Multi level DQN

Itay Golan, Saar Barkai
Supervisors: Tom Zahavy and Daniel Mankowitz
CRML lab, Electrical Engineering Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

In this project we research DQN performance in multi-level games with Qbert as a test case. We show that Qbert is a high risk game and the exploration step results in a glass ceiling. Then, we show a specific solution for Qbert and purpose a potential solution for the general problem using multiple agents with automatic clustering.

Introduction

DQN is a reinforcement learning algorithm which is known to achieve state-of-the-art results on different Atari games as published at Mnih et al. (2015). Although the results often surpasses human experts, it was observed that games which contained different levels, like Qbert, in which most of the change from previous or future levels was in the way the screen was displayed (e.g. color), were learning rather fast at first but after a few levels they reached saturation. That happened even though the agent succeed learning to play the first few levels and thus virtually ”solved” the game and only needs to transfer the knowledge to the higher levels. The challenge of learning new tasks while retaining the knowledge about previous one was addressed in Kirkpatrick et al. (2017), which tried to avoid this catastrophic forgetting by manipulating each weight update according to is relevance for the task.

Solving this problem can potentially produce an algorithm which learns these optimal actions and thus reaches near-optimal results in most Reinforcement Learning problems and specifically in life-long learning. Once solved, the solution is almost guaranteed to be implemented in the ”general-optimal” learning algorithm.

In this project, the Atari game Qbert was used as a test case for multi-level games, and most of this work was done on Qbert.

It is unclear why the current DQN is unable to overcome this obstacle. The most probable reason is due to the fact that the state of the agent in the DQN algorithm is determined primarily by the screen it is given, and since the screen is affected mostly by its RGB colors, changing the colors greatly influences the state the agent is at and also the decision making, accordingly. Since the DQN algorithm is rather new, not all its aspects have been examined. This aspect is one of them.

Background

Reinforcement learning is a method where an agent interact with an environment by taking actions that change the environment state and rewarding the agent with reward. On time t the agent observes state s_t, according to the agent's policy π, it reacts with an action $a_t = \pi_t(s_t)$ and receiving reward a_t. The discounted cumulative reward is defined by $R_t = \sum_{t'=t}^{t'=\infty} \gamma^{t'} r_{t'}$, which called the infinite horizon problem with discount factor $\gamma \in (0, 1)$. The action-value function $Q^\pi(s,a) = \mathbf{E}[R_t | s_t = s, a_t = a, \pi]$ represents the expected reward after observing state s and taking an action a according to the policy π. The optimal action-value function obeys a fundamental recursion known as the Bellman equation:

$$Q^*(s,a) = \mathbf{E}[r_t + \gamma max_{a'} Q^*(s_{t+1}, a')]$$

Deep Q Network The DQN algorithm approximates the optimal Q function with a Convolutional Neural Network (CNN) Krizhevsky, Sutskever, and Hinton (2012), by minimizing the expected Temporal Difference (TD) error of the optimal bellman equation:

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

where

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

The tuples s_t, a_t, r_t, s_{t+1} are collected from the agents experience and are stored in the Experience Replay (ER) Lin (1993), making it an offline learning algorithm. DQNs dealing with the dilemma of exploring more unknown states versus exploiting the current knowledge with $\epsilon - greedy$ policy where the agent takes a random action with probability ϵ to explore new unvisited states which might improve its' policy.

Qbert

Qbert: is an Atari game of a player that needs to ”touch” all 21 bricks of a pyramid. Along the way it might lose lives by encountering a monster or falling of the edge of the pyramid. The player has 4 lives and when it finished to "touch" all the bricks it moves onto the next level with more complicated monsters. *Risk in Qbert:* We show that the $\epsilon - greedy$ policy in Qbert leads to a score limit both by probability analysis and simulations. Since in this game there are many risky states that a single action might cause to lose life, the $\epsilon - greedy$ policy will limit also an agent with optimal policy. *Probability analysis:* The analysis assumes uniform distribution on the player's position and no monsters - which is makes the analysis stricter. The player has 6 possible actions to take, the pyramid has 8/21 bricks with no fatal action, 6/21 bricks with 1 fatal action called, 5/21 with 2 fatal actions and 2/21 with 3 fatal actions, those actions will be called fatal0, fatal1, fatal2 and fatal3 with edge0, edge1, edge2 and edge 3 as the corresponding bricks. Taking the parameters from DQN paper, the values of ϵ are 1 to 0.1 on the learning phase, and 0.05 on evaluation. With $\epsilon = 0.05$, we have the following probability for death due to random action:

$$P((death to random action)) =$$

$$(P(fatal0|edge0)*P(edge0)+P(fatal1|edge1)*P(edge1)+$$

$$P(fatal2|edge2)*P(edge2)+P(fatal3|edge3)*P(edge3))*\epsilon$$

$$= (\frac{0}{6}*\frac{8}{21}+\frac{1}{6}*\frac{6}{21}+\frac{2}{6}*\frac{5}{21}+\frac{3}{6}*\frac{2}{21})*\epsilon$$

Practically, this implies death due to random action every 114 steps when playing with epsilon 0.05. This project is not the first to use low value of epsilon, the known Double DQN, presented in Van Hasselt, Guez, and Silver (2016) and achieved high results, used epsilon that goes down to 0.01 during learning and 0.001 on evaluations. To show the effect of the epsilon-greedy policy, we trained multiple agents, each agent learns and play on a single level of Qbert and stop learning once it pass the level without losing lives. The evaluations were taken both with epsilon 0 and epsilon 0.05. Between those 2 epsilon values the score gap on evaluations reaches as high as 160% in favour to epsilon 0.

Method

Our solution suggest to use multiple agents and KMeans algorithm enhanced by spatio-tempral clustering algorithms described in Zrihem, Zahavy, and Mannor (2016). The usage of spatio-temporal clustering increasing the accuracy as Zahavy, Ben-Zrihem, and Mannor (2016) revealed that DQNs are automatically learning temporal representations such as hierarchical state aggregation and temporal abstractions. Each cluster will be learned and played by a different agent. This way, each cluster/level is learned individually and also the exploration can be manipulated individually, according to the agent's performance rmence. Thus, the exploration-explotation dilemma is solved by exploiting more when we are more confident that we achieve good results and also exploring more when we are unsure of the results. The algorithm, which is a modification of the original DQN algorithm, is described in Algorithm 1. It is suppose to take the concept of experiment 3, and generalize. Figure 2 shows a block diagram of the method.

Algorithm 1 Multiple agents with auto clustering

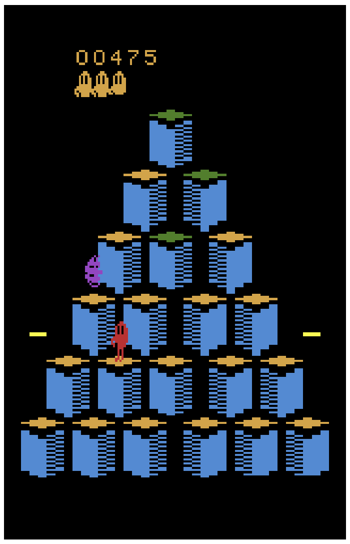
```
1. Run regular DQN with a single agent for 50 evaluations.
2. Clustering
   (a) Evaluate and save "future clustering network" activations
   (b) Cluster the saved activations using spatio-temporal KMeans
   (c) Save "future clustering network" as "current clustering network"
3. Continue learning and for every state and reward do:
   (a) Forward state through "current clustering network" and get activations
   (b) Assign cluster to the activations according to the closest KMeans centroid
   (c) Send state and reward to the cluster's agent
   (d) Send reward and state to "future clustering network"
4. On every evaluation:
   (a) Turn off exploration and learning for agents with performance which exceeds a certain threshold
   (b) Turn on exploration and learning for agents with performance which fell below the threshold
5. Repeat Clustering stage every 30 evaluations
```

The previous algorithm (Vanila DQN) only had a single agent and therefore the exploration rate (ϵ) was global and shared between every possible state at a given time (step). The addition of multiple agents allows us to control our exploration-explotation ratio in a manner that differs from different clusters of states to others. For example, if we played and learned a certain state many times, because of the DQN convergence, the possibility that we are choosing the optimal action is high and therefore we would like to exploit rather than explore (lower ϵ) on that specific state. Assuming that similar states are also close in time and similarly visited, by clustering the state-space to clusters of similar states, we can control the exploration-exploitation ratio per cluster. Also, dividing the state-space into several agents, helps to prevent deterioration of early states since it negates dependancy between state which are similar in representation but they are actually far apart in time, thus allowing to better differ between the states. The idea of freezing networks of solved tasks is not new, Rusu et al. (2016) also froze part of the network, and outperformed common algorithms.

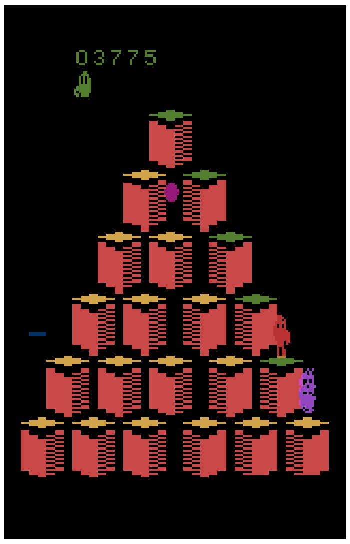
Experiments

Experiment 1 - same colors

Our first experiment was to see whether displaying different levels in the same color as the first level allows the agent to solve new levels using the knowledge of the levels it had already learned. We implemented the experiment by first identifying which level we are in using the score and the colors of the bricks. We then manipulated the brick colors to match



Level 1



Level 2

Figure 1: Qbert level 1 and 2

the first level's colors and sent the new colors to the agent to make him think the colors had never changed. The results, as you can see at Figure 3, showed that the agent can project the knowledge it learned from previous levels onto the next levels, thus providing a proof of concept. However, The overall results still reached a rather low total score. Furthermore, the results had a very high variance with respect to the vanilla DQN variance.

We concluded that the reason for the saturation was due to 2 major possibilities:

- When states which appear the same (screen-wise) are actually different (come from different levels) it confuses and deters the learning. This happens because the expected reward can be calculated according to state 1 while in fact we are in state 2. For example, when the pawn is at the starting point of two different levels.
- The agent reaches *terminal-state* at random with a rather high probability, regardless of learning.

Experiment 2 - multiple agents

In another experiment we created several agents (7 to be exact) and made sure that every agent will learn and play a different level. The objective of this simulation was to eliminate the dependancy between states so that there will be no *confusion* between levels while learning. We set the experiment by generating 7 agents. Each new state we got, we calculated the level, the same way we did in experiment 1, only this time, the colors remained unchanged and the new state was only sent to its corresponding agent. The evaluations were done with both epsilon 0 and epsilon 0.05 to allow the comparison.

As shown in Figure 4, the results were still poor so we concluded that the main problem was reaching *terminal-state* at random. We did see a significant improvement in the variance so we deduced that having multiple agent really does help learning. We assessed the relevant probabilities and concluded that while $\epsilon = 0.05$, the probability of reaching *terminal-state* from a random move in **Qbert** is 0.174. Meaning that on average, solely due to ϵ, the agent makes a *fatal* action every 115 moves.

Experiment 3 - multiple agents, changing ϵ

Similar to the last experiment, in this experiment we also used a different agent for every level. The difference was that once an agent passes the level without reaching terminal, its network is immediately frozen and its ϵ is lowered to zero, instead of the usual 0.05 in evaluation mode. We also split the agent each time we reached a new level. Meaning that each new agent was initialized with the previous agent's weights. We compared the results with experiment 2 results in order to get a reliable reference.

As shown in Figure 5, The results of the *ϵ-manipulated* simulation is greatly higher than the other ones which points to the fact that in certain scenarios (Qbert among them), it is a lot better to continue to minimize the exploration but it must be done with respect to the "experience" already gained.

Experiment 4 - Auto clustering

On this experiment, we implemented the method described in Algorithm 1, which includes the spatio-temporal KMeans clustering and epsilon manipulation. The preliminary results are poor, but it is possible in future research, with parameter-tuning and slight changes in the algorithm, this method will score better than Experiment 3. Part of the proof of concept of this experiment was to cluster activations from Vanilla DQN simulation. The results of this clustering is shown in Figure 6 on a TSNE map, and it visible that the accuracy is high. We noticed this high accuracy, around 90% also in this auto clustering experiment.

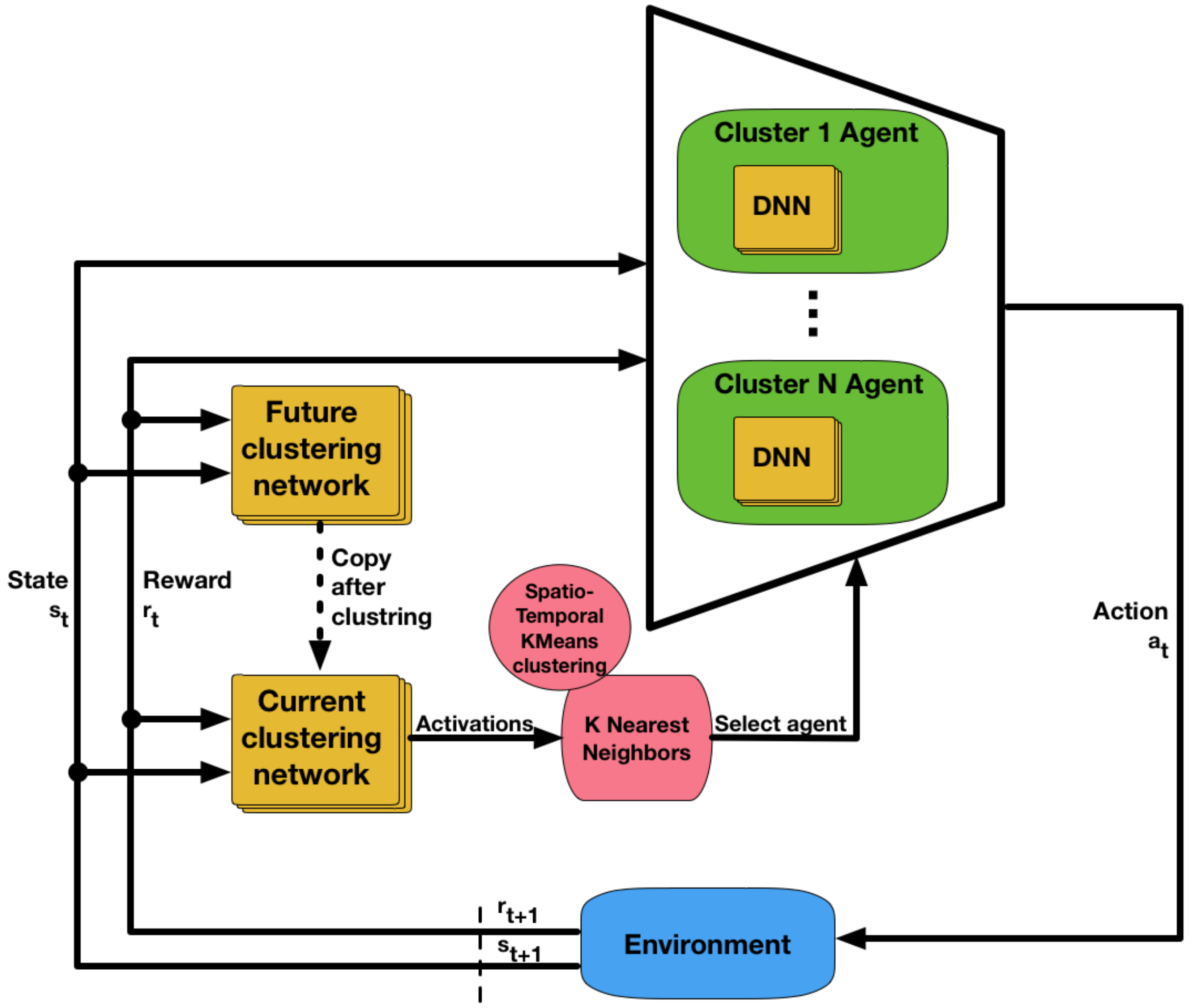


Figure 2: Block diagram of the auto clustering method

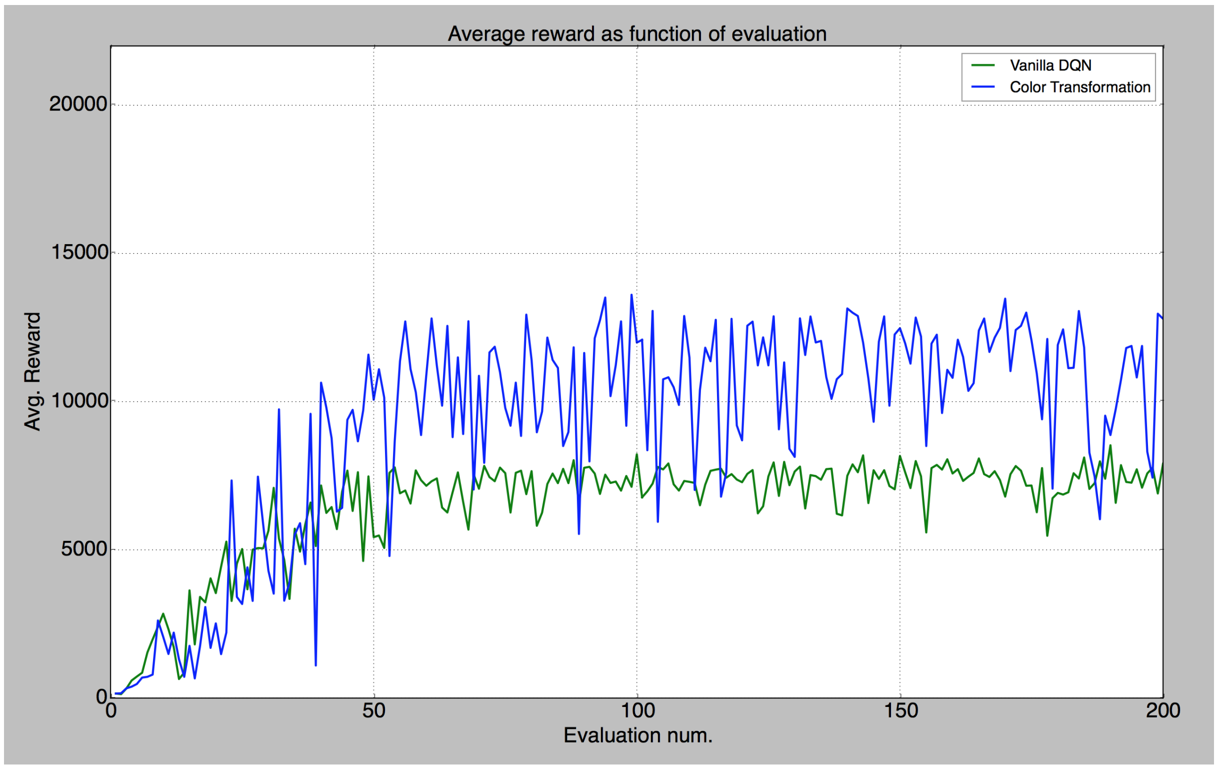


Figure 3: Experiment 1 - Color transformation

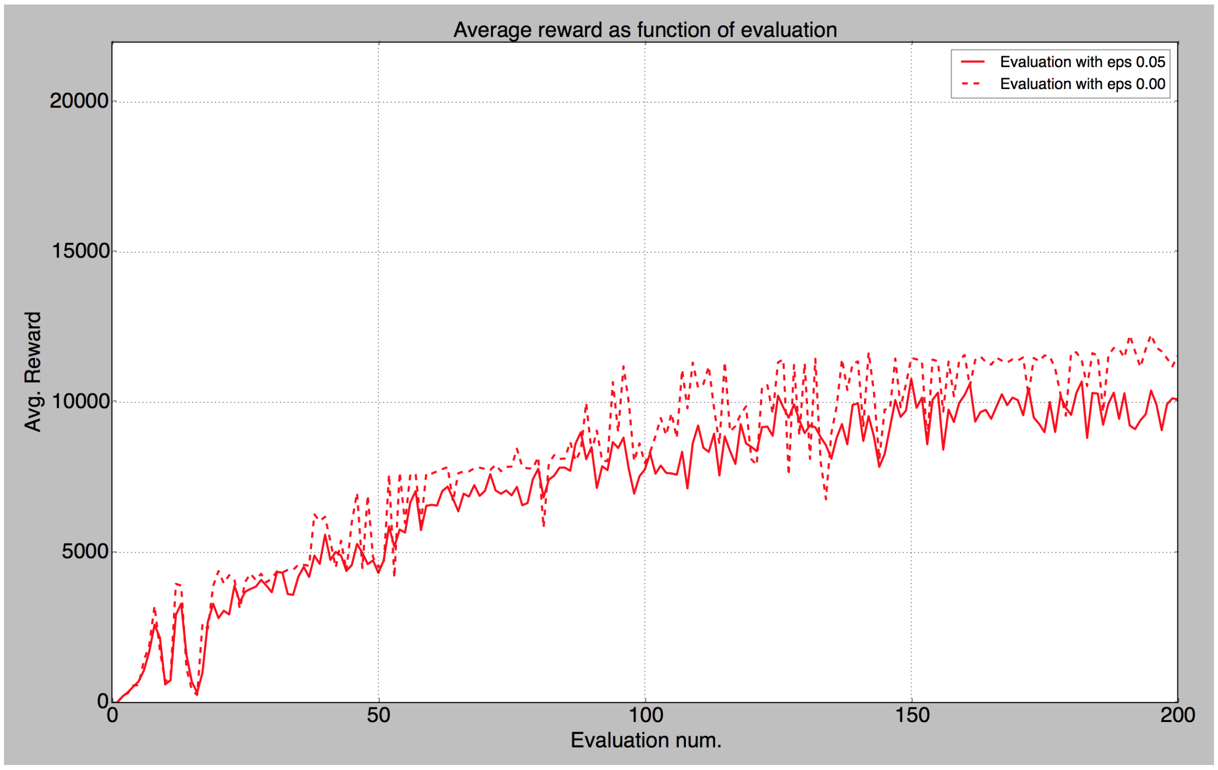


Figure 4: Experiment 2 - Agent per level

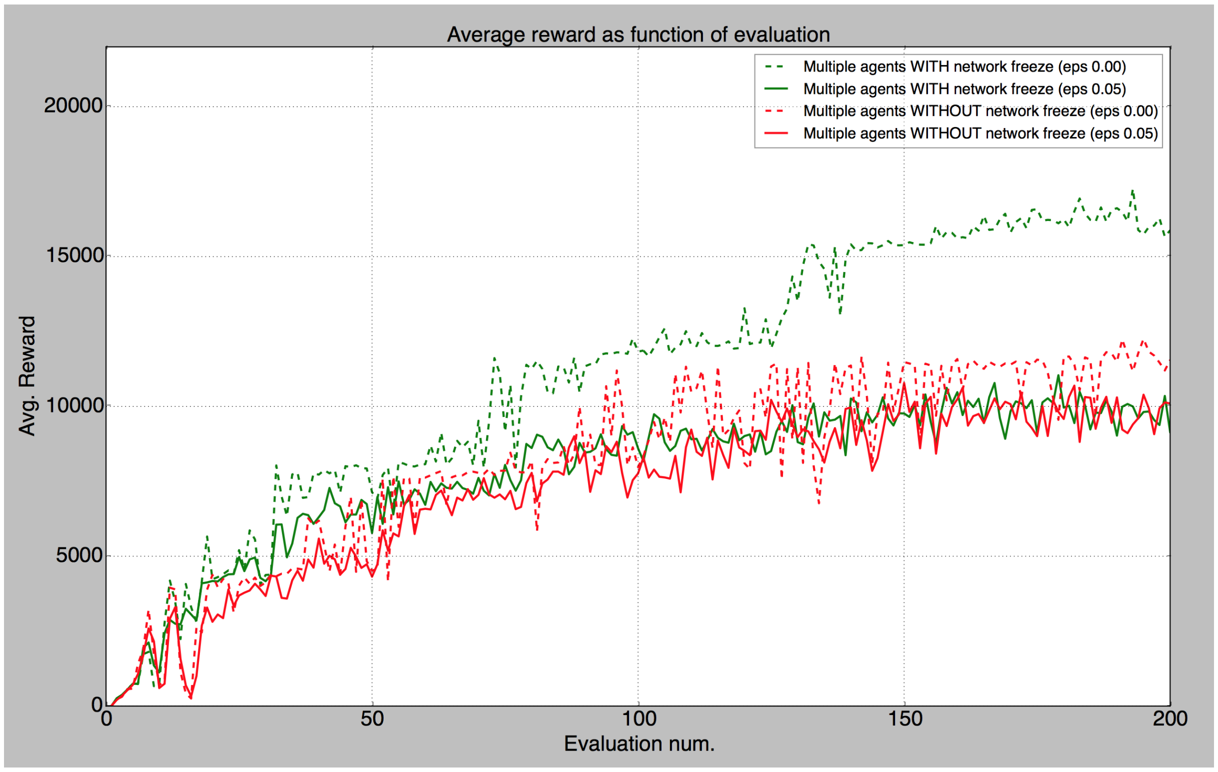
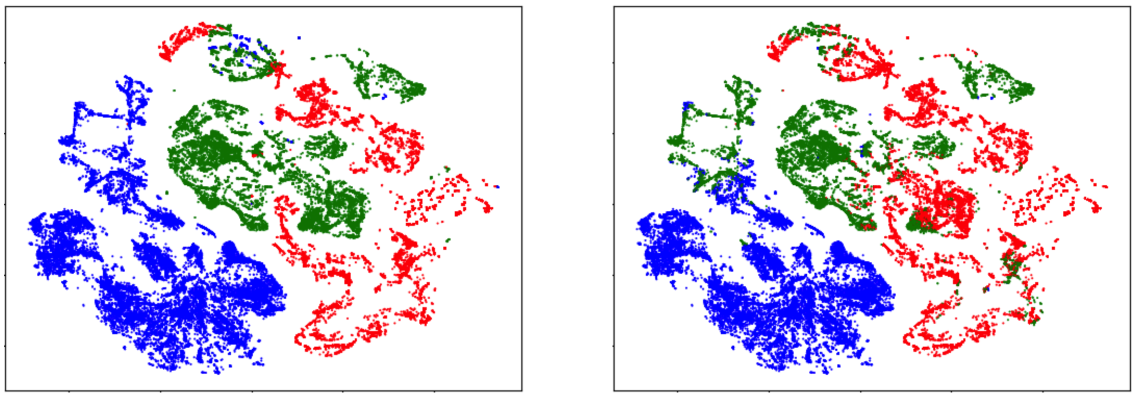


Figure 5: Experiment 3 - Agent per level with changing Epsilon, compared to Experiment 2 results



Correct level tagging Clustering by activations

Figure 6: Clustering results

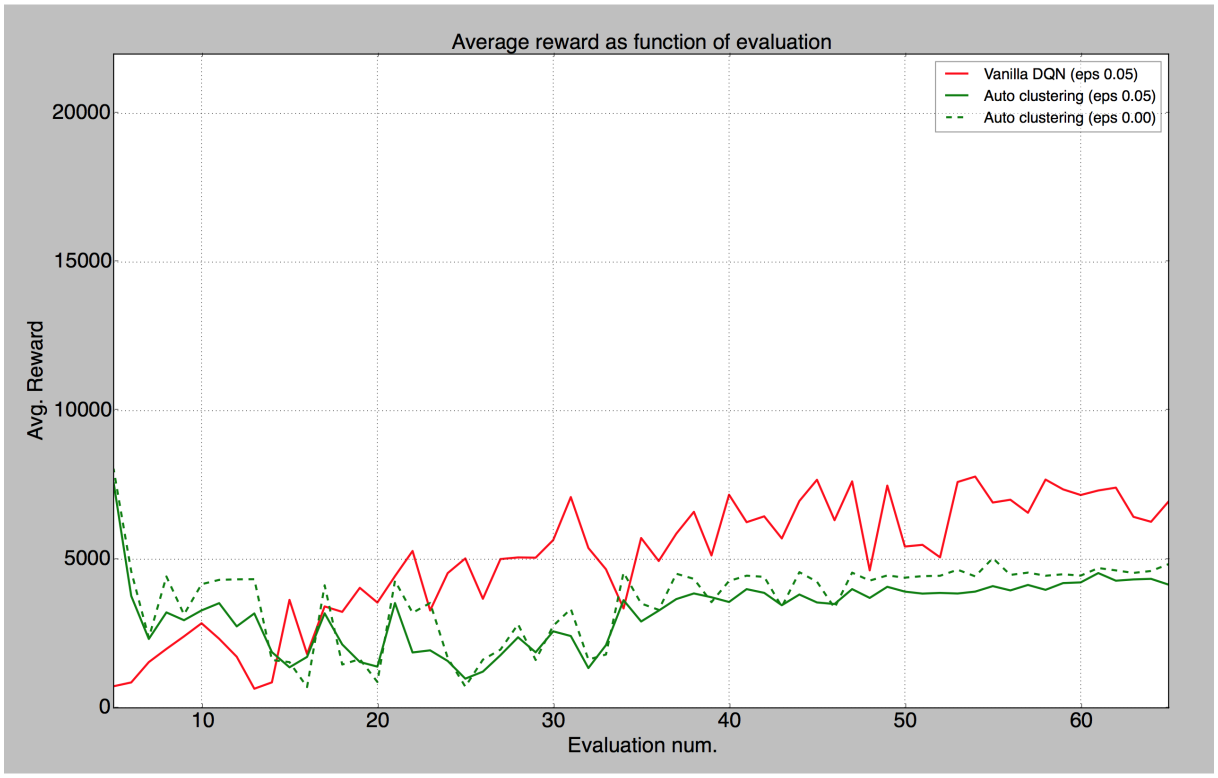


Figure 7: Experiment 4 - Auto clustering

Discussion

To conclude, we have discovered the following things:

- Controlling the exploration-exploitation ratio in a state-independent way throughout the learning and evaluating is crucial to effectively achieve high results.
- Using multiple agents can make the learning itself more independent and thus remove possible conflicts between states.
- Using a spatio-temporal clustering method as described above can be a general way to cluster different types of learning-based platforms. This way, many games which previously performed poorly, will now probably achieve significantly higher results.

Automatic clustering of tasks can lead to a significant improvement in life-long learning. For example, Tessler et al. (2017) used multiple networks for different skills, but the skills were determined manually. Letting the agent divide the learning process into sub-tsaks will generalize this solution better. Advantages of the method:

- The greatest advantage of this method is that it can be quite simply be implemented into any learning-platform.
- This method will also probably improve the results compared to a regular Vanila DQN.

This method has a few disadvantages:

- Assumptions - This method relies on the fact that we can determine a threshold from which the agent performs near-perfectly. This, of course, doesn't apply to some of the learning-based platforms. This assumption is needed in order to determine when to begin reducing/increasing the exploration.
- Independency - If two agents are dependent of each other and one of them reaches the performance threshold, meaning that its exploration is significantly reduced, The other agent might have a difficulty in performing well and reaching its own threshold. This might be solved by randomly increasing exploration in previous agents when reaching stagnation at a specific agent but we had not tested this notion in our work.
- Memory - Using multiple agent requires a lot of memory, when on top of that we use the described clustering method, the process may take a considerable amount of time to achieve high results. This might be solved by using policy distillation as presented on Rusu et al. (2015).

We recommend that future work should first get the parameters tunes, and include testing this algorithm on other platforms to see whether its still gets better results. It would also be interesting to see how this algorithm would perform on platforms which achieve good results on the *vanila DQN*. We expect that the results will deteriorate but not significantly.

Citation

References

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 201611835.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295*.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, 1553–1561.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *AAAI*, 2094–2100.

Zahavy, T.; Ben-Zrihem, N.; and Mannor, S. 2016. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, 1899–1908.

Zrihem, N. B.; Zahavy, T.; and Mannor, S. 2016. Visualizing dynamics: from t-sne to semi-mdps. *arXiv preprint arXiv:1606.07112*.