Graying the black box: Understanding DQNs

Tom Zahavy* TOMZAHAVY@CAMPUS.TECHNION.AC.IL
Nir Ben Zrihem* NIRB@TX.TECHNION.AC.IL
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 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 long standing 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, hence, reducing the agent's flexibility and making the learning more tedious. Therefore, there is a growing interest in using nonlinear 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).

*These authors have contributed equally.

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

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 success 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. DQN achieved dramatically better results than earlier approaches, showing a robust ability to learn representations.

While using Deep Learning (DL) in RL seems promising, there is no free lunch. Training a Deep Neural Network (DNNs) is a complex two-levels optimization process. At the inner level we fix a hypothesis class and learn it using some gradient descent method. The optimization of the outer level addresses the choice of network architecture, setting hyper-parameters, and even the choice of optimization algorithm. 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 by either a trial and error methodology, or by exhaustively searching over the space of possible configurations. Moreover, 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).

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 ("labels") in contrast to other machine learning problems that enjoy local generalization. In fact, in many cases the optimal policy is not a smooth function of the state, thus using a linear approximation is far from being optimal. 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 (with: fill air, without: loose a life) and therefore the

optimal policy. Another cause for control discontinuities is that for a given problem, two states with 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)).

Differnt methods that focus on the temporal structure of the policy, has also 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 temporal representation of the states, varying levels of convergence guarantees can be promised (Dean & Lin, 1995; Parr, 1998; Hauskrecht et al., 1998; Dietterich, 2000).

In this work we analyze the state representation learned by DRL agents. We claim that the success of DQN should be attributed to its ability to learn spatio-temporal hierarchies using different sub manifolds. We show that by doing so we can offer an interpertation of learned policies that may help formalizing the outer optimization step. Our methodology is to record the neural activations of the last DQN hidden layer, 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 features of the policy and other 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:

Understanding: We show that DQNs are learning temporal abstractions of the state space such as hierarchical state aggregation and options. Temporal abstractions were known to the RL community before as mostly manual tools to tackle the curse of dimensionality; however, we observe that a DQN is finding abstractions automatically. Thus, we believe that our analysis explains the success of DRL from a reinforcement learning research perspective.
Interpretability: We give an interpretation for the agents policy in a clear way, thus allowing to understand what are its weaknesses and strengths.
Debugging: We propose debugging methods that help to reduce the hyper parameters grid search of DRL. Examples are given on game modelling, termination and initial states and score over-fitting.

2. Related work

The goal behind visualization of DNNs 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 data set that cause high activations of individual units. For example, the deconvolution method (Zeiler & Fergus, 2014) highlights the areas 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 had been 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. (2015), suggested to use saliency maps and analyzed which pixels affect network predications the most. Using this method they compared between the standard DQN and their duelling network architecture. Engel & Mannor (2001) learned an embedded map of Markov processes and visualized it on two dimensions. Their analysis is based on the state transition distribution while we will focus on distances between the features learned by DQN.

3. Background

The goal of **RL** agents is to maximize its expected total reward by learning an optimal policy (mapping 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 when choosing action a_t at state s_t, afterwards following 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,

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

Deep Q Networks (Mnih et al., 2015; 2013) approximate the optimal Q function using a Convolutional Neural Network (CNN). The training objective is to minimize the expected TD error of the optimal Bellman equation:

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

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

Notice that this is an 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 reward r_t is clipped to the range of $[-1, 1]$ to guarantee stability when training DQNs over multiple domains with different reward scales. The DQN algorithm maintains two separate Q-networks: one with parameters θ, and a second with parameters θ_{target} that are updated from θ every fixed number of iterations. In order to capture the game dynamics, the DQN algorithm represents a state by a sequence of history frames and pads initial states with zero frames.

t-SNE is a non-linear dimensionality reduction method used mostly for visualizing high dimensional data. The technique is easy to optimize, and 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 benchmarks and hyper-spectral remote sensing data (Lunga et al., 2014). t-SNE 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 lies on several low-dimensional manifolds. This enables us to visualize the different sub-manifolds learned by the network and interpret their meaning.

4. Methods

Our methodology comprises the following steps:

1. Train a DQN.
2. Evaluate the DQN. For each visited state record: (a) activations of the last hidden layer, (b) the gradient information and (c) the game state (raw pixel frame).
3. Apply t-SNE on the activations data.
4. Visualize the data.
5. Analyze manually.

Figure 1 presents our Visualization tool. Each state is represented as a point in the t-SNE map (Left). The color of the points is set manually using global features (Top right) or game specific hand crafted features (Middle right). Clicking on each data point displays the corresponding game image and saliency map (Bottom right). It is possible to move between states along the trajectory using the F/W and B/W buttons.

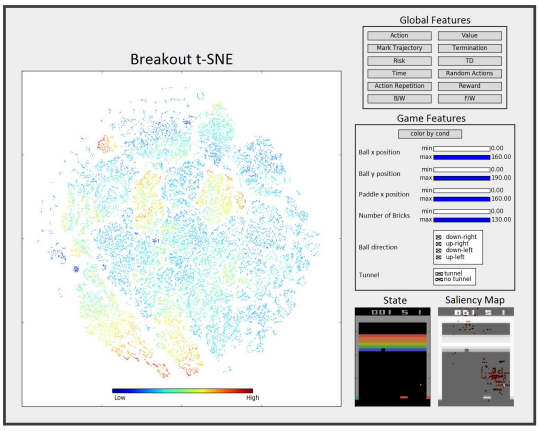

Figure 1. Graphical user interface for our methodology.

Feature extraction: During the DQN run, we collect statistics for each state such as Q values estimates, generation time, termination signal, reward and played action. 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 color and filter points on the t-SNE map. 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. In all the figures below we use a heat color map (red corresponds to high values and blue to low ones). Similar to Engel & Mannor (2001) we visualize the dynamics (state transitions) of the learned policy. To do so we use a 3D t-SNE state representation wich we found insightfull. The transistions are displayed with arrows using Mayavi (Ramachandran & Varoquaux, 2011).

t-SNE: We apply the t-SNE algorithm directly on the collected neural activations, similar to Mnih et al. (2015). The input $X \in \mathbf{R}^{120k \times 512}$ consists of $120k$ game states with 512 features each (size of the last layer). Since these data are 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).

Saliency maps: We generate Saliency maps (similar to Simonyan & Zisserman (2014)) by computing the Jacobian of the network with respect to the input, and presenting it above the input image itself (Figure 1, bottom right). These maps helps to understand which pixels in the image affect the value prediction of the network the most.

Analysis: Using these features we are able to understand the common attributes of a given cluster (e.g, Figure 3 in the appendix). Moreover, by analyzing the dynamics between clusters we can identify a hierarchical aggregation of the state space. We define clusters with a clear entrance and termination areas as options, and interpretat the agent policy there. For some options we are able to derive rules

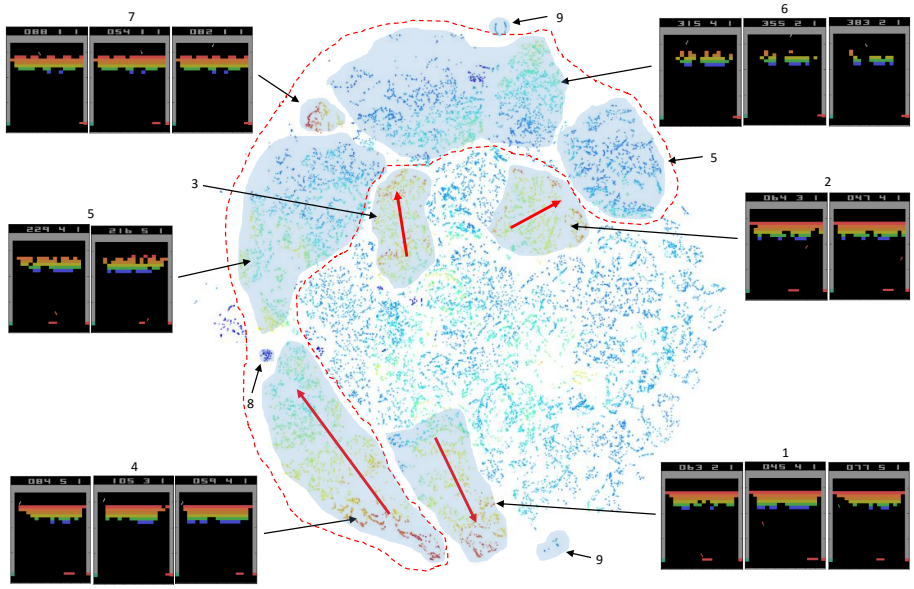

Figure 2. Breakout aggregated states on the t-SNE map.

for initiation and termination (i.e., landmark options, Sutton et al. (1999); Mann et al. (2015)).

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 finally analyze 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. When a brick is hit, the ball bounces away, the brick is destroyed and the player receives reward. 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 (a tunnel exists when there is a clear path between the area below the bricks and the area above it, and 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 agent to achieve high reward while being safe from losing the ball. By introducing a discount factor to Breakout's MDP, this strategy becomes even more favourable since it achieves high immediate reward.

Figure 2 presents the t-SNE of Breakout. The agent learns a hierarchical policy: (a) carve a tunnel in the left side of the screen and (b) 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 exit until the tunnel is carved (see Figure 4). We identify these clusters as a landmark option. 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 where 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), differentiated by the ball position with regard to the bricks (see Figure 2). 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).

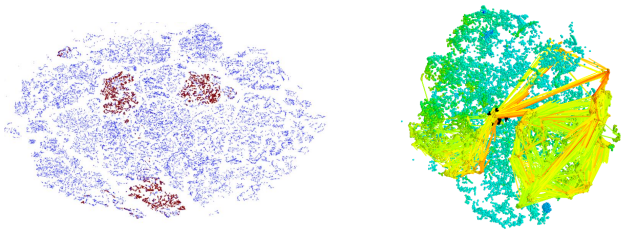

Figure 4. Breakout tunnel digging option. **Left:** states that the agent visits once it entered the option clusters (1-3 in Figure 2) until it finishes to carve the left tunnel are marked in red. **Right:** Dynamics is displayed by arrows above a 3d t-SNE map. The option termination zone is marked by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

Cluster 7 is created due to a bug in the emulator that allows the ball to pass the tunnel without completely carving it. The agent learned to represent this incident 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 on the pixels.

By coloring the t-SNE map by time, we can identify some policy downsides. States with only a few remaining bricks are visited for multiple times along the trajectory (see Figure 1 in the appendix). In these states, the ball bounces without hitting any bricks which causes a fixed reflection pattern, indicating that the agent is stuck in a local optima. We discuss the inability of the agent to perform well in the second screen 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 an extra bonus based on its remaining oxygen level. However, if it surfaces with less than six divers the oxygen fills up with no bonus, and if it surfaces with none it loses a life.

DQN's performance on Seaquest (∼5k) is inferior to human experts (∼100k). What makes Seaquest hard for DQN is that shooting enemies is rewarded immediately, while rescuing divers is rewarded only once six divers are collected and rescued to sea level. Moreover, the bonus points for collecting 6 divers is diminished by reward clipping. This sparse and delayed reward signal requires much longer planning that is harder to learn.

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 divers to use, and number of lives.

Figure 3 shows the t-SNE map divided into different clusters. We notice two main partitions of the t-SNE clusters: (a) by oxygen level (low: clusters 4-10, high: cluster 3 and other unmarked areas), and (b) by the amount of collected divers (clusters 2 and 11 represent having a diver). We also identified other partitions between clusters such as refuelling clusters (1: pre-episode and 2: in-episode), various termination clusters (8: agent appears in black and white,

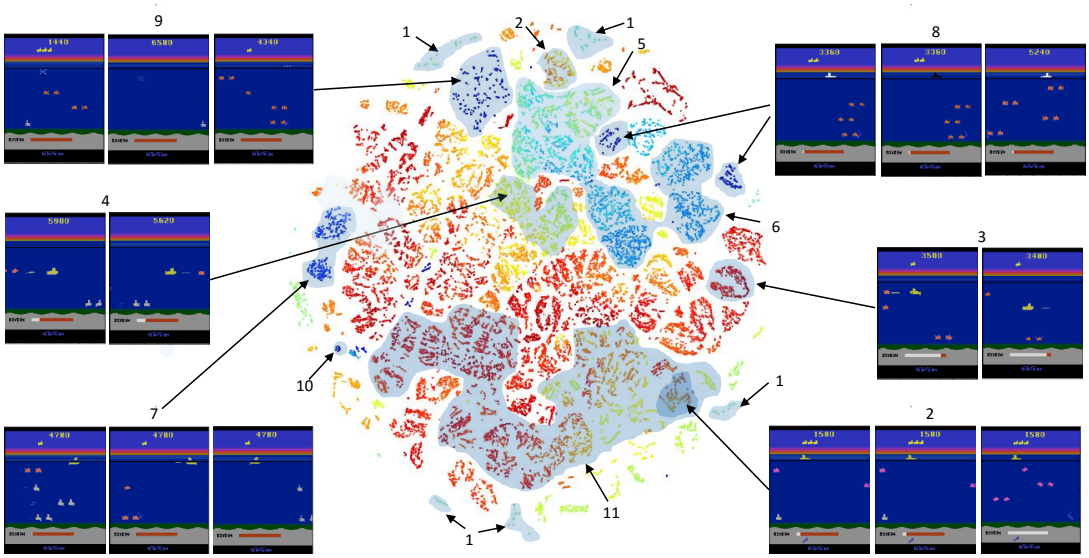

Figure 3. Seaquest aggregated states on the t-SNE map, colored by value function estimate.

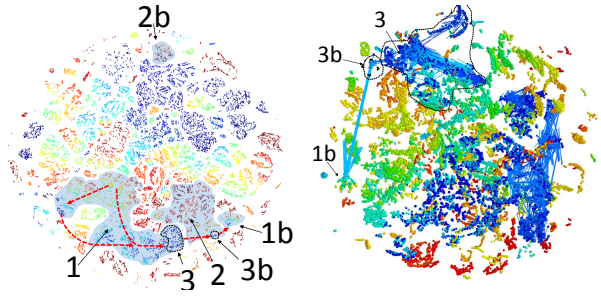

Figure 5. Seaquest refuel option. **Left:** Option path above the t-SNE map colored by the amount of remaining oxygen. Shaded blue mark the clusters with a collected diver, and red arrows mark the direction of progress. **Right:** 3d t-SNE with colored arrows representing transitions. All transitions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

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 with a collected diver, Figure 6 implies that the agent did not understand the concept of collecting a diver and sometimes treats it as an enemy. 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.
As a result, the agent's policy is to kill as many enemies as possible while avoiding being shot. 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. 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

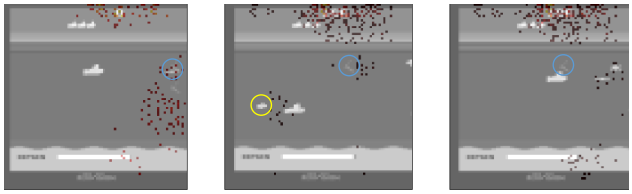

Figure 6. Salincy maps of states with available diver. **Left:** A diver is noticed in the saliency map but misunderstood as an enemy and being shot at. **Center:** Both diver and enemy are noticed by the network. **Right:** The diver is unnoticed by the network.

we identify a refuel option. We noticed that the option can 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 point further 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).

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) if collected.

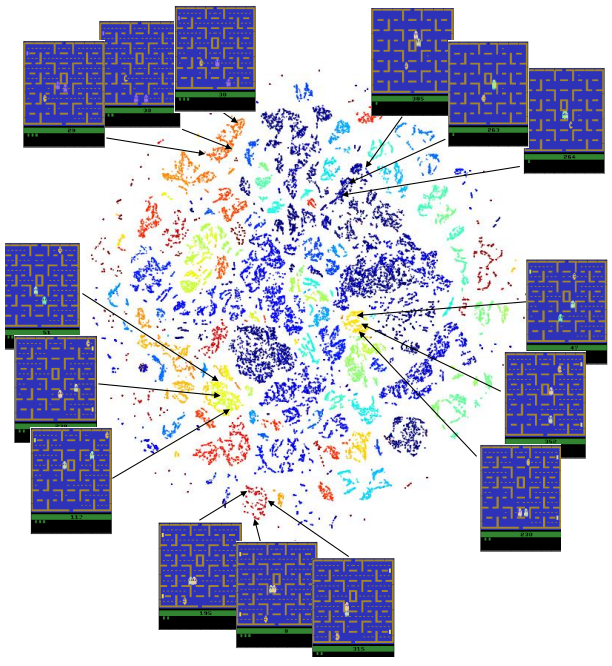

Figure 8. t-SNE for Pacman colored by the number of left bricks with state examples from each cluster.

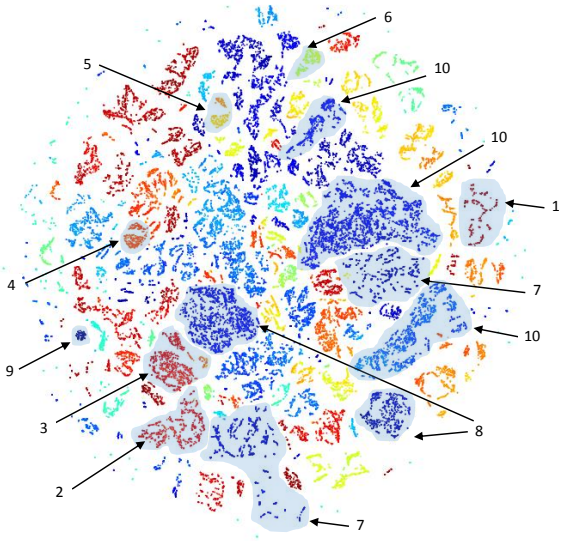

Figure 7. Pacman aggregated states on the t-SNE map colored by value function estimates.

We extract features for the 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 the highly valued box appears.

Figure 7 shows the t-SNE colored by value function, while Figure 8 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 8) we see that the clusters share specific bricks pattern and agent location.

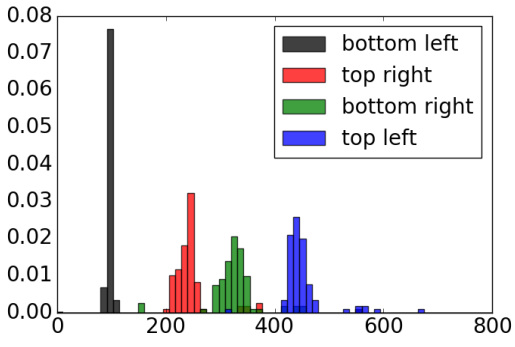

Figure 9. Histogram showing the time passed until each bonus brick is collected.

From Figure 9 we also learn that the agent collects the bonus bricks at very specific times and order. Thus, we conclude that the agent has learned a location-based policy that is focused on collecting the bonus bricks (similar to maze problems) while avoiding the ghosts.

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. The effects of reward-clipping can be clearly seen in the case of the bonus box. Cluster 10 comprises of states with a visible bonus box. However these states are assigned with a low value.

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.

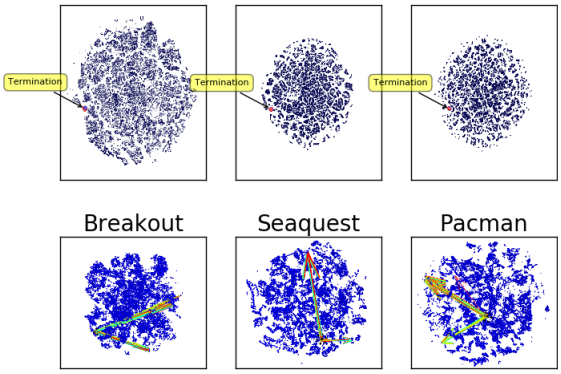

Figure 10. Terminal and initial states. **Top:** All terminal states are mapped into a singular zone(red). **Bottom:** Initial states are mapped into singular zones (pointed by colored arrows from the terminal zone) above the 3d t-SNE dynamics representation.

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 that was accumulated in the first screen. 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 effect 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. We suggest to further investigate this, for example we suggest to train an agent that does not receive those pixels as input.

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.

6. Conclusions

In this work we showed that the features learned by DQN map 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 were able to identify options with defined initial and termination rules. State aggregation gives the agent the ability to learn specific policies for the different regions, thus giving an explanation to the success of DRL agents. The ability to understand the hierarchical structure of the policy can help in distilling it into a simpler architecture (Rusu et al., 2015; Parisotto et al., 2015) and may help to design better algorithms. One possibility is to learn a classifier from states to clusters based on the t-SNE map, and then learn a different control rule at each. Another option is to allocate learning resources to clusters with inferior performance, for example by prioritized sampling (Schaul et al., 2015).

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. We believe that interpretation of policies learned by DRL agents is of particular importance by itself. First, it can help in the debugging process by giving the designer qualitative understanding of the learned policies. Second, there is a growing interest in applying DRL solutions to real-world problem such as autonomous driving and medicine. We believe that before we can reach that goal we will have to gain greater confidence on what the agent is learning. Lastly, understanding what is learned by DRL agents, can help designers develop better algorithms by suggesting solutions that address policy downsides.

To conclude, when a deep network is not performing well it is hard to understand the cause and even harder to find ways to improve it. Particularly in DRL, we lack the tools needed to analyze what an agent has learned and therefore left with black box testing. In this paper we showed how to gray the black box: understand better why DQNs work well in practice, and suggested a methodology to interpret the learned policies.

Acknowledgement

This research was supported in part by the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement 306638 (SUPREL) and 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.