Self-Organizing Maps as a Storage and Transfer Mechanism in Reinforcement Learning

Thommen George Karimpanal
Singapore University of Technology and Design
thommen_george@mymail.sutd.edu.sg

Roland Bouffanais
Singapore University of Technology and Design
bouffanais@sutd.edu.sg

ABSTRACT

The idea of reusing information from previously learned tasks (source tasks) for the learning of new tasks (target tasks) has the potential to significantly improve the sample efficiency reinforcement learning agents. In this work, we describe an approach to concisely store and represent learned task knowledge, and reuse it by allowing it to guide the exploration of an agent while it learns new tasks. In order to do so, we use a measure of similarity that is defined directly in the space of parameterized representations of the value functions. This similarity measure is also used as a basis for a variant of the *growing self-organizing map* algorithm, which is simultaneously used to enable the storage of previously acquired task knowledge in an adaptive and scalable manner. We empirically validate our approach in a simulated navigation environment and discuss possible extensions to this approach along with potential applications where it could be particularly useful.

KEYWORDS

Self-organizing maps; Q-learning; Transfer Learning; Multi-task Reinforcement Learning; Continual Learning

1 INTRODUCTION

The use of off-policy algorithms [5] in reinforcement learning (RL) [15] has enabled the learning of multiple tasks in parallel. This is particularly useful for agents operating in the real-world, where a number of tasks are likely to be encountered, and may be required to be learned [6, 16, 21]. Ideally, as an agent learns more and more tasks through its interactions with the environment, it should be able to efficiently store and extract meaningful information, which could be useful for accelerating its learning on new, possibly related tasks. This area of research, which aims at addressing the issue of effectively reusing previously accumulated knowledge is referred to as transfer learning [17].

Formally, transfer learning is an approach to improve learning performance on a new 'target' task M_T, using accumulated knowledge from a set of 'source' tasks, $M_S = \{M_{s_1} ... M_{s_i} ... M_{s_n}\}$. Here, each task M is a *Markov Decision Process (MDP)* [11], such that $M = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition function, and $\mathcal{R}$ is the reward function. In this work, we address the relatively simple case where tasks vary only in the reward function $\mathcal{R}$, while $\mathcal{S}, \mathcal{A}$ and $\mathcal{T}$ remain fixed across the tasks. For knowledge transfer to be effective, source tasks need to be selected appropriately. Reusing knowledge from an inappropriately selected source task could lead to negative transfer [8, 17], which is detrimental to the learning of the target task. In order to avoid such problems and ensure beneficial knowledge transfer, a number of MDP similarity metrics [3, 4] have been proposed. However, it has been shown that the utility of a particular MDP similarity metric depends on the type of transfer mechanism used [3]. In addition, these transfer mechanisms are generally not designed to handle situations involving a large number of source tasks. This could be limiting for both embodied as well as virtual agents operating in the real-world. For such an agent, the value functions pertaining to hundreds or thousands of tasks may be learned over a period of time. Some of these tasks may be very similar to each other, which could result in considerable redundancy in the stored value function information. From a continual learning perspective, a suitable mechanism may be needed to enable the storage of such information in a scalable manner. In the approach described here, the knowledge of a task is assumed to be contained in the value function (Q-function) associated with it. We assume that these value functions are represented using parameter weights, which are learned from the agent's interactions with its environment. We define a cosine similarity metric within this value function weight (parameter) space, and use this as a basis for maintaining a scalable knowledge base, while simultaneously using it to perform knowledge transfer across tasks.

The proposed mechanism enables the storage of value function weight vectors using a variant of the growing self organizing map (GSOM)[1]. The inputs to this GSOM algorithm consist of the value function weights of new tasks, along with any representative value function weights extracted from previously learned tasks. The resulting map would ideally correspond to value function weights representative of previously acquired task knowledge, topologically arranged in accordance with their relation to each other. As the agent interacts with its environment and learns the value function weights corresponding to new tasks, this new information is incorporated into the SOM, which evolves by growing to a suitable size in order to sufficiently represent all of the agent's gathered knowledge. Each element/node of the resulting map is a variant of the input value function weights (knowledge of previously learned tasks). These variants are treated as solutions to arbitrary source tasks, each of which is related to some degree to one of the previously learned tasks. The aim of storing knowledge in this manner is not to retain the exact value function information corresponding to all the previously learned tasks, but to maintain a compressed and scalable knowledge base that can approximate the value function weights of the previously learned tasks.

While learning a new target task, this knowledge base is used to identify the most relevant source task, based on the same similarity metric. The value function associated with this task is then greedily exploited to provide the agent with action advice to guide it towards achieving the target task. Due to random initialization,

the agent's initial estimates of the value function weights corresponding to the target task is poor. However, as it gathers more experience through its interactions with the environment, these estimates improve, which consequently improves its estimates of the similarities between the target and source tasks. As a result, the agent becomes more likely to receive relevant action advice from a closely related source task. This action advice can be adopted, for instance, on an ϵ-greedy basis, essentially substituting the agent's exploration strategy. In this way, the knowledge of source tasks is used to merely guide the agent's exploratory behavior, thereby minimizing the risk of negative transfer which could have otherwise occurred especially if value functions or representations were directly transferred between the tasks.

Apart from maintaining an adaptive knowledge base of value function weights related to previously learned tasks, the proposed approach aims to leverage this knowledge base to make informed exploration decisions, which could lead to faster learning of target tasks. This could be especially useful in real-world scenarios where factors such as learning speed and sample efficiency are critical, and where several new tasks may need to be learned continuously, as and when they are encountered. The overall structure of the proposed methodology is depicted in Figure 1.

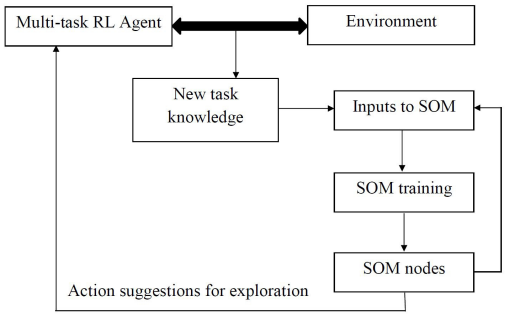

Figure 1: The overall structure of the proposed approach

2 RELATED WORK

The sample efficiency of RL algorithms is one of the most critical aspects that determines the feasibility of its deployment in real-world applications. Transfer learning is one of the mechanisms through which this can be addressed. Consequently, numerous techniques have been proposed [8, 17, 22] to efficiently reuse the knowledge of learned tasks. A number of these [2, 3, 14] rely on a measure of similarity between MDPs in order to choose an appropriate source task to transfer from. However, this can be problematic, as no such universal metric exists [3], and some of the useful ones may be computationally expensive [2]. Here, the similarity metric used is computationally inexpensive, and the degree of similarity between two tasks is based solely on the value function weights associated with them. Also, in the approach described here, once an appropriate source task is identified, its value functions are used solely to extract action advice, which is used to guide the exploration of the agent. Similar approaches to transfer learning using action advice exist [20, 22, 23], where a teacher-student framework for RL is adopted. The transfer mechanism described here is similar in principle, but is inherently tied to the SOM-based approach for maintaining the knowledge of learned tasks.

Other clustering approaches [3, 9, 19] have also been applied to achieve transfer learning in RL. In one of the earliest notable approaches to transfer learning, Thrun et al. [19] described a methodology for transfer learning by clustering learning tasks using a nearest neighbor clustering approach. Although similar approaches can be used, the SOM-based approach described here preserves the topological properties of the input space, due to which similar behaviors are placed closer to one another. This could give us a rough idea of the type of behavior to be expected, given some new, arbitrary value function weights.

Perhaps the most closely related work is the 'Actor-mimic' [10] approach, which also performs transfer using action advice. In this approach, useful behaviors of a set of expert networks are compressed into a single multi-task network, which is then used to provide action advice in an $\epsilon-$greedy manner. The authors also report the problem of dramatically varying ranges of the value function across different tasks, which is resolved by using a Boltzmann distribution function. In the present work, the use of the cosine similarity metric resolves this issue and ensures that the similarity measure between tasks is bounded.

In the context of continual learning [13], Ring et al. [12] described a modular approach to assimilate the knowledge of complex tasks using a training process that closely resembles SOM. In this approach, a complex task is decomposed into a number of simple modules, such that modules close to each other correspond to similar agent behaviors. Teng et al. [18] also proposed a SOM-based approach to integrate domain knowledge and RL, with the aim of developing agents that can continuously expand their knowledge in real time. These ideas of knowledge assimilation are also reflected in the present work. However, our approach also aims to reuse this knowledge to aid the learning of other related tasks.

3 METHODOLOGY

In this work, we present an approach that enables the reuse of knowledge from previously learned tasks to aid the learning of a new task. Our approach consists of two fundamental mechanisms: (a) the accumulation of learned value function weights into a knowledge base in a scalable manner, and (b) the use of this knowledge base to guide the agent during the learning of the target task. The basis for these mechanisms is centered around the task similarity metric we propose here. We consider two tasks to be similar based on the cosine similarity between their corresponding learned value function weight vectors. For instance, the cosine similarity c_{w_1,w_2} between two non-zero weight vectors $\vec{w}_1$ and $\vec{w}_2$ is given by:

$$c_{w_1,w_2} = \vec{w}_1.\vec{w}_2/|\vec{w}_1||\vec{w}_2|. \quad (1)$$

The key idea is that two tasks are more likely to be similar to each other if they have similar feature weightings. Using such a similarity metric has certain advantages, such as boundedness and the ability to handle weight vectors with largely different magnitudes. That is, even in the case of highly similar or dissimilar tasks, the cosine similarity remains in the range [-1,1]. During the construction of the scalable knowledge base, the mentioned similarity metric is used as a basis for training the self-organizing map. Once this map is constructed, the cosine similarity is again used as a basis for selecting

an appropriate source task weight vector to guide the exploratory behavior of the agent. We now describe these mechanisms in detail.

3.1 Knowledge Storage Using Self-Organizing Map

A self-organizing map (SOM) [7] is a type of unsupervised neural network used to produce a low-dimensional representation of its high-dimensional training samples. Typically, a SOM is represented as a two- or three-dimensional grid of nodes. Each node of the SOM is initialized to be a randomly generated weight vector of the same dimensions as the input vector. During the SOM training, an input is presented to the network, and the node that is most similar to this input is selected to be the 'winner'. The winning node is then updated towards the input vector under consideration. Other nodes in the neighborhood are also influenced in a similar manner, but as a function of their topological distances to the winner. The final layout of a SOM is such that adjacent nodes have a greater degree of similarity to each other in comparison to nodes that are far apart. In this way, the SOM extracts the latent structure of the input space.

For our purposes, the knowledge of an RL task is assumed to be contained in its parameterized representation of the value function ($Q-$function), obtained using linear function approximation [15]. A naïve approach to storing knowledge associated with multiple tasks is to explicitly store their value function parameters/weights. Apart from the scalability issue associated with such an approach, a high degree of redundancy in the learned knowledge may arise if several of these tasks are very similar or nearly identical to each other. A more generalized approach to knowledge storage would be to store the characteristic features of the weight vectors associated with the learned tasks. The ability of the SOM to extract these features in an unsupervised manner makes it an attractive choice for the knowledge storage mechanism proposed here.

In our approach, the inputs to the SOM are learned value function weights of previously learned tasks (input tasks). The hypothesis is that after training, the weight vectors associated with each node in the SOM have varying degrees of similarity to the input vectors, and hence, correspond to value function weights of tasks which may be related to the input tasks to varying degrees. Hence, each node in the SOM could be assumed to contain the value function information corresponding to a source task, and the weight vector associated with an appropriately selected SOM node could serve as source value function weights which could be used to guide the exploration of the agent while learning a target task.

In a continual learning scenario, a number of tasks with largely varying degrees of similarity (as per the similarity metric defined in Equation (1)) with each other may be encountered. A SOM containing only a few number of nodes may not be able to represent the knowledge of these tasks to a sufficient level of accuracy. Hence, the size of the SOM may need to adapt dynamically as and when new task knowledge is learned. We address this problem by allowing the number of nodes in the SOM to change, using a mechanism similar to that used in the GSOM algorithm. For a SOM containing N nodes, each node n_i is associated with an error e_{n_i} such that for a particular input vector $\vec{w}_{v_j}$, if node n_{win} (with a corresponding weight vector $\vec{w}_{s_{n_{\text{win}}}}$) is the winner, the error $e_{n_{\text{win}}}$ is updated as:

$$e_{n_{\text{win}}} \leftarrow e_{n_{\text{win}}} + 1 - c_{w_{v_j}, w_{s_{n_{\text{win}}}}}. \tag{2}$$

The term $(1 - c_{w_{v_j}, w_{s_{n_{\text{win}}}}})$ in Equation (2) is proportional to the Euclidean distance between the l^2- normalized versions of input vectors $\vec{w}_{v_j}$ and $\vec{w}_{s_{n_{\text{win}}}}$. Hence, the error update equation (Equation (2)) is equivalent to that used in [1]. Once all the input vectors are presented to the SOM, the total error, E of the network is computed as $E = \sum_{i=1}^{N} e_i$. This total error is computed for each iteration of the SOM. In subsequent iterations, if the increase in the total error exceeds a certain threshold G_T, new nodes are spawned at the boundaries of the SOM. The weight vectors of these nodes are initialized to the mean of their neighbors, and are subsequently modified by the SOM training process. Such a mechanism enables the SOM to grow in size and representation capacity, thereby allowing for a low network error to be achieved.

The nature of the described SOM algorithm is such that all the input vectors are needed during the training. However, for applications such as robotics, where the agent may have limited on-board memory, this may not be feasible. Thousands of tasks may be encountered during its lifetime, and the value function weights of all these tasks would need to be explicitly stored in order to train the SOM. Ideally, we would like the knowledge contained in the SOM to adapt in an online manner, to include relevant information from new tasks as and when they are learned. We achieve this online adaptation by making modifications to the training mechanism of the GSOM algorithm. Specifically, when a new task is learned, we update the SOM by presenting the newly learned weights, together with the weight vectors associated with the nodes of the SOM as inputs to the GSOM algorithm. The resulting SOM is then utilized for transfer. In summary, the weights of the SOM are recycled as inputs while updating the knowledge base using the GSOM algorithm. This can be observed in the overall structure of the proposed approach, shown in Figure 1. The implicit assumption here is that the weights associated with the SOM nodes sufficiently represent the knowledge of the previously learned tasks. This approach of updating the SOM knowledge base allows new knowledge to be adaptively incorporated into the SOM, while obviating the need to explicitly store the value function weights of all previously learned tasks. The overall storage mechanism is summarized in Algorithm 1.

3.2 The Transfer Mechanism

Once the knowledge of previously learned tasks has been assimilated into a SOM, it is reused to aid the learning of a target task. The weight vector associated with each SOM node is treated as the value function weight vector corresponding to an arbitrary source task. Among these source value function weight vectors, the one that is most similar (w_{s_*}) to the target value function weight vector w_T is chosen for transfer. That is, $s_* = \underset{i \in N}{argmax}(c_{w_{s_i}, w_T})$

In order to actually perform the transfer, the selected source task weights may be directly used to modify the value function weights of the target task. However, an insufficient degree of similarity of the source task weights could result in negative transfer. A safer approach is to allow the selected source value function weights to guide the exploratory actions of the agent. This guidance is provided by allowing the agent to act greedily as per the selected source value function weights with a fixed probability ϵ, while exploiting the target value function that is being learned, with a probability of

Algorithm 1 Knowledge storage using self-organizing maps

Inputs:
$\mathbf{w_v} = \{\vec{w}_{v_1} ... \vec{w}_{v_i} ... \vec{w}_{v_M}\}$: A set of value function weight vectors corresponding to M learned tasks. These are the input vectors to the GSOM algorithm.
N : Initial number of nodes in the SOM
σ_0 : Initial value of neighborhood function σ
τ_1 : Time constant to control the neighborhood function
κ_0 : Initial value of SOM learning rate κ
τ_2 : Time constant to control the learning rate
$\mathbf{w_s} = \{\vec{w}_{s_1} ... \vec{w}_{s_i} ... \vec{w}_{s_N}\}$: Initial weight vectors associated with the N nodes in the SOM
e : Error vector, initialized to be zero vector of length N
$E = 0$: Initial value of average error
G_T : Growth threshold parameter
N_{iter} : Number of SOM iterations
for $i = 1 : N_{\text{iter}}$ **do**
  Randomly pick an input vector $\vec{x}$ from $\mathbf{w_v}$
  Select winning node n_{win} based on highest cosine similarity to input vector x
  $\sigma = \sigma_0 \exp(-i/\tau_1)$
  $\kappa = \kappa_0 \exp(-i/\tau_2)$
  for $j = 1 : N$ **do**
    Compute topological distance $d_{n_{\text{win}},j}$ between nodes n_{win} and j
    $h(n_{\text{win}}, j) = \exp(-d_{n_{\text{win}},j}/2\sigma^2)$
    $\vec{w}_{s_j} = \vec{w}_{s_j} + \kappa * h(n_{\text{win}}, j) * \|\vec{x} - \vec{w}_{s_{n_{\text{win}}}}\|$
  end for
  $e(n_{\text{win}}) = e(n_{\text{win}}) + 1 - c_{x, w_{s_{n_{\text{win}}}}}$
  $E_i = \sum_{k=1}^{N} e_k$
  if $(E_i - E_{i-1})/N > G_T$ **then**
    Spawn new nodes at the boundaries of the SOM
    Expand the error vector, with the values of new nodes initialized to the mean of the previous error vector.
    Update N as per the number of new nodes added
  end if
end for

$1 - \epsilon$. Such an approach is more unlikely to lead to drastic drops in the target task performance. In general, the guidance provided by the most similar source value function weights can be expected to allow the agent to execute more useful exploratory actions, thereby leading to accelerated learning of the target task.

4 RESULTS

We use the knowledge storage and reuse mechanisms described in Section 3 to accelerate the learning of target tasks in a navigation environment. In order to evaluate the described knowledge storage and reuse mechanisms, we allow the agent to explore and learn multiple tasks in the simulated environment shown in Figure 2. The environment is continuous, and the agent is assumed to be able to sense its horizontal and vertical coordinates, which constitute its state. The states are represented in the form of a feature vector $\vec{F}_a$ containing 100 elements for each state dimension. While navigating through the environment, the agent is allowed to choose from a set of 9 different actions: moving forwards, backwards, sideways, diagonally upwards or downwards to either side, or staying in place. The velocities associated with these movements is set to be 6 units/s, and new actions are executed every 200 ms.

As the agent executes actions in its environment, it autonomously identifies tasks using an adaptive clustering approach similar to that described in Karimpanal et al. [6]. The clustering is performed on an additional feature vector $\vec{F}_e$ (environment feature vector) which contains elements describing the presence or absence of specific environment features. For instance, these features could represent the presence or absence of a source of light, sound or other signals from the environment that the agent is capable of sensing. In the simulations described here, the environment feature vector $\vec{F}_e$ contains 4 elements corresponding to 4 arbitrary environment stimuli distributed at different locations in the environment. As the agent interacts with its environment, clustering is performed on $\vec{F}_e$ in an adaptive manner, which helps identify unique configurations of $\vec{F}_e$ which may be of interest to the agent. During the agent's interactions with the environment, the mean of each discovered cluster is treated as the environment feature vector associated with the goal state of a distinct navigation task. In our simulations, the agent eventually discovers 5 such tasks, the corresponding goal locations of which are indicated by the colored regions in Figure 2. The value function corresponding to each of these tasks is learned using the $Q - \lambda$ algorithm [15]. For $Q-$learning, the reward structure is such that the agent obtains a reward (100) when it is in the goal state, a penalty (-100) for bumping into an obstacle, and a living penalty (-10) for every other non-goal state. In each episode, the agent starts from a random state and executes actions in the environment till it reaches the associated navigation target region (goal state), at which point, a positive reward is obtained, and the episode terminates. For each $Q-$learning task, the full feature vector $\vec{F}$ (where $\vec{F} = \{\vec{F}_e \cup \vec{F}_a\}$) is used, and the learning rate α is set to be 0.3, the discount factor γ is 0.9 and the trace decay parameter λ is set to be 0.9. The other hyperparameters described in Algorithm 1 are set to the following values: $N = 4$, $\sigma_0 = 50$, $\tau_1 = 250$, $\tau_2 = 0.1$, $G_T = 0.3$ and $N_{iter} = 1000$.

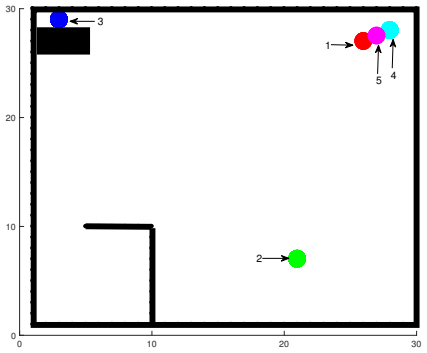

Figure 2: The simulated continuous environment with the navigation goal states of different tasks (numbered from tasks 1 to 5), indicated by the different colored circles.

Once a new navigation task t is identified, and its value function weight vector w_t is learned, we incorporate this new knowledge into the SOM knowledge base. To do this, the value function weight vector associated with this task, along with the weight vectors associated with the SOM are presented as input vectors to Algorithm 1. For instance, if the weight vectors of the SOM are given by $\mathbf{w_s} = \{\vec{w}_{s_1} ... \vec{w}_{s_i} ... \vec{w}_{s_N}\}$, then the subsequent input vectors $\mathbf{w}_{\text{inputs}}$ to Algorithm 1 are $\mathbf{w}_{\text{inputs}} = \{\mathbf{w_s} \cup \vec{w}_t\}$. By presenting the inputs to the GSOM algorithm in this manner, the resulting SOM approximates and integrates previously learned task knowledge and the knowledge of newly learned tasks.

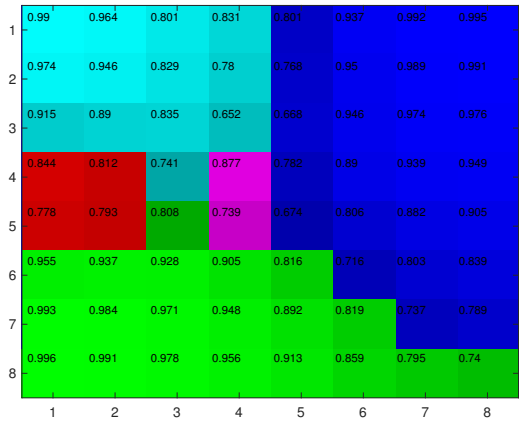

Figure 3: A visual depiction of an 8×8 SOM resulting from the simulations. The color of each node is derived from the most similar task in Figure 2. The intensity of the color is in proportion to the value of this similarity metric (indicated over each SOM node).

Figure 3 shows a sample 8×8 SOM, which was learned by the agent after 1000 $Q-$learning episodes. The color of each SOM node in Figure 3 corresponds to the task in Figure 2 that has the maximum cosine similarity between its value function weights and the weight vector associated with the SOM node. Further, the brightness of this color is in proportion to the value of this cosine similarity. In Figure 3, these values are overlaid and displayed on top of each node. The different colors and associated cosine similarity values of each SOM node in Figure 3 suggests that the SOM stores knowledge of a variety of related tasks in a structured manner.

It is also seen from Figure 3 that the nodes corresponding to the knowledge of tasks that are most closely related to tasks $1, 4$ and 5 are clustered together, and those related to tasks 2 and 3 are distinct, and are stored in separate clusters. In addition, the allocation of the SOM nodes occurs as per the differences in the tasks themselves. For example, in the SOM shown in Figure 3, 25 nodes are allocated to tasks that are most similar to task 3, 19 nodes to tasks related to task 2, and 20 nodes to tasks related to tasks $1, 4$ and 5 combined. This demonstrates that the allocation of nodes is done as per the characteristics of the tasks, and not merely according to the number of tasks. When a number of similar tasks are learned, simply storing their value function weights would result in significant redundancies. Such redundancies are avoided by the SOM-based approach described here.

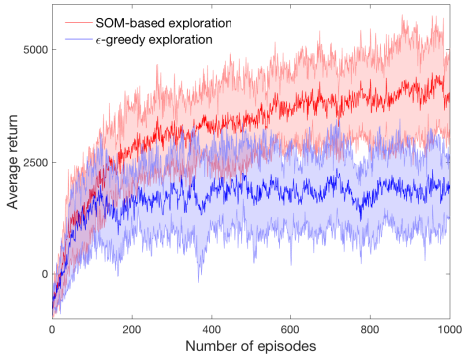

Figure 4: A sample plot of the nature of the learning improvements brought about by SOM-based exploration ($\epsilon = 0.3$, $G_T = 0.3$). The solid lines represent the mean of the average return for 10 $Q-$learning runs of 1000 episodes each, whereas the shaded region marks the standard deviation associated with this data.

Although the SOM does not necessarily retain the exact value functions of previously learned tasks, it can be used to guide the exploration of an agent while learning a new task. This is especially true if the new task is closely related to one of the previously learned tasks. Figure 4 depicts this phenomenon for task 5 ($\epsilon = 0.3$, $G_T = 0.3$), with higher returns being achieved at a significantly faster rate using the SOM-based exploration strategy in Section 3.2. In both exploration strategies, exploratory actions are executed with the same probability, but SOM-based exploration achieves a better performance, as knowledge of related tasks (in this case, tasks 1 and 4) from previous experiences allows the agent to take more informed exploratory actions. This is also supported by Figure 5, which shows

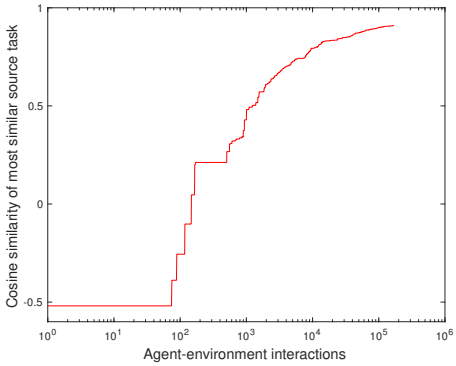

Figure 5: Cosine similarity between a target task and its most similar source task as the agent interacts with its environment

the evolution of the cosine similarity between the value function weights of the target task and the most similar weight vector in the SOM as the agent interacts with its environment. With a greater number of agent-environment interactions, the estimates of the agent's target task weight vector improves, and it receives more relevant advice from the SOM. This trend is probably responsible for the learning improvements seen in Figure 4.

Figure 6 shows the average return per episode for different tasks and different values of ϵ, using the two exploration strategies. The

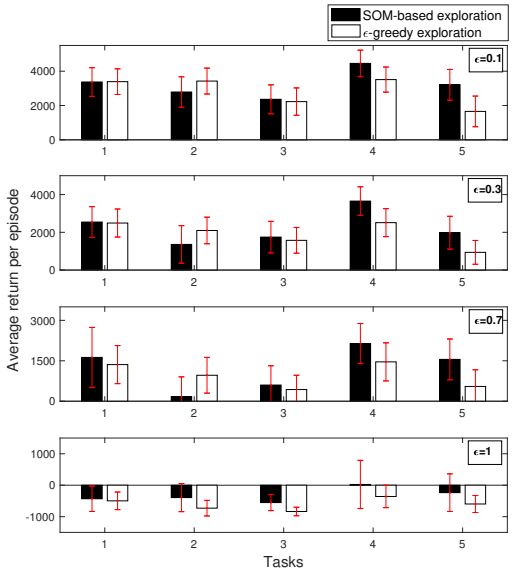

Figure 6: Comparison of the average returns accumulated for different tasks using the SOM-based and $\epsilon-$ greedy exploration strategies

values plotted are averaged over 10 runs. The return is computed after each episode by allowing the agent to greedily exploit the value functions starting from 100 randomly chosen points in the environment for 100 steps. As observed in Figure 6, SOM-based exploration consistently results in higher average returns for related tasks 4 and 5. Its performance on the unrelated tasks 2 and 3 are generally comparable to that of the $\epsilon-$greedy approach. Although task 1 is related to tasks 4 and 5, it is the first task learned, and hence, does not benefit from the use of previous knowledge. Hence, the transfer advantage is not observed for task 1.

In addition to the improvements described, the SOM-based approach to conducting knowledge transfer also offers advantages in terms of the scalability of knowledge storage. This is depicted in Figure 7, which shows the number of nodes needed for storing the knowledge of up to 1000 tasks, with different values of the GSOM threshold parameter G_T. It is clear that as the number of learned tasks increases, the number of nodes required per task decreases, making the SOM-based knowledge storage approach more viable.

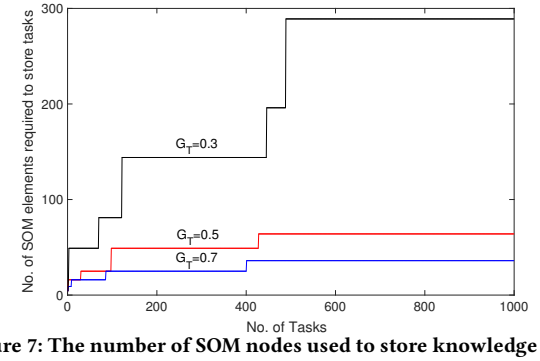

Figure 7: The number of SOM nodes used to store knowledge for up to 1000 tasks, for different values of growth threshold G_T

The simulations demonstrate that using a SOM knowledge base to guide the agent's exploratory actions help achieve faster learning when the target tasks are related to the previously learned tasks. Moreover, the nature of the transfer algorithm is such that even in the case where the source tasks are unrelated to the target task, the learning performance does not exhibit drastic drops, unlike the case where value functions of source tasks are directly used to initialize or modify the value function of a target task. Another advantage of the approach proposed here is that it can be easily applied to different representation schemes (tabular representations, neural networks etc.,), as long as the same action space and representation is used for the target and source tasks. In addition, with regards to the storage of knowledge of learned tasks, we demonstrated that the SOM-based approach offers a scalable alternative to explicitly storing the value function weights of all the learned tasks.

Despite these advantages, several issues remain to be addressed. The most fundamental limitation of this approach is that it is applicable only to situations where tasks differ solely in their reward functions. This may prohibit its use in many practical applications. Moreover, the approach as described executes any action advice that it is provided with. The decision to execute the advised actions could be carried out in a more selective manner, perhaps based on the cosine similarity between the target task and the advising node of the SOM. Apart from this, and the several other possible variants to this approach, ways to automate the selection of the threshold parameters, establishing theoretical bounds on the learning performance and approaches to quantify the efficiency of the knowledge storage mechanism may be future directions for research.

5 CONCLUSIONS

We described an approach to efficiently store and reuse the knowledge of learned tasks using self organizing maps. We applied this approach to an agent in a simulated multi-task navigation environment, and compared its performance to that of an $\epsilon-$greedy approach for different values of the exploration parameter ϵ. Results from the simulations reveal that a modified exploration strategy that exploits the knowledge of previously learned tasks improves the agent's learning performance on related target tasks. Overall, our results indicate that the approach proposed here transfers knowledge across tasks relatively safely, while simultaneously storing relevant task knowledge in a scalable manner. Such an approach could prove to be useful for agents that operate using the reinforcement learning framework, especially for real-world applications such as autonomous robots, where scalable knowledge storage and sample efficiency are critical factors.

ACKNOWLEDGEMENTS

This work is partially supported by a President's Graduate Fellowship (T.G.K., Ministry of Education, Singapore)

REFERENCES

[1] Damminda Alahakoon, Saman K Halgamuge, and Bala Srinivasan. 2000. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on neural networks* 11, 3 (2000), 601–614.

[2] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. 2014. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence.*

[3] James L Carroll and Kevin Seppi. 2005. Task similarity measures for transfer in reinforcement learning task libraries. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 2. IEEE, 803–808.

[4] Norm Ferns, Prakash Panangaden, and Doina Precup. 2004. Metrics for finite Markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 162–169.

[5] Matthieu Geist, Bruno Scherrer, et al. 2014. Off-policy learning with eligibility traces: a survey. *Journal of Machine Learning Research* 15, 1 (2014), 289–333.

[6] Thommen George Karimpanal and Erik Wilhelm. 2017. Identification and off-policy learning of multiple objectives using adaptive clustering. *Neurocomputing* 263 (2017), 39 – 47. https://doi.org/10.1016/j.neucom.2017.04.074 Multiobjective Reinforcement Learning: Theory and Applications.

[7] Teuvo Kohonen. 1998. The self-organizing map. *Neurocomputing* 21, 1 (1998), 1–6.

[8] Alessandro Lazaric. 2012. *Transfer in Reinforcement Learning: A Framework and a Survey*. Springer Berlin Heidelberg, Berlin, Heidelberg, 143–173. https://doi.org/10.1007/978-3-642-27645-3_5

[9] Miao Liu, Girish Chowdhary, Jonathan P How, and L Carrin. 2012. Transfer learning for reinforcement learning with dependent Dirichlet process and Gaussian process. *NIPS, Lake Tahoe, NV, December* (2012).

[10] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342* (2015).

[11] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[12] Mark Ring, Tom Schaul, and Juergen Schmidhuber. 2011. The two-dimensional organization of behavior. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, Vol. 2. IEEE, 1–8.

[13] Mark Bishop Ring. 1994. *Continual learning in reinforcement environments*. Ph.D. Dissertation. University of Texas at Austin Austin, Texas 78712.

[14] Jinhua Song, Yang Gao, Hao Wang, and Bo An. 2016. Measuring the distance between finite markov decision processes. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 468–476.

[15] Richard S Sutton and Andrew G Barto. 2011. Reinforcement learning: An introduction. (2011).

[16] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 761–768.

[17] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[18] Teck-Hou Teng, Ah-Hwee Tan, and Jacek M Zurada. 2015. Self-organizing neural networks integrating domain knowledge and reinforcement learning. *IEEE transactions on neural networks and learning systems* 26, 5 (2015), 889–902.

[19] Sebastian Thrun and Joseph O'Sullivan. 1998. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to learn*. Springer, 235–257.

[20] Lisa Torrey and Matthew Taylor. 2013. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1053–1060.

[21] Adam White, Joseph Modayil, and Richard S Sutton. 2012. Scaling life-long off-policy learning. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*. IEEE, 1–6.

[22] Yusen Zhan and Matthew E Taylor. 2015. Online transfer learning in reinforcement learning domains. *arXiv preprint arXiv:1507.00436* (2015).

[23] Matthieu Zimmer, Paolo Viappiani, and Paul Weng. 2014. Teacher-student framework: a reinforcement learning approach. In *AAMAS Workshop Autonomous Robots and Multirobot Systems*.