Explaining the role of reward functions in deep reinforcement learning

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Faculty of Engineering

BY

Sambit Ghosh

Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012 (INDIA)

June, 2022

Declaration of Originality

I, **Sambit Ghosh**, with SR No. **17868** hereby declare that the material presented in the thesis titled

Explaining the role of reward functions in deep reinforcement learning

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2020-22**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date:Wednesday 29th June, 2022 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

[illegible]

Advisor Name: Dr. Shalabh Bhatnagar Advisor Signature

DEDICATED TO

My Family, Friends, Professors, and Mentors

Acknowledgements

My deepest gratitude is to my advisor, Prof. Shalabh Bhatnagar. I am very much fortunate to have an advisor who gave me the freedom to explore on our own, and at the same time the guidance to recover when I stuck somewhere. A very special thanks to Dr. Raghu Ram Bharadwaj D for the discussions and guidance throughout the project. Finally we would like to thank our family of CSA and IISc.

Abstract

Deep Learning (DL) has played an influential role in the success of many Artificial Intelligence (AI) applications like image classification and computer vision. Despite its popularity, it has been mainly perceived as a black box that was not interpretable. In recent times, researchers have focused on decoding this black box which has led to the area of Explainability. The objective here is to develop algorithms to understand "why the DL agent does what it does?". This amounts to answering questions like "what are the salient input features for making an optimal decision?". Reinforcement Learning (RL) deals with algorithms for sequential decision-making problems. RL combined with neural networks, i.e., Deep Reinforcement Learning (DRL), has achieved remarkable success in many human challenging domains like Chess, Go, and Atari. Naturally, there is greater importance here to understanding the actions of the DRL agent, determining the salient features, and comparing them with those of humans. Previous research has focused on extending the existing explainability concepts to the RL paradigm. However, the impact of reward function has not been studied yet. In this work, we highlight the importance of incorporating the reward function in explainability models. Subsequently, we explain the relationship between reward function and agent's behavior through experiments on benchmark RL tasks – Pong, Breakout and Space Invader, respectively.

Contents

List of Figures

Chapter 1

Introduction

Deep learning (DL) is a popular class of machine learning algorithm that employs artificial neural networks to extract increasingly high-level features from input and produce optimal results. The DL algorithms are based on the brain's structure and function [17]. They have been successfully deployed in fields such as computer vision [22], speech recognition [24], natural language processing [49], bio-informatics [29], material inspection [35], and board game programs [10], with results comparable to, and in some cases exceeding the performance of human experts. Irrespective of how well they perform on such tasks, they are frequently criticized for being black boxes, implying that it is difficult to understand how the desired results are obtained. This has led to the emergence of a new field of study known as "Explainable AI (XAI)"[39].

1.1 Explainable AI (XAI)

Explainable AI is a set of tools and frameworks for understanding and interpreting machine learning predictions. Some well-known explainable AI models include Occlusion [7], Attention [47], Grad-CAM [41]. Deep image classifiers and deep sequential models have been explained in several articles, such as [41].

1.2 Reinforcement Learning (RL)

Reinforcement Learning (RL) focuses mainly in building model-free and efficient algorithms for solving sequential decision-making problems formulated in the mathematical framework of Markov Decision Processes (MDPs) [45]. An RL agent is taught to take a series of optimal actions in order to maximise a long-term goal. In contrast to traditional machine learning methods, these algorithms provide feedback on the action at each training step via a scalar reward signal, which may or may not be completely precise.

1.3 Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) combines RL with deep learning to solve the problem of state and action explosion. DRL algorithms evaluate large amounts of data (for example, every pixel on a video game screen) and determine what actions to take in order to achieve a goal (e.g., maximizing the game score) and have been successfully deployed to learn optimal behavior in robotics [19], video games [42], natural language processing [46], computer vision [25], finance [27], and healthcare [3].

1.4 XAI in DRL

The DRL agents have been proven to function effectively in challenging situations involving with sparse and noisy rewards, as well as high-dimensional inputs. However, any DRL agent is susceptible to over-fitting, i.e., when presented with a new state configuration during deployment, the agent may fail to take the optimal action (i.e., after training). To address this issue, it becomes important to understand what part of the input it is focusing on. Improved tools are essential to comprehend their decision-making process in more depth. Hence, there is a greater need for developing efficient explainability models for DRL.

1.5 Why do we care about rewards in XAI?

Previous research on the explainability of DRL such as [13, 15] centered on developing models for explaining the actions of the DRL agent and identifying the salient input features for taking the actions. However, to the best of our knowledge, none of the previous works so far have focused on understanding the impact of reward functions on explainability models. It's worth noting that, in most applications, multiple reward functions can be constructed to solve the same problem using RL. However, only a subset of these reward functions can explain the actions of the agent optimally. Therefore, it's critical to create a reward function that not only solves the desired problem but also explains it well. Practitioners will get a better insight towards building better reward functions from this.

1.6 Contributions

In this work, we investigate how the saliency of an RL agent fluctuates depending on the reward structure of the environment. We use the visual saliency's utility, in particular, to explain how agents make decisions and whether or not changing the reward affects the agent's decision-making behavior. To best of knowledge, no comprehensive investigation of this behaviour using saliency maps has been conducted. As a result, there is no compelling evidence that the current environment's reward structure is the best for the agent to make decisions.

Our key contribution in this work is to understand and analyze a series of experiments to investigate the influence of reward variation on the decision-making of DRL agents. For this purpose, we primarily select Atari games and first train an RL agent to solve the games using both Deep Q-Network and Deep Actor-Critic algorithms. We then obtain the saliency maps for these games. Next, we vary the reward function and obtain new saliency maps. Finally, we analyze and present the comparison studies. We also demonstrate the motivation for this research by conducting experiments on small dimensional systems such as a Lunar Lander, a Mountain Car, and a Chess Game.

Chapter 2

Related Work

The field of explainability has received a lot of attention in recent times, both in the general context of Deep Learning (DL) and also Deep Reinforcement Learning (DRL). We will now discuss some of the related works, first in the context of DL, followed by DRL.

In [20], a perturbation-based technique is proposed to understand the saliency of input by looking at how the output of the model changes when the information about the input is altered and gives importance to the particular portion of the input that is perturbed.
In [11], a perturbation-based technique has been proposed to create a meaningful saliency by injecting noise, removing a region, and adding a constant value (noise) to the region. In [38], the saliency of the features is computed by observing the weight vectors of the model.

Although there have been many improvements in the field of explainability of DL [1, 8, 26, 16, 2, 36], the models for explaining DRL are limited. We now discuss works that propose explainable DRL models.

Deep RL agent's explanation: In [50], tools for describing deep RL rules in visual domains have been proposed. The authors employ Atari 2600 settings as interpretable test-beds. The primary contribution here is a technique for simulating the behavior of deep RL policies using Semi-Aggregated Markov Decision Processes (SAMDPs). This is employed to learn about the policy's higher-level temporal structure.

Gradient based saliency method[21]: In this method, the feature importance is determined using a chain rule-based saliency method. The saliency is calculated by taking the Jacobian with respect to the output of interest. Unfortunately, it has been noted in the literature that the Jacobians rarely produce saliency maps that are easily understood by humans. Following this work, new studies have emerged that aim at changing gradients to achieve more meaningful saliency.

Visualizing and Understanding Atari Agents: In [13], a perturbation-based technique to generate the saliency-based visualization model has been proposed. The agents have first trained via the Asynchronous Advantage Actor-Critic (A3C) algorithm on Atari environments. Then, input features are manipulated by a masked interpolation between the original image I and a different image A, where A is chosen to introduce as little new information as possible. Then they answer the question: "How much does removing information from the region around location change the policy?". Finally, a score is assigned to the particular pixels, and saliency maps are generated.

Specific and Relevant feature attribution: The approach in [37] is similar to the previous work with a modified and improved score function. A saliency score function that identifies both specific and relevant features is proposed, which results in better explainability models.

Explanations for Attributing Deep Neural Network Predictions: In [12], they presented Meta-Predictors as Explanations, a principled framework for learning explanations for any black box algorithm, as well as Meaningful Perturbations, which is concerned with determining which features of an input (i.e., regions of an input image) are responsible for a model's output (i.e., a CNN classifier's object class prediction).

Towards better interpretability in deep q-networks: In [4], they propose an interpretable neural network architecture for Q-learning that uses key-value memories, attention, and reconstructible embeddings to provide a global explanation of the model's behaviour. Their model achieved training rewards comparable to state-of-the-art deep Q-learning models using a directed exploration strategy.

Analysing deep reinforcement learning agents trained with domain randomisation: In [9], a combination of out-of-distribution generalization tests and post-hoc interpretability method is proposed to understand the strategies of the DRL agents. This work aims to understand how the agents trained with visual domain randomization (DR) generalized from simulation-based training to the real world. Results show that DR leads to more robust, entangled representations, accompanied by greater spatial structure in convolutional filters.

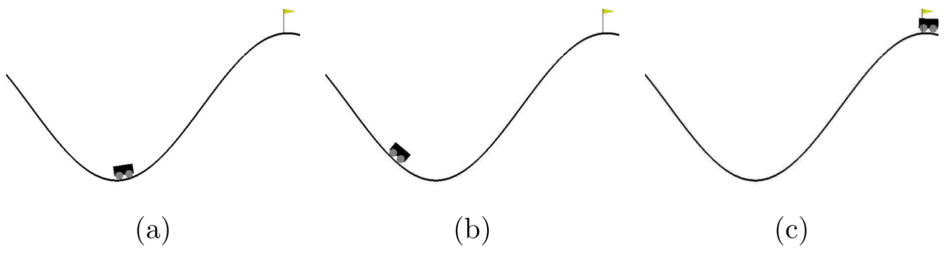


Figure 2.1: Mountain Car

Bridging the Gap: Providing Post-Hoc Symbolic Explanations for Sequential Decision Making Problems with Inscrutable Representations: In [44], approaches for giving contrastive explanations in terms of user-specified concepts in sequential decision-making scenarios have been proposed. The algorithms were tested on popular Atari games and Sokoban versions (a well-known planning benchmark).

Automatic discovery of interpretable planning strategies: In [43], the AI-Interpret – a general method for transforming idiosyncratic policies into simple and interpretable descriptions has been proposed. This algorithm combines recent advances in imitation learning and program induction with a new clustering method for identifying a large subset of demonstrations that can be accurately described by a simple, high-performing decision rule.

Widening the Pipeline in Human-Guided Reinforcement Learning with Explanation and Context-Aware Data Augmentation: In [14], the first investigation of employing human visual explanations for human-in-the-loop reinforcement learning have been carried out. This work focuses on the job of feedback learning, in which the human trainer not only offers binary evaluative "excellent" or "poor" feedback for requested state-action pairings, but also provides a visual explanation by annotating key aspects in photos. They propose "EXPAND (EXplanation Augmented Feedback)" which is a context-aware data augmentation approach that only perturbs irrelevant characteristics in human salient information to urge the model to encode task-relevant features. They chose five tasks, Pixel-Taxi and four Atari games, to assess the approach's performance.

Towards Interpretable Reinforcement Learning Using Attention Augmented Agents: In [34], a model has been proposed that bottlenecks the view of an agent by a top-down attention mechanism. This model achieves performance competitive to state-of-the-art models on Atari tasks. The model's output allows for direct observation of the information used by the agent to select its actions.

Despite these advancements, none of the DRL explainability models proposed so far have considered the effect of reward function on saliency. We aim to bridge this gap by studying the impact of reward function on DRL explainability.

Chapter 3

Problem Formulation

The Markov Decision Process (MDP) [5] is a popular framework for solving sequential decision-making problems. An MDP model is defined by a tuple (S, A, T, R, γ), where S is the state space of the environment, A is the action space, T is the probability transition matrix, $R(s, a)$ represents the reward received by the agent for taking action a in state s and $\gamma \in [0, 1)$ denotes the discount factor.

Let $\pi : S \to A$ denote the policy that specifies the action to be chosen in each state. For a MDP, when starting in state s, following a policy π, the value function of s is represented as $V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi[\sum_{t=0}^{\tau} \gamma^t R(s_t, a_t)|s_0 = s]$, where s_τ is the terminal state [45].

Similarly, the action-value function, also known as the Q-value function, can be defined as a cumulative reward obtained by an agent starting from a state s, taking action a in s, and subsequently following a policy π. Mathematically speaking, $Q_\pi(s, a) = \mathbb{E}_\pi\{\sum_{t=0}^{\tau} \gamma^t R_t|s_0 = s, a_0 = a\}$. Q-learning [48] is a popular model-free algorithm employed to compute the $Q-$value function, from which the optimal policy can be derived.

When the cardinality of state-space becomes enormous, it is challenging to update the Q-value of each state-action pair. To overcome this problem, deep RL algorithms have been proposed that combine the ideas of RL with deep neural networks [31, 32].

We now describe our problem statement. Given a system, with a current state s given as an image or a vector, the reward for all actions is defined as r, and the feature f is defined by a particular pixel of the input image or by a element in the input vector. The saliency of feature

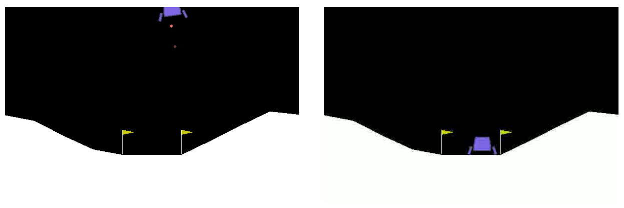

(a) At starting position (b) At ending position

Figure 3.1: Lunar Lander

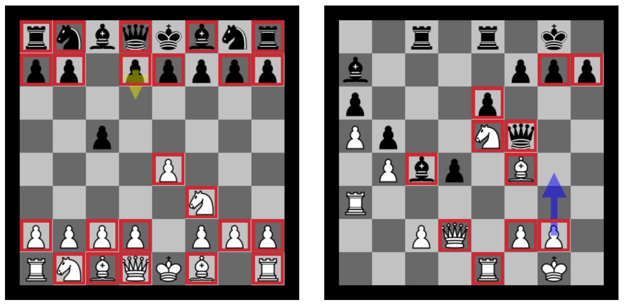

(a) (b)

Figure 3.2: Chess board with action explanation

f at a given state is defined by $S[f]$, where S is the function mapping from the feature to its saliency value. Changing the reward from r to r' leads to a new saliency weight attributed to feature f, represented as $S'[f]$, where S' is the new function mapping from feature to its saliency value. We would like to see how different $S[f]$ is from $S'[f]$. If there are no changes in saliency values of feature f ($\forall f$ in the input), that means the reward change has no effect on the existing system. The agent can thus generalize its decision to all reward structures. Otherwise, we want to find a pattern, if it exists, between reward and the agent's decision-making process.

We have used "specific and relevant feature attribution (SARFA)"[37] as the metric to find the saliency of each feature and used the Deep Q-network [31] algorithm to explain the agent's behavior. In [13], they proposed another metric that assigns a score to each feature by calculating the difference in the value functions [45] before and after perturbing that feature. We have also tested with that metric given in [13] and with the A3C [33] algorithm.

Chapter 4

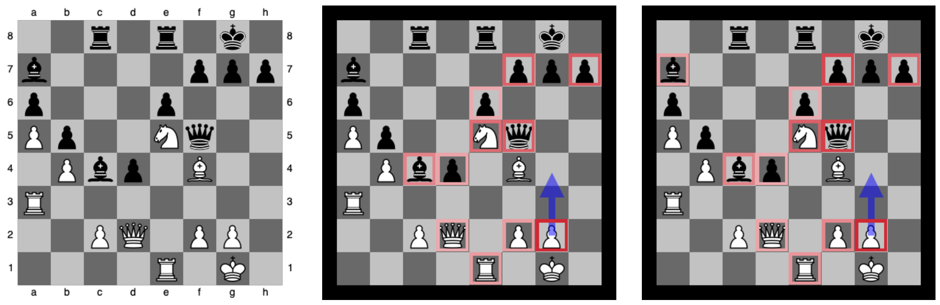
Reward Structure

In RL [45], after each action, a reward is obtained by the agent from the environment. Depending on the rewards obtained, the agent updates its policy. The agent eventually finds the optimal policy through this continuous interaction with the environment, maximizing the total expected reward. There are primarily two types of reward in the Atari environment [30]: positive rewards and negative or zero rewards. A positive reward in Atari indicates that the agent has performed a good action. In contrast, a negative or zero reward indicates that the agent's action was unsuccessful.

The goal of RL is to make the agent smarter and a better decision-maker to find the optimal policy of maximizing reward in a few steps. Finding the optimal policy is a reward-driven procedure. Therefore, the reward function plays a pivotal role when considering an agent's ability to attain the goal of maximizing reward.

The practitioners of RL intend to have a reward function that can help the agent achieve the goal early and that generalizes well (even when presented with new state configurations). One way to understand the impact of reward while explaining the agent's behavior is through changing or perturbing the reward function of the environment. We construct a reward function that helps us find the pattern between the reward and saliency maps of the agent's action. We design it in a way that the influence of reward in explainability is brought out.

In [18], the authors proposed a technique called "magnify saltatory reward (MSR)". MSR dynamically changes the reward depending upon the previous experience. It is a reward function optimization algorithm with adjustable parameters that does not interfere with the agent's interaction with the environment. However, it does change some rewards, for instance, by adding



(a) (b) (c)

Figure 4.1: chess board's saliency with different reward

some quantity before storing the experience in the experience pool. This is our motivation for adding a constant to the reward function.

Expectancy theory[28] suggests that the performance of an employee in an organization depends upon the outcomes (reward) she gets. If the employee can get higher reward, she will try to perform better to achieve it. We mimic a similar idea for our new reward function. The reward update we propose is as follows:

$$r_{new} = r_{old} * k$$

where $k \in \mathbb{R}^+$. The idea is to give more reward if the agent performs well and less if the agent shows poor performance. We varied the reward with the multiples of the actual reward given by the environment of OpenAI Gym [6]. An important point to note here is that the modified reward function still preserves the structure of the actual reward function.

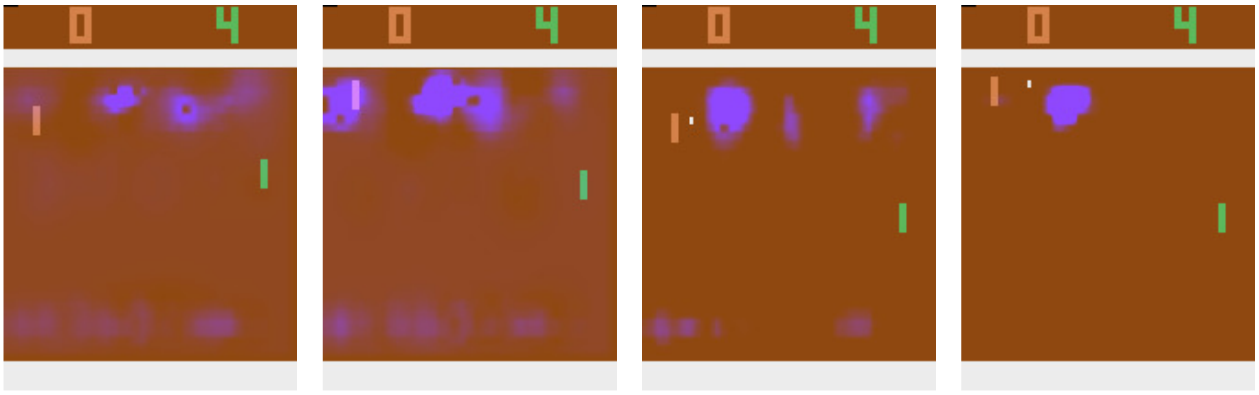
Chapter 5

Saliency Metric

Currently the state-of-the-art metric to explain an RL agent's action is SARFA [15]. It successfully produces understandable and meaningful saliency maps to explain the agent's behaviour. When using SARFA, only the impact of perturbation to the Q-value of the action to be explained is captured by *specificity* and *relevance* lessen the importance of features that change the expected rewards of actions other than the one being explained. By combining these two, we generated saliency maps to explain the Atari games. The metric SARFA is given by

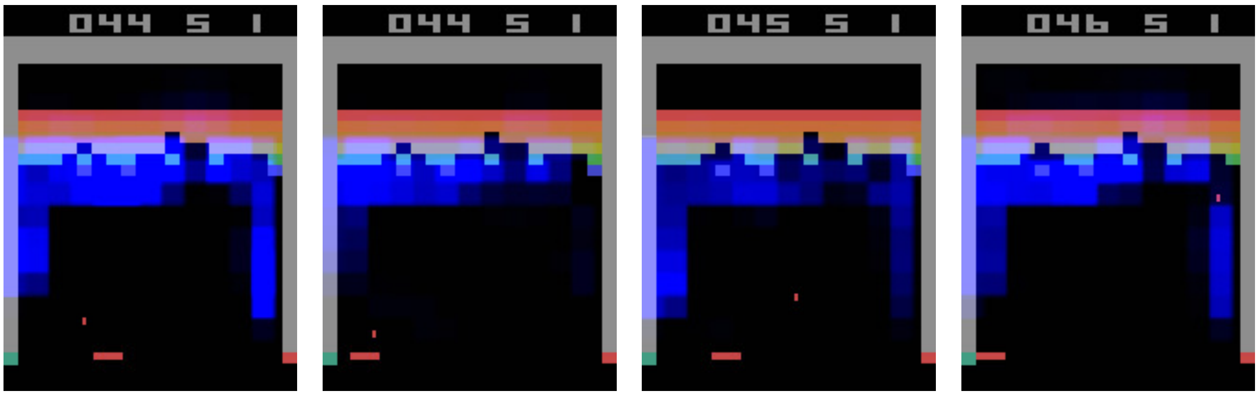
$$\begin{aligned}
S[f] &= \frac{2K\Delta P}{K + \Delta P} \\
K &= \frac{1}{1 + D_{KL}} \\
D_{KL} &= P_{rem}(s', a) || P_{rem}(s, a) \\
P_{rem}(s, a) &= \frac{exp(Q(s, a))}{\sum_{a \neq \tilde{a}} Q(s, \tilde{a})} \\
\Delta P &= P(s, \tilde{a}) - P(s', \tilde{a}) \\
P(s, \tilde{a}) &= \frac{exp(Q(s, \tilde{a}))}{\sum_{a} exp(Q(s, a))}
\end{aligned}$$

Here $\tilde{a}$ is the action that is being explained in state s. s' is the perturbed state after perturbation of a feature f. $S[f]$ captures the intended saliency value of a given feature f. If D_{KL} is high, it suggests that the influence of the feature f is spread across all actions and hence is not relevant for action $\tilde{a}$. Moreover, a higher value of ΔP indicates that the feature f is more specific in explaining action $\tilde{a}$. Therefore, the saliency metric $S[f]$ is derived in [37] by taking the harmonic mean between K and ΔP.



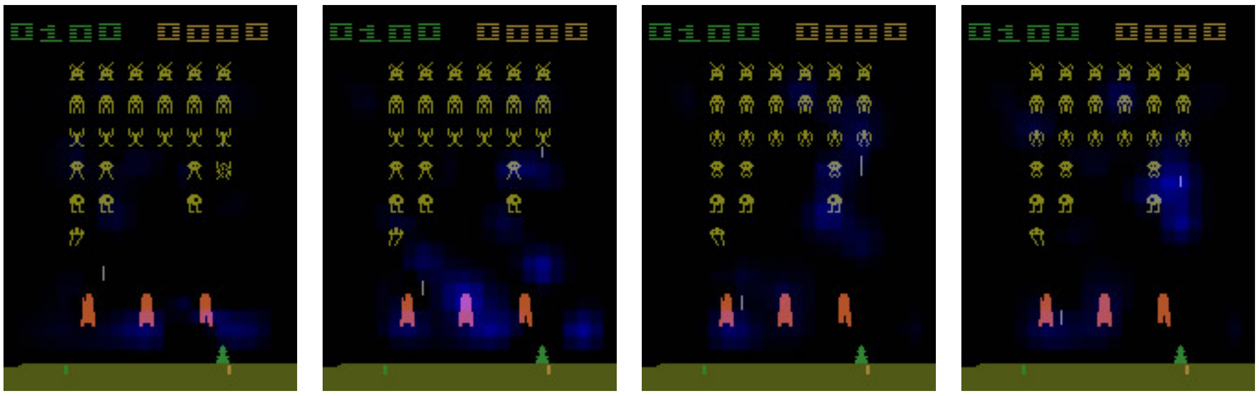
(a) (b) (c) (d)

Figure 5.1: Pong trained by DQN Learning: What features are important for $k = 1$



(a) (b) (c) (d)

Figure 5.2: Breakout trained by DQN Learning: What features are important for $k = 1$



(a) (b) (c) (d)

Figure 5.3: SpaceInvaders trained by DQN Learning: What features are important for $k = 1$

Chapter 6

Numerical Experiments

This section goes over the environments and algorithms that were used in this study. We get a quick overview of the system that will serve as the study's environment in Section 4.1. Section 4.2, on the other hand, includes a brief description of the algorithm as well as the model's input and output.

6.1 Setup and Implementation Details

We consider various experimental settings involving Lunar Lander, Mountain Car, Chess, as well as Atari games.

6.1.1 Lunar Lander and Mountain Car Settings:

We start our experiments with small dimensional input for the settings of Lunar Lander and Mountain Car because they are easy to comprehend. Lunar Lander and Mountain Car agents are trained using the DQN algorithm [31]. The system has been given a one-dimensional vector as input, which predicts the desired action at that state. The input for the Lunar Lander is a 1×8 vector, while the input for the Mountain Car is a vector of 1×2 dimension, respectively.

The following are the eight components of the Lunar Lander's state: x is the lander's horizontal coordinate, y is the lander's vertical coordinate, v_x is the horizontal velocity, and v_y is the vertical velocity, θ is the spatial orientation, the angular velocity is represented by v_θ, right leg touching the ground (Boolean) and left leg touching the ground (Boolean), respectively. Do nothing (0), fire left engine (1), fire right engine (2), and fire main engine (3) are the four discrete actions available to the agent.

For the mountain car, the first component represents the position of the car and the second component represents the velocity of the car. At any given time, the car can only take one of three actions: accelerate forward (2), accelerate backward (0), or do nothing (1).

The agents are trained here using the OpenAI Gym API [6]. In order to perturb the state, a small noise with 0 mean and variance of $1e-5$ has been added to each element of the state vector while explaining the agents. SARFA [37] has been used as a metric to find the importance of each feature.

6.1.2 The Game of Chess:

We next use chess as the setting for our experiments. The stockfish engine's reward structure is used, and the engine, trained using RL, also decides on the actions. For this engine, we want explain it as well as find any dependencies with a reward. Reward for the usual moves are given by chess.uci.score class and the mate point is 40 for white and -40 for black in the uci.score class.

Each state of the state board is represent by a 8×8 vector. Each element of the vector represents the position of a piece of chess. Location of each piece can be represented by a corresponding row and column number of the board. In the board given in figure 4.1(a), the location of kings are $f5, d2$ for white and black king respectively.

Perturbation is carried out by removing each piece one by one out of all of the 64 positions of the chess board for each action. If a position is empty, then it is skipped. As previously, explanation of the model is done by creating saliency map for each action using the SARFA metric.

6.1.3 Atari Games Settings – Pong, Breakout and SpaceInvaders:

All of our Atari agents are trained using both Deep-Q-network(DQN) [31] and Asynchronous actor-critic (A3C) [33] algorithms. They are well-known for their ease of use and high performance in Atari environments. Each Atari agent interacts with the environment through a sequence of observations, rewards, and actions.

The input to the DQN and A3C are the preprocessed Atari frames described in [31] and [13], respectively. DQN tries to predict the $Q-$value of each action at a given state as input. For the A3C algorithm, the agent needs to learn a policy distribution (actor) and a value estimate (critic).

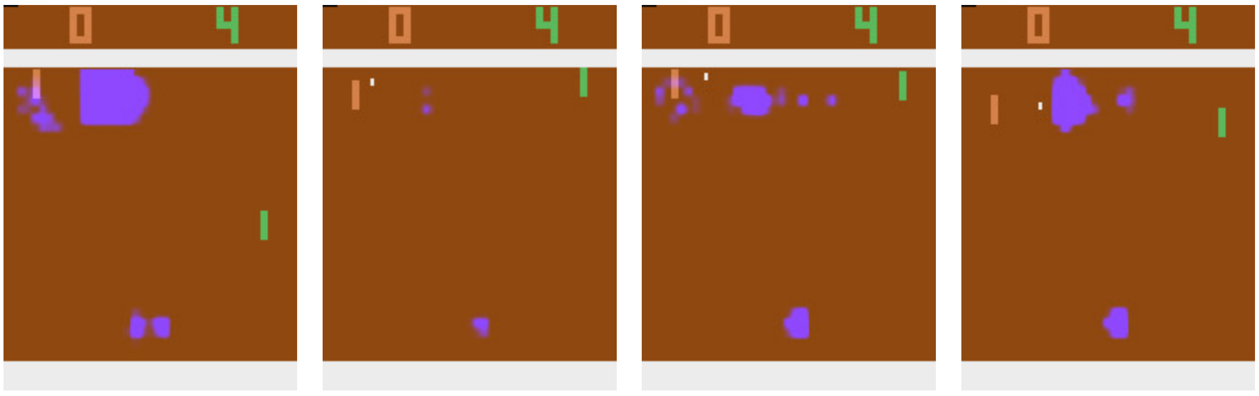

(a) (b) (c) (d)

Figure 6.1: Pong trained by DQN Learning: What features are important for $k = 2$

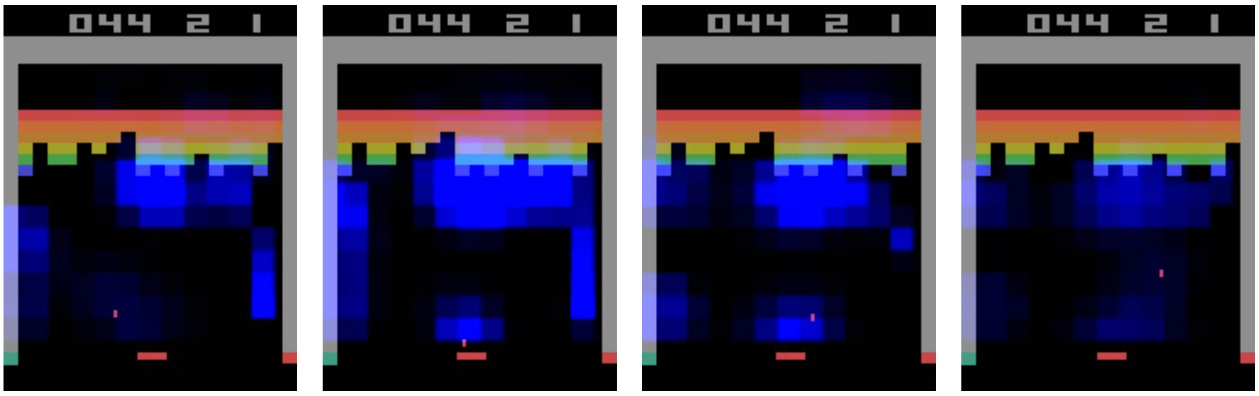

(a) (b) (c) (d)

Figure 6.2: Breakout trained by DQN Learning: What features are important for $k = 2$

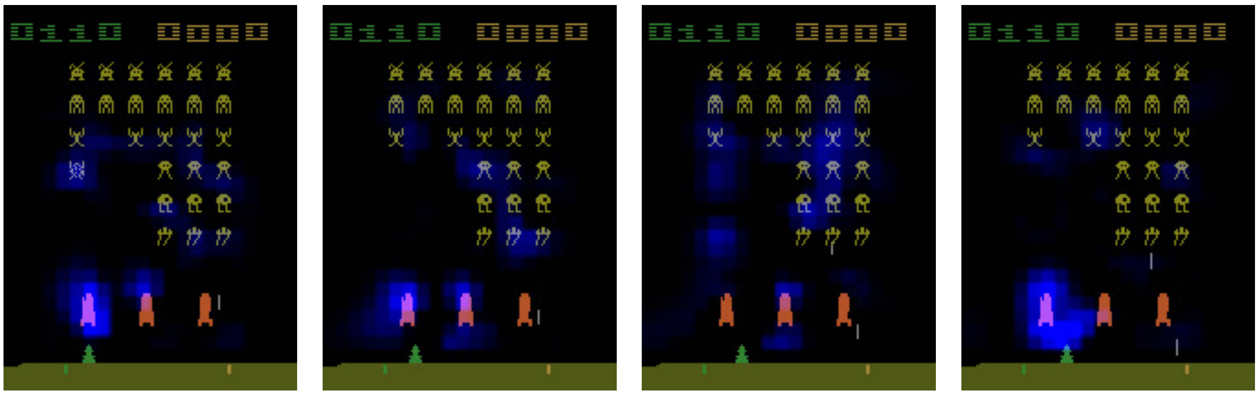

(a) (b) (c) (d)

Figure 6.3: SpaceInvader trained by DQN Learning: What features are important for $k = 2$

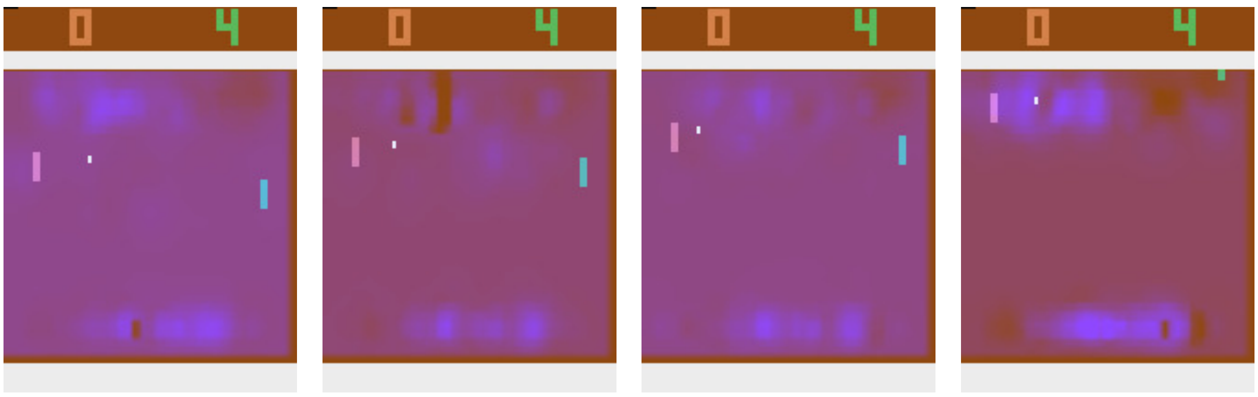


Figure 6.4: Pong trained by DQN Learning: What features are important for $k = 0.5$

To create the saliency map, we have used a perturbation-based technique similar to [13]. The perturbation has been created by finding interpolation between an actual preprocessed input image and the Gaussian blur of that image. The interpolation coefficient is given by a 2D-Gaussian centered around a pixel of that image. The saliency metric of the pixel is calculated using the SARFA formulation discussed in Section 5.

Using the OpenAI Gym API [6], we trained agents on Pong, Breakout, and SpaceInvaders for our series of experiments. We chose these environments because each of them presents unique challenges, and deep RL algorithms have previously outperformed humans on these tasks [31].

6.2 Details of Experiments

We first experiment with the DQN [31] agent for all the experiments and explain it using the SARFA [37] metric. DQN is an end-to-end training procedure and can utilize the feature-extracting capabilities of DL and the flexibility of RL. It uses a deep convolutional neural network architecture to approximate the optimal Q value function [31] defined below.

$$
\begin{aligned}
Q^*(s,a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\
\mid s_t = s, a_t = a, \pi],
\end{aligned}
$$

where γ is the discount factor and r_τ is the reward obtained by the agent at time step τ. After making an observation s and taking an action a following a behavior policy $\pi = P(a|s)$, the maximum sum of rewards r_t, discounted by γ at each time step t is defined as $Q^*(s,a)$.

It offers strong versatility because of its usage in many different applications of RL. It learns the optimal Q-value function by sampling the historical data. It uses two networks, online and target network. These both networks try to approximate $Q(s, a, \theta^+)$ and $Q(s, a, \theta^-)$ respectively. With each training sample, the parameter θ^+ of the online network gets updated. After every N iterations, θ^+ is copied to the target network and update θ^-. The online network is used to predict the action during inference time. The loss function of the DQN at each iteration is given by

$$L_i(\theta_i^+) = \mathop{\mathbb{E}}_{(s,a,r,s')}[(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i^+))^2].$$

In the actor-critic algorithm [23], the agent has a policy distribution and a value function estimate as an actor and a critic, respectively. After each action drawn from the policy distribution, the critic evaluates the next state to determine the effectiveness of the learned policy. A deep neural network can be used to approximate the parameters of an actor and a critic. Update of the parameters happens after every t number of actions or until a terminal state is reached [32].

The main difference between A3C and DQN is that in A3C, only one neural network interacts with a single environment. There is a global network with various agents, each with their own set of parameters. It creates each agent's situation by allowing them to interact with their surroundings and collecting a variety of unique learning experiences for overall training. This also addresses RL sample correlation, which is a major problem for neural networks, which are built on the assumption that input samples are unrelated. Actor-Critic is made up of two neural networks: Actor and Critic. As a result, it provides an ideal setting for an agent to reap the greatest benefits from rapid learning.

In the case of A3C, all Atari agents shared a recurrent structure that is the same as the architecture given in [13]. Similar to [13], we also calculated the policy's loss using Generalized Advantage Estimation [40] with a λ value of 1.0. The timescale of TD updates is controlled by λ.

For creating saliency values for all features for each action of a learned policy, the Q-values for each state and action pair are required. For some systems, such as Atari and Chess, we can create a saliency map to visualise the important features. A system like a mountain car must rely on the numerical saliency values of each feature.

The saliency value of each element of the feature has been calculated for Lunar Lander and

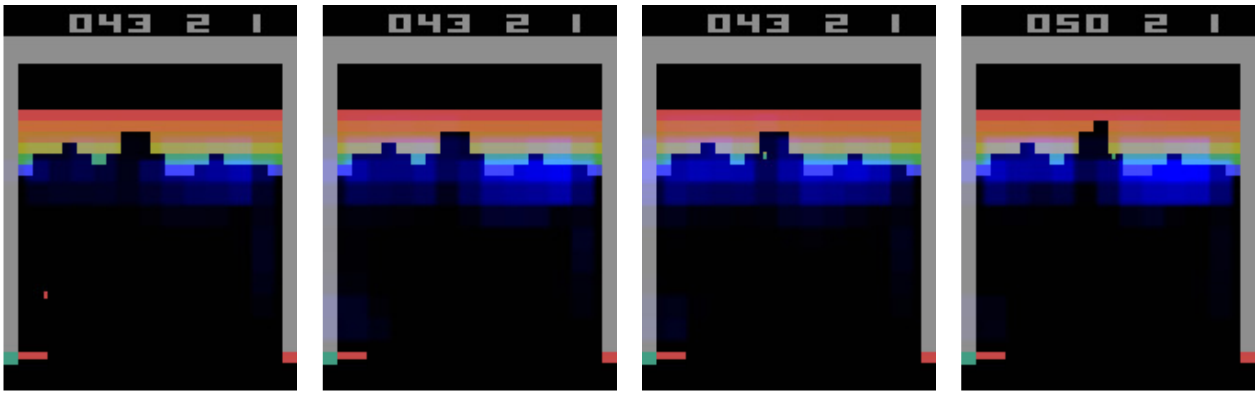

(a) (b) (c) (d)

Figure 6.5: Breakout trained by DQN Learning: What features are important for $k = 0.5$

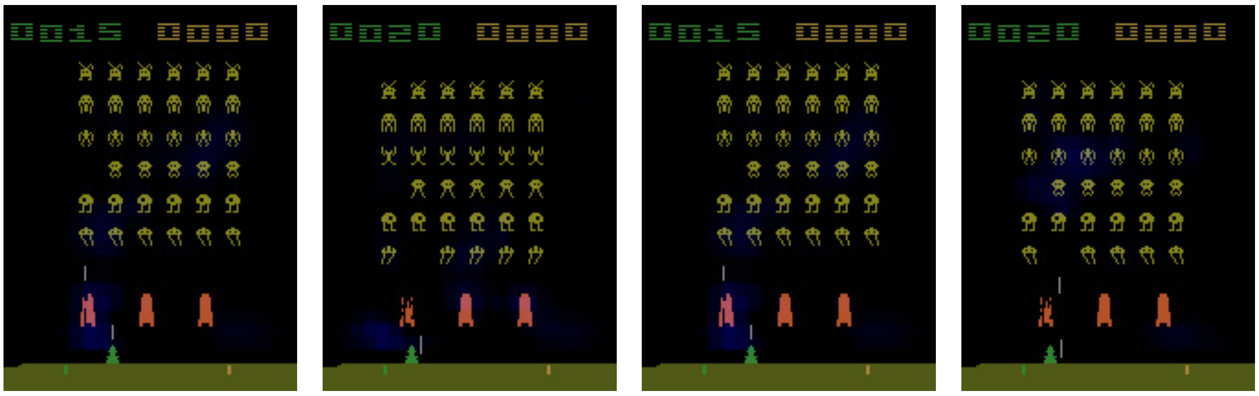

(a) (b) (c) (d)

Figure 6.6: SpaceInvaders trained by DQN Learning: What features are important for $k = 0.5$

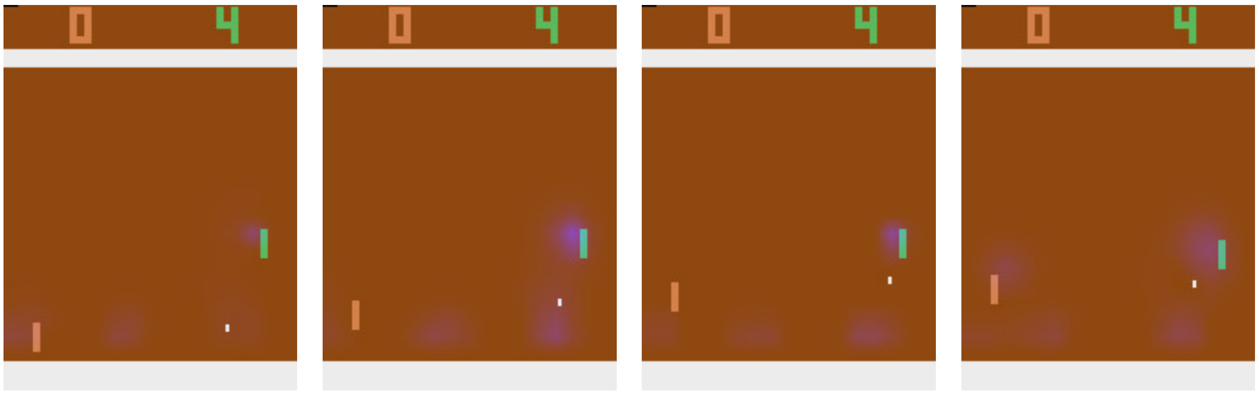

(a) (b) (c) (d)

Figure 6.7: Pong trained by A3C Learning: What features are important for $k = 0.5$

Mountain Car environments, where the input is a state vector rather than an image. The state of the Mountain Car is $[-0.47209626, 0]$ at one point in the episode, and the action is 0, which is to push left. The first and second elements have saliency values of 0.41 and 0.39, respectively. At a given state of Lunar Lander, say at $[0.00309277, 1.4079185, 0.31324595, -0.13340081$ $, -0.00357693, -0.07095474, 0, 0]$ the action 1, which means firing the left engine, the highest saliency value corresponds to the 5th feature, i.e., angular velocity of the lander.

For Atari and chess agents, We impose the saliency maps upon the input images to highlight the essential pixels because of which the agent decides some action. For example, Figure 5.1 shows the saliency maps of Pong. In Figure 5.1(c), when the ball moves upward, then the green paddle, which is the RL agent, adjusts itself by focusing on the highlighted blue part at the top and bottom of the frame. Similarly, Figures 5.2 and 5.3 contain a snapshot of the Breakout and SpaceInvaders games with the highlighted features.

We experiment with different values of k for the reward function defined above. Because of the space constraints, we only provide the results here for $k = 0.5, 1$, and 2, respectively.

Chapter 7

Results and Observation

7.1 Mountain Car Environment

At the start of the game, the car is in the middle position (Figure 2.1(a)) and the agent pushes the car towards right so that it can achieve a momentum to cross the flag (Figure 2.1(c)). As mentioned previously, at the start of the game, 2.1(a), the state is $[-0.47209626, 0]$ with saliency values 0.41 and 0.39 respectively and the action is 0, which is push left. One can see, in the beginning, the position of the car is more important than the velocity for action 0. As a result, the velocity of the car changes and the state of the car becomes $[-0.473481, -0.00138473]$. Here the velocity becomes negative which means car is going upward to the left.

However, this perturbation method has not worked perfectly for this system. When the system's state is $[-0.6467985, -0.01597446]$, which is nearer to the left of the car in Figure 2.1(b) and the agent is pushing it to the left (action is 0) then ideally it should focus on either one of the features more. But in this case, the agent focuses on none of the features, which means the saliency value is 0 for both of the features.

The above implies that when the state is small-dimensional like mountain car, the perturbation technique does not provide a fruitful explanation. As a result, it is not a great system for observing the phenomenon of reward change.

7.2 The Lunar Lander Environment

There is a rocket in this environment (Figure 3.1(a)), and the agent must park the rocket between the flags on the ground (Figure 3.1(b)). We used a few techniques to explain this system. The first is similar to the mountain car, in which we add noise to each feature and calculate the saliency value of those features to determine which is the most important. This system has also been used to investigate the phenomenon of reward change. Following that, we expanded it by combining multiple features at a time and repeating the process. In this case, more than one feature will be important for each agent's action. Finally, in order to gain a better understanding, we grouped similar states together and looked for important features in each cluster. The most important feature in each cluster for the same action should be the same. We will primarily consider the agent's starting and ending states when comparing different states and saliencies.

One important feature: As previously stated, the rocket's initial state (figure 3.1(a)) is $[0.00309277, 1.4079185, 0.31324595, -0.13340081, -0.00357693, -0.07095474, 0, 0]$, as well as the action 1, which entails firing the left engine. The most important feature with the highest saliency value in this state is the 5th feature, i.e., the lander's angular velocity. Its state changes to a new state that is closer to the previous state's value. The action is also 1 in the new state. However, the most important feature in the new version is 1, which is the lander's y co-ordinate.

When the state of the environment is $[-1.29458711e-01, -1.64168118e-03, 4.87604659e-08, 1.95178051e-09, 1.23818675e-02, -1.03578472e-07, 1.00000000e+00, 1.00000000e+00]$ as the lander approaches the landing area (Figure 3.1(b)), and the agent isn't doing anything, so the action is 0. In that state all the feature's saliency values are zero. This implies that the agent is focusing on no feature which made the agent to perform the action 0. This is an undesirable behaviour. To make some decision agent must focus on some features. Similarly like mountain car, perturbation based techniques perform poorly in the lunar lander environment.

We replicate the same experiments after changing the lander's reward. Here reward change means we increase all the rewards by some quantity for example 2, 5, 10 etc. But we have not discovered any pattern in the saliency values of features in this environment.

More than one important features: It's possible that the agent is concentrating on more than one feature of the state while taking some action. In that case, we add noise to each possible combination of states and compute the saliency metric for that combination by finding the difference between Q-values. When only two features are considered at a time, at the start of the episodes, the most important features are 2 and 7, which are angular velocity and right leg touching the ground. However, because the lander is in the starting position, which is high above the ground (Figure 3.1(a)), the right leg of the lander cannot touch the ground. In that case, we can conclude that the explainability model isn't functioning properly in the current environment.

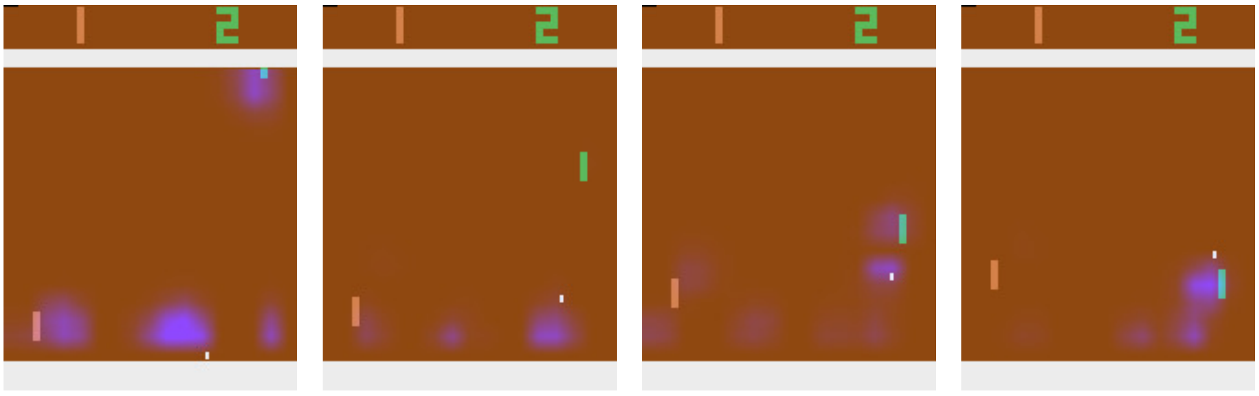
Similarly, we ran our tests with 3, 4, and 5 different feature combinations. With 3 and 5 feature combinations, the important features at the start are $[4, 5, 7]$ and $[1, 2, 3, 5, 7]$. For the same reason as before, the 7th feature cannot be crucial to the episode's beginning state. The 7th feature is not important for the starting position in the 4 combination, but there is no particular pattern in the saliency values that we can find.

Clustering states: As the state of a lunar lander can be represented by a vector we cannot visualise it using saliency maps. To gain better understanding of the explainability model, we introduce a cluster analysis technique. The agent must perform some action at each position of the lander. Taking some action modifies the state slightly but not dramatically. The agent should take the same actions when the state values are close together. As a result, we used k-means clustering to cluster the states. We identified the key characteristics in each cluster. For example, the state at the start of the episode is cluster 1, and the action in that state is fire left engine. The most important feature in that state is angular velocity. This action sends the lander to a different state, which is also in cluster 1, and it also fires the left engine. The y coordinate is the most important feature in this state. In the same cluster, for the same action, the salient features are different for different states. This is an undesirable observation. As a result, we have observed that there is no discernible pattern for saliency values across clusters and actions.

As can be seen from the above, perturbation-based explainability models are not well suited to small-dimensional systems. One of the possible causes is that adding some of the states causes it to overshoot to another valid state. As a result, we switched to a chess engine to see if there was any correlation between saliency and reward.

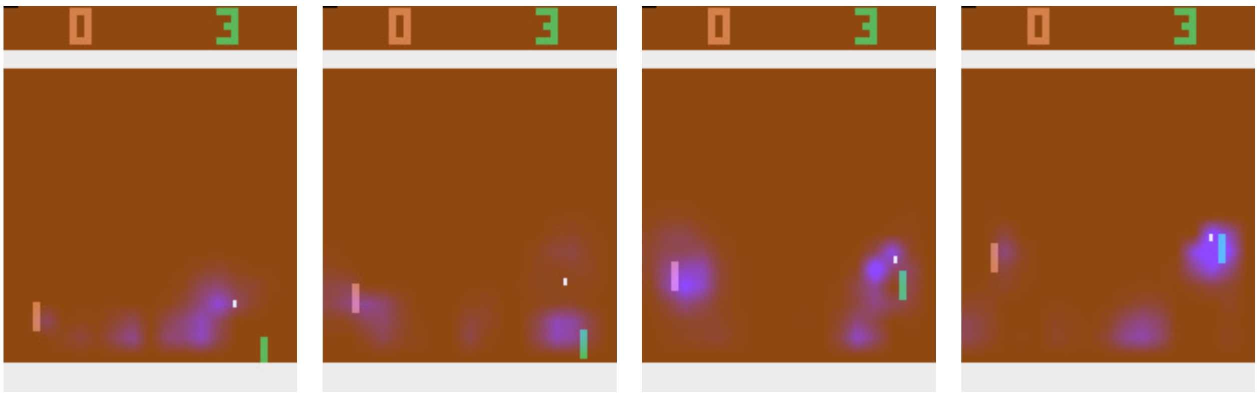
7.3 The Chess Environment

Explaining the chess engine's action will highlight the important block required for that particular action. For example in Figure 3.2, the action is indicated by a yellow and blue arrow. The action's corresponding important blocks are bordered by a red colour. The intensity of the red colour increases as the importance of a feature increases. That is, if one square's boundary has a denser red colour than the others, then this block is more important in the decision-making process for that action.



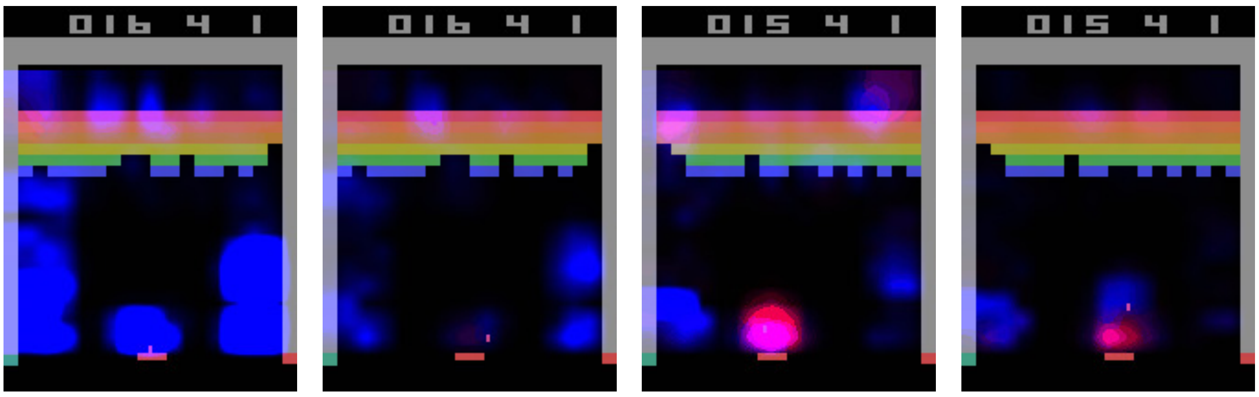
(a) (b) (c) (d)

Figure 7.1: Pong trained by A3C Learning: What features are important for $k = 1$



(a) (b) (c) (d)

Figure 7.2: Pong trained by A3C Learning: What features are important for $k = 2$



(a) (b) (c) (d)

Figure 7.3: (a), (b) Breakout for $k = 1$; (c), (d) for $k = 2$ explained by Greydanus et. al. [13]

Now changing the reward, has given us the result shown in the Figure 4.1, where Figure 4.1(b) is with normal reward structure of chess engine and Figure 4.1(c) is with rewards incremented by 0.5. One can see from these figures that there is a difference between Figures 4.1(b) and (c). In 4.1(b), less number of blocks are being focused, but intensity of some of the blocks ($e5, f5, f7$) become higher in 4.1(c). These actions are taken by the chess engines, which is being trained by reinforcement learning algorithm. Using SARFA [37] we explain the engines action and also changes the reward of the engine.

The state of a chess board is defined by an 8×8 vector, which a small dimensional vector compare to Atari. We observe that the reward change affects the saliency maps of the chess engine. Due to this change in saliency value, it motivates us to look forward with the high dimensional input system such as Atari, where the agent is trained by a deep reinforcement learning algorithm.

7.4 The Atari Games Environments

We first experiment with the DQN [31] agent on the Atari games and explain it using the SARFA [37] metric. This requires creating saliency maps for each action of a learned policy. We impose the saliency maps upon the input images to highlight the essential pixels because of which the agent decides some action. For example, Figure 5.1 shows the saliency maps of Pong. In Figure 5.1(c), when the ball moves upward, then the green paddle, which is the RL agent, adjusts itself by focusing on the highlighted blue part at the top and bottom of the frame. Similarly, Figures 5.2 and 5.3 contain a snapshot of the Breakout and SpaceInvaders games with the highlighted features. We experiment with different values of k for the reward function defined above in section 4.

To compare the results obtained from different values of k, we mainly look into two aspects of the generated saliency maps. These two aspects of saliency are as follows:

Confidence: This attribute of saliency maps tells how confident the agent is when taking certain actions. The density of the blue part around a pixel in the input image is used to deduce confidence. If a pixel is highlighted more, it means the agent is focusing more on that pixel, which in turn means that the agent is more confident about that pixel.

Sparsity: This indicates how sparse the highlighted features are. This means that if the agent is focusing on a limited yet important portion of the input image while taking some action, then sparsity will be high. But if a large and unimportant portion of the input is also highlighted, then the sparsity will be low.

In an ideal scenario, the confidence and sparsity should be high for an agent. This means that important features should have high saliency value and less important features should have low saliency value. Confidence captures the weight of the important feature, and Sparsity down-weighs the irrelevant features. A well-trained agent should focus on less features with high confidence. We will compare different saliency maps with these two metrics.

Figures 6.1, 6.2, and 6.3 capture the saliency maps generated by SARFA when $k = 2$. In Figure 6.1, the highlighted portion of the input image is reduced. The lower part of the frames in Figure 5.1 have an obscure blue color, but in the case of Figure 6.1, the blue color becomes more prominent. There is also no unfocused blue color in Figure 6.1. This indicates that confidence and sparsity become higher for Pong agents when $k = 2$. Similarly, in Figure 6.2, the sparsity increases, and the breakout agent's focus reduces to a subset of the features highlighted in Figure 5.2. As the focus becomes concentrated on fewer features, the confidence becomes high for the breakout agent. Again, in Figure 6.3, the blue part in the saliency maps becomes more prominent. Therefore, confidence becomes higher. But in the case of sparsity, for both $k = 1$ and $k = 2$, saliency maps are sparse for SpaceInvaders.

When $k = 0.5$, the saliency maps generated by the explainability models for Pong are captured in Figure 6.4. In that figure, all of the input frames have been covered with blue. That means the XAI models give more weight to irrelevant features, which reduces sparsity and lowers the agent's confidence. Similarly, for Breakout, the highlighted part becomes obscure and spreads out to most of the lower bricks of the input image in Figure 6.5. Therefore, reducing the reward to half makes the Breakout agent less confident and less sparse. For the SpaceInvader game in Figure 6.6, the explainability model has given a low value to all the features of the input irrespective of their importance. Because of this, the features do not even appear as highlighted by the same configuration as others in the figure. This means that important features also have a very low score, i.e., the confidence of the agent is very low. Moreover, it gives small weight to every feature and not just unimportant features, as sparsity down-weighs the irrelevant features only. This suggests that the sparsity for this agent is low as well.

From the above results, we can conclude that boosting the reward makes the saliency of an agent more confident and sparse. And reducing the reward lowers the confidence of the agent. With regards to the configuration given in the paper [13], it became difficult to differentiate between the saliency maps generated for the trained A3C agents. However, looking carefully at the agent's behavior, one can observe that for similar actions, the reward change influences the highlighted feature. For example, when the ball is moving towards the agent (green paddle) in Figures 6.7, 7.1 and 7.2, the confidence level of the Pong agent increases. This explanation comes from the use of the metric SARFA.

With the metric given in [13], a lot of redundant features of the input have been focused on by the explainability model [37]. Because of that, it is very difficult to spot the difference between saliency maps for different values of k. Again comparing similar actions, one can observe the change in the saliency values. For instance, Figure 7.3(c)-7.3(d)'s agent appears smarter than Figure 7.3(a)-7.3(b)'s agent.

Chapter 8

Conclusion and Future Work

To the best of our knowledge, ours is the first attempt at studying the effect of the reward function on the saliency maps generated by the DRL agents. Through extensive experimental evaluation in Atari environments, we have shown that agents are more confident of their actions when the reward function is amplified by a factor of more than 1. On the other hand, when the reward function is multiplied by a factor less than 1, the atari agents focus on both relevant and irrelevant features with less confidence. Similarly, we also have shown this perturbation based explanation technique works poorly for low-dimensional systems. In the future, we would like to develop novel techniques to systematically perturb the reward function to gain an even better understanding on the effect of the reward function on saliency. Apart from that, we would also like to find the optimal reward structure for Atari and verify it visually.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018. 4

[2] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems*, 34, 2021. 4

[3] Andrés Anaya-Isaza, Leonel Mera-Jiménez, and Martha Zequera-Diaz. An overview of deep learning in medical imaging. *Informatics in Medicine Unlocked*, 26:100723, 2021. ISSN 2352-9148. doi: https://doi.org/10.1016/j.imu.2021.100723. URL `https://www.sciencedirect.com/science/article/pii/S2352914821002033`. 2

[4] Raghuram Mandyam Annasamy and Katia Sycara. Towards better interpretability in deep q-networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4561–4569, 2019. 5

[5] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957. 8

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`. 11, 15, 17

[7] Himanshu Chandel and Sonia Vatta. Occlusion detection and handling: a review. *International Journal of Computer Applications*, 120(10), 2015. 1

[8] Thomas Y. Chen. Interpretability in convolutional neural networks for building damage classification in satellite imagery. *CoRR*, abs/2201.10523, 2022. URL `https://arxiv.org/abs/2201.10523`. 4

[9] Tianhong Dai, Kai Arulkumaran, Samyakh Tukra, Feryal Behbahani, and Anil Anthony Bharath. Analysing deep reinforcement learning agents trained with domain randomisation. *CoRR*, abs/1912.08324, 2019. URL http://arxiv.org/abs/1912.08324. 5

[10] Francisco Elizalde, Luis Sucar, Manuel Luque, Francisco Díez, and Alberto Reyes Ballesteros. Policy explanation in factored markov decision processes. pages 97–104, 01 2008. 1

[11] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *CoRR*, abs/1704.03296, 2017. URL http://arxiv.org/abs/1704.03296. 4

[12] Ruth Fong and Andrea Vedaldi. *Explanations for Attributing Deep Neural Network Predictions*, pages 149–167. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28954-6. doi: 10.1007/978-3-030-28954-6_8. URL https://doi.org/10.1007/978-3-030-28954-6_8. 5

[13] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. In *International Conference on Machine Learning*, pages 1792–1801. PMLR, 2018. v, 2, 5, 9, 15, 17, 18, 24, 27

[14] Lin Guan, Mudit Verma, and Subbarao Kambhampati. Explanation augmented feedback in human-in-the-loop reinforcement learning. *CoRR*, abs/2006.14804, 2020. URL https://arxiv.org/abs/2006.14804. 6

[15] Piyush Gupta, Nikaash Puri, Sukriti Verma, Dhruv Kayastha, Shripad Deshmukh, Balaji Krishnamurthy, and Sameer Singh. Explain your move: Understanding agent actions using focused feature saliency. *CoRR*, abs/1912.12191, 2019. URL http://arxiv.org/abs/1912.12191. 2, 12

[16] Peter Hase, Harry Xie, and Mohit Bansal. Search methods for sufficient, socially-aligned feature importance explanations with in-distribution counterfactuals. *CoRR*, abs/2106.00786, 2021. URL https://arxiv.org/abs/2106.00786. 4

[17] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2009. 1

[18] Zijian Hu, Kaifang Wan, Xiaoguang Gao, and Yiwei Zhai. A dynamic adjusting reward function method for deep reinforcement learning with adjustable parameters. *Mathematical Problems in Engineering*, 2019, 2019. 10

[19] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning; lessons we've learned. *CoRR*, abs/2102.02915, 2021. URL https://arxiv.org/abs/2102.02915. 2

[20] Maksims Ivanovs, Roberts Kadikis, and Kaspars Ozols. Perturbation-based methods for explaining deep neural networks: A survey. *Pattern Recognition Letters*, 150:228–234, 2021. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2021.06.030. URL https://www.sciencedirect.com/science/article/pii/S0167865521002440. 4

[21] Ashkan Khakzar, Soroosh Baselizadeh, and Nassir Navab. Rethinking positive aggregation and propagation of gradients in gradient-based saliency methods. *CoRR*, abs/2012.00362, 2020. URL https://arxiv.org/abs/2012.00362. 4

[22] Asharul Islam Khan and Salim Al-Habsi. Machine learning in computer vision. *Procedia Computer Science*, 167:1444–1451, 2020. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.03.355. URL https://www.sciencedirect.com/science/article/pii/S1877050920308218. International Conference on Computational Intelligence and Data Science. 1

[23] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999. 18

[24] L Ashok Kumar, D Karthika Renuka, S Lovelyn Rose, M C Shunmuga priya, and I Made Wartana. Deep learning based assistive technology on audio visual speech recognition for hearing impaired. *International Journal of Cognitive Computing in Engineering*, 3: 24–30, 2022. ISSN 2666-3074. doi: https://doi.org/10.1016/j.ijcce.2022.01.003. URL https://www.sciencedirect.com/science/article/pii/S2666307422000031. 1

[25] Ngan Le, Vidhiwar Singh Rathour, Kashu Yamazaki, Khoa Luu, and Marios Savvides. Deep reinforcement learning in computer vision: A comprehensive survey. *CoRR*, abs/2108.11510, 2021. URL https://arxiv.org/abs/2108.11510. 2

[26] Haoyu Liang, Zhihao Ouyang, Yuyuan Zeng, Hang Su, Zihao He, Shu-Tao Xia, Jun Zhu, and Bo Zhang. Training interpretable convolutional neural networks by differentiating class-specific filters. *CoRR*, abs/2007.08194, 2020. URL https://arxiv.org/abs/2007.08194. 4

[27] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. Finrl: A deep reinforcement learning library for automated

stock trading in quantitative finance, 2020. URL https://arxiv.org/abs/2011.09607. 2

[28] Isaac Mathibe. Expectancy theory and its implications for employee motivation. *Academic Leadership: the Online Journal*, 6(3):8, 2008. 11

[29] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in Bioinformatics*, 18(5):851–869, 07 2016. ISSN 1467-5463. doi: 10.1093/bib/bbw068. URL https://doi.org/10.1093/bib/bbw068. 1

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602. 10

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 8, 9, 14, 15, 17, 25

[32] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016. 8, 18

[33] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783. 9, 15

[34] Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J. Rezende. Towards interpretable reinforcement learning using attention augmented agents. *CoRR*, abs/1906.02500, 2019. URL http://arxiv.org/abs/1906.02500. 6

[35] Will Nash, Tom Drummond, and Nick Birbilis. A review of deep learning in the study of materials degradation. *npj Materials Degradation*, 2(1):1–12, 2018. 1

[36] Jayneel Parekh, Pavlo Mozharovskyi, and Florence d'Alché Buc. A framework to learn with interpretation. *Advances in Neural Information Processing Systems*, 34, 2021. 4

[37] Nikaash Puri, Sukriti Verma, Piyush Gupta, Dhruv Kayastha, Shripad Deshmukh, Balaji Krishnamurthy, and Sameer Singh. Explain your move: Understanding agent actions using specific and relevant feature attribution. *arXiv preprint arXiv:1912.12191*, 2019. 5, 9, 12, 15, 17, 25, 27

[38] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL `http://arxiv.org/abs/1602.04938`. 4

[39] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *CoRR*, abs/1708.08296, 2017. URL `http://arxiv.org/abs/1708.08296`. 1

[40] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 18

[41] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1

[42] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *CoRR*, abs/1912.10944, 2019. URL `http://arxiv.org/abs/1912.10944`. 2

[43] Julian Skirzyński, Frederic Becker, and Falk Lieder. Automatic discovery of interpretable planning strategies. *Machine Learning*, 110(9):2641–2683, 2021. 6

[44] Sarath Sreedharan, Utkarsh Soni, Mudit Verma, Siddharth Srivastava, and Subbarao Kambhampati. Bridging the gap: Providing post-hoc symbolic explanations for sequential decision-making problems with black box simulators. *CoRR*, abs/2002.01080, 2020. URL `https://arxiv.org/abs/2002.01080`. 6

[45] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`. 1, 8, 9, 10

[46] Víctor Uc-Cetina, Nicolás Navarro-Guerrero, Anabel Martín-González, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *CoRR*, abs/2104.05565, 2021. URL https://arxiv.org/abs/2104.05565. 2

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762. 1

[48] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992. 8

[49] Stephen Wu, Kirk Roberts, Surabhi Datta, Jingcheng Du, Zongcheng Ji, Yuqi Si, Sarvesh Soni, Qiong Wang, Qiang Wei, Yang Xiang, et al. Deep learning in clinical natural language processing: a methodical review. *Journal of the American Medical Informatics Association*, 27(3):457–470, 2020. 1

[50] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. *CoRR*, abs/1602.02658, 2016. URL http://arxiv.org/abs/1602.02658. 4