EXPLAIN YOUR MOVE: UNDERSTANDING AGENT ACTIONS USING FOCUSED FEATURE SALIENCY

Piyush Gupta∗, Nikaash Puri∗, Sukriti Verma∗& Balaji Krishnamurthy
Media and Data Science Research
Adobe Systems Inc.
Noida, Uttar Pradesh, India 201301
{piygupta, nikipuri, sukrverm, kbalaji}@adobe.com

Dhruv Kayastha†
Indian Institute of Technology Kharagpur
West Bengal, India 721302
{dhruvkayastha}@iitkgp.ac.in

Shripad Deshmukh†
Indian Institute of Technology Madras
Chennai, India 600036
{shripad}@smail.iitm.ac.in

Sameer Singh
Department of Computer Science
University of California, Irvine
California, USA
{sameer}@uci.edu

ABSTRACT

As deep reinforcement learning (RL) is applied to more tasks, there is a need to visualize and understand the behavior of learned agents. Saliency maps explain agent behavior by highlighting the features of the input state that are most relevant for the agent in taking an action. Existing perturbation-based approaches to compute saliency often highlight regions of the input that are not relevant to the action taken by the agent. Our approach generates more focused saliency maps by balancing two aspects (specificity and relevance) that capture different desiderata of saliency. The first captures the impact of perturbation on the relative expected reward of the action to be explained. The second downweights irrelevant features that alter the relative expected rewards of actions other than the action to be explained. We compare our approach with existing approaches on agents trained to play board games (Chess and Go) and Atari games (Breakout, Pong and Space Invaders). We show through illustrative examples (Chess, Atari, Go), human studies (Chess), and automated evaluation methods (Chess) that our approach generates saliency maps that are more interpretable for humans than existing approaches.

1 INTRODUCTION

Deep learning has achieved success in various domains such as image classification (He et al., 2016; Krizhevsky et al., 2012), machine translation (Mikolov et al., 2010), image captioning (Karpathy et al., 2015), and deep Reinforcement Learning (RL) (Mnih et al., 2015; Silver et al., 2017). To explain and interpret the predictions made by these complex, "black-box"-like systems, various gradient and perturbation techniques have been introduced for image classification (Simonyan et al., 2013;}

∗These authors contributed equally
†Work done during the Adobe MDSR Research Internship Program.
We are given an agent $M$, operating on a state space $S$, with the set of actions $A_s$ for $s \in S$, and a $Q$-value function denoted as $Q(s, a)$ for $s \in S$, $a \in A_s$. Following a greedy policy, let the action that was selected by the agent at state $s$ be $\hat{a}$, i.e. $\hat{a} = \arg \max_a Q(s, a)$. The states are parameterized in terms of state-features $F$. For instance, in a board game such as chess, the features are the 64 squares. For Atari games, the features are pixels. We are interested in identifying which features of the state $s$ are important for the agent in taking action $\hat{a}$. We assume that the agent is in the exploitation phase.

1 https://stockfishchess.org/
2 We follow the coordinate naming convention where columns are 'a-h' (left-right), rows '8-1' (top-bottom), and pieces are labeled using the first letter of its name in upper case (e.g. 'B' denotes the bishop). A move consists of the piece and the position it moves to, e.g. 'Bb6' indicates that the bishop moves to position 'b6'.

We follow the coordinate naming convention where columns are 'a-h' (left-right), rows '8-1' (top-bottom), and pieces are labeled using the first letter of its name in upper case (e.g. 'B' denotes the bishop). A move consists of the piece and the position it moves to, e.g. 'Bb6' indicates that the bishop moves to position 'b6'.
We will propose an alternative approach to compute \( S \). The general outline of perturbation based saliency approaches is as follows. For each feature \( f \), first perturb \( s \) to get \( s' \). For instance, in chess, we can perturb the board position by removing the piece in the \( f \)th square. In Atari, Greydanus et al. (2018) perturb the input image by adding a Gaussian blur centered on the \( f \)th pixel. Second, query \( M \) to get \( Q(s', a) \) \( \forall a \in A_s \cap A_{s'} \). We take the intersection of \( A_s \) and \( A_{s'} \) to represent the case where some actions may be legal in \( s \) but not in \( s' \) and vice versa. For instance, when we remove a piece in chess, actions that were legal earlier may not be legal anymore. In the rest of this section, when we use “all actions” we mean all actions that are legal in both the states \( s \) and \( s' \). Finally, compute \( S[f] \) based on how different \( Q(s, a) \) and \( Q(s', a) \) are, i.e. intuitively, \( S[f] \) should be high if \( Q(s, a) \) is significantly different from \( Q(s', a) \). Greydanus et al. (2018) compute the saliency map using \( S_1[f] = \frac{1}{2} \pi_s - \pi_{s'} \), and \( S_2[f] = \frac{1}{2} (V(s) - V(s'))^2 \). G"{u}rverbs"{u} et al. (2018) use \( S[f] = Q(s, \hat{a}) - Q(s', \hat{a}) \). In this work, we will propose an alternative approach to compute \( S[f] \).

**Properties** We define two desired properties of an accurate saliency map for policy-based agents:

1. **Specificity**: Saliency \( S[f] \) should focus on the effect of the perturbation specifically on the action being explained, \( \hat{a} \), i.e. it should be high if the \( f \)th feature of the state reduces the relative expected reward of the selected action. Stated another way, \( S[f] \) should be high if \( Q(s, \hat{a}) - Q(s', \hat{a}) \) is substantially larger than \( Q(s, a) - Q(s', a), a \neq \hat{a} \). For instance, in figure 1, removing pieces such as the white queen impact all actions uniformly \((Q(s, a)-Q(s', a)\) is roughly equal for all actions). Therefore, such pieces should not be salient for explaining \( \hat{a} \). On the other hand, removing pieces such as the white knight on a4 specifically impacts the move \((\hat{a} = Bb6)\) we are trying to explain \((Q(s, Bb6)-Q(s', Bb6) \gg Q(s, a) - Q(s', a) \) for other actions \( a \). Therefore, such pieces should be salient for \( \hat{a} \).

2. **Relevance**: Since the \( Q \)-values represent the expected returns, two states \( s \) and \( s' \) can have substantially different \( Q \)-values for all actions, i.e. may be higher for \( s' \) for all actions if \( s' \) is a better state. Saliency map for a specific action \( \hat{a} \) in \( s \) should thus ignore such differences, i.e. \( s' \) should contribute to the saliency only if its effects are relevant to \( \hat{a} \). In other words, \( S[f] \) should be low if perturbing the \( f \)th feature of the state alters the expected rewards of actions other than \( \hat{a} \). For instance, in figure 1, removing the black pawn on c6 increases the expected reward of other actions (in this case, Bb4). However, it does not affect the expected reward of the action to be explained (Bb6). Therefore, the pawn is not salient for explaining the move. In general, such features that are irrelevant to \( \hat{a} \) should not be salient.

Existing approaches to saliency maps do not capture these properties in how they compute the saliency. Both the saliency approaches used in Greydanus et al. (2018), i.e. \( S_1[f] = \frac{1}{2} (V(s) - V(s'))^2 \) and \( S_2[f] = \frac{1}{2} \pi_s - \pi_{s'} \), are not focusing on the action-specific effects since they aggregate the change over all actions. Although the saliency computation in G"{u}rverbs"{u} et al. (2018) is somewhat more specific to the action, i.e. \( S[f] = Q(s, \hat{a}) - Q(s', \hat{a}) \), it is ignoring whether the effects on \( Q \) are relevant only to \( \hat{a} \), or effect all the other actions as well. This is illustrated in Figure 1.

![Figure 1: Saliency maps generated by existing approaches](image)
Identifying Specific Changes To focus on the effect of the change on the action, we are interested in whether the relative returns of \( \hat{a} \) change with the perturbation. Using \( Q(s, \hat{a}) \) directly, as in Iyer et al. (2018), does not capture the relative changes to \( Q(s, a) \) for other actions. To support specificity, we use the softmax over Q-values to normalize the values (as is also used in softmax action selection):

\[
P(s, \hat{a}) = \frac{\exp(Q(s, \hat{a}))}{\sum_a \exp(Q(s, a))}
\]

and compute \( \Delta p = P(s, \hat{a}) - P(s', \hat{a}) \), the difference in the relative expected reward of the action to be explained between the original and the perturbed state. A high value of \( \Delta p \) thus implies that the feature is important for the specific choice of action \( \hat{a} \) by the agent, while a low value indicates that the effect is not specific to the action.

Identifying Relevant Changes Apart from focusing on the change in \( Q(s, \hat{a}) \), we also want to ensure that the perturbation leads to minimal effect on the relative expected returns for other actions. To capture this intuition, we will compute the relative returns of all other actions, and compute saliency in proportion to their similarity. Specifically, we normalize the Q-values using a softmax apart from the selected action \( \hat{a} \).

\[
P_{\text{rem}}(s, a) = \frac{\exp(Q(s, a))}{\sum_{a' \neq \hat{a}} \exp(Q(s, a'))} \quad \forall a \neq \hat{a}
\]

We use the KL-Divergence \( D_{KL} = P_{\text{rem}}(s', a)||P_{\text{rem}}(s, a) \) to measure the difference between \( P_{\text{rem}}(s', a) \) and \( P_{\text{rem}}(s, a) \). A high \( D_{KL} \) indicates that the relative expected reward of taking some actions (other than the original action) changes significantly between \( s \) and \( s' \). In other words, a high \( D_{KL} \) indicates that the effect of the feature is spread over other actions, i.e. the feature may not be relevant for the selected action \( \hat{a} \).

Computing the Saliency To compute salience \( S[f] \), we need to combine \( \Delta p \) and \( D_{KL} \). If \( D_{KL} \) is high, \( S[f] \) should be low, regardless of whether \( \Delta p \) is high; the perturbation is affecting many other actions. Conversely, when \( D_{KL} \) is low, \( S[f] \) should depend on \( \Delta p \). To be able to compare these properties on a similar scale, we define a normalized measure of distribution similarity \( K \) using \( D_{KL} \):

\[
K = \frac{1}{1 + D_{KL}}
\]

As \( D_{KL} \) goes from 0 to \( \infty \), \( K \) goes from 1 to 0. Thus, \( S[f] \) should be low if either \( \Delta p \) is low or \( K \) is low. Harmonic mean provides this desired effect in a robust, smooth manner, and therefore we define \( S[f] \) to be the harmonic mean of \( \Delta p \) and \( K \):

\[
S[f] = \frac{2K \Delta p}{K + \Delta p}
\]

Equation 4 captures our desired properties of saliency maps. If perturbing the \( f^{th} \) feature affects the expected rewards of all actions uniformly, then \( \Delta p \) is low and subsequently \( S[f] \) is low. This low value of \( \Delta p \) captures the property of specificity defined above. If perturbing the \( f^{th} \) feature of the state affects the rewards of some actions other than the action to be explained, then \( D_{KL} \) is high, \( K \) is low, and \( S[f] \) is low. This low value of \( K \) captures the property of relevance defined above.

3 RESULTS

To show that our approach produces more meaningful saliency maps than existing approaches, we use sample positions from Chess, Atari (Breakout, Pong and Space Invaders) and Go (Section 3.1). To show that our approach generates saliency maps that provide useful information to humans, we conduct human studies on problem-solving for chess puzzles (Section 3.2). To automatically compare the saliency maps generated by different perturbation-based approaches, we introduce a Chess saliency dataset (Section 3.3). We use the dataset to show how our approach is better than existing approaches in identifying chess pieces that humans deem relevant in several positions. In Section 3.4, we show how our approach can be used to understand common tactical ideas in chess by interpreting the action of a trained agent.
To show that our approach works for black-box agents, regardless of whether they are trained using reinforcement learning, we use a variety of agents. We only assume access to the agent’s $Q(s,a)$ function for all experiments. For experiments on chess, we use the Stockfish agent\(^3\). For experiments on Go, we use the pre-trained MiniGo RL agent\(^4\). For experiments on Atari agents and for generating saliency maps for Greydanus et al. (2018), we use their code and pre-trained RL agents\(^5\). For generating saliency maps using Iyer et al. (2018), we use our own implementation\(^6\). All of our code and more detailed results are available in our Github repository\(^6\).

3.1 Illustrative Examples

In this section, we provide examples of generated saliency maps to highlight the qualitative differences between our approach that is action-focused and existing approaches that are not.

**Chess** Figure 1 shows sample positions where our approach produces more meaningful saliency maps than existing approaches for a chess-playing agent (Stockfish). Greydanus et al. (2018) and Iyer et al. (2018) generate saliency maps that highlight pieces that are not relevant to the move played by the agent. This is because they use differences in $Q(s,a)$, $V(s)$ or the $L_2$ norm of the policy vector between the original and perturbed state to calculate the saliency maps. Therefore, pieces such as the white queen that affect the value estimate of the state are marked salient. In contrast, the saliency map generated by our approach only highlights pieces relevant to the move.

**Atari** To show that our approach generates saliency maps that are more focused than those generated by Greydanus et al. (2018), we compare the approaches on three Atari games: Breakout, Pong, and Space Invaders. Figures 2, 3, and 4 shows the results. Our approach highlights regions of the input image more precisely, while the Greydanus et al. (2018) approach highlights several regions of the input image that are not relevant to explain the action taken by the agent.

**Go** Figure 5 shows a board position in Go. It is black’s turn. The four white stones threaten the three black stones that are in one row at the top left corner of the board. To save those three black stones, black looks at the three white stones that are directly below the three black ones. Due to another white stone below the three white stones, the continuous row of three white stones cannot be captured easily. Therefore black moves to place a black stone below that single white stone in an attempt to start capturing the four white stones. It takes the next few turns to surround the structure of four white stones with black ones, thereby saving its pieces. The method described in Greydanus et al. (2018) generates a saliency map that highlights almost all the pieces on the board. Therefore, it reveals little about the pieces that the agent thinks are important. On the other hand, the map produced by Iyer et al. (2018) highlights only a few pieces. The saliency map generated by our approach correctly highlights the structure of four white stones and the black stones already present around them that may be involved in capturing them.

\(^3\)https://stockfishchess.org
\(^4\)https://github.com/tensorflow/minigo
\(^5\)https://github.com/greydanus/visualize_atari
\(^6\)https://github.com/rl-interpretation/understandingRL
Figure 3: Comparing saliency of RL agents trained to play Atari Pong

Figure 4: Comparing saliency of RL agents trained to play Space Invaders

Figure 5: Comparing saliency maps generated by different approaches for the MiniGo agent
Table 1: Results of Human Studies for solving chess puzzles

<table>
<thead>
<tr>
<th></th>
<th>No Saliency</th>
<th>Our Approach</th>
<th>Greydanus et al.</th>
<th>Iyer et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>56.67%</td>
<td><strong>72.41%</strong></td>
<td>37.48%</td>
<td>59.51%</td>
</tr>
<tr>
<td>Average time taken</td>
<td>77.53 sec</td>
<td><strong>52.95 sec</strong></td>
<td>60.23 sec</td>
<td>59.75 sec</td>
</tr>
</tbody>
</table>

Figure 6: ROC curves comparing approaches on the chess saliency dataset

3.2 HUMAN STUDIES: CHESS

To show that our approach generates saliency maps that provide useful information to humans, we conduct human studies on problem-solving for chess puzzles. We show fifteen chess players (ELO 1600-2000) ten chess puzzles from https://www.chess.com (average difficulty ELO 1800). For each puzzle, we show either the puzzle without a saliency map, or the puzzle with a saliency map generated by our approach, Greydanus et al. (2018), or Iyer et al. (2018). The player is then asked to solve the puzzle. We measure the accuracy (number of puzzles correctly solved) and the average time taken to solve the puzzle, shown in Table 1. The saliency maps generated by our approach are more helpful for humans when solving puzzles than those generated by other approaches. We observed that the saliency maps generated by Greydanus et al. (2018) often confuse humans, because they highlight several pieces unrelated to the tactic. The maps generated by Iyer et al. (2018) highlight few pieces and therefore are marginally better than showing no saliency maps for solving puzzles.

3.3 CHESS SALIENCY DATASET

To automatically compare the saliency maps generated by different perturbation-based approaches, we introduce a Chess saliency dataset. The dataset consists of 100 chess puzzles. Each puzzle has a single correct move. For each puzzle, we ask three human experts (ELO > 2200) to mark the pieces that are important for playing the correct move. We take a majority vote of the three experts to obtain a list of pieces that are important for the move played in the position. The complete dataset is available in our Github repository.

We use this dataset to compare our approach to existing approaches (Greydanus et al., 2018; Iyer et al., 2018). Each approach generates a list of squares and a score that indicates how salient the piece on the square is for a particular move. We scale the scores between 0 and 1 to generate ROC curves. Figure 6a shows the results. Our approach generates saliency maps that are better than existing approaches at identifying chess pieces that humans deem relevant in certain positions.

To evaluate the relative importance of the two components in our saliency computation ($S[f]$; Equation 4), we compute saliency maps and ROC curves using each component individually, i.e. $S[f] = \Delta p$ or $S[f] = K$, and compare harmonic mean to other ways to combine them, i.e. using the average, geometric mean, and minimum of $\Delta p$ and $K$. Figure 6b shows the results. Combination

https://github.com/rl-interpretation/understandingRL
Figure 7: Saliency maps generated by our approach that demonstrate common tactical motifs in chess

of the two properties via harmonic mean leads to more accurate saliency maps than alternative approaches.

3.4 EXPLAINING TACTICAL MOTIFS IN CHESS

In this section, we show how our approach can be used to understand common tactical ideas in chess by interpreting the action of a trained agent. Figure 7 illustrates common tactical positions in chess. The corresponding saliency maps are generated by interpreting the moves played by the Stockfish agent in these positions.

In Figure 7a, it is white to move. The surprising Rook x d6 is the move played by Stockfish. Figure 7d shows the saliency map generated by our approach. The map illustrates the key idea in the position. Once black’s rook recaptures white’s rook, white’s bishop pins it to the black king. Therefore, white can increase the number of attackers on the rook. The additional attacker is the pawn on e4 highlighted by the saliency map.

In Figure 7b, it is white to move. Stockfish plays Queen x h7. A queen sacrifice! Figure 7e shows the saliency map. The map highlights the white rook and bishop, along with the queen. The key idea is that once black captures the queen with his king (a forced move), then the white rook moves to h5 with checkmate. This checkmate is possible because the white bishop guards the important escape square on g6. The saliency map highlights both pieces.

In Figure 7c, it is black to move. Stockfish plays the sacrifice rook x d4. The saliency map in Figure 7f illustrates several key aspects of the position. The black queen and light-colored bishop are threatening mate on g2. The white queen protects g2. The white rook on a5 is unguarded. Therefore, once white recaptures the sacrificed rook with the pawn on c3, black can attack both the white rook and queen with the move bishop to b4. The idea is that the white queen is “overworked” or “overloaded” on d2, having to guard both the g2-pawn and the a5-Rook against black’s double attack.

3.5 ROBUSTNESS TO PERTURBATIONS

We are also interested in evaluating the robustness of the generated saliency maps: is the saliency different if non-salient changes are made to the state? To evaluate the robustness of our approach, we perform two irrelevant perturbations to the positions in the chess saliency dataset. First, we
pick a random piece amongst the ones labeled non-salient by human experts in a particular position, and remove it from the board. We repeat this for each puzzle in the dataset to generate a new perturbed saliency dataset. Second, we remove a random piece amongst ones labeled non-salient by our approach for each puzzle, creating another perturbed saliency dataset. In order to evaluate the effect of non-salient perturbations on our generated saliency maps, we compute the AUC values for the generated saliency maps, as above, for these perturbed datasets. Since we remove non-salient pieces, we expect the saliency maps and subsequently AUC value to be similar to the value on the original dataset. For both these perturbations, we get an AUC value of 0.92, same as the value on the non-perturbed dataset, confirming the robustness of our saliency maps to these non-relevant perturbations.

4 RELATED WORK

Since understanding RL agents is important both for deploying RL agents to the real world and for gaining insights about the tasks, a number of different kinds of interpretations have been introduced.

A number of approaches generate natural language explanations to explain RL agents (Dodson et al., 2011; Elizalde et al., 2008; Khan et al., 2009). They assume access to an exact MDP model and that the policies map from interpretable, high-level state features to actions. More recently, Hayes & Shah (2017) analyze execution traces of an agent to extract explanations. A shortcoming of this approach is that it explains policies in terms of hand-crafted state representations that are semantically meaningful to humans. This is often not practical for board games or Atari games where the agents learn from raw board/visual input. Zahavy et al. (2016) apply t-SNE (Maaten & Hinton, 2008) on the last layer of a deep Q-network (DQN) to cluster states of behavior of the agent. They use Semi-Aggregated Markov Decision Processes (SAMDPs) to approximate the black box RL policies. They use the more interpretable SAMDPs to gain insight into the agent’s policy. An issue with the explanations is that they emphasize t-SNE clusters that are difficult to understand for non-experts. To build user trust and increase adoption, it is important that the insight into agent behavior should be in a form that is interpretable to the untrained eye and obtained from the original policy instead of a distilled one.

Most relevant to our approach are the visual interpretable explanations of deep networks using saliency maps. Methods for computing saliency can be classified broadly into two categories.

Gradient-based methods identify input features that are most salient to the trained DNN by using the gradient to estimate their influence on the output. Simonyan et al. (2013) use gradient magnitude heatmaps, which was expanded upon by more sophisticated methods to address their shortcoming, such as guided backpropagation (Springenberg et al., 2014), excitation backpropagation (Zhang et al., 2018), DeepLIFT (Shrikumar et al., 2017), GradCAM (Selvaraju et al., 2017), and GradCAM++ (Chattopadhay et al., 2018). Integrate gradients (Sundararajan et al., 2017) provide two axioms to further define the shortcomings of these approaches: sensitivity (relative to a baseline) and implementation invariance, and use them to derive an approach. Nonetheless, all gradient-based approaches still depend on the shape in the immediate neighborhood of a few points, and conceptually, use perturbations that lack physical meaning, making them difficult to use and vulnerable to adversarial attacks in form of imperceivable noise (Ghorbani et al., 2019). Further, they are not applicable to scenarios with black-box access to the agent, and even with white-box access to model internals, they are not applicable when agents are not fully differentiable, such as Stockfish for chess.

We are more interested in perturbation-based methods for black-box agents: methods that compute the importance of an input feature by removing, altering, or masking the feature in a domain-aware manner and observing the change in output. It is important to choose a perturbation that removes information without introducing any new information. As a simple example, Fong & Vedaldi (2017) consider a classifier that predicts ‘True’ if a certain input image contains a bird and ‘False’ otherwise. Removing information from the part of the image which contains the bird should change the classifier’s prediction, whereas removing information from other areas should not. Several kinds of perturbations have been explored, e.g. Zeiler & Fergus (2014); Ribeiro et al. (2016) remove information by replacing a part of the input with a gray square. Note that these approaches are implementation invariant by definition, and are sensitive with respect to the perturbation function.

Existing perturbation-based approaches for RL (Greydanus et al., 2018; Iyer et al., 2018), however, by focusing on the complete $Q$ (or $V$), tend to produce saliency maps that are not specific to the
action of interest. Our proposed approach addresses this by measuring the impact only on the action being selected, resulting in more focused and useful saliency maps, as we show in our experiments.

5 LIMITATIONS AND FUTURE WORK

Saliency maps focus on visualizing the dependence between the input and output to the model, essentially identifying the situation-specific explanation for the decision. Although such local explanations have applications in understanding, debugging, and developing trust with machine learning systems, they do not provide any direct insights regarding the general behavior of the model, or guarantee that the explanation is applicable to a different scenario. Attempts to provide a more general understanding of the model include carefully selecting prototype explanations to show to the user (van der Linden et al., 2019) and crafting explanations that are precise and actionable (Ribeiro et al., 2018). We will explore such ideas for the RL setting in future, to provide explanations that accurately characterize the behavior of the policy function, in a precise, testable, and intuitive manner.

There are a number of limitations of our approach to generating saliency maps in our current implementation. First, we perturb the state by removing information (removing pieces in Chess/Go, blurring pixels in Atari). Therefore, our approach cannot highlight the importance of absence of certain attributes, i.e. saliency of certain positions being empty. In games such as Chess and Go, an empty square or file (collection of empty squares) can often be important for a particular move. Future work will explore perturbation functions that add information to the state (perhaps in the form of adding pieces in Chess/Go). Such functions, along with our approach, can be used to calculate the importance of empty squares. Second, it is possible that perturbations may explore states that lie outside the manifold, i.e. they lead to invalid states. For example, unless explicitly prohibited like we do, our approach will compute the saliency of the king pieces by removing them, which is not allowed in the game, or remove the paddle from Pong. In future, we will explore strategies that take the valid state space into account when computing the saliency. Last we are estimating the saliency of each feature independently, ignoring feature dependencies and correlations, which may lead to incorrect saliency maps. We will investigate approaches that perturb multiple features to quantify the importance of each feature (Ribeiro et al., 2016; Lundberg & Lee, 2017), and combine them with our approach to explaining black-box policy-based agents.

6 CONCLUSION

We presented a perturbation-based approach that generates more focused saliency maps than existing approaches by balancing two aspects (specificity and relevance) that capture different desired characteristics of saliency. We showed through illustrative examples (Chess, Atari, Go), human studies (Chess), and automated evaluation methods (Chess) that our approach generates saliency maps that are more interpretable for humans than existing approaches. The results of our technique show that saliency can provide meaningful insights into a black-box RL agent’s behavior.

ACKNOWLEDGEMENTS

sameer: need to add

REFERENCES


A EXPERIMENTAL DETAILS

For experiments on chess, we use the latest version of the Stockfish agent: https://stockfishchess.org/. Stockfish works using a heuristic-based measure for each state along with Alpha-Beta Pruning to search over the state-space.

For experiments on Go, we use the pre-trained MiniGo RL agent: https://github.com/tensorflow/minigo. This agent was trained using the AlphaGo Algorithm (Silver et al., 2016). It also adds features and architecture changes from the AlphaZero Algorithm Silver et al. (2017).

For experiments on Atari agents and for generating saliency maps for Greydanus et al. (2018), we use their code and pre-trained RL agents available at https://github.com/greydanus/visualize_atari. These agents are trained using the Asynchronous Advantage Actor-Critic Algorithm (A3C) (Mnih et al., 2016).

For generating saliency maps using Iyer et al. (2018), we use our implementation. All of our code and more detailed results are available in our Github repository: https://github.com/rl-interpretation/understandingRL.

For chess and Go, we perturb the board position by removing one piece at a time. We do not remove a piece if the resulting position is illegal. For instance, in chess, we do not remove the king. For Atari, we use the perturbation technique described in Greydanus et al. (2018). The technique perturbs the input image by adding a Gaussian blur localized around a pixel. The blur is constructed using the Hadamard product to interpolate between the original input image and a Gaussian blur. The saliency maps for Atari agents have been computed on the frames provided by Greydanus et al. (2018) in their code repository.

The puzzles for conducting the Chess human studies, creating the Chess Saliency Dataset, and providing illustrative examples have been taken from Lichess: https://database.lichess.org/. The puzzles for illustrative examples on Go have been taken from OnlineGo: https://online-go.com/puzzles.

B SALIENCY MAPS FOR TOP 3 MOVES

Figure 8 shows the saliency maps generated by our approach for the top 3 moves in a chess position. The maps highlight the different pieces that are salient for each move. For instance, Figure 8a shows that for the move Qd4, the pawn on g7 is important. This is because the queen move protects the pawn. For the saliency maps in Figures 8b and 8c, the pawn on g7 is not highlighted.

C SALIENCY MAPS FOR LEELAZERO

To show that our approach generates meaningful saliency maps in Chess for RL agents, we interpret the LeelaZero Deep RL agent https://github.com/leela-zero/leela-zero. Figure 9 shows the results. As discussed in Section 1, the saliency maps generated by (Greydanus et al., 2018;
Figure 9: Saliency maps generated by different approaches for the LeelaZero Deep Reinforcement Learning Agent

Iyer et al., 2018) highlight several pieces that are not relevant to the move being explained. On the other hand, the saliency maps generated by our approach highlight the pieces relevant to the move.