Compressive Buffer in Model-Free Reinforcement Learning

COMP-767: Research Project Report

Anthony G. Chen
Department of Computer Science
McGill University

1 Introduction

Deep reinforcement learning (RL) brings together RL theories [1] with nonlinear function approximation, allowing for a principled way of solving difficult control tasks in high dimensions. This approach has seen many success recently, such as playing games at super-human levels [2; 3], and solving complex motor control tasks [4]. Many recent success are achieved using *model-free* RL, with one popular choice being the deep-Q network (DQN) [2] and its family [5; 6; 7; 8]. Specifically, DQN and other *off-policy* algorithms [1] keep a *memory buffer*: a non-parametric buffer storing single timepoint experiences. The RL agent samples its memory buffer for mini-batches of previous experiences which it learns from. This has the benefit of sample-efficiency (since previous experiences can be used multiple times) and more independent and identically distributed (i.i.d.) data, which may help with more stable training [2].

However, DQN is not without its flaws. Specifically, the *memory buffer* grows to be very large. In the original implementation [2], a buffer with 1,000,000 entries is used, with each entry containing a 84×84 pixel greyscale image (observation), reward, and action. A careful implementation will take a minimum of $\sim$ 7 GB of memory, with naive implementations taking many factors more. This results in an inability to scale DQN to more complicated observation spaces (such as high definition RGB video-cameras, which may be used for robotics systems), as well as being computationally costly and slow to train.

Similar to RL, there are many recent advancements in the use of generative models [9; 10; 11; 12], which learn the distribution of input data. Specifically, variational autoencoders (VAE) [10] not only allows for the modelling of the input distribution, but the encoded low-dimensional space also act as a *compressed* representation of the input.

One observation is that DQN stores all of its observations in their raw form (e.g. raw pixels from input images). This is wasteful. Instead, it would be more frugal if the input can be stored in a compressed form. The capability of VAE to do arbitrary learned compression make the two a suitable union.

This project investigates the possibility of doing model-free, off-policy RL where the memory buffer is stored in a compressed form. Specifically, a DQN agent learns to play the Atari game "McPacman" [13]. Instead of storing all memory entries as raw pixels, it learns a compression to store samples in their comrpessed form. A vector-quantized variational autoencoder (VQ-VAE) is used, which has a discrete (vector-quantized) latent space, converges quickly, and do not suffer from the "posterior collapse" issue [11]. The agent is empirically evaluated by its performance (i.e. cumulative reward) on the Atari game, and compared against a similarly sized agent which do not use compression.

[1] I will define model-free and off-policy in the subsequent section

2 Related Work

2.1 Experience Replay

RL agents learn through trial-and-error. This can be very expensive for real-world applications, where error can result in severe outcomes. To allow (model-free) learning to be more efficient, experience replay store past experiences and re-sample them for further learning [14]. Currently, most model-free, off-policy methods employ experience replay [2; 6; 7; 15; 8]. One specific example of this is the Deep Q Network (DQN) [2], which extends Q-learning [16] to work with function approximators in a stable way.

Naive experience replay is done by uniformly sampling the memory buffer. More refined ways of replay include prioritizing replay order based on learning error [6], or storing longer sequences of experiences [17]. However, all above methods rely on the presence of a large non-parametric memory buffer. The goal of this project is to reduce the memory usage through compression of the experience tuple stored in the buffer.

2.2 Continual Learning

Continual learning (CL) defines the problem of continuously learning a (possibly non-stationary) input stream of data, without "catastrophically" forgetting previously learned skills [18]. While there are various approaches to get at this by putting constraints or dynamics on the network itself [19; 20; 21], I am mainly interested in CL approaches which store examples to be *replayed* during future task training [22; 23]. In fact, even simple replay buffers in CL can lead to state-of-the-art performance [22]. More sophisticated approaches have been taken, such as by generative modelling of previous examples [24].

RL is similar to CL in the sense that the input stream is continuous, and the agent must learn from a non-stationary distribution (since its experience distribution changes as its policy improves). Specifically, this project takes inpiration from [23], which uses a VQ-VAE to learn compression of input data and has "memory modules" consisting of compressed and uncompressed buffers to do experience replay in continual learning. Ideally, a similar approach can be used to reduce the memory usage in RL.

3 Background

3.1 Model-free RL

We define the RL environment to be a fully observable Markov decision process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, denoting the set of states, actions, transitions, and rewards, respectively. As the Atari environment [13] is actually only partially observable given a single frame (i.e. the velocity of moving objects cannot be inferred from a single frame), a common approach used is to stack the past 4 observations to form a single state [2].

The DQN algorithm is used [2]. DQN is a model-free algorithm in the sense that it does not learn a transition model (i.e. $\Pr(s'|s,a), s \in \mathcal{S}, a \in \mathcal{A}$) of the environment, but instead tries to learn a direct mapping (the Q function) between state-action pairs and (discounted) *return* directly [1]:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | S = s, A = a] = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S = s, A = a\right] \tag{1}$$

Where $Q(s,a)$ is the Q function under policy π, and $G_t = R_1 + \gamma R_2 + ...$ is the cumulative discounted reward (i.e. return). Intuitively, the Q function models the question: "if I take action a from state s (and follow policy π afterwards, how much total (discounted) reward do I expect to get?".

DQN is also an *off policy* algorithm as it learns using Q-learning [16]. A single Q learning step is defined by [1]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)) - Q(S_t, A_t)\right] \tag{2}$$

Where α is the learning rate. To do a single step of Q-learning, one requires an *experience tuple* - $(s_t, a_t, r_{t+1}, s_{t+1})$ - consisting of the current state, current action, reward received and the next state. As Q-learning does not depend on the current policy (note the $\max_a$ in Eq. 2 is greedy with respect to Q and does not depend on the current policy), *any* experience tuple can be used to update the Q network. Therefore it is sensible to store experience tuples in memory and re-use them multiple times for training. DQN stores 1 million such tuples.

3.2 Vector-quantized VAE

For learning compression, a vector-quantized variational autoencoder (VQ-VAE) is used [11]. A VQ-VAE models its prior as a discrete set of latent (quantization) vectors. When an input (x) is encoded $(z_e(x))$, it is mapped to the nearest latent vector (e_k), which is then used for decoding. The overall training objective is as follows [11]:

$$L = \log \Pr(x|z_q(x)) + ||\text{StopGrad}[z_e(x)] - e||_2^2 + \beta||z_e(x) - \text{StopGrad}[e]||_2^2 \tag{3}$$

Where the three terms optimize the reconstruction loss, the location of the quantization vectors, and the commitment of the encoder to the vectors, respectively. Unlike a "traditional" VAE [10], the prior is uniform discrete, not Gaussian. This also allows the KL-term to be ignored in the optimization process (see [11] for more details on how learning is done).

4 Compressive Replay Buffer

4.1 Architecture

The core DQN architecture is identical to the one used in [2]. However, the memory buffer has been modified (see Fig.1) such that it includes an *raw* (i.e. uncompressed) and a *compressed* component equipped with a VQ-VAE. As the agent interacts with the world, it receives an observation (i.e. an Atari game frame) at every time-point. The observation is passed through VQ-VAE to train it and evaluate the reconstruction loss (measured in mean squared error). Should the reconstruction loss be lower than a certain threshold, the observation is encoded and stored in the *compressed buffer* (Fig.1, top pathway). If the observation cannot be well-reconstructed, it is instead stored in the *raw buffer* (Fig.1, bottom pathway). For training the policy network, the agent samples with 0.5 probability either from the raw or the compressed buffer a minibatch of experience tuples for Q-learning (if the compressed buffer is sampled the samples are first decoded using the current decoder).

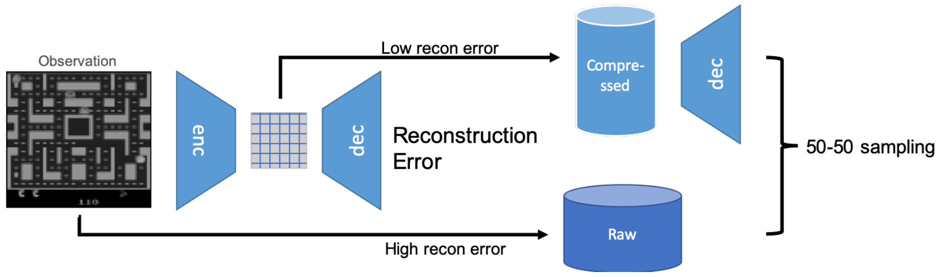


Figure 1: Set-up of the memory buffer and the VQ-VAE. All incoming observations are encoded, with ones that can be well-reconstructed stored in the compressed buffer. The buffers are sampled 50-50 to train the Q-network.

4.2 Experiments

A number of experiments are ran with the goal of reducing memory usage. All experiments are ran using the OpenAI gym [25] and the Atari environment [13]. Preprocessing of the environment and input frames are done in the same standard as [2]. The `McPacman-v0` environment is used as it is easy to show different levels of performance (and the VQ-VAE has to deal with the difficult task of reconstruction individual Pacman "snacks" which are only a few pixels each but hold high gameplay significance). Three random seeds are used for all trials and returns are averaged over 100 episodes. All agent-related code is written in PyTorch [26].

There are two main unknowns associated with using a compressed buffer: (i) how much raw memory is needed for DQN to be able to learn; (ii) can compressed memory be a useful substitute for raw memory. Since there is no good intuition on either, I first reduced the size of the raw memory buffer until DQN experiences a degradation in its learning capabilities. After this point, I will add the compressed buffer in hopes of "rescuing" DQN's performance.

Other empirical experimented are ran. I tested the effect of different thresholds on the reconstruction error. In theory, a strict (i.e. low) threshold would result in the compressed buffer storing only experiences which can be decoded with high quality. However, this might come at a cost of fewer examples being stored. On the contrary, a lenient (high) threshold would allow many examples to be stored in the compressed buffer, but at the cost of their quality.

Next, I tried the effect of different objective weighing. Since I primarily value good reconstruction (for DQN training), I evaluated the effect of reducing the weight on the latent loss (3rd term in equation 3) to encourage good reconstruction.

Additionally, since the VQ-VAE is continuously learning *online*, one possibility is that the encoder-decoder representation can change over time, resulting in catastrophic forgetting of previously learnt representation and poor decoding [18]. I run experiments where I *freeze* the VQ-VAE after an initial warm-up period (of 500,000 training steps), and compare its performance with a continuously learning VQ-VAE.

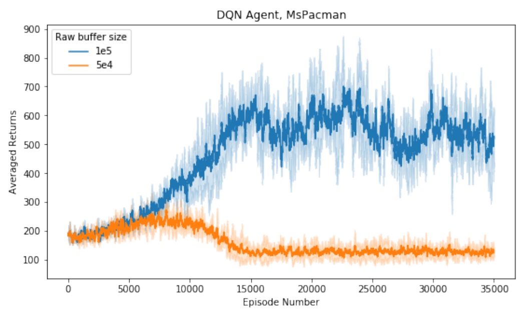
Finally, I attempt to train an agent using *only* compressed memories and compare its performance with an agent trained with both compressed and raw memories.

5 Empirical Results

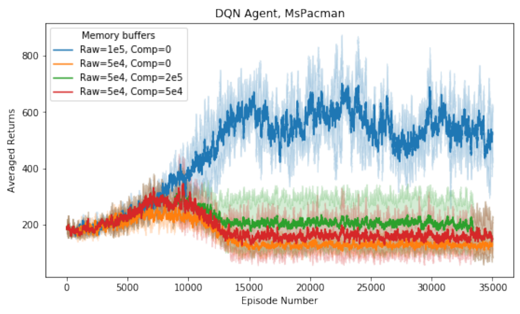
Unless otherwise stated in the experiment, all agents are trained with a set of default parameters which are close to the parameter set used in [2]. See A.1 for more details.

5.1 Compressed buffer adds small performance improvement to small raw buffer

As one decreases the (raw) memory buffer capacity of the DQN agent, a drop below approximately 100,000 buffer entries results in the agent losing its ability to learn - specifically when comparing between an agent with 100,000 raw buffer entries and another with 50,000 entries (Fig.2a). I attempt to "rescue" this agent by adding a *compressed buffer* with sizes: (i) 50,000, matching the total buffer entry sizes of 1e+5; and (ii) 200,000, matching the total physical memory usage.



(a) With only raw buffers



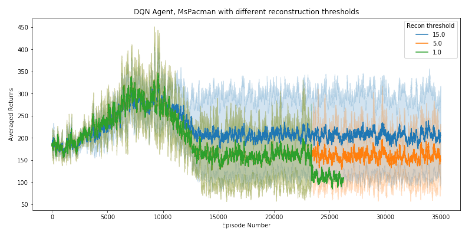
(b) Raw buffer + compressed buffer. `Raw` and `Comp` refer to raw and compressed (buffer sizes)

Figure 2: DQN performance at different sizes of raw and compressed buffers. `Comp=5e4` brings the total number of entries to 1e5, while `Comp=2e5` brings the total physical memory used to be the same as a raw buffer with 1e5 entries (the compression ratio is 4:1).

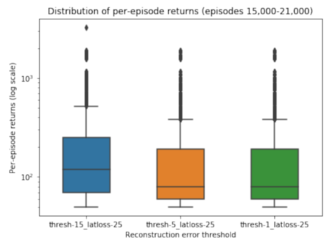
Evident from Figure 2b, adding the compressed buffer to the small raw buffer increased the performance, albeit by a small amount. Having a larger compressed buffer (of size 2e+5) resulted in a larger improvement. However, overall, adding the compressed buffer did not "rescue" the performance of the DQN agent to the same level as having a large, raw buffer (of size 1e+5).

5.2 More lenient reconstruction error threshold results in better performance

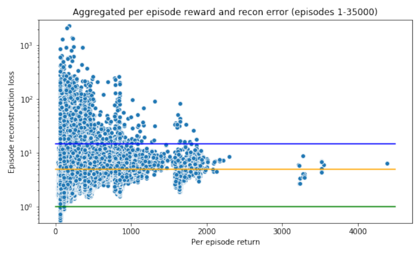
I investigated whether the *reconstruction error threshold* for compressed buffer storage plays a factor in determining the performance of the agent. This is done by running the default agent but changing the threshold from 15 to 5, then 1. Evident from Figure 3a, 3b, having a more *lenient* error threshold of 15 resulted in better performance over having stricter thresholds for compressed buffer storage (p=2.4e-95, t-test between independent samples).



(a) Average episode return over time



(b) Distribution of per-episode return over a small range



(c) Correlation between per-episode return and (averaged) reconstruction error

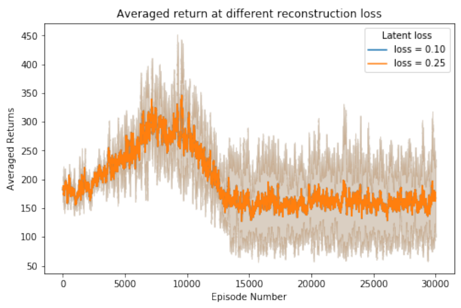
Figure 3: DQN performance at different reconstruction error (MSE) thresholds to put the compressed observation in buffer. Blue, orange and green refer to a threshold of 15, 5 and 1, respectively.

Figure 3c sheds some light onto why a higher threshold may be better. Having a threshold of 1 means that only a very limited subset of experiences are captured in the compressed buffer - typically episodes with little to no reward. In other words, the agent only stores the beginning few steps of a game in the compressed buffer, which is not very useful for learning (especially as the agent is sampling from the two buffers 50-50). Having an error threshold 15 means that a wide variety of experiences can be stored by the compressed buffer (evident by many of the high-return episodes having an average reconstruction error of <15 and being below the blue line in Fig 3c). This likely helps the agent to learn from a wider set of (albeit poorer reconstructed) experiences, resulting in better performance.

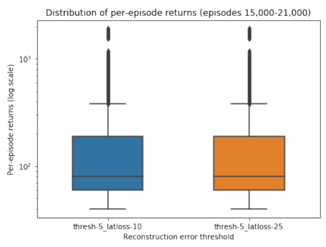
5.3 Higher reconstruction loss do not change performance

One reason for why having a larger compressed buffer does not result in a performance gain (over an equal-size raw buffer) is that the frames are not well-reconstructed. This can potentially be addressed by increasing the weight associated with reconstruction error in the VQ-VAE's optimization objective. Equivalently, I attempt to encourage better reconstruction by lowering the weight on latent loss in Equation 3 (3rd term) from the default 0.25 to 0.1.

Evident from Figure 4, encouraging better reconstruction at the level I tested did not bring about a noticeable change in the performance of the agent. This corresponds to the original author's observation that the algorithm is quite robust to changes in β in Equation 3 [11]. A larger different in β should be tested before the effect of encouraging reconstruction in the objective function can be concluded.



(a) Averaged per-episode return

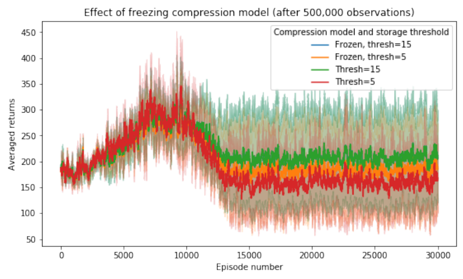


(b) Distribution of per-episode return over a small range

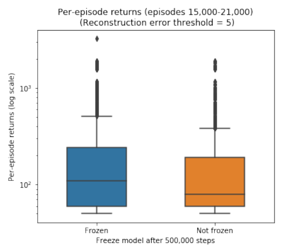
Figure 4: DQN performance with different values of latent loss in the VQ-VAE objective. Lower latent loss encourage better reconstruction.

5.4 Freezing the VQ-VAE improves performance slightly

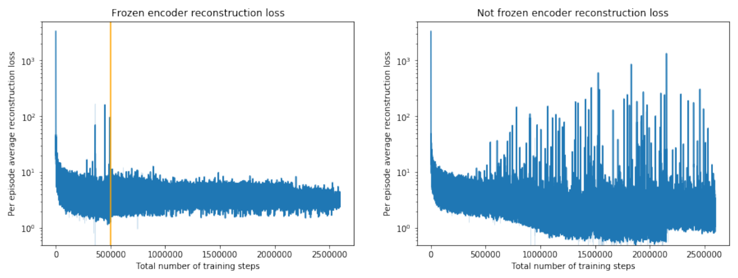
Another possibility for why having a compressed buffer does not increase performance by much may be because the VQ-VAE is continuously learning from a non-i.i.d. input stream of information (i.e. the agent's sensory experience). In such a case, the representation formed by the VQ-VAE can shift over time, leading to poor decoding of previously stored examples. To overcome this, I freeze the VQ-VAE training after an initial warm-up period of 500,000 training examples.



(a) Averaged per-episode return



(b) Distribution of per-episode return for reconstruction error threshold of 5.



(c) Reconstruction error over training steps for error threshold of 5. Orange line indicates the timepoint after which the VQ-VAE is frozen.

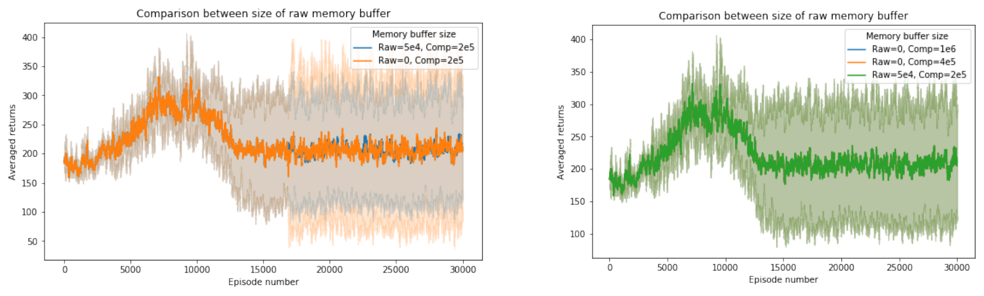
Figure 5: DQN performance with freezing the VQ-VAE training after 500,000 initial examples.

Evident from Figure 5a, freezing the VQ-VAE made little difference for an agent with a reconstruction error threshold of 15. However, freezing the VQ-VAE resulted in a small improvement if the reconstruction threshold is 5 (Fig.5b, p=4.29e-30, independent t-test). The effect of freezing the VQ-VAE is visualized in Figure 5c. The frozen VQ-VAE maintains a steady reconstruction error for the remainder of the run (also see A.3 for examples of raw and compressed images stored by the agent with a frozen VQ-VAE). The continually learning VQ-VAE, on the other hand, attempts to reduce the reconstruction error over time (evident by it reaching lower error), but the representation changes also likely results in "spikes" of high error rate. This may be why freezing the VQ-VAE is always a preferable option, though why it has no effect with an error threshold of 15 is left for further investigations.

5.5 Using only compressed buffer results in the same performance

Finally, I evaluate the potential for the agent to learn using *only* the compressed buffer. This is done by getting rid of the raw buffer altogether, but keeping the same compression scheme (i.e. a frame is only stored if it can be reconstructed with a low error) for the compressed buffer

In Figure 6a, I compare the performance of the best-performing compressed memory agent so far (raw buffer size of 5e4, compressed buffer size of 2e5) with a similar agent with equal amounts of compressed memory but no raw memory. Intriguingly, the two agents has exactly the same performance. This potentially suggest that the improvement in performance seen in Figure 2b may be largely attributed to the compressed buffer alone.



(a) Learning solely from compressed buffer with continually learning VQ-VAE

(b) Learning solely from compressed buffer with frozen VQ-VAE

Figure 6: Learning solely from a compressed buffer without a raw buffer.

Motivated by this, I re-run two more DQN agents with much larger compressed memories (1e+6 and 4e+5 entries, respectively) and no raw memories, as well as freezing the VQ-VAE after the initial warm-up period (Fig.6b). However, the two agents appear to perform exactly the same as the previous two (see A.4 for the reconstruction loss over each episode). It is likely that the performance achievable with a compressed buffer plateaus at the level observed in Figure 6, irrespective of how much larger the buffer gets. Exactly why this is the case is currently unclear.

6 Discussion

Interpretation of observation There appears to be an upper-bound on agent performance when learning using a compressed buffer. The reason for this is not completely clear. One explanation could be the error introduced by the learned lossy compression. Despite the fast convergence of the VQ-VAE, learning a good representation is difficult since the agent is encountering a continuous stream of non-i.i.d. data. Without a good policy, the agent is unlikely to get very far without dying, meaning that the input data is heavily imbalanced and dominated by the first few hundred frames of a game. This likely results in poor reconstruction once the agent ventures to previously unseen state spaces (which may be highly rewarding), preventing the agent from learning a better policy from the poorly reconstructed data. The vicious cycle continues.

Another limit in the method is that the raw and compressed buffers are sampled naively with 50% probability each, irrespective of their size or type of experience stored. As mentioned previously, a overly-strict error threshold means that the replay buffer will only store a few largely identical examples. On the other hand, an overly-lenient threshold means that the raw buffer will only store examples from the first few episodes (while the VQ-VAE has not yet produced a good compression). This, paired with the 50-50 sampling scheme, results in Q-learning often receiving very old experiences. Despite Q-learning being an off-policy algorithm, recent work have shown that when paired with function approximation, learning from examples far from the current policy can result in divergence of the value function [27].

It is also possible that the use of the learned compression has resulted in additional stochasticity in the Q-learning process. Specifically, when learning from decoded frames, the agent no longer has access to the full state of its environment, but must deduce the *true frame* from the lossy decoding produced by the VQ-VAE. This effectively turns the problem into a partially-observable MDP, which DQN is not built to handle.

Future work The limited time-frame of this project has allowed me to test with only 3 random seeds per experiment (as each experiment takes over a day to run). Since RL can face various reproducibility issues [28], it is prudent to re-run experiments with additional seeds to evaluate their performance. Furthermore, a more dynamic sampling scheme between the raw and compressed buffer should be devised, perhaps one that accounts for the recency of observation, or the current policy. Also, should one chooses to learn *only* from compressed data, an interesting avenue that is not well-explored is to learn directly from the *latent space representation* of the VQ-VAE, similar to [29]. Finally, another generative model can be learned over the latent space of the VQ-VAE (such as a PixelCNN [12], which has been paired with a VQ-VAE for image generation [30]) to generate previous examples, further reducing the memory need.

Conclusion

Overall, this project has provided a framework for compression of previous experiences in a model-free, off-policy RL setting. Various empirical results are shown to showcase the limits of online-learned compression, and interpretations are provided to guide future work.

Acknowledgments

I would like to thank Dr. Joelle Pineau, Dr. Blake Richards and Dr. Siamak Ravanbakhsh for their helpful comments about the project idea, as well as the various individuals in my COMP-767 class and at Mila for their discussions.

References

[1] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 2. MIT press Cambridge, 1998.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.

[5] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning. corr (2015),” *arXiv preprint cs.LG/1509.06461*, 2015.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[7] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449–458, JMLR. org, 2017.

[8] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[11] A. van den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems*, pp. 6306–6315, 2017.

[12] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[13] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[14] L. J. Lin, "Programming robots using reinforcement learning and teaching.," in *AAAI*, pp. 781–786, 1991.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[18] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[20] C. Kaplanis, M. Shanahan, and C. Clopath, "Continual reinforcement learning with complex synapses," *arXiv preprint arXiv:1802.07239*, 2018.

[21] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[22] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "Continual learning with tiny episodic memories," *arXiv preprint arXiv:1902.10486*, 2019.

[23] L. Caccia, E. Belilovsky, M. Caccia, and J. Pineau, "Online learned continual compression with stacked quantization module," *arXiv preprint arXiv:1911.08019*, 2019.

[24] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[27] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

[28] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters. arxiv, cs," 2017.

[29] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[30] A. Razavi, A. v. d. Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," *arXiv preprint arXiv:1906.00446*, 2019.

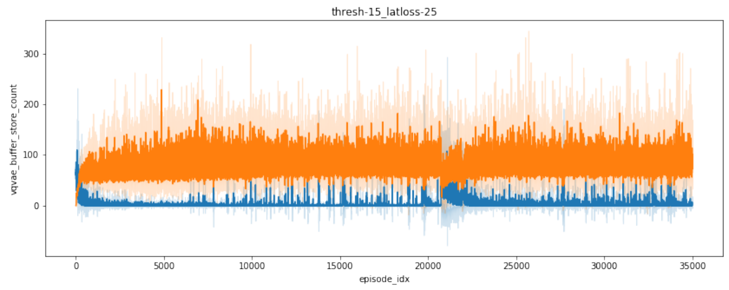
A Appendix

A.1 Default parameters for DQN and VQ-VAE training

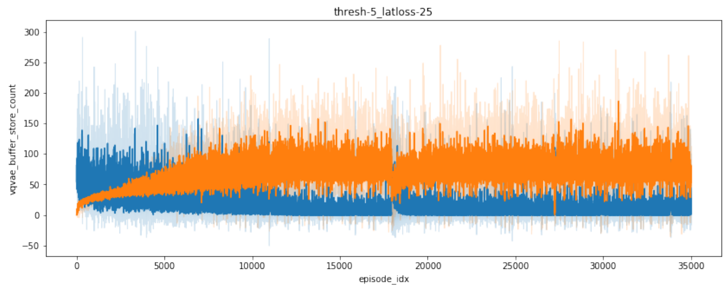
Parameter	Value
Atari environment name	MsPacman-v0
Number of frames to skip between observations	4
Number of observation to make a state	4
Number of actions between SGD updates	4
Number of actions between target network updates	4
Discount factor	0.99
Minibatch size for policy net	32
Initial exploration	1.0
Final exploration	0.1
Exploration decay period	1,000,000
Minimum observations before replay starts	50,000
VQ-VAE quantization vector dimension	1
VQ-VAE default reconstruction error threshold	15.0
Default raw buffer size	50,000
Default compressed buffer size	200,000
VQ-VAE default probability to sample compressed buffer	0.5
VQ-VAE default latent loss weight (β)	0.25

Table 1: Parameters for DQN and VQ-VAE. For DQN, the parameters are kept as close as possible to [2].

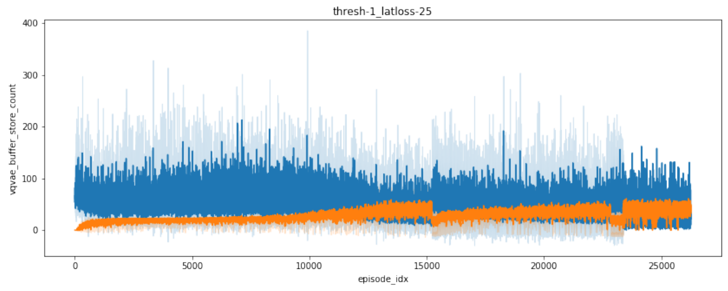
A.2 Storage into compressed buffer at different thresholds



(a) MSE threshold = 15



(b) MSE threshold = 5



(c) MSE threshold = 1

Figure 7: Per-episode examples stored into each buffers at different thresholds. Blue is storage into the raw buffer and orange is storage into the compressed buffer.

A.3 Example raw and reconstructed game frames

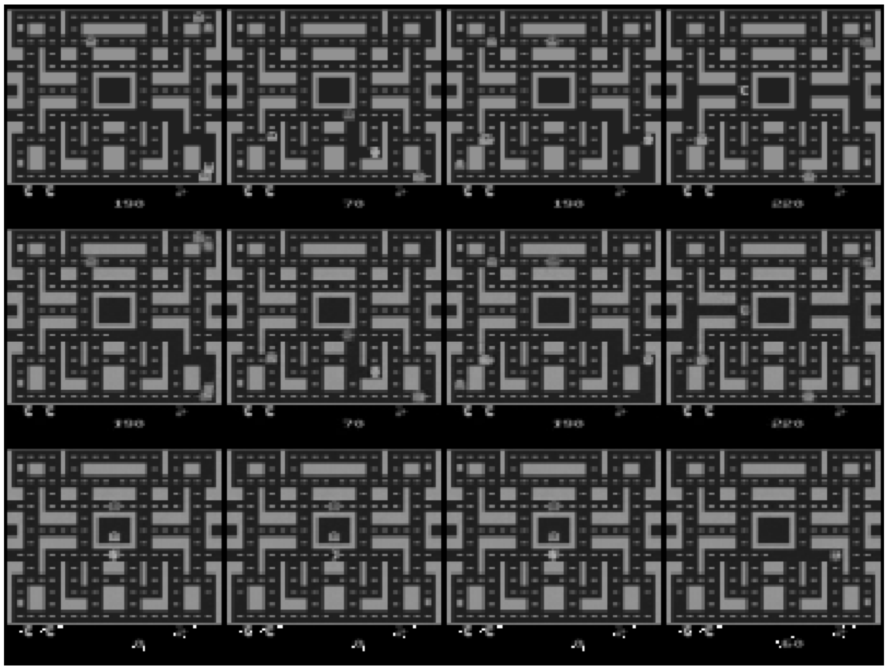


Figure 8: Example game frames from `McPacman-v0` on episode 30,000. Agent has a frozen VQ-VAE and a storage reconstruction error threshold of 5.0. The first row are samples from the raw buffer. Second row is reconstructed versions of the first row using the VQ-VAE. Third row are decoded samples from the compressed buffer.

A.4 Reconstruction loss of VQ-VAE of agent with no raw buffer

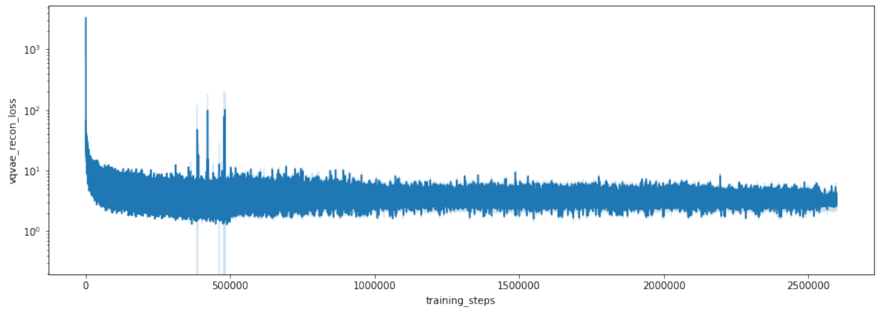


Figure 9: Reconstruction loss over training steps

B Software Note

The code use in the RL experiments can be found here: `https://github.com/im-ant/COMP767-PGM/tree/master/final_project`