Successor Feature Neural Episodic Control

David Emukpere, Xavier Alameda-Pineda, Chris Reinke
RobotLearn
INRIA Grenoble, LJK, UGA

Abstract

A longstanding goal in reinforcement learning is to build intelligent agents that show fast learning and a flexible transfer of skills akin to humans and animals. This paper investigates the integration of two frameworks for tackling those goals: episodic control and successor features. Episodic control is a cognitively inspired approach relying on episodic memory, an instance-based memory model of an agent's experiences. Meanwhile, successor features and generalized policy improvement (SF&GPI) is a meta and transfer learning framework allowing to learn policies for tasks that can be efficiently reused for later tasks which have a different reward function. Individually, these two techniques have shown impressive results in vastly improving sample efficiency and the elegant reuse of previously learned policies. Thus, we outline a combination of both approaches in a single reinforcement learning framework and empirically illustrate its benefits.

1 Introduction

The idea of building intelligent agents and systems that learn purely by interaction with their environment, known as reinforcement learning [Sutton and Barto, 2018], is an appealing approach to artificial intelligence with solid connections to neuroscience and psychology [Niv, 2009, Botvinick et al., 2020]. Reinforcement learning has generated significant interest both in the research community and in public awareness, especially in combination with deep learning [LeCun et al., 2015], a paradigm known as deep reinforcement learning [Arulkumaran et al., 2017]. It has given rise to impressive achievements in various contexts, including building champion game players [Silver et al., 2016, Vinyals et al., 2019, OpenAI et al., 2019, Schrittwieser et al., 2020], and solving long-standing problems in biology [Jumper et al., 2021] to list a few. However, human intelligence has defining characteristics lacking in state-of-the-art deep reinforcement learning systems.

One important restriction of these systems is that they require significantly large amounts of data to learn [Lake et al., 2017, Tsividis et al., 2017], as they need a lot of (repeated) exposure to learn rules/concepts contained in data samples which manifests as *slow* learning. In contrast, humans can learn quickly and efficiently, making use of little data. As pointed out by Botvinick et al. [2019], a source of slowness in deep reinforcement learning can be attributed to the requirement for incremental parameter adjustment in gradient-based optimization of deep neural networks. A technique that has been proposed to tackle the data efficiency problem is Neural Episodic Control (NEC) [Pritzel et al., 2017]. Instead of gradually learning a representation of the solution, i.e. the expected future reward of an action in a certain situation, it stores observed experiences, i.e. the resulting rewards of an action, directly in a memory. Encountering a similar situation again, the experiences in the memory are recalled to decide which action yields the best outcome. As a result, episodic control learns significantly faster than gradient-based techniques.

Another typical trait of human learning is the ability to seamlessly transfer knowledge across similar tasks leading to a faster learning process in new tasks. This problem is typically tackled under the

Source code: `https://gitlab.inria.fr/robotlearn/sfnec`

5th Workshop on Meta-Learning at NeurIPS 2021, Sydney, Australia.

frameworks of meta [Hospedales et al., 2021] and transfer learning [Taylor and Stone, 2009, Zhu et al., 2020b]. One such ability of humans is to reevaluate previously learned behaviors given a new task setting [Momennejad et al., 2017]. For example, to reevalute all the possible ways you learned to drive home from work while maximize a new weighted combination of using minimum time and having scenic views. The framework of Successor Features and Generalized Policy Improvement (SF&GPI) provides a mechanisms to replicate this human ability. It decomposes the representation of learned behaviors in an environment dynamics part, i.e. what will happen when I do this behavior, and a reward part, i.e. how to evaluate this outcome. Given a set of learned behaviors, i.e. their environment dynamics, and a new reward function the expected return for each behavior can be computed and the best behavior chosen.

The central idea proposed in this paper is a framework combining NEC with SF&GPI, which we call Successor Feature Neural Episodic Control (SFNEC). We hypothesize that this would provide advantages from both approaches by merging the learning speed conferred by episodic control with flexible transfer from SF&GPI. We choose these two frameworks for the following reasons. First, episodic control has both a well-founded cognitive science inspiration [Tulving et al., 1972, Lengyel and Dayan, 2007] and displays impressive sample efficiency results in reinforcement learning tasks [Blundell et al., 2016, Pritzel et al., 2017]. Likewise, for successor features, the elegance of the SF&GPI framework and the connections of successor representation [Dayan, 1993, Gershman, 2018] to neuroscience form the basis of our motivation. Additionally, a recent study [Tomov et al., 2021] suggests that humans use a strategy similar to SF&GPI for multi-task reinforcement learning.

To summarize, our main contributions are:

- Introduction of SFNEC, a novel approach integrating sample-efficient learning using episodic control with meta learning using SF&GPI
- Empirical validation of SFNEC by showing its advantage over baseline SF&GPI, and NEC

2 Background

2.1 Reinforcement Learning

Reinforcement learning [Sutton and Barto, 2018] refers to a learning process where an agent attempts to maximize cumulative rewards it can obtain while interacting with its environment. Reinforcement learning problems are formalized as *Markov Decision Processes* (MDPs) [Puterman, 1994]. A MDP is a tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, p is the state transition probability distribution function $p(s_{t+1}|s_t, a_t)$ defining the probability of ending in state $s_{t+1} \in \mathcal{S}$ after an agent takes action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ at the current time step t, and $\mathcal{R}$ is the reward function associated with a transition (s_t, a_t, s_{t+1}). The goal of a reinforcement learning agent is to learn a *policy* π, a mapping from states to actions, so as to maximize the expected sum of discounted rewards $G_t = \mathbb{E}^\pi[\sum_{j=0}^{\infty} \gamma^j r_{t+j}]$, called the *return*, where r_{t+j} are the rewards received at each time step, and $\gamma \in [0, 1)$ is the discount factor used to determine how much weight is accorded to future rewards.

Value function based methods represent a large class of reinforcement learning algorithms based on classical *dynamic programming* [Bellman and Dreyfus, 2010]. They learn a value function, here an *action value function*, that can be recursively represented according to the Bellman equation:

$$Q^\pi(s_t, a_t) = \mathbb{E}^\pi \left[\sum_{j=0}^{\infty} \gamma^j r_{t+j} \right] = \mathbb{E}\left[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \right] .$$

The policy is then defined by maximizing the Q-function: $\pi(s) = \operatorname{argmax}_a Q^\pi(s, a)$. A widely used method within this class of algorithms is Q-learning [Watkins and Dayan, 1992] trying to learn the optimal Q-function: $Q^*(s_t, a_t) = \mathbb{E}\big[r_t + \gamma \operatorname{argmax}_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1})\big]$.

Classical Q-learning is restricted to small problems because it requires a table of all state-action pairs which becomes prohibitive or even unfeasible when attempting to scale to high dimensional state spaces. Thus, more recently, powerful function approximators such as deep neural networks are used which allow methods like Q-learning to scale to high dimensional state spaces, as exemplified in Deep Q-Network [Mnih et al., 2013] and more recent variants.

2.2 Episodic Control

Episodic memory [Tulving et al., 1972] is a model from the field of psychology, which refers to an autobiographical kind of memory about one's personal experiences. Likewise, episodic control [Lengyel and Dayan, 2007] implies the utilization of episodic memory for reinforcement learning by replaying stored action sequences from previous experiences.

Neural Episodic Control: Neural Episodic Control (NEC) [Pritzel et al., 2017] is a computational model of episodic control. Central to NEC, is a memory structure called *differentiable neural dictionary* (DND) which is a table M_a of a pair of dynamically growing arrays of keys and values (K_a, V_a) for each action $a \in \mathcal{A}$. The keys here represent a learned representation of the agent state, while the values are Q-value estimates. To estimate the Q-value for a particular (s, a) pair, a lookup is performed with the corresponding DND for action a using a query key h, which is a lower-dimensional representation of s. The governing equation is: $Q(s, a) = \sum_i w_i v_i$, where v_i corresponds to Q-values stored in V_a and w_i are weights corresponding to the result of a normalized kernel k between the query key h and keys h_i in K_a as follows $w_i = k(h, h_i)/\sum_j k(h, h_j)$.

Two techniques were employed to enable the scalability of this model. First, the number of elements involved in lookups was limited to the top 50 nearest neighbours, efficiently found using a k-d tree. Second, the sizes of the DNDs were kept limited by removing the least recently used items. Furthermore, the values stored in memory were N-step Q-value estimates:

$$Q^{(N)}(s_t, a_t) = \mathbb{E}\left[\sum_{j=0}^{N-1} \gamma^j r_{t+j} + \gamma^N \max_{a'} Q(s_{t+N}, a')\right] .$$

Updates to keys already found in the DND memory store while learning were done using Q-learning as $Q_i \leftarrow Q_i + \alpha(Q^{(N)}(s, a) - Q_i)$.

2.3 Meta and Transfer Learning

Meta [Li, 2018] and transfer learning [Taylor and Stone, 2009, Lazaric, 2012, Zhu et al., 2020b] refer to methods that allow knowledge learned from one or several tasks to be reused when faced with new tasks. In reinforcement learning, these tasks are defined by a set of MDPs $\mathcal{M}$. To have a transfer between MDPs, some shared structure must exist between them. For example, consider an agent facing a set of navigation tasks in a sequence. Assuming the dynamics remain the same across these tasks, we can specify the desired behaviour by rewarding the agent to reach specific locations. Thus, specifying different tasks for an agent in this environment corresponds to different reward functions based on its location. In this paper, we are interested in this setting for transfer where successive tasks solved by an agent only differ in their reward functions. A prominent method for this setting is the SF&GPI framework [Barreto et al., 2017].

Successor Features: Successor features (SF) are based on the idea of learning a value function representation that decouples rewards from environment dynamics [Barreto et al., 2017]. This is accomplished under the assumption that rewards are a linear combination of features $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t, a_t, s_{t+1}) \in \mathbb{R}^n$ that depend on a state transition and a weight vector $\mathbf{w} \in \mathbb{R}^n$: $r_t = \boldsymbol{\phi}_t^\top \mathbf{w}$. Features describe the essential aspects of states for evaluating them with a reward function in a low-dimensional representation. Each MDP in $\mathcal{M}$ has a different weight vector that defines its reward function, and the features are shared between all MDPs. As a result, the Q-function rewrites as:

$$Q^\pi(s_t, a_t) \;=\; \mathbb{E}^\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} r_i\right] \;=\; \mathbb{E}^\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i^\top \mathbf{w}\right] \;=\; \boldsymbol{\psi}^\pi(s_t, a_t)^\top \mathbf{w} \;,$$

where $\boldsymbol{\psi}^\pi(s, a)$ are known as the successor features (SF) of (s_t, a_t) under policy π. Also, SF satisfy a Bellman equation: $\boldsymbol{\psi}(s_t, a_t) = \boldsymbol{\phi}_t + \gamma \mathbb{E}^\pi[\boldsymbol{\psi}^\pi(s_{t+1}, \pi(s_{t+1}))]$, and thus can be learned using conventional reinforcement learning methods.

Generalized Policy Improvement: Generalized policy improvement (GPI) is an operation to combine multiple policies, i.e. policies that were learned in previous tasks, to define a policy $\pi(s)$ for a new task: $\pi(s) \in \operatorname{argmax}_a \max_i Q^{\pi_i}(s, a)$. Using the SF decomposition $Q^{\pi_i}(s_t, a_t) =$

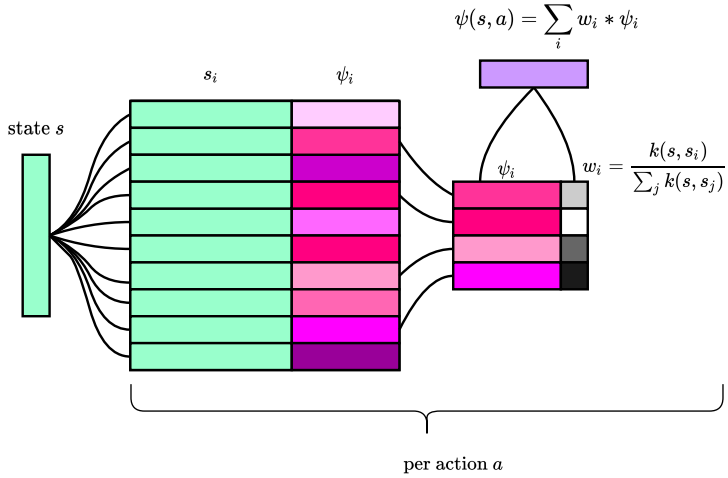

Figure 1: SFNEC architecture to store in an episodic manner $\boldsymbol{\psi}$-values.

$\boldsymbol{\psi}^{\pi_i}(s_t, a_t)^\top \mathbf{w}$, the GPI operator becomes $\pi(s) \in \operatorname{argmax}_a \max_i \boldsymbol{\psi}^{\pi_i}(s_t, a_t)^\top \mathbf{w}$. As a result, the operator allows reevaluating old policies in a new task with reward weight vector $\mathbf{w}$ to chose the best action according to them. Based on this, an algorithm for transfer using SF&GPI was proposed in [Barreto et al., 2017] called SFQL.

3 Method: The SFNEC Model

Our proposed model, SFNEC (Fig. 1), extends NEC to learn successor features $\boldsymbol{\psi}(s, a) \in \mathbb{R}^n$ in place of scalar action-values. Like NEC, which learns N-step Q-values, SFNEC learns N-step $\boldsymbol{\psi}$-values:

$$\boldsymbol{\psi}^{(N)}(s_t, a_t) = \mathbb{E}\left[\sum_{j=0}^{N-1} \gamma^j \boldsymbol{\phi}_{t+j} + \gamma^N \max_{a'} \boldsymbol{\psi}(s_{t+N}, a')\right] . \tag{1}$$

To perform a lookup using the SFNEC model, we use Equation 2:

$$\boldsymbol{\psi}(s_t, a) = \sum_i \frac{k(s_t, s_i)}{\sum_j k(s_t, s_j)} * \boldsymbol{\psi}_i \ , \tag{2}$$

where $\boldsymbol{\psi}_i$ corresponds to a previously stored $\boldsymbol{\psi}_i$-value for a state s_i in memory, and k is the kernel used to compute a similarity score between the query state s_t and states in memory s_i. We note that we directly used the state vector s_t as keys in memory for our experiments. In general, the NEC architecture allows training an embedding network to learn a lower-dimensional state embedding to be used as keys in memory. Similar to NEC, we limit the elements in memory used during lookups to the top nearest neighbours, e.g., 50. Likewise, we also used the inverse distance kernel used in [Pritzel et al., 2017]: $k(s, s_i) = \frac{1}{\|s - s_i\|_2^2 + \delta}$.

During training, $\boldsymbol{\psi}$-values are updated after observing N transitions. When the $\boldsymbol{\psi}$-value for a state action pair (s, a) does not exist previously in memory, N-step estimates computed using Equation 1 are inserted in the corresponding DND for action a. On the other hand, $\boldsymbol{\psi}$-values already in memory are updated using: $\boldsymbol{\psi}_i \leftarrow \boldsymbol{\psi}_i + \alpha(\boldsymbol{\psi}^{(N)}(s, a) - \boldsymbol{\psi}_i)$.

We defer for details of the algorithm and the training procedure to Appendix B.

4 Experiments

We evaluated SFNEC in the two-dimensional object collection environment (Fig. 2) proposed by Barreto et al. [2017]. The environment consists of four rooms with a start location in the bottom left denoted 'S', and a goal location in the top right denoted 'G'. Multiple objects exist within the rooms belonging to three classes shown as a circle, square, and triangle. The goal is to navigate from the start position to the goal position while picking up objects to maximize the cumulative rewards. Objects once picked up disappear and reappear at the beginning of a new episode. To utilize the SF&GPI framework as defined in [Barreto et al., 2017], it is necessary to have a linear decomposition of rewards into features and weights as $r_t = \boldsymbol{\phi}^\top \mathbf{w}$. The features describe the picked up object classes and if the goal state is reached using a binary representation: $\boldsymbol{\phi} \in \{0, 1\}^4$. The first three feature components represent if an object belonging to one of the three classes has been picked, while the last component represents if the goal state is reached. Thus, rewards associated with each transition can be expressed as a dot product between these features and weight vectors $\mathbf{w} \in \mathbb{R}^4$ that contain the reward associated with picking up each object class and reaching the goal. Different tasks in the environment are then defined by setting a weight vector $\mathbf{w}$.

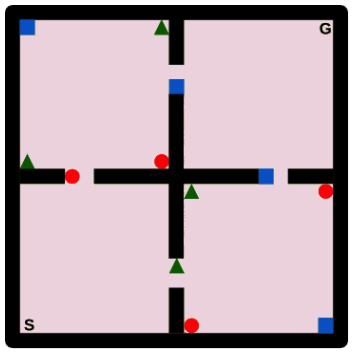

Figure 2: 2-D object collection environment proposed in [Barreto et al., 2017].

To demonstrate good performance, an agent faces a series of tasks, each being a different instantiation of $\mathbf{w}$ with the aim of maximizing the sum of rewards accumulated by the agent. In general, we follow the same setup for this environment as in [Barreto et al., 2017] with further details given in Appendix C. We compare SFNEC with SFQL, NEC, and a version of SFNEC without GPI.

Results: We compared the average return per task over 10 runs of each algorithm (Fig. 3). SFNEC with GPI performs best, outperforming NEC and SFQL agents. We expect this as SFNEC combines learning speed from episodic control on each task with the strong transfer conferred by SF&GPI.

Furthermore, we note that NEC and SFNEC without GPI show a significantly improved learning speed over the SFQL baseline during the first 10 tasks due to their episodic memory even without having a transfer mechanism.

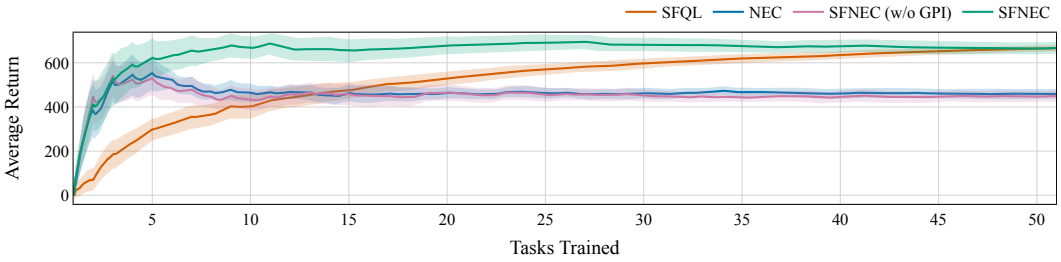

Figure 3: SFNEC has a higher learning speed than other methods. Depicted is the mean of the average return per task over 10 runs with the standard error shown as the shaded regions.

5 Discussion

Complementary benefits of episodic control and SF&GPI: Like SFQL, SFNEC learns $\boldsymbol{\psi}$-values and utilizes GPI. However, SFNEC uses episodic memory, which means the main advantage we expect would be improved learning speed compared to SFQL. This is confirmed in the results (Fig. 3) as SFNEC rises in performance much faster than SFQL, even though they reach similar performance levels in the long run due to both algorithms utilizing GPI for transfer. On the other hand, NEC and

SFNEC without GPI also employ episodic control but differ from SFNEC by not utilizing GPI but attempting to learn each task from scratch rapidly. More specifically, NEC directly learns Q-values, while SFNEC without GPI learns $\boldsymbol{\psi}$-values. We observe that SFNEC without GPI and NEC perform similarly and can demonstrate reasonably strong performance in learning all tasks individually due to their usage of episodic control. However, they are not able to match the long-run performance of SFNEC. Putting both comparisons together, we deduce that the combination of episodic control with transfer using the SF&GPI framework in SFNEC brings together rapid learning and transfer. However, as can be seen towards the right end of Fig. 3, SFQL would most probably overtake SFNEC in subsequent phases. This is similar to the observation reported in Pritzel et al. [2017] where parametric methods like DQN outperform NEC in the long run. We leave an investigation of integrating methods proposed to tackle this into SFNEC for subsequent work.

Learning a lower-dimensional state embedding: We conducted further preliminary experiments with SFNEC that attempted to learn a lower-dimensional state embedding for the keys of the DND, like the original setup in NEC. However, this did not yield good results yet. We hypothesize that this might be due to higher approximation errors introduced when learning an embedding. With the method we used in this paper, where we did not learn an embedding, we have the advantage that the keys are stable as they come directly from environment observations. Thus, learning is easier for the agent as it is not burdened with a representation learning stage. Logically, we can expect that if the agent cannot learn a good embedding to produce keys in the DND for a particular task, then reusing this on a new task can lead to erroneous predictions of the $\boldsymbol{\psi}$-values. Essentially, this would mean the approximation error could be high for the policies involved in the GPI procedure because of inaccurately learned embeddings, which could result in poor performance.

6 Related work

Episodic Control: Several improvements and extensions to NEC have been proposed. In [Lin et al., 2018] Episodic Memory Deep Q Network was proposed, an architecture that augments DQN with an episodic memory-based estimate. They found that combining this with a TD estimate improved sample efficiency and long-term performance over DQN and NEC. Sarrico et al. [2019] investigated adding principled exploration to NEC by combining episodic control with maximum entropy mellowmax policy. [Agostinelli et al., 2019] on the other hand, proposed using dynamic online k-means to improve the memory efficiency of NEC. Likewise, [Zhu et al., 2020a] proposed to further optimize the usage of the contents of the episodic memory store by considering the relationship between contents of episodic memory. Finally, [Hu et al., 2021] recently introduced *Generalizable Episodic Memory* which extends the applicability of episodic control to continuous action domains. These extensions are parallel developments to SFNEC and could be integrated with it.

Successor Features: A few extensions to successor features exist. A relaxation of the condition that reward functions be expressed as a linear decomposition, as well as a demonstration of how to combine deep neural networks with SF&GPI was introduced in [Barreto et al., 2019]. Another direction aims to learn appropriate features from data such as by optimally reconstruct rewards [Barreto et al., 2017], using the concept of mutual information [Hansen et al., 2019], or the grouping of temporal similar states [Madjiheurem and Toni, 2019]. A further direction is the generalization of the ψ-function over policies [Borsa et al., 2018] analogous to universal value function approximation [Schaul et al., 2015]. Similar approaches use successor maps [Madarasz, 2019], goal-conditioned policies [Ma et al., 2020], or successor feature sets [Brantley et al., 2021]. However, none of these extensions studied the usage of SF in combination with episodic memory.

7 Conclusion

We introduced SFNEC, and showed its viability as a framework that combines rapid learning and transfer. However, a few problems would need to be addressed to obtain a robust practical implementation. An example would be investigating methods for reducing memory requirements as this is a real-world constraint. Similarly, deciding when to learn tasks or automatically detect task switches is an area to tackle. Some suggestions for tackling such problems have been pointed out in [Barreto et al., 2017], and we believe it would be fruitful work to investigate applying SFNEC on real-world tasks with an evaluation of different techniques to handle these various challenges.

References

A. Agostinelli, K. Arulkumaran, M. Sarrico, P. Richemond, and A. A. Bharath. Memory-efficient episodic control reinforcement learning with dynamic online k-means. *arXiv:1911.09560 [cs, stat]*, Nov. 2019. arXiv: 1911.09560.

K. Arulkumaran, M. Deisenroth, M. Brundage, and A. Bharath. A brief survey of deep reinforcement learning. *ArXiv*, abs/1708.05866, 2017.

A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, D. Silver, and H. V. Hasselt. Successor features for transfer in reinforcement learning. In *NIPS*, 2017.

A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. J. Mankowitz, A. Zídek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *CoRR*, abs/1901.10964, 2019.

R. Bellman and S. Dreyfus. *Dynamic programming*. Princeton Landmarks in mathematics. Princeton University Press, Princeton, NJ, 1. princeton landmarks in mathematics ed., with a new introduction edition, 2010. ISBN 9780691146683.

J. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18: 509–517, 1975.

C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. W. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *ArXiv*, abs/1606.04460, 2016.

D. Borsa, A. Barreto, J. Quan, D. Mankowitz, R. Munos, H. van Hasselt, D. Silver, and T. Schaul. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*, 2018.

M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis. Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*, 23(5):408–422, May 2019. ISSN 13646613. doi: 10.1016/j.tics.2019.02.006.

M. Botvinick, J. X. Wang, W. Dabney, K. J. Miller, and Z. Kurth-Nelson. Deep reinforcement learning and its neuroscientific implications. *Neuron*, 107(4):603–616, Aug. 2020. ISSN 08966273. doi: 10.1016/j.neuron.2020.06.014.

K. Brantley, S. Mehri, and G. J. Gordon. Successor feature sets: Generalizing successor representations across policies. *arXiv preprint arXiv:2103.02650*, 2021.

P. Dayan. Improving generalization for temporal difference learning: the successor representation. *Neural Computation*, 5(4):613–624, July 1993. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1993.5.4.613.

S. J. Gershman. The successor representation: its computational logic and neural substrates. *The Journal of Neuroscience*, 38(33):7193–7200, Aug. 2018. ISSN 0270-6474, 1529-2401. doi: 10.1523/JNEUROSCI.0151-18.2018.

S. Hansen, W. Dabney, A. Barreto, T. Van de Wiele, D. Warde-Farley, and V. Mnih. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

T. M. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2021.

H. Hu, J. Ye, Z. Ren, G. Zhu, and C. Zhang. Generalizable episodic memory for deep reinforcement learning. In *ICML*, 2021.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals,

A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, July 2021. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-021-03819-2.

B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017. ISSN 0140-525X, 1469-1825. doi: 10.1017/S0140525X16001837.

A. Lazaric. Transfer in reinforcement learning: A framework and a survey. In *Reinforcement Learning*, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539.

M. Lengyel and P. Dayan. Hippocampal contributions to control: The third way. In *NIPS*, 2007.

Y. Li. Deep reinforcement learning, 2018.

Z. Lin, T. Zhao, G. Yang, and L. Zhang. Episodic memory deep q-networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2433–2439, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization. ISBN 9780999241127. doi: 10.24963/ijcai.2018/337.

C. Ma, D. R. Ashley, J. Wen, and Y. Bengio. Universal successor features for transfer reinforcement learning. *arXiv preprint arXiv:2001.04025*, 2020.

T. J. Madarasz. Better transfer learning with inferred successor maps. *arXiv preprint arXiv:1906.07663*, 2019.

S. Madjiheurem and L. Toni. State2vec: Off-policy successor features approximators. *arXiv preprint arXiv:1910.10277*, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

I. Momennejad, E. M. Russek, J. H. Cheong, M. M. Botvinick, N. D. Daw, and S. J. Gershman. The successor representation in human reinforcement learning. *Nature human behaviour*, 1(9): 680–692, 2017.

Y. Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, June 2009. ISSN 00222496. doi: 10.1016/j.jmp.2008.12.005.

OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680 [cs, stat]*, Dec. 2019. arXiv: 1912.06680.

A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell. Neural episodic control. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2827–2836, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley Series in Probability and Statistics. Wiley, 1 edition, Apr. 1994. ISBN 9780471619772 9780470316887. doi: 10.1002/9780470316887.

M. Sarrico, K. Arulkumaran, A. Agostinelli, P. Richemond, and A. A. Bharath. Sample-efficient reinforcement learning with maximum entropy mellowmax episodic control. *arXiv:1911.09615 [cs, stat]*, Nov. 2019. arXiv: 1911.09615.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320, 2015.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec. 2020. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-020-03051-4.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature16961.

R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 9780262039246.

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, 2009.

M. S. Tomov, E. Schulz, and S. J. Gershman. Multi-task reinforcement learning in humans. *Nature Human Behaviour*, 5(6):764–773, June 2021. ISSN 2397-3374. doi: 10.1038/s41562-020-01035-y.

P. Tsividis, T. Pouncy, J. L. Xu, J. Tenenbaum, and S. Gershman. Human learning in atari. In *AAAI Spring Symposia*, 2017.

E. Tulving, W. Donaldson, G. H. Bower, and United States, editors. *Organization of memory*. Academic Press, New York, 1972. ISBN 9780127036502.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov. 2019. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-019-1724-z.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00992698.

G. Zhu, Z. Lin, G. Yang, and C. Zhang. Episodic reinforcement learning with associative memory. In *ICLR*, 2020a.

Z. Zhu, K. Lin, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020b.

A Supplemental Experiments

We now discuss a few supplemental experiments run to understand our model.

What is the effect of learning $\mathbf{w}$? Here, we run experiments where the reward weight vector $\mathbf{w}$ is not provided to agents; rather, it is approximated while interacting with the environment for algorithms that need $\mathbf{w}$ i.e., SFQL and SFNEC. As shown in Figure 4, we noticed a reduction in the performance across all agents that rely on $\mathbf{w}$.

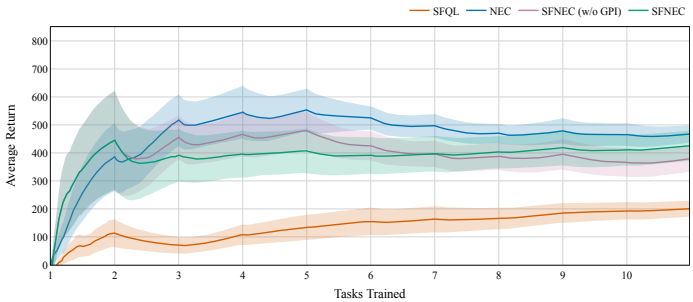

Figure 4: Average return of SFQL, NEC, and SFNEC agents on the four-room environment while learning $\mathbf{w}$. Performance drops for SFQL and SFNEC agents that depend on $\mathbf{w}$ compared to Figure 3. NEC performance remains the same and performs best in this setting. Averages are taken over 10 runs with the standard error shown as the shaded region around solid lines.

We posit that approximation errors from estimating $\mathbf{w}$ and the successor features $\boldsymbol{\psi}$ might lead to this reduced performance for agents using successor features, especially as the best performing agent in this setup is the NEC agent that does not rely on $\mathbf{w}$ nor successor features. Nonetheless, there are many application domains where the reward function given by $\mathbf{w}$ would be known, and the SFNEC model with GPI would perform best in this case.

What is the effect of varying memory capacity? We know an essential consideration in a real-life implementation of our proposed SFNEC model and algorithm is how well it will scale with many tasks. However, the scheme scales linearly, and this might be too expensive if one keeps a large DND memory per action per task. Thus, we were interested in observing the degradation of the agent's performance with the DND capacity.

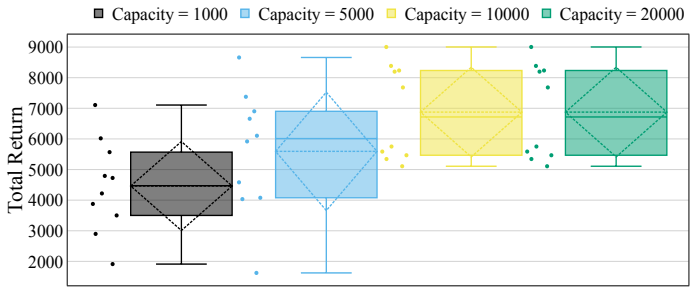

Figure 5: Total return over 10 runs of SFNEC agents on the four-room environment when varying the DND memory capacity. Beyond 10000, further increase in capacity does not lead to improved performance.

As shown in Figure 5, there is a point beyond which increasing capacity does not lead to improvement in performance. Practically, this means we might be able to use the minimum possible capacity for memory that guarantees good performance, knowing that a larger capacity would not result in performance gains. Also, the gradual degradation in performance shows that the method can be

flexibly tuned to achieve the desired tradeoff between performance and memory requirements for a particular application.

What is the effect of varying number of neighbours? A critical parameter when performing value estimation using our model is the number of neighbours used. This induces a sort of bias-variance tradeoff, and we experimented with different settings shown in Figure 6.

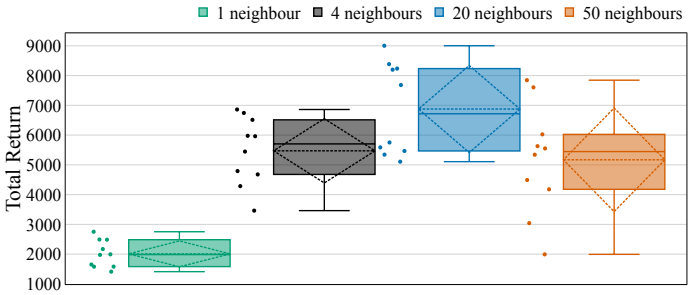

Figure 6: Total Return over 10 random runs of SFNEC agents on the four-room environment when varying number of neighbours used for value estimation. The plot depicts more that the performance follows a 'U-curve'.

Our results indicate that varying the number of neighbours should follow a 'U-curve' for a particular application, reducing performance when going towards either extreme of estimating with a single neighbour or the entire memory. This is expected because, on one end, we will have a method that obtains estimates that overfits to its nearest neighbour (i.e. high variance). On the other end, we have a method that tries to fit its estimate over the entire dataset provided by the memory module (i.e., high bias).

B Agents

In this section, we give a few more details on the agents in this paper. Source code is available at `https://gitlab.inria.fr/robotlearn/sfnec`.

SFQL agent: As our baseline agent for transfer learning, we used an implementation of SF&GPI according to the pseudocode provided in [Barreto et al., 2017] which we call SFQL agent.

NEC agent: We implement an agent following the Neural Episodic Control architecture and algorithm outlined in [Pritzel et al., 2017] as the baseline for learning speed on a single task. We relied on a publicly available implementation provided by Kai Arulkumaran on GitHub[2] as an inspiration for the implementation of our NEC agent. For our purposes, we modified our implementation to allow memory updates to occur either immediately after a horizon of N steps or batching updates at the end of episodes. This is different from the original description and the reference implementation, which suggest batching memory updates at the end of episodes. Our primary motivation for this modification is that we would like our method to be applicable even in learning scenarios that do not divide into episodes, i.e., continuing tasks.

Estimating action values with NEC involves combining the values for previously-stored keys in memory similar to a query key representing the agent's state. Thus, it is necessary to perform an efficient similarity search. In [Pritzel et al., 2017] it was suggested to enable this efficiency by performing approximate searches using a k-d tree [Bentley, 1975]. Contrary to this, to keep our initial implementation simple while laying out the general idea of our framework, we utilize brute-force searches for all agents that need to perform a similarity search in stored memory(NEC and SFNEC).

For this, we used the Facebook AI similarity search library[3] [Johnson et al., 2017]. We chose to use this library because it is open-source, simple to use, and well-optimized for brute-force and approximate nearest neighbour searches.

[2] `https://github.com/Kaixhin/EC`

[3] `https://github.com/facebookresearch/faiss`

SFNEC agent: The algorithm for SFNEC is given in Algorithm 1. We note that we allow for both cases when reward descriptions $\mathbf{w}_i$ for each task are provided or not with the boolean condition learn_w. Additionally, we note that we can use the algorithm without GPI by simply making j equal i in line 8 similar to the SFQL algorithm described in [Barreto et al., 2017].

Finally, we highlight that we also update policies used for GPI action selection in lines 22-25 as done in the SFQL algorithm. The essence of this update is to continually refine the successor features of policies that remain pertinent for GPI action selection in line 8. A crucial difference for updating these policies is that we cannot obtain the features $\boldsymbol{\phi}$ needed to compute N-step $\boldsymbol{\psi}$ estimates as the agent is acting according to a different policy at this update point. Thus, we are constrained to utilizing a single-step *off-policy* update.

Algorithm 1 Successor Feature Neural Episodic Control (SFNEC)

	α_w	learning rate for $\mathbf{w}$
	learn_w	condition to indicate if to learn $\mathbf{w}$ for each task
	$\boldsymbol{\phi}$	features of state transitions
	$\mathbf{w}_i$	optionally given for each task i
Require:	N	n-step horizon for $\psi^{(N)}$ estimates
	$\mathcal{D}_i$	replay buffer of (h, a, $\psi^{(N)}$) tuples for each task i
	M_{ai}	a DND for each action a per task i
	num_tasks	number of tasks to be learned

1: **for** $i = 1, \ldots,$ num_tasks **do**
2: **if** learn_w **then**
3: Initialize $\mathbf{w}_i$ with small random values
4: **end if**
5: **for** each episode **do**
6: **for** $t = 1, \ldots, T$ **do**
7: Get observation s_t from the environment and its embedding h_t
8: $j \leftarrow \text{argmax}_{k \in \{1,\ldots,i\}} \max_b \boldsymbol{\psi}_k(s_t, b)^\top \mathbf{w}_i$ {j is the index of policy selected according to GPI}
9: **if** $rand[0, 1) < \epsilon$ **then**
10: $a_t \leftarrow$ select an action uniformly at random
11: **else**
12: $a_t \leftarrow \text{argmax}_b \boldsymbol{\psi}_j(s_t, b)^\top \mathbf{w}_i$
13: **end if** {ϵ-greedy action selection}
14: Take action a_t and observe reward r_t, and observation s_{t+1}
15: **if** learn_w **then**
16: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_w[r - \boldsymbol{\phi}(s_t, a_t, s_{t+1})^\top \mathbf{w}_i]$ {learn $\mathbf{w}$ for task i}
17: **end if**
18: Compute $\boldsymbol{\psi}^{(N)(s_t,a_t)}$ using eqn. 1
19: Append $(h_t, \boldsymbol{\psi}^{(N)}(s_t, a_t))$ to $M_{a_t i}$
20: Append $(s_t, a_t, \boldsymbol{\psi}^{(N)}(s_t, a_t))$ to $\mathcal{D}_i$
21: Train on a random minibatch from $\mathcal{D}_i$
22: **if** $j \neq i$ **then**
23: $a' = \text{argmax}_b \boldsymbol{\psi}_j(s_{t+1}, b)^\top \mathbf{w}_j$
24: update $\boldsymbol{\psi}_j(s_t, a_t)$ using the one-step TD target: $\boldsymbol{\phi}(s_t, a_t, s_{t+1}) + \gamma \boldsymbol{\psi}_j(s_{t+1}, a')$
25: **end if**
26: **end for**
27: **end for**
28: **end for**

C Experimental details

As mentioned in the main paper, we follow the same setup as described in [Barreto et al., 2017] with essential details recapitulated below. There are twelve objects in the environment and three object classes (four objects per class). The rewards associated with each class changes after $20,000$ transitions, and they are sampled uniformly at random from $[-1, 1]$ while reaching the goal always gave a reward of $+1$. The agent's observations provided from the environment consists of two parts. The first part is the activations of the agent's (x, y) position on a 10×10 grid of radial basis functions over the entire four rooms. The second part consists of object detectors indicating the presence or absence of objects in the environment. For our experiments, we provide both the state features $\boldsymbol{\phi}(s_t, a_t, s_{t+1})$ and reward weight vector $\mathbf{w}$. The state features are boolean vectors containing elements indicating whether the agent is over an object present in the environment or over the goal position. The reward weight vector contains rewards associated with picking up an object and reaching the goal position. We note that providing both elements to the agent means the reward function is fully specified to the agent according to the decomposition: $r(s_t, a_t, s_{t+1}) = \boldsymbol{\phi}(s_t, a_t, s_{t+1})^\top \mathbf{w}$. Nonetheless, it is possible to approximate these quantities as shown in [Barreto et al., 2017]. We refer the reader to Appendix B in [Barreto et al., 2017] for further details on this environment.

Setup: We use 50 tasks for our experiments. For obtaining our hyperparameters, we carried out a grid search for the NEC and SFNEC agents. At the same time, we relied on reported values for SFQL agent parameters from [Barreto et al., 2017]. We now report values used for our search and the final values chosen for our experiments. We tested values in $\{0.05, 0.15\}$ for ϵ used for ϵ-greedy exploration. For the learning rate used in network optimization: $\{0.01, 0.05, 0.1\}$, for number of neighbours used in estimating ψ-values: $\{1, 4, 10, 20, 50\}$. Furthermore, we limited the DND capacity to $10,000$ most recent entries. For the fast learning rate used to update re-encountered keys in the DND, we tried values in $\{0.1, 0.3, 0.5\}$, and for horizon length N, we used the set: $\{8, 16, 32\}$. We show the best configurations found for each agent in Table 1.

For our NEC and SFNEC agents, we obtain the keys used as the compact state representation in memory by directly using the observation from the environment. Additionally, the training method we use for NEC and SFNEC keeps a single sample in the replay buffer of these agents. Training proceeds by using each sample sequentially as they are collected in the environment. We did this to allow a more direct comparison to SFQL, which uses a "true" stochastic gradient descent method for its optimization.

Agent	ϵ	Network learning rate	Neighbours	DND learning rate	N
SFQL	0.15	0.01	-	-	-
NEC	0.15	0.01	20	0.1	8
SFNEC	0.15	0.05	20	0.1	8

Table 1: Best parameter configurations found for agents in the four-room environment