Visualizing Dynamics: from t-SNE to SEMI-MDPs

Nir Ben Zrihem* BENTZINIR@GMAIL.COM
Tom Zahavy* TOMZAHAVY@CAMPUS.TECHNION.AC.IL
Shie Mannor SHIE@EE.TECHNION.AC.IL
Electrical Engineering Department, The Technion - Israel Institute of Technology, Haifa 32000, Israel

Abstract

Deep Reinforcement Learning (DRL) is a trending field of research, showing great promise in many challenging problems such as playing Atari, solving Go and controlling robots. While DRL agents perform well in practice we are still missing the tools to analayze their performance and visualize the temporal abstractions that they learn. In this paper, we present a novel method that automatically discovers an internal Semi Markov Decision Process (SMDP) model in the Deep Q Network's (DQN) learned representation. We suggest a novel visualization method that represents the SMDP model by a directed graph and visualize it above a t-SNE map. We show how can we interpret the agent's policy and give evidence for the hierarchical state aggregation that DQNs are learning automatically. Our algorithm is fully automatic, does not require any domain specific knowledge and is evaluated by a novel likelihood based evaluation criteria.

1. Introduction

DQN is an off-policy learning algorithm that uses a Convolutional Neural Network (CNN) (Krizhevsky et al., 2012) to represent the action-value function and showed superior performance on a wide range of problems (Mnih et al., 2015). The success of DQN, and that of Deep Neural Network (DNN) in general, is explained by its ability to learn good representations of the data automatically. Unfortunately, its high representation power is also making it complex to train and hampers its wide use.

Visualization can play an essential role in understanding DNNs. Current methods mainly focus on understanding the spatial structure of the data. For example, Zeiler & Fergus (2014) search for training examples that cause high neural activation at specific neurons, Erhan et al. (2009) created training examples that maximizes the neural activity of a specific neuron and Yosinski et al. (2014) interpreted each layer as a group. However, none of these methods analyzed the temporal structure of the data.

Good temporal representation of the data can speed up the prerformence of Reinforcement Learning (RL) algorithms (Dietterich, 2000; Dean & Lin, 1995; Parr, 1998; Hauskrecht et al., 1998), and indeed there is a growing interest in developing hierarchical DRL algorithms. For example, Tessler et al. (2016) pre-trained skill networks using DQNs and developed a Hierarchical DRL Network (H-DRLN). Their architecture learned to control between options operating at different temporal scales and demonstrated superior performance over the vanilla DQN in solving tasks at Minecraft. Kulkarni et al. (2016) took a different approach, they manually pre-defined sub-goals for a given task and developed a hierarchical DQN (h-DQN) that is operating at different time scales. This architecture managed to learn how to solve both the sub-goals and the original task and outperformed the Vanilla DQN in the the challenging ATARI game 'Montezuma's Revenge'. Both these methods used prior knowledge about the hierarchy of a task in order to solve it. However it is still unclear how to automatically discover the hierarchy in a specific domain a-priori.

Interpretability of DQN policies is an urging issue that has many important applications. For example, it may help to distil a cumbersome model into a simple one (Rusu et al., 2015) and will increase the human confidence in the performance of DRL agents. By understanding what the agent has learned we can also decide where to grant it control and where to take over. Finally, we can improve learning algorithms by finding their weaknesses.

The internal model principle (Francis & Wonham, 1975) states that every good solution to a control problem must be a model of the problem it solves ("Every good key must be a model of the lock it opens"). This line of thought has an interesting application to control theory and biology (Yi et al., 2000). It suggests that to do the best job of reg-

* These authors have contributed equally

ulating some system, a control apparatus should include a model of that system. Sontag (2003) formulated these ideas mathematically for linear and non linear control systems, claiming that if a system Σ is solving a control task under reasonable technical assumptions, then Σ must necessarily contain a subsystem which is capable of predicting the dynamics of the system. In this work we follow the same line of thought. We claim that DQNs are learning an underlying Semi Markov Decision Process (SMDP) of a problem, without implicitly being asked.

Zahavy et al. (2016) showed that by using hand-crafted features, they can interpret the policies learned by DQN agents using a manual inspection of a t-Distributed Stochastic Neighbor Embedding (t-SNE) map (Van der Maaten & Hinton, 2008). They also revealed that DQNs are automatically learning temporal representations such as hierarchical state aggregation and temporal abstractions. On the other hand, they use a manual reasoning of a t-SNE map, a tedious process that requires careful inspection as well as an experienced eye.

However, we suggest a method that is fully automatic. Instead of manually designing features, we use clustering algorithms to reveal the underlying structure of the t-SNE map. But instead of naively applying classical methods, we designed novel time-aware clustering algorithms that take into account the temporal structure of the data. Using this approach we are able to automatically reveal the underlying dynamics and rediscover the temporal abstractions showed in (Zahavy et al., 2016). Moreover, we show that our method reveals an underlying SMDP model and confront this hypothesis qualitatively, by designing a novel visualization tool, and quantitatively, by developing likelihood criteria which we later test empirically.

The result is an SMDP model that gives a simple explanation on how the agent solves the task - by decomposing it automatically into a set of sub-problems and learning a specific skill at each. Thus, we claim that we have found an internal model in DQN's representation, which can be used for automatic sub-goal detection in future work.

2. Background

We briefly review the standard reinforcement learning framework of discrete-time, finite Markov decision processes (MDPs). In this framework, the goal of an RL agent is to maximize its expected return by learning a policy $\pi : S \to \Delta_A$ which is a mapping from states $s \in S$ to a probability distribution over some action space A. At time t the agent observes a state $s_t \in S$, selects an action $a_t \in A$, and receives a reward r_t. Following the agents action choice, it transitions to the next state $s_{t+1} \in S$. We consider infinite horizon problems where the cumulative return at time t is given by $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$, and $\gamma \in [0, 1]$ is the discount factor. The action-value function $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$ represents the expected return after observing state s, taking action a after which following policy π. The optimal action-value function obeys a fundamental recursion known as the optimal Bellman equation,

$$Q^*(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')\right]$$

Deep Q Networks: The DQN algorithm (Mnih et al., 2015) approximates the optimal Q function using a CNN, by optimizing the network weights such that the expected TD error of the optimal Bellman equation is minimized:

$$\mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \left\| Q_\theta\left(s_t, a_t\right) - y_t \right\|_2^2 \tag{1}$$

where

$$y_t = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q_{\theta_{target}}\left(s_{t+1}, a^{'}\right) & \text{otherwise} \end{cases} .$$

Notice that this is an offline learning algorithm, meaning that the tuples $\{s_t, a_t, r_t, s_{t+1}, \gamma\}$ are collected from the agents experience and are stored in the **Experience Replay (ER)** (Lin, 1993). The ER is a buffer that stores the agents experiences at each time-step t, for the purpose of ultimately training the DQN parameters to minimize the loss function. When we apply mini-batch training updates, we sample tuples of experience at random from the pool of stored samples in the ER. The DQN maintains two separate Q-networks. The current Q-network with parameters θ, and the target Q-network with parameters θ_{target}. The parameters θ_{target} are set to θ every fixed number of iterations. In order to capture the game dynamics, the DQN represents the state by a sequence of image frames.

Skills, Options, Macro-actions (Sutton et al., 1999), are temporally extended control structure, denoted by σ. A skill is defined by a triple $\sigma =< I, \pi, \beta >$ where I is the set of states where the skill can be initiated, π is the intra-skill policy, which determines how the skill behaves when encountering states, and β is the set of termination probabilities determining when a skill will stop executing. β is typically either a function of state s or time t. Any MDP with a fixed set of skills is a **Semi-Markov Decision Process (SMDP)**. Planning with skills can be performed by predicting for each state in the skill initiation set I, the state in which the skill will terminate and the total reward received along the way. More formally, an SMDP can be defined by a five-tuple $< S, \Sigma, P, R, \gamma >$ where S is a set of states, Σ is a set of skills, and P is the transition probability kernel.

$$R_s^\sigma = \mathbb{E}[r_s^\sigma] = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} | s_t = s, \sigma] \tag{2}$$

represents the expected discounted sum of rewards received during the execution of a skill σ initialized from a state s, and $\gamma \in [0,1]$ is the discount factor. The **Skill Policy** $\mu : S \to \Delta_\Sigma$ is a mapping from states to a probability distribution over skills Σ. The action-value function $Q : S \times \Sigma \to R$ represents the long-term value of taking a skill $\sigma \in \Sigma$ from a state $s \in S$ and thereafter always selecting skills according to policy μ and is defined by $Q(s,\sigma) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t | (s,\sigma), \mu]$. We denote the skill transition probability as

$$P_{s,s'}^\sigma = \sum_{j=0}^{\infty} \gamma^j Pr[k = j, s_{t+j} = s' | s_t = s, \sigma]$$

Under these definitions the optimal skill value function is given by the following equation (Stolle & Precup) as:

$$Q_\Sigma^*(s,\sigma) = \mathbb{E}[R_s^\sigma + \gamma^k \max_{\sigma' \in \Sigma} Q_\Sigma^*(s', \sigma')] \ . \tag{3}$$

3. Methodology

For each domain:

1. **Learn :** Train a DQN agent.
2. **Evaluate :** Run the agent, record visited states, neural activations and Q-values.
3. **Reduce :** Apply t-SNE on the neural activation to obtain a low dimensional representation.
4. **Cluster :** Apply clustering on the data.
5. **Model :** Fit an SMDP model. Estimate the transition probabilities and reward values.
6. **Visualize :** Visualize the SMDP on top of the t-SNE map.

The rest of the paper is organized as follows: Section 4 explains how we create the t-SNE maps from raw pixel states using the DQN algorithm. In section 5 we present the clustering methods that we developed and the quantitative evaluation criteria. Section 6 explains our visualization method and Section 7 presents examples for Atari2600 games using our method. Finally Section 8 summarizes our work.

4. From DQN to t-SNE

We train DQN agents using the Vanilla DQN algorithm (Mnih et al., 2015). When training is done, we evaluate the agent at multiple episodes, using an ϵ-greedy policy. We record all visited states and their neural activations, as well as the Q-values and other manually extracted features. We keep the states in their original visitation order in order to maintain temporal relations. Since the neural activations are of high order we apply t-SNE dimensionality reduction so we are able to visualize it.

t-SNE is a visualization technique for high dimensional data that assigns each data point a location in a two or three-dimensional map. It has been proven to outperform linear dimensionality reduction methods and non-linear embedding methods such as ISOMAP (Tenenbaum et al., 2000) in several research fields including machine-learning benchmark datasets and hyper-spectral remote sensing data (Lunga et al., 2014). At its core, The t-SNE algorithm defines two similarity measures from the euclidean distances between points. The first measure: $p_{i,j}$ is defined over pairs of points x_i, x_j in the high dimension space using a Gaussian distribution:

$$\begin{aligned} p_{j|i} &= \frac{exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \\ p_{i,j} &= \frac{p_{i|j} + p_{j|i}}{2} \end{aligned} \tag{4}$$

The second measure $q_{i,j}$ is defined over pairs of points y_i, y_j in the desired low-dimension space using a Student-t distribution:

$$q_{i,j} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_j\|^2)^{-1}} \tag{5}$$

The algorithm defines a cost function between the measures:

$$Cost = KL(P||Q) = \sum_{i,j} p_{i,j} log \frac{p_{i,j}}{q_{i,j}} \tag{6}$$

And minimize it by following a gradient descent approach:

$$\begin{aligned} \frac{\delta Cost}{\delta y_i} &= 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \\ Y^t &\leftarrow Y^{t-1} + \eta \frac{\delta Cost}{\delta Y} + \alpha(t)(Y^{t-1} - Y^{t-2}) \end{aligned} \tag{7}$$

The technique is relatively easy to optimize, and reduces the tendency to crowd points together in the center of the map by using the heavy tailed Student-t distribution (Equation 5) in the low dimensional space. It is known to be particularly good at creating a single map that reveals structure at many different scales, which is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds. Therefore we can visualize the different sub-manifolds learned by the network and interpret their meaning.

5. From t-SNE to SMDP

In this section we explain the clustering methods which we developed for this work and show how to create an SMDP model. We define an SMDP model over the set of t-SNE points using a vector of cluster labels C, and a transition probability matrix P where $P_{i,j}$ indicates the empirical probability of moving from cluster i to j. We define the entropy of a model by: $e = -\sum_i \{|C_i| \cdot \sum_j P_{i,j} \log P_{i,j}\}$, i.e., the average entropy over transition probability from each cluster weighted by its size.

We note that threw this entire paper, by an SMDP model we only refer to the induced Markov Reward Process of the DQN policy. Recall that the DQN agent is learning a deterministic policy, therefore, in deterministic environments (e.g., the Atari2600 emulator), the underlying SMDP should in fact be deterministic and an entropy minimizer.

The data that we collect from the DQN agent is highly correlated since it was generated from an MDP. However, standard clustering algorithms assume the data is drawn from an i.i.d distribution, and therefore result with clusters that overlook the temporal information. This results with high-entropy SMDP models that are too complicated to analayze and are not consistent with the data. For this aim, we propose 3 clustering methods that incorporate the data temporal information. We present two variants of K-means and a variant of hierarchical clustering.

5.1. K-means based methods

K-means (MacQueen et al., 1967) is a method commonly used to automatically partition a data set into k groups. Given a set of observations $(x_1, x_2, \cdots, x_n)$, where each observation is a d-dimensional real vector, K-means clustering aims to partition the n observations into $k(\leq n)$ sets $C = (C_1, C_2, \cdots, C_k)$ so as to minimize the within-cluster sum of squares:

$$\underset{\mathbf{C}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{8}$$

where μ_i is the mean of points in C_i. It proceeds by selecting k initial cluster centers and then iteratively refining them as follows:

1. **Assignment step,** each observation x_i is assigned to its closest cluster center:

$$C_i^{(t)} = \{x_p : \|x_p - \mu_i^{(t)}\|^2 \leq \|x_p - \mu_j^{(t)}\|^2 \\ \forall j, 1 \leq j \leq k\}. \tag{9}$$

2. **Update step,** each cluster center μ_j is updated to be the mean of its constituent instances:

$$\mu_i^{(t+1)} = \frac{1}{|C_i^{(t)}|} \sum_{x_j \in C_i^{(t)}} x_j.$$

The algorithm converges when there is no further change in the assignment of instances to clusters.

Spatio-Temporal Cluster Assignment.
Our first K-means variant modifies the assignment step in the vanilla K-means algorithm to include temporal information. Here, each observation x_p is a t-SNE point with time index p along the trajectory. The K-means algorithm, presented with the coordinates of the points, takes care of clustering points with spatial proximity. In order to encourage temporal coherency, we modify the assignment step in the following way:

$$C_i^{(t)} = \{x_p : \|X_{p-w:p+w} - \mu_i^{(t)}\|^2 \leq \|X_{p-w:p+w} - \mu_j^{(t)}\|^2, \\ \forall j, 1 \leq j \leq k\} \tag{10}$$

Where $X_{p-w:p+w}$ is the set of $2w$ t-SNE points before and after x_p along the trajectory. In this way, a point x_p is assigned to a cluster μ_j, if its neighbours along the trajectory are also close to μ_j.

Entropy Regularization Cluster Assignment.
In order to create simpler models, we suggest to add an entropy regularization term for the K-mean assignment step:

$$C_i^{(t)} = \{x_p : \|x_p - \mu_i^{(t)}\|^2 + d \cdot e_{x_p \to i}^{t-1} \leq \\ \|x_p - \mu_j^{(t)}\|^2 + d \cdot e_{x_p \to j}^{t-1}, \\ \forall j, 1 \leq j \leq k\}. \tag{11}$$

Where d is the penalty weight, and $e_{x_p \to i}^{t-1}$ indicates the entropy gain of changing x_p assignment to cluster i in the SMDP obtained at iteration $t-1$. This is equivalent to minimizing an energy function which is the sum of the K-means objective function (Equation 8) and an entropy term.

5.2. Agglomerative clustering approach

Our third method for creating an SMDP model is a variant of hierarchical clustering. Agglomerative clustering is a bottom-up hierarchical approach. It begin when each observation forms a unique cluster. Then, pairs of clusters are merged together so as to minimize some linkage criteria. Most popular here are the single-linkage criteria: $c(A,B) = \min_{a,b}\{|x_a - x_b\| : a \in A, b \in B\}$, and the complete-linkage criteria: $c(A,B) = \max_{a,b}\{|x_a - x_b\| : a \in A, b \in B\}$. In order to encourage temporal coherency in cluster assignments we form a new linkage criteria based on ward's (Ward, 1963) criteria:

$$c(A, B) = (1-\lambda) \cdot mean\{\|x_a - x_b\| : a \in A, b \in B\} + \lambda \cdot e_{\{A,B\} \to AB} \quad (12)$$

where $e_{\{A,B\} \to AB}$ measures the difference between the entropy of the corresponding SMDP before and after merging clusters A, B.

5.3. Evaluation criteria

We follow the analysis of (Hallak et al., 2013) and define criteria to measure the fitness of a model empirically. We define the **Value Mean Square Error(VMSE)** as the normalized distance between two value estimations:

$$\text{VMSE} = \frac{\|v^{DQN} - v^{SMDP}\|}{\|v^{DQN}\|}.$$

The SMDP value is given by

$$V_{SMDP} = (I + \gamma^k P)^{-1} r \quad (13)$$

and the DQN value is evaluated by averaging the DQN value estimates over all MDP states in a given cluster (SMDP state): $v^{DQN}(c_j) = \frac{1}{|C_j|} \sum_{i: s_i \in c_j} v^{DQN}(s_i)$. Finally, the greedy policy with respect to the SMDP value is given by:

$$\pi_{greedy}(c_i) = \underset{j}{\text{argmax}}\{R_{\sigma_{i,j}} + \gamma^{k_{\sigma_{i,j}}} v_{SMDP}(c_j)\} \quad (14)$$

The **Minimum Description Length** (MDL; (Rissanen, 1978)) principle is a formalization of the celebrated Occams Razor. It copes with the over-fitting problem for the purpose of model selection. According to this principle, the best hypothesis for a given data set is the one that leads to the best compression of the data. Here, the goal is to find a model that explains the data well, but is also simple in terms of the number of parameters. In our work we follow a similar logic and look for a model that best fits the data but is still simple.

Instead of considering "simple" in terms of the number of parameters, we measure the simplicity of the spatio-temporal state aggregation. For spatial simplicity we define the Inertia: $I = \sum_{i=0}^{n} \min_{\mu_j \in C}(||x_j - \mu_i||^2)$ which measures the variance of MDP states inside a cluster (AMDP state). For temporal simplicity we define the entropy: $e = -\sum_i \{|C_i| \cdot \sum_j P_{i,j} \log P_{i,j}\}$, and the *Intensity Factor* which measures the fraction of in/out cluster transitions: $F = \sum_j \frac{P_{jj}}{\sum_i P_{ji}}$.

6. Visualization: fusing SMDP with t-SNE

In Section 4 we explained how to create a t-SNE map from DQN's neural activations and in Section 5 we showed how to automatically design an SMDP model using temporal-ware clustering methods. In this section we explain how to fuse the SMDP model with the t-SNE map for a clear visualization of the dynamics.

In our approach, an SMDP is represented by a directed graph. Each SMDP state is represented by a node in the graph and corresponds to a cluster of t-SNE points (game states). In addition, the transition probabilities between the SMDP states are represented by weighted edges between the graph nodes. We draw the graph on top of the t-SNE map such that it reveals the underlying dynamics. Choosing a good layout mechanism to represent a graph is a hard task when dealing with high dimensional data (Tang et al., 2016). We consider different layout algorithms for the position of the nodes, such as the spring layout that position nodes by using the Fruchterman-Reingold force-directed algorithm and the spectral layout that uses the eigenvectors of the graph Laplacian (Hagberg et al., 2008). However, we found out that simply positioning each node at the average coordinates of each t-SNE cluster gives a more clear visualization. The intuition behind it is that the t-SNE algorithm was planned to solve the crowding problem and therefore outputs clusters that are well separated from each other.

Finally, each node in the graph is represented by its centroid. For example, if each state is an image, then a node is represented using the mean or median images. Another approach is to represent a node using its most significant features. In this approach each node is annotated with a few features that are considered most distinct, e.g., by the feature with lowest variance in the cluster.

7. Experiments

Experimental set-up. We evaluated our method on two Atari2600 games, Breakout and Pacman. For each game we collected 120k game states. We apply the t-SNE algorithm directly on the collected neural activations of the last hidden layer, similar to Mnih et al. (2015). The input $X \in \mathbf{R}^{120k \times 512}$ consists of $120k$ game states with 512 features each (the size of our DQN last layer). Since this data is relatively large, we pre-processed it using Principal Component Analysis to dimensionality of 50 and used the Barnes Hut t-SNE approximation (Van Der Maaten, 2014). All experiments were performed with 3000 iterations perplexity of 30. The input $X \in \mathbf{R}^{120k \times 3}$ to the clustering algorithm consists of $120k$ game states with 3 features each (two t-SNE coordinates and the Value estimate). We applied the Spatio-Temporal Cluster Assignment with *k=20* clusters and *w=2* temporal window size (Equation 10). We run the algorithm for 160 iterations and choose the best SMDP in terms of minimum entropy (we will consider other measures in future work). Finally we visualize the SMDP using the visualization method explained in Section 6.

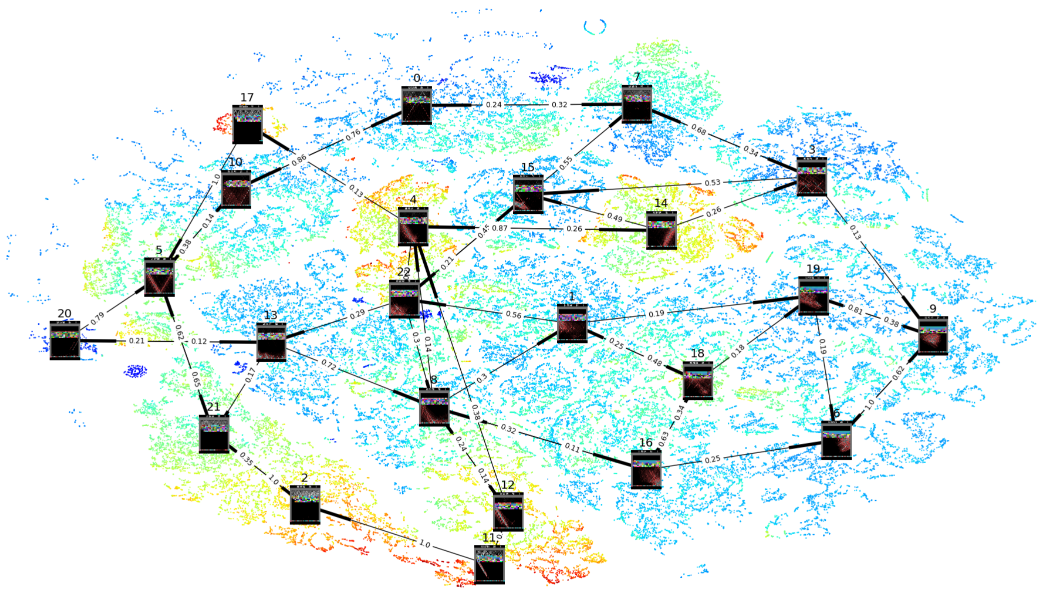

Figure 1. SMDP visualization for Breakout.

Simplicity. Looking at the resulted SMDPs it is interesting to note that the transition probability matrix is very sparse, i.e., the transition probability from each state is not zero only for a small subset of the states, thus, indicating that our cluster are located in time. Inspecting the mean image of each cluster we can see that the clusters are also highly spatially located, meaning that the states in each cluster share similar game position.

Figure 1 shows the SMDP for **Breakout**. The mean image of each cluster shows us the ball location and direction (in red), thus characterizes the game situation in each cluster. We also observe that states with low entropy follow a well defined skill policy. For example cluster 10 has one main transition ans show a well defined skill of carving the left tunnel (see the mean image). In contrast, clusters 6 and 16 has transitions to more clusters (and therefore higher entropy) and a much less defined skill policy (presented by its relatively confusing mean state).

Figure 2 shows the SMDP for **Pacman**. The mean image of each cluster shows us the agent's location (in blue), thus characterizes the game situation in each cluster. We can see that the agent is spending its time in a very defined areas in the state space at each cluster. For example, cluster 19 it is located in the north-west part of the screen and in cluster 9 it is located in south-east. We also observe that clusters with more transitions, e.g., clusters 0 and 2, suffer from less defined mean state.

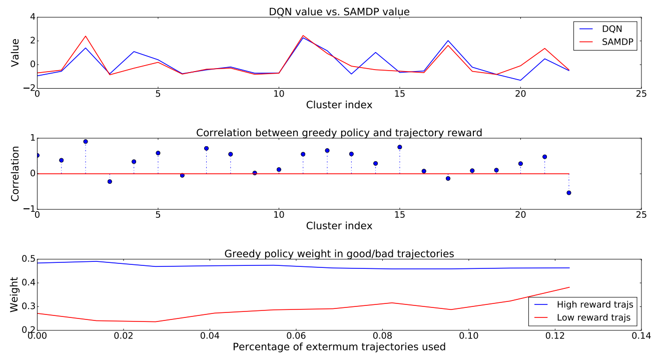

Figure 3. **Model Evaluation. Top:** Value function consistency. **Center:** greedy policy correlation with trajectory reward. **Bottom:** top (blue), least (red) rewarded trajectories.

Model Evaluation. We evaluate our model using three different methods. First, the VMSE criteria (Figure 3, top): high correlation between the DQN values and the SMDP values gives a clear indication to the fitness of the model to the data. Second, we evaluate the correlation between the transitions induced by the policy improvement step and the trajectory reward R^j. To do so, we measure P_i^j : the empirical distribution of choosing the greedy policy at state c_i in that trajectory. Finally we present the correlation coefficients at each state: $corr_i = corr(P_i^j, R^j)$ (Figure 3, center). Positive correlation indicates that following the greedy policy leads to high reward. Indeed for most of the states we observe positive correlation, supporting the consistency

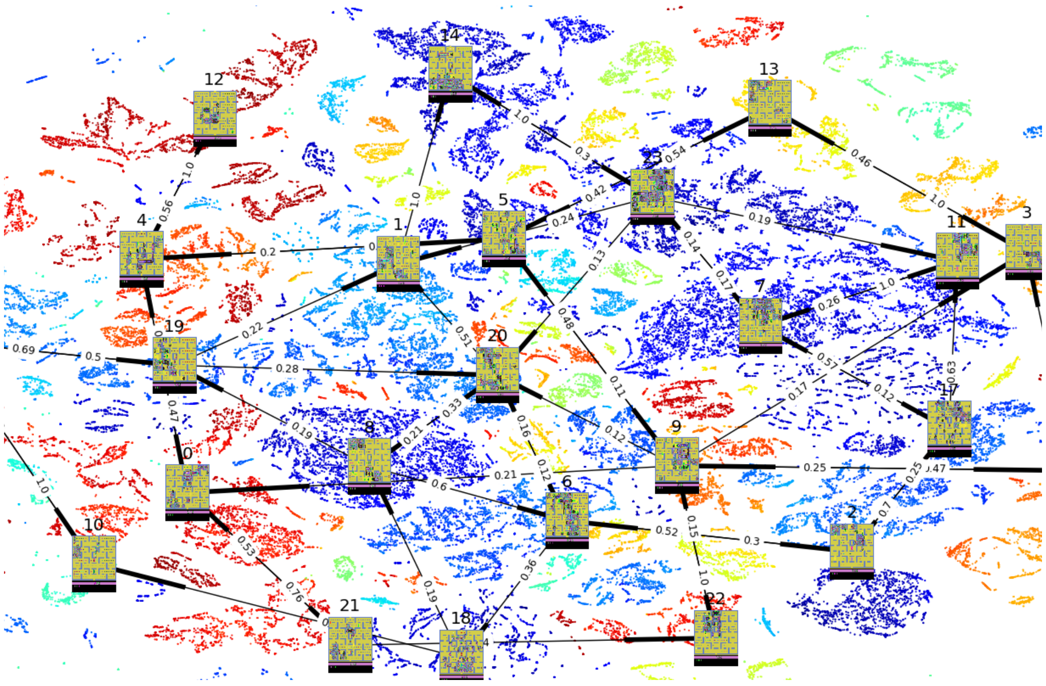

Figure 2. SMDP visualization for Pacman.

of the model. The third evaluation is close in spirit to the second one. We create two transition matrices T^+, T^- using k top-rewarded trajectories and k least-rewarded trajectories respectively. We measure the correlation of the greedy policy T^G with each of the transition matrices for different values of k (Figure 3 bottom). As clearly seen, the correlation of the greedy policy and the top trajectories is higher than the correlation with the bad trajectories.

8. Discussion

In this work we considered the problem of visualizing dynamics. Starting with a t-SNE map of the neural activations of a DQN and ending up with an SMDP model describing the underlying dynamics. We developed clustering algorithms that take into account the temporal aspects of the data and defined quantitative criteria to rank candidate SMDP models based on the likelihood of the data and an entropy simplicity term. Finally we showed in the experiments section that our method can successfully be applied on two Atari2600 benchmarks, resulting in a clear interpretation for the agent policy.

Our method is fully automatic and does nor require any manual or game specific work. We note that this is a work in progress, it is mainly missing the quantitative results for the different likelihood criteria. In future work we will finish to implement the different criteria followed by the relevant simulations.

References

Dean, Thomas and Lin, Shieu-Hong. Decomposition techniques for planning in stochastic domains. 1995.

Dietterich, Thomas G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000.

Duda, Richard O, Hart, Peter E, and Stork, David G. *Pattern classification*. John Wiley & Sons, 2012.

Erhan, Dumitru, Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep*, 4323, 2009.

Francis, Bruce A and Wonham, William M. The internal model principle for linear multivariable regulators. *Applied mathematics and optimization*, 2(2), 1975.

Hagberg, Aric A., Schult, Daniel A., and Swart, Pieter J. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, August 2008.

Hallak, Assaf, Di-Castro, Dotan, and Mannor, Shie. Model selection in markovian processes. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013.

Hauskrecht, Milos, Meuleau, Nicolas, Kaelbling, Leslie Pack, Dean, Thomas, and Boutilier, Craig. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 220–229. Morgan Kaufmann Publishers Inc., 1998.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Kulkarni, Tejas D, Narasimhan, Karthik R, Saeedi, Ardavan, and Tenenbaum, Joshua B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

Lin, Long-Ji. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

Lunga, Dalton, Prasad, Santasriya, Crawford, Melba M, and Ersoy, Ozan. Manifold-learning-based feature extraction for classification of hyperspectral data: a review of advances in manifold learning. *Signal Processing Magazine, IEEE*, 31(1):55–66, 2014.

MacQueen, James et al. Some methods for classification and analysis of multivariate observations. 1967.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.

Parr, Ronald. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 422–430. Morgan Kaufmann Publishers Inc., 1998.

Rissanen, Jorma. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Rusu, Andrei A, Colmenarejo, Sergio Gomez, Gulcehre, Caglar, Desjardins, Guillaume, Kirkpatrick, James, Pascanu, Razvan, Mnih, Volodymyr, Kavukcuoglu, Koray, and Hadsell, Raia. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Sontag, Eduardo D. Adaptation and regulation with signal detection implies internal model. *Systems & control letters*, 50(2):119–126, 2003.

Stolle, Martin and Precup, Doina. Learning options in reinforcement learning. Springer.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1), August 1999.

Tang, Jian, Liu, Jingzhou, Zhang, Ming, and Mei, Qiaozhu. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016.

Tenenbaum, Joshua B, De Silva, Vin, and Langford, John C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500), 2000.

Tessler, Chen, Givony, Shahar, Zahavy, Tom, Mankowitz, Daniel J, and Mannor, Shie. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Van Der Maaten, Laurens. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

Van der Maaten, Laurens and Hinton, Geoffrey. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Ward, Joe H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

Yi, Tau-Mu, Huang, Yun, Simon, Melvin I, and Doyle, John. Robust perfect adaptation in bacterial chemotaxis through integral feedback control. *Proceedings of the National Academy of Sciences*, 97(9):4649–4653, 2000.

Yosinski, Jason, Clune, Jeff, Bengio, Yoshua, and Lipson, Hod. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pp. 3320–3328, 2014.

Zahavy, Tom, Zrihem, Nir Ben, and Mannor, Shie. Graying the black box: Understanding dqns. *arXiv preprint arXiv:1602.02658*, 2016.

Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pp. 818–833. Springer, 2014.