Generation of Explanations in Reinforcement Learning

Renato Cardoso[1]

Abstract. Vanilla Reinforcement Learning has a clear interpretation on its solution based on the maximization of the total discounted reward. Nevertheless having a mathematical explanation does not make it intuitive and easy to understand the resulting policy even in small problems.

We should be able to explain to a user why a given solution is better than an alternative beyond saying that the cost of the alternative is higher. We consider 6 different types of explanations for different questions a user might have. The question we aim to answer are Why this action in this state, How to reach the optimal state, Why not a different path, What happens in this path, What happens in a state and What happens when the policy is executed with errors.

We contribute with a toolkit to provide explanations, new algorithmic tools to generate explanations, and show several examples to illustrate the approach.

1 Introduction

Nowadays machine learning is used in a widespread of critical areas like financial services, medicine and education with a great potential to improve safety and efficiency. To achieve these goals we need to trust these systems, and understand their answer.

This is even more important in cases where we relinquish control to the machine as is the case when using reinforcement learning methods.

Reinforcement learning is usually considered to be interpretable because there is a clear mathematical interpretation to its solution, e.g. the maximization of the total expected reward. But having this interpretation does not help a user, that might not even be true to an expert in reinforcement learning, really understand the why of a solution or why an alternative is not as good. This is clearly the case when the reward is not aligned.

Interpretability is thus more than improving the mathematical interpretation of an algorithm. It is to understand complex solutions, help humans, even without expert knowledge in machine learning, to diagnose and correct machine learning models.

1.1 Problem Description

The problem at hand is the interpretation of reinforcement learning policies, more concretely the optimal policy for a Markov Decision Process, looking at the results during its training and final solution. For that we will be looking into possible interpretations and explanations for machine learning methods, as well as, different types of possible solutions to the problem.

[1] Instituto Superior Técnico, Portugal, email: renatocardoso1996@tecnico.ulisboa.pt

1.2 Contributions

The Contribution of this thesis will be:

- The formalization of several questions that can be posed by an user regarding Reinforcement Learning policies, namely questions about why a certain action is better for a certain state, or how a certain path behave.
- Construction of a toolkit that provides the interpretation for the queries related to the users questions.
- Help expert and non-expert users understand the decision and solution of reinforcement learning.
- Help users be able to create better representations for reinforcement learning models.
- Inform the user of the various types of complexities in a model based MDP, namely the bifurcation of a path or how likely is the path to make errors.

2 Background

2.1 Reinforcement Learning

We consider a standard reinforcement learning problem where the agent wants to maximize the total delayed reward.

$$V^*(s) = \max_{\pi} \mathbf{E}(\sum_{i=0}^{\infty} \gamma^i r_i)$$

Where π is the policy followed by the agent, γ is the discounted factor and r_i is the reward obtained when following the policy at step i.

We can model it as a **Markov Decision Process(MDP)** [7, 2, 3, 6, 11], which consists of:

- Set of States **S**
- Set of Actions **A**
- a reward function R : S $\times$ A $\rightarrow \Re$
- a state transition function T : S $\times$ A $\times$ S $\rightarrow \prod(S)$, where $\prod(S)$ is a probability distribution over the set S. We can write T(s, a, s') for the probability of making a transition from state s to state s' using action a.

Now that we defined what is a MDP, and before considering learning algorithms, it is important to define techniques for determining the optimal policy. As said before, we will be considering the infinite-horizon discounted model and we will be relaying that it exists an optimal deterministic stationary policy for this type of model [2].

The optimal value of a state is defined as the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Using π as the decision policy and

model decided previously, using the Bellman's equation we can define the optimal value, this can be written as the solution of multiple equations, using actions and states:

$$V^*(s) = \max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), \forall s \in S \quad (1)$$

which means that the value of state s is the expected instantaneous reward plus the expected discount value of the next state when used the best action possible. Given the optimal value function, we can now define the optimal policy as:

$$\pi^*(s) = \arg\max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')) \quad (2)$$

To find an optimal policy we can find the optimal value function, this can be determined by an iterative algorithm called **value iteration** that converges to the correct V^* values [2, 3].

The problem with value iteration is that, it is not obvious when to stop. One way is to use the difference between two successive value functions, so that if the maximum difference between two successive value functions is less than a certain value ϵ, then the value of the greedy policy differs from the value of the optimal policy by no more than $2\epsilon\gamma/(1-\gamma)$ at any state [15]. Another important result is that the greedy policy is guaranteed to be optimal in some finite number of stops even before the value function has converged [3].

2.2 Distributional Representation

It is possible to use a distributional model to train and show the results of a reinforcement learning policy [1], in contrast with the common approach which uses the value to model the expectation of the result, this approach uses a value distribution to model the random return of the agent.

The main objective of this paper, is to model the random return Z, whose expectation is the value Q, following a recursive bellman equation:

$$Z(x,a) \stackrel{D}{=} R(x,a) + \gamma Z(X', A') \quad (3)$$

This equation states that the distribution Z for a state x and an action a, is characterized by the reward the same pair state and action, plus the next state-action (X',A') and its random return Z(X',A'). The authors called this quantity the value distribution.

The authors then propose an algorithm to calculate the distribution following the bellman optimality operator. They use, as parameters, N $\in N$ as the number of atoms, V_{MIN}, $V_{MAX} \in R$ as the lower and upper limit of the distributional values, and the set of atoms $z_i = V_{MIN} + i\triangle z : 0 \leq i < N$, $\triangle z := \frac{V_{MAX} - V_{MIN}}{N-1}$. This atoms are the canonical returns of the distribution that are given by a parametric model $Z_\theta(x,a) = z_i$.

The authors further elaborate that the update TZ_θ and the parametrization Z_θ almost always have disjoint supports, to correct this they project the sample Bellman update TZ_θ onto the support of Z_θ.

Furthermore the authors introduce an algorithm called categorial algorithm 1 that receives a transition $x_t, a_t, r_t, x_{t+1}, \gamma_t$ and returns the updated distributed probability for the state-action pair (x_t, a_t), following the recursive bellman equation.

It is important to notice that the authors use a epsilon-greedy search while training the model and they state that the best value for N they got was 51 thus they used the name C51 when referring to the algorithm.

Algorithm 1 Categorial Algorithm

Require: A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0,1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, ..., N-1$
  repeat
    for $j \in 0, ..., N-1$ **do**
      {# Compute the projection of $\hat{T}z_j$ onto the support z_j}
      $\hat{T}z_j \leftarrow [r_t + \gamma_t z_j]_{V_{MIN}}^{V_{MAX}}$
      $b_j \leftarrow (\hat{T}z_j - V_{MIN})/\triangle z$
      $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
      {# Distribute the probability of $\hat{T}z_j$}
      $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$
      $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$
    end for
  until policy good enough

Even though this algorithm is implemented using deep reinforcement learning with some alterations it is possible to use it in standard reinforcement learning, moreover the ability to already have the distribution of all the pairs state-action is very useful when trying to explain the resulting value model of a certain MDP.

3 Related Work

In order for a model to be interpretable various authors define different types of explanations. The work in [8] enumerates 8 types of explanations and definitions, the first four being model independent: **Inputs explanations**, **Outputs explanations**, **What explanations**, **What If explanations**, and the other four model dependent: **Why explanations**, **Why not explanations**, **How to explanations**, **Certainty explanations**, in our case we are focused on explanations that could be used to explain the policy a reinforcement learning agent reaches.

Another author that takes a look to what interpretability is) [9]. Instead of focusing on explanations this author tries to explain where the necessity of interpretability arises, as well as, characterize what should be an interpretation and the properties of an interpretable model. The paper says that the demand for interpretability arises when there is a mismatch between the real world costs and the formal objectives of the application. This means that, normally the evaluation metrics employed by the application require only predictions and ground truths, however sometimes these metrics are not enough to characterize the model because our real world objectives are difficult to code as real-valued functions. He specifies that the necessity of interpretability comes from the user needing to **Trust** the results [12], to be able to be as **Informative** as possible, having the solution **transparent**, transparency salients that the user, when shown a solution, can retrace the steps that lead to that solution.

Finally the paper [9], defines that there are post-hoc interpretability, used to best present the user the explanation obtain from a certain question. IN our case we are interested in **Text explanations**,explanation using simple text, and **Visualization**, explanations using images such has graphs.

There are a few examples of successfully usage of explanations to clarify the solution obtained in a machine learning problem. For example, the LIME algorithm [12] which shows in which part of the input the solution is getting its information, that way differentiating

good explanation from bad explanations.

To get some information about what it means to better understand a solution we can see the works that try to provide extra cues to allow a reinforcement learner learn faster. There different types of training some of which explain the behaviour of the agent, such as [4] that uses cues to train a robot dog, this training cues can then be further used to explain the behaviour of the agent. Another author [13], uses training by human guidance, the same way as before, the guidance provided can then be used as explanations.

For reinforcement learning the explanations proposed vary depending on the problem at hand. One author uses a new type explanations, which the paper defined as costrastive explanations [14]. The paper uses this type of explanations in order to answer 'why' type questions, with user input and policy/reward manipulation to obtain an explanation. Another, focus on explanations to robots using reinforcement learning [5], leading to a context awareness of the agent, as well as, a focus towards explanations about its control logic and its internal representation.

3.1 Contributions

When looking at works outside the area if reinforcement learning we noticed that a considerable amount of research has been done. For instance if we take a look at Neural Networks, such as the LIME algorithm, due to its inherited black box nature, explanations are in demand to confirm the validation of the results, in order to help the user trust the model obtained.

On the other hand when talking to reinforcement learning models we see that two aspect come up. For one, we have research on how we can train a model an what better ways can we train it so the user as a better understanding on how it behaves, which could be used to explain the model. However when talking specifically on interpretation in reinforcement learning, we see that not much research has been made, and the one that has been made is limited to small policy manipulation.

Our contributions is a toolkit that aims to explain the behaviour of a reinforcement learning agent, focusing on the policy obtained. We will not only be using policy manipulation but also path manipulation, while taking a look at elements such has bifurcations and security for a certain policy. We aim that our toolkit is more related to a deep understanding of the policy in the same order that we could verify when explaining Neural Networks.

4 Types and usage of explanations

Explanation Type	Information Extracted
What if	Different Inputs/Redefinition of the MDP
What	Inform about actual/previous state
How to	Reach a certain state
Why Not	Possible Paths/Possible Actions
Why	State Explanation
Certainty	Possible Paths / Bifurcations / Security

Table 1: Explanation Type and what information they refer to.

We will be taking a look at the previously mentioned 8 types of explanations [8], by dividing the explanations by which part of the application they will explain, this is where they will be extracting information, see table 1.

If we look at a reinforcement learning problem we see that input and output are not defined in reinforcement learning, so when looking to the explanations we will be looking instead to the definition of the MDP, the various states, and the correspondent action obtain from the policy to a certain state. Furthermore when talking about distributions we will be looking at reward distributions, this means values of reward obtained by the agent and the probability of achieving each of these values of reward.

4.1 Why this action in this state

The first type of explanation is **why this action in this state**, this is used when the user has a question about the action, defined by the policy, for a certain state. The objective of this explanation is, given a certain state, for the toolkit to generate a set of explanations, that shows why this action is the best.

This question is important because it lets the user identify what results can be produced for each of the actions in a state. And after identifying each of these results the user can then compare each of them to conclude, for himself, what is the best action.

The answer to this question should be a distribution for each action the user has doubts about. The distribution should be clearly enough so the user can identify to which of the actions this distribution represents. It needs to show each of the actions individually or be able to show the whole conjunct of actions.

Due to the nature of this explanation once we know why this action in this state, it is also possible to show **why not another action**, by showing all the distributions for the actions of this state.

4.2 How to reach the goal state

To further the previous explanation, the toolkit is also prepared to, given a state, generate all the paths that start at that state and end at a goal state, this is, all the optimal paths. And by joining both type of explanation the user is able to obtain the optimal path and the distribution values of the optimal path.

There are two elements that make this explanation important, one is if the user doesn't actually know what the optimal path is and/or how to reach the goal state. The second important element is, to show if there are more than one optimal path besides the one specified by the optimal policy.

The answer is the elements that constitute any of the optimal paths starting at a certain state, this means that it should include the state, the action that it will do and the state it will go to, all this elements until it reaches a certain goal state.

4.3 Why not a different path

With the previous why explanation we were able to explain a certain pair state-action, however in this explanations we explain **why not a different path**. While the previous one did the optimal path after it forced the action, with this explanation, the agent follows a specific path, and the action between each state in the path, to generate a distribution based on the user conditions. It is obvious that if a path has bifurcations, a certain pair state-action might lead to a state outside the path, meaning that it will not fully complete the path, resulting in a number of failed paths.

The importance of this explanation is that it uses a path defined by the user and compute the reward distribution of this path. And after identifying the result produced by this path the user can compare it to

the results obtained from the optimal path, concluding, for himself, what is the best path.

The answer for this question its a single distribution for the path the user has defined, as well as, the number of times it has failed while doing the path. The user need to obtain two different distributions, to be able to compare two paths.

4.4 What happens in this path

It was stated in the previous explanation that the path could fail due to bifurcations presented on the path and to help the user identify this bifurcations we had the explanation to know **what happens in this path**.

This explanation is important because it explicitly shows the user the points where the path may differ from what is expected. In the previous section we said a path could fail due to bifurcations, this explanation identifies said bifurcation, letting the user understand why it failed.

The answer to this question should be for each pair state-action in the path a list of bifurcations, the possible next states for a pair state-action.

4.5 What happens in a state

We already shown Why this action in this state, however in that explanation we were forcing the action after which it would follow a certain optimal set of actions until the goal. In this explanation we will see **what happens in a state** by using only distributions of the states it can make a transition to.

The importance of this explanation is to show the full spectrum of the distributions for a state, and if the user decides to use a certain pair state-action we can see how the bifurcation of a pair state-action will behave, to see if it is the best option or not.

The answer to this question will be a distribution that shows how the agent behaves for each of the bifurcations, so it is important to two types of answers can be achieved. One if the user asks about a certain number of actions, so the distribution for each action will be shown, similar to the first question. Secondly if the user only wants to know about the bifurcation of a single pair state-action then it is more important to show the distributions of the next states, so the user can see which is the best.

With the detail of showing, the various distributions of the bifurcations, we can also show how **certain is this pair state-action**.

4.6 What happens when the policy is executed with errors

Finally we look into cases when there is a certain probability of occurring an error when executing the optimal policy. For that we use the explanation to see **what happens when the policy is executed with errors**, by running a certain path, the agent has a certain probability to fail to do the action specified and do a random one, with this we are able to see how secure a certain path is, as well as, to see how **certain is a path**.

It is important to use this explanations because sometimes, in the real world, the agent might do another action besides the one it is specified by the optimal policy. We use this explanation to show the impact of failing to do the optimal action, and how much this impact the values of reward obtained.

To answer this question we will be using a distribution that varies depending of the probability of doing a random action. The user will be able to generate an explanation for each set of paths and probabilities of doing a random action.

4.7 Presentation of the results

With this we have showed the various types of explanations that we are going to employ, as well as , what kind of information we are expected to explain. However we are still left to touch on the subject of presentation, that is how do we intend to present the results.

For that we take a look at post-hoc interpretability [9], in there we can identify two main way to present our results, those are **text explanations** and the other is **visualization**.

Visualization will be main way to way to present the results. Visualization generates explanations by rendering a 2D visualization, in our case we will be using a distribution approach, by showing the graph produced by such approach the user can understand how the values of the policy are distributed, as well as, easily compare a certain distribution with another one. These distributions will be using the values of reward obtained by the agent and the probability of achieving each of these values of reward.

On the other hand, text explanation serve a huge purpose when trying to justify a certain decision, mainly when the decision could be justified verbally. In our case, there are certain things that cannot be convene as a simple graph, such as possible bifurcations in a path, or other possible path for a certain policy, thus using text explanations for such cases.

5 Explanation Generation Algorithms

In this sections we will be defining the various types of algorithm that make up the core elements of the explainer. We will be using the already defined explanations in the previous section, defining the algorithms that produce the appropriate answer. We have implemented the following algorithms in python [2].

5.1 The state explainer

The state explainer is used when questions about a single state are presented, this means that the objective of this explainer to answer two question, **Why this action in this state** and **How to reach the optimal state**.

5.1.1 Why this action in this state

When defining this explanation we stated that the objective is to generate a set of distributions that show why a certain action is the best action. In order to achieve this the algorithm needs to, starting at a certain state, follow the optimal policy until it reaches a goal state, returning the number of steps and the reward obtained. By doing this a certain number of times we can obtain a distribution of the values for the reward and steps.

An important notion of this algorithm is that, as it ends in the goal state it will not accumulate the reward of staying iteratively on the goal state, which happens in the value iteration.

There are two main variations of this algorithm. Though both of them are very similar and use the same base algorithm, variations in the overall algorithm make it possible to reach different conclusions.

[2] https://github.com/recardoso/Toolkit-Interpretability-in-Reinforcement-Learning

The first variation restrain the overall environment by directly changing the transition matrix so the only action possible to do in the state relevant for the question is only one, so if it goes back to this relevant state it will need to do the action stated by the user. This will show how important is this transition, making that sometime without being able to do other actions in this state the agent can't reach the goal state.

The second variation doesn't restrain the environment, it only makes sure that the first action that is made by the agent is the one specified by the user in the relevant state. If the agent might then decides that it is best to immediately return to the relevant state and do another action, or at anytime if it ends at the relevant state again it might do another action. The objective of this variation is to see the impact of starting an action further.

Both variation start by initialization the dictionaries that will contain the distributions, this means that, the item of the dictionary is the value obtained for the reward, and the value the number of times that reward was obtained.

Algorithm 2 Single point run

Require: A policy Pol and a start position x
  Initialize: $pos \leftarrow x; action \leftarrow Pol[pos]; steps \leftarrow 0; reward \leftarrow Rl[pos, action] \times \gamma^{steps}$
  repeat
    $pos \leftarrow$ random choice from $Pl[pos, action]$
    $action \leftarrow Pol[pos]$
    $steps \leftarrow steps + 1$
    $reward \leftarrow reward + Rl[pos, action] Rl[pos, action] \times \gamma^{steps}$
  until pos is goal

The algorithm for the first variation begins by saving the original transition matrix, followed by changing the transition matrix so that it can only go to another state following the action which has chosen. This will maintain the original probability of transition of that action to any other state, however only in the chosen state, all other actions will have a probability of 0 to transit to another state.

After this change it then needs to recalculate the new Policy and it verifies if there is a possible next state, this is necessary, for example, when a certain state is a border state and the user would chose an action that is impossible.

Now the only thing that needs is to follow the new policy until it reaches a goal state, using the algorithm 2. It repeat this step n times in order to obtain the correct distribution.

Finally it need to correct the transition matrix using the original, saved previously, and in order to obtain the values of the distribution in probabilistic terms it divides every value of the distribution by the value n.

The algorithm for the second variation is near similar to the previous one, however, in this case we don't need to change the transition matrix but just calculate all possible states for this pair state-action, followed by the algorithm 2 n times, this time starting at the next-states.

However in order to obtain the values for the initial state we need to join all the values obtained from the various next-states, for that we multiply the values of the distribution by the probability of the transition to that next-state, following equation 4.

$$distribution[s, a, s'] = distribution[s, a, s'] \times Pl[s, a, s'] \quad (4)$$

The rest of this algorithm is identical to the previous one.

5.1.2 How to reach the optimal state

By the definition of the explanation, the objective of this algorithm, is to inform the user the possible optimal path that exist starting from a given state until it reaches a goal state.

In order to correctly determinate the optimal paths, it is kept a list of explored path, and with this the user has two option on how the results are presented, the list of optimal paths can show path cycles and show self-transition. The path cycles are transitions to an element already in the path, the objective is to not present already visited paths.The self-transitions are transitions from an element to itself.

The algorithm is a recursive algorithm, that receives a single state and the explored path, advancing in case of a not explored next-state and stops in a goal state, returning a list.

Even if in the optimal policy there is only one possible action, we use the Q-values to determine all the equal valued actions for a certain state, guarantying that all the optimal paths are chosen.

5.2 The path explainer

The path explainer includes question directly correlated with the path, and is used to compare different paths, explaining what is the best and to clarify certain elements of the path. This include the questions **Why not a different path** and **What happens in this path**.

5.2.1 Why not a different path

As stated previous the objective of this algorithm is for the agent follows a specific path, and the action between each state in the path, to generate a distribution based on the user conditions. The agent will be following the algorithm 3.

The algorithm for the path explainer is similar to the algorithm 2, however in this case as we are following a certain path, the distribution comes from this path and this path only. This means that if, when doing this path, the pair state-action results in a next-state that is not the next-state predefined by the user, then this will count as a incomplete path, seen in the algorithm as a $return false$, and the total probability of obtaining the rewards will be lowered because of this, meaning it will not add up to one. The number of incomplete path will be accessible for the user.

5.2.2 What happens in this path

The objective of this algorithm will be to help the user identify the bifurcations present in a certain path. Besides the ability to given a path identify the bifurcations by tuning down the algorithm it is possible to show the bifurcations to a single state.

With this in mind there are three specif ways to inform the user of bifurcations, by showing the bifurcations in a path, removing the bifurcations that doesn't make the path fail, by showing all the bifurcations in a path and by showing only the bifurcation of a single state.

For the **path bifurcation** we are only trying to see the next-states that are not in the defined path. In order to achieve this the algorithm will iterate through the elements of the path, except the final one, adding to a list the next-states that, for the state-action in the path being iterated, have a transition probability bigger than 0, excluding those that are already present in the path.

Algorithm 3 Path run

Require: A path $path$ and the actions for the path $pathaction$
  Initialize: $pos \leftarrow path[0]; i \leftarrow 1; action \leftarrow pathactions[i-1]; steps \leftarrow 0; reward \leftarrow Rl[pos, action] \times \gamma^{steps}$
  repeat
    $nextpos \leftarrow path[i]$
    $action \leftarrow pathactions[i-1]$
    $state \leftarrow$ random choice from $Pl[pos, action]$
    if $state$ is $nextpos$ **then**
      $i \leftarrow i + 1$
      $pos \leftarrow nextpos$
    else if $state$ is pos **then**
      continue
    else
      if $state$ in $path$ **then**
        $pos \leftarrow state$
        $i \leftarrow path.index(pos) + 1$
      else
        return false
      end if
    end if
    $steps \leftarrow steps + 1$
    $reward \leftarrow reward + Rl[pos, action] Rl[pos, action] \times \gamma^{steps}$
  until $i > path$ size

The reason this exist is in case the user wants to see potential bifurcations outside the path. In case the return is an empty list for all elements in the path, then the conclusion it that this path is the only way, starting at the initial state, to reach the destination, following this actions.

On the other hand, for **all the path bifurcations** the algorithm is similar, it will iterate through the elements of the path, except the final one, adding to a list the next-states that, for the state-action in the path being iterated, have a transition probability bigger than 0, however in this case we will maintain elements that are in the path.

In this case if there is only one way starting at the initial state, to reach the destination, following this actions, it will return a list with each element in the path. This is the most conventional way of showing the bifurcations, however is it harder to identify that there are no bifurcations in the path.

As an added feature it is possible to see the **state bifurcations**. In this case the algorithm will receive a pair state-action and in it will return a list with all the states in which the probability of transition from a this pair state-action to another is bigger than 0.

5.3 The bifurcation explainer

The bifurcation explainer is solely used to answer the question **What happens in a state**, as in this question we are mostly interested in the bifurcations for a certain state.

5.3.1 What happens in a state

The objective of this algorithm is, by using distributions, to explain the bifurcation for a certain state, by iteratively, calculating the distributions for the states we can reach the optimal distributions, just like how we do with value iteration to obtain the Q-values. This algorithm can also be used as an alternative for the state explainer.

This algorithm used the aforementioned Categorial Algorithm 1, however certain alteration where necessary to do so that the algorithm would work for model based reinforcement learning.

Though the alterations for the Categorial Algorithm are not major and are corrections to it, the important additions to this is in the initialization and the post-processing of the results that make the algorithm work for our toolkit.

The first step of this algorithm is the initialization of every pair state-action, the original C51 algorithm obtains the probabilities from neural networks, following a more model-free kind of behaviour, however in our case we can't do that. In our case, we initialize the distributions, with probability equal to one for the value correspondent to the reward of this pair state-action. We do this for to reasons, first by initialization with the reward we will have one less step in our iteration, the other is if we didn't initialize the values, as the Categorial Algorithm uses the best next-state for the pair state-action distribution, if the values of the probability are 0 so will be this ones, resulting in a global 0 distribution.

After this the C51 algorithm proposes an epsilon-greedy search to obtain an approximation to the optimal values of the distribution, however in our case we can employ a full iteration for every transition state, action, next-state, where the probability is bigger than 0, stopping when the difference between the previous iteration and the new iteration is small enough. We do this in order to obtain the optimal distribution, that his almost guarantied to happen, in contrary to the epsilon-greedy search. However one down side is that this could possibly take more time than the epsilon-greedy search.

Another important thing is how are we going to transform the values resulted from the Categorial Algorithm when the Categorial Algorithm one only takes a single transition, as an argument, instead of all possible transitions for a single pair state-action. For that we need, for a certain pair state-action, to do all the next states one after the other, multiply the probability by the probability of the transaction to that next-state and then joining all together.

To follow the same thought as the value iteration, that maintains the reward even if there is no transition, we will maintain the reward value even if the Categorial Algorithm cannot be applied.

With this algorithm we obtain all the distribution for every state and by using the same method we used when transforming the resulted values, we can show the user the distribution of a certain state bifurcation.

5.4 The security explainer

The security explainer is used to see **What happens when the policy is executed with errors**, which show how secure a certain path is.

5.4.1 What happens when the policy is executed with errors

The objective of this algorithm is to obtain a distribution that shows how secure is a certain path, following a certain epsilon probability of failing to do an action in the path. One thing important to notice is that the path needs to end at a goal state, because of how it behaves in case it fails in any moment in the path, this is by trying to follow the optimal policy for that moment on for instance.

The algorithm 4 is similar to the algorithm for the path explainer however in this case we try to solve how the agent will behave in case it fails the path, for that we use 3 methods, it can always return to the path, it can return to the path if the place it returns is the previous state or a better state, or it can follow the optimal policy until it reaches

Algorithm 4 Secure Path run

Require: A $path$; the actions for the path $pathaction$; epsilon ϵ;
Require: flags: $Alwaysreturn$; $returnifbetter$; $Dontreturn$
  Initialize: $pos \leftarrow path[0]; i \leftarrow 1; steps \leftarrow 0; pathexplored \leftarrow []; reward \leftarrow 0$
  repeat
    if pos in $path$ **then**
      get random probability $prob$
      if $prob >= \epsilon$ **then**
        $action \leftarrow pathaction[i-1]$
      else
        $action \leftarrow greedyaction(pos, \epsilon)$
      end if
      Add pos to $pathexplored$
    else if $Alwaysreturntopath$ == **true then**
      $action \leftarrow alwaysreturnpath(pos, path, pathexplored, \epsilon)$
    else if $returntopathifsameornew$ == **true then**
      $action \leftarrow returnpathifbetter(pos, path, pathexplored, \epsilon)$
    else if $Dontreturntopath$ == **true then**
      $action \leftarrow greedyaction(pos, \epsilon)$
    end if
    $reward \leftarrow reward + Rl[pos, action] \times \gamma^{steps}$
    $pos \leftarrow$ random choice from $Pl[pos, action]$
    if pos in $path$ **then**
      $i \leftarrow path.index(pos) + 1$
    end if
    $steps \leftarrow steps + 1$
  until pos is goal
  $reward \leftarrow reward + Rl[pos, action] \times \gamma^{steps}$

the goal state or until it gets back to the path. Only one of this option can be active at a time. However even when trying to go back to the path it has a chance of doing a random action, equal to the value of epsilon.

By iterating through the transition matrix of a pair state-action it is possible to identify which are the next states, and if this next-state is in the path we know if the action can return to the path.

In a standard type of problem in which the security of a path is more concerning, such as a cliff-walk, it is normal that in case it fails the path it gets back to the initial state and retries, however that is the same thing as the path explainer.

For the **Always return to path** part of the algorithm, the algorithm will always try to go back to the path, even if is a worse option than continuing, for this reason the first thing it does is to calculate which action have a transition to the path and of this actions which are a better state, not yet explored, or those that have already been explored, by using the algorithm 5. With algorithm 5 we get the list of explored actions and not explored.

After that, and in case is not doing a random action, it will choose one of the action that are better, one that lets it return to a non explored part of the path, if it can't it will chose an action that goes to an explored part of the path, only then, if it is not possible to return, it will do the best possible action, following the optimal policy.

In contrary to the previous possibility, if chosen the option to **Return to path if same or new**, after calculating which action have a transition to the path and of this actions it will only maintain those that are not explored yet. This means that in algorithm 5 it will only keep the list of not explored actions.

After that, and in case it is not doing a random action, it will chose between the actions that are not explored and if it cant it will go for

Algorithm 5 Always return path

Require: The position pos; path $path$; list of explored path $pathexplored$; epsilon ϵ;
  Initialize: $actioninpath \leftarrow$ **false**; $actionexplored \leftarrow$ **true**; $listexploredactions \leftarrow [\,]$; $listnotexploredactions \leftarrow [\,]$
  for all actions **do**
    if action returns to $path$ **then**
      $pathinaction \leftarrow$ **true**
      for all nextstates **do**
        if nextstate not in $pathexplored$ **then**
          $actionexplored \leftarrow$ **false**
        end if
      end for
      if $explored ==$ **true then**
        add action to $listexploredactions$
      else
        add action to $listnotexploredactions$
      end if
    end if
  end for

the optimal action.

Finally, in case it chooses to **Don't return to path** it will simply try to do the optimal action if is not doing a random action. If it ends up returning to an element of the path it will then try to do the path from that point onward.

6 Tests and Validation

We now present two examples of the use of our approach to explain reinforcement learning policies. We will be showing the results using 2 examples in order to show how different types of problems can be explained, and how the toolkit approaches them.

The first one, figure 1 is a random generated MDP with a reward of 750 at the goal, a reward of -10 at the penalties and a reward of -100 at the failures, this one followed by a transition to state 0, the blue lines will make so that half the time it can continue the other half it will stay in the same state. This example is a the standard for our toolkit as it is the easiest type of problem to obtain an explanation.

The second example, figure 2, is a mountain car problem, following the equations presented by Andrew Moore [10]. The state position x, in meters, will vary between [-1.2, 0.5], and the velocity, in meters per second, will vary between [-1.5,1.5], with three possible actions being left, no operation, and right. It was necessary to use a discretization approach in order to be able to use this example with our toolkit. We divided the space for the positions and the velocity into 30 chunks each, combining the position chunks with the velocity chunks, we obtained a state space with 900 elements. This example is a standpoint for reinforcement learning problems and it is very common to use it to validate the results of an algorithm related to reinforcement learning.

We will be intercalating the results between each of the examples depending on which is better suitable to explain a certain question.

6.1 The state explainer

The state explainer can answer two specific questions **Why this action in this state**, **How to reach the optimal state**. Both examples can be used to explain this questions, however, as the second exam-

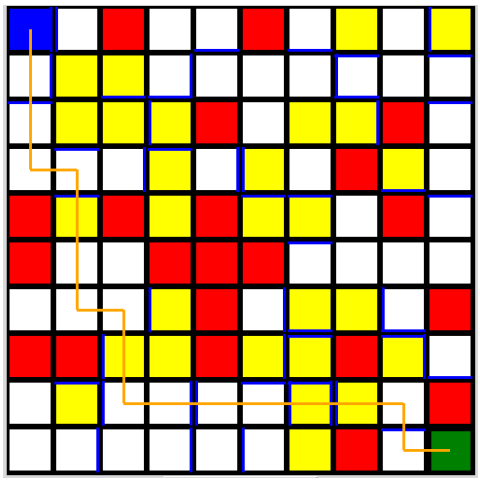

Figure 1: Example 1; A 10x10 random generated grid, with probability of transition equal to 1 where the side lines are black and 0.5 where the side lines are blue. The blue square is the position 0, the green the goal, the red a failure and the yellow a penalty. At The orange line represents a possible optimal path.

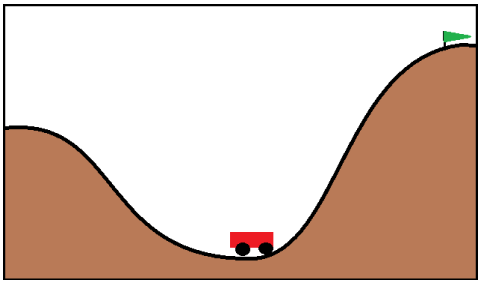

Figure 2: A standard mountain car problem, where the agent, the car, needs to reach the goal, represented as a banner, by gaining momentum going up and down the two hills

ple is harder to understand we will be using this example to explain the state explainer.

The first thing we will be looking into is the question **How to reach the optimal state**, one thing to notice about the toolkit is that even though the user can pass as an argument the optimal policy and correspondent values, it is not mandatory. When the values for the policy are not passed, the value iteration algorithm will be run obtaining then the optimal policy, after which the user can use the toolkit to obtain the optimal path for the optimal policy.

In this example we will be using the position -0.5 as the initial position and a velocity of zero, which corresponds roughly to state 374. With this state we can then use the toolkit to obtain the optimal path.

We can then translate the states, the same way we did when preparing the problem for the toolkit, and obtain a graph showing the variation of the position and the velocity for the optimal policy, figure 3.

Looking at the results from the optimal path we can see that the best actions are to go to the left, gain velocity, and then go to the right until it reaches the goal state. With this we validate the result for the optimal path, and with this we can proceed to show the results for the rest of the toolkit.

Next we will be using the two variations of the state explainer to

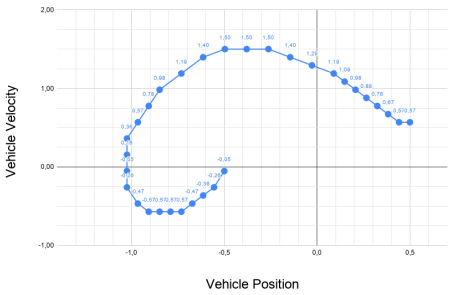

Figure 3: The optimal path for the mountain car problem, presented in a chart

obtain the results for the state 374, as this present different conditions for the agent. The result are in figure 4.

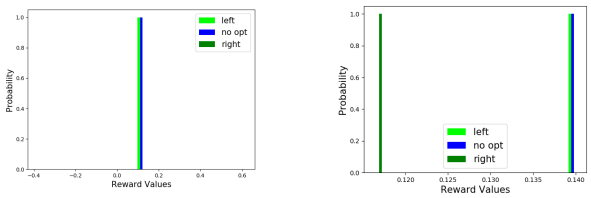

Figure 4: Results for Test1 at the left and Test2 at the right, for the state explainer of the mountain car example.

First thing we can notice is that the result for the distribution are a single straight line for each of the actions, this is due to the probabilities of transition for each of the states being equal to one, so the results being a single bar are the correct result.

Again the objective of this explainer is to know **Why this action in this state**, and in this example we can take different conclusions for each of the variations.

If we look at the second variations, the one on the right, we can see that the best action will either go left or do no operation. The LEFT action is the obvious choice however, the reason to the no operation being having an equal values is due to the nature of the problem, the problem was divided in 900 states however the values for the states are not the exact number, for instance the values for the state 374, used in here is (-0.496, -0.0517), which is the closest to the one we where after (-0.5, 0.0) however it as a small velocity to the left thus resulting in the no operation action being also suitable.

When we look at the results from the first variation, we can see why the RIGHT action on this state is so bad. Looking at the result, the graph for the RIGHT action doesn't appear, meaning that only doing the right action on state 374 will make the agent unable to reach the goal, thus putting more importance on why the left action on this state.

With this example it is possible to show how the user can draw conclusions for the problem that are hard to see only looking at the optimal policy.

6.2 The path explainer

For the path explainer tests we will be using the grid based example, because with this example we can easily choose a certain path leading to the goal. We will be using two paths one being the optimal

path marked in orange and an alternative path marked in purple in figure 5.

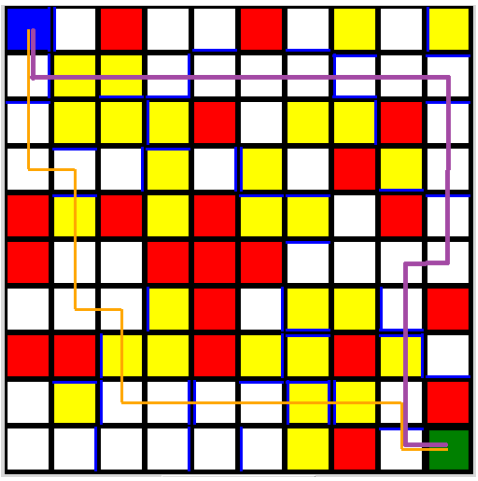

Figure 5: Two paths for the path explainer test, in orange the optimal path and in purple an alternative test.

The objective of this test is to show **Why not a different path**, in order two do that besides the optimal path we introduced an alternative path, that though it takes more steps than the optimal path, the number of penalties it passes through is smaller. With these two paths and the results in 6 we aim to answer the question above and validate the behaviour of the toolkit.

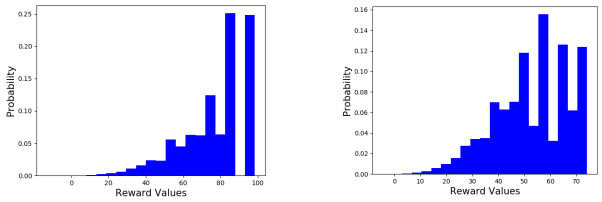

Figure 6: Results for the test for the path explainer, the optimal path on the left and the alternative path on the right

First thing that it is possible to verify is, which one of the paths is the best. In this case by looking at the results we see that the optimal path reaches higher values of reward with higher probability, a probability of 25% of having a reward rounding the 100, contrary to the alternative path which reaches a reward of around 75. This is what it should be expected, as the optimal path is following the optimal policy, thus validating the results obtained.

Further if we use the toolkit to see which bifurcation are in the path, to see **What happens in this path**, we obtain only 2 states with 2 next states, with a bifurcation, states 83 and 86. Both of them with a a transition to itself and to the next element in the path. Though none of this bifurcation make the agent leave the path, it only increases the total number of steps.

Another thing we can look are the behaviours of the algorithm, previously we mentioned that for a certain example, the paths could be incomplete if a bifurcation existed, however in this case even though there are bifurcations, neither of this bifurcations make the agent leave the path, that way it completes the paths, thus the accumulative probability of the distribution is equal to 1.

6.3 The bifurcation explainer

The objective of the bifurcation explainer is to answer **What happens in a state** for that it is best to use a pair state action that has a bifurcation. From the two previous examples the only one that has bifurcation is the grid example.

In this example we can look at a pair that has a 50% chances of continuing to the next state and a 50% chances of staying in the same state. We can easily obtain such pair by looking at state 83, and the RIGHT action. In this case it seem important to test both the option to show all action for a single state and only one action for a single state. The results are in figure 7.

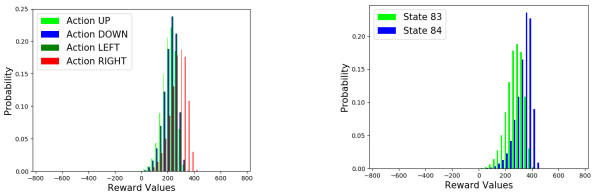

Figure 7: Results for the test for the bifurcation explainer. All action on the left only the RIGHT action on the right.

As we can see, on the first plot we only see the division by actions, which represents a similar behaviour to the previously mentioned state explainer, however one major difference is imposed, while on the state explainer the objective is to distinguish the various actions to see which is the best, in this case we want to known what happens in this state, thus instead of the trajectory of a state ending in the goal state, it will instead take the information from the next states to obtain the distribution. And we can observe that the best action is the RIGHT action.

On the second plot we can see the division by possible states of transition from state 0 using the right action. The objective of this algorithm is to show the bifurcations, so when only on action is passed by default it will show instead the bifurcations by possible states. In this case we can clearly see that the best action is to continue to state 84 instead of staying in state 83, which is the obvious answer.

6.4 The security explainer

The objective of the security explainer is to see **What happens when the policy is executed with errors**, no matter how small the probability of occurring errors is the behaviour of the agent is what we are trying to evaluate.

For this test we will be using the mountain car example, as this one is the most likely to have drastic result for the smallest probability of errors. We intend to show how a simple alteration of 10%, on the optimal path, can change the results. To that end we obtain the results in figure 8.

Looking at the optimal path, epsilon equal to 0, we can see a distribution of 100% around the 0.1 value of reward. In the result obtained for the epsilon equal to 0.1 we can see that this line still maintains however the value of the probability has decreased to 80%. The rest of the probability is distributed to lower values of reward.

This shows how flimsy the optimal path is, that a small of 10% actually change the values in 20%, this is due to the nature of the

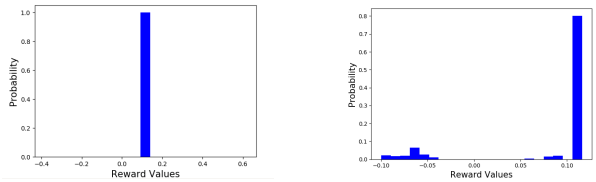

Figure 8: Results for the security explainer for the mountain car problem. Epsilon equal to 0 on the left and epsilon equal to 0.1 on the right.

problem. Doing an incorrect action might lead the agent to restart the whole process again which lead to an increasing number of steps, decreasing the overall reward.

The objective of this example is to show that the security explainer can easily identify high risk path, as well as, the consequences of deviating from said path.

7 Limitations and Discussion

Now that we have fully explained the various elements of our toolkit as well as, showed a few cases of tests and examples for the toolkit we need to address the various limitations, not only of the various algorithms but also the whole toolkit.

The particular element of our toolkit is the necessity to specify the goal states, in the case of the state explainer or the optimal path informer it is imperial that the goal state is specified, otherwise the algorithm would not know where to stop. Though this doesn't limit the user to use the whole toolkit, it means that if the user doesn't know before hand which are the goal states, this explanations are not usable, thus limiting the functionalities.

Finally there are a few particular limitations with the bifurcation explainer, one is the usage of minimum value and the maximum value for the distribution. The algorithm behind the the bifurcation explainer, the categorial algorithm, needs these values to work, that if passed incorrectly will change the results provided, what this means is that though the resulting distribution might only vary between certain values, through the training phase the the values might vary in a higher spectrum thus limiting the final solution to a broader set of values than necessary. There is a new proposed algorithm by the same authors that aims to correct this limitation, by iteratively changing the limits, however, like the first algorithm it is done for a deep reinforcement learning problem.

The other limitation for the bifurcation explainer is that this algorithm uses a various number of passages in order to reach the optimal solution, which could lead to a huge number of iterations and time consumption if the size of the problem is to big.

8 Conclusion and Future Work

We have shown a toolkit in order to obtain explanations for reinforcement learning, that contains 6 main cores (state explainer, path explainer, optimal path informer, bifurcation informer, bifurcation explainer and security explainer). It has the ability to receive any kind of model-based reinforcement learning MDP, and by accepting a numerous types of queries for each core, it can generate explanations in the for of text and visually to the usage of distributions. The construction of the toolkit makes it easier for the user to interpret and question the various elements of an model-based MDP.

In the future not only we expect to increase the number of possible types of explanations that may appear in the future, we would also like to extend the support of these explanations to other types of reinforcement learning models. Further we would like to increase the speed of the overall toolkit by introducing mechanism such as parallelization of the various algorithms, specifically the various explainers.

REFERENCES

[1] Marc G. Bellemare, Will Dabney, and Rémi Munos, 'A distributional perspective on reinforcement learning', *CoRR*, **abs/1707.06887**, (2017).
[2] Richard Bellman, 'A markovian decision process', *Journal of Mathematics and Mechanics*, **6**(5), 679–684, (1957).
[3] Dimitri P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.
[4] Daniel Grollman and Odest Jenkins, 'Dogged learning for robots', pp. 2483 – 2488, (05 2007).
[5] Bradley Hayes and Julie A. Shah, 'Improving robot controller transparency through autonomous policy explanation', in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, pp. 303–312, New York, NY, USA, (2017). ACM.
[6] R. A. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, MA, 1960.
[7] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore, 'Reinforcement learning: A survey', *CoRR*, **cs.AI/9605103**, (1996).
[8] Brian Y. Lim and Anind K. Dey, 'Toolkit to support intelligibility in context-aware applications', in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp '10, pp. 13–22, New York, NY, USA, (2010). ACM.
[9] Zachary Chase Lipton, 'The mythos of model interpretability', *CoRR*, **abs/1606.03490**, (2016).
[10] Andrew William Moore, 'Efficient memory-based learning for robot control', Technical report, (1990).
[11] Martin L Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.
[12] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin, '"why should I trust you?": Explaining the predictions of any classifier', *CoRR*, **abs/1602.04938**, (2016).
[13] Andrea Thomaz and Cynthia Breazeal, 'Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance.', volume 1, (01 2006).
[14] Jasper van der Waa, Jurriaan van Diggelen, Karel van den Bosch, and Mark A. Neerincx, 'Contrastive explanations for reinforcement learning in terms of expected consequences', *CoRR*, **abs/1807.08706**, (2018).
[15] R. Williams and L. Baird, 'Tight performance bounds on greedy policies based on imperfect value functions', Technical Report NU-CCS-93-14, Northeastern University, (1993).