Visual Sparse Bayesian Reinforcement Learning: A Framework for Interpreting What an Agent Has Learned

Indrajeet Mishra, Giang Dao, and Minwoo Lee
Department of Computer Science
University of North Carolina at Charlotte
{imishra, gdao, minwoo.lee}@uncc.edu

***Abstract*— This paper presents a Visual Sparse Bayesian Reinforcement Learning (V-SBRL) framework for recording the images of the most important memories from the past experience. The key idea of this paper is to maintain an image snapshot storage to help understanding and analyzing the learned policy. In the extended framework of SBRL [1], the agent perceives the environment as the image state inputs, encodes the image into feature vectors, train SBRL module and stores the raw images. In this process, the snapshot storage keeps only the relevant memories which are important to make future decisions and discards the not-so-important memories. The stored snapshot images enable us to understand the agent's learning process by visualizing them. They also provide explanation of exploited policy in different conditions. A navigation task with static obstacles is examined for snapshot analysis.**

I. INTRODUCTION

There have been many recent developments that adopt deep learning to reinforcement learning (RL) problems that are driving innovation at the cutting edge of machine learning. Mnih, et al. [2] proposed a deep Q network (DQN) function approximation to play Atari games. DQN has convolutional neural network (CNN) layers to receive video image clips as state inputs to develop a human-level control policy. Silver, et al. [3] suggested a deterministic policy gradient, an actor-critic algorithm for continuous action space. Both the actor and critic are neural networks where the actor adjusts the policy in the direction of action-value gradient and the critic updates the action-value function. Mnih, et al. [4] discuss a framework which uses asynchronous variant of actor-critic algorithm where multiple agents are trained in parallel on the multiple instances of the environment which makes the learning faster and was able to achieve state of the art performance in the Atari games. Timothy, et al. [5] presents an actor-critic model-free algorithm based on deterministic policy gradient that operates over continuous action-space. It uses a CNN in order to learn directly from the raw image pixels and predicts the optimal action to take. These advancements lead a large number of successful applications to motion planning [6], game playing [7], natural language processing [8] and self-driving cars [6].

Although the deep reinforcement learning models have been highly successful in solving complex problems in many areas, the black box model makes it difficult to interpret what an agent learns. Yosinski, et al. [9] pointed out the major problem of deep learning as lack of visualization tools to understand the computation performed in the hidden layers. For this reason, many recent researches [10], [11], [12], [13], [14] have attempted different types of visualization to explore the learning process of a deep learning model. However, these methods focus on visualizing how the features are computed in the intermediate layers in supervised learning, so it is not enough to fully explain what it has learned and what the learned knowledge represents in reinforcement learning. Although the recent visualization efforts for reinforcement learning [15], [16] provides ways to interpret the learning of RL-Agents, there are still needs for tools that can explain the learning and exploitation process for trusty and robust model construction through interpretation.

Recently, Lee [1] proposed a Sparse Bayesian Reinforcement Learning (SBRL) approach to memorize the past experiences during the training of a reinforcement learning agent for knowledge transfer [17] and continuous action search [18]. SBRL provides an efficient understanding of the state-action space by analyzing the relevant data samples after the end of training that can explain how the learning has been influenced by the agent's past experience. Although SBRL can address the lack-of-interpretability issue in deep reinforcement learning, but it requires handcrafted feature engineering for state inputs, thus it can not be widely applicable, especially to the vision-based task where the agent perceives the environment through the vision sensors.

In this research, we extend the SBRL to make it work with the RL-based applications by creating end-to-end model with the visual presentation of experience to retrieve the significant snapshot images and store them in a snapshot storage. The proposed framework, named as Visual SBRL (V-SBRL), answers to the problem of lack of interpretability for various vision-based reinforcement learning problems and complements the previous reinforcement learning visualization approaches. The V-SBRL agent maintains a snapshot storage to store important experiences which can be easily visualized in order to understand what the agent remembers from the past and why the agent makes all the decisions. For instance, if an agent does not have any relevant snapshot for the state or neighboring states that require significant decision making, the agent does not know how to behave properly with insufficient knowledge in the new, unexperienced situations. This can be caused by lack of training,

978-1-5386-9276-9/18/$31.00 ©2018 IEEE

poor training strategy, or poor modeling of the environment, which can lead to mistakes or tragic failure. On the other hand, well-established snapshot storage can make the agent develop an optimal policy to select a right action at each time step as explain in section V.

The V-SBRL framework learns to remember important experiences and to discard the trivial ones to maintain sparse set of images. The framework contains a sparsity filter to sift the relevant states only. This helps tackle the other reinforcement learning problems efficiently by lowering the computational cost. The small set of images help us analyze the learning process and its outcomes more efficiently.For this, the framework has three layers of sparsity control. First, kernel function in relevance vector machine function approximator can be adjusted to the level of sparsity. Next, sparsity filter after mini-batch training of SBRL checks the validity of trained model to determine the new snapshot candidates to the storage. Last, the storage filters out entering the similar samples to existing snapshot by measuring the similarity with the kernel function.

The main contribution of this paper is that 1) it *extends* the SBRL framework to vision-based problem domains. 2) For this, adopting an autoencoder, the framework *encodes* the raw image state inputs for faster training. 3) It stores the past experiences to an image snapshot storage, which is used to *visualize* what the agent has learned from the interaction with an environment. 4) Lastly, we also present the *interpretation* which explains learning process, learned policies, and learned behavior in different environments.

In Section II, we review the visualization methods that have been proposed to understand the computational processes of deep learning. Section III explains about the different components of the framework, State Image Encoder, SBRL and the Snapshot Storage and how they are used in the framework as a whole. Section IV discusses the results achieved by the framework on the the navigation task and how the learning improves during training. Section V presents a detailed interpretation of the learned policy and how the collected snapshots affects the decisions made by the agent in different situations. It also discusses which experience plays an important role at any point of the time. Finally, we conclude in Section VI.

II. Related Work

Olah, et al. [10] combine the existing interpretation methods such as feature visualization [14] and attribution [19], [20]) to show what a CNN detects (feature visualization), and they explain how the detected features affect the outputs (attribution). It facilitates a way to understand the learnings of a neural network by visualizing the value computed by each neuron in all the network layers. For instance, visualization shows some neurons responsible to detect a cat's eye or a dog's ear.

Ribeiro, et al. [11] explain the prediction of a learning model with an approach, called LIME. This method focuses on finding an interpretable model that has an approximately same behavior as the original model. It seeks for an interpretable representation of the prediction model. In addition, the authors provide a method SP-LIME that can select a set of instances representing the task domain efficiently and explain the predictions with the selected instances in the domain.

Selvaraju et al. [21] present a generic method to generate the visual explanation for understanding the decisions made by CNN. This method uses the gradient that flows through the last convolution layer to create a localization map highlighting the important regions in the image, and it helps to understand the significance of the connections between the local receptive fields and the corresponding neurons.

For reinforcement learning interpretation, Zahavy, et al. [22] propose a model which enables us to visualize the learning strategy of a DQN agent using a Semi Aggregated Markov Decision to identify the spatio-temporal abstraction Processes (SAMDP) and apply t-SNE to the neural activation data for well-separated representation. This helps us understanding the similar groups of activation related to an action, but it requires a handcrafted feature engineering for it.

Thus, Greydanus, et al. [16] propose a visualization strategy for understanding the policy learned by a reinforcement learning agent using a perturbation-based saliency method. It was applied to Atari games where they observe the change in the behavior policy by making changes in the input images to understand the salient pixel locations in the image.

The methods proposed in [10], [11], [21] are more suitable for classification problems. The reinforcement learning visualization methods [16], [15] are not enough to interpret the learning progress and the learned policy. V-SBRL proposes a complementary approach to visualize what the agent has remembered from the past and how it helps the agent to take actions in the future.

Our proposed framework is analogous to the human learning and development. Humans often encounter a situation that is similar to prior experience. Instead of learning from scratch, we utilize our past experiences to adapt or master new environments or tasks. Focusing knowledge retention step, V-SBRL framework models the human-like learning to help a reinforcement learning agent to act the same way humans do by remembering the past experiences. It is very difficult for humans as well as a machine to remember everything from the past because of the limited memory capacity and computational/retrieval time so we only remembers the important experiences and use them in the new related tasks. Details of the proposed method are described in the following section.

III. Methods

Fig. 1 illustrates the proposed Visual Sparse Bayesian Reinforcement Learning (V-SBRL). It consists of three major components. First, the State Image Encoder deals with the high dimensionality of the image data. Second, the learning workhorse, SBRL [1], trains with mini-batch samples to estimate the Q values for the relevant states and action inputs. Using relevance vector machines [23], it captures

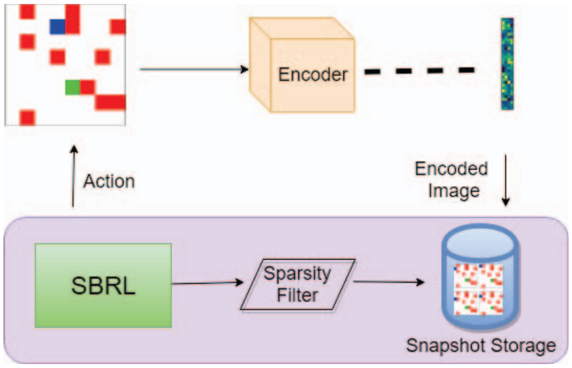


Fig. 1: Visual Sparse Bayesian Reinforcement Learning Framework

the significant samples (relevance vectors) to be examined as a candidate for snapshot recording. Third, the Snapshot Storage memorizes the relevant states action pairs from the candidates found in each episode training. The storage keeps re-evaluating the importance of relevant states found in the earlier episodes. The stored images are analyzed to enhance understanding of the learning experience.

A. *State Image Encoder*

State Image Encoder (SIE) reduces the dimensionality of the state image representation into a 64-dimensional feature vector without loosing any important information. The reduced dimensionality in feature representation enables efficient computation in training V-SBRL. Table. I presents the network architecture of SIE with encoder and decoder units. The SIE is a CNN-based autoencoder, which uses 4 convolution with 4 max-pooling and 1 fully connected layer for encoding and symmetric architecture consist of 4 convolution with 4 up-sampling and 1 fully connected layer for decoding. Turchenko, et al. [24] presented a CNN-based autoencoder which used similar network configuration of convolution, pooling and up-sampling and it provided the state-of-the-art performance. The SIE is pre-trained with a number of randomly generated images of the environment states by using the same image as the input to the encoder as well as the target outputs for the decoder. The difference between the pixel values of the original image and the reconstructed image is computed in the loss function to train the SIE network. The pre-trained SIE with the simulated sample generation converts the image inputs into feature vector representation as in Fig. 3. The encoded image seems to be lossless because of the simple environmental setup. However, the framework does not make any assumption or restriction on lossless encoding.

B. *Sparse Bayesian Reinforcement Learning (SBRL)*

Sparse Bayesian Reinforcement Learning [1] is a learning framework which follows the human traits of decision making via knowledge acquisition and retention. It refers to the past experiences stored in the snapshot storage and then finding similar tasks to current state, it evaluates the value of actions to select one in a greedy manner. It collects the training samples and then identifies the important moments (state-action pair), called snapshot, in each training episode. It adds the previously found relevant snapshots to the collected sample and re-evaluate their importance. The SBRL uses the relevance vector machines (RVM) [23] to identify the relevant snapshots and compute the posterior distribution of the snapshots. The SBRL transforms the state-action pair into kernel features for the nonlinear mapping. Since the action and state domain are orthogonal, the kernel function in Eq. (1) is defined as a product of separately defined Gaussian kernels for each state k_s and action k_a. The computed kernel feature vector is denoted as ϕ:

TABLE I: SIE-Architecture

Layer (Type)	Shape
input	32x32x3
Convolution	32x32x32
Max-Pooling	16x16x32
Convolution	16x16x64
Max-Pooling	8x8x64
Convolution	8x8x128
Max-Pooling	4x4x128
Convolution	4x4x256
Max-Pooling	2x2x256
Flatten	1x1024
Fully Connected	1x64
Fully Connected	1x1024
Reshape	2x2x256
Up-Sampling	4x4x256
Convolution	4x4x256
Up-Sampling	8x8x256
Convolution	8x8x128
Up-Sampling	16x16x128
Convolution	16x16x64
Up-Sampling	32x32x64
Convolution	32x32x3
Total number of parameters	1,481,219

$$\phi = k_s(\mathbf{s}, \mathbf{s}') \times k_a(\mathbf{a}, \mathbf{a}') \tag{1}$$

where

$$k_s(\mathbf{s}, \mathbf{s}') = e^{-\gamma_s ||\mathbf{s}-\mathbf{s}'||^2},$$
$$k_a(\mathbf{a}, \mathbf{a}') = e^{-\gamma_a ||\mathbf{a}-\mathbf{a}'||^2}.$$

Here, the Gaussian kernel parameters γ_s and γ_a can be separately defined.

SBRL [1] assumes that the target $\mathbf{Q}$ is a weighted sum of the feature vectors $\mathbf{\Phi w}$ such that:

$$\mathbf{Q}_{actual} = \mathbf{Q}_{prediction} + \boldsymbol{\epsilon} = \mathbf{\Phi w} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon}$ is zero-mean Gaussian noise with variance σ^2. Tipping et al. [25] suggested the initial variance: $\sigma^2 = 0.1 \times var(\mathbf{Q}_{actual})$.

Let vector $\boldsymbol{\alpha}$, a set of hyper-parameters controlling the strength of the prior over the corresponding weights, be infinity except for one starting as:

$$\alpha_i = \frac{||\phi_i||^2}{||\phi_i^\top \mathbf{Q}_{actual}||^2/||\phi_i||^2 - \sigma^2}.$$

We compute mean and standard deviation of weights $\mathbf{w}$ with $\mathbf{A} = \boldsymbol{\alpha}\mathbf{I}$:

$$\mathbf{\Sigma} = (\mathbf{A} + \sigma^{-2}\mathbf{\Phi}^\top\mathbf{\Phi})^{-1} \quad \text{and} \quad \boldsymbol{\mu} = \sigma^{-2}\mathbf{\Sigma}\mathbf{\Phi}^\top\mathbf{Q}_{actual}.$$

$\boldsymbol{\alpha}_i$ (when $q_i^2 > s_i$) and the noise level σ will be updated as following:

$$\alpha_i = \frac{s_i^2}{q_i^2 - s_i}, \qquad \sigma^2 = \frac{||\mathbf{Q}_{actual} - \mathbf{Q}_{prediction}||^2}{N - M + \sum_m \alpha_m \boldsymbol{\Sigma}_{mm}}$$

where $s_i = \boldsymbol{\phi}_i^\top \mathbf{C}_{-i}^{-1} \boldsymbol{\phi}_i$ and $q_i = \boldsymbol{\phi}_i^\top \mathbf{C}_{-i}^{-1} \mathbf{Q}_{actual}$ with the covariance of marginal likelihood without i-th sample contribution $\mathbf{C}_{-i}$ as defined in [25].

The algorithm re-computes $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ and the followed process until a convergent condition is met. The relevant state and action pairs can be traced back by the $\boldsymbol{\phi}$ left in the model. The predict Q-value for new data $\mathbf{x}^{new}$:

$$\mathbf{Q}_{prediction}^{new} = k(\mathbf{x}^{base}, \mathbf{x}^{new})\mathbf{w}^{base}$$

where $\mathbf{x}^{base}$ are significant samples, and $\mathbf{w}^{base}$ are the weights along with the samples.

To make the SBRL sparse, we have added a filter which checks whether the newly found relevant snapshots are good or not by comparing the mean squared error (MSE) of the values predicted with new snapshots and the actual target to the MSE of mean value of the target to the actual target.The snapshot is added to the storage if the difference between the above two MSE is less then a threshold value (user selected parameter) which leads to sparser solution.

$$\begin{aligned}\frac{1}{m} \times \sum_{i=0}^{m} (\mathbf{Q}_{actual}(\mathbf{s}_t, \mathbf{a}_t) - \mathbf{Q}_{prediction}(\mathbf{s}_t, \mathbf{a}_t))^2 < \\ t_h \times \frac{1}{m} \times \sum_{i=0}^{m} (\mathbf{Q}_{actual}(\mathbf{s}_t, \mathbf{a}_t) - \mathbf{Q}_{mean})^2\end{aligned} \tag{2}$$

where $\mathbf{Q}_{actual}$ is the actual target Q values, $\mathbf{Q}_{prediction}$ is the predicted Q values and $\mathbf{Q}_{mean}$ is the mean of all the actual Q values, t_h is the threshold value and m denotes the number of samples.

Apart from the MSE filtering, we also reduce the value of the kernel parameters, which result in the increase in the variance of Gaussian kernel and eventually increase the number of samples with high similarity measurements. As more similar samples are captured, RVM and Snapshot Storage can maintain a certain level of sparsity.

After the SBRL training is completed, the collected snapshots are used to predict the Q-values of any given state-action pair by applying the kernel to obtain the kernel features and then multiplying learned weights to it:

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{i=0}^{n} k(\mathbf{x}_i, \mathbf{x}_t) \times \mathbf{w}^i$$

where $\mathbf{x}_t = [\mathbf{s}_t, \mathbf{a}_t]$, and $\mathbf{w}^i$ denotes the weight of the i^{th} snapshot $\mathbf{x}_i$.

C. Snapshot Storage

The Snapshot Storage maintains the collection of all the snapshots (for important state-action pairs) learned by the SBRL during training. We add new snapshots from each episode to the storage after some filtering and similarity checks. The snapshots in the storage are appended to the new training samples in each epoch for training the SBRL so that we can keep re-evaluating the relevance of the snapshot found in earlier epochs because many of the snapshots in the early stage of training can be highly irrelevant. Moreover, this process implements the relevant experience replay [1].

To control the sparsity of the V-SBRL, we use three methods, and we have discussed the sparse kernel and the filter in the previous section. Third and last sparsity control is applied when the Snapshot Storage encounters a snapshot candidate that is already in the storage or a similar snapshot exists. For the same entry, Snapshot Storage updates the corresponding weights using the following convex combination:

$$\mathbf{w}^i = (1 - c) \times \mathbf{w_{old}}^i + c \times \mathbf{w_{new}}^i,$$

where $\mathbf{w_{old}}^i$ is the previous weight of the i^{th} snapshot, $\mathbf{w_{new}}^i$ is the new estimation of the weight of i^{th} snapshot and c is a convex combination coefficient that controls the effect of newly estimated weight on the old weight of the snapshot to be merged. For highly similar (not exactly matching) entry, it applies additional weighting of the kernel similarity as in

$$\mathbf{w}^i = (1 - c) \times \mathbf{w_{old}}^i + c \times (\sum_j k(\mathbf{x}_j, \mathbf{x}_i) \times \mathbf{w_{new}}^j).$$

where $\mathbf{w_{new}}^j$ is the weights of the snapshots $\mathbf{x}_j$ which are similar to the current snapshot $\mathbf{x}_i$.

This extra filtering leads to sparser snapshot solutions and help to avoid having too many similar snapshots in the storage. At the end of the SBRL training, the storage contains the manageable number of snapshots for analysis. The snapshot images are stored with corresponding actions taken and weight distributions. The magnitude of weights reflect the significance of the taken moment. Thus, for analysis, we read the weight distributions and relation to the learned policy.

IV. Experiments

We have applied the V-SBRL framework to the navigation task with static obstacles. The environment contains 64 cell blocks that can be obstacles, a goal or open spaces. Fig. 2 shows one of the randomly chosen navigation environments for training, testing and analysis. In this navigation environment, an agent interacts with the environment to learn the optimal policy which helps the agent to reach the goal as quickly as possible. There are four possible moves that the agent can take: moving up, moving down, moving left and moving right. An agent receives a positive reward of $+30$ if it reaches the goal and negative reward of -5 when it hits an obstacle. A reward of -1 is given as the free space navigation cost.

We run the experiment on the 8x8 cell navigation task environment. We have created a visual representation of the navigation task environment using the following notations:

- Each state of the navigation task environment (e.g. (0,0) or (5,3)) is represented by 4x4x3 pixels.
- All 4x4 pixels at that position will have pixel values (0, 0, 255) if it is the current position of the agent.

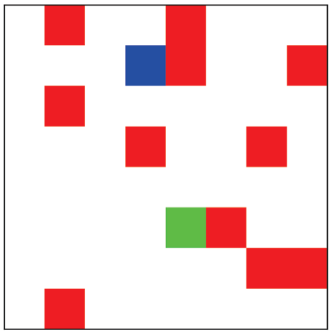


Fig. 2: A navigation task with static obstacles. The blue block represents the agent, the green block represents the goal and the red blocks represent obstacles.

- All 4x4 pixels at that position will have pixel values (0, 255, 0) if it is the goal.
- All 4x4 pixels at that position will have pixel values (255, 0, 0) if it contains an obstacle.
- All 4x4 pixels at that position will have pixel values (255, 255, 255) if it is a free space.
- States are encoded into a 64 feature vectors.

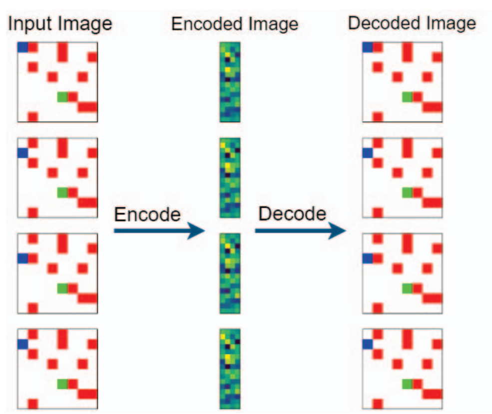


Fig. 3: Encoding and decoding with State Image Encoder

Fig. 3 shows some samples for encoding and decoding from pre-training. The outputs of the encoder are decoded to recover the original state inputs. In the figure, the first column displays the randomly selected, original image inputs, the second column in the middle displays the encoded features and the last column represents the reconstructed, decoded images.

Fig. 4 shows the collected snapshots during training an V-SBRL agent. The curve represents the total rewards earned by the agent in each episode. We observe that V-SBRL performs at par with one of the state-of-the-art algorithm, DQN. The images stored in Snapshot Storage are shown in the plot and the snapshot image set is augmented as the agent gains more experience. The agent initially has very small number of image snapshots, which increases as the agent gains more experience in the environment. However, it stops increasing after 250 epochs as the agent has identified the important snapshot to remember as it converges to near-optimal policies.

V. Snapshot Image Analysis

We analyze both training and testing stages to examine the efficacy of the recorded images in the Snapshot Storage. Fig. 5a shows the snapshot images just after the first epoch of training. Due to the limited experience, the agent has the small number of snapshots and even their weights are premature (mostly negative, denoted by the red frame) which are increased as the training proceeds. Snapshot Storage at the end of training contains increased number of snapshots with trained weights. The snapshots which are closely related with the important moments for optimal decisions have positive weights and the other snapshots have negative or close-to-zero weights (Fig. 5b).

Fig. 6 presents the means and standard deviations of the weights for the corresponding snapshots. This information reveals how important each snapshot is, how closely it is related to the learned optimal policy, and how confident the agent is about the impacts of each snapshot. The agent remembers the same state with different actions as a snapshot but the action which is most relevant for taking has higher weights and is selected as an optimal action. On the other hand, the snapshots with poor actions (i.e., hitting the obstacles or moving out of the working space) have negative weights which warns or prevents the agent from taking those actions.

We observe that the weight variances of the snapshots that are part of the optimal trajectory from the training start location are very low. On the contrary, the snapshots which do not fall in the optimal trajectory have larger variances which imply that the agent is not confident enough about those areas of the environment. This comes from the lack of exploration thus lack of experience. Thus, this analysis gives us a hint for improving exploration strategy or experience replay to increase more sampling on the under-samples areas.

The estimated Q-values after the training for all the states are presented in Fig. 7. The contour plot shows the maximum Q-value for the states over all the possible actions at that given state. For better understanding of how the snapshot affects the estimation of the Q values some of the snapshots are placed over the Q-plot. They indicate that snapshots with higher weights causes the Q-value to be higher and the snapshots with lower weights lead to small Q-values. Low variance of the snapshot in the high-Q value region illustrates that agent has explored this region enough and hence it is certain about the weights it has learned whereas the snapshots in the unexplored territory has high variance. This plot can gives a lucid understanding of how the learned policy is developed using the stored snapshot because the agent has taken the actions which lead to higher Q-value hence the snapshots with high similarity and high weights contribute more to that decision making.

Fig. 8 shows the exploitation trajectory followed by the agent from start location [1,3] and start location [0, 0] (same as training) along with the snapshot at some of the locations where the agent made an important decisions by

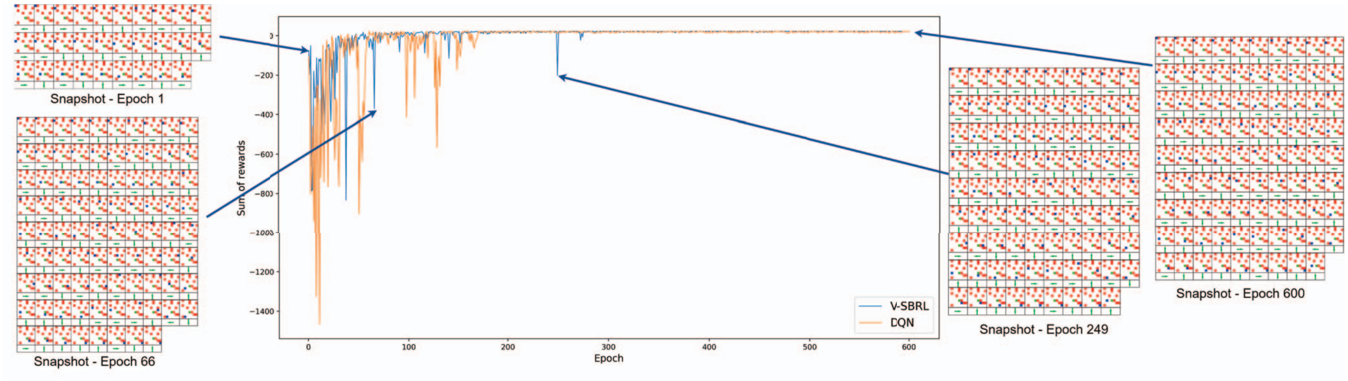
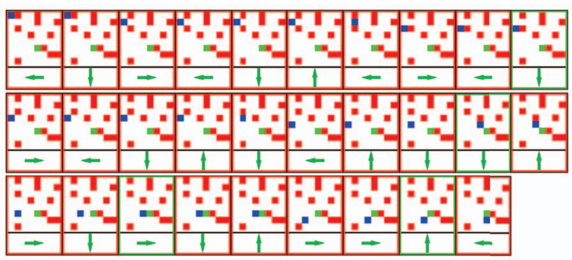
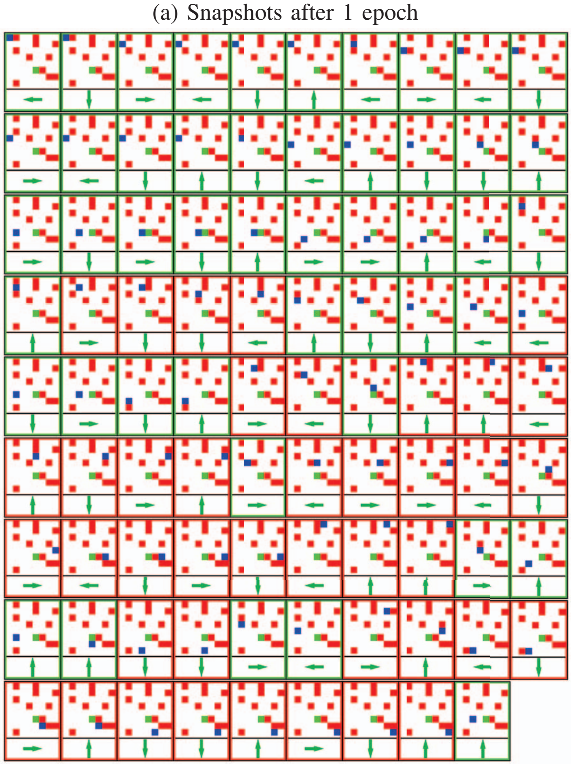


Fig. 4: Snapshot recording while training: It shows the sum of rewards for V-SBRL and DQN during the training along with the snapshot storage at some epochs in V-SBRL.Each snapshot contains the image of the environment state and the action taken from the state (denoted by arrow).



(a) Snapshots after 1 epoch



(b) Snapshots after 600 epoch

Fig. 5: Collected snapshots during training. Red boundary represents the negative weights and green boundary represents positive weights, The arrow represents the action taken in that state.

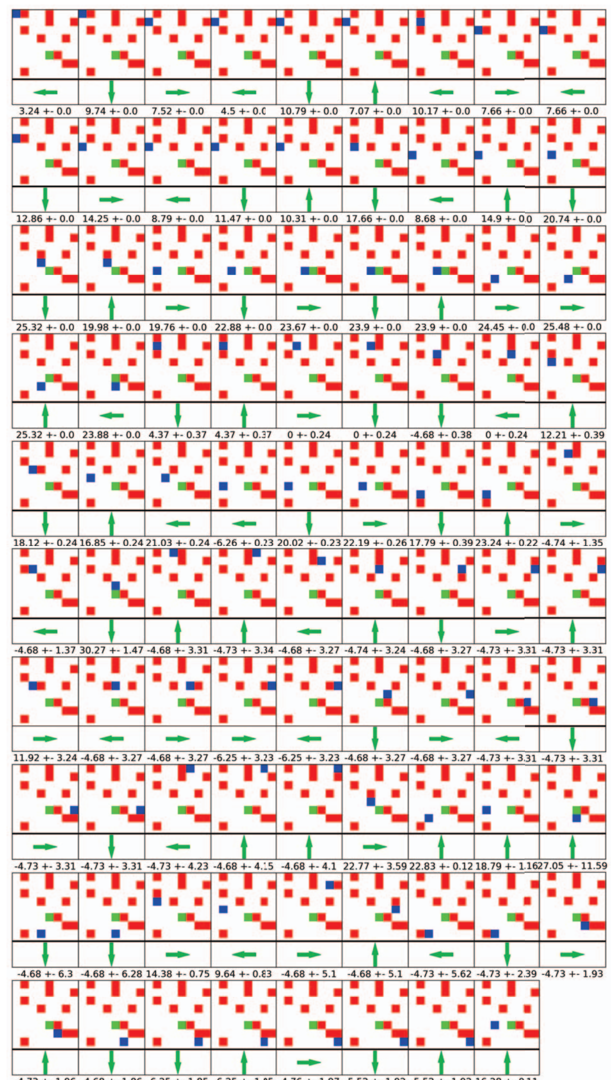


Fig. 6: The obtained snapshots after training with the corresponding weight distributions: the numbers below each snapshot represent the mean and standard deviation of the weight.

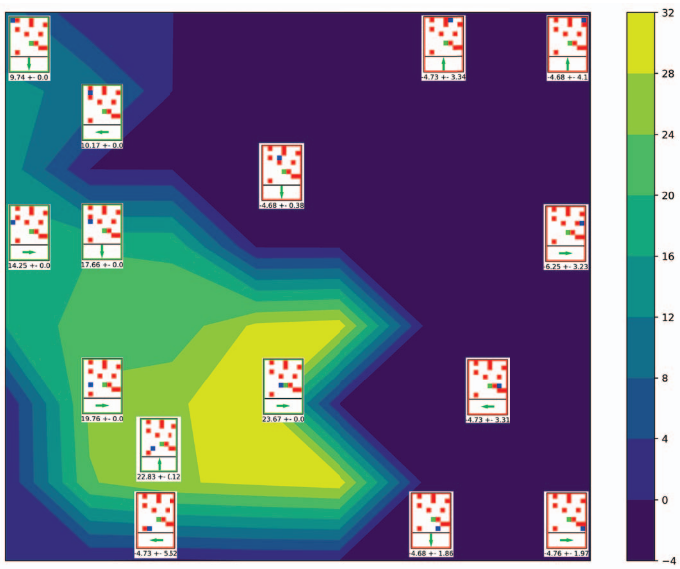
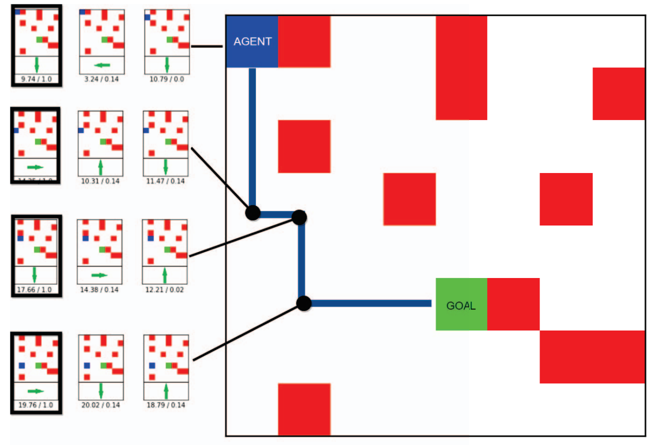
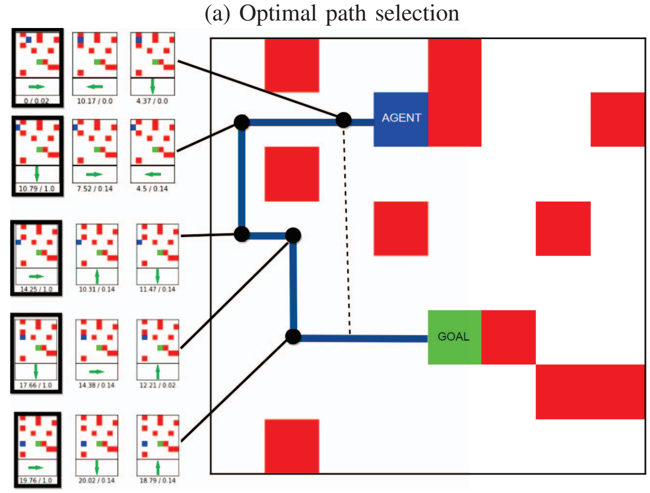


Fig. 7: Relevant snapshots over Q contour plot: The plot also shows few of the snapshot from the storage and it shows that the snapshots which are part of optimal trajectory contribute positively to decision making and agent is more confident on this learning, whereas the snapshots in non-optimal or not-explored paths do not have same contribution.



(a) Optimal path selection



(b) Wrong decision causes a suboptimal solution.

Fig. 8: Exploitation of the learned policy with different starting positions: small images on the left are the snapshots for each circled moment. The numbers under each snapshot are weights and kernel similarity values.

changing the direction of the motion. Snapshots in Fig. 8a shows an optimal trajectory followed by the agent. The three snapshots shown with the connection to the denoted state are highly relevant snapshots to the current state, which is evaluated with the kernel function by the agent. Correctly capturing most significant decisions at the right moment with corresponding positive weights lead the agent to follow the optimal path. On the other hand, Fig. 8b shows a non-optimal trajectory from a different start location (not same as training). The first row of the snapshot does not contain proper decision records, which lead the agent to make a wrong decision at the first junction. At this first circled point in the trajectory, all the snapshots have very low similarity and hence the wrong decision is expected. This tells us the training strategy from the same starting point generates biased samples, which cause lack of exploration when it is away from the optimal path (the dashed line).

The V-SBRL enables us to understand the policy learned by the agent and how the learning evolved over the training. Having a tool to visualize the learned policy helps us better understanding “why” and “how” the agent makes a decision. For instance the Fig. 8b shows non-optimal trajectory followed by the agent, which we can be attributed to the less exploration of that region after seeing the related snapshots which bears low similarity with the state–this shows that the agent does not have similar experience to make a right decision.

Though the proposed framework works well in this task but it has collected 89 snapshots which can be further reduced. It can be reasoned by the fact that it is a discrete environment hence states-action pairs bears a low similarity between each state which results in more number of snapshots. The V-SBRL will be more useful in a continuous state-action environment, which can lead to sparse snapshots storage. The continuous domains have large number of similar state-action samples observed and they can be easily reduced one snapshot representation using the suggested sparsity control and further improved one in the future.

When the quality of snapshots are ensured, the recorded image snapshots can be a good knowledge representation for analyzing transfer learning. It will be able to adapt to the similar task environments using the previous experiences which was may be from a similar but a different task. For example, if we change position of some of the obstacles in this navigation task, the agent would still be able to find a solution if the relevant snapshots are well collected. The agent will easily recall and reference its past experience with high similarity, especially when the new tasks do not alter the source task to a very large extent (smooth shaping [26]).

VI. Conclusions

In this paper, we present a visualizable and explainable reinforcement learning framework V-SBRL which extends SBRL [1] to enable visual perception to store the important past experiences as images. This helps us to understand how the agent makes decisions based on previously memorized experiences. Finding the most similar snapshots to the given state, the agent retakes the remembered decisions of the same or similar situations (this action choice is determined through Q function evaluation). The framework contains State Image Encoder, a CNN auto-encoder which reduces the dimensionality of the image inputs to lower dimensional feature vector for efficient computation. This automated feature mapping enables the SBRL to capture the significant samples in the mini-batch samples in the reduced feature space. The Snapshot Storage stores the relevant experiences found in each episode but at the same time it also updates the weight of each snapshot if it changes in later episodes. We have proposed three techniques to control the sparsity of the snapshot storage: 1) evaluating snapshots using MSE to check if the snapshot are good, 2) capturing the similar samples efficiently via the choice of the kernel and its parameters, and 3) filtering out the similar snapshots to avoid unnecessary snapshot entries in the storage. The V-SBRL provides us the small number of experiences for an agent to remember in order to develop an optimal policy in other next tasks.

This leads us to the next step of our research to further investigate sparse module/kernel/filter design. Improved sparsity is expected to make the analysis easier and lower the computational complexity. To examine sparsity and efficacy of model, we will extend the experiments to complex problems in continuous state-action space (i.e., Five, Atari or Flash games). We believe that the V-SBRL would give more meaningful analysis based on a sparser set of snapshot images.

Acknowledgment

This work was supported, in part, by funds provided by the University of North Carolina at Charlotte. The Titan Xp used for this research was donated by the NVIDIA Corporation.

References

[1] M. Lee, "Sparse bayesian reinforcement learning," Ph.D. dissertation, Colorado State University, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*, 2014.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[8] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," *CoRR*, vol. abs/1606.01541, 2016. [Online]. Available: http://arxiv.org/abs/1606.01541

[9] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *CoRR*, vol. abs/1506.06579, 2015. [Online]. Available: http://arxiv.org/abs/1506.06579

[10] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, 2018, https://distill.pub/2018/building-blocks.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1135–1144.

[12] H. Li, Z. Xu, G. Taylor, and T. Goldstein, "Visualizing the loss landscape of neural nets," *CoRR*, vol. abs/1712.09913, 2017. [Online]. Available: http://arxiv.org/abs/1712.09913

[13] G. Montavon, W. Samek, and K. Müller, "Methods for interpreting and understanding deep neural networks," *CoRR*, vol. abs/1706.07979, 2017. [Online]. Available: http://arxiv.org/abs/1706.07979

[14] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization.

[15] T. Zahavy and N. Baram, "Graying the black box: Understanding DQNs." 2017. [Online]. Available: https://arxiv.org/pdf/1602.02658.pdf

[16] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017.

[17] M. Lee and C. W. Anderson, "Can a reinforcement learning agent practice before it starts learning?" in *International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 4006–4013.

[18] M. Lee and C. W. Anderson, "Relevance vector sampling for reinforcement learning in continuous action space," in *Proceedings of 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 774–779.

[19] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[21] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR, abs/1610.02391*, vol. 7, 2016.

[22] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," *CoRR*, vol. abs/1602.02658, 2016. [Online]. Available: http://arxiv.org/abs/1602.02658

[23] M. E. Tipping, "The relevance vector machine," in *Advances in Neural Information Processing Systems*. MIT Press, 2000, vol. 12, pp. 652–658. [Online]. Available: http://papers.nips.cc/paper/1719-the-relevance-vector-machine.pdf

[24] V. Turchenko, E. Chalmers, and A. Luczak, "A deep convolutional auto-encoder with pooling-unpooling layers in caffe," *arXiv preprint arXiv:1701.04949*, 2017.

[25] M. E. Tipping, A. C. Faul *et al.*, "Fast marginal likelihood maximisation for sparse bayesian models," in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

[26] T. Erez and W. D. Smart, "What does shaping mean for computational reinforcement learning?" in *7th IEEE International Conference on Development and Learning (ICDL)*, 2008, pp. 215–219.