

---

# Memory-Efficient Episodic Control Reinforcement Learning with Dynamic Online $k$ -means

---

**Andrea Agostinelli**

Department of Bioengineering  
Imperial College London  
aa7918@ic.ac.uk

**Kai Arulkumaran**

Department of Bioengineering  
Imperial College London  
kailash.arulkumaran13@imperial.ac.uk

**Marta Sarrico**

Department of Bioengineering  
Imperial College London  
mvs918@ic.ac.uk

**Pierre Richemond**

Data Science Institute  
Imperial College London  
phr17@ic.ac.uk

**Anil A. Bharath**

Department of Bioengineering  
Imperial College London  
a.bharath@imperial.ac.uk

## Abstract

Recently, neuro-inspired episodic control (EC) methods have been developed to overcome the data-inefficiency of standard deep reinforcement learning approaches. Using non-/semi-parametric models to estimate the value function, they learn rapidly, retrieving cached values from similar past states. In realistic scenarios, with limited resources and noisy data, maintaining meaningful representations in memory is essential to speed up the learning and avoid catastrophic forgetting. Unfortunately, EC methods have a large space and time complexity. We investigate different solutions to these problems based on prioritising and ranking stored states, as well as online clustering techniques. We also propose a new dynamic online  $k$ -means algorithm that is both computationally-efficient and yields significantly better performance at smaller memory sizes; we validate this approach on classic reinforcement learning environments and Atari games.

## 1 Introduction

Reinforcement learning (RL), using neural networks as function approximators, have surpassed human performance in a wide range of environments [18]. However, these approaches are sample-inefficient: they can require hundreds of times the experience of a human to reach the same level of performance during the early stages of learning [13]. Recently, new neuro-inspired algorithms known as episodic control (EC) methods [5, 19], implementing non-/semi-parametric models, have outperformed the speed of learning of state-of-the-art deep RL algorithms. EC methods rely on looking up past transitions from memory, where novel states are evaluated based on their similarity to past states. The lookups use  $k$ -nearest neighbours ( $k$ -NN) search, which is demanding in terms of both space and time complexity, and as the lookup operation is performed every time the agent encounters a new state, this makes it very difficult to scale up existing EC algorithms.

The novel problem that we aim to address, with the online nature of the data distribution in RL, is how to reduce the size of the memory in EC methods. And while reducing the memory size, the memory structure still has to support the RL agent in learning from recently-observed states without catastrophically losing past knowledge, referred to as the stability-plasticity dilemma in memory retention [1]. In consideration of these problems, our aim is to investigate and design novel memory storage approaches for EC algorithms that retain performance while reducing the amount of memory needed. We make the following contributions: Firstly, we evaluate 5 different memory storage strategies, including a novel online clustering algorithm, applied to 2 EC algorithms, in both

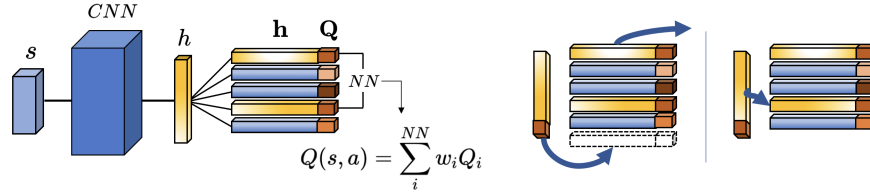


Figure 1: NEC inference and updating. (left) A state  $s$  is turned into an embedding/key  $h$  via a neural network, after which its  $Q$ -value is estimated through a weighted average among its nearest neighbours (NN). (right) New states are stored either by completely replacing another stored state, or merging it into an existing cluster of states.

classic RL environments and Atari games. Secondly, we show that replacing least-recently-used states or using online clustering techniques achieves the best performance across a range of settings and environments. Finally, we propose a new online clustering algorithm, which outperforms the other memory storage strategies when using smaller memory sizes.

## 2 Background

**Reinforcement Learning:** RL is the study of optimising the behaviour of an agent embodied in an environment. In the RL framework, the agent observes at timestep  $t$  the current *state*  $s_t$ , interacts with the environment using *action*  $a_t$ , thereby generating the transition to the successive *state*  $s_{t+1}$  and receiving a feedback signal (*reward*)  $r_{t+1}$ . The behaviour of an agent is controlled by the *policy*  $\pi(a_t|s_t)$ . The final goal is to learn the optimal policy  $\pi^*$  that maximises the expected return in the environment; the optimal policy  $\pi^*$  is defined as:  $\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}[R | \pi]$ , where the return  $R$  is the cumulative,  $\lambda$ -discounted reward of its sequence of experiences:  $R = \sum_{t=0}^{T-1} \lambda^t r_{t+1}$ ;  $\lambda \in [0, 1]$ .

**Episodic Control:** Memory and learning are supported in the brain by two main systems, the hippocampus and the neocortex. In particular, neocortical changes are related to long-term memory and learning of statistical models of sensory experiences, while hippocampal activity seems to perform fast instance-based learning on recent experiences; the latter system is thought to be the main location of rapid learning in humans [17]. Recently the behaviour of the hippocampus has been translated into a novel EC RL algorithm, where Q-value estimation is performed as weighted  $k$ -NN regression, emulating the hippocampal instance-based retrieval of the past.

In model-free EC (*MFEC*) [5] the agent is comprised of per-action tables  $Q^{EC}$  containing a list of the highest returns ever obtained by taking action  $a$  from state  $s$ :  $Q^{EC}(s, a)$ . MFEC is non-parametric, and uses either Gaussian random projections [10] or variational autoencoders [12, 20] to reduce the dimensionality of the observations. A semi-parametric EC model was introduced as neural EC (NEC) [19], a deep RL agent that uses a combination of a gradient descent to slowly improve the representation of the state through a neural network, and a quickly-updated value function through “differentiable neural dictionaries”, similar to MFEC per-action tables. The complete state-value estimation process is shown in Figure 1, where the weights for the average in the  $Q$ -value calculation are determined by the inverse distance weighted kernel:  $w_i = k(\mathbf{h}, \mathbf{h}_i) = \frac{1}{\|\mathbf{h} - \mathbf{h}_i\|_2^{2+\delta}}$ , where  $\mathbf{h}_i$  is a learned embedding of a state  $s_i$ . The parameters of NEC are updated through minimising the residual between the current Q-value and  $n$ -step episodic returns.

**Memory Storage Strategies:** After filling a memory with finite storage, there are two main strategies to deal with new observations, as illustrated in Figure 1; **prioritising** the storage of states with specific characteristics and reducing the number of elements with **clustering techniques**.

In MFEC and NEC the update of the memory is done by removing the least-recently-used entries. Outside of EC, Isele *et al.* [9] illustrated four selection strategies to manage the storage of past transitions  $\{s_t, a_t, s_{t+1}, r_{t+1}\}$  in experience replay [15]: prioritising surprising states, favouring higher rewards, narrowing the set of experiences trying to match the global distribution of states, and maintaining a heterogeneous distribution of the state-space.

Generally, when the agent operates in a streaming setting, using data that is not independent and identically distributed (i.i.d.), conventional first-in-first-out (FIFO) memory buffers will fail due to the

non-i.i.d. data stream [8]. Clustering algorithms are an effective class of methods that can mitigate catastrophic forgetting, reducing the number of redundant datapoints. Classic clustering techniques have the disadvantage of being applicable only to static sets of data, but online clustering methods can be applied successfully in streaming settings [8].

### 3 Implemented Strategies

#### 3.1 Prioritised Memories

Using MFEC and NEC as baseline EC algorithms, we investigate the influence of different memory storage strategies. The following strategies apply when the memory is completely filled; a ranking rule prioritises the replacement of the states in the memory.

**Maximise Surprise (SUR)** [9]: States with the highest prediction error (surprise) could be more useful to store, as compared to states that are well-predicted with the current memory. The surprise of a given state is calculated as the absolute value of the difference between the future discounted return and its predicted  $Q$ -value; therefore we drop the state  $\mathbf{s}$  with the minimum surprise:

$$\operatorname{argmin}_{\mathbf{s} \in \text{memory}} |R - Q(\mathbf{s}, \mathbf{a})|.$$

**Maximise Rewards (REW)** [9]: States with the highest return may be the best to retain. With this strategy the state with the lowest return is replaced:

$$\operatorname{argmin}_{\mathbf{s} \in \text{memory}} R.$$

**Least Recent Used (LRU)** [5]: The state that has been least recently used by the memory is replaced by the new state. This is the default strategy for MFEC and NEC, inspired by the behaviour of the brain in forgetting old experiences.

#### 3.2 Clustering Techniques

An alternative to ranking strategies is to reduce the number of states by merging them into clusters, with newly observed states merged with their nearest (Euclidean) neighbour.

**Online  $k$ -means (kM)**: There are several versions of the “online  $k$ -means“ algorithms [22, 14, 11], but we use the computationally-efficient and simple version from [8]. After filling the memory, a separate vector  $\mathbf{n}$  is used to store the number of elements of each cluster, and after observing a new state the memory is updated by:

$$\mathbf{c}_i \leftarrow \frac{n_i \mathbf{c}_i + \mathbf{s}}{n_i + 1}, \quad Q_i \leftarrow \frac{n_i Q_i + R}{n_i + 1}, \quad n_i \leftarrow n_i + 1,$$

where  $\mathbf{c}_i$  is the closest neighbour to the new state  $\mathbf{s}$  with associated  $Q$ -value  $Q_i$ , and  $n_i$  is the number of elements of  $\mathbf{c}_i$ , subsequently incremented by one.

**Dynamic online  $k$ -means (DkM)**: This is our novel clustering algorithm, based on online  $k$ -means. We use a new heuristic for the update, based on the memory size  $N$ :

$$\mathbf{c}_i \leftarrow \frac{n_i \mathbf{c}_i + \mathbf{s}}{n_i + 1}, \quad Q_i \leftarrow \frac{n_i Q_i + R}{n_i + 1}, \quad n_i \leftarrow n_i + 1, \quad \mathbf{n} \leftarrow \mathbf{n} - \frac{1}{N}.$$

In comparison to online  $k$ -means, the “size” of the clusters is reduced over time. Once  $n_i \leq 0$  the cluster is completely replaced by the new state  $\mathbf{s}$ , with  $n_i$  reset to 1. Our heuristic, inspired by LRU, aims to delete clusters that are not frequently encountered, and also prevents clusters from becoming static.

Once the nearest neighbour is found for a memory of size  $N$ , kM has an additional complexity of  $O(1)$ , and DkM has an additional complexity of  $O(N)$ , which is negligible compared to the original lookup ( $O(N^2)$  for naive  $k$ -means or  $O(dN \log(N))$  for  $k$ -d/cover trees [4], with  $d$  proportional to the state dimensionality).

Table 1: Total rewards in the Cartpole domain, after  $2 \times 10^4$  (NEC) and  $1.5 \times 10^4$  (MFEC) steps; the MFEC agents typically converged within  $0.5 \times 10^4$  steps, while the NEC agents continued to improve. The values indicate the mean of the last 10 evaluations, averaged over 5 random seeds.

Memory Size per Action	MFEC					NEC					D3QN
	LRU	REW	SUR	kM	DkM	LRU	REW	SUR	kM	DkM	
50	96 ± 21	<b>180 ± 47</b>	51 ± 24	133 ± 66	123 ± 13	161 ± 15	107 ± 29	195 ± 88	136 ± 72	<b>326 ± 31</b>	±143
100	110 ± 9	134 ± 25	57 ± 15	108 ± 37	<b>153 ± 25</b>	235 ± 53	132 ± 42	215 ± 97	216 ± 159	<b>339 ± 32</b>	
150	117 ± 13	125 ± 30	83 ± 15	74 ± 21	<b>142 ± 26</b>	246 ± 22	167 ± 51	215 ± 36	151 ± 56	<b>290 ± 24</b>	
500	142 ± 19	61 ± 7	102 ± 12	<b>244 ± 131</b>	174 ± 34	247 ± 43	117 ± 33	218 ± 48	211 ± 79	<b>271 ± 37</b>	
1000	<b>197 ± 37</b>	49 ± 13	151 ± 36	196 ± 26	181 ± 18	220 ± 46	131 ± 39	145 ± 44	237 ± 65	<b>257 ± 46</b>	
3000	276 ± 31	45 ± 8	183 ± 18	<b>359 ± 81</b>	209 ± 13	254 ± 42	110 ± 44	220 ± 15	<b>300 ± 61</b>	255 ± 50	
5000	259 ± 43	54 ± 13	210 ± 23	<b>352 ± 86</b>	213 ± 34	282 ± 38	114 ± 14	219 ± 32	<b>343 ± 97</b>	302 ± 53	
10000	325 ± 55	65 ± 11	266 ± 28	<b>344 ± 30</b>	228 ± 15	309 ± 23	124 ± 42	250 ± 27	<b>330 ± 27</b>	265 ± 25	

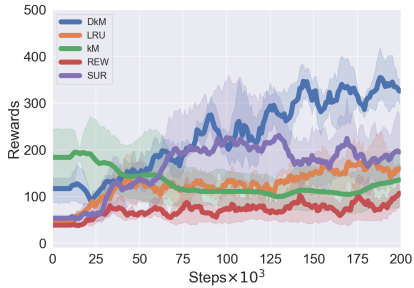


Figure 2: Learning on Cartpole using NEC and a small memory size of 50. DkM is the only method that reaches average total rewards > 200.

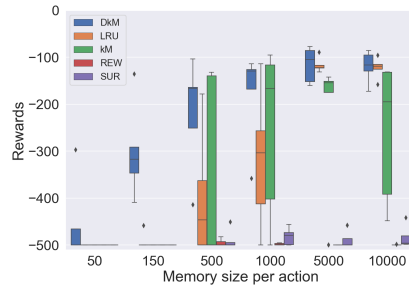


Figure 3: Total rewards on Acrobot, using NEC and different memory sizes. Overall, DkM yields high return episodes with low variance.

## 4 Experiments

We investigated the importance of using alternative memory strategies in both simple and complex RL domains, showing results for different memory capacities. We use the original configurations of MFEC and NEC as EC baselines, with the dueling double DQN (D3QN) as a baseline [18, 7, 21]. We tested our implementations in three sets of environments: classic control [6], room domains (re-implemented from [16]), and Atari games [3]. For every environment we use a discount factor  $\lambda$  of 0.99; a full set of hyperparameters per environment are available in Section 10. Every agent is evaluated with an argmax policy over 10 different episodes every evaluation interval.

**Classic Control:** We considered Acrobot and Cartpole, which have a continuous state space and 3 and 2 actions respectively. Table 1 summarises results for Cartpole. Clustering techniques perform the best. In particular DkM applied to NEC consistently shows the best performance for low buffer sizes. Memory size 50, where DkM significantly outperforms the other techniques, is illustrated in Figure 2. In Acrobot, DkM outperforms all the other methods, as shown in Table 2. Here our novel method achieves the best rewards in 10 out of 12 settings, and always when using memory sizes less than 5000. In the final 2 settings (MFEC with memory sizes 5000 and 10000), DkM is second only to LRU. With NEC, DkM is always the best memory storage method (Figure 3).

**Gridworld:** We recreated two room environments, OpenRoom and FourRoom. The agent can move in 4 directions and is given a reward of 1 only when the agent moves to the goal. OpenRoom is a 10x10 open room, and FourRoom is a set of four interconnected rooms, totalling 11x11. Clustering techniques are particularly efficient in the OpenRoom domain, with kM performing the best at

Table 2: Total rewards in the Acrobot domain, after  $2 \times 10^4$  steps. The values indicate the mean of the last 10 evaluations, averaged over 5 initial random seeds for all methods.

Memory Size per Action	MFEC					NEC					D3QN
	LRU	REW	SUR	kM	DkM	LRU	REW	SUR	kM	DkM	
50	-500	-500	-304 ± 28	-500	<b>-298 ± 108</b>	-500	-500	-500	-500	<b>-453 ± 79</b>	-336 ±149
150	-499 ± 2	-500	-431 ± 136	-500	<b>-165 ± 45</b>	-492 ± 17	-500	-500	-500	<b>-300 ± 91</b>	
500	-486 ± 19	-500	-493 ± 9	-425 ± 149	<b>-164 ± 41</b>	-397 ± 121	-495 ± 7	-489 ± 19	-354 ± 179	<b>-220 ± 108</b>	
1000	-387 ± 87	-500	-479 ± 42	-430 ± 141	<b>-345 ± 190</b>	-317 ± 132	-498 ± 2	-482 ± 17	-256 ± 164	<b>-179 ± 92</b>	
5000	<b>-368 ± 167</b>	-500	-499 ± 1	-478 ± 44	-397 ± 155	-116.4 ± 14	-500	-489 ± 16	-225 ± 138	<b>-115.8 ± 34</b>	
10000	<b>-284 ± 173</b>	-500	-497 ± 5	-497 ± 4	-395 ± 132	-123 ± 20	-500	-483 ± 22	-260 ± 134	<b>-120 ± 30</b>	

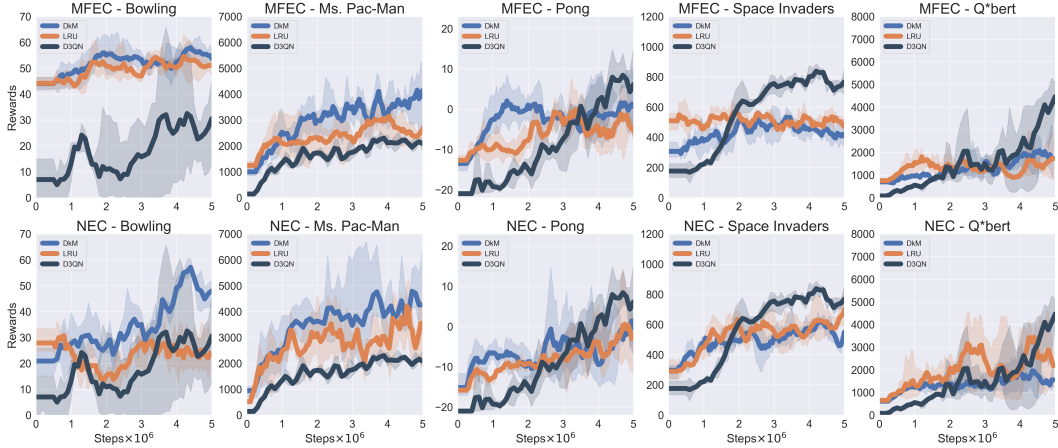


Figure 4: Learning curves of 5 Atari games using MFEC, NEC and D3QN. EC methods with a memory size of  $10^4$  per action. DkM generally improves upon LRU.

Table 3: Mean and standard deviations of final scores calculated across 3 seeds on 5 Atari games, using a memory size of  $10^4$  (on the left) and  $10^5$  (on the right). Total rewards calculated after  $5 \times 10^6$  steps, and  $3.5 \times 10^6$  for NEC when using a memory size of  $10^5$  per action.

Atari Game	MFEC $10^4$ size		NEC $10^4$ size		MFEC $10^5$ size		NEC $10^5$ size		D3QN
	LRU	DkM	LRU	DkM	LRU	DkM	LRU	DkM	
Bowling	$51.1 \pm 2.8$	<b><math>53.8 \pm 2.8</math></b>	$23.9 \pm 6$	<b><math>48 \pm 5</math></b>	<b><math>62.8 \pm 1.8</math></b>	$61 \pm 2.9$	$21.6 \pm 6.4$	<b><math>43.8 \pm 13.5</math></b>	$30.5 \pm 13.6$
Ms. Pac-Man	$2660 \pm 252$	<b><math>4143 \pm 1153</math></b>	$3556 \pm 783$	<b><math>4264 \pm 1558</math></b>	<b><math>5245 \pm 456</math></b>	$2862 \pm 1388$	<b><math>5225 \pm 856</math></b>	$4092 \pm 856$	$2090 \pm 275$
Pong	$-5.3 \pm 1.9$	<b><math>0.63 \pm 5.3</math></b>	$-2.9 \pm 6.3$	<b><math>-0.35 \pm 4.3</math></b>	<b><math>18.6 \pm 1.4</math></b>	$15.3 \pm 2.8$	$11.1 \pm 3$	<b><math>15.9 \pm 1.6</math></b>	$6.1 \pm 9.6$
Space Invaders	<b><math>472 \pm 28</math></b>	$408 \pm 60$	<b><math>694 \pm 105</math></b>	$549 \pm 42$	<b><math>806 \pm 88</math></b>	$430 \pm 61$	<b><math>854 \pm 67</math></b>	$526 \pm 88$	$767 \pm 72$
Q*bert	$1724 \pm 846$	<b><math>1728 \pm 942</math></b>	<b><math>2168 \pm 685</math></b>	$1556 \pm 342$	<b><math>4930 \pm 787</math></b>	$1140 \pm 241$	<b><math>5093 \pm 1480</math></b>	$3389 \pm 745$	$4463 \pm 460$

memory sizes 2 and 5, and DkM performing best overall across all settings. DkM performs well with a memory size of 10 (or greater), which is 5 times smaller than what is required for the LRU method. However, in FourRoom only LRU was capable of perfectly solving the environment using high memory sizes, bigger than 150 per action. Tables and learning curves are provided in Section 8.

**Atari:** A set of five games with varied gameplay types are evaluated, namely: Ms. Pac-Man, Q\*bert, Pong, Space Invaders and Bowling. As in prior work, we use standard preprocessing of the visual observations [18], and use Gaussian random projections for MFEC [5]. As preferring either highly rewarding (REW) or highly surprising states (SUR) as the memory storage strategy achieved poor performance in both Classic Control and Gridworld domains, we did not evaluate these methods on Atari games due to limited computational resources. For this reason we also did not evaluate kM, as it was generally outperformed by DkM. We ran experiments for 5 million steps, where every agent step represents 4 game frames (action repeat of 4). Due to limited computational resources we ran experiments with NEC for 3.5 million steps, when using  $10^5$  items per action. As in the original papers [5, 19], we use 11 nearest-neighbours and a final  $\epsilon$  of 0.005 for MFEC, and 50 nearest neighbours and a final  $\epsilon = 0.001$  for NEC. We did, however, use a key size of 128 and the inverse distance weighted kernel for MFEC, as this performed the same or better than a simple average in our initial experiments, and makes these hyperparameters the same as for NEC.<sup>1</sup>

Figure 4 shows the learning curve of 5 different Atari games, using a memory size of  $10^4$  items per action. DkM performed the best across the experiments, compared to LRU (Table 3). In Bowling, Ms. Pac-Man and Pong, DkM achieves the best results, for either MFEC and NEC. The performance of EC methods increases using a bigger memory size of  $10^5$  items per action, where they show much better sample efficiency in comparison to fully parametric approaches such as the D3QN (Table 3). EC methods are faster in achieving high rewards. LRU outperforms DkM for high memory size; it always achieves better rewards in Space Invaders and Q\*bert for both MFEC and NEC.

<sup>1</sup>As a result we do not include the original results as hyperparameter changes mean they are not directly comparable.

## 5 Discussion

We investigated various methods to improve the memory complexity of EC methods, and proposed a novel online clustering algorithm (DkM) that outperforms other methods for small memory sizes. In simple environments, it can require 10 to 20 times less memory to effectively train the agent, as compared to the original LRU strategy (Tables 1, 2). We speculate that DkM outperforms the alternative methods because of its ability to reduce redundant information through clustering, and concurrently the ability to adapt to changes in the state distributions, as investigated in Section 6. An interesting observation is that while LRU tends to improve monotonically with an increase in the buffer size (Tables 1, 2), the online clustering methods do not share that property—investigating this could be an interesting avenue for further understanding the role of clustering representations in EC.

In Atari games, the improvements hold for smaller buffer sizes ( $10^4$ ), but not for larger buffer sizes ( $10^5$ ), where the original LRU strategy performs best. The operation of merging states into clusters could increase state aliasing, making it more difficult for an agent to properly differentiate such situations (e.g., small pixel differences). Possible improvements could be to make the clustering process itself dependent on the  $Q$ -values.

Overall, DkM allows the use of smaller memory sizes for EC methods, making them more memory- and computationally-efficient. Combined with the sample-efficiency of EC methods, we believe that it is a promising technique to apply to real-world scenarios where resources can be limited and the acquisition of new data expensive.

## References

- [1] Wickliffe C Abraham and Anthony Robins. Memory retention—the synaptic stability versus plasticity dilemma. *Trends in Neurosciences*, 28(2):73–78, 2005.
- [2] Adelchi Azzalini and Antonella Capitanio. Statistical applications of the multivariate skew normal distribution. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):579–602, 1999.
- [3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
- [4] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *International Conference on Machine Learning*, pages 97–104. ACM, 2006.
- [5] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.
- [6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [7] Hado V Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.
- [8] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *International Conference on Robotics and Automation*, pages 9769–9776, 2019.
- [9] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI Conference on Artificial Intelligence*, 2018.
- [10] William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.
- [11] Angie King. Online k-means clustering of nonstationary data. *Prediction Project Report*, pages 1–9, 2012.
- [12] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

- [13] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.
- [14] Edo Liberty, Ram Sriharsha, and Maxim Sviridenko. An algorithm for online k-means clustering. In *Meeting on Algorithm Engineering and Experiments*, pages 81–89. SIAM, 2016.
- [15] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.
- [16] Marios C Machado, Marc G Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pages 2295–2304, 2017.
- [17] James L McClelland, Bruce L McNaughton, and Randall C O’Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419, 1995.
- [18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [19] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836, 2017.
- [20] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.
- [21] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.
- [22] Shi Zhong. Efficient online spherical k-means clustering. In *IEEE International Joint Conference on Neural Networks*, volume 5, pages 3180–3185. IEEE, 2005.

# Appendix

## 6 Properties of Memory Storage Strategies

To better understand the behaviour of our DkN algorithm with respect to the other strategies, we created a simple synthetic 2D dataset and evaluated the  $k$ -means, online  $k$ -means, DkN and LRU methods on this data.

For the online methods, we simulated an online data stream first by a series of 2D points uniformly distributed on a grid, followed by a series of 2D points from a skew normal distribution [2], which represents random exploration of the state space, and then convergence. In order to qualitatively explore the properties of the different methods, we plotted a heatmap of the distribution of states using kernel density estimation.

In Figure 5a we notice that different strategies cover different parts of the data distribution.  $k$ -means applied to the whole dataset distributes the clusters across the entire support of the data independently of when the data was first observed. kM and LRU show completely different distributions; the former favours the first data distribution (uniform) due to the static nature of clusters, while the latter favours the second data distribution (skew normal) after forgetting the least-frequently-visited states. In comparison, DkM is able to retain old states, while biasing clusters towards the new distribution. Figure 5b shows the evolution of the clusters in DkM after observing 25, 50, 75 and 100% of the dataset. DkM behaves like a mix between kM, where the estimated data distribution covers the support well, and LRU, which prioritises new observations.

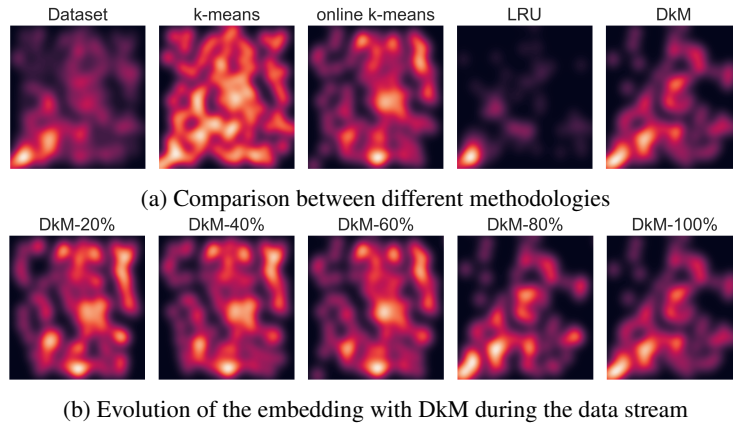


Figure 5: Comparison between state distributions generated from the different techniques. DkM shows the best approximation to the real dataset. Kernel density estimates have been fit separately for each method.



## 7 Classic Control

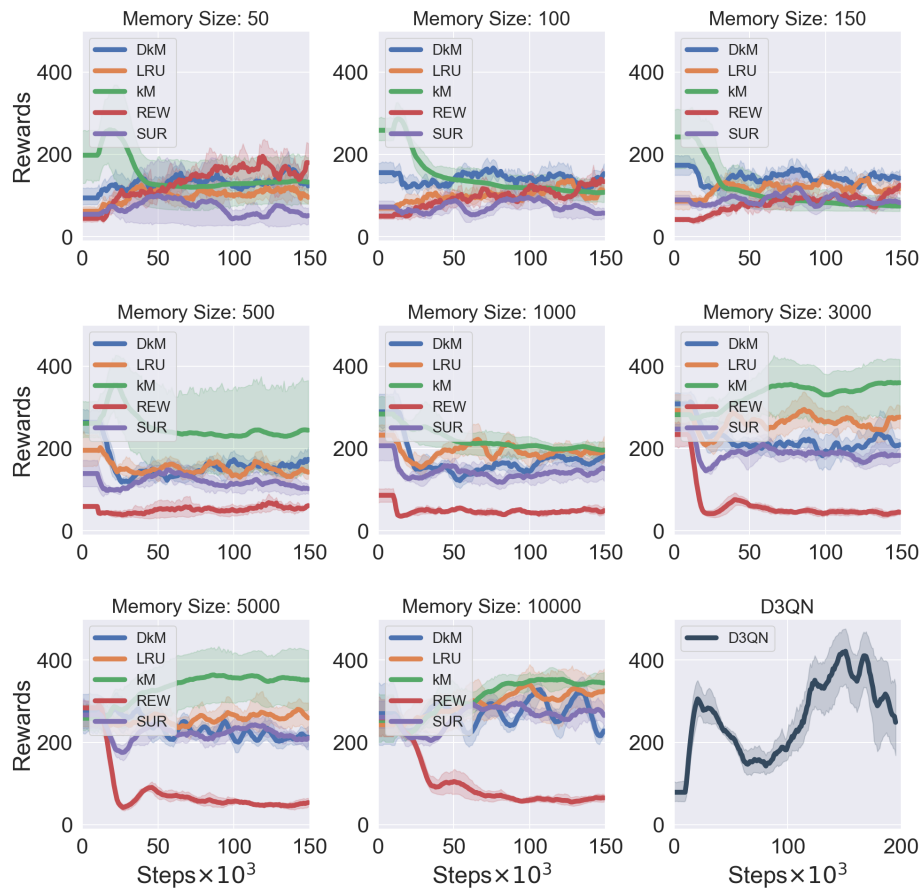


Figure 6: Learning curves of Cartpole for different memory sizes, using MFEC and D3QN

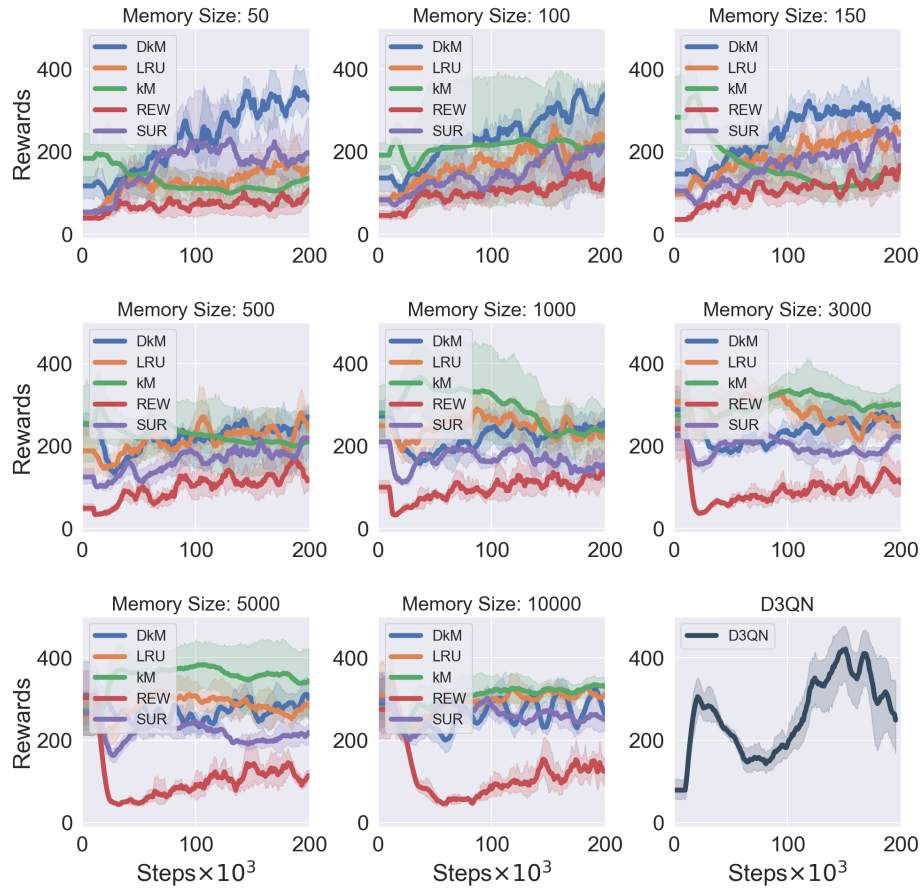


Figure 7: Learning curves of Cartpole for different memory sizes, using NEC and D3QN

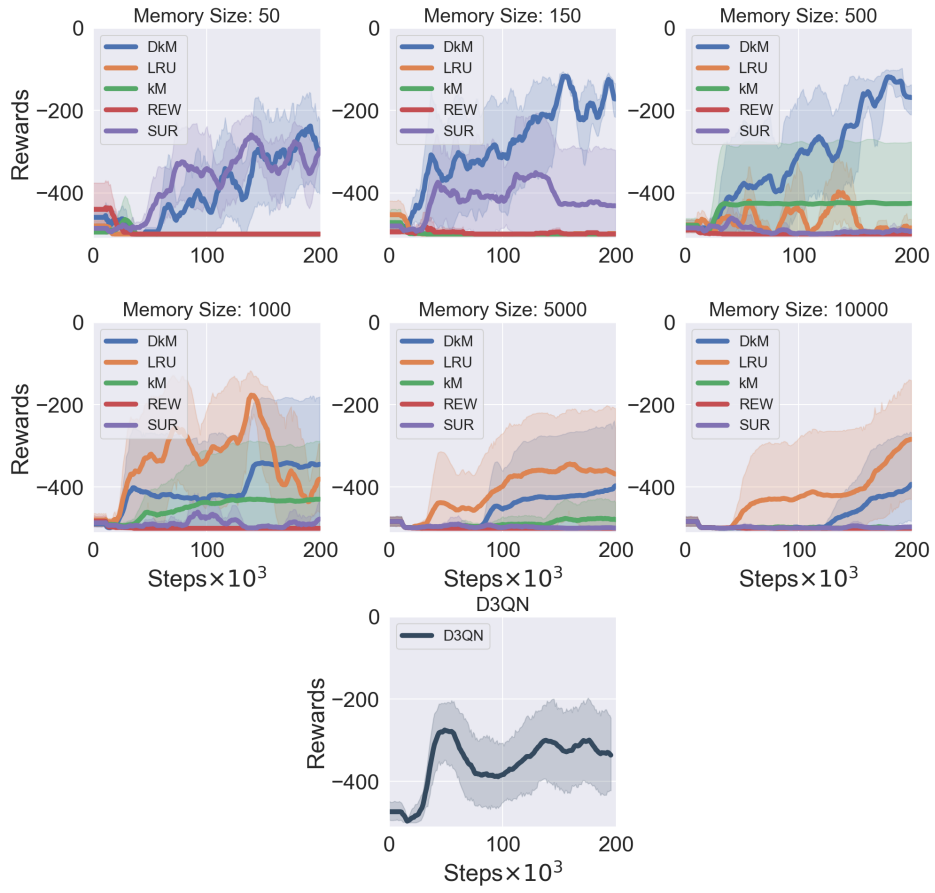


Figure 8: Learning curves of Acrobot for different memory sizes, using MFEC and D3QN

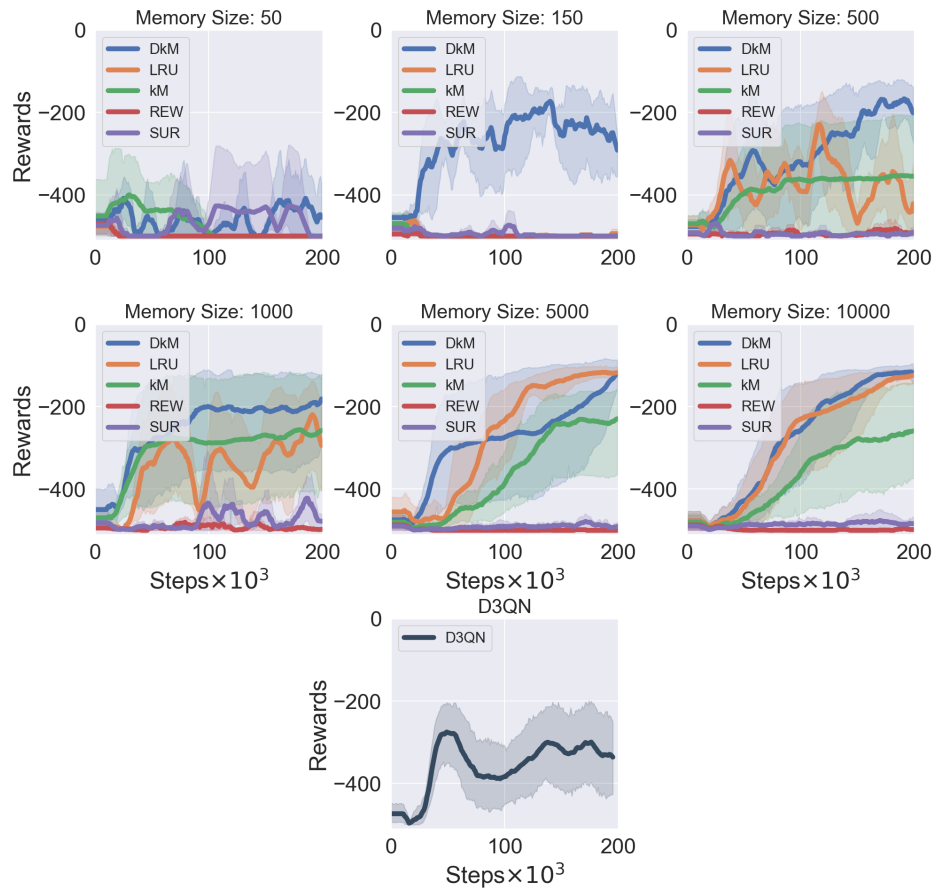


Figure 9: Learning curves of Acrobot for different memory sizes, using NEC and D3QN

## 8 Gridworld

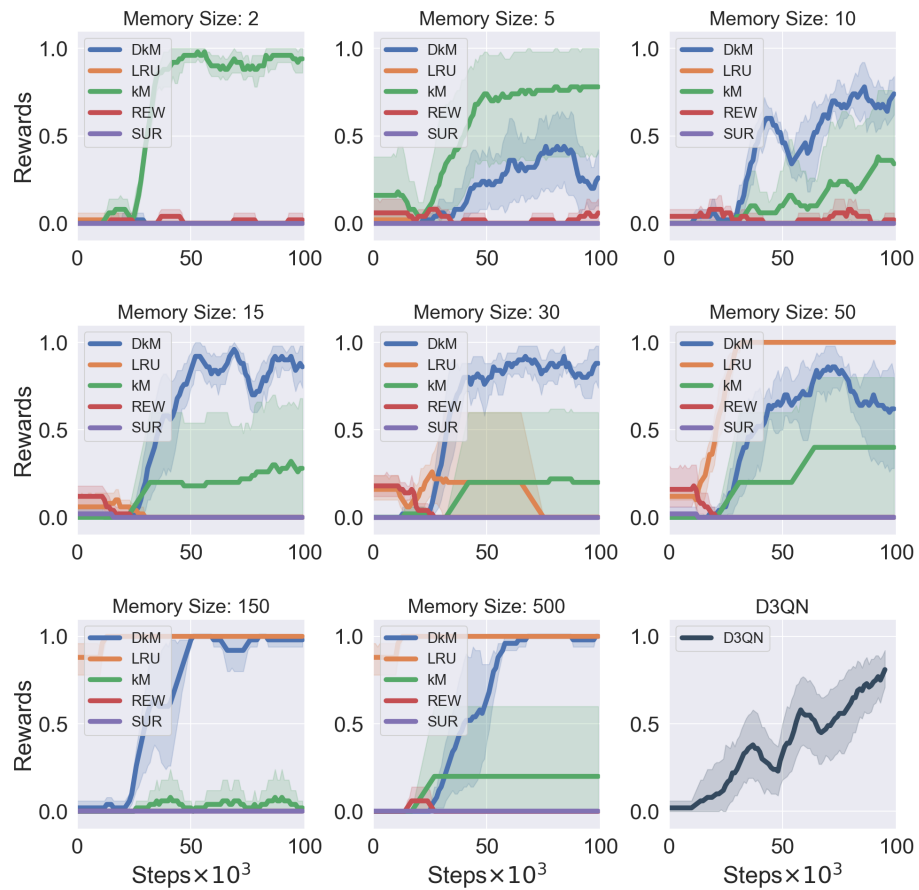


Figure 10: Learning curves of OpenRoom for different memory sizes, using MFEC and D3QN

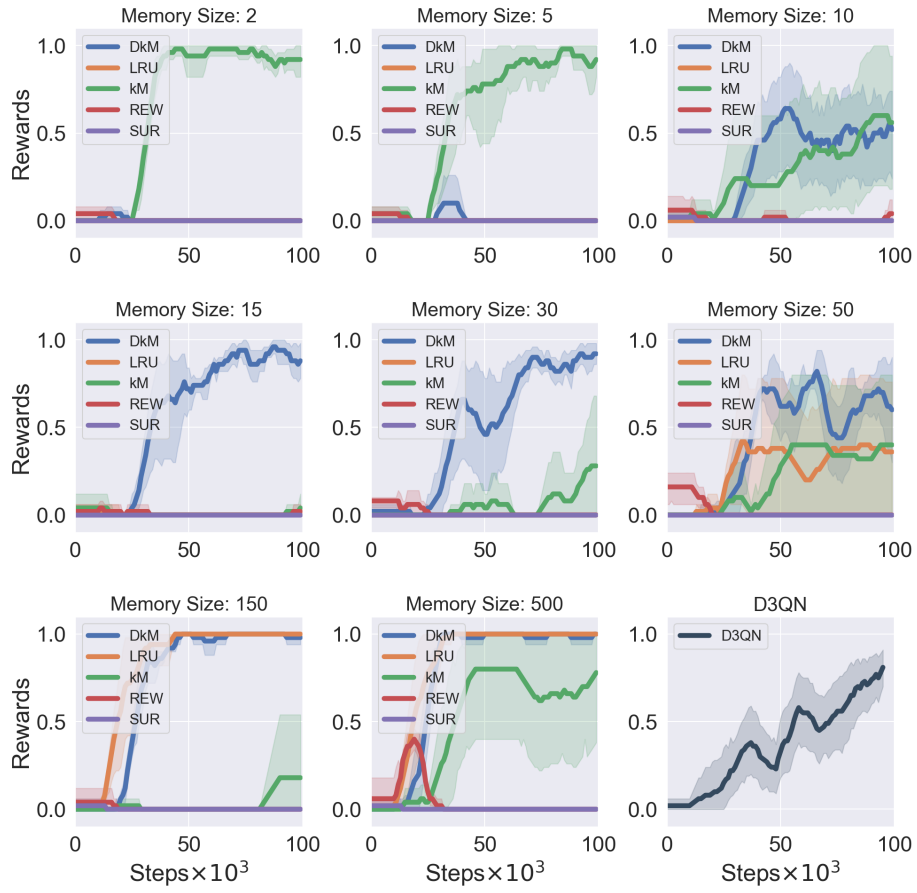


Figure 11: Learning curves of OpenRoom for different memory sizes, using NEC and D3QN

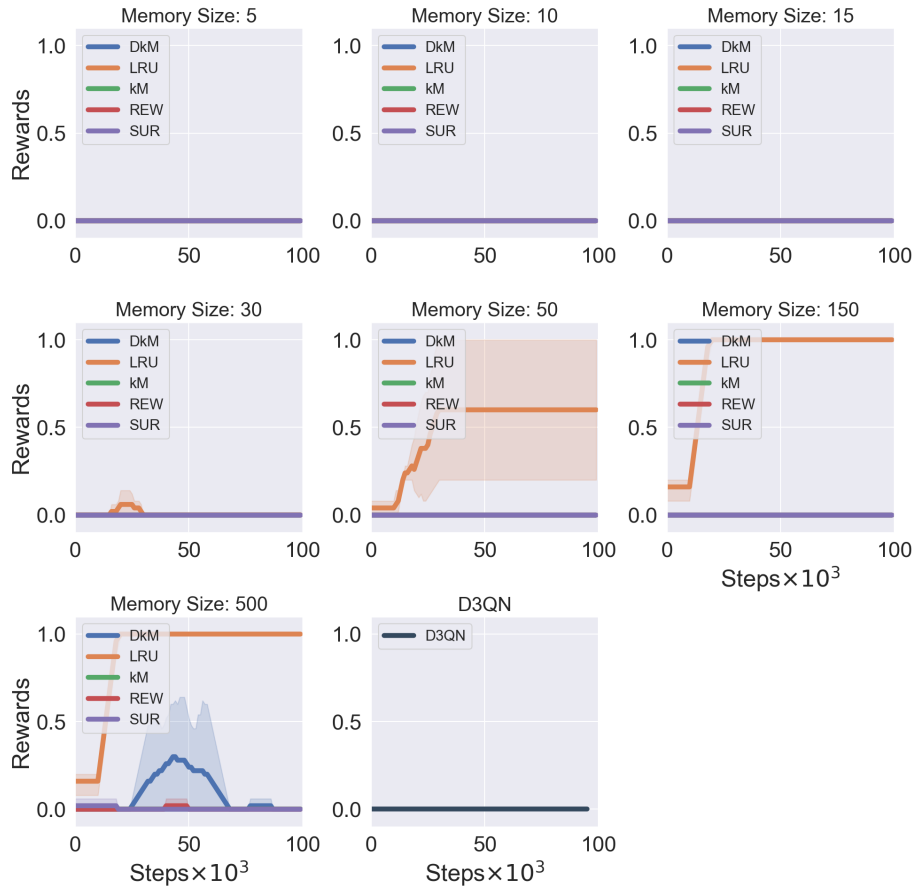


Figure 12: Learning curves of FourRoom for different memory sizes, using MFEC and D3QN

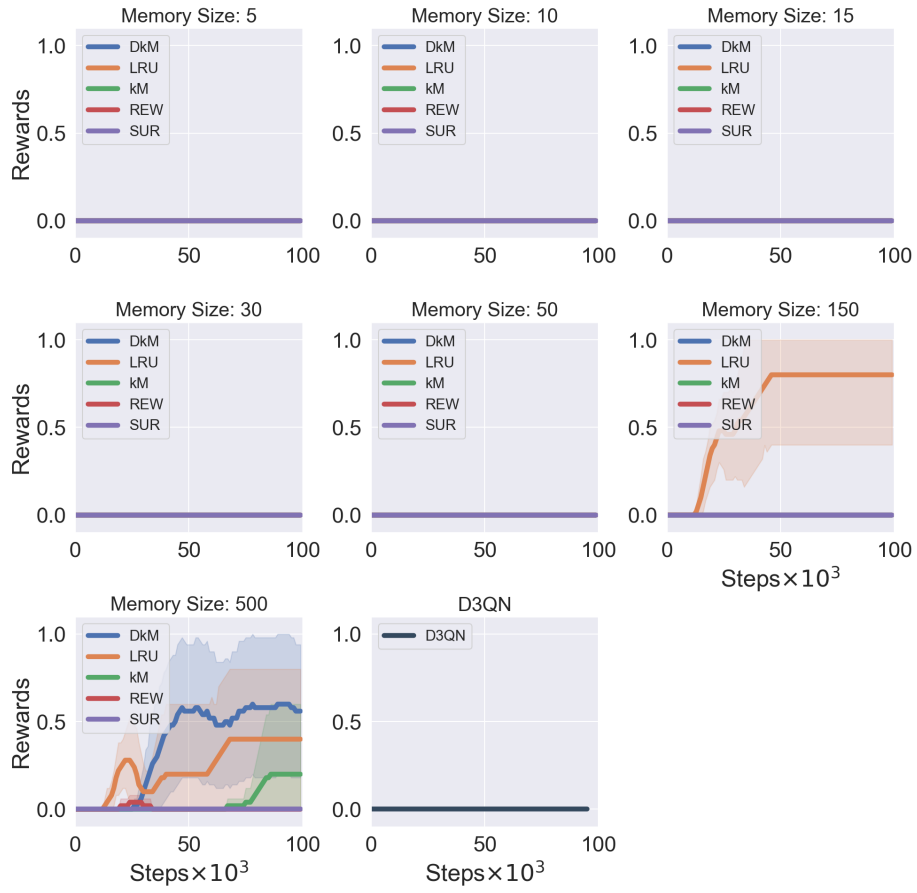


Figure 13: Learning curves of FourRoom for different memory sizes, using NEC and D3QN



## 9 Atari games

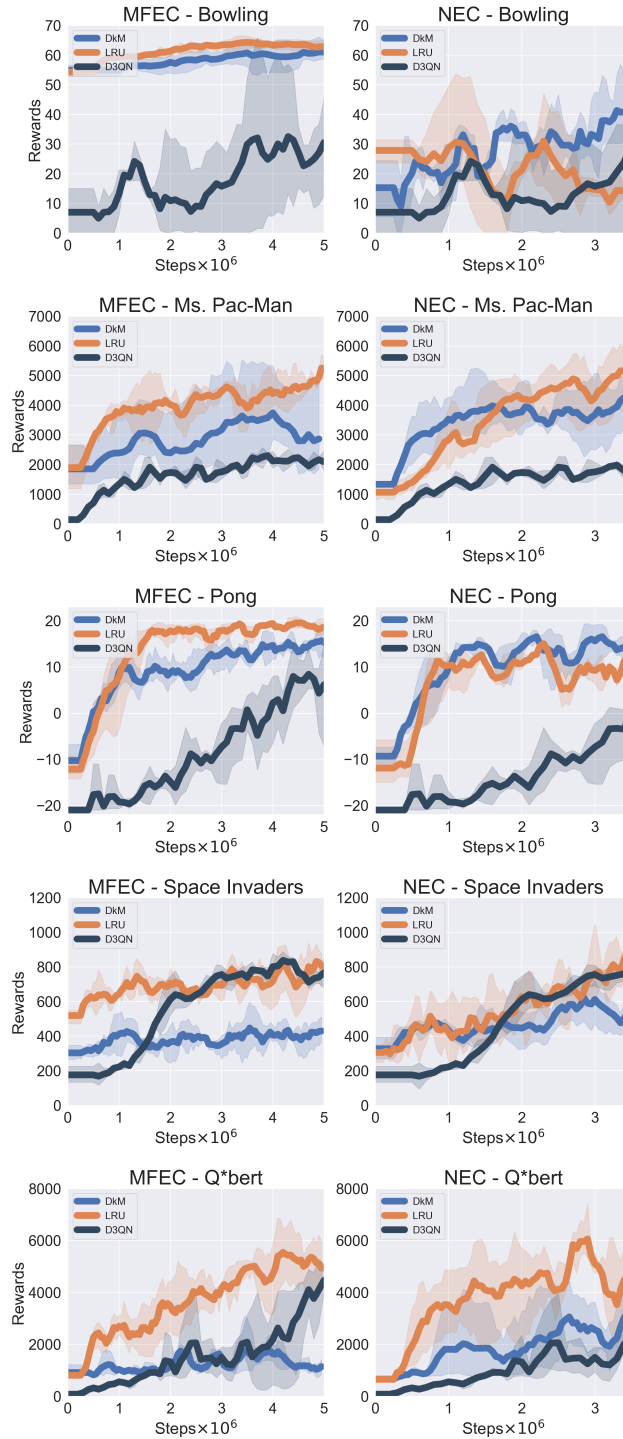


Figure 14: Learning curves of 5 Atari games using MFEC, NEC and D3QN. EC methods with a memory size of  $10^5$  per action.

## 10 Hyperparameters

Table 4: Hyperparameters used with MFEC across all the environments.

Parameters name	Classic Control	Room Domains	Atari Games
Number of neighbours $k$	11	11	11
$\epsilon$ initial	1	1	1
$\epsilon$ final	$5 \times 10^{-3}$	$5 \times 10^{-3}$	$5 \times 10^{-3}$
$\epsilon$ anneal start (steps)	$5 \times 10^3$	$5 \times 10^3$	$5 \times 10^3$
$\epsilon$ anneal end (steps)	$25 \times 10^3$	$25 \times 10^3$	$25 \times 10^3$
Discount factor $\lambda$	0.99	0.99	0.99
Reward clip	None	None	None
Kernel delta $\delta$	$10^{-3}$	$10^{-3}$	$10^{-3}$
Observation projection	None	None	Gaussian
Projection key size	None	None	128

Table 5: Hyperparameters used with NEC across all the environments.

Parameters name	Classic Control	Room Domains	Atari Games
Number of neighbours $k$	11	11	50
Experience replay size	$10^5$	$10^5$	$10^5$
Memory learning rate $\alpha$	0.1	0.1	0.1
RMSprop learning rate	$7.92 \times 10^{-6}$	$7.92 \times 10^{-6}$	$7.92 \times 10^{-6}$
RMSprop momentum	0.95	0.95	0.95
RMSprop $\epsilon$	$10^{-2}$	$10^{-2}$	$10^{-2}$
$\epsilon$ initial	1	1	1
$\epsilon$ final	$5 \times 10^{-3}$	$5 \times 10^{-3}$	$10^{-3}$
$\epsilon$ anneal start (steps)	$5 \times 10^3$	$5 \times 10^3$	$5 \times 10^3$
$\epsilon$ anneal end (steps)	$25 \times 10^3$	$25 \times 10^3$	$25 \times 10^3$
Discount factor $\lambda$	0.99	0.99	0.99
Reward clip	None	None	None
Kernel delta $\delta$	$10^{-3}$	$10^{-3}$	$10^{-3}$
Batch size	32	32	32
$n$ -step return	100	100	100
Key size	64	64	128
Training start (steps)	$10^3$	$10^3$	$5 \times 10^4$

Table 6: Hyperparameters used with D3QN across all the environments.

Parameters name	Classic Control	Room Domains	Atari Games
Experience replay size	$10^5$	$10^5$	$10^6$
RMSprop learning rate	$25 \times 10^{-5}$	$25 \times 10^{-5}$	$25 \times 10^{-5}$
RMSprop momentum	0.95	0.95	0.95
RMSprop $\epsilon$	$10^{-2}$	$10^{-2}$	$10^{-2}$
$\epsilon$ initial	1	1	1
$\epsilon$ final	$5 \times 10^{-3}$	$5 \times 10^{-3}$	$10^{-2}$
$\epsilon$ anneal start (steps)	1	1	1
$\epsilon$ anneal end (steps)	$5 \times 10^4$	$5 \times 10^4$	$10^6$
Discount factor $\lambda$	0.99	0.99	0.99
Reward clip	Yes	Yes	Yes
Batch size	32	32	32
Training start (steps)	$5 \times 10^3$	$5 \times 10^3$	$12.5 \times 10^3$
Target network update (steps)	$7.5 \times 10^3$	$10^3$	$10^3$