Visual Analytics for RNN-Based Deep Reinforcement Learning

Junpeng Wang, Wei Zhang, Hao Yang, Chin-Chia Michael Yeh, and Liang Wang

Abstract—Deep reinforcement learning (DRL) targets to train an autonomous agent to interact with a pre-defined environment and strives to achieve specific goals through deep neural networks (DNN). Recurrent neural network (RNN) based DRL has demonstrated superior performance, as RNNs can effectively capture the temporal evolution of the environment and respond with proper agent actions. However, apart from the outstanding performance, little is known about how RNNs understand the environment internally and what has been memorized over time. Revealing these details is extremely important for deep learning experts to understand and improve DRLs, which in contrast, is also challenging due to the complicated data transformations inside these models. In this paper, we propose *Deep Reinforcement Learning Interactive Visual Explorer* (*DRL*IVE), a visual analytics system to effectively explore, interpret, and diagnose RNN-based DRLs. Focused on DRL agents trained for different Atari games, *DRL*IVE targets to accomplish three tasks: game episode exploration, RNN hidden/cell state examination, and interactive model perturbation. Using the system, one can flexibly explore a DRL agent through interactive visualizations, discover interpretable RNN cells by prioritizing RNN hidden/cell states with a set of metrics, and further diagnose the DRL model by interactively perturbing its inputs. Through concrete studies with multiple deep learning experts, we validated the efficacy of *DRL*IVE.

Index Terms—Deep reinforcement learning (DRL), recurrent neural network (RNN), model interpretation, visual analytics.

1 Introduction and Motivation

Reinforcement learning (RL) [1] strives to train a software agent that can issue proper actions to interact with an environment and maximize the accumulated reward. With the recent wide-adoption of deep neural networks (DNNs), deep reinforcement learning (DRL) has also achieved multiple breakthroughs, most notably in playing the Go [2] and Atari [3], [4] games. Each game is an *environment* with varying *states* at different game stages. The *agent* consumes the states and responds by issuing *actions* following certain playing strategies. The environment, in turn, provides *rewards* (along with the next game state) as feedback to the agent to optimize its strategies. A typical RL interaction loop with Atari games is shown in Fig. 1.

Two major approaches to implementing DRL are convolutional neural network (CNN) based [3] and recurrent neural network (RNN) based [5]. Their major difference lies in the way of encoding the dynamic states' information from the environment. For example, the states of the Pong game should reflect not only the position (static information) but also the speed and direction (dynamic information) of the ball. CNN-based approaches capture the dynamic information by combining a sequence of consecutive game screens as a state [3], whereas RNN-based approaches encode this through RNNs' internal cells [5]. We focus on RNN-based approaches in this work for three reasons. First, RNN-based implementations usually involve a CNN component to preprocess individual game screens and thus are more complicated. Second, although there have been multiple works interpreting RNNs [6], [7], [8], [9], [10], [11], most of them are designed for text or linguistic data and few discoveries on RNNs for DRLs have been reported. Third, compared to other time-series data, the game state sequences from DRLs are more interesting, complicated, and challenging.

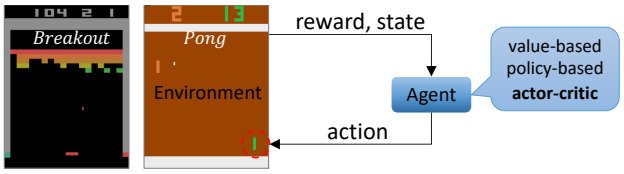


Fig. 1. The RL interaction loop. Atari games (e.g., Breakout and Pong) are the environments that the agents interact with to maximize rewards.

Apart from the superhuman performance of RNN-based DRL models, exploring, interpreting, and diagnosing these models remain challenging. *First*, there are few tools that DRL researchers can directly use to efficiently explore a long game episode (from the game-start to the game-end). Often, the evaluation of a DRL agent is limited to aggregated statistics (e.g., rewards per episode) or eyeballing cherry-picked game snippets. Many details about the agent's playing strategies are waiting to be revealed. *Second*, identifying the "interpretable RNN cells" and interpreting what they have captured in the context of different games are challenging, due to the high-dimensional (HD) data representations inside RNNs. *Third*, the capability of flexibly diagnosing a DRL model at certain game steps is still missing. Given that DRL researchers often need to verify various hypotheses on the model's functionalities, this capability is in strong need.

To address these challenges, we propose a visual analytics system (*DRL*IVE) to ***explore***, ***interpret***, and ***diagnose*** RNN-based DRLs. We focus on DRLs trained on different Atari games [4], as they are well-known by the public and the most commonly used RL testbeds. Three analytical tasks

• *J. Wang, W. Zhang, H. Yang, C.-C. M. Yeh, and L. Wang are with Visa Research, Palo Alto, CA, 94301.*
E-mail: {junpenwa, wzhan, haoyang, miyeh, liawang}@visa.com

Manuscript received April xx, 20xx; revised August xx, 20xx.

have been targeted and carried out in this work:

- Enable efficient and thorough explorations of a long game episode to understand and interpret the playing strategies learned by DRL agents (Sec. 5.1, 5.2 and 6.1);
- Identify critical hidden/cell states of RNN-based DRLs and interpret what they have captured (Sec. 5.3 and 6.2);
- Empower DRL researchers to interactively perturb different DRL models' inputs (i.e., game screens) to probe and verify the model's functionalities (Sec. 5.4 and 6.3).

Our work is conducted in close collaboration with multiple DNN researchers and we verify the efficacy of *DRL*IVE with them through concrete studies. In summary, we contribute to visualization research from the following perspectives:

1) *Visual knowledge discovery in the context of RNN-based DRL.* Collaborating with multiple deep learning researchers, we distill detailed design requirements in analyzing RNN-based DRL models, based on which, we develop an integrated multi-view visual analytics system (*DRL*IVE). Our studies with the system, as well as the derived insights, demonstrate that visual analytics can effectively facilitate the process of exploring, interpreting, and diagnosing RNN-based DRL models.
2) *Comparative analysis on high-dimensional RNN states.* We introduce a scalable method to prioritize a large number of high-dimensional RNN state dimensions, so that the interpretable RNN cells can be discovered more effectively. The method relies on different metrics derived by relating the RNN states with the temporal DRL statistics (e.g., reward sequences, actor/critic value sequences). Sorting RNN state dimensions by these metrics, we can efficiently identify the most important ones.
3) *An interactive saliency map generation algorithm.* We employ a perturbation-based method to generate saliency maps that annotating the critical regions from the inputs of DRL models. By interacting with *DRL*IVE, users can generate these maps at critical game steps on-demand to verify the functionality of different RNN cells, providing visual evidence to support their analysis.

2 Related Work

Visual Interpretation of DNNs. As shown by many recent works [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], interpreting DNNs with visual analytics has achieved great success. According to their learning styles, DNNs are often categorized into supervised, unsupervised, and reinforcement learning. Visual analytics works have covered all three types. For supervised DNNs, visualization solutions have been proposed to interpret [25], [26], diagnose [27], [28], and even improve [29], [30], [31] them. For unsupervised DNNs, many visual analytics attempts focus on deep generative models to monitor the training process [32], compare the generated data [33], and visualize the adversarial training process [34]. For DRLs, DQNViz [35], DynamicsExplorer [36], and DRLViz [37] are three pioneering works, to which our work is closely related.

DQNViz [35] was proposed to investigate a CNN-based DRL model, named Deep Q-Network [3]. It took advantage of pattern mining algorithms to identify typical agent movement patterns, and used them to understand the agents' playing strategies. Those strategies, in turn, helped to refine the model. Nevertheless, DQNViz is readily applicable to a few Atari games with 1D movement space only. DynamicsExplorer [36] analyzed the behavior of an RNN-based DRL agent under different dynamics settings to cluster the settings of success and/or failure cases. With this tool, domain experts were able to characterize the environmental changes and relate them with RNNs' internal states, which helped the experts in understanding how the agent worked. This tool was designed particularly for a game simulator called "ball-in-maze" to serve a robot-control task. DRLViz [37] focused on ViZDoom [38] and designed scalable heatmaps to disclose the internal states of RNN-based DRLs. Different from the above works, we design *DRL*IVE without using any game-specific settings and intend to make it applicable for all RNN-based DRL models. The three components of *DRL*IVE (Sec. 5) all take common forms of RL data and the cross-game findings (Sec. 6) provide insight not only into the RL environments but also into the underlying RNN models. Moreover, we empower interactive perturbation on RNN-based DRLs, so that DRL researchers can interactively verify the functionality of different internal RNN state dimensions.

Visual Analytics for RNN Models. We focus on RNN-based DRL models, and there are several outstanding visualization works for RNN interpretations [6], [7], [8], [10], [11]. For example, RNNVis [7] focused on text data and interpreted RNNs by investigating the expected response of internal states to different input texts. The proposed visualization can be adapted to both Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), the two most popular RNN architectures. LSTMVis [6] allowed users to effectively select an interested range of the sequential inputs and match the state change patterns from large textual datasets. Based on the collective behavior of LSTMs' internal states, the authors illustrated how LSTMs could capture the structures of sequential inputs. Karpathy et al. [8] decoded how RNN hidden/cell states captured the structure of a sentence using heatmaps, which significantly improved the understanding of RNNs (e.g., characters inside/outside of quotation marks correspond to different active-levels of the RNN states). Our work has a similar goal of revealing what different RNN internal states have captured. However, we focus on more complicated sequential inputs from different video games in the context of reinforcement learning.

Focused on ***the actor-critic DRLs*** [39] for Atari games, Greydanus et al. [40] proposed a method to perturb the input game screens to see their impact on the *actor* and *critic* values. Specifically, for each game screen, they generated a Gaussian blurred image and interpolated it with the original screen at individual pixels through a Gaussian mask to generate numerous perturbed images. Feeding the perturbed images into the studied DRL model, they could check what perturbations change the *actor* and *critic* values more significantly, and thus generate a saliency map. Extending the process to all game screens, they generated saliency-map videos for different games for analysis. Also, the authors explored the impact of RNN memory cells and presented some preliminary results. Gupta et al. [41] further enhanced this approach by balancing the *specificity* and *relevance* during the perturbation. We adopt a similar approach to interactively perturb the input game screens, but our perturbation focuses more on investigating the RNN hidden/cell states

(rather than the *actor* or *critic* values). Moreover, instead of blindly applying the perturbation method to all game steps, we only apply it to the important RNN state dimensions at critical game steps (identified from our visual analytics system). These on-demand perturbations help to efficiently capture the important agent's behaviors and effectively interpret the functionality of the corresponding RNN states.

3 Background

The interaction between an RL agent and an environment generates a sequence of states, actions, and rewards (denoted as s_t, a_t, and r_t, $t\in[1,T]$). The goal of the agent is to maximize the cumulated reward from any time t ($\sum_{i=t}^{T} r_i$). In general, there are three RL methods to achieve this [42]: policy-based, value-based, and actor-critic RLs. The policy-based approach [43] trains the agent by optimizing the probability distribution of the actions (conditioned on states, i.e., $\pi(a_t|s_t)$) to derive the optimal policy, e.g., REINFORCE [44]. The value-based approach shares the same goal of deriving the optimal action policy, but it achieves this by optimizing an associated value function, e.g., Q-learning [45]. The actor-critic approach [39] combines the merits of both. The ***actor*** part employs the policy-based approach to optimize the action policy, whereas the ***critic*** part uses the value-based approach to optimize a critic value, which tells how good the current policy is. We focus on the actor-critic approach in this work and its loss function is the sum of a value loss and a policy loss, i.e., $\mathcal{L}=\mathcal{L}_{value}+\mathcal{L}_{policy}$,

$$\mathcal{L}_{value}=\sum_{i=t}^{T}(R_i - V(s_i))^2, \quad R_i=r_i+\gamma V(s_{i+1})$$

$$\mathcal{L}_{policy}=-\sum_{i=t}^{T}\log(\pi(a_i|s_i))A_i$$

- **The value loss** $\mathcal{L}_{value}$ minimizes the error between the critic value, i.e., $V(s_i)$, estimated by DRL models and the expected future reward R_i. R_i is the sum of the reward from the current step r_i and the discounted rewards from future steps. The discount factor γ ranges from 0 to 1 (here γ=0.99), which punishes future rewards (predicted by the DRLs) to accommodate the stochastic environment [46].
- **The policy loss** $\mathcal{L}_{policy}$ maximizes the probability of generating a_i when seeing s_i, i.e., $\pi(a_i|s_i)$. A_i is the *advantage function* measuring how much better taking action a_i could be when seeing s_i. As this function is complicated and its understanding does not affect us in introducing our work, we put its explanations into our supplemental material.

$V(s_i)$ and $\pi(a_i|s_i)$, i.e., the ***critic value*** and ***actor policy*** function, are parameterized by our DRL model, which uses an RNN on top of CNN extracted activations (from individual game screens) to capture the game's temporal evolution.

Fig. 2 shows our DRL model's architecture in details. For a game screen at step t, i.e., obs_t with size $160\times210\times3$ ($width\times height\times channel$), the model first converts it to gray-scale and shrinks it to $80\times80\times1$, and then applies a CNN on top of it to extract the essential information. In our case, a CNN with four convolutional layers is used, each halves the size of the input, and the size of the last layer activation is $5\times5\times32$ (s_t), which is the input of the RNN.

Two major architectures of RNNs are LSTM and GRU. LSTM has both hidden and cell states, whereas GRU contains only hidden states. Here, we focus on interpreting

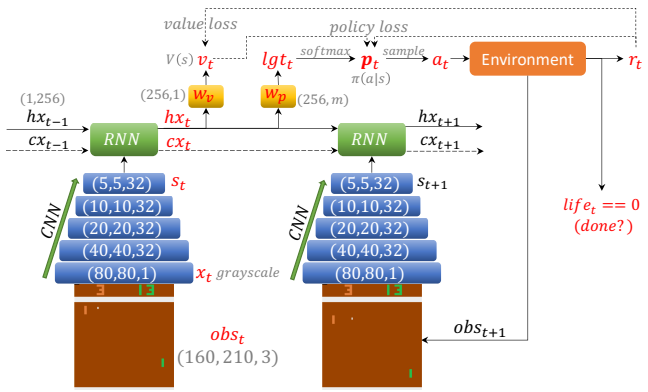


Fig. 2. The architecture and data-flow of our DRL model. The CNN processes individual game screens first to extract the essential game state information. The RNN then takes the sequence of extracted information (CNN activations) as input to capture the temporal game evolution.

LSTM, as it is more complicated. Each RNN cell can be considered as a black-box, which combines the current game state (s_t), the previous hidden (hx_{t-1}) and cell state (cx_{t-1}) to produce a new hidden (hx_t) and cell state (cx_t), i.e.,

$$(hx_t,\ cx_t)=RNN(s_t,\ (hx_{t-1},\ cx_{t-1})),\quad s_t=CNN(x_t).$$

hx_t is then used to generate the value of $V(s_t)$ and $\pi(a_t|s_t)$, i.e., v_t and $\mathbf{p}_t$, through two separate fully-connected layers whose weights are w_v and w_p respectively. We use an RNN with D hidden state dimensions (D=256). Therefore, w_v has a shape of $D\times1$ and w_p has a shape of $D\times m$ (m is the number of possible actions). The ith component of w_p (w_p^i) is the weight used to derive the probability of the ith action,

$$v_t=\sum_{d=0}^{D-1}(w_v[d]\cdot hx_t[d]) + bias_v$$

$$\mathbf{p}_t=softmax(lgt_t)=softmax(\sum_{d=0}^{D-1} w_p[d]\cdot hx_t[d]+bias_p).$$

Sampling from the policy distribution will generate an action ($a_t\sim\mathbf{p}_t$), which will be sent to the environment to generate the reward (r_t) and next game screen (obs_{t+1}). r_t provides feedback to v_t to update the network parameters (value loss). r_t and v_t together guide the change of $\mathbf{p}_t$ (policy loss). With hx_t, cx_t, and obs_{t+1}, the above process will be repeated until the end of a game episode.

Note that, our work is not limited to the specific network architecture presented in Fig. 2. Since we only use the input and output of the CNN and RNN models, their architecture details (e.g., number of convolutional layers, filter sizes, RNN states' dimensionality) have little impact on our work.

4 System Design Requirements

Given a well-trained DRL agent, what playing strategies has the agent learned and how did the agent learn them? We study this problem in collaboration with five deep learning (DL) experts ($E1\sim E5$). $E2$ is a research scientist focusing on time-series data analysis and sequence modeling. $E1$, $E3\sim E5$ are principal researchers using DL models for various prediction tasks in their daily work. All experts have 5+ years of experience with DNNs, especially RNNs. $E1$, $E2$, and $E3$ participated since the task analysis stage, during which, we met with them to discuss how the problem was approached conventionally, what inconveniences were facing, what would be the desired visualizations helping to

understand the problem, etc. For example, they mentioned that people often analyze DRL agents through aggregated metrics (e.g., average rewards per episode) using TensorBoard and randomly eyeball game snippets in hoping to catch the agent's frequently used playing strategies, which is not efficient and cannot understand the agent in-depth. Some metrics guiding them to the important game steps and UIs helping to conveniently replay the game would be very useful. Over these discussions, we elicited and iteratively refined three themes of design requirements on (R1) ***episode exploration***, (R2) ***RNN internal state investigation***, and (R3) ***interactive model perturbation***, detailed as follows.

- ***R1: efficient game episode overview and on-demand explorations.*** This requirement serves the goal of enabling the experts to efficiently explore a long game episode and investigate the behaviors of a trained agent. As a game episode could last for thousands of steps, an efficient and effective overview often works as the starting point for further investigations. This requires our system to:
  - R1.1: monitor the evolution of user-interested RL data sequences/metrics, e.g., action or reward sequences.
  - R1.2: provide an overview of all steps and the internal representation of the step data (e.g., extracted CNN activations) at different intermediate stages of the model.
  - R1.3: identify critical game steps over an episode and replay the game from any steps of interest on-demand.
  - R1.4: flexibly interact with (e.g., select, filter, or aggregate) critical game steps or game sequences to reveal the agent's playing strategies or behavior patterns.
- ***R2: in-depth hidden/cell state investigation and interpretation.*** The ultimate goal of this requirement is to identify the "interpretable RNN cells" from the trained DRL models. The existence of these cells has been shown by Karpathy et al. [8] in the text-domain. However, several challenges arise when analyzing the more complicated DRL models, which needs *DRL*IVE to:
  - R2.1: efficiently identify important RNN hidden/cell state dimensions, and the dimensions that significantly impact critical steps, e.g., steps of kicking the ball.
  - R2.2: answer how an agent notices the environmental changes (e.g., the ball direction in Pong) and what the agent has memorized over time using RNN states.
  - R2.3: recognize dimensions of the RNN hidden/cell states that behave dissimilarly in different subsets of steps, e.g., steps with and without rewards.
- ***R3: flexibly perturb DRLs' inputs to verify the functionality of RNN states.*** With the identified RNN state dimensions, DRL researchers often need to validate the dimensions' functionality at critical game steps. One practical way of doing this is to investigate how the model behaves if removing some critical input features [40], e.g., would a dimension still be active if the ball did not appear in the Pong game? Therefore, this requirement targets to provide users the flexibility in perturbing a selected game screen (from a critical game step) and interactively diagnose the functionality of different internal states.

5 VISUAL ANALYTICS SYSTEM: *DRL*IVE

*DRL*IVE is designed to *explore*, *interpret*, and *diagnose* DRLs through three coordinated views. The ***Episode*** view (Fig. 3a, R1) adopts the *Overview+Details* exploration to enable users to efficiently explore long game episodes and flexibly dive into important game snippets. The ***Projection*** view (Fig. 3c, R1) presents the intermediate data from five stages of the DRL data transformation pipeline, empowering users to see through the DRL "black-box". The ***Dimension*** view (Fig. 3b, R2) allows users to sort RNN hidden/cell state dimensions based on different metrics to quickly identify the important ones and interpret their encoded semantics. All views are coordinated to help users identify critical game steps and RNN dimensions. With these steps/dimensions, users can interactively perturb the model's input for diagnosis (R3).

5.1 Episode View

A common scenario in analyzing DRL agents is to explore game episodes with lots of steps and examine a few critical ones to understand the agents' behaviors. The Episode view (Fig. 3a), empowered with the *"Overview+Details"* exploration capability via a scalable line-chart and a game-replay view, targets to facilitate this exploration process (R1).

The **line-chart** (Fig. 3-a1) presents a temporal sequence over a game episode (R1.1), which could be the *critic values* ($v_t, t\in[1,T]$), rewards (r_t), RNN hidden states of a particular dimension d (i.e., $hx_t[d]$), etc. Multiple sequences can be superimposed by interacting with the widget in Fig. 3-a3. To present the *actor policy* $\pi(a_t|s_t)$, we compute the entropy of $\pi(a_t|s_t)$ at individual steps and the resulting entropy sequence can similarly be presented as a curve (the blue one in Fig. 3-a1). A large entropy indicates a flat probability distribution, whereas a low entropy implies a distribution dominated by a single action. Low entropy steps are of more interest, as they reflect the agent's on-purpose behaviors.

The **game-replay view** (Fig. 3-a2) plays a selected sequence of game screens (obs_t) as an animation. From the line-chart, users can zoom into a particular step range for focus. In the focused range (Fig. 4b), circles with different colors are used to represent the actions taken at the corresponding steps. Clicking on any circle will trigger the game-replay view to replay the game from that step. The widget in Fig. 3-a4 controls the number of steps to be replayed. The bar-chart on the right of the animation (Fig. 3-a2) presents the action policy $\pi(a_t|s_t)$ predicted for the selected step. Different glyphs are designed to represent different actions (Fig. 4d). The one with a shaded background (i.e., *down* in Fig. 4d) is the action taken at the current step.

Users can select steps with values greater or less than a threshold through a thresholding line, i.e., dragging the red horizontal line in Fig. 3-a1. Steps on the blue curve with values *less* than the threshold are selected (in shade) and they are highlighted in the Projection view (the orange circles in Fig. 3-c2). One can also interact with the thresholding line to select steps with values *greater* than a particular threshold.

5.2 Projection View

DRL models are often regarded as "black-boxes" that consume game screens and generate agent actions. To understand them, researchers need to know how they transform data internally. Based on our discussions with the experts, the data from 5 essential model stages have been identified (R1.2): the input game screens, CNN extracted features,

Fig. 3. *DRL*IVE: (a) the Episode view provides a quick overview of the entire game episode and allows users to zoom into specific game steps for detailed investigation; (b) the Dimension view empowers users to explore the numerous hidden/cell states of RNN models and interpret them; (c) the Projection view presents data in five stages of the DRL data transformation pipeline (i.e., from input game screens to output action probabilities).

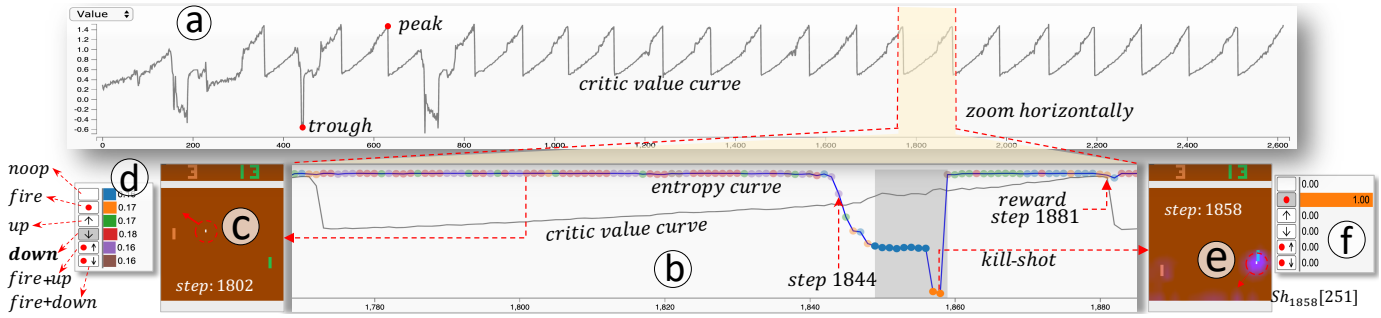


Fig. 4. Overview one game episode (a) and dive into a reward cycle to investigate the agent's playing strategies (b). The action probabilities, game screens, and saliency maps can be retrieved on-demand (c-f). The design and exploration details of this view are in Sec. 5.1 and 6.1 respectively.

RNN hidden states, RNN cell states, and the final action logits. These data are denoted as x_t, s_t, hx_t, cx_t, and lgt_t ($t \in [1, T]$) respectively in Fig. 2. Missing data from any above stages will fail to present the full picture of the data transformation pipeline. Adding data from more intermediate stages will help but also more-or-less overlap with these data. Therefore, we focus on these five stages and present the HD data in five synchronized tSNE projections (Fig. 3c).

If one game episode contains T steps and the number of possible actions is m, our DRL model will transform the data from $T\times80\times80$ (gray-scale screens) to $T\times5\times5\times32$ (CNN activations), $T\times D$ (hidden states), $T\times D$ (cell states), and $T\times m$ (action logits). We reduce the five sets of HD data to $T\times2$ through five tSNE projections [47] and present them in five juxtaposed views, each has a scatterplot and a bar-chart.

The **scatterplot** (Fig. 3-c2) shows the tSNE [47] projection result. Each point represents one game step and is colored based on the action of that step. A consistent color-mapping is used in this view and the Episode view (i.e., the points on the curve of Fig. 4b and the bar-chart in Fig. 4d). PCA can also be used here. However, for this problem, tSNE gives better results, and its non-linear dimensionality reduction power is also preferred by the experts. Semantic zoom is enabled in this view to address scalability challenges and mitigate the overlap among data points. Specifically, when zooming into a small region, the scatterplot will present the data points in that region only, with the same point size. As a result, overlapped points can be separated in the zoomed-in view. Also, this view is equipped with lasso selection to allow users to flexibly define the area of interest and select step clusters (please see our associated video for these interaction details). When a cluster of steps is selected, the game-replay view will show an *average screen* (e.g., Fig. 9e-f, R1.4). Each pixel of this screen is the average of the

corresponding pixels from the screens of the selected steps.

The **bar-chart** (Fig. 3-c1) presents the distribution of the distances (in the projected space) between points of consecutive steps. The horizontal axis represents the distance values and the vertical axis denotes the counts on a logarithmic scale. The red vertical line over this bar-chart specifies a threshold. Consecutive game steps with a distance less than it will be connected in the scatterplot to form a game segment. For example, the threshold is 12 in Fig. 5-a1, and points that have a distance to their previous point less than 12 are connected to their previous point (Fig. 5-a2). A small threshold results in many shorter segments, whereas a large one leads to fewer but longer segments. This function is to merge steps with smooth changes and highlight steps with sudden changes (i.e., the starting points of individual segments). The sudden changes often correspond to big state updates and deserve more attention (R1.3, R2.2).

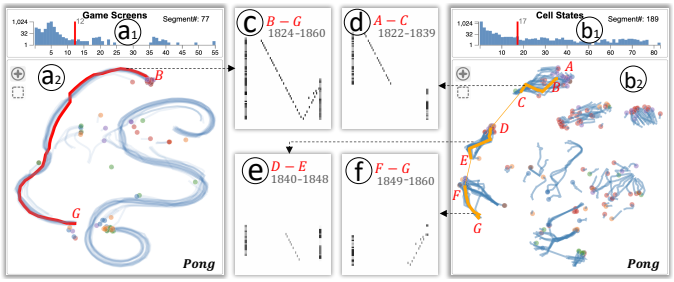


Fig. 5. Connecting consecutive game steps to form segments. The red segment in (a2) is in selection (step B-G). The corresponding steps in (b2) form three segments (step B-C, D-E, F-G) and they are highlighted to synchronize the selection. Clicking a segment will show the corresponding screens' variance map (c-f, the numbers show the step range).

Users can change the threshold by dragging the red line in the bar-chart and the segments in the scatterplot will be updated accordingly. Clicking on a segment will select the corresponding sequence of steps (highlighted in Fig. 5-a2), and the game-replay view will show a *variance map* for those steps' screens (Fig. 5c). Each pixel of this map is the variance of the corresponding pixels from the selected screens (darker colors correspond to larger variances). This map effectively "replays" the segment through a static image. All Projection views are synchronized, e.g., the selected segment in Fig. 5-a2 (game screen projection) is in three segments in Fig. 5-b2 (cell state projection) connected with the thinner orange lines. Fig. 5d-f show the corresponding variance maps.

Three optimizations have been applied to the Projection view. *First*, we break a sequence of game steps into segments with an overlapped sliding window, and project the segments (instead of steps) to stabilize the projection. In Fig. 3-c2, each point represents a sequence of consecutive game screens, i.e., $[x_{t-n}{:}x_t]$ (n=20). This optimization works well to stabilize the projections of x_t and s_t. However, it is not applied to the projections of hx_t, cx_t, or lgt_t, as they have already captured the temporal information through the RNN. *Second*, to reduce the tSNE projection cost for x_t and s_t (these two have much higher dimensionalities), we apply PCA on the data first to reduce their dimensionality to 256 before tSNE projections. *Third*, the tSNE result of x_t (the 2D coordinate of each step) is used to initialize the other four layouts. With this common initialization, the same steps in different views will be located at similar positions, making the explorations easier. For example, the highlighted points in Fig. 3c represent the same game steps, and they always appear in the bottom-left corner across all five projections.

5.3 Dimensions View

Analyzing the RNN hidden/cell states is the key to understanding the power of RNNs [6], [7], [8]. However, due to the high dimensionality of those states, which dimension(s) an analysis should start with often puzzles deep learning experts. The Dimension view prioritizes RNN state dimensions by soring them using aggregated metrics from (1) all game steps, (2) a single step, (3) or two subsets of steps. Our following explanations focus on hidden states (hx) with D dimensions. Cell states (cx) can be investigated similarly.

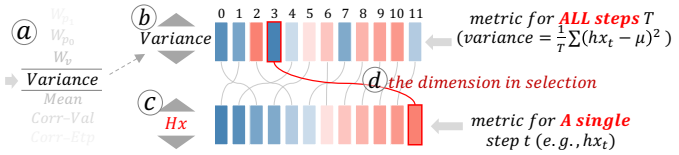


Fig. 6. Dimension view. (a) The variable-picker allows users to scroll up/down to switch among metrics. (b, c) Metrics aggregated from all steps and a single step are presented as two rows of rectangles (each rectangle for one dimension), the top/bottom row is unsorted/sorted. (d) The same dimensions across rows are linked with curves for tracking.

All-Step Metrics. The metrics aggregated from ***all game steps*** reflect the importance of RNN states over the entire game episode. They can be categorized into three groups. The ***first*** one includes the mean ($\mu[d]{=}\frac{1}{T}\sum_{t=1}^{T} hx_t[d]$) and variance ($\sigma^2[d]{=}\frac{1}{T}\sum_{t=1}^{T}(hx_t[d]{-}\mu[d])^2$) of each hx dimension d ($d{\in}[0, D{-}1]$). Sorting by these metrics identifies the most active hx dimensions, which are often of more interest. The ***second*** group is the model's weights for individual hx dimensions when deriving the *critic value* and *actor policy*, i.e., w_v and w_p in Sec. 3. Sorting by them is to prioritize the dimensions based on their contributions to the agent's actor or critic values, as dimensions with more contributions will have more controls on the agent's behaviors. The ***third*** group is the correlation between hx and the actor/critic values, i.e., $Cor(H(\pi(a_t|s_t)), hx_t[d])$ or $Cor(v_t, hx_t[d])$, where $H()$ computes the entropy and $Cor()$ measures the Pearson correlation between two value sequences across all T steps. The rationale for these metrics is that the hx dimensions correlated more with the actor/critic values will have stronger positive or negative impacts on the agent's behaviors.

We design a picker-wheel widget in the Dimension view to switch among different metrics, as shown in Fig. 6a. When hovering over the name of a metric from the widget, a tooltip will pop up with a brief description to explain the metric. The two triangles in Fig. 6b help to roll up/down the wheel and perform the metric-switching. For the selected metric, its D-dimensional value is presented as a row of D rectangles. Colors from blue over white to red reflect values from negative over 0 to positive (Fig. 3-b1, Fig. 6).

Single-Step Metrics. The metrics for ***a single step*** disclose the importance of hx when examining the model at a particular step. They include (1) the value of hx at the focused step t (hx_t), (2) the difference between hx_t and its previous step ($hx_t{-}hx_{t-1}$), and (3) the result of multiplying

hx_t with the actor/critic weights ($hx_t \cdot w_v$ or $hx_t \cdot w_p^i$). These metrics are presented as another row of rectangles (Fig. 6c).

Both rows of rectangles can be sorted to prioritize the dimensions (R2.1). For example, Fig. 6b shows the default status and the dimensions are arranged by their index, whereas Fig. 6c shows the dimensions sorted by hx_t. Rectangles for the same dimension across rows are linked, showing the order difference between the selected global and local metrics. From Fig. 6d, dimension 3 has the maximum hidden state at the selected step. Clicking the rectangle for this dimension will visualize the corresponding hidden state values across steps ($hx_t[d], d{=}3, t{\in}[1, T]$) as a curve in the Episode view. The two rows of rectangles can also be scaled horizontally (e.g., the bottom row in Fig. 3-b1), making this view applicable to a large number of dimensions.

Subset-Step Metric. The metric for ***two subsets of steps*** identifies dimensions behaving dissimilarly in the two subsets (e.g., steps with or without rewards). These dimensions are very likely to be more responsible for the corresponding behavior, and therefore are of more interest (R2.3). We define the metric as the JSD between the two distributions formed by the hx values from the two subsets of steps, i.e.,

$$JSD(Dist(hx_t[d], \forall t{\in}P) || Dist(hx_t[d], \forall t{\in}Q)), d{\in}[0, D{-}1]$$

where P and Q represent the two subsets, $Dist()$ is to form a distribution, and $JSD()$ computes the Jensen-Shannon Divergence. The above computation will be extended to all D dimensions, and the result will be appended as a new metric to the bottom row in Fig. 3-b1 for dimension sorting.

*DRL*IVE allows users to select two subsets of steps from a symmetric histogram (Fig. 3-b3). The two halves of the histogram show the value distribution for the same metric and the metric could be: critic value, actor policy entropy, or step reward (specified from the dropdown list in Fig. 3-b2). Users can brush on the top/bottom histogram to select steps with desired values. In Fig. 3-b3, two subsets with small (gray brush) and large (red brush) *critic values* are selected. The two pie charts on the right show the action distributions in the two subsets, and the values below them indicate the numbers of selected steps. After any selections, the JSD for the two subsets of hx values will be computed automatically and used to sort dimensions. Next, users can select dimensions with larger JDS by brushing the rectangles in Fig. 3-b1. The distribution pairs of the selected dimensions will be shown in Fig. 3-b4 as superimposed histograms. The histogram in gray/red corresponds to the distribution formed by the steps in the gray/red brush in Fig. 3-b3.

5.4 Interactive Perturbation

Beyond locating important steps (Episode and Projection view) and dimensions (Dimension view), DRL researchers also have the desire to directly interact with the model and validate the findings from the step- and dimension-oriented analysis. Interactive perturbation (R3) is proposed for this matter to diagnose the model. Our diagnosis is based on the method of Greydanus et al. [40], in which, the authors perturbed game screens (x_t in Fig. 2) to study how they affect DRL's *actor* and *critic* values. Some preliminary perturbation results on agents' memory were also presented by the authors. Our work extends their method and thoroughly examines the effects of individual RNN state dimensions.

Algorithm 1 Interpreting RNN states by pixel perturbation.

1: x_t, hx_{t-1}, cx_{t-1} // the game screen, previous hidden and cell state
2: $(hx_t, cx_t) = RNN(CNN(x_t), (hx_{t-1}, cx_{t-1}))$
3: **for** $i = 0;\ i < x_t.width;\ i+ = s$ **do**
4: **for** $j = 0;\ j < x_t.height;\ j+ = s$ **do**
5: $mask = gaussian\_mask(i, j, \sigma)$ //Gaussian centered at (i, j)
6: $x'_t = x_t * (1 - mask) + black\_img * mask$
7: $(hx'_t, cx'_t) = RNN(CNN(x'_t), (hx_{t-1}, cx_{t-1}))$
8: $Sh_t[i, j] = |hx'_t - hx_t|$ // saliency map with D channels
9: $Sc_t[i, j] = |cx'_t - cx_t|$ // saliency map with D channels
10: **end for**
11: **end for**

Algorithm 1 shows the details of our algorithm. *First*, we feed the screen x_t, the previous hidden (hx_{t-1}) and cell state (cx_{t-1}) to the DRL model to derive the hidden (hx_t) and cell state (cx_t) for the current step (line 2). *Then*, the screen x_t is interpolated with a black screen through a Gaussian mask centered at pixel (i, j) to generate a perturbed screen x'_t (line 6, Fig. 7). *Third*, feeding x'_t to the DRL model again, we get the perturbed hidden (hx'_t) and cell state (cx'_t) (line 7). *Lastly*, the difference between hx_t and hx'_t reflects the importance of pixel (i, j) (line 8). Extending the computation to all pixels (line 3-4), we get a saliency map for the hidden state at step t, denoted as Sh_t. Note that the map will have D channels, each for one dimension of the hidden state. For a single dimension, we can visualize the corresponding channel and overlay it on top of the screen (obs_t). For example, Fig. 4e shows the saliency map for hx dimension 251 at step 1858 ($Sh_{1858}[251]$), and we can clearly see that this low entropy step uses dimension 251 to capture the ball. Sc_t (saliency map for cell states) can be generated similarly (line 9).

There are two hyperparameters in Algorithm 1. One is the σ of the Gaussian mask (line 5), controlling how big the mask region should be. Another is the stride factor s when iterating through all pixels. We set both to 5, but they can be adjusted for the performance-accuracy trade-off.

*DRL*IVE empowers users to flexibly interact with the saliency map from a single game step in two ways. *First*, they can check the full saliency map at the step for a selected hx/cx dimension. Specifically, when changing the mode of the Episode view from "Animation" to "Saliency" (Fig. 3-a5), the game-replay view (Fig. 3-a2) will show the saliency map for the selected hx/cx dimension overlaid on top of the regular game screen, e.g., Fig. 4e. *Second*, users can also perturb a single pixel across all hx/cx dimensions. Specifically, clicking any pixel of the screen in Fig. 3-a2 will perturb the corresponding pixel, when the "Pixel Perturbation" function (Fig. 3-a6) is enabled. The users can then sort hx/cx dimensions based on their changes to the perturbation to identify the most related ones. For example, perturbing the pixels around the ball and sorting hx/cx dimensions based on this perturbation can identify dimensions that are more sensitive to the ball's positions, from which, we can easily discover the dimensions tracking the ball. Please check our associated video for these interaction details.

6 Case Studies with Domain Experts

We worked with the same 5 deep learning experts to explore the power of *DRL*IVE through multiple case studies. The studies were conducted following the protocol of guided-exploration with think-aloud discussions. Specifically, we

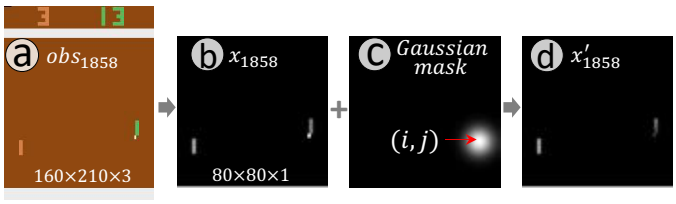


Fig. 7. A game screen (a) is converted to a gray-scale image (b, CNN input), and interpolated with a black image through a Gaussian mask (c). The pixels for the ball are masked out in the perturbation result (d).

first explained the design goals derived from the task analysis and how individual visualization components served these goals. Next, we walked the experts through the cases in Sec. 6.1∼6.3, guided them to explore the system, think aloud on the insight/findings, and discuss the potential merits/limitations. Finally, we conducted open-ended interviews to collect their feedback and suggestions (Sec. 6.4).

We explored two Atari games, Pong and Breakout, which are the two most frequently used testbeds in DRL. However, *DRL*IVE can be easily extended to other Atari games, and we include more results in our supplementary material.

- ***Atari Pong*** (Fig. 1) simulates the Ping-Pong game between two players. One (the environment) controls the paddle on the left; the other (the DRL agent) controls the paddle on the right. The player loses one point when he fails to catch the ball and gets one point when he makes the opponent fail to catch the ball. The game ends whenever one player gets 21 points. The six possible agent actions in this game are shown in Fig. 4d.
- In ***Breakout*** (Fig. 1), the agent controls the paddle to catch the ball and destroy the six rows of bricks. Destroying a brick from the bottom/middle/top two rows will get 1/4/7 point(s) respectively. However, failing to catch the ball will result in a life-loss.

Our explorations were carried out in a top-down manner. First, we used *DRL*IVE to get an overview of a game episode and reveal different playing strategies adopted by the agent (Sec. 6.1). Then, by opening the black-box of DRL models, we externalized the roles that different RNN cells played in different playing strategies (Sec. 6.2). Lastly, drilling down to particular game steps, we validated the functionality of critical RNN cells through interactive perturbations (Sec. 6.3).

6.1 Episode Exploration and Agent's Strategies (R1)

To study a trained DRL agent, DNN researchers often start by checking how smart it is when interacting with the game environment. For example, the followings are some typical questions that the experts are interested in. "Is the agent repetitively using similar movement patterns (i.e., playing strategies) to get rewards?" "How many strategies the agent has learned, and what strategy is used in different game scenarios?" The strategies reveal how good the model is and what the agent is capable of, which are critical details to understand the agent's strengths and weaknesses. In the following exploration case, we show how the experts can use the Episode and Projection view of *DRL*IVE to accomplish the goal of systematically overviewing all playing strategies and investigating detailed agent behaviors on-demand.

From the Episode view in Fig. 4a, the periodical patterns (R1.1) of the *critic values* (v_t) are easily observable. 21 peaks and 3 troughs were identified, corresponding to the steps where the agent obtained and lost rewards. The curve patterns also gave us hints to zoom into certain game steps of interest. For example, a randomly picked reward cycle (step 1770-1885) is shown in Fig. 4b (R1.3). We can see the *value* curve (in gray) kept increasing and a sudden drop appeared right after the agent got a reward at step 1881. The *entropy* curve (in blue) kept high except step 1844-1858, i.e., when the ball flew towards the right paddle. In other words, the agent learned that it only needs to carefully choose actions at these game steps to bounce the ball and get rewards (actions for other steps have little impact). For example, at step 1802 (Fig. 4c-d), the ball was flying towards the left and the probabilities for different actions were almost the same (i.e., similar to a random guess). The most critical 10 steps (1849-1858) are highlighted in Fig. 4b (in shade), where the agent was confident in issuing *noop* (in blue) and very confident in issuing $fire$ actions (in orange). The step with 100% confidence happened exactly when the ball hit the paddle (Fig. 4e-f), reflecting the agent's intelligence in controlling the paddle. This "kill-shot" brought the agent a reward at step 1881 eventually. As demonstrated, the design of our scalable line-chart provided an effective overview of the long episode and the game-replay view brought back the detailed game contexts on-demand.

Apart from the above "ad-hoc" exploration, *DRL*IVE can also provide an effective overview of all playing strategies (R1.4). This analysis can be started by focusing on steps with low entropies, i.e., the steps that the agent confidently controlled, and they can be filtered out by the thresholding line in the Episode view. For example, the steps with entropy less than the red thresholding line in Fig. 3-a1 are highlighted in the five scatterplots in Fig. 3c. Fig. 8a shows an enlarged version of the scatterplot for the hidden states with four clusters. The cluster in Fig. 8-a1 is the biggest. We randomly picked one point from this cluster (step 1960) and the corresponding screen is shown in Fig. 8c. From the Episode view, we selected a sequence of steps surrounding this step to show the game context (step 1902-1985). The variance map for these steps is shown in Fig. 8b, which reveals the trajectory of the ball, i.e., *A-B-C-D-E-F-G*. Step 1960 happened at point E, where the agent issued a $fire$ action to kick the ball towards F and eventually made the opponent fail to catch it at point G. The kill-shot step we analyzed before in Fig. 4 is also in this cluster. The Projection view saved the experts from further analyzing more steps with similar playing strategies individually (R1.2).

During this exploration, we also noticed the two sub-clusters in Fig. 8-a1 and wanted to know the subtle differ-

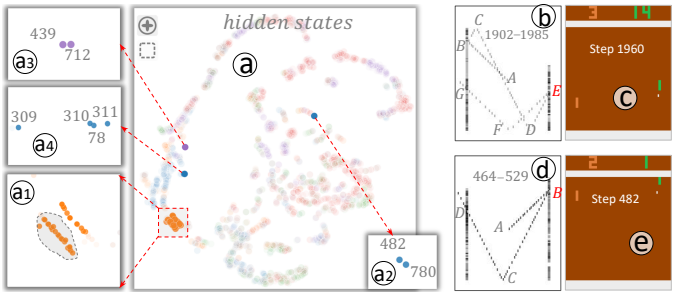


Fig. 8. Steps with low entropies are in four clusters (a1-a4). (b, c) and (d, e) show the typical playing strategies in cluster a1 and a2 respectively.

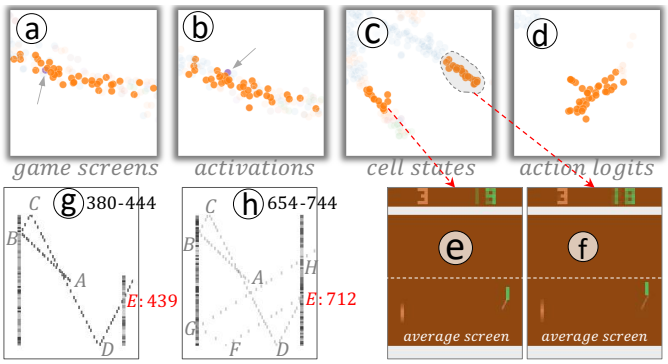


Fig. 9. (a-d) The other four projections for the cluster in Fig. 8-a1. (e, f) The average screens for the two sub-clusters of steps. (g, h) The playing strategies for the two steps in Fig. 8-a3, i.e., the two failed cases.

ence between them. The zoomed-in views for this cluster in the other four projections are shown in Fig. 9a-d (R1.2). The steps only diverged in the hidden and cell state projections (Fig. 8-a1, 9c), indicating the game screens, CNN activations, and action logits were similar, but the agent processed them differently inside the model. Through synchronized lasso selections, we found the bottom cluster in Fig. 8-a1 corresponded to the right cluster in Fig. 9c. The average screens for the two sub-clusters are shown in Fig. 9e-f (R1.4). Both the left paddle and the ball had different positions, as reflected by the corresponding blurry regions. In contrast, the right paddle did not move in individual sub-clusters, but its position differed in the two average screens. A white dashed line is added (on top of the two vertically aligned average screens) to show this subtle difference, which reflects the minor difference between two playing strategies. One is to use the front face of the right paddle to bounce the ball at a lower position (Fig. 9e), the other is to use the bottom face of the paddle to bounce the ball at a relatively higher position (Fig. 9f). This marginal difference is barely visible if only investigating the game screens without exposing the details of the internal RNN hidden/cell states.

Fig. 8-a2 shows another playing strategy. Different from the case in Fig. 8b, the ball flew to the right paddle initially (Fig. 8d, A-B). We picked one of the two blue steps (*noop* actions) for analysis, i.e., step 482 (Fig. 8e). The variance map for a sequence of context steps is shown in Fig. 8d, in which, the ball follows the trajectory A-B-C-D, and point B is step 482. At this step, the agent issued *noop* to wait for the ball coming toward the right paddle. The ball then bounced at point C and the opponent failed to catch it at point D.

The two purple steps in Fig. 8-a3 (step 439, 712) have similar game screens with the steps in Fig. 8-a1, as the purple and orange points are mixed in the game screen and CNN activation projections in Fig. 9a-b. However, they are the failed cases of the agent. At step 439 (Fig. 9g), the agent failed to catch the ball at point E. At step 712 (Fig. 9h), the agent caught the ball, but the angle/speed of the ball was not good enough for a kill-shot, and it was not ready for the ball bounced back from point G. Eventually, it lost the ball at point H. The cases in Fig. 8-a4 and more cases can be analyzed similarly. From these explorations, the Projection view, the *average* and *variance* maps successfully identified and replayed the critical game steps, especially the failure cases. Knowing these details is the starting point for the experts to diagnose the failed cases. Moreover, focusing on the identified critical steps, one can use the Dimension view to prioritize the RNN internal cells and examine the important ones at these steps (details in Sec. 6.2).

The playing strategies in Breakout can also be easily discovered with *DRLIVE*. Replaying the steps around the obvious increasing point of the *critic value* curve from the Episode view (shown later in Fig. 12b, step 2515), we noticed the agent learned the trick to send the ball through bricks and keep bouncing between the top boundary and top rows of bricks to get high rewards frequently. The guidance from the curve was very helpful to identify this important moment (R1.3). Additionally, by comparing the segments in the cell state projection (Fig. 10a), two strategies at the early and later game stages were clearly revealed (R1.2, R1.4). At the early stage (Fig. 10b, variance map for step 242-571), the agent used the side walls to bounce the ball and hit lower layer bricks. The ball traveled a longer path, e.g., Fig. 10b A-B-C. At the later stage (Fig. 10c, variance map for step 2337-2601), the agent frequently moved the ball between point A and B to directly hit bricks around point C, and many bricks from the higher layers got destroyed.

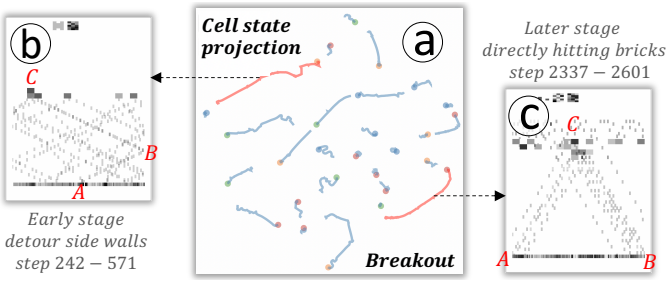


Fig. 10. The cell state projection (a) and two different playing strategies: (b) using side walls to hit lower-layer bricks with longer paths (e.g., A-B-C); (c) directly hitting bricks with shorter paths (e.g., A-C or B-C).

6.2 Hidden/Cell State Investigation (R2)

With an overview of the game episode and various playing strategies, our next goal is to reveal what RNN hidden/cell states control those strategies. For example, does the agent really notice the difference between different game states (e.g., different ball directions)? Are there certain hx/cx dimensions tracking these states and informing the agent? The purpose of finding and investigating these hx/cx dimensions is to understand how the model works internally, and diagnose if the abnormal behaviors of these dimensions contribute to the failure cases or not.

As explained in Sec. 5.3, we can identify the important hx/cx dimensions by prioritizing them in three levels.

***First**, we can prioritize RNN state dimensions using metrics aggregated from **all steps**.* For example, sorting the top row of rectangles in the Dimension view using w_v, we identified important hx dimensions for rewards. The dimension with the largest w_v was 185 (see the orange curve of $hx_t[185]$, $t \in [1, T]$, in Fig. 11a). Most values of this dimension were close to 0, except steps where the agent gained or lost rewards. As a result, it flagged the reward-related steps and overlapped with the reward curve (in green), except step 529 and 824. From the entropy curve (blue in Fig. 11c), we found

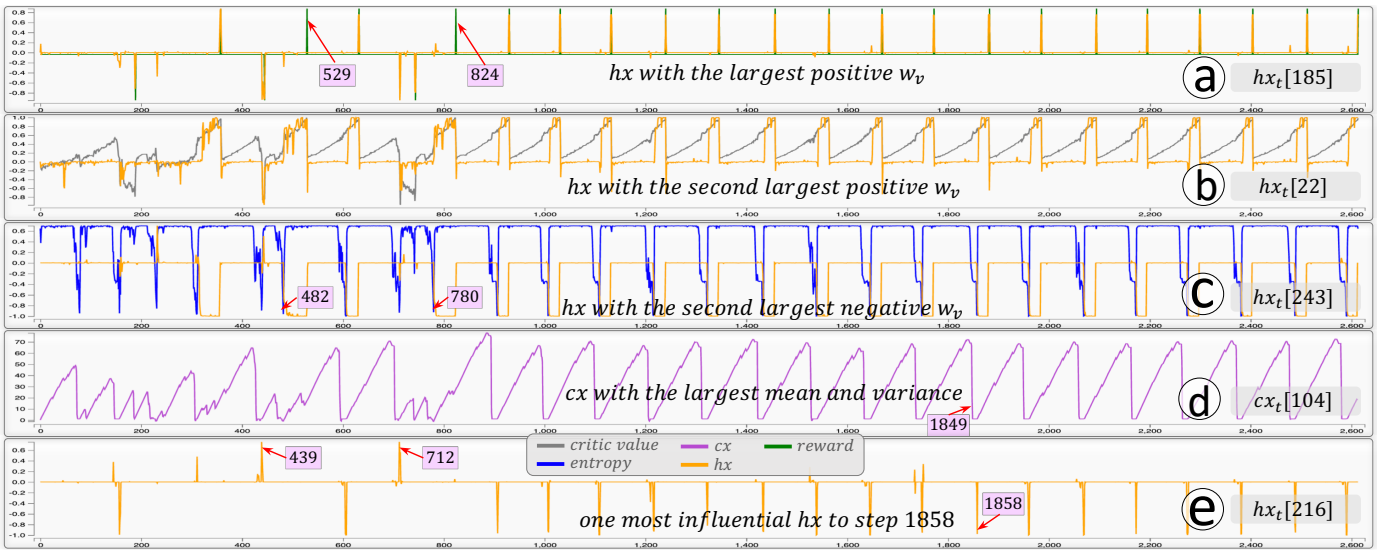


Fig. 11. (a, b, c) Three hidden state dimensions with important impacts on the *critic* values are identified after sorting the dimensions by w_v. (d) A cell state dimension (a step-counter) is identified by the aggregated mean and variance value. (e) The hidden state dimension that is crucial to the agent's behavior at step 1858. These dimensions control the agent's behaviors and show periodical patterns synchronized with the reward cycles.

steps with the most confident actions right before these two steps were step 482 and 780 respectively. The agent used the playing strategy in Fig. 8-a2, 8d-e in these steps, indicating that $hx_t[185]$ might not be aware of the changes to the *critic values* introduced by this particular strategy.

Dimension 22 (i.e., $hx_t[22]$) had the second-largest w_v. It activated between the step of the smallest entropy and the reward step (Fig. 11b, orange), making the *critic values* in this range large (R2.1). In contrast, dimension 243 (Fig. 11c, orange) had the large negative w_v. It was negatively correlated with dimension 22, but had the same effect after being multiplied with its negative weight w_v. Sorting hx dimensions by their correlations with the *critic values* verified the importance of these two dimensions, as they had the largest positive- and the second-largest negative-correlations with the *value* curve. Consequently, if any abnormal behaviors were observed from the *value* loss ($\mathcal{L}_{value}$), these two dimensions could be examined first.

Soring hx dimensions by their variance helped us identify many dimensions that learned nothing over the training. For example, $hx_t[200]$, $hx_t[47]$, and $hx_t[156]$ had very small variance (around 0) across the entire episode, indicating a higher-than-needed dimensionality (D=256) was used. This observation also triggered the domain experts to improve the model by compressing/pruning it (i.e., removing these redundant dimensions and further fine-tuning the model).

Sorting cx dimensions by their mean, we found dimension 104 with the maximum value. It (i.e., $cx_t[104]$) worked like a step-counter (Fig. 11d), which increased $\sim$1 every single step and got reset every reward cycle. This revealed a possible reason on how the agent got to know the beginning of every periodical cycle. Moreover, we were curious about the consequence of disabling this step-counter dimension and further experiments would be carried out.

***Second**, we can investigate what dimensions have played important roles at **a single step** of interest.* For example, what dimension(s) contributed the most to the $fire$ action at step 1858, i.e., the most confident "kill-shot" step in Fig. 4e? As the action index of $fire$ is 1, we sorted hx dimensions by $hx_{1858}{\cdot}w_p^1$ (the bottom row of Fig. 3-b1). The top-5 most important dimensions were 57, 142, 164, 216, 85, and we further checked them individually in the Episode view. For example, $hx_t[216]$ (Fig. 11e) had a large negative value at step 1858. However, since $w_p^1[216]$ was also large and negative (tracing the Bézier curves to see the sorting of w_p^1 in the top row of Fig. 3-b1), their multiplication became positive, making $hx_{1858}{\cdot}w_p^1$ large and contribute positively to the predicted $fire$ action. Moreover, the large negative value of $hx_t[216]$ appeared almost every "kill-shot" step, indicating the agent repetitively used the same trick internally.

We also noticed that $hx_t[216]$ had irregularly large positive values at step 439 and 712 (Fig. 11e), i.e., the failed cases in Fig. 8-a3. These positive values, multiplying the negative $w_p^1[216]$, contributed negatively to $hx_{439}{\cdot}w_p^1$ and $hx_{712}{\cdot}w_p^1$, and made the probability of issuing $fire$ smaller, which were possible reasons for the two failed cases explained in Fig. 9g-h. These cases might be avoidable, if $hx_t[216]$ had large negative values, like other "kill-shot" steps. The interesting observation provided a starting point to diagnose the model, e.g., modifying the value of $hx_t[216]$ at these steps to check the agent's behaviors.

Steps with sudden state changes also deserve more attention (R2.1). For instance, after selecting the segment in Fig. 5-a2 (a sequence of steps with smooth game screen changes), we found it broke into three segments in the cell state projection (Fig. 5-b2). The break at point F indicated that the agent noticed the change in the ball's moving direction (Fig. 5e-f). Focused on point F (step 1849), we sorted cx dimensions based on the difference with their previous step ($cx_{1849}[d]{-}cx_{1848}[d]$), and the biggest change came from dimension 104. As explained in Fig. 11d, this dimension is a step-counter and step 1849 is where the counter gets reset.

***Third**, we can discern dimensions that behave differently in **two subsets of steps** (R2.3).* For example, to identify dimensions affecting the *critic values*, we selected two subsets of steps with small (gray brush) and large (red brush) *critic val-*

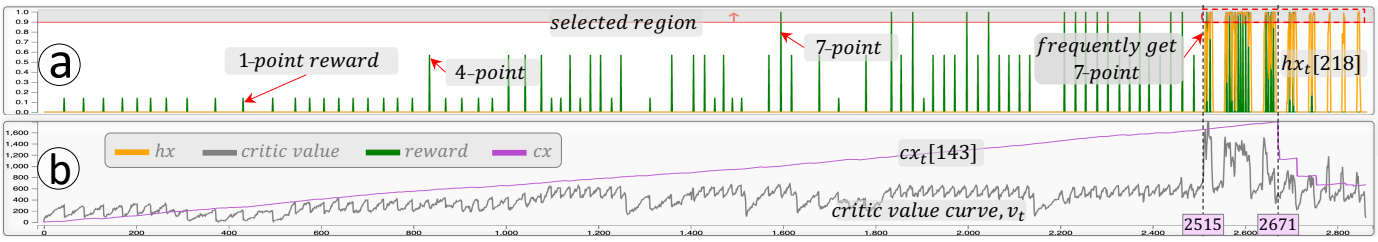


Fig. 12. Interpretable dimensions from the Breakout: (a) $hx_t[218]$ gets activated (the large value of the yellow curve on the right) only when the ball is above all bricks (steps in the red dashed rectangle); (b) $cx_t[143]$ is a step-counter, which increases a constant amount every step until step 2671.

ues (Fig. 3-b3). Dimension 22 was identified again, as it had the largest JSD value (Fig. 3-b4). This observation echoed our previous findings that $hx_t[22]$ contributed positively to the *critic values* (Fig. 11b), and the exploration deepened the experts' understanding on the model.

Similar analysis also applies to the Breakout game and we present three interpretable dimensions identified over our explorations with the experts in the following.

First, sorting hx dimensions based on their correlation with the *entropy* curve, we found $hx_t[19]$ had the fifth largest correlation value and it roughly captured the ball's moving direction (R2.2). Fig. 13a-b show a zoomed-in segment of $hx_t[19]$ ($2303 \leq t \leq 2349$) and the corresponding variance map. We cut the segment into 4 stages. At stage 1, the ball went towards the paddle (Fig. 13b, A-B) and $hx_t[19]$ was around 0. At stage 2, the ball was bounced back towards the bricks (Fig. 13b, B-A) and $hx_t[19]$ was around 1. The ball followed the trajectory A-C-D and D-E at stage 3 and 4 respectively. It seemed this dimension kept informing the agent with the ball's moving direction. When the ball flies towards the paddle, the entropy values are usually small, as the agent needs on-purpose moves to catch the ball. In contrast, when the ball flies away from the paddle, the entropy values are often large. $hx_t[19]$ correlated with the entropy, thus roughly captured the ball's moving direction.

Second, sorting hx dimensions based on their correlation with *critic values*, we found $hx_t[218]$ had the second-largest positive correlation and it activated whenever the ball was above all bricks (R2.2). As shown in Fig. 12a (orange curve), $hx_t[218]$ kept to be 0, until step 2515. Replaying the game, we found this step was where the agent sent the ball through the bricks, so that it could bounce between the top boundary and bricks to get frequent rewards. The frequent rewards led to high *critic values*. Therefore, dimensions correlated with *critic values* could capture this moment. To verify this, we used the thresholding line in Fig. 12a to select steps with $hx_t[218]$ value larger than the threshold (in the red dashed rectangle). The variance map for these steps (Fig. 13c) verified that lots of the top-row bricks were destroyed, and the ball was always above the bricks.

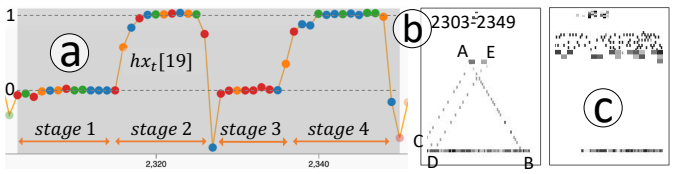


Fig. 13. (a) The curve for $hx_t[19]$ of the Breakout agent. (b) The corresponding variance map of the steps in (a). $hx_t[19]$ roughly captures the ball's moving direction. (c) The variance map of steps with large $hx_t[218]$ values. The ball is always above all bricks at these steps.

Third, sorting cx dimensions identified $cx_t[143]$, i.e., the dimension with the largest mean and variance (Fig. 12b). The value of this dimension increased a constant amount every step until step 2671, i.e., another step-counter (R2.1, R2.2). It is different from the step-counter in the Pong game (Fig. 11d), as it does not reset periodically. The reason is probably that the game scene of the Breakout game keeps changing (as more and more bricks get destroyed), and there is no reappeared scene to form periodical cycles.

6.3 Interactive Perturbation (R3)

After finding the RNN state dimensions that in charge of different functions, our next goal is to validate their functionalities and shed some light on how they have worked, e.g., how could $hx_t[185]$ (Fig. 11a) notice the reward gain/loss?

From the saliency map of dimension 185 ($Sh_t[185]$), we found it became salient only at the end of each game cycle. Checking different game steps, as the two examples shown in Fig. 14a-b, we found the saliency map highlighted the ball only when it was on the left of the left paddle (reward-gain) or on the right of the right paddle (reward-loss). This reveals that the reward-gain/loss steps could be identified using the relative position between the ball and paddles. Considering this function only, the above relative positions could be encoded with 3 numerical values (indicating if the ball is on the left/middle/right of the two paddles) and the use of a large CNN might be avoided. Whether this would work requires further investigations, but the insight here directly leads to explorations of new model structures.

However, why $hx_t[185]$ did not capture the reward gain at step 529 and 824 (Fig. 11a)? Nothing was highlighted from the saliency map of these two steps (Fig. 14c-d). As explained earlier, the agent used a different strategy at these steps and the ball left the scene at a higher position (point D in Fig. 8d), which might be out of the region that dimension 185 could monitor. However, this does not mean that the agent was not aware of them. By interactively perturbing the ball pixels, and sorting hx dimensions by the perturbation, we found dimension 217 and 22 were the most sensitive ones at step 529 and 824 respectively (Fig. 14e-f). They, like extra "eyes" of the agent, worked in complementary with dimension 185 to recognize reward-gain/loss.

We also found dimensions with similar functions. For example, dimension 186 tracked the ball in most of the steps, except step 1833 (Fig. 14, the middle row of images). By perturbing the ball pixels at this step, we found dimension 230 was the most sensitive one. Checking the saliency map of dimension 230 from the surrounding steps, we found it

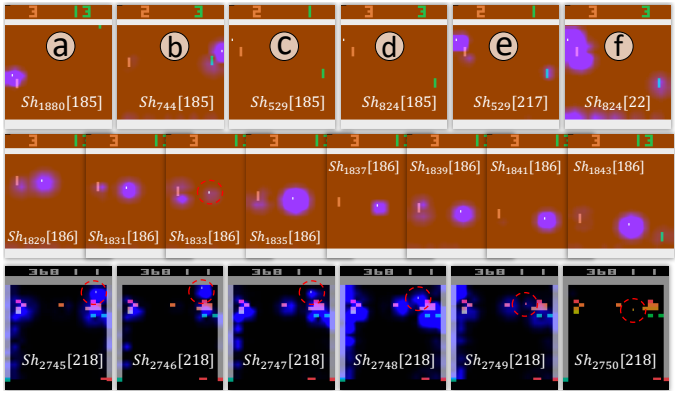


Fig. 14. Diagnosing hx dimensions with pixel perturbations: (top) dimensions flagging the reward-gain/loss, (middle) dimension tracking the ball, (bottom) dimension monitoring if the ball is above the bricks or not.

also closely monitored the ball. The experts explained that since no constraint was enforced to "regularize" what individual dimensions should learn, the functions of different state dimensions could get entangled very easily.

We also conducted similar experiments with the Breakout game. For example, the bottom row of images in Fig. 14 shows the saliency maps when diagnosing $hx_t[218]$ (the dimension that gets activated whenever the ball is above all bricks). The first four frames show that the agent closely tracked the ball when it was above the bricks. However, the ball was not identified in the rightmost two saliency maps, when it went into the bricks. The observation is consistent with what we have observed from Fig. 12a and Fig. 13c.

6.4 Domain Experts' Feedback

This section summarizes the feedback from the five experts ($E1{\sim}E5$) during our final interviews. Among the three major components of *DRL*IVE, the Episode view is the most intuitive one and the "overview+details" exploration of long game episodes can be easily adopted by the experts. The Projection views and their synchronized interactions reveal the power of coordinated visualizations, which the experts can hardly do with their favorite Python code. Additionally, the interactive perturbation is a very enlightening function to the experts, which deepens their understanding of DRL.

For the functionality of individual hx/cx dimensions, $E1$ and $E2$ echoed our findings on the step-counters, as they had "*similar observations on the cell state*" of their RNN-based classification models. However, they only found them through random explorations. $E2$ was surprised to see that "*simply sorting by mean could be an effective way to identify them*". $E1$ explained, "*as the cell state values are not bounded, the step-counter dimension will quickly distinguish from others as its value keeps increasing*". The findings on the functionality of RNN states appeased the curiosity of the experts and met their goals in identifying interpretable RNN cells (R2).

$E1$, $E3$, and $E5$ discussed the impact of RNNs' model architectures. First, based on their experience on profiling the size of hx dimensions, 128, 256, and 512 might not show fundamental difference in certain applications. "*As sorting by variance can easily identify redundant dimensions, it could be a good metric for RNN model-pruning [48]*", commented by $E1$. Second, $E3$ and $E5$ liked the metric of computing dimensions' correlation with the *value* and *entropy* curves, as it reflects the *critic* and *actor* part of the DRL respectively. $E5$ commented that "*it is not surprising to see dimensions with entangled functions, as there is no restriction to regularize them*" in the loss function. Third, $E1$ also had experience with stacked RNNs. He commented that "*hx and cx from higher RNN layers are usually more chaos*", and he wanted to use *DRL*IVE to further investigate stacked RNNs.

Both $E4$ and $E5$ commented that "*the interactive perturbation is very enlightening*". $E4$ asked "*what if we perturb the model in early training stages?*" and he suspected that the highlighted regions in the saliency map would be scattered everywhere. Therefore, tracking a model's ability in concentrating on certain pixel regions could be a potential way to monitor the model's quality. $E5$ said "*the interactive perturbation is very helpful to understand the CNN/RNN by relating the input game screens with the internal model states*". He also mentioned that people often prefer tree-based models due to their interpretability. "*The demonstrated perturbation method helped a lot to interpret the decision-making process, which would definitely promote the adoption of DNNs in production*". These comments reflect the effectiveness of our method to interactively diagnose the model and verify its functionality (R3).

The experts also mentioned some desired features and suggestions to improve *DRL*IVE. First, the Projection view did not show very obvious cluster patterns, which was a little out of expectation. $E3$ suggested using hierarchical clustering on individual steps to reveal the cluster patterns. He also suggested integrating some popular motif/discord mining algorithms into the process of identifying critical steps. Second, $E5$ suggested providing some guidance when selecting the metrics for sorting in the Dimension view. For example, to find reward-related dimensions, one should sort the dimensions by w_v or their correlation to the *critic values*.

7 Discussion

Although we have noticed that using 256 hx/cx dimensions is more than needed in our case, some models may use an even higher dimensionality. This brings a potential **scalability** challenge to the Dimension view. However, note that, after sorting by different metrics, we only need to explore a few prioritized dimensions. Therefore, *DRL*IVE should scale well to RNNs with more state dimensions. Furthermore, the capability of zooming in/out of the two rows of rectangles also helps to mitigate the scalability problem.

*DRL*IVE has good **generalizability** to other Atari games since we designed it without any game-specific features. To show this and enable more cross-game comparisons, we include the results from several other Atari games in our supplementary material. Second, as *DRL*IVE always takes the common forms of RNN-based DRL data as inputs (e.g., game screens, reward sequences, RNN hidden/cell states), it is context-agnostic and can be easily adapted to visualize other DRL testbeds with different CNN and/or RNN models, e.g., ViZDoom [38]. Lastly, apart from DRLs, we can further extend our approach to other RNN-based models, e.g., RNN-based classifiers. Most of the metrics introduced here can be reused to prioritize the RNN dimensions and discover interpretable cells. For example, as mentioned by the experts, similar step-counter dimensions have also been observed from their RNN-based predictive models.

Visualization lessons learned from *DRL*IVE. From this work, we first learned that dimensionality reduction algorithms (tSNE in our case) can be easily adapted for the analysis and visualization of time-series data. The data of individual time steps can be connected in the projection space to show the overall temporal trend, and the distance between consecutive steps is a good indicator of abnormal events. Second, from the visualization perspective, we believe the presented average screens (e.g., Fig. 9e-f), variance maps (e.g., Fig. 9g-h), and saliency maps (Fig. 14) are effective for the analysis of other image-based DNN models, where the temporal sequences of input screens/images often carry semantic information for the interpretation of the corresponding models. Lastly, we want to further emphasize the importance of closely collaborating with domain scientists when working on domain-specific visualization problems. In our case, the deep learning experts contributed a lot in the initial task analysis process and provided concrete feedback for us to iteratively improve the visualizations.

How useful are the findings? We believe the discoveries from our work are valuable to RL researchers from the following perspectives. First, they can help to *interpret* the evolution of DRL models. For example, from Fig. 8, we can easily overview different playing strategies adopted by the agent. From Fig. 10, we can effectively disclose the evolution of the strategies and explain why the agent becomes more and more intelligent. Second, knowing the functionality of individual RNN states can help to *diagnose* a DRL model, especially the failure cases of the model. For instance, as noticed in Fig. 11d and Fig. 12b, different agents always use a cell state dimension to count steps. In the failure cases, we can immediately check the corresponding cell state (at the failed step) to see if the counter increases or not and verify whether or not the step has been properly counted. Also, as shown in Fig. 5, a big environmental change (e.g., inversed ball direction) often leads to discontinuities in the projected cell state segments. Thus, we can use the Projection view to examine if a failure case is caused by the unawareness of the big change. Third, knowing how the DRL agents memorize different game states can potentially help to *improve* DRL models. For example, when the ball in Breakout keeps bouncing between the top boundary and bricks, the agent will receive rewards frequently. As shown in Fig. 12a and Fig. 13c, the agent uses a hidden state dimension to flag the occurrence of this pattern. A potential change to the model would be incorporating the behavior of this hidden state dimension into the value loss to reward this good pattern and encourage its occurrence.

8 Limitations and Future Work

One limitation of our work is our limited ways of finding the interpretable dimensions. We agree that some interpretable dimensions can be identified in a more efficient way through game-specific features. For example, we could identify dimensions tracking the ball through advanced image processing algorithms. However, the approach only applies to limited games, and will undermine the generalizability of *DRL*IVE. We, therefore, did not pursue that direction.

Second, our reported findings are mostly game-specific, as we always need a specific game context to describe the visualization results. Also, due to the inherent randomness of DNN training, the function of the same RNN state can be different from different trainings. For example, we may see another dimension (instead of $cx_t[143]$ in Fig. 12b) working as the step-counter in Breakout. Therefore, it is of little value to memorize which dimensions were responsible for what in a specific game. Being able to efficiently identify the functional dimensions and identify them across games means more, which is the exact goal of our work. Comparing the cross-game findings and their commonality will shed light on how the underlying model has really worked.

Third, the complexity of our system also raises concerns about its usability. During our studies with the experts, they wanted to have more detailed instructions when conducting some of the analysis. For example, one expert pointed out that the Dimension view lacked guidance for him to select relevant metrics for dimension sorting. This may elongate the exploration process, as users have to explore the prioritized dimensions from all metrics. In the future, we plan to better organize those metrics and suggest relevant ones based on users' investigation focus. We also plan to include more system tooltips and guiding tours to help users get familiar with our system more easily.

Lastly, our current evaluation of *DRL*IVE is limited to the case studies with expert users. Formal user studies with statistical tests on different hypotheses (e.g., can interactive perturbation increase users' confidence in interpreting DRL models?) would more rigorously prove the efficacy and usefulness of our system. We plan to conduct these studies with more users in the future.

9 Conclusion

In this work, we propose *DRL*IVE, a visual analytics system to effectively explore, interpret, and diagnose DRL models. *DRL*IVE is equipped with three coordinated views, targeting to meet three practical needs of game episode exploration, RNN state interpretation, and interactive model perturbation. Through interactive visual analysis, *DRL*IVE enables users to flexibly explore game episodes, prioritize RNN state dimensions by sorting them with different metrics, and verify their functionality by perturbing the input game screens. Concrete case studies, conducted together with multiple deep learning experts, validated the efficacy of *DRL*IVE.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] 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, pp. 529–533, 2015.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[5] 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 *International conference on machine learning*, 2016, pp. 1928–1937.

[6] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 667–676, 2017.

[7] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding hidden memories of recurrent neural networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2017, pp. 13–24.

[8] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.

[9] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush, "Seq2seq-vis: A visual debugging tool for sequence-to-sequence models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 353–363, 2018.

[10] A. Madsen, "Visualizing memorization in rnns," *Distill*, 2019, https://distill.pub/2019/memorization-in-rnns.

[11] D. Cashman, G. Patterson, A. Mosca, N. Watts, S. Robinson, and R. Chang, "Rnnbow: Visualizing learning via backpropagation gradients in rnns," *IEEE Computer Graphics and Applications*, vol. 38, no. 6, pp. 39–50, 2018.

[12] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 8, pp. 2674–2693, 2018.

[13] Q. Wang, J. Yuan, S. Chen, H. Su, H. Qu, and S. Liu, "Visual genealogy of deep neural networks," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[14] L. Jiang, S. Liu, and C. Chen, "Recent research advances on interactive machine learning," *Journal of Visualization*, vol. 22, no. 2, pp. 401–417, 2019.

[15] J. Choo and S. Liu, "Visual analytics for explainable deep learning," *IEEE Computer Graphics and Applications*, vol. 38, no. 4, pp. 84–92, 2018.

[16] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 101–110, 2016.

[17] S. Liu, X. Wang, M. Liu, and J. Zhu, "Towards better analysis of machine learning models: A visual analytics perspective," *Visual Informatics*, vol. 1, no. 1, pp. 48–56, 2017.

[18] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg, "Visualizing dataflow graphs of deep learning models in tensorflow," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 1–12, 2017.

[19] J. Wang, W. Zhang, and H. Yang, "Scanviz: Interpreting the symbol-concept association captured by deep neural networks through visual analytics," in *2020 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2020, pp. 51–60.

[20] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert, "Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 364–373, 2018.

[21] M. Liu, S. Liu, H. Su, K. Cao, and J. Zhu, "Analyzing the noise robustness of deep neural networks," in *IEEE Conference on Visual Analytics Science and Technology (VAST)*, 2018, pp. 60–71.

[22] Y. Ma, T. Xie, J. Li, and R. Maciejewski, "Explaining vulnerabilities to adversarial machine learning through visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 1075–1085, 2019.

[23] L. Gou, L. Zou, N. Li, M. Hofmann, A. K. Shekar, A. Wendt, and L. Ren, "VATLD: a visual analytics system to assess, understand and improve traffic light detection," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 261–271, 2021.

[24] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, and S. Liu, "A survey of visual analytics techniques for machine learning," *Computational Visual Media*, pp. 1–34, 2020.

[25] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91–100, 2016.

[26] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, "ActiVis: Visual exploration of industry-scale deep neural network models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 88–97, 2017.

[27] Y. Ming, H. Qu, and E. Bertini, "RuleMatrix: Visualizing and understanding classifiers with rules," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 342–352, 2018.

[28] J. Wang, L. Gou, W. Zhang, H. Yang, and H.-W. Shen, "DeepVID: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 6, pp. 2168–2180, 2019.

[29] A. Bilal, A. Jourabloo, M. Ye, X. Liu, and L. Ren, "Do convolutional neural networks learn class hierarchy?" *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 152–162, 2017.

[30] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova, "DeepEyes: Progressive visual analytics for designing deep neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 98–108, 2017.

[31] G. Li, J. Wang, H.-W. Shen, K. Chen, G. Shan, and Z. Lu, "CNNPruner: Pruning convolutional neural networks with visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1364–1373, 2021.

[32] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu, "Analyzing the training processes of deep generative models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 77–87, 2017.

[33] J. Wang, L. Gou, H. Yang, and H.-W. Shen, "GANViz: A visual analytics approach to understand the adversarial game," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 6, pp. 1905–1917, 2018.

[34] M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg, "GAN Lab: Understanding complex deep generative models using interactive visual experimentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 1–11, 2018.

[35] J. Wang, L. Gou, H.-W. Shen, and H. Yang, "DQNViz: A visual analytics approach to understand deep q-networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 288–298, 2018.

[36] W. He, T.-Y. Lee, J. van Baar, K. Wittenburg, and H.-W. Shen, "Dynamicsexplorer: Visual analytics for robot control tasks involving dynamics and lstm-based control policies," in *2020 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2020, pp. 36–45.

[37] T. Jaunet, R. Vuillemot, and C. Wolf, "DRLViz: Understanding decisions and memory in deep reinforcement learning," *Computer Graphics Forum*, vol. 39, no. 3, pp. 49–61, 2020.

[38] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[39] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[40] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding Atari agents," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 1792–1801.

[41] P. Gupta, N. Puri, S. Verma, D. Kayastha, S. Deshmukh, B. Krishnamurthy, and S. Singh, "Explain your move: Understanding agent actions using focused feature saliency," in *International Conference on Learning Representations (ICLR)*, 2020.

[42] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[43] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.

[44] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[45] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[46] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[47] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

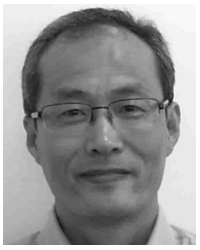
[48] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *International Conference on Learning Representations (ICLR)*, 2018.

Junpeng Wang is a staff research scientist at Visa Research. He received his B.E. degree in software engineering from Nankai University in 2011, M.S. degree in computer science from Virginia Tech in 2015, and Ph.D. degree in computer science from the Ohio State University in 2019. His research interests are broadly in visualization, visual analytics, and explainable AI.

Wei Zhang is a principal research scientist and research manager at Visa Research and interested in big data modeling and advanced machine learning technologies for payment industry. Prior to joining Visa Research, Wei worked as a Research Scientist in Facebook, R&D manager in Nuance Communications and also worked in IBM research over 10 years. Wei received his Bachelor and Master degrees from Department of Computer Science, Tsinghua University.

Hao Yang is the VP of Data Analytics at Visa Research and he is leading a team for advanced machine learning research to tackle challenging problems in the payment industry and develop the world's best commerce intelligence engine. Hao received his B.S. and M.S. degrees from University of Science and Technology of China (USTC) and Chinese Academy of Sciences (CAS) respectively, and his Ph.D. degree in Computer Science from University of California, Los Angeles (UCLA). He has published over forty papers in international conferences and journals, including a Best Paper Award from the International Conference on Parallel Processing (ICPP) in 2008.

Chin-Chia Michael Yeh is currently a Staff Research Scientist at Visa Research. Michael received his Ph.D. in Computer Science from University of California, Riverside. His Ph.D. thesis "Toward a Near Universal Time Series Data Mining Tool: Introducing the Matrix Profile," received Doctoral Dissertation Award Honorable Mention at KDD 2019. He has published papers in top venues, including KDD, VLDB, ICDM and others. His research interests are in data mining, machine learning, and time series analysis.



Liang Wang is a principal research scientist in the Data Analytics team at Visa Research. His research interests are in data mining, machine learning, and fraud analytics. Liang received his Ph.D. degree in Computer Science from Faculté Polytechnique de Mons, Mons, Belgium with highest honors, and his BS degree in Electrical Engineering & Automation and his MS degree in Systems Engineering, both from Tianjin University. Prior to joining Visa, He was a senior principal data scientist at Yahoo!, responsible for building traffic protection solutions for Yahoo!'s advertising platforms. Before Yahoo!, he was a distinguished scientist at eBay/PayPal, leading projects on risk detection for PayPal's core payment system. Liang also worked at FICO as a senior scientist, focusing on bankcard fraud detection. He is the inventor of over 20 patents and has published over 30 papers in international journals and conferences.