will use XAI to calculate attributions for the input features in relation to the action distribution. This concept is presented in Figure 5.7.

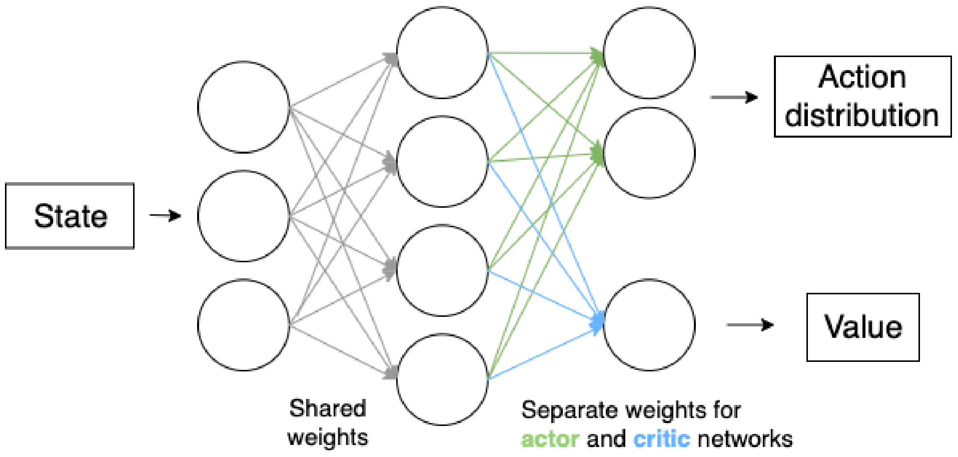

Figure 5.7: Simple draft of the PPO model's actor and critic networks.

The attributions for PPO are in the same style as the ones for DQN, where the first layer - known as a feature extractor, enforces the kind of attributions: bar plot for MLP, such as one in Figure 5.3 and 2D plot for CNN, like the one in Figure 5.6. There are four architectures used with the PPO algorithm: MLP, CNN, MLP Recurrent, and CNN Recurrent.

5.4 Summary

This chapter presented the proposed methodology on how to interpret the decisions of the agent that manages cloud computing resources. There are shown multiple approaches to interpretation for different DRL algorithms used and the underlying type of the DNN that will be absolutely necessary to obtain a good understanding of the analysis presented in the following Chapter 6.

Chapter 6

Analysis and Results

This chapter will provide a methodology for the explainability analysis of the different agents, present training and interpretability results, and perform the analysis of those results.

6.1 Training

In Figure 6.1 we can see mean episode rewards during the training of all of the agents and in Figure 6.2 we can find mean episode rewards during the training of the PPO agents.

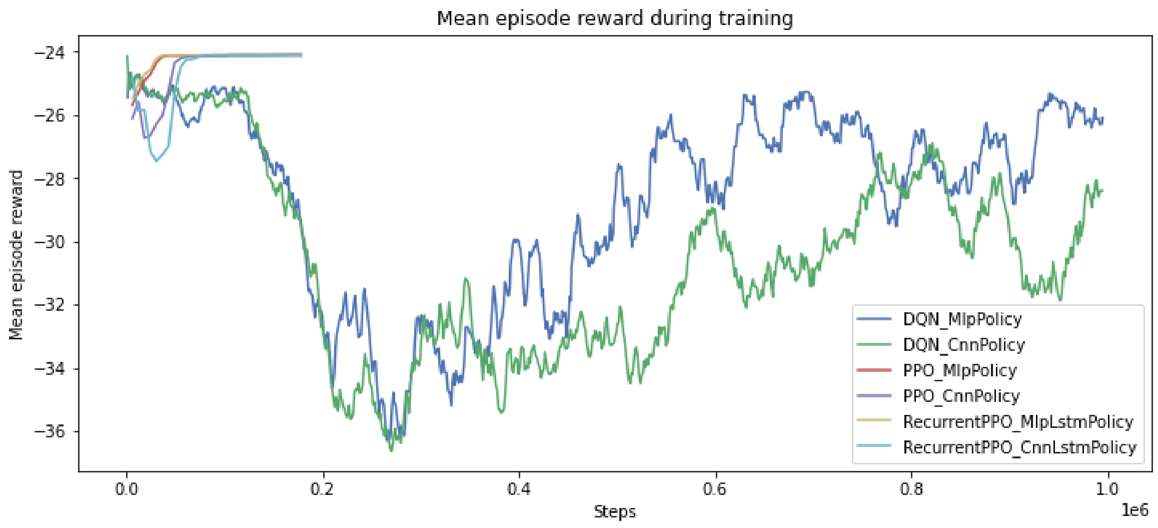

Figure 6.1: Mean episode rewards during the training process of the agents.

Conclusions from data in Figure 6.1:

- PPO agents reach optimal rewards much faster than DQN agents
- PPO agents achieve higher maximum mean episode reward
- DQN agents needed to train five times longer to achieve optimal results

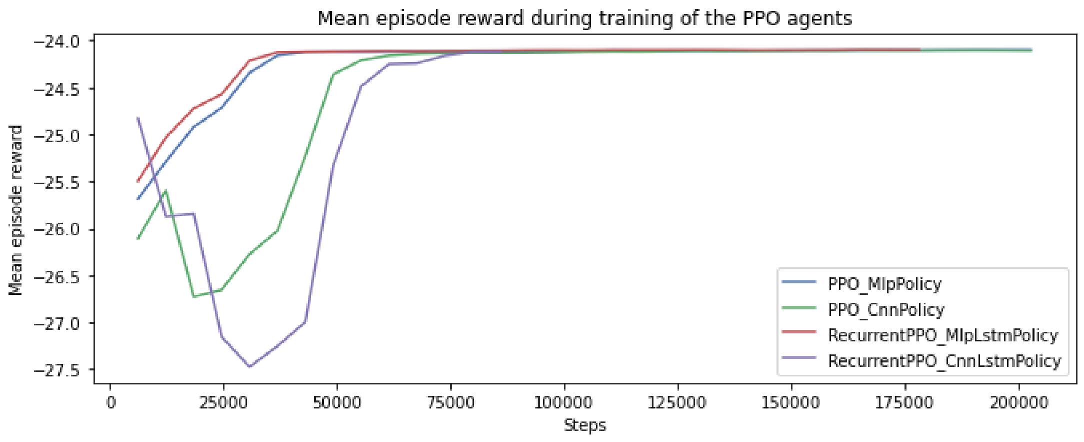

Figure 6.2: Mean episode rewards during the training process of the PPO agents.

Conclusions just for PPO agents from data in Figure 6.2:

- All of the PPO agents went through a very similar training process and achieved the same optimal, maximum mean episode reward
- Recurrent PPO agents have more parameters which should be reflected in the slower training process at the beginning and Recurrent PPO agents with CNN as feature extractor support this
- In contrast, Non-recurrent PPO agents have fewer parameters which should be reflected in the faster training process at the beginning, and both non-recurrent agents support this.

6.2 Example run

This section presents example runs of the agent in order to understand how they are working:

- how are they reacting to the workload spikes
- how fast do they increase the number of VMs
- how fast are they executing tasks and decreasing the queue

Conclusions:

- DQN-MLP model increases the number of VMs the slowest and is mostly reactive to the workload changes
- Both DQN models do not use more than 65% of available VMs as shown in Figures 6.3, 6.4
- PPO agents use up to 100% of available VMs as shown in Figures 6.5, 6.6
- Both Recurrent PPO models usually allocate 100% of resources even before the spikes which could be attributed to their memory component as shown in Figures 6.7, 6.8

- Recurrent PPO-CNN is quite chaotic, with a lot of small adjustments
- All of the models besides DQN-MLP show signs of being a proactive algorithm because they allocate resources even before the workload spikes by predicting future possible workload increases

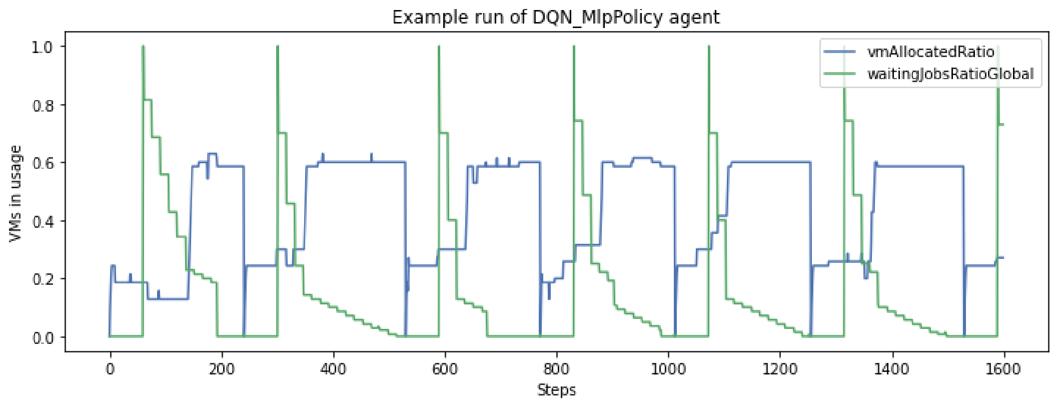

Figure 6.3: Example run of the DQN agent with MLP feature extractor.

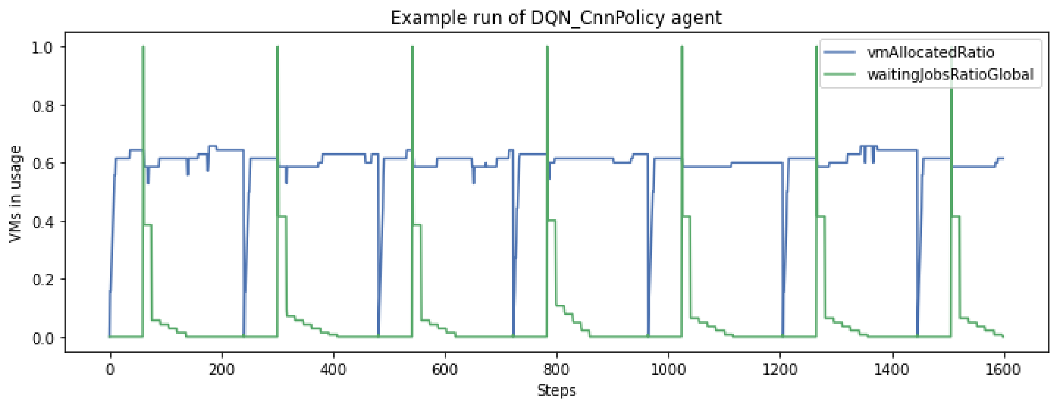

Figure 6.4: Example run of the DQN agent with CNN feature extractor.

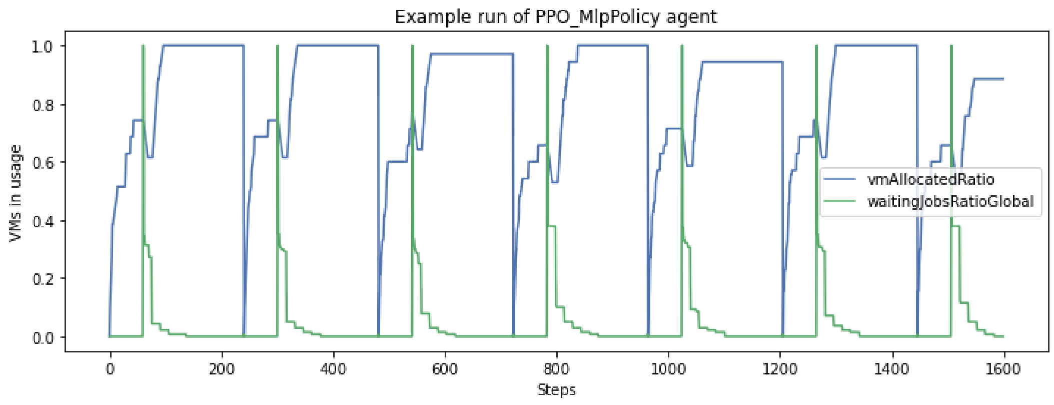

Figure 6.5: Example run of the PPO agent with MLP feature extractor.

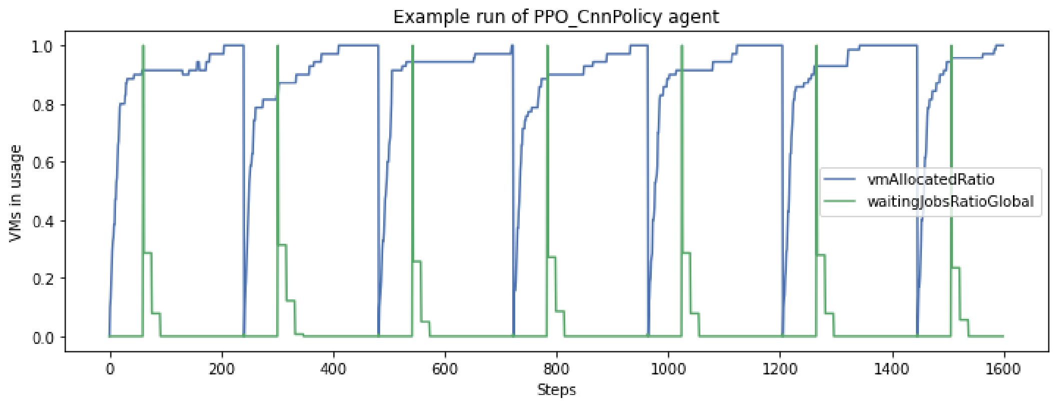

Figure 6.6: Example run of the PPO agent with CNN feature extractor.

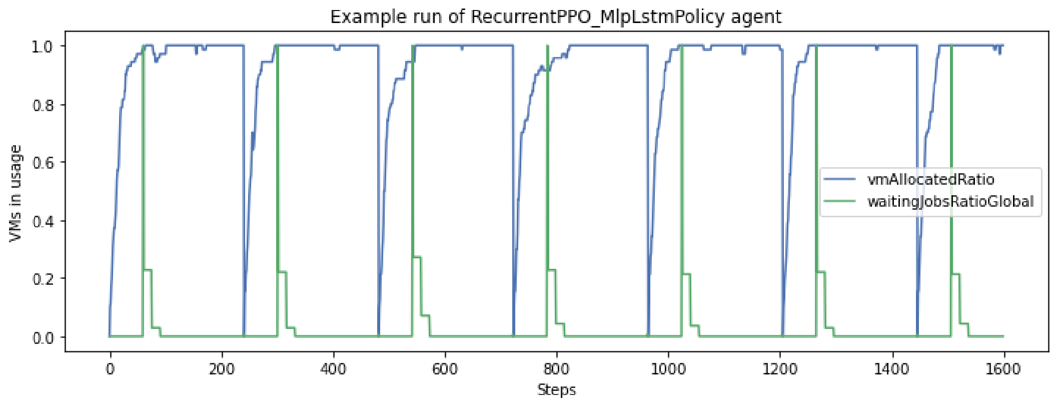

Figure 6.7: Example run of the Recurrent PPO agent with MLP feature extractor.

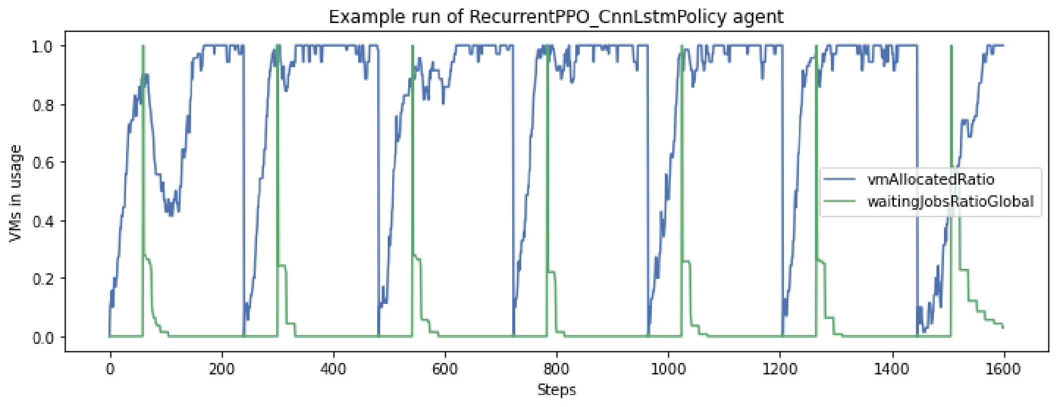

Figure 6.8: Example run of the Recurrent PPO agent with CNN feature extractor.

6.3 Evaluation

The section presents the results of the evaluation of the DRL agents plus a baseline model - Rule based model, in which actions are chosen according to the average CPU utilization thresholds, as shown in Figure 6.9. We can see that all of the agents performed similarly well on the same workload that was used for their training except for the DQN agent with MLP architecture. We can also see that the DRL agents (except DQN-MLP) performed better than the baseline threshold-based model. As for the evaluation of the test workload, shown in

Figure 6.10, the results were quite similar with PPO-CNN and Recurrent PPO-MLP agents being the best.

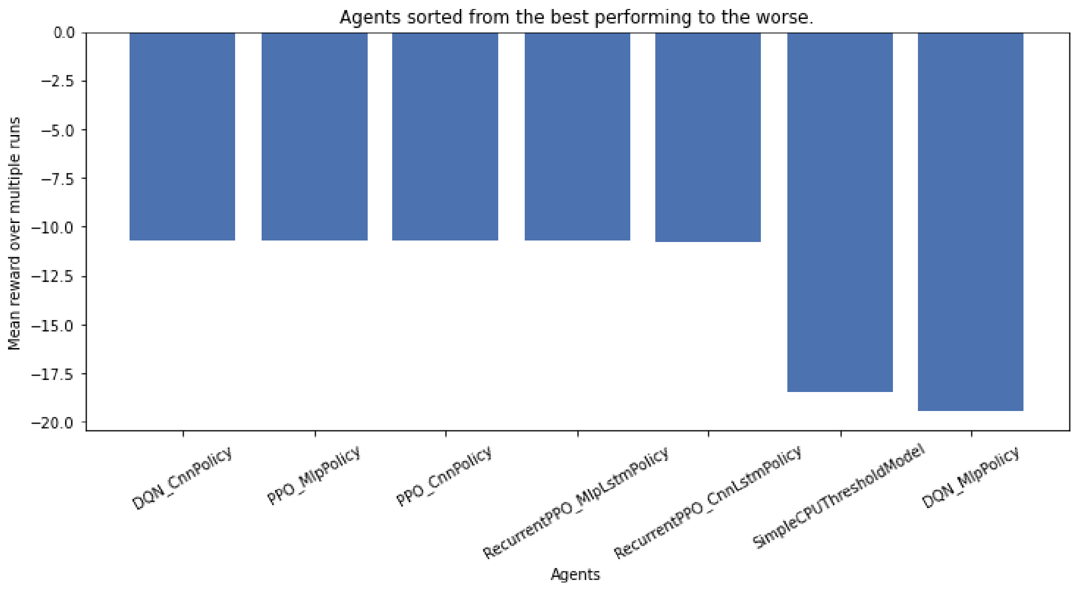

Figure 6.9: Evaluation of the trained agents on the training workload.

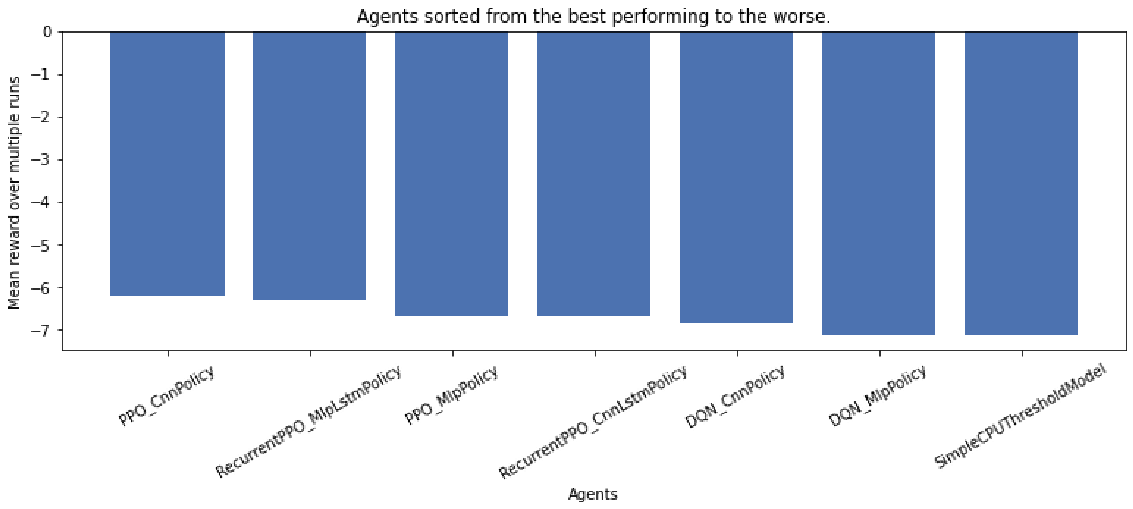

Figure 6.10: Evaluation of the trained agents on the test workload.

6.4 Action histograms

This section presents histograms of agent actions from an example run in order to understand the agent's decisions in a cumulative manner. Conclusions:

- DQN agents, as shown in Figures 6.11, 6.12 are much less balanced than PPO agents, displayed in Figures 6.13, 6.14, where they use mostly 1-2 actions which we would attribute to the sampling of the actions from the distribution in the PPO agents.
- Some models increase resources allocation much more aggressively in order to prepare the environment for the workload spike such as PPO-MLP or Recurrent PPO-MLP, shown in Figure 6.15
- PPO-CNN agent slightly more aggressively frees the resources then allocates them

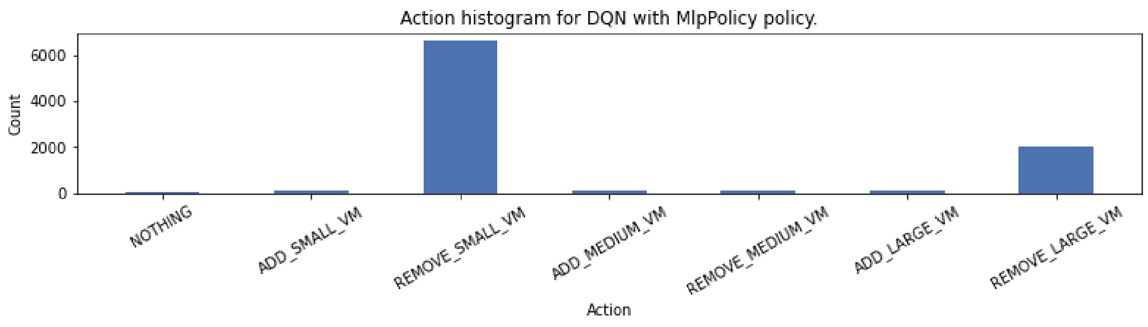

Figure 6.11: Action histogram for DQN model with MLP policy over multiple runs.

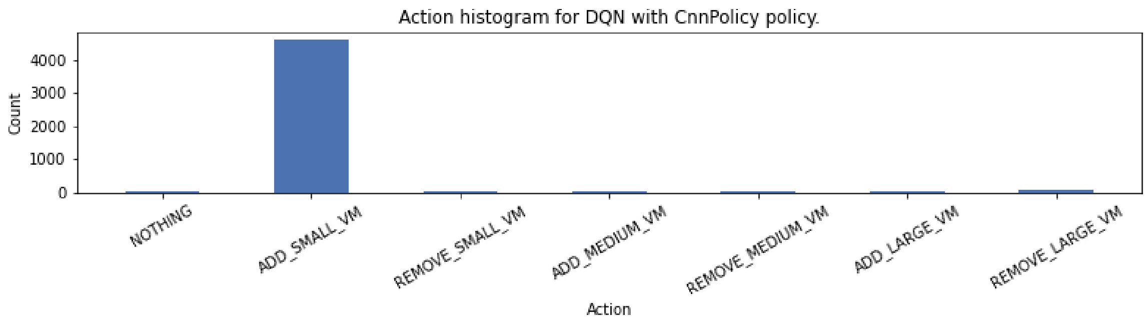

Figure 6.12: Action histogram for DQN model with CNN policy over multiple runs.

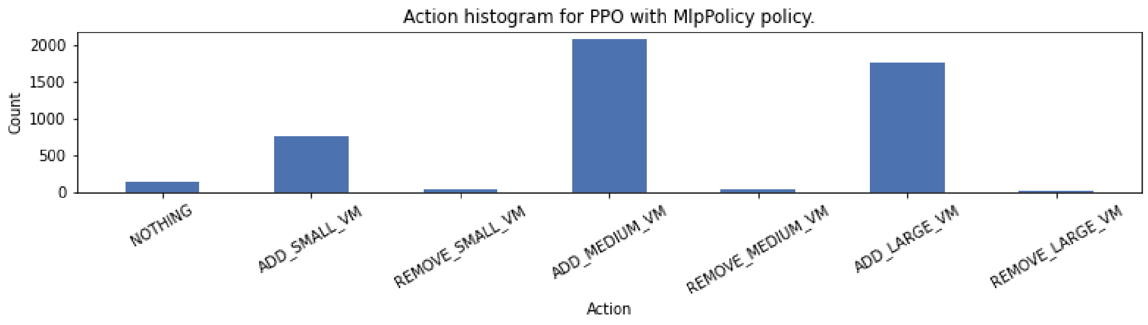

Figure 6.13: Action histogram for PPO model with MLP policy over multiple runs.

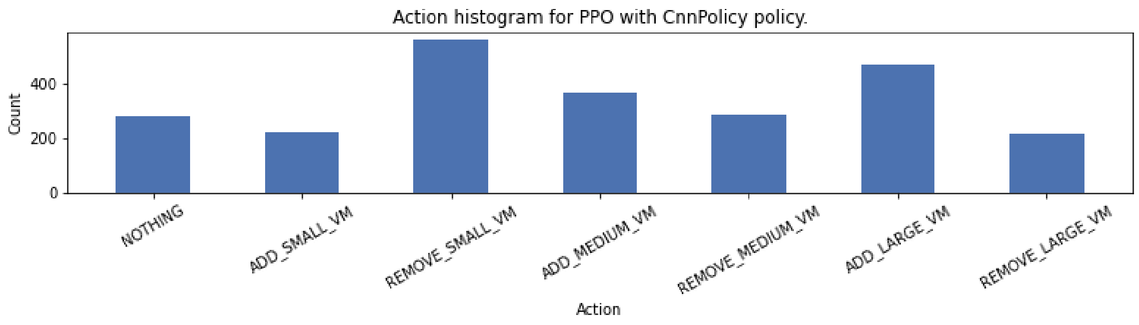

Figure 6.14: Action histogram for PPO model with CNN policy over multiple runs.

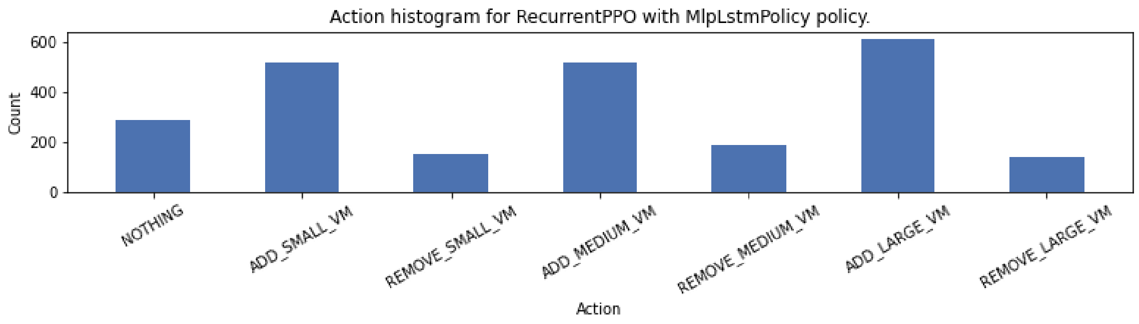

Figure 6.15: Action histogram for Recurrent PPO model with MLP policy over multiple runs.

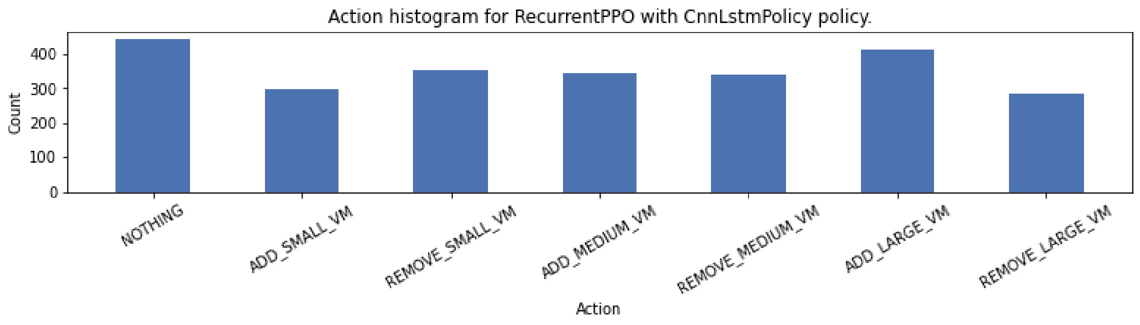

Figure 6.16: Action histogram for Recurrent PPO model with CNN policy over multiple runs.

6.5 Policy summarization

This section will provide insight into the agent policies as a whole using global interpretation by calculating: absolute mean attributions - which can be understood as global feature importance for that policy, and mean positive and negative attributions that explain which features influence the decisions - in a positive or negative manner.

Based on Figure 6.17, DQN agent with MLP architecture bases his predictions mainly on the three features: avgCPUUtilization, vmAllocatedRatio, and p90CPUUtilization.

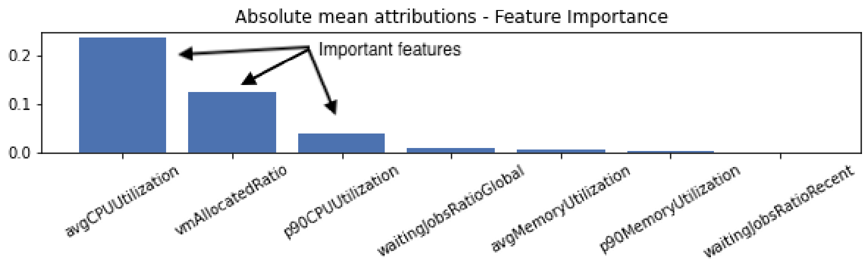

Figure 6.17: Absolute mean attributions - Feature Importance - DQN-MLP.

The feature vmAllocatedRatio attributes mostly negatively and feature avgCPUUtilization attributes mostly positively, as shown in Figure 6.18.

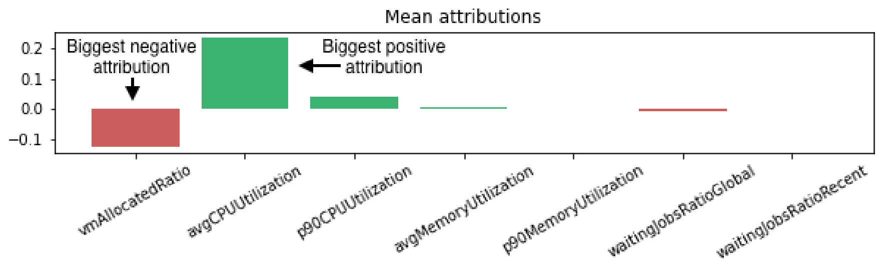

Figure 6.18: Mean attributions - DQN-MLP.

Based on the data displayed in Figures 6.19 and 6.20, we inferred that DQN agent with CNN architecture bases his predictions mainly on the three features: avgCPUUtilization, vmAllocatedRatio, and waitingJobsRatioGlobal in a couple of most recent time steps beside waitingJobsRatioGlobal which is also used from the older time steps.

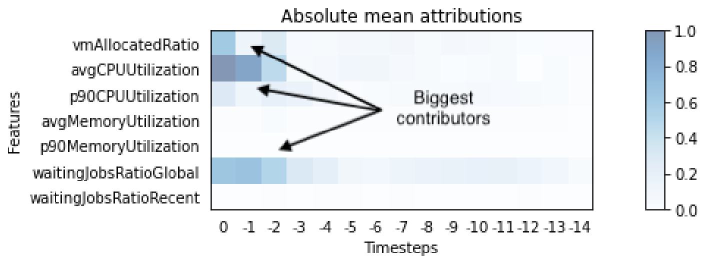

Figure 6.19: Absolute mean attributions - Feature Importance - DQN-CNN.

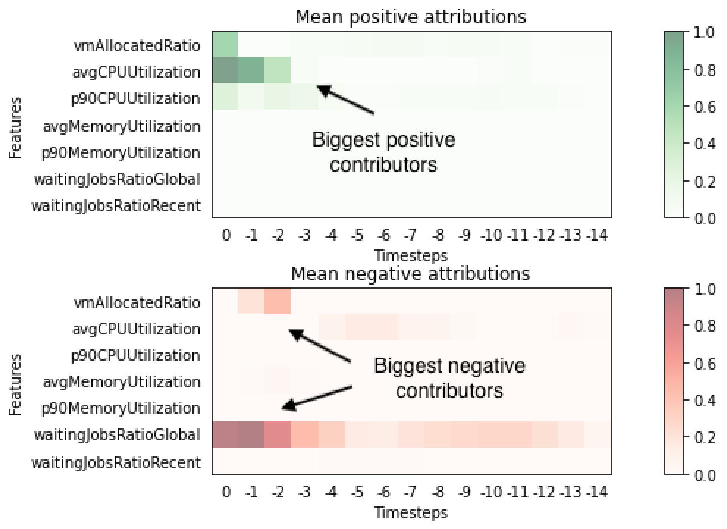

Figure 6.20: Mean attributions - DQN-CNN.

Based on the data displayed in Figures 6.21 and 6.22, we inferred that PPO agent with MLP policy bases his predictions mainly on the three features: waitingJobsRatioGlobal, vmAllocatedRatio, and avgCPUUtilization.

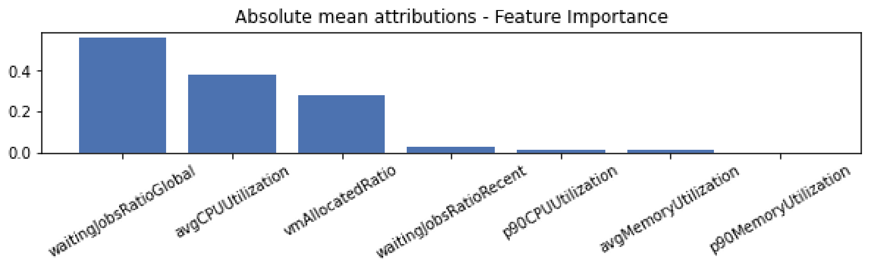

Figure 6.21: Absolute mean attributions - Feature Importance - PPO-MLP.

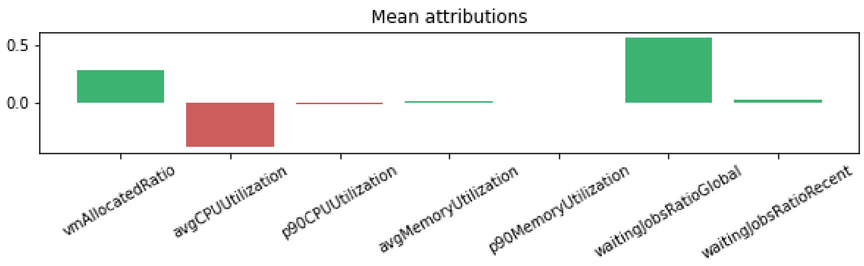

Figure 6.22: Mean attributions - PPO-MLP.

Based on the data displayed in Figures 6.23 and 6.24, we inferred that PPO agent with CNN model architecture focuses mostly on the feature vmAllocatedRatio and also a little bit on the feature waitingJobsRatioGlobal. Positive attributions for vmAllocatedRatio are mostly concentrated in the recent time steps whereas negative attributions are in the older time steps.

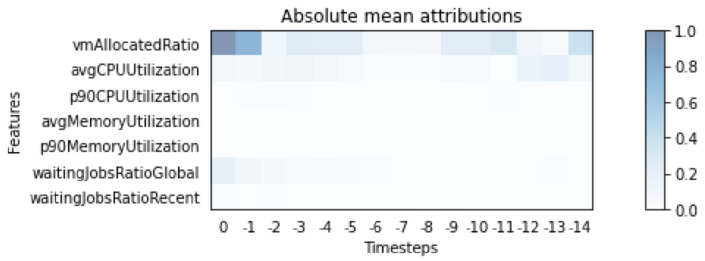

Figure 6.23: Absolute mean attributions - Feature Importance - PPO-CNN.

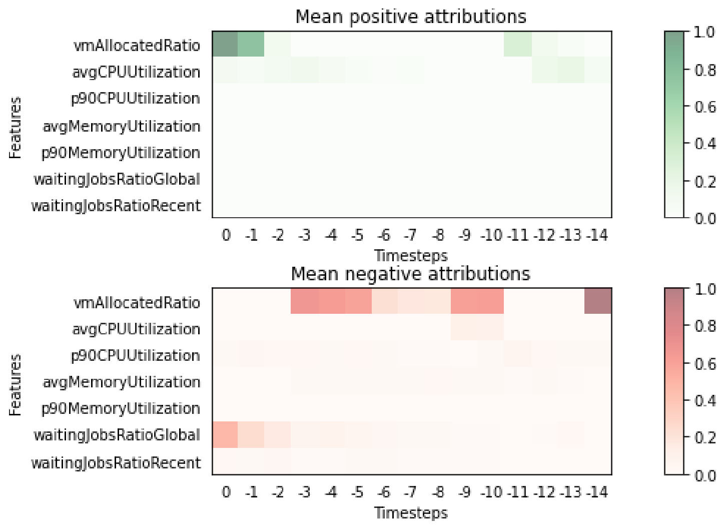

Figure 6.24: Mean attributions - PPO-CNN.

Recurrent PPO agent with MLP policy bases its decision mostly on the feature vmAllocatedRatio as a positive attribution with slight input from the feature avgCPUUtilization as negative attributions, as presented in Figures 6.25, 6.26

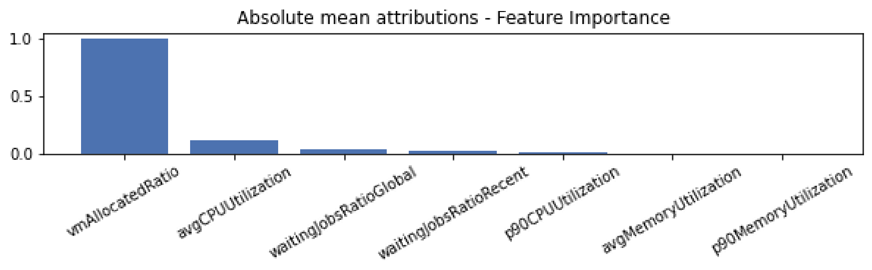

Figure 6.25: Absolute mean attributions - Feature Importance - Recurrent PPO-MLP.

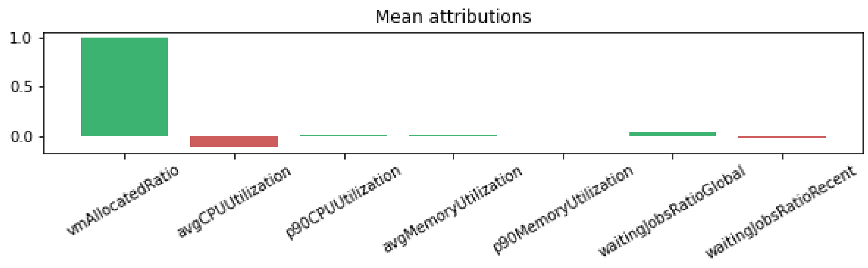

Figure 6.26: Mean attributions - Recurrent PPO-MLP.

PPO agent with CNN architecture has the most balanced feature attributions because it bases its decisions on five out of seven available features across all time steps. The agent bases its decisions mostly on the feature p90CPUUtilization for both positive and negative attributions. It is also visible that negative attribution appears mostly in recent time steps in comparison to positive attributions which appear in older time steps, as presented in Figures 6.27, 6.28

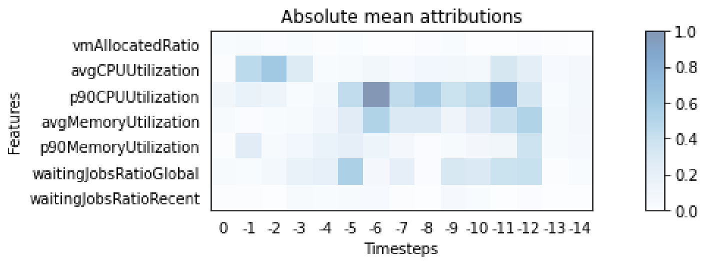

Figure 6.27: Absolute mean attributions - Feature Importance - Recurrent PPO-CNN.

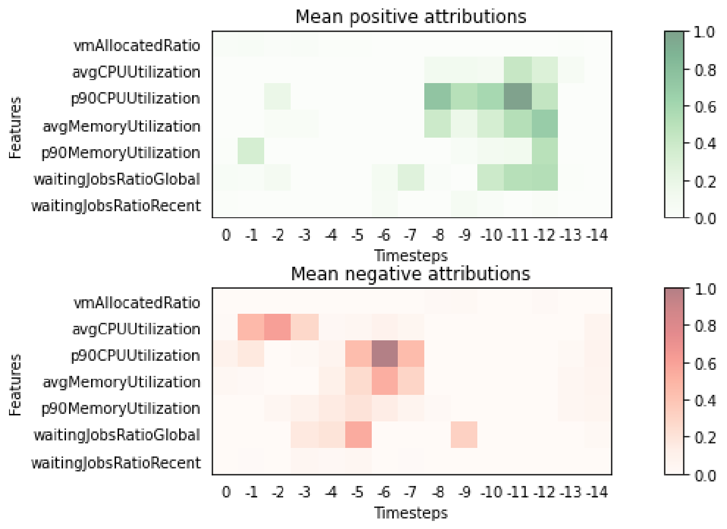

Figure 6.28: Mean attributions - Recurrent PPO-CNN.

6.6 Watching agent learn

This section shows how attribution changes during the learning process. We can see that agents display a significant change in their attributions as training progresses. Insight from this analysis can be super helpful, seeing how and what an agent learns can be used to optimize the training process and the agent's architecture. The figures below will show the attribution changes of the agents with MLP and CNN as feature extractors over a couple of evenly distributed stages of the training processes. The plots are sorted downwards from the model without any training to the final model after the training. We will split the analysis of the agents into two sections. The first section will analyze attribution changes for the fixed observation and the second will analyze mean attributions changes.

6.6.1 Feature attribution changes throughout the training process for the example observation - CNN architecture

In Figure 6.29, the positive attribution changes of the DQN agent with CNN as a feature extractor are shown. We can see that initial model attributes both present features pretty uniformly over the temporal axis. It is clearly visible that the model starts focusing only on the one feature - avgCPUUtilization and at first puts more attention into past time steps but as the training continues it focuses more and more on the fewer and more recent time steps. This development is super insightful for the CNN layers architecture so that in the next iteration of the agent design we can decrease the number of past time steps.

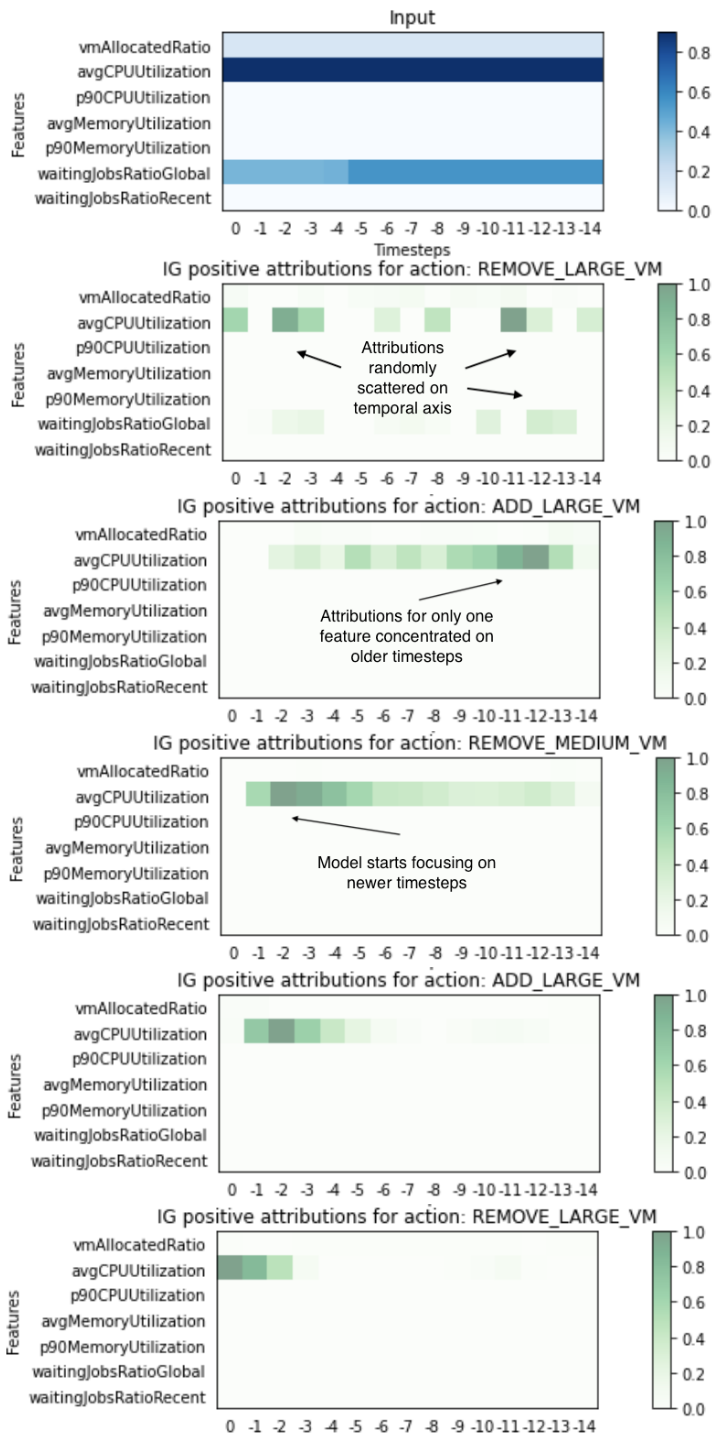

Figure 6.29: Observation and positive attributions throughout the training process of the DQN agent with CNN policy.

In Figure 6.30, the negative attribution changes of the DQN agent with CNN as a feature extractor are shown. We can see that the initial model attributes both present features with low values and pretty uniformly over the temporal axis. During the training, the model starts focusing only on one feature - waitingJobsRationGlobal and at first distributes attention evenly over the temporal axis but as the training continues it focuses more and more on the fewer and more recent time steps.

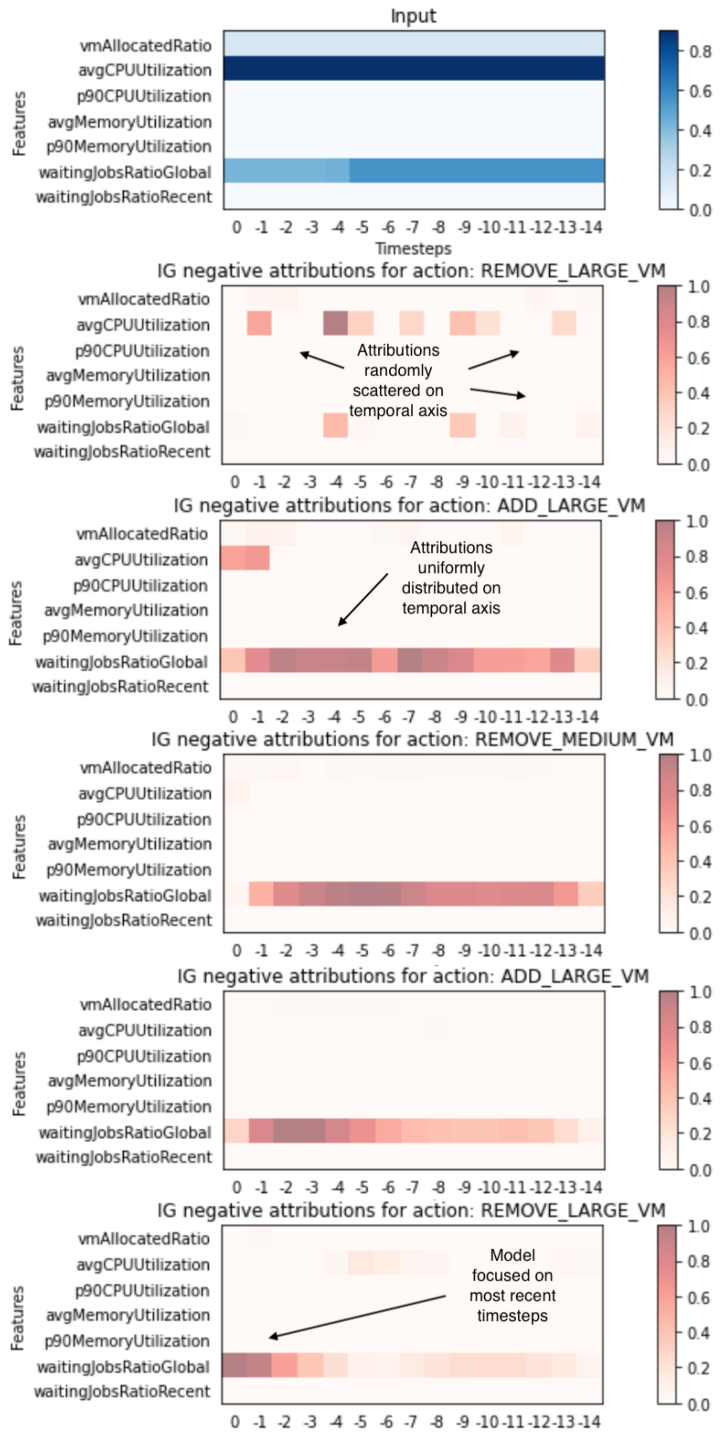

Figure 6.30: Observation and negative attributions throughout the training process of the DQN agent with CNN policy.

6.6.2 Absolute mean attribution changes throughout the training process - CNN architecture.

Absolute mean attribution changes are displayed in Figure 6.31. Firstly, attributions are scattered over multiple features and time steps but during the training, the policy starts to emerge and attributions are present for fewer features - at the end only for three: vmAllocatedRatio, avgCPUUtilization, and waitingJobsRatioGlobal. Policy evolved also in the temporal axis - the agent stopped paying attention to the older time steps and focused on the three most recent time steps instead.

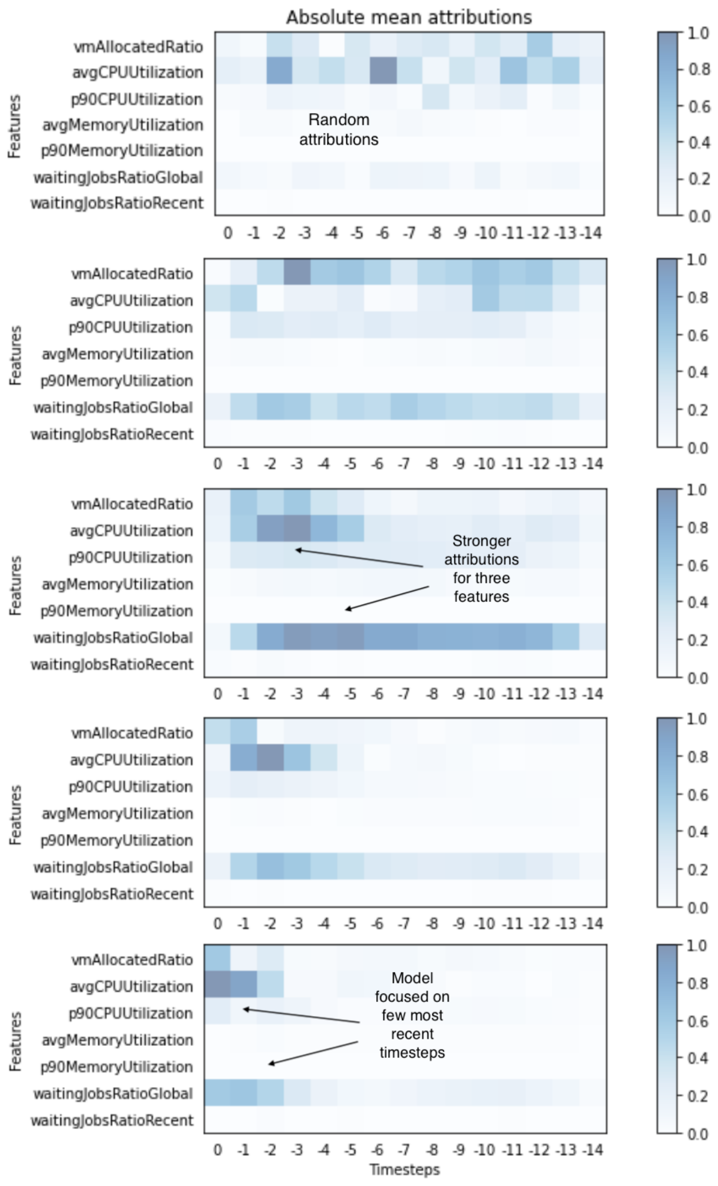

Figure 6.31: Mean absolute attributions throughout the training process of the DQN agent with CNN policy.

6.6.3 Feature attribution changes throughout the training process for the example observation - MLP architecture

In Figure 6.32, the positive and negative feature attributions of the DQN agent with the MLP architecture are shown. During training, the vmAllocatedRatio feature attributions change from highly negative to slightly negative whereas avgCPUUtilization feature attributions change from almost non-existent to highly positive.

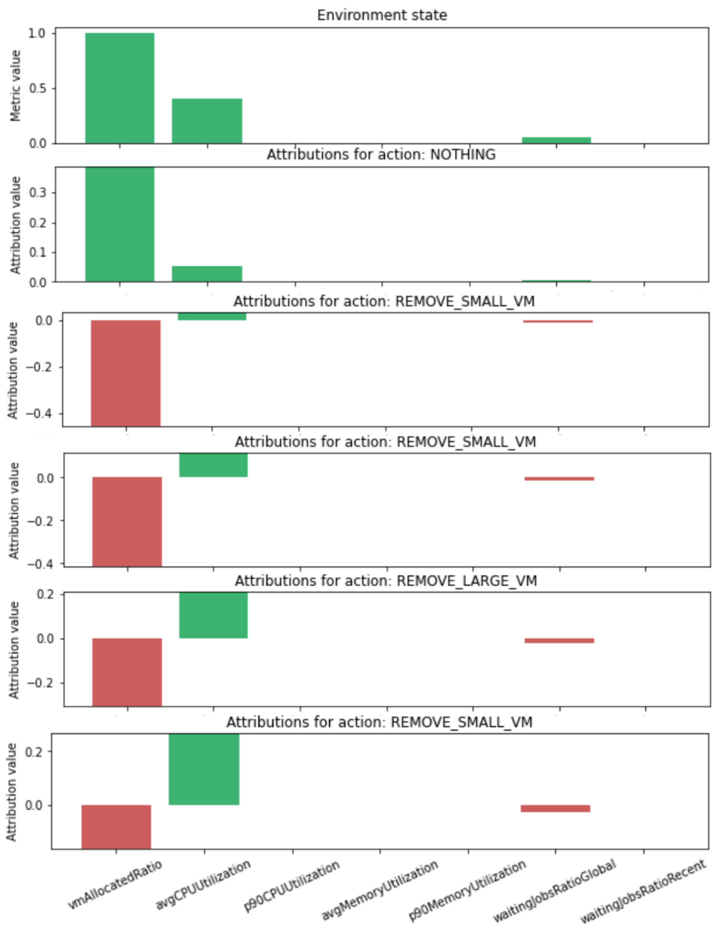

Figure 6.32: Observation, negative and positive attributions throughout the training process of the DQN agent with MLP policy.

6.6.4 Absolute mean attribution changes throughout the training process - MLP architecture.

In Figure 6.33, the absolute mean feature attributions of the DQN agent with the MLP architecture are shown. Initially, the model puts a lot of attention only to one feature - vmAllocatedRatio but during training vmAllocatedRatio feature attributions diminish and the agent starts paying attention to two other features: avgCPUUtilization and waitingJobsRatioGlobal.

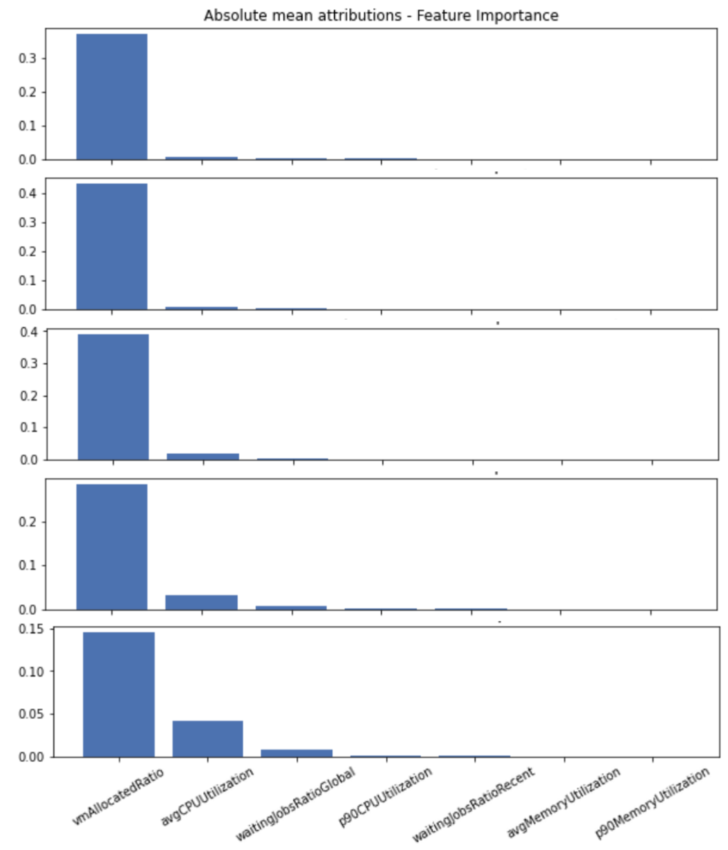

Figure 6.33: Mean absolute attributions throughout the training process of the DQN agent with MLP policy.

6.7 Debugging

The interpretability component can be used to debug an agent's decision for certain example observations in order to understand why that decision was made. One case would be to see the reasoning behind the right decision, was that decision based on right or wrong reasons? Also, it would be great to see why the agent made a bad decision, and which feature values prompted him to that decision. The examples being analyzed will be split into two groups: examples where the right decision was made and examples where the wrong decision was made.

6.7.1 Right decisions

First example - right decision for right reasons - PPO CNN

In Figure 6.34 we can find attributions behind the chosen action - removing a large VM, which at the time was subjectively the right decision. Looks like the decision was based solely on the feature - vmAllocatedRatio which had a very high value meaning that almost all of the available VMs were being used while the values of all the other features were zero or close to zero. An especially important feature to look at in this example is waitingJobsRatioGlobal which shows how many jobs are in the queue, in this example there were very few jobs in the queue and negative attributions reflect that. To sum up, the decision to remove large VM was right because at the time a lot of VMs were being used but there were almost no jobs waiting to be executed.

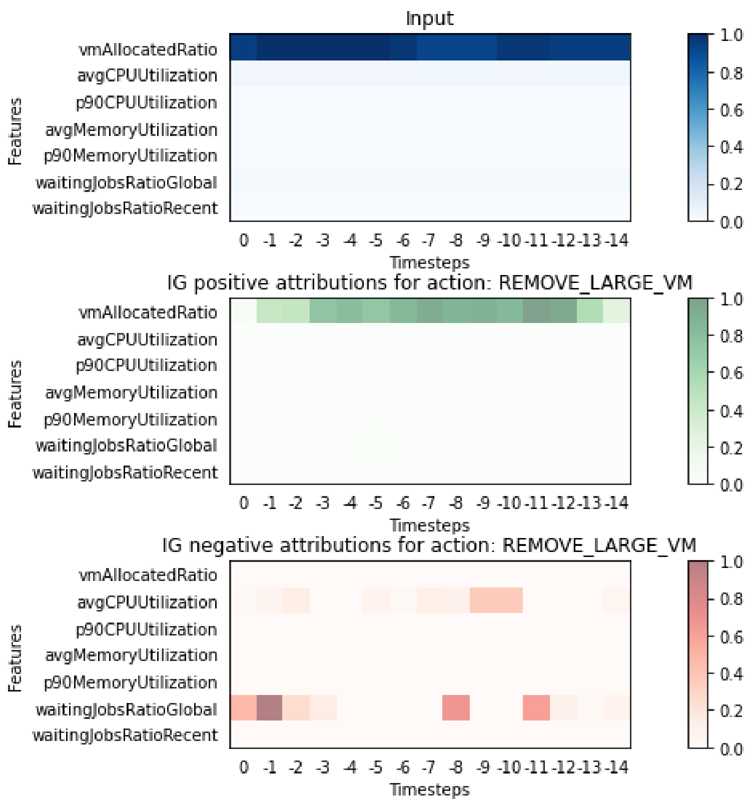

Figure 6.34: Attributions for example observation - PPO agent with CNN architecture.

Second example - right decision for right reasons - PPO MLP

In Figure 6.35 we can see attributions behind the chosen action - adding a small VM, which at the time was subjectively the right decision. The decision was based on the value of two features: vmAllocatedRatio and avgCPUUtilization. The CPU utilization was around 0.5 which means that it wasn't high enough to add a medium or large VM but it was high enough to add a small VM. The agent also recognized that almost all of the available VMs were being used which could explain negative attribution for feature vmAllocatedRatio, the value was very small as the agent does not receive penalties for using all of the available VMs.

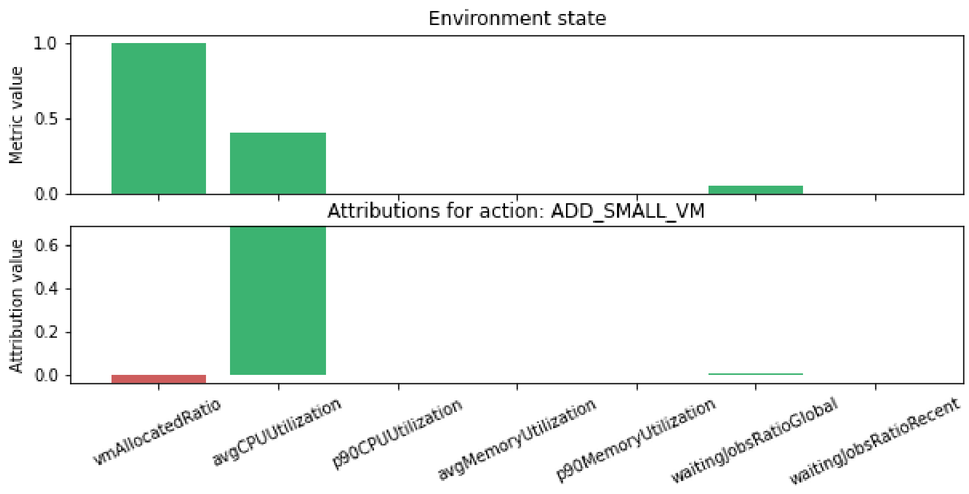

Figure 6.35: Attributions for example observation - PPO agent with MLP architecture.

Third example - the right decision for wrong reasons - PPO CNN

In Figure 6.36 we can find attributions behind the chosen action - removing a large VM, which at the time was subjectively the right decision but made for the wrong reasons. The decision was based on the value of feature p90CPUUtilization from two to four-time steps from the past. The value of the CPU utilization was very high at that time so so it logically doesn't make sense to remove a VM, especially a large VM based on these reasons. It could be logical to remove a VM based on the last two-time steps where CPU utilization decreased.

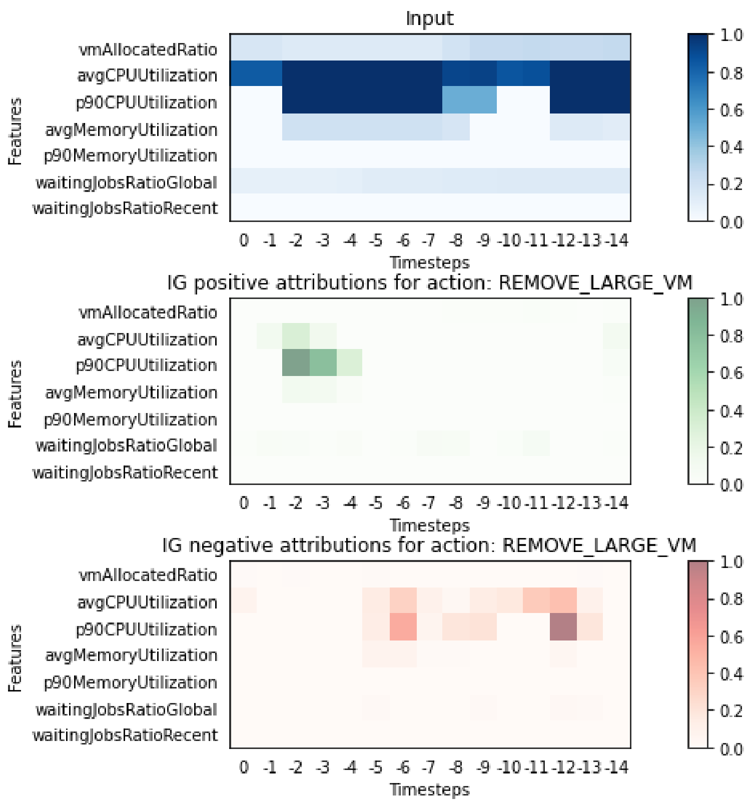

Figure 6.36: Attributions for example observation - PPO agent with CNN architecture.

6.7.2 Wrong decisions

In Figure 6.37 we can see attributions behind the chosen action - removing medium VM, which at the time was subjectively the wrong decision. The decision was based on the value of feature avgCPUUtilization whose value is at the maximum which should result in the decision to add VM instances, not to remove them.

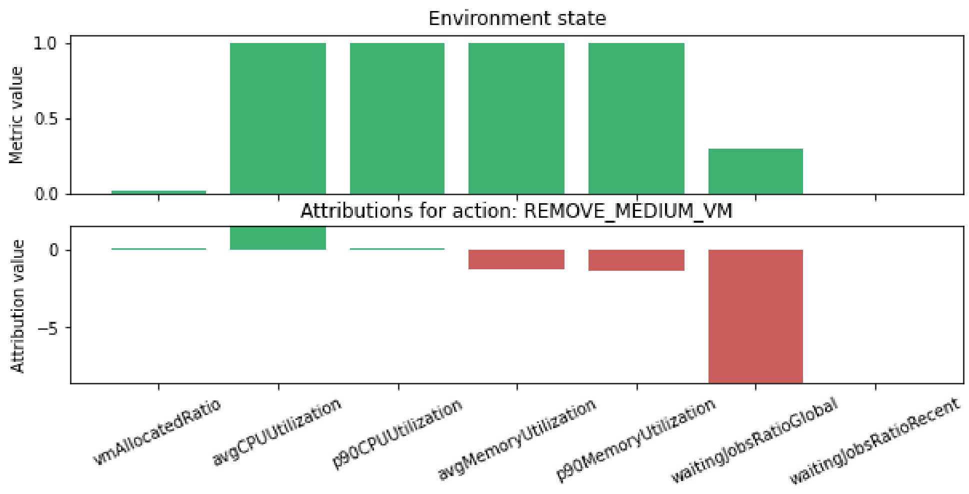

Figure 6.37: Attributions for example observation - Recurrent PPO agent with MLP architecture.

6.8 Feature selection based on feature attribution

Feature attributions as a result of global interpretation of the model can be used in order to select only the features that attribute to the decision of the agent. As long as the performance of the agent doesn't diminish, there are a lot of advantages of using fewer features:

- simpler and smaller model
- easier interpretation of the agent's decisions
- faster training time

The comparison between two models with different numbers of features was performed for the Recurrent PPO agent with MLP architecture. In Figure 6.38 we can find mean attributions for the Recurrent PPO-MLP agent which demonstrates that the feature waitingJobsRatioRecent is almost not being used by the agent so the lack of it should not cause a performance drop. In order to decide whether feature attributions are a good metric for feature selection, the feature waitingJobsRatioRecent was removed from the environment and the agent was trained on the modified environment.

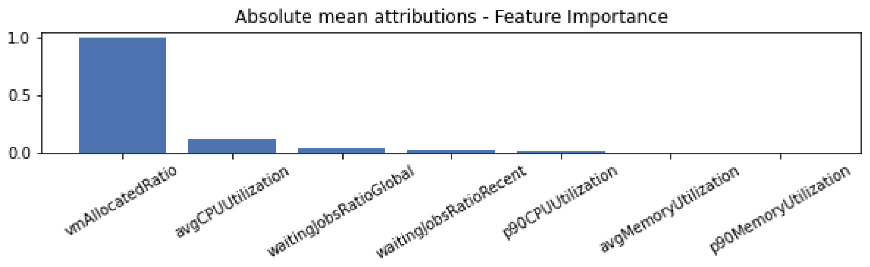

Figure 6.38: Absolute mean attributions - Feature Importance - Recurrent PPO-MLP.

In Figure 6.39, we can see absolute mean attributions for the agent without the feature waitingJobsRatioRecent and it shows that the distribution of the attributions is very similar to the original agent trained with all the available features.

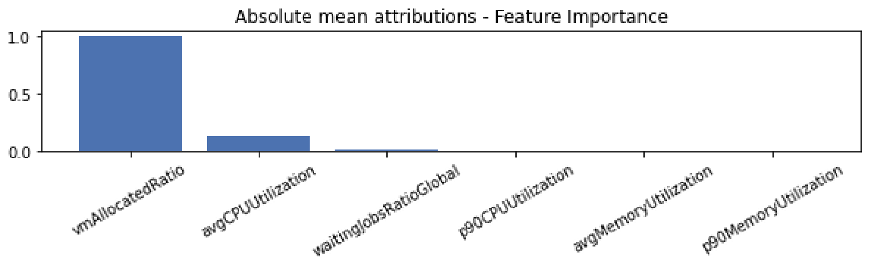

Figure 6.39: Absolute mean attributions - Feature Importance - Recurrent PPO-MLP.

In table 6.1, the performance of both agents on the training and test workload is displayed. It shows that the agent trained on all features achieved the same performance as the agent trained without the feature waitingJobsRatioRecent and thus proving that XRL techniques such as IG can be used successfully for feature selection of DRL agents.

	Agent with all features	Agent without the feature waitingJobsRatioRecent
Training workload mean reward	-6.14	-6.16
Test workload mean reward	-11.84	-11.84

Table 6.1: Comparison between the performance of the agent trained on all features and the agent trained without the feature waitingJobsRatioRecent.

6.9 Summary

This chapter presented the results of the training and evaluation of the multiple agents that were created. Subsequently, it showed the multi-step analysis based on the actions of the agents - statistical analysis and interpretability-based analysis. The conclusions inferred from these analyses were the base of the thesis achievements section of the following Chapter 7.

Chapter 7

Conclusions and future work

7.1 Thesis Achievements

We have built several DRL agents, two agents trained with DQN algorithm consisting of a DNN model with MLP or CNN architecture and also four agents trained with PPO algorithm having MLP, CNN, recurrent with MLP feature extractor or recurrent with CNN feature extractor architectures. We have performed several analyses aimed at a better understanding of what the DRL agents used to optimally manage resources of the cloud computing platform does and why.

Starting with the classical, statistical analysis such as looking at how the agent interacts with the spikes in the workload, we were able to differentiate between the proactive and reactive agents. A reactive agent such as DQN-MLP was only able to react to the increase in the workload and with some delay. On the other hand, proactive agents are able to predict the spikes and provide the necessary computing resources ahead or just in time. We were also able to discern between agents which increase or decrease the computing resources for the application aggressively or slowly.

Upon evaluation of the agents, it can be said that all of the agents besides the DQN-MLP agent achieved similar performance on both training and testing workload which was better than the simple, rule-based baseline.

We also performed the analysis of the action histograms of the agents and found that

PPO-based agents tend to have more balanced action histograms than the DQN agents which can be attributed to the sampling of the actions from the parameterized distribution in the PPO agents.

We have applied interpretability techniques in order to understand the reasoning behind the decisions of the DRL agent. The XRL algorithm applied - IG, produces saliency maps in a form of 1D feature attributions for MLP architectures of the model or 2D (features x time) feature attribution map for CNN architectures. We managed to create agent policy summarization based on the absolute mean attributions or positive/negative mean attributions. We were able to infer which features agents bases their decisions on and the magnitude and direction of the impact of the attribution.

Watching an agent learn provides a lot of insightful information into the direction of the learning process, changes between the consecutive iterations of the agent, and prove that the agent is indeed learning useful things. Let's take the DQN-CNN agent as an example, at first, the agent used a couple of features from a wide range of time steps to make its decisions but during the training, the policy started to crystallize as the agent found the three main features on which it based its decisions and later on limit the temporal impact of the features to the few most recent time steps.

Feature attributions are also an excellent tool that can be used to debug the agent's predictions. We can analyze why the agent made the wrong decisions or even find examples where the agent made the right decision but based it on the wrong reasons thus making the agent less believable.

Feature selection is an important part of deep learning and by using absolute mean attributions we were able to successfully remove from the environment the feature, that had a small impact on the agent's decision and showed that there was no performance drop in the agent trained without that feature while making the training faster and the agent simpler, smaller and easier to understand.

7.2 Lessons learnt

There are a lot of lessons to be learned from this Thesis. Mainly, it shows that the DRL agents, trained with DQN algorithm and MLP and CNN architectures and also agents trained with

PPO algorithm and MLP, CNN, recurrent MLP, recurrent CNN architectures, can optimally manage resources of the cloud computing platform while their actions can be understood by the human without expert ML knowledge. We have learned that statistical analysis can provide a lot of information about how the agents work and help with differentiating them and finding ones that suit our needs the most. We have also discovered that XAI technique IG can be successfully applied to perform post hoc interpretability of the agent actions which results in 1D saliency map for MLP architecture and 2D saliency map for CNN architecture. The 1D saliency map provides a small amount of information and is easy to interpret while 2D saliency map (features x time steps) provides much more information but is more difficult to interpret. DRL agent explainability proves very useful in feature selection and the debugging of the agent. The last thing we learned is that increase in the DRL agent's architecture complexity doesn't necessarily cause an increase in the performance.

7.3 Future work

There are a lot of directions for further research and improvements. Regarding the performance improvements, one could perform more precise and robust hyperparameters optimization than the short sensitivity analysis that was performed. Feature selection could also be taken to the next level by adding a lot of new environment metrics and based on the feature attributions selecting the most influential features. Another useful thing to add would be an online, real-time decision support system that would assist the human operator of the computing platform. The next research steps could include using different DNN explainability techniques or the usage of self-explanatory DNN architectures such as attention models [55].

List of Algorithms

Glossary of Acronyms

1D one-dimensional.

2D two-dimensional.

A3C Asynchronous-Advantage-Actor-Critic.

AI Artificial Intelligence.

AWS Amazon Web Services.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

CSP Cloud Service Provider.

DDQN Double Deep Q-Networks.

DL Deep Learning.

DNN Deep Neural Network.

DQN Deep Q-Networks.

DRL Deep Reinforcement Learning.

FNN Feedforward Neural Network.

GDPR	General Data Protection Right.
IaaS	Infrastructure as a Service.
IG	Integrated Gradients.
ML	Machine Learning.
MLP	MultiLayer Perceptron.
MNIST	Modified National Institute of Standards and Technology.
MSE	Mean Square Error.
NLP	Natural Language Processing.
PaaS	Platform as a Service.
PPO	Proximal Policy Optimization.
QoS	Quality-of-Service.
R2N2	Residual Recurrent Neural Network.
RAM	Random-Access Memory.
ReLU	Rectified Linear Unit.
RL	Reinforcement Learning.
RNN	Recurrent Neural Network.
SaaS	Software as a Service.
SGD	Stochastic Gradient Descent.
SHAP	SHapley Additive exPlanations.

SLA	Service-Level-Agreement.
TD	Temporal Difference.
TRPO	Trust Region Policy Optimization.
UI	User Interface.
VM	Virtual Machine.
VPG	Vanilla Policy Gradient.
XAI	Explainable Artificial Intelligence.
XRL	Explainable Reinforcement Learning.

Glossary of Terms

A3C Deep reinforcement learning algorithm that uses an actor-critic neural network architecture. This algorithm learns asynchronously by using multiple agents (each agent has its own network parameters and a copy of the environment) [21].

attention map A scalar matrix representing the relative importance of layer activations at different 2D spatial locations with respect to the target task [10].

attention model Attention model is a deep neural network with the attention mechanism that enables to dynamically highlight relevant features of the input data [15].

explainability Explainability, also known as "interpretability", is the concept that an ML model and its predictions can be explained to the average human, without expert ML knowledge in a way so that he understands the reasoning of how the model went from input features to the prediction.

feature A feature is a measurable property, usually of the input data which is fed into the ML model. An example of a feature would be a single metric of the computing environment such as CPU utilization.

feature attributions Feature attributions are values that demonstrate how much each input data feature contributed to the predictions of the model for each given instance.

IaaS Type of cloud computing service that offers compute, storage, and networking resources.

ML Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [12].

PaaS Type of cloud computing service that offers complete development and deployment environment in the cloud.

percentile A measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall [1].

QoS Measurement of the overall performance of a cloud computing service from the viewpoint of the user.

R2N2 RNN with the residual connection that provides additional spatial shortcut path by skipping some layers.

SaaS Type of cloud computing service that provides a complete software solutions.

saliency map The saliency map displays the impact of the input features on the predictions of the ML model. 1D saliency map shows the contributions of each feature value to the prediction and the 2D saliency map shows the contributions of each pixel to the prediction.

SLA A commitment between a service provider and a client which describes aspects of the service such as quality, availability and responsibilities.

Bibliography

[1] Percentile. In: *IAHPC Pallipedia*. URL https://pallipedia.org/percentile/.

[2] Adadi A., Berrada M.: Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). In: *IEEE Access*, vol. 6, pp. 52138–52160, 2018. URL http://dx.doi.org/10.1109/ACCESS.2018.2870052.

[3] Anuradha V.P., Sumathi D.: A survey on resource allocation strategies in cloud computing. In: *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pp. 1–7. 2014. URL http://dx.doi.org/10.1109/ICICES.2014.7033931.

[4] Barredo Arrieta A., Díaz-Rodríguez N., Del Ser J., Bennetot A., Tabik S., Barbado A., Garcia S., Gil-Lopez S., Molina D., Benjamins R., Chatila R., Herrera F.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. In: *Information Fusion*, vol. 58, pp. 82–115, 2020. ISSN 1566-2535. URL http://dx.doi.org/https://doi.org/10.1016/j.inffus.2019.12.012.

[5] Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., Zaremba W.: OpenAI Gym. In: , 2016.

[6] Cheng M., Li J., Nazarian S.: DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. pp. 129–134. 2018. URL http://dx.doi.org/10.1109/ASPDAC.2018.8297294.

[7] Cobbe K., Klimov O., Hesse C., Kim T., Schulman J.: Quantifying Generalization in Reinforcement Learning, 2018. URL `http://dx.doi.org/10.48550/ARXIV.1812.02341`.

[8] Cuayáhuitl H.: SimpleDS: A Simple Deep Reinforcement Learning Dialogue System, 2016. URL `http://dx.doi.org/10.48550/ARXIV.1601.04574`.

[9] Deng L.: The mnist database of handwritten digit images for machine learning research. In: *IEEE Signal Processing Magazine*, vol. 29(6), pp. 141–142, 2012.

[10] Draelos R.: Learn to Pay Attention! Trainable Visual Attention in CNNs. In: *Towards Data Science*. URL `https://towardsdatascience.com/learn-to-pay-attention-trainable-visual-attention-in-cnns-87e2869f89f1`.

[11] Dutta S., Gera S., Verma A., Viswanathan B.: SmartScale: Automatic Application Scaling in Enterprise Clouds. In: *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 221–228. 2012. URL `http://dx.doi.org/10.1109/CLOUD.2012.12`.

[12] Education I.C.: Machine Learning. In: *IBM*. URL `https://www.ibm.com/cloud/learn/machine-learning`.

[13] Funika W., Koperek P.: *Evaluating the Use of Policy Gradient Optimization Approach for Automatic Cloud Resource Provisioning*, pp. 467–478. 2020. ISBN 978-3-030-43228-7. URL `http://dx.doi.org/10.1007/978-3-030-43229-4_40`.

[14] Funika W., Koperek P., Kitowski J.: *Automatic Management of Cloud Applications with Use of Proximal Policy Optimization*, pp. 73–87. 2020. ISBN 978-3-030-50370-3. URL `http://dx.doi.org/10.1007/978-3-030-50371-0_6`.

[15] Galassi A., Lippi M., Torroni P.: Attention in Natural Language Processing. In: *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32(10), pp. 4291–4308, 2021. URL `http://dx.doi.org/10.1109/tnnls.2020.3019893`.

[16] Gandhi A., Dube P., Karve A., Kochut A., Zhang L.: Modeling the Impact of Workload on Cloud Resource Scaling. In: *2014 IEEE 26th International Symposium on Computer*

Architecture and High Performance Computing, pp. 310–317. 2014. URL http://dx.doi.org/10.1109/SBAC-PAD.2014.16.

[17] Goodfellow I.J., Bengio Y., Courville A.: *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[18] Gregurić M., Vujić M., Alexopoulos C., Miletić M.: Application of Deep Reinforcement Learning in Traffic Signal Control: An Overview and Impact of Open Traffic Data. In: *Applied Sciences*, vol. 10(11), 2020. ISSN 2076-3417. URL http://dx.doi.org/10.3390/app10114011.

[19] Greydanus S., Koul A., Dodge J., Fern A.: Visualizing and Understanding Atari Agents, 2017. URL http://dx.doi.org/10.48550/ARXIV.1711.00138.

[20] Guo W., Wu X., Khan U., Xing X.: EDGE: Explaining Deep Reinforcement Learning Policies. In: M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan, eds., *Advances in Neural Information Processing Systems*, vol. 34, pp. 12222–12236. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/65c89f5a9501a04c073b354f03791b1f-Paper.pdf.

[21] Gupta A.: Asynchronous Advantage Actor Critic (A3C) algorithm. In: *Geeksforgeeks*. URL https://www.geeksforgeeks.org/asynchronous-advantage-actor-critic-a3c-algorithm/.

[22] van Hasselt H., Guez A., Silver D.: Deep Reinforcement Learning with Double Q-learning, 2015. URL http://dx.doi.org/10.48550/ARXIV.1509.06461.

[23] Heuillet A., Couthouis F., Rodríguez N.D.: Explainability in Deep Reinforcement Learning. In: *CoRR*, vol. abs/2008.06693, 2020. URL https://arxiv.org/abs/2008.06693.

[24] Hickling T., Zenati A., Aouf N., Spencer P.: Explainability in Deep Reinforcement Learning, a Review into Current Methods and Applications, 2022. URL http://dx.doi.org/10.48550/ARXIV.2207.01911.

[25] Hilton J., Cammarata N., Carter S., Goh G., Olah C.: Understanding RL Vision. In: *Distill*, 2020. URL `http://dx.doi.org/10.23915/distill.00029`. Https://distill.pub/2020/understanding-rl-vision.

[26] Kahn J.: Artificial Intelligence Has Some Explaining to Do Software makers offer more transparent machine-learning tools—but there's a trade-off. In: *Bloomberg*. URL `https://www.bloomberg.com/news/articles/2018-12-12/artificial-intelligence-has-some-explaining-to-do#xj4y7vzkgn`.

[27] Kokhlikyan N., Miglani V., Martin M., Wang E., Alsallakh B., Reynolds J., Melnikov A., Kliushkina N., Araya C., Yan S., Reblitz-Richardson O.: Captum: A unified and generic model interpretability library for PyTorch, 2020. URL `http://dx.doi.org/10.48550/ARXIV.2009.07896`.

[28] LeCun Y., Boser B.E., Denker J.S., Henderson D., Howard R.E., Hubbard W.E., Jackel L.D.: Backpropagation Applied to Handwritten Zip Code Recognition. In: *Neural Comput.*, vol. 1(4), pp. 541–551, 1989. URL `http://dx.doi.org/10.1162/neco.1989.1.4.541`.

[29] Lundberg S., Lee S.I.: A unified approach to interpreting model predictions. In: *CoRR*, vol. abs/1705.07874, 2017. URL `http://dblp.uni-trier.de/db/journals/corr/corr1705.html#LundbergL17`.

[30] Manela B.: Deep Reinforcement Learning for Complex Manipulation Tasks with Sparse Feedback, 2020. URL `http://dx.doi.org/10.48550/ARXIV.2001.03877`.

[31] Marsland S.: *Machine Learning - An Algorithmic Perspective.* Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press, 2009. ISBN 978-1-4200-6718-7.

[32] Milani S., Topin N., Veloso M., Fang F.: A Survey of Explainable Reinforcement Learning, 2022. URL `http://dx.doi.org/10.48550/ARXIV.2202.08434`.

[33] Mitsopoulos K., Somers S., Schooler J., Lebiere C., Pirolli P., Thomson R.: Toward a Psychology of Deep Reinforcement Learning Agents Using a Cognitive Architecture.

In: *Topics in Cognitive Science*, vol. n/a(n/a). URL http://dx.doi.org/https://doi.org/10.1111/tops.12573.

[34] Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M.: Playing Atari with Deep Reinforcement Learning. In: , 2013. URL http://arxiv.org/abs/1312.5602. Cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

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

[36] Mott A., Zoran D., Chrzanowski M., Wierstra D., Rezende D.J.: Towards Interpretable Reinforcement Learning Using Attention Augmented Agents, 2019. URL http://dx.doi.org/10.48550/ARXIV.1906.02500.

[37] Olah C., Satyanarayan A., Johnson I., Carter S., Schubert L., Ye K., Mordvintsev A.: The Building Blocks of Interpretability. In: *Distill*, 2018. URL http://dx.doi.org/10.23915/distill.00010. Https://distill.pub/2018/building-blocks.

[38] OpenAI, Akkaya I., Andrychowicz M., Chociej M., Litwin M., McGrew B., Petron A., Paino A., Plappert M., Powell G., Ribas R., Schneider J., Tezak N., Tworek J., Welinder P., Weng L., Yuan Q., Zaremba W., Zhang L.: Solving Rubik's Cube with a Robot Hand, 2019. URL http://dx.doi.org/10.48550/ARXIV.1910.07113.

[39] Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, eds., *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[40] Raffin A., Hill A., Ernestus M., Gleave A., Kanervisto A., Dormann N.: Stable Baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[41] Ribeiro M.T., Singh S., Guestrin C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, p. 1135–1144. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 9781450342322. URL http://dx.doi.org/10.1145/2939672.2939778.

[42] Rittinghouse J.W.: *Cloud computing : implementation, management, and security / John W. Rittinghouse, James F. Ransome.* CRC Press, Boca Raton, 1st edition ed., 2009. ISBN 1-351-61536-X.

[43] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning Representations by Back-propagating Errors. In: *Nature*, vol. 323(6088), pp. 533–536, 1986. URL http://dx.doi.org/10.1038/323533a0.

[44] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning Representations by Back-propagating Errors. In: *Nature*, vol. 323(6088), pp. 533–536, 1986. URL http://dx.doi.org/10.1038/323533a0.

[45] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O.: Proximal Policy Optimization Algorithms, 2017. URL http://dx.doi.org/10.48550/ARXIV.1707.06347.

[46] Selvaraju R.R., Cogswell M., Das A., Vedantam R., Parikh D., Batra D.: Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In: *International Journal of Computer Vision*, vol. 128(2), pp. 336–359, 2019. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

[47] Shalev-Shwartz S., Ben-David S.: *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press, 2014. ISBN 978-1-10-705713-5.

[48] Campos da Silva Filho M., Oliveira R., Monteiro C., Inácio P., Freire M.: CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. pp. 400–406. 2017. URL `http://dx.doi.org/10.23919/INM.2017.7987304`.

[49] Silver D., Huang A., Maddison C., Guez A., Sifre L., Driessche G., Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., Dieleman S., Grewe D., Nham J., Kalchbrenner N., Sutskever I., Lillicrap T., Leach M., Kavukcuoglu K., Graepel T., Hassabis D.: Mastering the game of Go with deep neural networks and tree search. In: *Nature*, vol. 529, pp. 484–489, 2016. URL `http://dx.doi.org/10.1038/nature16961`.

[50] Sundararajan M., Taly A., Yan Q.: Axiomatic Attribution for Deep Networks, 2017. URL `http://dx.doi.org/10.48550/ARXIV.1703.01365`.

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

[52] Tighe M., Bauer M.: Integrating cloud application autoscaling with dynamic VM allocation. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9. 2014. URL `http://dx.doi.org/10.1109/NOMS.2014.6838239`.

[53] Tighe M., Bauer M.: Topology and Application Aware Dynamic VM Management in the Cloud. In: *Journal of Grid Computing*, vol. 15, 2017. URL `http://dx.doi.org/10.1007/s10723-017-9397-z`.

[54] Tuli S., Ilager S., Ramamohanarao K., Buyya R.: Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks. In: *IEEE Transactions on Mobile Computing*, vol. 21(3), pp. 940–954, 2022. URL `http://dx.doi.org/10.1109/tmc.2020.3017079`.

[55] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I.: Attention Is All You Need, 2017. URL `http://dx.doi.org/10.48550/ARXIV.1706.03762`.

[56] Vennam S.: Cloud Computing. In: *IBM*. URL `https://www.ibm.com/cloud/learn/cloud-computing`.

[57] Vinyals O., Babuschkin I., Czarnecki W.M., Mathieu M., Dudzik A., Chung J., Choi D.H., Powell R., Ewalds T., Georgiev P., Oh J., Horgan D., Kroiss M., Danihelka I., Huang A., Sifre L., Cai T., Agapiou J.P., Jaderberg M., Vezhnevets A.S., Leblond R., Pohlen T., Dalibard V., Budden D., Sulsky Y., Molloy J., Paine T.L., Gulcehre C., Wang Z., Pfaff T., Wu Y., Ring R., Yogatama D., Wünsch D., McKinney K., Smith O., Schaul T., Lillicrap T.P., Kavukcuoglu K., Hassabis D., Apps C., Silver D.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. In: *Nature*, pp. 1–5, 2019.

[58] Wang Z., Gwon C., Oates T., Iezzi A.: Automated Cloud Provisioning on AWS using Deep Reinforcement Learning, 2017. URL `http://dx.doi.org/10.48550/ARXIV.1709.04305`.

[59] Xie Z., Berseth G., Clary P., Hurst J., van de Panne M.: Feedback Control For Cassie With Deep Reinforcement Learning, 2018. URL `http://dx.doi.org/10.48550/ARXIV.1803.05580`.

[60] Zhang Y., Yao J., Guan H.: Intelligent Cloud Resource Management with Deep Reinforcement Learning. In: *IEEE Cloud Computing*, vol. 4(6), pp. 60–69, 2017. URL `http://dx.doi.org/10.1109/MCC.2018.1081063`.