Towards Explainable Reinforcement Learning Using Scoring Mechanism Augmented Agents

Yang Liu, Xinzhi Wang(✉), Yudong Chang, and Chao Jiang

School of Computer Engineering and Science, Shanghai University, Shanghai, China
{lliuyang,wxz2017,cydshu,superjiang}@shu.edu.cn

Abstract. Deep reinforcement learning (DRL) is increasingly used in application areas such as medicine and finance. However, the direct mapping from state to action in DRL makes it challenging to explain why decisions are made. Existing algorithms for explaining DRL policy are posteriori, explaining to an agent after it has been trained. As a common limitation, these posteriori methods fail to improve training with the deduced knowledge. Face with that, an end-to-end trainable explanation method is proposed, in which an Adaptive Region Scoring Mechanism (ARS) is embedded into DRL system. The ARS explains the agent's action by evaluating the features of the input state that are most relevant action before DRL re-learn from task-related regions. The proposed method is validated on Atari games. Experiments demonstrate that agent using the explainable proposed mechanism outperforms the original models.

Keywords: Deep reinforcement learning · Explainable AI · Adaptive region scoring mechanism

1 Introduction

In recent years, deep reinforcement learning (DRL) has achieved unprecedented success in many practical applications [5]. DRL models train agents that process continuous input information from the environment to learn and implement a policy that maximizes the expected returns. This structure has proven to be very effective. Unfortunately, both deep learning and reinforcement learning are poorly explainable. It is not easy to understand how decisions are made, what information is used, and why mistakes are made, all of which are necessary for many real-world application fields, such as finance, medical care, and robotics. This motivates the design of explainable DRL agents and modifying existing architectures for easier explanation.

Conventional DRL algorithms have low explainability and process state features uniformly at the beginning of training, which prevents the agent from focusing on valuable features quickly. Researchers have noticed that the human

G. Memmi et al. (Eds.): KSEM 2022, LNAI 13369, pp. 547–558, 2022.
https://doi.org/10.1007/978-3-031-10986-7_44

visual system cannot perceive and process all visual information presented at once. Instead, we selectively focus on different parts of the visual input to collect relevant information sequentially and attempt to combine each step of information over time to construct an abstract representation of the entire input [11]. The emergence of a saliency map [8,10] proves that the deep learning model will focus particular attention on a specific area when making decisions rather than treating all input information equally. Part of the interpretive approach shifts from trying to explain models that have been trained toward building models that are self-explanatory [10]. First, self-explanatory methods can be embedded in DRL models to generate saliency maps that relate to decisions generated during the decision-making process without additional supervision. Second, since the explanation is generated within the model, the agent can optimize the DRL agent according to the explanation information it has generated during the training process.

When faced with large-scale or high-dimensional state-space tasks, DRL agents can use a convolutional neural network (CNN) to extract the features of the image state and then train the model on the extracted information through reinforcement learning. However, the knowledge acquired by the agent consists of all the features of the whole original image. The model cannot focus on valuable feature information, and some critical information is lost in the forward propagation process [9]. The same weight is applied to calculating each feature image, but some features play a vital role in the image description, and the traditional CNN algorithm cannot fully use key features at the beginning of training. This paper proposes an evaluation mechanism to explain an agent's decision and help the agent focus on features with high policy value. This mechanism enables the agent to learn the optimal policy accurately and quickly.

Inspired by the popularity of saliency map, a method called *Adaptive Region Scoring* (ARS) is proposed to improve both the explainability and performance of the DRL agent. The ARS method changes the architecture of the feature extractor by incorporating a scoring module between each convolution layer. The ARS generates the score maps by evaluating the task-related regions in the state. The score maps can then be integrated to reveal how an agent makes decisions. In addition, ARS merges the score maps with the original state representations to optimize the training process.

The main contributions of this paper are highlighted as follows:

- A method is proposed to explain the actual rationale used in inference for decision-making by generating score maps. We can understand how the agent solves the task by analyzing the resulting score maps.
- The score maps are incorporated with the original unweighted representation of states, forcing the agent to focus on task-related information, which can effectively improve policy learning of the agent system.
- Experiments are conducted on the Atari platform. The experimental results show that the ARS module can effectively improve policy learning. In contrast, previous posterior explanation approaches do not improve performance.

2 Preliminaries

The standard DRL setting, where an agent learns to solve a sequential decision problem, is modeled as a Markov decision process (MDP), which can be denoted as $(\mathcal{S}, \mathcal{A}, \mathcal{P}(s, s^{'}), r_a(s, s^{'}), \gamma)$. Here, $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is the unknown state-transition probability function, $r_a(s, s^{'})$ is the immediate reward associated with taking action $a \in \mathcal{A}$ while transitioning from state $s \in \mathcal{S}$ to $s^{'} \in \mathcal{S}$, and $\gamma \in [0, 1]$ is the discount factor that represents a tradeoff between maximizing immediate returns versus future returns. The goal of the agent is to identify a policy π to maximize its expected reward, where the cumulative return at each time step is:

$$R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau \tag{1}$$

The purpose of policy gradient algorithms is to maximize the cumulative expected rewards $L = \mathbb{E}[\sum_t r(s_t, a_t)]$. The most commonly used gradient of objective function L with baseline can be written in the following form:

$$\nabla_\theta L = \int_S \mu(s) \int_A \nabla_\theta \pi_\theta(a|s) A(s, a) \tag{2}$$

where $\mathcal{S}$ denotes the set of all states and $\mathcal{A}$ denotes the set of all actions. $\mu(s)$ is an on-policy distribution over states. π is a stochastic policy that maps state $s \in \mathcal{S}$ to action $a \in \mathcal{A}$, and A is an advantage function.

Gradient-based actor-critic methods split the agent into two components: an actor that interacts with the environment using policy $\pi_\theta(a|s)$ and a critic that assigns values to these actions using value function $V_\theta(s)$. Both the policy and the value function are directly parameterized by θ. The policy and value function is updated through gradient descent:

$$\theta_{t+1} = \theta_t + \nabla_{\theta_t}(a_t|s_t) A_t(a_t|s_t) \tag{3}$$

The advantage function represents how good a state-action pair is compared with the average value of the current state, $A(a|s) = Q(a|s) - V(s)$. The most commonly used technique for computing the advantage function is generalized advantage estimation (GAE) [6]. One very common style that can be easily applied to A2C or any policy-gradient-like algorithm is:

$$A_t(a_t|s_t) = r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{t+1} + \gamma r^{T-t} V_{s_T} - V(s_t) \tag{4}$$

where T denotes the maximum length of a trajectory but not the terminal time step of a complete task, and γ is a discounted factor. If the episode terminates, we only need to set $V(s_t)$ to zero, without bootstrapping, which becomes $A_t = R_t - V(s_t)$. To ensure exploration early, the entropy regularization term H is introduced into the policy gradient:

$$\theta_{t+1} = \theta_t + \nabla_{\theta_t} log \pi_{\theta_t}(a_t|s_t) A_t + \beta \nabla_{\theta_t} H(\pi_{\theta_t}(s_t)) \tag{5}$$

where β is a hyperparameter that discounts the entropy regularization.

3 Proposed Method

This section proposes and provides a comprehensive description of our novel ARS algorithm. Traditional DRL agents rely on convolution and fully connected components to process input information step by step. This structure does not help people understand how the agents make decisions, what information they use, and why they make mistakes, nor can it enable agents to make a timely modification of their decisions based on the explanation information. First, the ARS evaluates the action of agents and output explanation that humans understand easily. Second, the agents utilize the evaluation results to modify their actions to focus on task-related information.

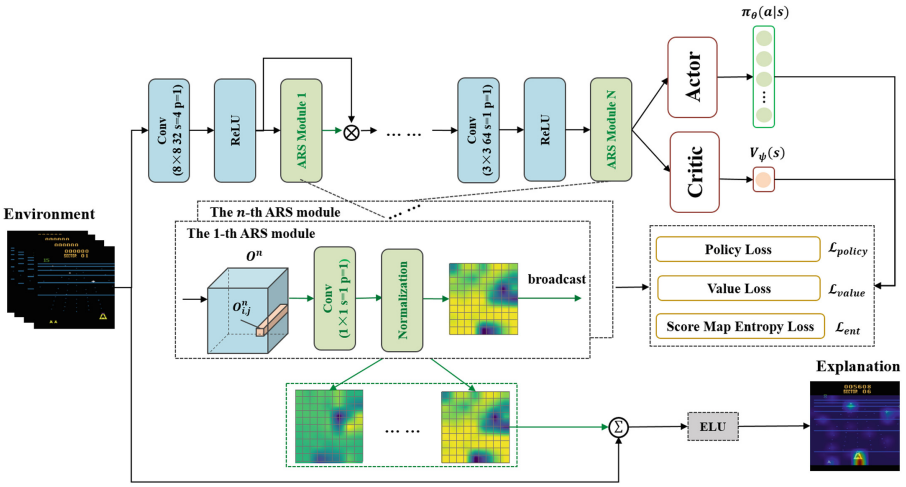


Fig. 1. Overview of the proposed method.

3.1 Model Overview

The complete architecture is illustrated in Fig. 1. Our image encoder Ψ_θ is a three-layer CNN interleaved with ReLU activation functions. The inputs to the channel are the images where the preprocessing procedure follows [1] and hence consists of a sequence of 4 frames stacked together, where each frame is 84×84 in grayscale. We use 4 frames because a single image state in an Atari game is non-Markovian. For example, the direction of an object's movement is ambiguous if we only see a single frame. As a built-in module, ARS calculates importance scores for state sub-regions based on the results of each convolutional layer. The results can be incorporated to generate saliency maps. Each score map is combined with the original state representation, forcing the agent to focus on task-relevant sub-regions during strategy learning.

The policy network π_θ and value network V_θ are encoded by multi-layer perceptron (MLP) with parameter θ. The policy network is a 3-layer MLP with a size of 64 for both hidden layers and a ReLU activation function. The output layer of the policy network has 16 units, producing the mean and the standard deviation for each action dimension. Before executing an action in the environment, a Tanh activation function is applied to enforce action bounds in the range of $[-1, 1]$. The value network is represented as a 2-layer MLP that outputs a scalar value specifying the corresponding value of a state. The value network uses a hidden layer size of 128 units with a ReLU activation function.

3.2 Adaptive Region Scoring Mechanism

At each time step t, an observation $s_t \in \mathbb{R}^{H\times W\times C}$ (here a sequence frames stacked together of height H and width W) is passed through the image encoder Ψ_θ with parameters θ. The visual frame s_t is fed into the model and aims to predict the action a ($a \in \mathcal{A}$) taken by the agent. Our motivation is that the agent should focus on the most relevant part of the observation, which is controllable by the agent, to be able to classify the actions.

We determine whether each region in a $H \times W$ grid is useful for predicting the agent's action. The feature map $O^n = \psi^n_\theta(O^{n-1}) \in \mathbb{R}^{h^n\times w^n\times c^n}$ is computed based on the observation s_t ($O^0 = s_t$), where ψ^n_θ is the n-th convolutional layer of Ψ_θ, $n \in N$ represents the serial number of the ARS module, $O^n \in \mathbb{R}^{h^n\times w^n\times c^n}$ is the n-th feature map, c^n denotes the size of the channel dimension and h^n, w^n denotes the height and width dimensions. We estimate a set of feature vectors, denoted $O^n_{i,j} \in \mathbb{R}^{c^n}$, for action classification from each grid cell (i, j) of the convolutional feature map. The feature map are converted to $h^n \times w^n$ vectors in which each vector has c^n dimension as follows:

$$O^n = [O^n_{1,1}, O^n_{1,2}, ..., O^n_{i,j}, ..., O^n_{h^n,w^n}], O^n_{i,j} \in \mathbb{R}^{c^n} \tag{6}$$

where $O^n_{i,j}$ corresponds to the features extracted by ψ^n_θ at different image regions. Then, tensor O^n is fed to the n-th ARS model to compute a score map that describes the importance of the state feature vector at the corresponding location. In other words, the input frame is divided into $h^n \times w^n$ regions, and the ARS mechanism attempts to score their relevance.

The ARS module takes the n-th convolution features O^n as input to derive the score maps $m^n \in \mathbb{R}^{h^n\times w^n}$ as:

$$\begin{aligned} m^n &= Q^n_\rho(O^n|W^n_\rho, B^n_\rho) \\ &= [m^n_{1,1}, m^n_{1,2}, ..., m^n_{i,j}, ..., m^n_{h^n,w^n}], m^n_{i,j} \in \mathbb{R} \end{aligned} \tag{7}$$

where W^n_ρ and B^n_ρ are parameters for the calculation, and function Q^n_ρ includes one convolution calculation with an 1×1 kernel whose stride is set to be 1. Each element in m^n corresponds to a spatial position on O^n and describes the importance of the image feature vector at that position. The score maps m^n are then passed to the normalization layer to generate a meaningful probability distribution over $h^n \times w^n$ regions. In the experiment, the softmax function is used

to implement the normalization layer. Let p^n be denoted as the final probability distributions after normalizing m^n, which can be considered as the amount of the importance of the corresponding vector $m^n_{i,j}$ among of $h^n \times w^n$ vectors in the input image:

$$\begin{aligned} p^n &= \frac{exp(m^n_{i,j})}{\sum_{i',j'} exp(m^n_{i',j'})}, \\ &= [p^n_{1,1}, p^n_{1,2}, ..., p^n_{i,j}, ..., p^n_{h^n,w^n}], p^n_{i,j} \in [0,1] \end{aligned} \tag{8}$$

where $p^n \in \mathbb{R}^{h^n \times w^n}$ is the counterpart of the learned probability distributions. The ARS module attempts to identify which regions of $O^n_{i,j}$ are important for the agent in taking action. The degree of correlation between regions and decisions is described by an probability of importance score $p^n_{i,j}$, where $p^n_{i,j} \in [0,1]$ denotes the correlation of regions (i,j) of $O^n_{i,j}$ for the agent taking action. A higher value indicates that regions (i,j) of O^n is more important for the agent when taking action.

The score map p^n is used to visualize the decision-making basis of the agent. The probability p^n should be high only on regions (i,j) that are predictive of the agent's actions. The resulting score map p^n can be bilinearly extrapolated to the size of the input state s_t to obtain $\hat{p}^n$, which can then be overlaid on top of the state to produce a high-quality heatmap that indicates regions that motivate the agent to take action. Let s^a_{ARS} be denoted as the final heatmap that indicates relevant regions that indicate regions that motivate the agent to take action a, $\hat{p}^n$ be the converted score map for state s_t in module n. The state is then weighted by the score map and passes through an Exponential Linear Unit (ELU) activation function to produce a high-quality heatmap, which can be denoted as:

$$s^a_{ARS} = \frac{1}{N} ELU(\sum_{n=1}^{N} \hat{p}^n \cdot s_t) \tag{9}$$

where $\hat{p}^n$ has values in the range $[0,1]$ with higher weights corresponding to a stronger response to the input state. The ELU function has been chosen in favour of the ReLU due to the dying ReLU effect. A visual representation of this process is depicted in Fig. 3.

The score map p^n can evaluate which positions in the current state representation are important. The representation $O^n_{i,j}$ are linearly combined using the score probabilities $p^n_{i,j}$, each score map p^n is broadcast along the channel dimension of tensor O^n and point-wise multiplied to produce the next result tensor $\hat{O}^n \in \mathbb{R}^{h^n \times w^n \times c^n}$, which can be described as:

$$\hat{O}^n = p^n \cdot O^n \tag{10}$$

The agent can learn to emphasize the high-scoring part of the input frame based on the given state. Note that the ARS model is fully differentiable, which

allows training the system in an end-to-end manner. After the score map linearly weights the feature map vector set, the new vector set replaces the original vector set. At this time, the agent will focus on the information of value from the feature map to strengthen the influence of important features on the subsequent training. This operation forces the agent system to locate the object of interest, and then the final convolution features O^N for consecutive frames are fed into different multi-layer perceptron (MLP) to derive $V_\theta(s_t)$ and $\pi_\theta(a_t|s_t)$:

$$V_\theta = MLP_{value}(O^N) \in \mathbb{R} \tag{11}$$

$$\pi_\theta = softmax(MLP_{policy}(O^N) \in \mathbb{R}^{|\mathcal{A}|}. \tag{12}$$

3.3 Training

The model is optimized with the standard cross-entropy loss $\mathcal{L}_{action} = \mathbb{E}[\mathcal{L}_{policy} + \mathcal{L}_{value}]$ with respect to the ground-truth action $a^* \in \mathcal{A}$ that the agent actually has taken. The cost function $\mathcal{L}_{action}$ is based on:

$$\mathcal{L}_{policy} = -log\pi_\theta(a_t|s_t)(R_t^n - V_\theta(s_t)) - \alpha\mathcal{H}_t(\pi_\theta) \tag{13}$$

$$\mathcal{L}_{value} = \frac{1}{2}(V_\theta(s_t) - R_t^n)^2 \tag{14}$$

$$\mathcal{H}_t(\pi_\theta) = -\sum_a \pi_\theta(a|s_t)log\pi_\theta(a|s_t) \tag{15}$$

where $R_t = \sum_{-=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V_\theta(s_{t+n})$ is the n-step bootstrapped return and α is a weight for the standard entropy regularization loss term $\mathcal{H}(\pi_\theta)$. According to this formulation, the score map P^n should be high only on regions (i, j) that are predictive of the agent's actions. Our formulation enables learning to localize related regions in a self-supervised manner without any additional supervisory signal.

Here, we adopt a few additional objective functions. We encourage the score map to attain a high entropy by including a score entropy regularization loss, $\mathcal{L}_{ent} = -\sum_n^N \mathcal{H}(P^n)$. This term penalizes overconfident score maps, making the scores closer to uniform whenever actions cannot be predicted, and allows the model to learn from unseen observations even when the score fails to perform well at first. The entire training objective becomes:

$$\mathcal{L}_{all} = \mathcal{L}_{action} + \lambda_{ent}\mathcal{L}_{ent} \tag{16}$$

where λ_{ent} is a mixing hyperparameter.

4 Experiments and Results

In this section, we conduct experiments to evaluate the explanations and performance of the proposed method. We trained the agent system with an A2C algorithm on the Atari platform. The following subsections will cover details about the environment and results.

Table 1. Hyperparameters of the experiments.

Hyperparameter	Value	Hyperparameter	Value
Activation function	ReLU	Number of parallel environments	16
Roll-out Steps	5	Max grad norm	0.5
Optimizer	RMSprop	Value loss coefficient	0.5
Entropy coefficient	0.01	GAE coefficient	0.95
Seeds	[10,100]	Total frames	20000000

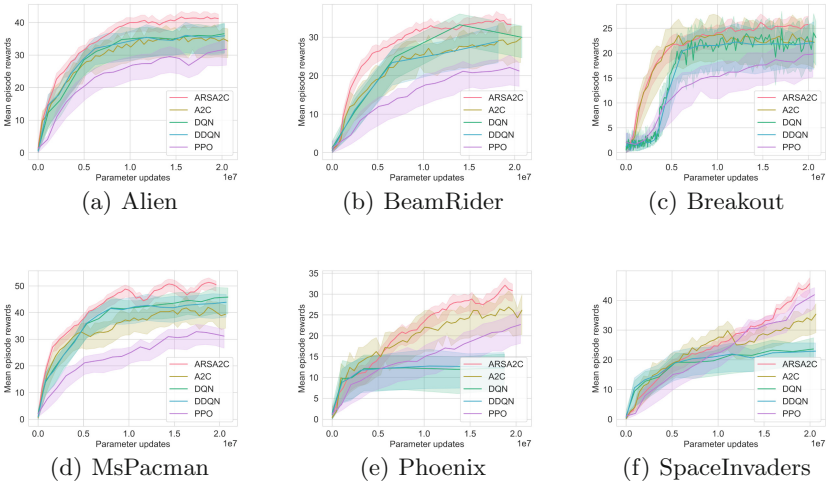
4.1 Settings

The proposed algorithm was tested on environments provided by the OpenAI Gym library [12], specifically their NoFrameskip-V4 versions, which are very challenging for reinforcement learning and provide a wide range of interesting games that are useful as a standard test for evaluating the proposed algorithms. The agent uses the game coding interface provided by the Gym platform to obtain the dynamic pictures and scores to self-learn the game. To evaluate the performance and verify that our algorithm can easily explain the agents, we conducted experiments on 6 Atari games: Phoenix, Alien, Breakout, Seaquest, Beamrider, Frostbite, MsPacman, and SpaceInvaders. All baseline agents were trained using the publicly available code for A2C with the same hyperparameters and model details. Details on the hyperparameter settings are shown in Table 1.

To extract convolutional features, we used three stacks of convolutions plus ReLU activation layers with filters 32 8×8, 64 4×4, and 64 3×3 and strides 4, 2, and 1. For the preprocessing step, we followed[4], in which each frame is converted from RGB format into single-channel grayscale and downsampled from a resolution of 210×160 to 84×84 via bilinear interpolation. In this way, to prevent causing loss of information, the computational resources and duration of the network training were reduced. At each time step, four consecutive preprocessed frames were stacked along the channel dimension as input. During the training, rewards were clipped in the range of $[-1, 1]$. To ensure stable learning, gradients were clipped to a value of 0.5, the discount factor was set to $\gamma = 0.99$, and the networks were trained for 2 million frames. All network weights were updated by the RMSProp optimizer with a decay factor of 0.95 and momentum of 0.1. The learning rate was 0.0002. Advantage actor-critic used entropy regularization with weight 0.01. Training and testing for all the games were performed with the same network architecture and hyperparameters.

4.2 Performance

For each environment, the agent was trained with different random seeds. We recorded a smoothed curve for the agent's episode rewards during training. The reward curves in Fig. 2 show the mean score at every timestep. Experience was collected in 16 threads that were executed 5 steps at a time under default hyperparameters, for a total of 80 environment frames between agent updates. We reported the average score across 100 test episodes for the final performance evaluation.



(a) Alien (b) BeamRider (c) Breakout

(d) MsPacman (e) Phoenix (f) SpaceInvaders

Fig. 2. Reward curves during training. Cumulative reward comparison on 6 Atari tasks.

Figure 2 shows the reward curves obtained during training. From these plots, it can be seen that our ARS-A2C architecture, which is the baseline model with an extra scoring module, performs well, while the A2C architecture achieves worse rewards. Experimental results show that the ARS scoring module improves the performance of the A2C algorithm. As the figure shows, the ARS-A2C algorithm has a better learning effect than the A2C algorithm at the early stage of training. The ARS-A2C agent can quickly obtain high scores, and its learning performance tends to be stable and improves in later training. There is little difference between the ARS-A2C algorithm and the A2C algorithm in the early stage of training. However, as the training stage increases, the average reward value of the ARS-A2C algorithm gradually exceeds that of the A2C algorithm, which proves that the ARS-A2C algorithm still has better learning performance than the A2C algorithm. We conclude that the ARS module helps the agent focus on the important regions of the input image by scoring the regions, which avoids incurring a higher calculation cost to process parts with low policy value. Therefore, in the subsequent training process, there will be no significant variance. The change in the shaded area of the curve shows that ARS-A2C is better than A2C in learning performance and obtains a more minor variance when converging, which alleviates the volatility problem of the A2C algorithm to a certain extent.

For comparisons, we use DQN [5], DDQN [3], A2C [4] (both with separate and shared actor-critic networks), and PPO [7] algorithms. Table 2 shows that ARS-A2C achieves better performances than other algorithms for all 6 games. Compared with the best score of other algorithms, ARS-A2C achieves 7.0% point improvement on Alien, 9.5% point improvement on Beamrider, 4.2% point improvement on Breakout, 32.5% point improvement on MsPacman, and 3.5% point improvement on Phoenix, 9.5% point improvement on SpaceInvaders.

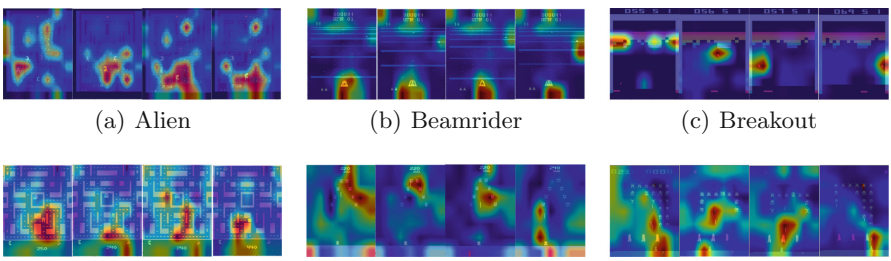
Table 2. Performance comparison on 6 Atari tasks.

Environment	Random	DQN	DDQN	PPO	A2C	ARS-A2C
Alien	240	2391	2041	1970	2091	**2560**
BeamRider	264	3627	3172	2750	3164	**3975**
Breakout	3	518	520	417	435	**542**
MsPacman	150	3180	2960	2350	2880	**4215**
Phoenix	440	10840	12250	20840	22530	**23340**
SpaceInvaders	120	3929	3672	4855	4673	**5320**

The ARS explains the agent through the scoring mechanism and stabilizes the training process to some extent. The learning performance of the agent with ARS is better than the other algorithms, which proves that ARS can help the agent process the game state information more accurately so that the agent can make the optimal decision more quickly and efficiently.

4.3 Explanations

Although score maps may not explain the entire decision-making process, they reveal some of the strategies used by the agent. To visualize the score map, we displayed the original input frame and overlaid the score map, producing bright areas indicating high scores and darker areas indicating low scores. Furthermore, the range of score values indicated that as the training process proceeded, the weight of the score graph for areas unrelated to the reward goal became very close to zero, meaning that little information was "mixed" in these areas during the summation process in Formula 10. The general pattern we observed is that agents learned to focus on task-related regions in the scene. This usually means that the agent received higher value rewards from the relevant region, which is essential in calculating the value function.



(a) Alien (b) Beamrider (c) Breakout

(d) MsPacman (e) Phoenix (f) Spaceinvaders

Fig. 3. Game explanations. Areas related to the agent for decision making are shown by adding the information from the scores map to the image, red regions corresponds to high score while blue corresponds to low. (Color figure online)

The score maps reflect the agent's degree of attention to each area when making decisions, including strengthening the tracking of targets and focusing on multiple targets, which can help the agent better understand the spatial information from the state input. Agents that have received ARS feedback also show a higher level of ability in a multi-target environment. As shown in Fig. 3(f) and Fig. 3(b), the agent determines whether the shield is protecting it by focusing on the area above the spacecraft and firing at the enemy. Figure 3(e) shows the ability of an agent trained using ARS to focus on multiple enemies in the environment; the agent is concerned about the enemy plane and making life and death decisions in the environment. Figure 3(d) and Fig. 3(a) show the ARS-A2C agent's score for its nearby environment. Figure 3(c) shows the agent repeatedly orienting the ball to part of the brick wall in order to pass through it through the tunnel.

The proposed method focuses on improving the ability of the agent to process the input information. The ARS module scores the state, which is then combined with the original input to produce a more meaningful state embedding so that the agent can better understand the current input. This ability to better understand input states allows the agents to make more informed decisions, which is essential for any reinforcement learning task.

5 Conclusion

This paper explains the DRL model from the perspective of a built-in explainer and explores how to use the explanation to improve the model's performance. An end-to-end trainable explanation method based on the A2C algorithm is proposed, in which an Adaptive Region Scoring mechanism is embedded into the agent. The proposed ARS approach embeds explainability into the agent system, achieving clear information visualization and excellent performance. Experimental results show that the model with the ARS outperformed the baseline model in various environments. In addition, the agent with ARS is proven to be capable of easily generating explanations, providing value for real-world applications. In future work, we plan to use a post-hoc explanation method to optimize the agents, and it will be interesting to integrate our ARS module with other DRL models, such as SAC [2] and Proximal Policy Optimization [7].

Acknowledgement. This work is sponsored by Shanghai Sailing Program (NO. 20YF1413800).

References

1. Brockman, G., et al.: OpenAI gym. arXiv preprint arXiv:1606.01540 (2016)
2. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870 (2018)

3. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094–2100 (2016)
4. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
5. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
6. Schulman, J., Moritz, P., Levine, S., Jordan, M.I., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. In: 4th International Conference on Learning Representations (2016)
7. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
8. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034 (2013)
9. Wang, X., Sugumaran, V., Zhang, H., Xu, Z.: A capability assessment model for emergency management organizations. Inf. Syst. Front. **20**(4), 653–667 (2018)
10. Wang, X., Yuan, S., Zhang, H., Lewis, M., Sycara, K.P.: Verbal explanations for deep reinforcement learning neural networks with attention on extracted features. In: 28th IEEE International Conference on Robot and Human Interactive Communication, pp. 1–7 (2019)
11. Wang, X., Lian, L., Yu, S.X.: Unsupervised visual attention and invariance for reinforcement learning. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 6677–6687 (2021)
12. Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1995–2003 (2016)