DynamicsExplorer: Visual Analytics for Robot Control Tasks involving Dynamics and LSTM-based Control Policies

He, Wenbin; Lee, Teng-Yok; van Baar, Jeroen; Wittenburg, Kent B.; Shen, Han-Wei

TR2020-011 January 24, 2020

Abstract

Deep reinforcement learning (RL), where a policy represented by a deep neural network is trained, has shown some success in playing video games and chess. However, applying RL to real-world tasks like robot control is still challenging. Because generating a massive number of samples to train control policies using RL on real robots is very expensive, hence impractical, it is common to train in simulations, and then transfer to real environments. The trained policy, however, may fail in the real world because of the difference between the training and the real environments, especially the difference in dynamics. To diagnose the problems, it is crucial for experts to understand (1) how the trained policy behaves under different dynamics settings, (2) which part of the policy affects the behaviors the most when the dynamics setting changes, and (3) how to adjust the training procedure to make the policy robust. This paper presents DynamicsExplorer, a visual analytics tool to diagnose the trained policy on robot control tasks under different dynamics settings. DynamicsExplorer allows experts to overview the results of multiple tests with different dynamics-related parameter settings so experts can visually detect failures and analyze the sensitivity of different parameters. Experts can further examine the internal activations of the policy for selected tests and compare the activations between success and failure tests. Such comparisons help experts form hypotheses about the policy and allows them to verify the hypotheses via DynamicsExplorer. Multiple use cases are presented to demonstrate the utility of DynamicsExplorer.

IEEE Pacific Visualization Symposium (PacificVis)

DynamicsExplorer: Visual Analytics for Robot Control Tasks involving Dynamics and LSTM-based Control Policies

Wenbin He[*] [†]
The Ohio State University

Teng-Yok Lee[‡]
Mitsubishi Electric Research Labs (MERL)

Jeroen van Baar[§]
Mitsubishi Electric Research Labs (MERL)

Kent Wittenburg[¶]
Mitsubishi Electric Research Labs (MERL)

Han-Wei Shen[‖]
The Ohio State University

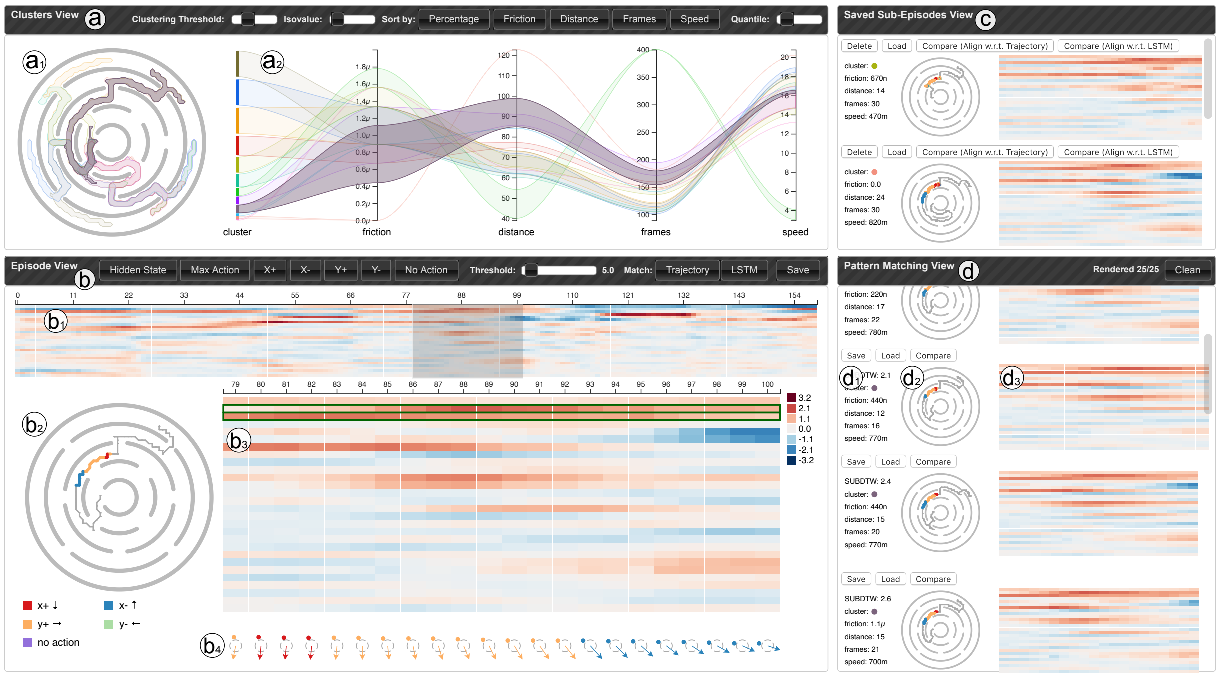

Figure 1: Overview of DynamicsExplorer, a visual analytics tool to diagnose trained policies on robot control tasks involving dynamics. (a) the Cluster view presents the overview of the testing episodes as clusters of ball trajectories (a1) and the corresponding statistical summaries of each cluster in a parallel coordinate plot (a2); (b) the Episode view shows the relationship between the hidden states of the LSTM and the status of the environment for selected episodes. We present the hidden states over an entire episode as a heatmap (b1). The dark grey region represents a sub-episode selection. The trajectory of the selected sub-episode is displayed on the maze (b2) and the corresponding hidden states are shown in the heatmap exploration area (b3). The ball location and maze tilt for the sub-episode are shown below the heatmap exploration area, see (b4); (c) the Saved Sub-Episodes view shows the sub-episodes that have been saved by users for comparison; (d) the Pattern Matching view shows the matches to a pattern query specified by users in the Episode view. For example, the selected hidden states channels, shown as green bounding boxes in the heatmap exploration area (b3), were queried using the LSTM query button in the Episode View bar. Each match displayed in (d) contains information about matching scores and dynamics parameters (d1), the matched sub-trajectory drawn on the maze (d2), and the corresponding LSTM hidden states (d3). DynamicsExplorer is described in detail in Section 5.

Abstract

Deep reinforcement learning (RL), where a policy represented by a deep neural network is trained, has shown some success in playing video games and chess. However, applying RL to real-world tasks like robot control is still challenging. Because generating a massive number of samples to train control policies using RL on real robots is very expensive, hence impractical, it is common to train in simulations, and then transfer to real environments. The trained policy, however, may fail in the real world because of the difference between the training and the real environments, especially the difference in dynamics. To diagnose the problems, it is crucial for experts to understand (1) how the trained policy behaves under different dynamics settings, (2) which part of the policy affects the behaviors the most when the dynamics setting changes, and (3) how to adjust the training procedure to make the policy robust.

This paper presents DynamicsExplorer, a visual analytics tool to diagnose the trained policy on robot control tasks under different dynamics settings. DynamicsExplorer allows experts to overview the results of multiple tests with different dynamics-related parameter settings so experts can visually detect failures and analyze the sensitivity of different parameters. Experts can further examine the internal activations of the policy for selected tests and compare the

[*]The first author started this work when interning in MERL.
[†]e-mail: he.495@osu.edu
[‡]e-mail: teng-yok.t.lee@ieee.org
[§]e-mail: jeroen@merl.com
[¶]e-mail: kentwitt@gmail.com
[‖]e-mail: shen.94@osu.edu

activations between success and failure tests. Such comparisons help experts form hypotheses about the policy and allows them to verify the hypotheses via DynamicsExplorer. Multiple use cases are presented to demonstrate the utility of DynamicsExplorer.

1 Introduction

Since the success of AlphaGo [22], deep reinforcement learning (RL) has attracted an increasing amount of attention and become a very active research topic. With carefully-designed reward functions, well-trained deep RL models (deep neural networks representing control policies) are able to play chess and video games surpassing human-level performance. Recently, a major focus in the machine learning field is to utilize RL to solve tasks in real environments such as robot control that are difficult to annotate [25]. However, as deep RL requires many samples for training, it is impractical to directly apply them to areas like robotics due to cost and safety concerns. Hence, training deep RL models on simulations and subsequently transferring the models to real environments, dubbed Sim2Real, is often employed.

The major goal of Sim2Real is to transfer the trained RL model to real environments with as little fine-tuning as possible, which is challenging for the following reasons. First, real-world tasks often involve *dynamics* (i.e., mechanics that deals with the motion and equilibrium of systems under the action of forces), which cannot be fully simulated even with advanced physical engines, e.g., MuJoCo [24]. One major reason is that simulations are often controlled by different dynamics parameters (such as frictions), whose value in real environments is unknown to the domain experts. Second, because deep RL models are often highly complex and difficult to interpret, it is challenging to adjust the model directly to cope with new environments. As a result, transferring deep RL models to real-world tasks often requires a considerable amount of fine-tuning, which is time-consuming and sometimes impractical.

To address the aforementioned challenges, it will be ideal if domain experts could understand how the model reacts to different dynamics settings using simulations, so more stratified approaches could be developed to fine-tune the network. Even better, if the experts can analyze the network activations and understand how information is encoded in the network, they may be able to adjust the encoding to obtain desired behavior or substantially reduce the amount of fine-tuning time. While various visual analytics approaches have been successfully applied to interpret and diagnose deep learning models [14, 23, 27], visualizing and understanding deep RL models under different dynamics settings is non-trivial for the following reasons. First, the dynamics are often encoded in high dimensional vectors that change over time. Second, the dynamics corresponding to the status update of the 3D environment has a very complex relation to the behavior of the RL model.

In this work, we propose DynamicsExplorer, a visual analytics system to assist domain experts to visualize, understand, and explore the behavior of RL models under various dynamics settings such that the amount of fine-tuning can be reduced. Given a trained deep RL model that is tested with different dynamics settings, DynamicsExplorer first provides an overview of the test results, which allows experts to not only detect failure cases but also examine whether the failure cases are correlated to certain dynamics parameters. Experts can then further visualize and compare the change of environment and model activations over time for selected cases. Based on the visual comparison of success and failure cases, experts can form hypotheses about which part of the model contributes to the abnormal behaviors, and then use DynamicsExplorer to verify the hypotheses by examining all the test cases.

This paper is structured as follows. Section 2 reviews a specific robot control task and the corresponding RL model that motivates DynamicsExplorer. Section 3 lists the requirements for the visual analytics tool, and Section 4 discusses prior research and notes where they fall short of satisfying the requirements. Section 5 describes DynamicsExplorer in detail. Section 6 lists various use cases of DynamicsExplorer, including the exploration of dynamics parameters, the guidance of fine-tuning, and the understanding of how the dynamics are encoded. After the discussion of experts' feedback and the future work in Section 7, Section 8 concludes this paper.

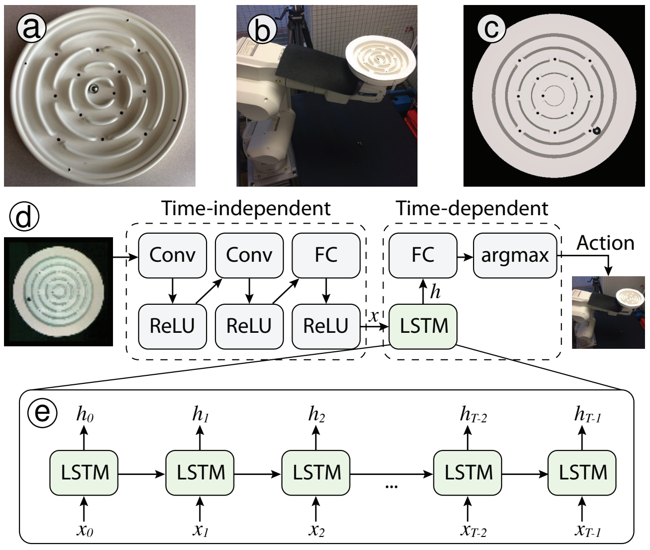

Figure 2: Overview of our robot control task [25]. (a) The ball-in-maze game, whose goal is to move a ball from the outermost ring to the maze center. (b) The ball-in-maze game on a robot arm. An overhead camera provides observations of the maze. (c) A rendered image of our simulation to mimic the real maze in (a). (d) Architecture of our neural network, which represents a control policy for the robot. (e) Unrolled representation of the LSTM layer in our model, which is used to encode the dynamics of the game.

2 Background: Ball-in-Maze Game

DynamicsExplorer is motivated by a specific robot control task [25], which is developed as a test bed to study more general Sim2Real algorithms. The task is to train a robot to solve a circular ball-in-maze game from only image observations as input. The goal is to move the ball from the outermost ring through a series of gates into the innermost ring, by tilting the maze. The maze is placed on a robot arm, and an overhead camera captures images of the tilting maze and resulting ball motion. See Figures 2(a) and (b).

A deep RL model is trained to play the game by mapping states of the game to actions, whose architecture is shown in Figure 2(d). The model takes images from the camera as inputs in sequence and outputs a sequence of discrete actions to tilt the maze using the robot arm. At each frame, the captured image is passed through two convolutional layers and one fully connected layer to extract image features such as the ball location and encode them into a latent vector x. ReLU activations are used to introduce nonlinearity to the model by clamping negative values to 0. The latent vectors $x_0, x_1, \dots, x_{T-1}$ are then fed into a long short-term memory (LSTM) layer [5] in sequence. At each timestep, the LSTM layer takes the latent vector and the output of the previous timestep as inputs and passes the output to the next timestep. Essentially, a LSTM layer recursively fuses the image features over time and generates a sequence of hidden states $h_0, h_1, \dots, h_{T-1}$ as shown in Figure 2(e). Each hidden state h is a high dimensional vector that encodes the dynamics of the game, and each dimension is named a channel of the hidden state. The hidden states are then input to a fully connected layer, and an argmax operation finally determines which action to take. There are five discrete actions to control the maze: tilting clockwise or counterclockwise around the x- or y-axis, denoted as X+, X-, Y+, and Y-, respectively. The fifth action is No-Op, i.e., do nothing.

Arguably this task can be trained by a method other than RL. However, RL is very general and can be used to train a variety of tasks, for both discrete and continuous action spaces [19, 20]. The network architecture is based on [18], which showed that LSTMs with prior action and reward as additional input tend to outperform non-recurrent models.

Since training the model on the real robot is not practical due to cost and safety concerns, the domain experts developed a 3D simulation to mimic the physics of the ball rolling in the maze using MuJoCo [24] and render the maze as shown in Figure 2(c). The simulation also allows for mimicking of multiple robots in parallel, which is not affordable with real robots due to the cost. While the model can be trained efficiently on simulations, transferring the model to the real world settings requires additional training, i.e., fine-tuning, to cope with real environments [25]. In particular, the LSTM layer of the model is harder to transfer to the real world than the time-independent convolution layers. In a preliminary study [11], the researchers found it was sufficient to only fine-tune the layers after the LSTM layer. They also confirmed that the convolution layers can highlight the ball locations with both rendered images and captured video frames. In contrast, the LSTM layer is sensitive to (unknown) changes in dynamics between the simulation and the real world, making the fine-tuning unavoidable.

The goal of this work is to develop a visual analytics system to help experts explore and analyze the testing runs (i.e., episodes) of a trained model with different dynamics settings on the simulation. With the system, we aim to help experts understand (1) how the model, especially the LSTM layer, reacts to different dynamics settings such as the friction of the maze surface, (2) how the dynamics impact the result of the game, which is represented by the ball trajectories, and (3) how the dynamics are encoded in the LSTMs to guide fine-tuning and ultimately to directly intervene the models behavior.

3 System Requirements

Based on the challenges described in the previous section, we identify a set of requirements in order to build a powerful analytics tool. Our goal is for the tool to afford insights for domain experts so that failure cases can be identified; hypotheses can be formed as to the cause of the failures; hypotheses can be verified; and the RL model can be improved. Here is our proposed set of requirements:

R1: Provide summaries of testing runs. To understand the behavior of the model with respect to various parameters (e.g., initial ball locations, friction), 100s of testing runs are conducted. The target system needs to effectively summarize the testing results with respect to the input parameter settings as well as measurable output parameters (distance, time of ball travels). Experts should be able to

- R1.1: identify success and failure sequences through parameters within a large number of testing runs;
- R1.2: filter on the basis of these parameters to focus on cases of interest;
- R1.3: correlate the testing results with the parameter settings to analyze the sensitivity of the parameters.

R2: Present changes in the environment and the model activations over time. Visualizing how the model reacts to the changes of environment and how the decisions made by the model in turn change the environment is crucial for domain experts to understand the relationship between the model and the dynamics. In detail, the target system should fulfill the following requirements.

- R2.1: Characterize the change of the model activations (i.e., LSTM hidden states) over time, which are high-dimensional vectors (256 in our application) with subtle interactions.
- R2.2: Characterize the changes in the environment over time in a compact and comprehensible way. Robotics applications involve an environment that changes in a multidimensional space. Only the most significant changes should be represented.
- R2.3: Associate the environment and the model with a common temporal reference such that experts are able to correlate the change of the environment with the model's internal states and identify patterns.
- R2.4: Compare differences of the environment and model activations between different testing runs with elastic time windows, such that experts can have better understanding of how the differences of the environment influence the model's internal states and vice-versa. Comparable sequences may have slightly different time windows. (This requirement is particularly useful when it comes to comparing successful sub-episodes to failures.)

R3: Verify hypotheses regarding the association of patterns in the model activations and the environment. Once experts have come up with hypotheses regarding the role of the model in influencing the environment and vice-versa, the user of the system should be able to

- R3.1: search over all testing run sub-sequences using specified patterns in the model activations;
- R3.2: search over all testing run sub-sequences using specified patterns in the environment.

The goal is to determine whether hypotheses involving an association of sub-sequence patterns in the environment with patterns in the hidden states hold true in general.

4 Related Work

Visual analytics approaches [1, 15] have been playing an increasingly important role in interpreting and diagnosing various deep learning models, such as supervised [8, 14, 23], unsupervised [13, 28], and reinforcement learning models [6, 27]. In the following, we review previous work in visualization of deep RL models and recurrent neural networks (RNNs), which are closely related to this study.

Visualization of Deep RL Models

Several visual analytics approaches have been proposed to interpret deep RL models. A commonly used technique [18, 20, 29] is to project and visualize the model activations with t-SNE [26]. Although the t-SNE projection is effective in providing an overview of the model activations, it is difficult to correlate the model activations with the environment. Recently, Wang et al. [27] proposed DQNViz to visualize the training process of DQN-based reinforcement learning, but DQNViz focuses on models that are not dependent on dynamics, which play an important role in our application. Jaunet et al. [6, 7] proposed visual analytics tools to interpret the internal states of RL models and correlate them with the environment. However, their approaches do not consider the change of simulation parameters and how the parameters influence the internal states of the model.

Visualization of RNN Models

Most of the visual analytics approaches for RNN models focus on applications in natural language processing. Karpathy et al. [9] overlaid the LSTM hidden states with the input sentences to interpret the LSTM models. Li et al. [12] compared different RNN models by visualizing the change of hidden states over time with heatmaps. Ming et al. [17] proposed RNNVis to understand the relations between the hidden states and the input sentences with biclustering. Hendrik et al. [23] proposed LSTMVis, a visual analytics tool to help experts identify patterns in LSTM hidden states and verify hypotheses on those patterns. Compared with these previous approaches, this study focuses on more complex input sequences (i.e., images that represent the changes in a 3D environment). Moreover, in our application, the environment (i.e., the input to the model) can be influenced by the output actions of the model. This makes our application more complicated than in the previous work because in their applications the input texts are not dependent on the outputs of the model.

5 Visual Analytics System: DynamicsExplorer

Following the requirements described in Section 3, we designed and developed DynamicsExplorer with multiple coordinated views as shown in Figure 1 and detailed below.

5.1 Clusters View

The Clusters view (Figure 1(a)) provides an overview of the testing episodes by clustering of the ball trajectories and the corresponding statistical summaries of each cluster (R1). This view contains two juxtaposed sub-views, as shown in Figure 1(a1) and 1(a2).

5.1.1 Trajectory Variability Plot View

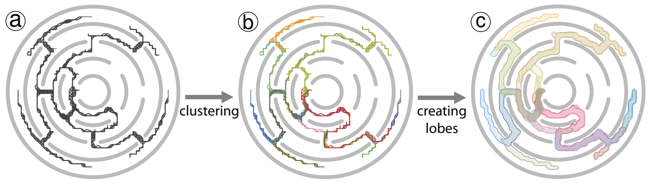

Figure 3: Process of creating the trajectory variability plot. Ball trajectories (a) are first grouped into clusters (b) with agglomerative hierarchical clustering. Then, the clusters are visualized as confidence lobes (c).

The Trajectory Variability Plot view (Figure 1(a1)) shows the clusters of the ball trajectories and visualizes the clusters as confidence lobes [4]. This view provides a quick overview of all the trajectories and enables users to identify failure cases effectively (R1.1). In the following, we discuss how to cluster the trajectories and create the confidence lobe for each cluster in detail.

Given a set of ball trajectories as shown in Figure 3(a), we first compute the distance between each pair of trajectories. Because the trajectories may start from different locations and have different lengths, point-wise distances such as the $L2$-norm are not directly applicable. In this work, we measure the distance between trajectories using the *mean of closest point distance* d_M [2], which is widely used to compare flowlines [16] and fibers [2]. Given two trajectories T_i and T_j, d_M computes the minimal distance from each point of one trajectory to the other, and then computes the distance between T_i and T_j as the average of all minimal distances, which is defined as

$$\begin{aligned} d_M(T_i,T_j) &= \text{mean}(d_m(T_i,T_j), d_m(T_j,T_i)) \\ \text{with } d_m(T_i,T_j) &= \text{mean}_{p_l \in T_i} \text{min}_{p_k \in T_j} \|p_l - p_k\|, \end{aligned} \tag{1}$$

where p are sample points on the trajectories. Then, we group the trajectories into clusters (Figure 3(b)) using agglomerative hierarchical clustering with average linkage, which is often used to cluster flowlines [4]. Users can adjust the clustering threshold via a slider to decide how similar the trajectories of a cluster should be.

After clustering, we visualize each cluster of trajectories as a confidence lobe (Figure 3(c)) instead of spaghetti plots (Figure 3(b)) to reduce visual clutter and convey the variability of the trajectories to users. Given a cluster of trajectories, we follow the method proposed in [4] to create the confidence lobe. We first discretize the 2D space with a regular grid and create a visitation map by counting for each grid cell the number of trajectories that have passed through it. Then, we smooth the visitation map with a Gaussian kernel and extract the confidence lobe from the visitation map as an isocontour. Users can adjust the isovalue to change the size of the confidence lobes interactively.

5.1.2 Parallel Coordinates Plot (PCP) View

The PCP view (Figure 1(a2)) aims to provide the statistical summaries of each cluster, including the input parameters used to generate the testing episodes (e.g., the starting position and the friction) and the characteristics of the testing episodes (e.g., the distance the ball moves and the number of frames) (R1.2). Each attribute in the statistical summary is represented as one coordinate in the PCP, and each cluster is represented as a band across the coordinates. The range of a band on each coordinate covers certain percentages of the data around the median of the cluster, which could be adjusted by users. Note that the Trajectory Variability Plot view and the PCP view are linked such that selecting a cluster on one view can highlight the corresponding cluster in the other view (R1.3).

With the two sub-views in the Clusters view, users can quickly identify failure cases and form hypotheses about the cause. For example, in Figure 1(a2), the selected episodes took a small number of frames but the ball moved a relatively longer distance, which indicates overshooting failure cases as shown in Figure 1(a1). By checking the input parameters, users can further hypothesize that these failure cases may be caused by low friction values. Then, users can select the median of the failure cases and do further analysis as detailed in the following sections.

5.2 Episode View

The Episode view (Figure 1(b)) is the main view of the system, which visualizes and compares episodes in terms of the hidden states of the LSTM and the status of the environment (i.e., the ball location and the tilt of the maze) (R2). In the following, we first discuss the visualizations of the LSTM hidden states and the status of the environment with respect to a single episode. Then, we extend the visualizations to compare two episodes.

5.2.1 Visualization of the LSTM Hidden States

Visualizing the change of hidden states along the time dimension (R2.1) is crucial to understand how the dynamics are encoded in the model. Previous work typically visualizes the raw hidden state vectors directly with either heatmaps [9,17] or line charts [23]. However, there are two limitations in the previous work. First, as the dimensionality of the hidden state vectors gets higher, visualizing the raw vectors directly does not scale well. The heatmaps will become too big to interact with and the line charts will become cluttered as the number of lines increases. Second, each channel in the hidden states is treated equally without considering its contribution to the final actions, which can obscure the significant data and lead to imprecise findings. To address these limitations, we first perform dimensionality reduction on the hidden states with principal component analysis (PCA) and then calculate the contributions of each principle component (PC) to the final actions.

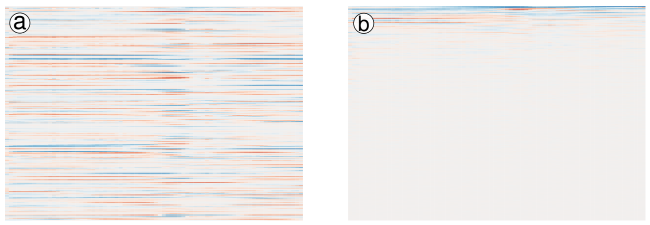

Figure 4: LSTM hidden states in the original space (a) and PCA-space (b). A small subset of PCs is sufficient to perserve the variance.

Dimensionality Reduction with PCA We collect the hidden state vectors $h_0, h_1, \ldots, h_{N-1}$ from all the testing episodes and treat each hidden state vector as a point in the high dimensional space (256 dimensions in our application). Then, we perform PCA on the data points and transform each hidden state vector into the PCA-space by subtracting the mean vector $\bar{h}$ and then multiplying with the rotation matrix M as

$$h'_i = (h_i - \bar{h})M. \tag{2}$$

In the PCA-space, a small subset of PCs is often sufficient to preserve the variance of the data. For example, Figure 4(a) shows the LSTM hidden states of an episode using a heatmap and Figure 4(b) shows the hidden states in the PCA-space, where each row represents one channel/PC. We can see that only a small subset of PCs on the top have values with high variance in Figure 4(b). While there are various dimensionality reduction techniques, we use PCA in this work for the following reasons:

- PCA preserves the structure of the data as it just applies linear transformations (translation and rotation) on the data without changing the overall shape of the data.
- PCA minimizes the correlation between PCs such that different PCs are more likely to encode different dynamics information.
- Points in the PCA-space can be transformed back to the original space, which plays an important role in computing the PC-wise contribution to the actions.

After the PCA-based dimensionality reduction, users can focus on the change of the hidden state over time with a relatively small number of PCs to identify patterns effectively.

Computation of PC-wise Contribution To interpret the LSTM hidden states, visualizing the hidden state values alone is not sufficient because of the lack of connections to the actual actions. In this work, the mapping from the LSTM hidden states to the action scores is achieved via a fully connected layer with a $n_a \times n_l$ weight matrix W, where n_a and n_l denote the number of actions and LSTM channels, respectively. Hence, given a hidden state h, its contribution to the actions can be computed by multiplying it with W. As we are visualizing the hidden states in the PCA-space, it is important to understand how each PC contributes to the actions. Given a hidden state h' in the PCA-space, we compute the contribution c_j of the j-th PC as follows. First, we set the values of other PCs to 0, which gives us a new vector denoted as $h'^{(j)}$ and then transform it back to the original space by multiplying it with the inverse of the rotation matrix M^{-1}. In the end, we transform the resulting vector with the weights W to compute the contribution of the PC to the actions. In summary, the PC-wise contribution computation can be written as

$$c_j = h'^{(j)} M^{-1} W. \tag{3}$$

Visual Design We visualize the hidden state values or contributions in the PCA-space as heatmaps using the PCs that explain more than 90% variance of the data, as shown in Figure 1(b1). We use heatmaps instead of other visual designs such as line charts because the scalability issue is alleviated with a reduced number of dimensions and heatmaps preserve the order of the PCs, which is important as the PCs are sorted based on the data variances they explain. In addition, as one episode is typically a long time sequence (hundreds of frames), the system lets users focus on particular sub-episodes through brushing and zooming as shown in Figure 1(b1 and b3).

5.2.2 Visualization of the Environment

Visualizing the status of the environment and aligning it with the LSTM hidden states is important for users to explore and analyze how the dynamics (e.g., the ball location and speed) are encoded in the LSTM (R2.2, R2.3). To this end, we develop two sub-views detailed as follows.

Trajectory View The Trajectory view (Figure 1(b2)) provides the overview of the status of the simulation by visualizing the trajectory of the ball. We use the color of the trajectory to encode additional information such as the action the model takes at each frame. We link the Trajectory view with the LSTM hidden states through brushing and zooming as mentioned early. When users select a sub-episode through brushing, the trajectory and the LSTM hidden states that correspond to the selected sub-episode are highlighted simultaneously.

Glyph View To visualize the status of the simulation at each frame in detail, we design glyphs as shown in the series in Figure 1(b4). Each glyph provides the location of the ball and the tilt of the maze. We visualize the ball and the walls of the maze together to indicate the location of the ball. Note that we only draw walls between the current and the inner ring to make the glyph more compact. For the tilt of the maze, we accumulate all the tilts along the x- and y-axis into a vector and visualize it as an arrow centered at the origin. As with the Trajectory view, we use color to encode which action was taken at that time step. We align the glyphs with the LSTM hidden states of the selected sub-episode vertically. In addition, to avoid overlapping, we change the number of glyphs dynamically as the number of frames changes in the selected sub-episode.

5.2.3 Comparison between Two Episodes

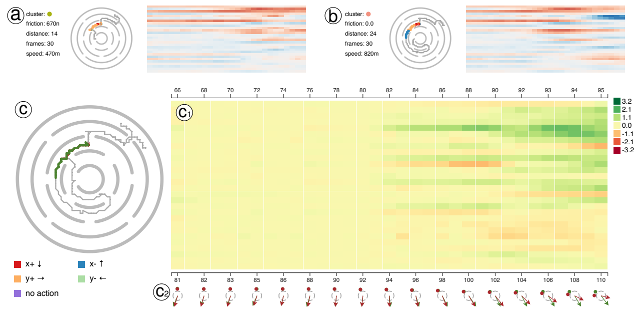

Figure 5: Comparing two sub-episodes (a) and (b) in the Episode view (c). The different activations of the LSTM hidden states are shown as a difference heatmap in (c1), and the status of the environment for both sub-episodes are overlaid for comparison as shown in (c2). Note for the difference heatmap in (c1), a different colormap (red for negative values, yellow for zeros, and green for positive values) from the original activation map is used.

Besides visualizing and analyzing a single episode, we extend the Episode view to compare two episodes with respect to the user selected sub-episodes (Figure 5). Visual comparison between sub-episodes is crucial for users to identify differences between cases and form hypotheses about the cause (R2.4). A straightforward solution to compare sub-episodes is to visualize them in juxtaposed views. However, it is difficult and inefficient for users to identify non salient differences from juxtaposed views. Hence, in this work, we compare two sub-episodes by visualizing them as a difference heatmap. As the sub-episodes in comparison may start from different statuses and have different number of frames, aligning the sub-episodes in time plays an important role for visual comparison. In this work, we use dynamic time warping (DTW) [21] to align the two sub-episodes as detailed below.

Alignment based on DTW We allow users to align sub-episodes using DTW based on either the ball locations or the LSTM hidden states. Here, we discuss the method using ball locations as an example. Given two time series of ball locations $p_0, p_1, \ldots, p_{m-1}$ and $q_0, q_1, \ldots, q_{n-1}$, where m and n are the number of frames in the selected sub-episodes, we first calculate the DTW matrix D based on the following equation:

$$D_{i,j} = dist(p_i, q_j) + \begin{cases} 0 & i = 0 \,\&\, j = 0 \\ D_{i,j-1} & i = 0 \,\&\, j > 0 \\ D_{i-1,j} & i > 0 \,\&\, j = 0 \\ \hat{D}_{i,j} & otherwise \end{cases}, \quad (4)$$

where $dist(p_i, q_j)$ is the Euclidean distance between p_i and q_j and $\hat{D}_{i,j} = \min(D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1})$. Based on D, we can extract the warping path by starting from $D_{m-1,n-1}$ and moving down and left along the path with minimum $D_{i,j}$. According to the warping path, the two series can be warped non-linearly to align with each other with a least distortion.

Visual Design After aligning the selected sub-episodes, we visualize and compare them in terms of the LSTM hidden states and the status of the environment. For the LSTM hidden states, we compute the difference between the sub-episodes and visualize the difference through heatmaps as shown in Figure 5(c1). Note that in order to remind users that the heatmap is showing the difference, not the original activation values, a different colormap is used to differentiate it with the hidden state values. To compare the status of the environment between the sub-episodes, we update the aforementioned Trajectory and Glyph views in Section 5.2.2 by superimposing the trajectories, ball locations, and tilts of the maze with different colors as shown in Figure 5(c2). During exploration, we allow users to save sub-episodes in Figure 1(c) for later comparison.

5.3 Pattern Matching View

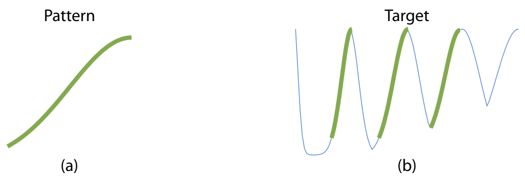

Figure 6: (a) Pattern defined by a time series of values in 1D. (b) Blue curve is the target time series, and the green curves are the matches found by SUBDTW.

By visualizing and comparing selected sub-episodes, users may identify interesting patterns and need to confirm certain hypotheses about the patterns in other episodes. To this end, we search the user specified pattern in the testing episodes with a pattern matching algorithm called SUBDTW [10]. Then, we visualize the pattern matching results in the Pattern Matching view to help users confirm or reject the hypotheses (R3).

Pattern Definition In this work, we define a pattern as a time series of ball locations or hidden state values (for user selected PCs). Users could search for similar ball trajectories to check whether the corresponding hidden states are similar (R3.1). Users could also search for similar hidden states patterns to confirm or reject that the patterns only correspond to certain dynamics (e.g., series of ball locations) (R3.2). In the following, we demonstrate the pattern matching algorithm with a time series of hidden state values for a PC as example. It is straightforward to extend to multiple PCs or the ball locations.

Pattern Matching with SUBDTW Let's define the pattern as a time series of hidden state values $p_0, p_1, \ldots, p_{m-1}$ for a particular PC, which is essentially a 1D curve as shown in Figure 6(a). The goal of the pattern matching is to find sub time series in each target episode that have similar patterns, as shown in Figure 6(b). The sub time series that match the pattern could start from arbitrary time frames and contain different numbers of frames. A robust distance metric is needed to compare the pattern and the target. The DTW mentioned earlier could address the issue of different numbers of time frames, but it forces the matching to start from the first frame. In this work, we use the SUBDTW algorithm, which extends the DTW such that the matching could start from arbitrary time frames in the target. Assuming the target is a time series of hidden state values $q_0, q_1, \ldots, q_{n-1}$, where n is typically much larger than m. The SUBDTW computes the DTW matrix with a modified version of Equation 4

$$D_{i,j} = dist(p_i, q_j) + \begin{cases} 0 & i = 0 \,\&\, j = 0 \\ 0 & i = 0 \,\&\, j > 0 \\ D_{i-1,j} & i > 0 \,\&\, j = 0 \\ \hat{D}_{i,j} & otherwise \end{cases}, \quad (5)$$

where the main difference is the case when $i = 0 \,\&\, j > 0$, we set $D_{i,j}$ to 0 instead of $D_{i,j-1}$. The idea is that we do not want to accumulate the distance from the starting frame of the target. Once the matrix D is computed, we compare the distances in row D_{m-1} with a user defined threshold value, and for any $D_{m-1,j}$ that is smaller than the threshold value, we extract the warping path using the method mentioned in Section 5.2.3 and find the matches.

Visualizing Matched Sub-episodes We list and visualize the matched sub-episodes in the Pattern Matching view. The matched sub-episodes are first sorted based on the SUBDTW distance starting from the most similar one. For each sub-episode, we first list the corresponding parameter settings (e.g., friction value) and statistical summaries (e.g., number of frames) for the sub-episode as shown in Figure 1(d1). Then, we visualize the matched sub-trajectory as a curve in Figure 1(d2) and the LSTM hidden states as a heatmap in Figure 1(d3). We allow users to compare each of the sub-episodes with the pattern in the Episode view to identify the difference between them in detail. Note that in this comparison, we directly use the warping path generated by SUBDTW to align the matched sub-episode with the pattern.

6 Use Cases

We worked closely with the domain expert who developed the RL model that solves the ball-in-maze task decribed in Section 2 on several use cases. In this section, we present two use cases to demonstrate the usefulness and effectiveness of DynamicsExplorer: one focuses on how DynamicsExplorer can help experts diagnose the trained model under different parameter settings to identify failure cases, form hypotheses about the cause, and further improve the model; the other aims to interpret the internal activations of the model and correlate them with changes of the environment.

For both use cases, we trained the ball-in-maze task using a neural network based on the implementation of our expert [25]. Training comprised four million iterations on the simulator, with a fixed setting of the dynamics-related parameters decided by the expert. We aim to understand LSTM encodings from first principles by testing the model under different parameter settings in simulation. Among the various parameters, our expert is particularly interested in two: the friction value of the maze material, and the starting location of the ball. Hence, we tested the model with the combinations of 12 initial ball locations, placed at regular intervals in the outermost ring, and 10 different friction values. The friction values for testing were regularly sampled from $[0, 2 \times 10^{-7}]$ (the friction value used in training was 1×10^{-7}). For each parameter setting, a testing episode was generated by executing the game with the trained model until the ball reaches the center of the maze, or the maximum number of frames is reached, which is 400 for our application.

6.1 Model Diagnosis and Fine-Tuning

In this use case, we demonstrate how DynamicsExplorer is used to diagnose the model under different parameter settings and guide the process of fine-tuning to improve the model.

6.1.1 Model Diagnosis

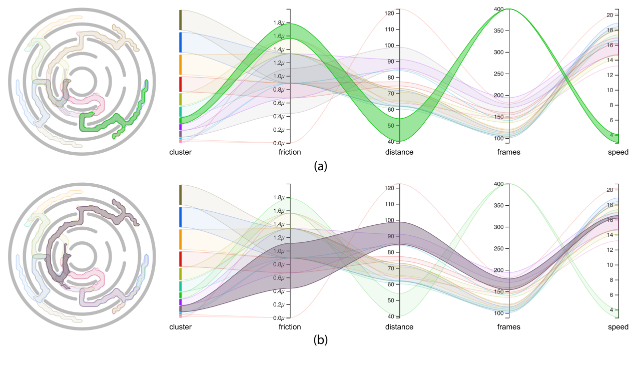

Figure 7: Two failure cases identified from the Clusters view: (a) a *stuck* case: the ball fails to reach the center and gets stuck in an earlier ring; and (b) an *overshooting* case: the ball passes a gate it would otherwise enter for an optimal control policy.

By exploring the clusters in the Clusters view, we can quickly identify when and where the model fails. Figure 7 shows the two main failure cases that we identified. The first case is where the ball fails to reach the center and gets *stuck* in an earlier ring, as shown in Figure 7(a). This can be quickly identified from the PCP (i.e., the green band in Figure 7(a)), as the ball moved a relatively short distance yet uses a large number of frames (the maximum episode length of 400 in the illustrated case). The second case is that the ball *overshoots*, i.e., it passes the gate it is supposed to enter, as shown in Figure 7(b). This case can be identified from the PCP as well.

We can also correlate the failure cases with the parameters (e.g., friction values of the maze material). In the PCP, we show the following parameters from left to right: cluster for the number of elements, friction value, distance as the length of the ball trajectory, frames, and speed, which is the ratio of distance divided by the number of frames. We find that the friction values are relatively higher for the stuck cases and relatively lower for the overshooting cases, compared with the friction value used in training. It is noteworthy that the initial ball locations are similar for the failure cases. For example, for the overshooting cases, the ball mainly starts from the top right corner of the maze. This was actually a surprise to the domain expert. While it is understandable that low friction values can speed up the ball, which could result in overshooting, he expected that it should be independent of the initial ball location, which is revealed to be not true, according to DynamicsExplorer.

After the identification of failure cases, we can further compare them with successful cases in terms of the model activations and the environment, to get more insights into the cause of the failures. Such comparisons are needed when troubleshooting the transfer of the simulation-based model to different real environments. Figure 8 shows the comparison between an overshooting case (Figure 8(a)) and a non-overshooting case (Figure 8(b)) starting from similar initial locations. The comparison is within a small time window right before the overshooting. After aligning the two cases based on the sub-trajectories, we can see that the maze has more tilting towards the right in the non-overshooting case after frame 83 compared with the overshooting case, as shown in Figure 8(c) and (d). The reason is that the overshooting case takes one more X+ action (red action in Figure 8(a)) compared with the non-overshooting case.

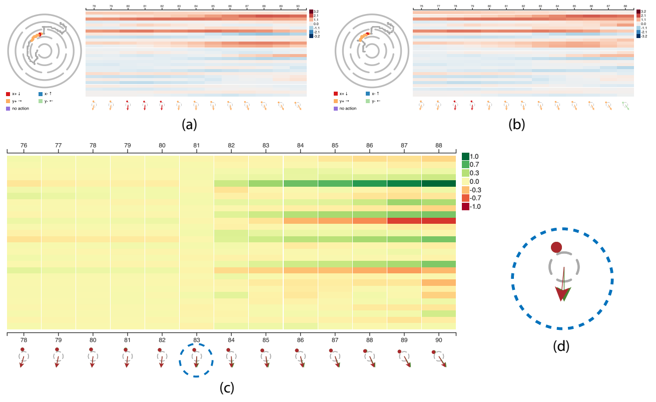

Figure 8: (*Best viewed electronically*) Comparison between an overshooting case (a) and a non-overshooting case (b) starting from similar initial ball locations. The differences between the two cases in terms of the hidden states and the environment are shown in (c). A zoomed in view of the difference circled in (c) is shown in (d).

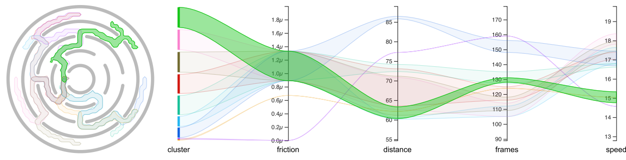

Figure 9: Clusters view for the testing runs generated from the fine-tuned model. All tests starting from the top right corner of the maze go across the nearest gate successfully without overshooting as shown in the green cluster. Compare to Figure 7(b).

6.1.2 Intelligent Fine-Tuning

Based on the aforementioned findings, the expert can fine tune the model more efficiently. Taking the overshooting case as an example: as we observed that the overshooting only happens when the ball starts from the top right corner of the maze, we can fine tune the model with the ball location fixed to the top right. The idea is similar to active learning [3], where a model is trained on data instances selected according to a certain confidence or failure criteria. After training for an additional 300 steps, we tested the model with the same set of friction values and initial ball locations. Figure 9 shows the Clusters view of the testing runs using the fine-tuned model, which confirms that the overshooting cases no longer exist. We also tested fine-tuning the model with random initial ball locations, which requires around 10,000 more steps to fix the overshooting.

6.2 Understanding Neural Network Hidden States

The ultimate goal for analyzing control policies represented by neural networks trained in simulation is to be able to efficiently adapt the control policy to differences in the real world environment. DynamicsExplorer employs interactive querying and comparison abilities to allow domain experts to visually investigate LSTM activation patterns and to explore their relationship with spatial and temporal properties in the environment: ball position, maze tilt, and ball dynamics. In this section, we first compare the activation patterns of the fine-tuned model with the original model to verify hypotheses related to the cause of the failure cases. Then, we conduct several case studies to shed some light on the general patterns of the LSTM activations for our application.

6.2.1 Comparison of Models Before and After Fine-Tuning

We compare the model activations before and after fine-tuning, to gain an understanding about which channels were changed by the fine-tuning that fixed the overshooting issue. We hope that eventually

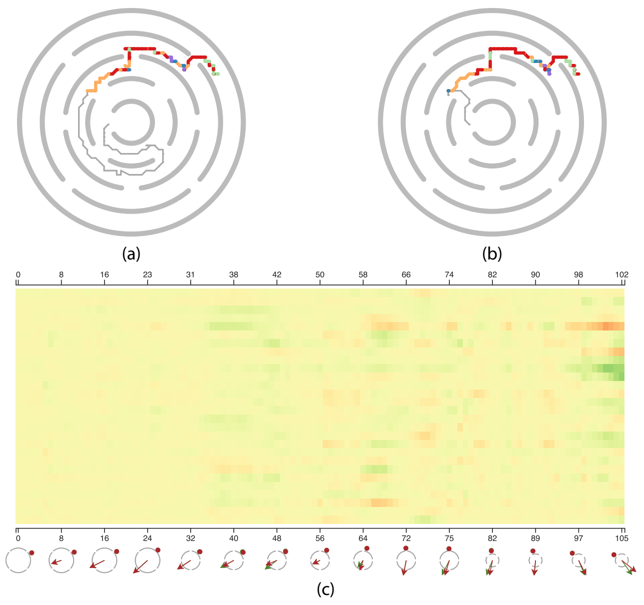

Figure 10: Comparison of the models before and after fine-tuning. (a) Episode with overshooting before fine-tuning. (b) Episode without overshooting after fine-tuning (using the same friction value and initial ball location as (a)). (c) Comparison of hidden states and environment of (a) and (b) of the selected time interval.

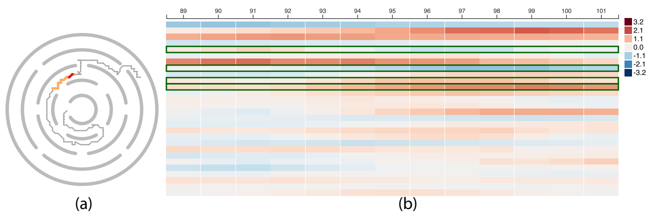

Figure 11: (*Best viewed electronically*) Sub-episode of the overshooting case in Figure 10. (a) Trajectory of the sub-episode. (b) LSTM hidden states of the sub-episode. Here we select the PCs where the overshooting case differs significantly from the non-overshooting case.

experts could adjust channels directly with no, or only minimal, fine-tuning. By combining the episodes of the two models in Section 6.1, we can compare their behaviors and LSTM hidden states.

To compare the LSTM hidden states before and after fine-tuning, we first select an episode with overshooting. Figure 10 (a) shows a trajectory from the original model where the part before overshooting is selected. It is colored based on the series of actions. By using this sub-trajectory to query, we can find the episode with the same initial ball location and friction values tested with the fine-tuned model, which can successfully turn the ball without overshooting as shown in Figure 10(b). Figure10(c) shows the difference in the LSTM activation sequences leading up to the divergent results. The difference heatmap indicates that the activations are very similar until just before the ball overshoots, after which several PCs diverge in value (darker colors on the right in Figure 10(c).

Next, we shrink the time window of the trajectory of Figure 10(a) according to where the LSTM hidden states start to deviate towards the right-hand side of Figure 10(c). The sub-trajectory is shown in Figure 11(a). We then select the PCs where the difference in Figure 10(c) is above a threshold. The selected PC channels are shown in green in Figure 11(b). Finally, we query all episodes with similar hidden state patterns in the selected PCs and the time interval to verify whether these PCs indeed contribute to the overshooting. Figure 12 shows the six most similar episodes, all of which were

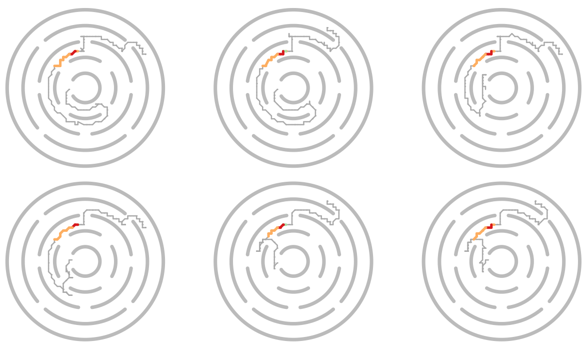

Figure 12: Query results of the selected PCs in Figure 11(b), which shows the six episodes with most similar activation values of selected PCs within the time range. In all of these 6 episodes, the ball overshoots at the same gate.

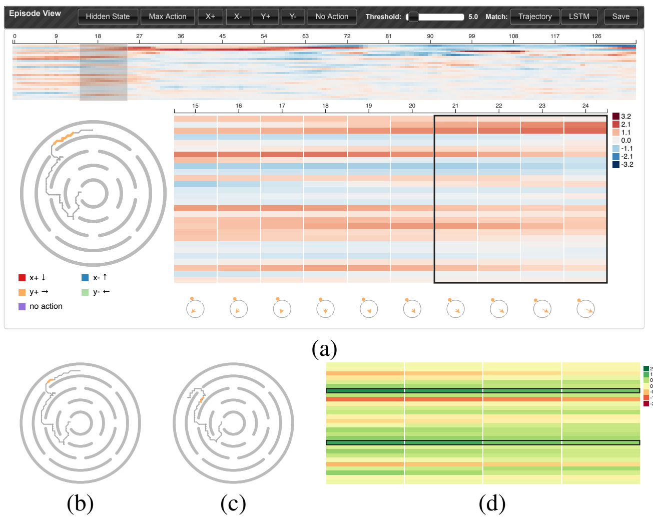

Figure 13: (*Best viewed electronically*) Comparing LSTM hidden states related to similar action sequences at different locations. (a) The episode view of a long sequence of action (Y+) in the outer most ring. While the actions are the same, the heatmap shows that the activation of several PC are changing. (b) The last 4 time steps of (a). (c) The sub-episode that is most similar (b) when using the 2nd PC from the top to query. (d) The difference of the PC between (b) and (c) where the rows with green color means the PC have higher values on (b) than on (c).

trained by the model before fine-tuning, and overshoot the ball near the same gate. Based on this finding, experts can confirm that the hidden state pattern did contribute to the overshooting, which was fixed after fine-tuning. In the future, we hope that experts can further study how to directly adjust the LSTM hidden states to avoid such problems.

6.2.2 Relating LSTM Channels to the Environment

As demonstrated in [23], LSTM channel activations correlate with concepts of the task it was trained on. In this section we describe how we use DynamicsExplorer to investigate if we can determine such channel correlations for our robot task as well.

We analyze the trajectory shown in Figure 13(a). The selected sub-trajectory consists of all Y+ (orange) tilt actions. The heatmap on the right shows, as expected, that as the ball position and maze tilt change, so do the PC activations, and clearly some more than others. We want to investigate whether certain PC channels correlate with ball position, maze tilt, or both.

We first narrow the sub-trajectory to four time steps, as shown by

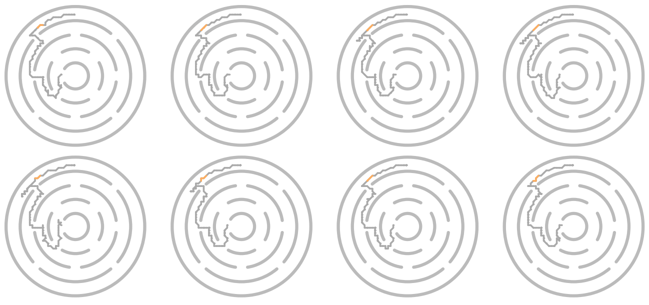

Figure 14: (*Best viewed electronically*) The most similar eight sub-episodes by using the 2 PCs with darkest green colors (highlighted by the black rectangles) of Figure 13(d) to query.

the black rectangle in Figure 13(a). The corresponding sub-trajectory is shown in Figure 13(b). We then query with individual PCs, for which the change in activations is relatively large, e.g., the second PC in the heatmap. With single PC selections we obtain matches which physically have little in common with the sub-trajectory in Figure 13(b), as ball location and maze tilt are very different. One such result is shown in Figure 13(c), where the ball is in a different location in the maze. However, the action sequence in this example is the same as the sub-trajectory we selected. We use DynamicsExplorer to investigate how the PCs of these sub-trajectories differ from each other. The heatmap of this difference is shown in Figure 13(d). Here, PCs with green colors have higher activation on the sub-trajectory of Figure 13(b) compared to the sub-trajectory of Figure 13(c), and a red color denotes lower activation. We select PCs with activation difference above some threshold, marked by the black rectangles in Figure 13(d). Using these two PCs to query, we obtain eight sub-trajectories (shown in Figure 14) with the same action sequence and ball locations as in Figure 13(a). This indicates that these two PCs are highly correlated to ball location and maze tilt near that particular gate. Although we could determine PCs which correlate with a particular ball location and maze tilt using DynamicsExplorer, we have not yet been able to generalize this across the maze.

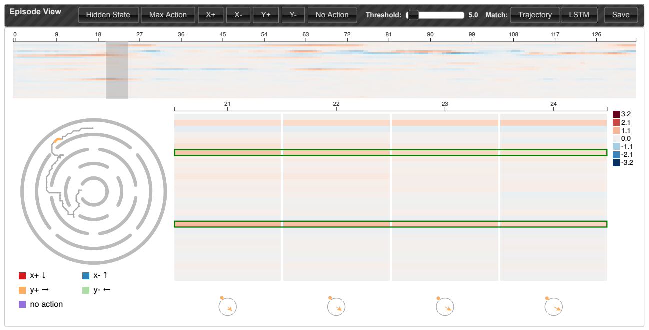

Figure 15: Episode view of the episode of Figure 13(b) where the top heatmap shows the contribution to the action Y+ over time. Compared to the hidden states values in Figure 13(a), the contributions to an action suppresses non-relevant PCs.

To help further investigate, DynamicsExplorer also shows the contribution to different actions. Figure 15, for instance, shows the contribution to the action Y+ of the episode of Figure 13(b). As is evident from the figure, not all high activations in the hidden state contribute significantly to the final actions, Y+ in this case. The bottom-right heatmap shows the same sub-episode of Figure 13(b). The PCs with high contribution to Y+ (shown by the green bounding boxes) are actually the two PCs we found in 13(d). This finding first confirms that these two PCs are highly correlated to ball location and maze tilt near that particular gate. In addition, it also indicates that these two PCs indeed contribute to the final action.

We have shown that DynamicsExplorer is a step towards understanding LSTM encodings for complex tasks. More interactive examples can be seen in the supplementary video. However, in order to fully understand LSTM encodings, DynamicsExplorer needs to be further extended. We will elaborate on this in the next section.

7 Discussion and Future Work

In this section, we discuss the domain expert's feedback, limitations of the current system, and potential future work to address them.

Feedback from the Domain Expert Overall, the expert believed that the tool provides a very powerful ability to diagnose the model by exploring and comparing episodes, particularly at the LSTM pattern level. This ability allows the expert to make Sim2Real for RL more efficient, by exploring problem cases, targeting them for fine-tuning, and validation of fine-tuning. Furthermore, the ability to compare sub-trajectories is a good start to understand the black box of RL policies represented by neural networks. Meanwhile, the expert also mentioned that the current analysis tool has limitations, described next.

Limitations The main limitation of the current system is in generalizing the relations between LSTM hidden states and the change of the environment. Though we are able to explore and identify LSTM patterns that relate to certain ball locations and tilt of the maze, it still relies on the expert to search the patterns to query. It will be ideal if the tool can summarize salient LSTM patterns, allowing the expert to effectively overview which LSTM channel encodes what information in the environment. Another limitation of the current tool is that it does not account for how the long-term memory of the LSTM, i.e., the cell state, relates to updates to the hidden states.

Deeper understanding of LSTM Based on the limitations outlined above, there are several directions we would like to explore in the future. First, we will continue to utilize our existing DynamicsExplorer system to find associations between hidden state patterns, environment states, and actions. This remains a challenging problem, however, and we expect to experiment with other problem domains that might be more amenable to human-comprehensible mappings between the domain environment and internal states of the LSTM. Moreover, based on the understanding of the LSTM patterns, we would like to directly manipulate the hidden state patterns to solve the failure cases, especially in the real-world environment. In addition, we would like to extend the study to the cell state of the LSTM to analyze the long-term memory of the LSTM. One possible extension of the current analysis tool is to incorporate the method presented in [11], and provide the user with an understanding of which previous frames contribute to what degree to the current decision.

Troubleshooting on real robots Besides extending the capacity to understand and to even manipulate the LSTM layers, another goal is using DynamicsExplorer to understand why the transferring to real robots can fail. While the relevant parameter values of real robots could be unknown, since the parameter values are fixed, we could still compare the episodes on real robots before and after the fine-tuning. Like the use case in Section 6.2.1, we believe that it is feasible to locate the LSTM channels adjusted by the fine-tuning for real robots.

Extra applications Meanwhile, we would like to further generalize the current prototype to other tasks. While our current prototype was original designed for specific ball-in-the-maze robot control tasks, several components are independent to this application, including the PCP in the Cluster view to explore the parameter space, the Episode view to overview the neural activation and to select activation patterns of interest, the matching of temporal activation patterns, and the comparison of different episodes with similar prefixes. These components can be applied to other tasks that utilize deep neural networks to make a sequence of decisions.

8 Conclusion

Our research with DynamicsExplorer was motivated by the need to open up the black box of RL models for problems having a significant dynamics component. We focused on a robotics control domain as our test case, and we chose to utilize an LSTM layer in our architecture to model the dynamics of the problem. We noted that the development of RL models for robotics control requires heavy use of simulation for training and that there are inevitably problems with transferring the trained system to the real world, where dynamics aspects will no doubt differ in subtle and significant ways from the simulator. Then our goal became how to gain insight into the behavior of the trained model to improve its performance. Such a goal requires understanding the relationships between the environment and the activations of the model.

Given this goal, we proposed a set of requirements and developed a tool we call DynamicsExplorer. This visual analytics system contains several innovations. Our use cases presented some of our experiences so far in utilizing DynamicsExplorer. We were able to discover failure instances in our dataset through visualizations of parallel coordinates as well as the trajectory variability plot. We confirmed these cases by utilizing the Episode View. For this particular problem, we were able to fine-tune the model with the guidance of DynamicsExplorer. Our second use case attempted to understand the hidden states of LSTM, which is critical when transferring the simulator-trained system to real environments. Using DynamicsExplorer we could demonstrate that certain LSTM channel combinations correlate with ball location and maze tilt, but only for specific trajectories, rather than in general.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive comments. Wenbin He was supported in part by US Department of Energy Los Alamos National Laboratory contract 47145 and UT-Battelle LLC contract 4000159447 program manager Laura Biven.

References

[1] J. Choo and S. Liu. Visual analytics for explainable deep learning. *IEEE Computer Graphics and Applications*, 38(4):84–92, 2018.

[2] I. Corouge, S. Gouttard, and G. Gerig. Towards a shape model of white matter fiber bundles using diffusion tensor MRI. In *Proceedings of 2004 IEEE International Symposium on Biomedical Imaging*, pp. 344–347 Vol. 1, 2004.

[3] S. Das, W. Wong, T. Dietterich, A. Fern, and A. Emmott. Incorporating expert feedback into active anomaly discovery. In *ICDM '16: Proceedings of IEEE International Conference on Data Mining*, pp. 853–858, 2016.

[4] F. Ferstl, K. Bürger, and R. Westermann. Streamline variability plots for characterizing the uncertainty in vector field ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):767–776, 2016.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computing*, 9(8):1735–1780, 1997.

[6] T. Jaunet, R. Vuillemot, and C. Wolf. DRLViz: Understanding decisions and memory in deep reinforcement learning. *arXiv:1909.02982*, 2019.

[7] T. Jaunet, R. Vuillemot, and C. Wolf. RLMViz: Interpreting deep reinforcement learning memory. In *Journée Visu 2019*, 2019.

[8] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

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

[10] T.-Y. Lee and H.-W. Shen. Visualization and exploration of temporal trend relationships in multivariate time-varying data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1359–1366, 2009.

[11] T.-Y. Lee, J. van Baar, K. Wittenburg, and A. Sullivan. Analysis of the contribution and temporal dependency of lstm layers for reinforcement learning tasks. In *CVPR '19 Workshops on Explanable AI*, 2019.

[12] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 681–691, 2016.

[13] 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*, 24(1):77–87, 2018.

[14] 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*, 23(1):91–100, 2017.

[15] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[16] K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. In *PacificVis '13: Proceedings of IEEE Pacific Visualization Symposium*, pp. 257–264, 2013.

[17] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding hidden memories of recurrent neural networks. In *VAST '17: Proceedings of IEEE Conference on Visual Analytics Science and Technology*, pp. 13–24, 2017.

[18] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. *arXiv:1611.03673*, 2016.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pp. 1928–1937, 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[21] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[23] 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*, 24(1):667–676, 2018.

[24] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033, 2012.

[25] J. van Baar, A. Sullivan, R. Cordorel, D. Jha, D. Romeres, and D. Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. *arXiv:1809.04720*, 2018.

[26] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[27] 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*, 25(1):288–298, 2019.

[28] J. Wang, L. Gou, H. Yang, and H. Shen. GANViz: A visual analytics approach to understand the adversarial game. *IEEE Transactions on Visualization and Computer Graphics*, 24(6):1905–1917, 2018.

[29] T. Zahavy, N. B. Zrihem, and S. Mannor. Graying the black box: Understanding DQNs. In *ICML*, pp. 1899–1908, 2016.