Graying the black box: Understanding DQNs

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

Abstract

In recent years there is a growing interest in using deep representations for reinforcement learning. In this paper, we present a methodology and tools to analyze Deep Q-networks (DQNs) in a non-blind matter. Using our tools we reveal that the features learned by DQNs aggregate the state space in a hierarchical fashion, explaining its success. Moreover we are able to understand and describe the policies learned by DQNs for three different Atari2600 games and suggest ways to interpret, debug and optimize of deep neural networks in Reinforcement Learning.

1. Introduction

In the reinforcement learning (RL) paradigm, an agent autonomously learns from experience in order to maximize some reward signal. Learning to control agents directly from high-dimensional inputs like vision and speech is a hard problem in RL, known as the curse of dimensionality. Countless solutions to this problem have been offered including linear function approximators (Tsitsiklis & Van Roy, 1997), hierarchical representations (Dayan & Hinton, 1993) state aggregation (Singh et al., 1995) and options (Sutton et al., 1999). These methods rely upon engineering problem specific state representations, thus making the agent not fully autonomous and reducing its flexibility. Therefore, there is a growing interest in using non-linear function approximators, that are more general and require less domain specific knowledge, e.g., TD-gammon (Tesauro, 1995). Unfortunately, such methods are known to be unstable or even to diverge when used to represent the action-value function (Tsitsiklis & Van Roy, 1997; Gordon, 1995; Riedmiller, 2005). The Deep Q-Network (DQN) algorithm (Mnih et al., 2015) increased training stability by introducing the target network and by using Experience Replay (ER) (Lin, 1993). Its promise was demonstrated in the Arcade Learning Environment (ALE) (Bellemare et al., 2012), a challenging framework composed of dozens of Atari games used to evaluate general competency in AI. It achieved dramatically better results than earlier approaches, showing a robust ability to learn representations.

While using Deep Learning (DL) in RL looks promising, there is no free lunch. Training a Deep Neural Network (DNNs) is a complex optimization process. At the inner level we fix a hypothesis class and learn it using some gradient descent algorithm. The optimization of the outer level addresses the choice of network architecture, setting hyper-parameters, and even the choice of optimization algorithm. In Deep Reinforcement Learning (DRL) we also need to choose how to model the environment as a Markov Decision Process (MDP), i.e., to choose the discount factor, the amount of history frames that represent a state, and to choose between the various algorithms and architectures (Nair et al., 2015; Van Hasselt et al., 2015; Schaul et al., 2015; Wang et al., 2015; Bellemare et al., 2015). While the optimization of the inner level is analytic to a high degree, the optimization of the outer level is far from being so. Currently, most practitioners tackle this problem either using a trial and error methodology, or by exhaustively searching over the space of possible configurations (moreover, the lack of interpretability regarding the success or failure of a DNN makes it a manually tedious task and diminishes it's wide use).

A key issue in RL is that of representation, i.e., allowing the agent to locally generalize the policy across similar states. Unfortunately, spatially similar states often induce different control rules in contrast to other machine learning benchmarks that enjoy local generalization. As a consequence, learning a policy (or a value function) using a function approximator is often hindered by discontinuities in the state. For example, in the Atari2600 game Seaquest, whether or not a diver has been collected (represented by a few pixels) controls the outcome of the submarine surface action (without-loose a life, with-fill air). Another cause for these discontinuities is that for a given problem, two states with

* These authors have contributed equally

similar representations may in fact be far from each other in terms of the number of state transitions required to reach one from the other. This observation can also explain the lack of pooling layers in DRL architectures(e.g., Mnih et al. (2015; 2013); Levine et al. (2015)).

Thus, methods that focus on the temporal structure of the policy has been proposed. Such methods decompose the learning task into simpler subtasks using graph partitioning (Menache et al., 2002; Mannor et al., 2004; Şimşek et al., 2005) and path processing mechanisms (Stolle, 2004; Thrun, 1998). Given a good representation of the states, varying levels of convergence guarantees has been promised (Dean & Lin, 1995; Parr, 1998; Hauskrecht et al., 1998; Dietterich, 2000).

In this work we analyze the representation of states learned by DRL agents. We argue that DQN is learning a hierarchical spatio-temporal aggregation of the state space using different sub manifolds to represent the policy thus explaining its success. We show that by doing so we can give a qualitative understanding of learned policies that can help the outer optimization loop and help in the debugging process. Our methodology is to extract the neural activations of the last DQN hidden layer, while it is playing the game, and then apply t-Distributed Stochastic Neighbor Embedding (t-SNE) (Van der Maaten & Hinton, 2008) for dimensionality reduction and visualization. On top of this low dimensional representation we display the dynamics, different game measures and hand crafted features, so that we are able to describe what each sub-manifold represents. We also use saliency maps to analyze the influence of different features on the network. In particular, our main contributions are the following:

- We suggest an explanation for the success of DQN based on its learned representation, by showing that DQN is mapping states to sub-manifolds with mutual spatial and temporal structure and learning specific policies at each cluster, so it can tackle state discontinuities.
- We give an interpretation for the agents policy in a clear way, thus allowing to understand what are its weaknesses and strengths.
- We propose methods for debugging and reducing the hyper parameters grid search of DRL and give examples on game modeling, termination and initial states, score over-fitting, etc.

2. Related work

The goal behind visualization of DNN is to give better understanding of these inscrutable black boxes. While some approaches study the type of computation performed at each layer as a group (Yosinski et al., 2014), others try to explain the function computed by each individual neuron (similar to the "Jennifer Aniston Neuron" Quiroga et al. (2005)). Dataset-centric approaches display images from the training or test set that cause high or low activations for individual units, e.g., the deconvolution method (Zeiler & Fergus, 2014) highlights the portions of a particular image that are responsible for the firing of each neural unit. Network-centric approaches investigate a network directly without any data from a dataset, e.g., Erhan et al. (2009) synthesized images that cause high activations for particular units. Other works used the input gradient to find images that cause strong activations (e.g., Simonyan & Zisserman (2014); Nguyen et al. (2014); Szegedy et al. (2013)).

Since RL research was mostly focused on linear function approximations and policy gradient methods, we are less familiar with visualization techniques and attempts to understand the structure learnt by an agent. Wang *et al.* (Wang et al., 2015) suggested to use saliency maps and analyzed which pixels are more active at network predications. Using this method they compared between the standard DQN and their dueling network architecture. Engel & Mannor (2001) learned an embedded map of Markov processes and visualized it on 2 dimensions, their analysis is based on the state transition distribution while we will focus on distances between the features learned by DQN.

3. Background

This section includes relative background on RL, DQN and t-SNE.

3.1. Reinforcement Learning

The goal of RL agents is to maximizes its expected total reward by learning an optimal policy (mapping from states to actions). At time t the agent observes a state s_t, selects an action a_t, and receives a reward r_t, following the agents decision it observes the next state s_{t+1} . We consider infinite horizon problems where the cumulative return is discounted by a factor of $\gamma \in [0, 1]$ and the return at time t is given by $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$, where T is the termination step. The action-value function $Q^\pi(s, a)$ measures the expected return after observing state s_t and taking an action under a policy π, $Q(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$. The optimal action-value obeys a fundamental recursion known as the Bellman equation,

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

3.2. Deep Q Networks

The DQN algorithm (Mnih et al., 2015; 2013) approximates the optimal Q function with a Convolutional Neural

Network (CNN), by optimizing the network weights such that the expected TD error of the optimal bellman equation is minimized:

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

$$y_t = \begin{cases} 0 & s_t \text{ is terminal} \\ \gamma \left[r_t + \max\limits_{a'} Q_{\theta_{target}}\left(s_{t+1}, a'\right) \right] & \text{else} \end{cases}$$

Notice that this is an off-line algorithm, meaning that the tuples $\{s_t, a_t, r_t, s_{t+1}, \gamma\}$ are collected from the agents experience, stored in the ER and later used for training. The DQN maintains two separate Q-networks with current parameters θ and target parameters θ_{target} that are updated from θ every fixed number of iterations. In order to capture the game dynamics, DQN represents a state by a sequence of history frames and pads initial states with zero frames.

3.3. t-SNE

t-SNE is a visualization technique for high dimensional data by giving 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). The technique is relatively easy to optimize, and reduces the tendency to crowd points together in the center of the map by employing a heavy tailed Student-t distribution 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.

4. Methods

Our tool is displayed in Figure 1. It contains a visualization of the t-SNE map (Left), tools for visualizing global features (top right), tools for visualizing game specific features (middle right), and visualization of the chosen state and its corresponding gradient image (bottom right).

4.1. t-SNE

To produce the t-SNE figures, we train a DQN for each Atari2600 game and then let it play with an epsilon greedy scheme. We collect 120k game states, for each we store the neural activations of the last hidden layer, and apply the Barnes Hut t-SNE (Van Der Maaten, 2014) which is useful for large data sets. We also pre-process the data using

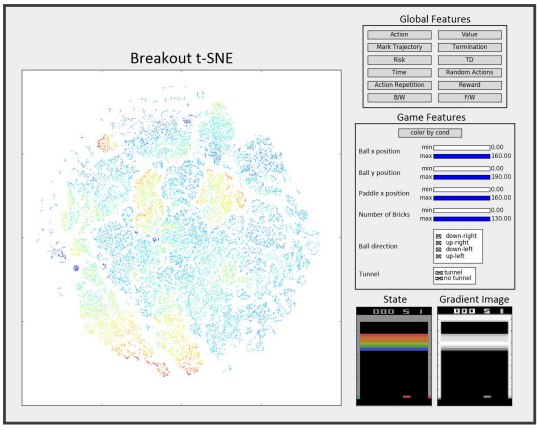

Figure 1. Graphical user interface for our methodology.

Principal Component Analysis to dimensionality of 50. All experiments were performed with 3000 iterations perplexity of 30.

4.2. Coloring the t-SNE map with different measures

We assign each state with its measure such as value estimates (value, Q, advantage), generation time, termination, (decide how too show the gui and detail all measure we use), and use it to color the t-SNE map. We also extract hand-crafted features, directly from the emulator raw frames, such as player position, enemy position, number of lives, and so on. We use these features to filter points in the t-SNE image. The filtered images reveal insightful patterns that cannot be seen otherwise. From our experience, hand-crafted features are very powerful, however the drawback of using them is that they require manual work. Similar to (Engel & Mannor, 2001) we visualize the dynamics of the learned policy. However we use a 3d t-SNE state representation and display transitions upon it with arrows using Mayavi (Ramachandran & Varoquaux, 2011).

4.3. Saliency maps

We generate Saliency maps, similar to (Simonyan & Zisserman, 2014), by computing the Jacobians of the trained value with respect to the input video. The Jacobians image is presented above the input image itself and helps to understand which pixels in the image affect the value prediction the most.

4.4. Analysis

Using these measures we are able to understand the common attributes of a given cluster (e.g Figure 3 in the apendix). From the analysis of the agent dynamics between

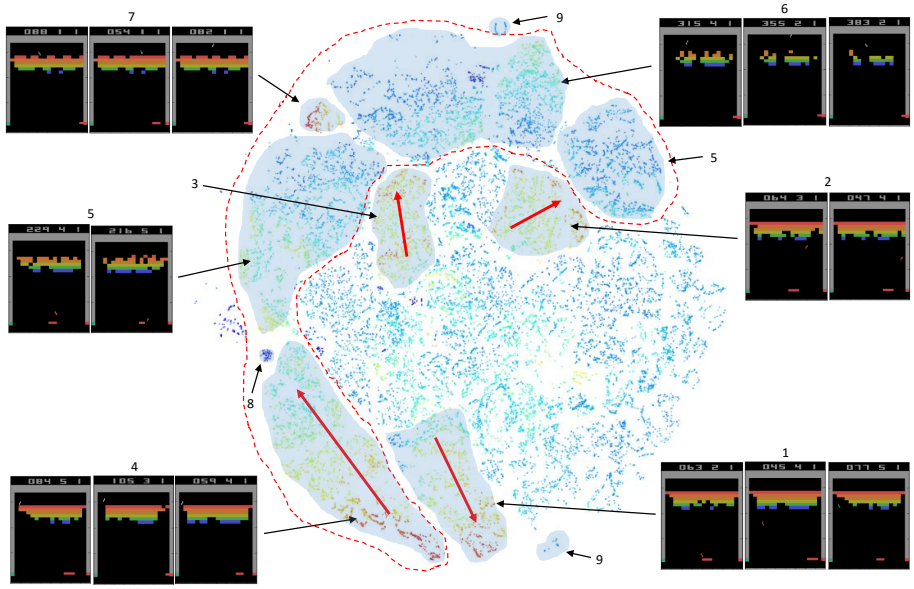

Figure 2. Breakout aggregated states on the t-SNE map.

clusters we can point a hierarchical aggregation of the state space. We define the clusters that has a clear entrance and termination areas as option clusters and we are able to give an interpretation for the agent policy in those clusters. For some options we are able to derive rules for initiation and termination, e.g., landmark options (Mann et al., 2015) has unique termination state.

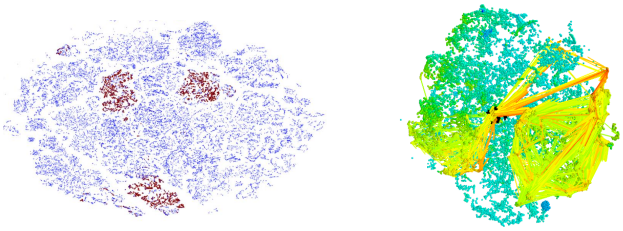

Figure 3. Breakout tunnel digging option. **Left:** states that the agent visits once it entered the option clusters (1-3 in Figure 2) until it finishes to carve the left tunnel are marked in red. **Right:** Dynamics is displayed by arrows above a 3d t-SNE map. The option termination zone is marked by a black annotaion box and corresponds to carving the left tunnel. All transitions from clusters 1-3 into clusters 4-7 pass threw a singular point.

5. Experiments

We applied our methodology on three ATARI games: Breakout, Pacman and Seaquest. For each one we give a short description of the game, analyze the optimal policy, detail the features we designed, interpret the DQN's policy and derive conclusions. We finish by analyzing initial and terminal states and the influence of score pixels.

5.1. Breakout

In Breakout, a layer of bricks lines the top of the screen. A ball travels across the screen, bouncing off the top and side walls of the screen. When a brick is hit, the ball bounces away, the brick is destroyed and the player receives reward based on the bricks layer. The player loses a turn when the ball touches the bottom of the screen. To prevent this from happening, the player has a movable paddle to bounce the ball upward, keeping it in play. The highest score achievable for one player is 896; this is done by eliminating exactly two screens of bricks worth 448 points each.

We extract features for the player (paddle) position, ball's position, balls direction of movement, number of missing bricks, and a tunnel feature. We say that a tunnel exists when there is a clear path between the area below the bricks and the area above it. We approximate this event by looking for at least one clear column of bricks.

A good strategy for Breakout is leading the ball above the bricks by digging a tunnel in one side of the bricks block. Doing so enables the ball to bounce on the bricks achieving reward while the agent is safe from loosing it. By introducing a discount factor to Breakout's MDP, this strategy becomes even better since it achieves high immediate reward.

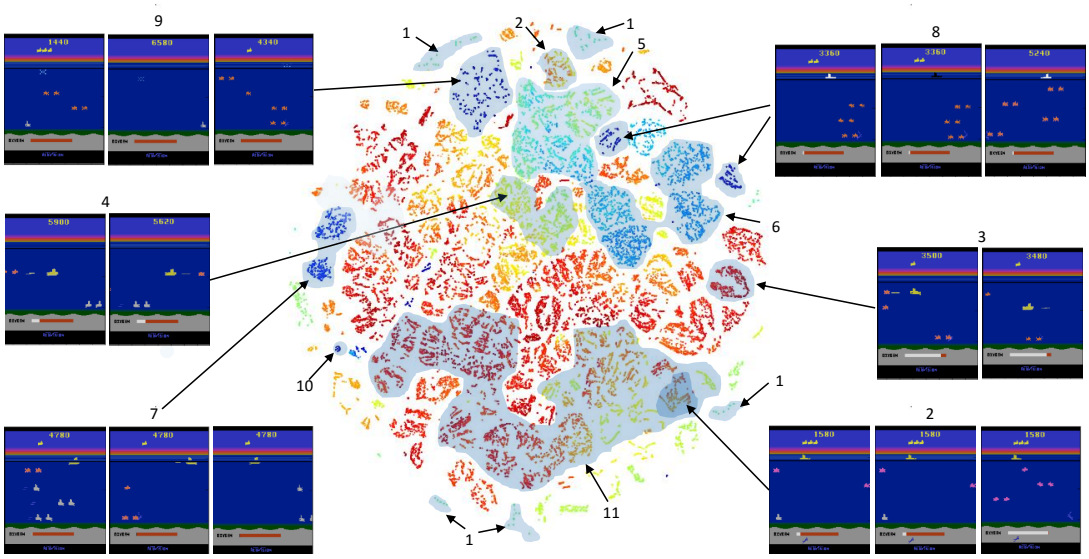

Figure 4. Seaquest aggregated states on the t-SNE map, colored by value function estimate.

Figure 2 presents the t-SNE of Breakout. The agent learns a hierarchical policy: first carve a tunnel in the left side of the screen and then keep the ball above the bricks as long as possible. In clusters (1-3) the agent is carving the left tunnel. Once the agent enters those clusters it will not leave them until the tunnel is carved (see Figure 14), thus, we define them as a landmark option (Mann et al., 2015). The clusters are separated from each other by the ball position and direction. In cluster 1 the ball is heading toward the tunnel, in cluster 2 the ball is at the right side and in cluster 3 the ball bounces back from the tunnel after hitting bricks. As less bricks remain in the tunnel the value is gradually rising till the tunnel is carved and the value is maximal (this makes sense since the agent is enjoying high reward rate straight after reaching it). Once the tunnel is carved, the option is terminated and the agent moves to clusters 4-7 (dashed red line), sorted by the ball position with regard to the bricks (see Figure 2). Again the clusters are distinguished by the ball position and direction. In cluster 4 and 6 the ball is above the bricks and in 5 it is below them. Clusters 8 and 9 represent termination and initial states respectively (See Figure 2 in the appendix for examples of states along the option path).

Cluster 7 is created due to a bug in the emulator that allows the ball to pass the tunnel without completely opening the tunnel, however the agent learned to represent this case to its own cluster and assigned it high value estimates (same as the other tunnel clusters). This observation is interesting since it indicates that the agent is learning a representation based on the game dynamics and not only from the pixels.

We also identify two policy downsides. First, by coloring the t-SNE based on time, we note that states with only a few bricks are visited for multiple times along the trajectory, indicating that the agent is stuck in a local optima (see Figure 1 in the appendix). We discuss the inability of the agent to perform well in the second level of the game in Section 5.5.

5.2. Seaquest

In Seaquest, the player's goal is to retrieve as many treasure-divers, while dodging and blasting enemy subs and killer sharks before the oxygen runs out. When the game begins, each enemy is worth 20 points and a rescued diver worth 50. Every time the agent surface with six divers, killing an enemy (rescuing a diver) is increased by 10 (50) points up to a maximum of 90 (1000). Moreover the agent is awarded with an extra bonus based on its remaining oxygen level and receives an extra life. However if it surface with less than six divers the oxygen fills up with no bonus, and if it surface with none it loses a life.

DQN's performance on Seaquest($\sim$5k) is inferior to human experts($\sim$100k). What makes Seaquest hard for DQN is that the shooting enemies is rewarded immediately, while rescuing divers is rewarded only once six divers are collected and rescued to sea level, therefore requires much longer planning.

We extract features for player position, direction of movement (ascent/descent), oxygen level, number of enemies, number of available divers to collect, number of available

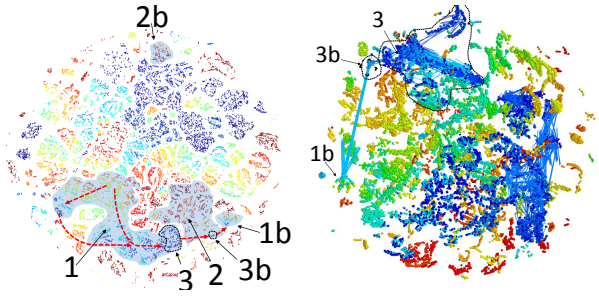

Figure 5. Seaquest refuel option. **Left:** Option path above the t-SNE map colored by the amount of remaining oxygen. Shaded blue mark the clusters with a collected diver, and red arrows mark the direction of progress. **Right:** 3d t-SNE with colored arrows representing transitions. All transitions from cluster 3 are leading to cluster 3b in order to reach the refuel cluster (1b), thus indicating a clear option termination structure.

divers to use, and number of lives.

Figure 4 shows the t-SNE map divided into different clusters. We notice two main partitions of the t-SNE clusters, between low (clusters 5-7) and high (cluster 1-3) oxygen levels, and by the amount of collected divers (clusters 2 and 11 both represent having a diver). We also identified other partitions between clusters such as refueling clusters (1: pre-episode refueling and 2: in-episode refueling), various termination clusters (8: agent appears in black and white, 9: agent's figure is replaced with drops, 10: agent's figure disappears) and low oxygen clusters characterized by flickering in the oxygen bar (4 and 7).

While the agent distinguishes states that are with a collected diver, Figure 6 implies that the agent did not understand the concept of collecting a diver and sometimes treats them as enemies. Moreover we see that the clusters share a similar value estimate that is highly correlated with the amount of remaining oxygen and the number of present enemies. However states with an available or collected diver do not raise the value estimates nor do states that are close to refueling. Moreover, the agent never explores the bottom of the screen, nor collects more than two divers. Thus it never experienced surfacing to sea level with 6 divers and never achieving the high reward bonus.

As a result, the agent's policy is to kill as many enemies as possible while avoiding being shot at. If it hasn't collected a diver, the agent follows a sub-optimal policy and ascends near to sea level as the oxygen decreases. There it continues to shoot at enemies (but not collecting divers as it should). However it also learned not to surface entirely without a diver.

If the agent collects a diver it moves to the blue shaded clusters (all clusters in this paragraph refer to Figure 5), where we identify a refuel option. We noticed that the option can

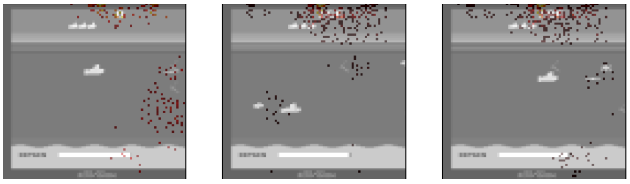

Figure 6. Salincy maps of states with available diver. **Left:** A diver is noticed in the saliency map but misunderstood as an enemy and being shot at. **Center:** Both diver and enemy are noticed by the network. **Right:** The diver is unnoticed by the network.

be initiated from two different zones based on the oxygen level but has a singular termination cluster (3b). If the diver is taken while having a high level of oxygen, then it enters the option at the northern (red) part of cluster 1. Otherwise it will enter on a later point along the direction of the option (red arrows). In cluster 1, the agent keeps following the normal shooting policy. Eventually the agent reaches a critical level of oxygen (cluster 3) and ascends to sea level. From there the agent jumps to the fueling cluster (area 1b). The fueling cluster is identified by its rainbow appearance because the level of oxygen is increasing rapidly. However, the refuel option was not learned perfectly. Area 2 is another refuel cluster, there, the agent does not exploit its oxygen before ascending to get air (area 2b).

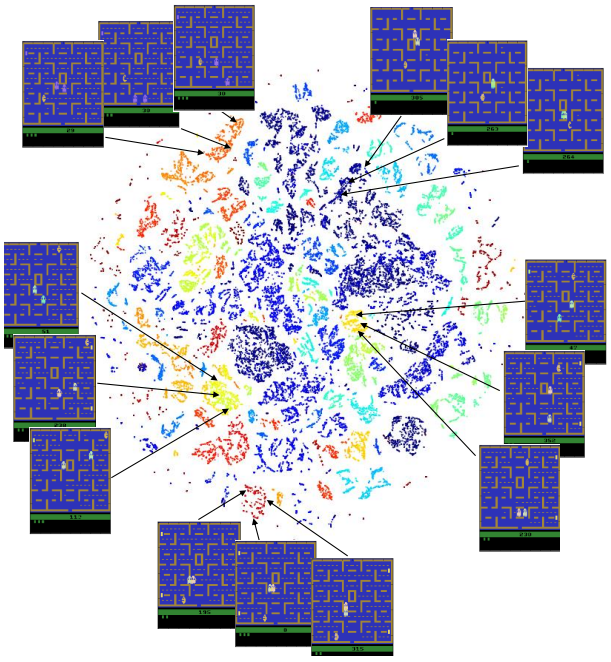

Figure 7. t-SNE for Pacman colored by the number of left bricks with state examples from each cluster.

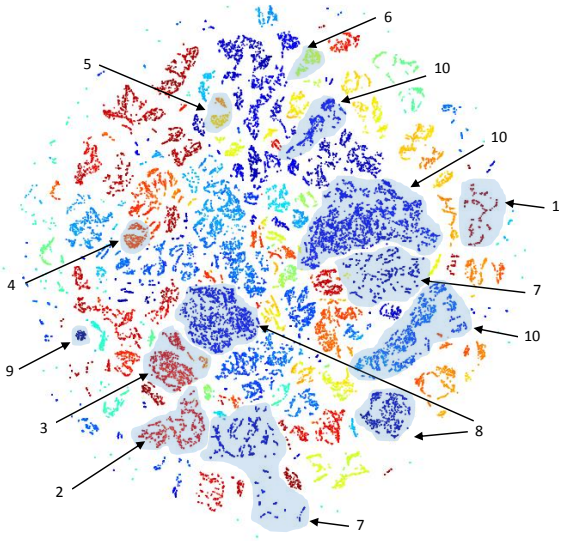

Figure 8. Pacman aggregated states on the t-SNE map colored by value function estimates.

5.3. Pacman

In Pacman, an agent navigates in a maze while being chased by two ghosts. The agent is positively rewarded (+1) for collecting bricks. An episode ends when a predator catches the agent, or when the agent collects all bricks. There are also 4 bonus bricks, one at each corner of the maze. The bonus bricks provide larger reward (+5), and more importantly, they make the ghosts vulnerable for a short period of time, during which they cannot kill the agent. Occasionally, a bonus box appears for a short time providing high reward (+100) once collected.

We extract features for player position, direction of movement, number of left bricks to eat, minimal distance (L1) between the player and the predators, number of lives, ghost mode that indicates that the predators are vulnerable, and bonus box feature that indicate when a highly valued box appears.

Figure 8 shows the t-SNE colored by value function, while Figure 7 shows the t-SNE colored by the number of left bricks with examples of states from each cluster. We can see that the clusters are well partitioned by the number of remaining bricks and value estimates. Moreover, examining the states in each cluster (Figure 7) we see that the clusters share specific bricks pattern and agent location. From Figure 9 we also learn that the agent collects the bonus bricks at very specific times. Thus, we understand that the agent has learned a location-based policy that is focused on collecting the bonus bricks, similar to maze problems, but one that is synchronous with avoiding the ghosts.

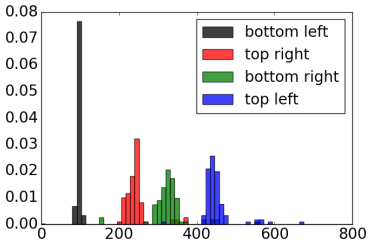

Figure 9. Histogram showing the time passed until each bonus brick is collected.

The agent starts at cluster 1, heading for the bottom-left bonus brick and collecting it in cluster 2. Once collected, it moves to cluster 3 where it is heading to the top-right bonus brick and collects it in cluster 4. The agent is then moving to clusters 5 and 6 where it is taking the bottom-right and top-left bonus bricks respectively. We also identified cluster 7 and 9 as termination clusters and cluster 8 as a hiding cluster in which the agent hides from the ghosts in the top-left corner since very few bricks remains.

The main downside of the learned policy is observed from looking at cluster 10. In this cluster, the bonus box is visible indicating that the agent learned to separate this situation from others, however we can see that the cluster has a low value estimate indicating that the agent did not learn the right value function yet.

5.4. Enviroment modeling

The DQN algorithm requires specific treatment for initial (padded with zero frames) and terminal (receive target value of zero) states, however it is not clear how to check if this treatment works well. Therefore we show t-SNE maps for different games with a focus on termination and initial states in Figure 10. We can see that all terminal states are mapped successfully into a singular zone, however, initial states are also mapped to singular zones and assigned with wrong value predictions. Following these observations we suggest to investigate different ways to model initial states, i.e., replicating a frame instead of feeding zeros and test it using our methodology. We also note that while terminal states seems to be modeled correctly, there seems to be a better way to model them, e.g., by setting the target to be zero if the next state is terminal.

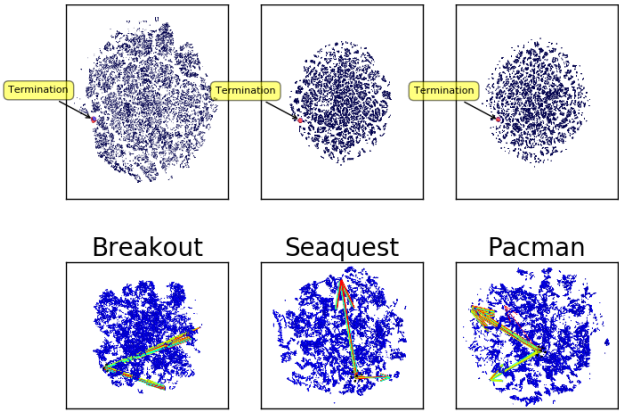

Figure 10. Terminal and initial states. **Top:** All terminal states are mapped into a singular zone(red). **Bottom:** Initial states are mapped into singular zones (pointed by colored arrows from the terminal zone) above the 3d t-SNE dynamics representation.

Figure 11. Saliency maps of score pixels. The input state is presented in gray scale while the value input gradients are displayed above it in red.

5.5. Score pixels

Some Atari2600 games include multiple repetitions of the same game. Once the agent finished the first screen it is presented with another one, distinguished only by the score pixels. Therefore, an agent might encounter problems with generalizing to the new screen if it over-fits the score pixels. In breakout, for example, the current state of the art architectures achieves a game score of around 450 points while the maximal available points are 896 suggesting that the agent is somehow not learning to generalize for the new level. We investigated the affect that score pixels has on the network predictions. Figure 11 shows the saliency maps of different games supporting our claim that DQN is basing its estimates using these pixels, and Figure 12 shows that the network maps states with different amount of score digits to different zones in the game of Seaquest. We suggest to further investigate these affects, for example we suggest to train an agent that does not receive those pixels as input.

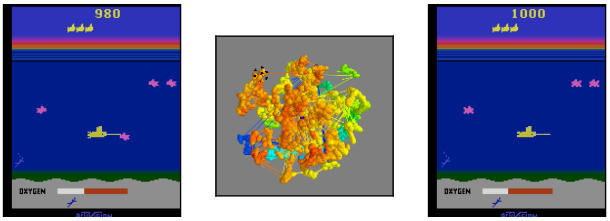

Figure 12. A transition from a state (left) with three score digits to its sequential (right) with four. The transition is displayed on the 3d t-SNE map (center) by a relative long arrow from the outlined area, thus indicate a transition between two different sub-manifolds.

6. Conclusions

Our experiments on ALE indicate that the features learned by DQN maps the state space to different sub-manifolds, in each, different features are present. By analyzing the dynamics in these clusters and between them we were able to identify hierarchical structures. In particular we are able to identify options with defined initial and termination rules. The states aggregation gives the agent the ability to learn specific policies for the different regions, thus giving an explanation to the success of DRL agents.

Similar to Neuro-Science, where reverse engineering methods like fMRI reveal structure in brain activity, we demonstrated how to describe the agent's policy with simple logic rules by processing the network's neural activity. This is important since often humans can understand the optimal policy and therefore understand what are the agent's weaknesses. The ability to understand the hierarchical structure of the policy can help in distilling it into a simpler architec-

ture (Rusu et al., 2015; Parisotto et al., 2015). Moreover, we can direct learning resources to clusters with inferior performance by prioritized sampling (Schaul et al., 2015).

Debugging the network using our methodology discovered that the network is over-fitting specific pixels, therfore, we suggest to crop the redundant pixels from the input frames. We also noticed problems in modeling initial and terminal states. We suggest to model initial states by replicating the state instead of padding it with zeroes and to model the terminal states in the network itself and not the target function.

The state aggregation learned by the agent may be used to design better algorithms. One possibility is to learn a classifier from a states to clusters based on the t-SNE map and then learn a different control rule at each cluster. Another option is to use human understanding of the learned clusters and to motivate the network to join redundant clusters and create new clusters. For example, in Seaquest, clusters that are separated by the amount of score digits may be joined to encourage better generalization, and a cluster that distinguishes states with an available diver should be created.

Acknowledgement

This research was supported in part by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

References

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.

Bellemare, Marc G, Ostrovski, Georg, Guez, Arthur, Thomas, Philip S, and Munos, Rémi. Increasing the action gap: New operators for reinforcement learning. *arXiv preprint arXiv:1512.04860*, 2015.

Dayan, Peter and Hinton, Geoffrey E. Feudal reinforcement learning. pp. 271–271. Morgan Kaufmann Publishers, 1993.

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

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

Engel, Yaakov and Mannor, Shie. Learning embedded maps of markov processes. In *in Proceedings of ICML 2001*. Citeseer, 2001.

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.

Gordon, Geoffrey J. Stable function approximation in dynamic programming. 1995.

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.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

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.

Mann, Timothy A, Mannor, Shie, and Precup, Doina. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53(1): 375–438, 2015.

Mannor, Shie, Menache, Ishai, Hoze, Amit, and Klein, Uri. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 71. ACM, 2004.

Menache, Ishai, Mannor, Shie, and Shimkin, Nahum. Q-cutdynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002*, pp. 295–306. Springer, 2002.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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):529–533, 2015.

Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, De Maria, Alessandro, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Nguyen, Anh, Yosinski, Jason, and Clune, Jeff. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

Parisotto, Emilio, Ba, Jimmy Lei, and Salakhutdinov, Ruslan. Actor-mimic: Deep multitask and transfer reinforcement learning, 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.

Quiroga, R Quian, Reddy, Leila, Kreiman, Gabriel, Koch, Christof, and Fried, Itzhak. Invariant visual representation by single neurons in the human brain. *Nature*, 435 (7045):1102–1107, 2005.

Ramachandran, P. and Varoquaux, G. Mayavi: 3D Visualization of Scientific Data. *Computing in Science & Engineering*, 13(2):40–51, 2011. ISSN 1521-9615.

Riedmiller, Martin. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pp. 317–328. Springer, 2005.

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

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Şimşek, Özgür, Wolfe, Alicia P, and Barto, Andrew G. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pp. 816–823. ACM, 2005.

Singh, Satinder P, Jaakkola, Tommi, and Jordan, Michael I. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, pp. 361–368, 1995.

Stolle, Martin. *Automated discovery of options in reinforcement learning*. PhD thesis, McGill University, 2004.

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):181–211, 1999.

Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian, and Fergus, Rob. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Tenenbaum, Joshua B, De Silva, Vin, and Langford, John C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Tesauro, Gerald. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Thrun, Sebastian. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

Tsitsiklis, John N and Van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5): 674–690, 1997.

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.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

Wang, Ziyu, de Freitas, Nando, and Lanctot, Marc. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

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.

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

7. Suplemantary figures

7.1. Breakout

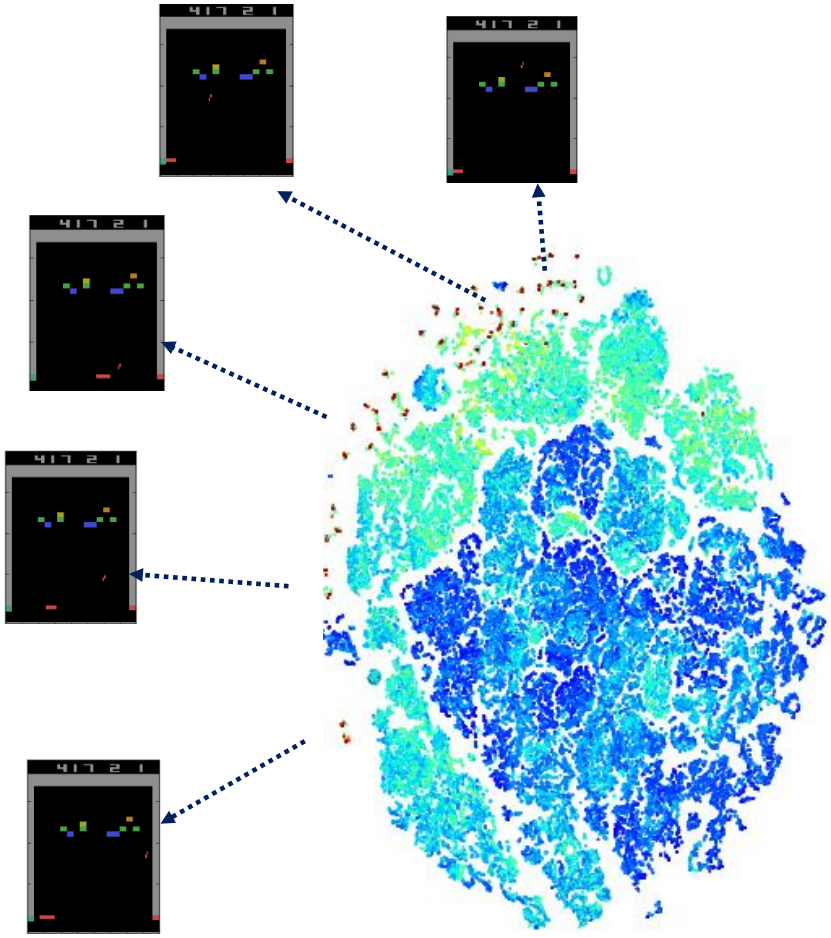

Figure 13. t-SNE for Breakout colored by time. Examples of repeated states by the agent are presented.

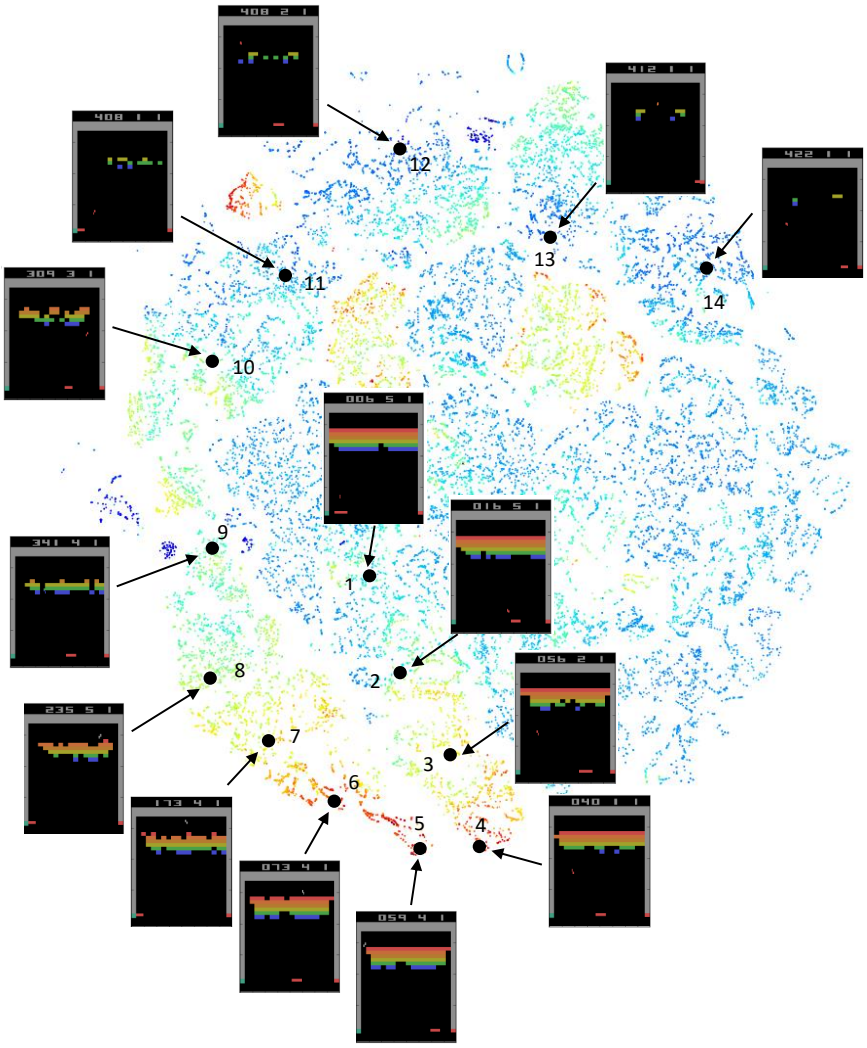

Figure 14. t-SNE for Breakout colored by time. Examples of states alon the option path are presented.

7.2. Seaquest

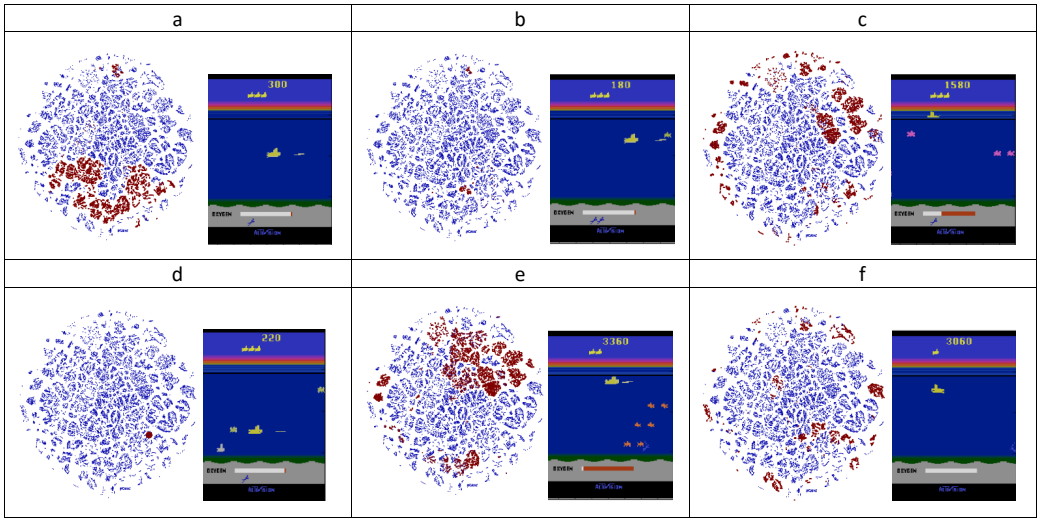

Figure 15. Seaquest states partitioned on the t-SNE map by different measure. a: Number of taken divers = 1. b: Number of taken divers = 2. c: Agent is next to sea level (filter by vertical position ¡ 60). d: Agent is never visiting the lower 1/3 part of the frame. We mark the state of maximal diving depth (filter by vertical position ¿ 128). e: Oxygen level ¡ 0.05. f: Oxygen level ¿ 0.98.

7.3. Pacman

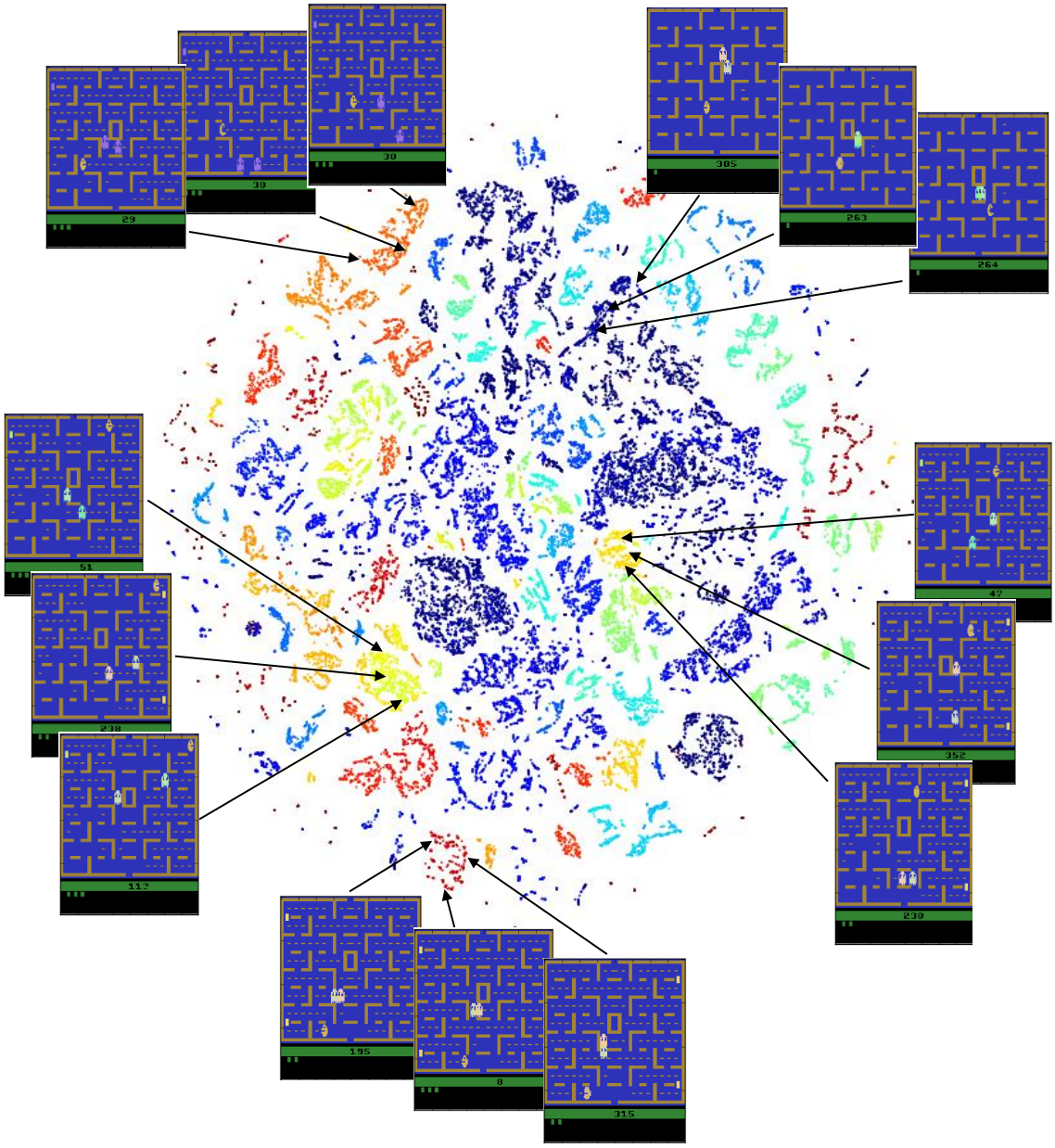

Figure 16. Pacman t-SNE with example states.