Adaptive Adjacency Kanerva Coding for Memory-Constrained Reinforcement Learning

Wei Li[0000−0003−0268−0903]✉ and Waleed Meleis

Northeastern University, Boston, MA 02115, USA
li.wei@husky.neu.edu, meleis@ece.neu.edu

Abstract. When encountering continuous, or very large domains, using a compact representation of the state space is preferable for practical reinforcement learning (RL). This approach can reduce the size of the state space and enable generalization by relating similar or neighboring states. However, many state abstraction techniques cannot achieve satisfactory approximation quality in the presence of limited memory resources, while expert state space shaping can be costly and usually does not scale well. We have investigated the principle of Sparse Distributed Memories (SDMs) and applied it as a function approximator to learn good policies for RL. This paper describes a new approach, adaptive adjacency in SDMs, that is capable of representing very large continuous state spaces with a very small collection of prototype states. This algorithm enhances an SDMs architecture to allow on-line, dynamically-adjusting generalization to assigned memory resources to provide high-quality approximation. The memory size and memory allocation no longer need to be manually assigned before and during RL. Based on our results, this approach performs well both in terms of approximation quality and memory usage. The superior performance of this approach over existing SDMs and tile coding (CMACs) is demonstrated through a comprehensive simulation study in two classic domains, Mountain Car with 2 dimensions and Hunter-Prey with 5 dimensions. Our empirical evaluations demonstrate that the adaptive adjacency approach can be used to efficiently approximate value functions with limited memories, and that the approach scales well across tested domains with continuous, large-scale state spaces.

Keywords: Reinforcement learning · Function approximation · Kanerva coding · Dynamic generalization.

1 Introduction

Reinforcement learning (RL) techniques have been shown to develop effective policies for diverse tasks and application domains. However, in general, applying RL to practical problems is challenging in the presence of continuous, large-scale state spaces. In this case, the size of the state space can grow exponentially with the number of state variables, causing a proportional increase in the size of the

value table needed to store the value functions. The training time needed to explore large state spaces is expensive and should be reduced. In the domains with continuous state spaces, it is impractical to enumerate and maintain all possible states. Associating approximating techniques with RL algorithms has enabled the approach to be effectively applied to many practical problems. However, some challenges still remain.

In the literature, a number of works [5,14,17] have proposed applying approximating techniques to large state-action spaces. Function approximation is a landmark technique that can effectively reduce the size of the search space by representing it with a compact and parameterized version [6]. The value table that stores value functions is replaced by an approximate and compact table. Tile coding (CMACs) [18], adaptive tile coding [12,19] and tree-based state space discretization [2] are prominent function approximation approaches that have been evaluated in various problem domains. However, when solving practical real-world problems, limitations on the coding schemes used in those approaches, e.g., inflexible partitions or computationally costly abstractions of the state space, make them hard to scale well or guarantee high performance in domains that are very large, or have continuous state and action spaces.

Kanerva coding [8], also known as Sparse Distributed Memories (SDMs), has emerged as an efficient and viable solution to deal with complex, large state spaces. This technique considers such a setting that the whole state space is represented by a substantially small subset of the state space. Since a set of states is used as the basis for approximation, the complexity of this scheme is dependent only on the number of states in the subset and not on the number of dimensions of the state space, making it suitable for solving problems with continuous, large state spaces. Kanerva-based learners' effectiveness in problems with continuous, large state spaces has been evaluated by [11,20]. However, to improve the performance of Kanerva coding algorithms, the allocation of the states in the subset need to be optimally arranged based on considered visited state areas which is still an ongoing research field and needs further exploration.

In this paper, based on the principles of SDMs, we describe a function approximator that adaptively adjusts the radii of receptive fields of selected *memory locations* (also called *prototypes*). Such adjustments can directly change the generalization abilities of memory locations to cover certain visited states when necessary, making it possible to fully utilize limited memory resources while capturing enough of the complexity of visited states. This approach can learn a good policy through its attempt to produce a high-quality generalization of the selected memory locations by appropriately adjusting and utilizing various levels of generalization for each memory location. Experimental results show the superior performance of this proposed approach both in terms of memory usage and approximation quality.

The rest of the paper is organized as follows. In Sect. 2, we give an overview of the RL Sarsa algorithm [18] and present the principals and limitations of existing Kanerva-based approaches that serve as a linear function approximator when applying RL algorithms in problem domains with continuous, very

large state spaces. In Sect. 3, we introduce our adaptive adjacency approach for flexible memory adjacency adjustments in SDMs, and then we describe the implementation of this approach. In Sect. 4, we show our experimental results and analysis. Finally, we give our conclusions in Sect. 5.

2 Related Work

RL has been successfully applied to a wide range of sequential decision problems and problem domains, such as video streaming [3], computer game playing [7], physical science [10] and networking [11,13]. RL can learn on-line without the need for initial input data due to its ability to learn through a trial-and-error mechanism. In RL, the learning agent receives a reward after performing particular actions in a state. It transits to a new state based on the environment's response to currently performed action. Its goal is to develop a policy, a mapping from a state space to an action space, that maximizes the long-term rewards. Many RL techniques have been proposed, i.e., Sarsa and Q-learning.

2.1 Sarsa Algorithm

Sarsa [18] is an on-policy temporal-difference (TD) control algorithm in RL. At each time step t, the learning agent observes a state s_t from the environment and receives a reward r_{t+1} after applying an action a_t to this state s_t. Current state s_t then transitions to next state s_{t+1} and the learning agent chooses an action a_{t+1} at state s_{t+1}. Sarsa algorithm updates the expected discounted reward of each state-action pair, in this case, the action-value function $Q(s_t, a_t)$, as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \ , \qquad (1)$$

where $\alpha_t(s_t, a_t)$ is the learning rate ($0 \leq \alpha_t(s_t, a_t) \leq 1$), and γ is the discount factor ($0 < \gamma \leq 1$). The value of each action-value function $Q(s_t, a_t)$ is learned and stored in a table, called Q-table. The use of Q-table is suitable for small state-action space but impractical with large or continuous state-action spaces due to the considerable memory consumption and unbearable long training time.

Although Sarsa performs well in terms of both learning quality and convergence time when solving problems with small state and action spaces, Sarsa performs poorly when applied to large-scale, continuous state and action spaces due to its inefficient use of Q-table. Function approximation techniques are therefore proposed to solve this problem where, on the one hand, the values of a relatively much small set of states/prototypes are stored instead of storing the Q-table for all visited states, and on the other hand, the knowledge among similar states are generalized and transferable.

2.2 Kanerva-Based Learning

Kanerva Coding (SDMs). The idea of Kanerva coding [18] considers to use a relatively small set of states as *prototypes* to store and estimate the value

functions. In Kanerva coding, a state s or prototype p_i consists of n state-variables and each state-variable describes a measurement in one dimension with a numeric value. A state s and a prototype p_i are said to be *adjacent* if s and p_i differ in at most a given number of state-variables, for example differing in at most one state-variable. The *membership grade* $\mu(s, p_i)$ of state s with respect to prototype p_i is defined where $\mu(s, p_i)$ is equal to 1 if s is adjacent to p_i, and 0 otherwise. Kanerva coding maintains a set of k prototypes as parameterized elements for approximation and a value $\theta(p_i, a)$ is stored and updated for each prototype p_i with respect to the action a.

The approximation of the value of a state-action pair (s, a) is calculated by a linear combination of θ-values of all adjacent prototypes of state s, defined as follows:

$$\hat{Q}(s, a) = \sum_{p_i \in D_s} \theta(p_i, a)\mu(s, p_i) \ , \tag{2}$$

where D_s is the set of adjacent prototypes with respect to state s. The θ-value of each prototype p_i with respect to action a is updated by the rule of Sarsa algorithm as follows:

$$\theta(p_i, a) \leftarrow \theta(p_i, a) + \frac{\mu(s, p_i)}{\sum_{p_k \in D_s} \mu(s, p_k)} \alpha(s, a)[r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)] \ . \tag{3}$$

Dynamic Kanerva Coding. Kanerva coding works well in large and complex problem domains. However its performance is sensitive to the allocation of prototypes in use [9]. The set of prototypes should be appropriately selected from the state space, i.e., choosing ones that are well distributed around the trajectories of visited states, otherwise, the quality of function approximation could be greatly degraded. In fact, selecting/reallocating the set of prototypes before and/or during the learning process to maintain a sufficient complexity of function approximation is the central issue that needs to be solved for any Kanerva-based algorithms.

There are two classes of solutions typically used to solve this problem. The first class of approaches starts with an empty set of prototypes and then incrementally builds up the set that covers areas of interest in the state space by adding a certain number of prototypes. One heuristic used to add prototypes is to add a new prototype if it is at least a certain distance away from all existing prototypes [4,16]. The heuristic is inflexible. When the complexity of the required approximations is already satisfied, there will be no need to introduce additional prototypes. This heuristic could introduce many redundant prototypes resulting in slow learning and extra memory costs. Another approach is to add a collection of new prototypes from the neighborhood area of a sample of recently visited states, provided each of them is far enough from all existing prototypes [15]. However, this heuristic requires a sophisticated strategy to select appropriate prototypes from the neighborhood area of a sample of visited states, particularly when avoiding conflicts with existing prototypes. In addition, both heuristics incur considerable computational cost, i.e., calculating and leveraging the distance from the visited state to all existing prototypes.

The second class of approaches [1,21] starts with an initial set of randomly selected prototypes. It has much less computational cost. To guarantee efficiency in function approximations, it tends to replace poorly-performing/rarely utilized prototypes with promising prototypes to gradually adjust the allocation of the original set of prototypes. However, prototypes should not be replaced only based on their periodically-observed performance and too-frequent prototype replacements could lose previously-learned information. In real implementation, both classes of approaches may combine together to pursue even better performance.

Our adaptive adjacency approach addresses the weaknesses of both classes of techniques while achieving satisfactory performance across multiple problem domains. This approach also lowers computational cost by eliminating the operations to add and delete prototypes.

3 Adaptive Adjacency Kanerva Coding

3.1 Prototype Adjacencies

In our experiments, the state space is continuous and each input sample data or prototype is represented by a sequence of real values, i.e., each dimension is represented by a real value. The distance between an input sample data $s = \langle v_1, ..., v_n \rangle$ and one of the prototypes, $p_i = \langle h_1, ..., h_n \rangle$, in the prototype set is defined by the following distance functions:

$$d(s, p_i) = \sum_{j=1}^{n} d_j(s, p_i) \ , \tag{4}$$

$$d_j(s, p_i) = \begin{cases} 0 : |v_j - h_j| \leq \sigma_i \beta_j \ , \\ 1 : \text{otherwise} \ . \end{cases} \tag{5}$$

Equation (5) is very important in our approach. In (5), β_j is the basis radius of the receptive field in j_{th} dimension, and σ_i is the *adjacency factor* that is multiplied by β_j to adjust the radii of the receptive field of each prototype p_i. Note that $d(s, p_i)$ represents the number of state variables at which state s and prototype p_i differ by more than $\sigma_i \beta_j$. The value of σ_i can be adaptively changed by our algorithm for each prototype p_i during the regular learning process. In our algorithm, β_j does not change.

We define the *membership grade* $\mu(s, p_i)$ of state s with respect to prototype p_i to be a continuous function of the distance between s and p_i as follows:

$$\mu(s, p_i) = e^{-d(s, p_i)} \ . \tag{6}$$

Instead of a binary membership grade, we use a continuous membership grade in $[0, 1]$ to assign a weighted value update (based on distance) to the θ-value of each adjacent prototype. Note that the membership grade of a state with respect to an identical or 0-distance prototype would be 1, and the membership grade of a state with respect to a distant prototype approaches to 0. And a state s

Algorithm 1: Adaptive Adjacency Kanerva Coding with Sarsa Algorithm

Input: $\boldsymbol{p}$: a set of randomly selected prototype samples, $\boldsymbol{\theta}$: associated with $\boldsymbol{p}$ and initialized to $\mathbf{0}$, $\boldsymbol{\sigma}$: associated with $\boldsymbol{p}$ and initialized to $\mathbf{1}$, $\boldsymbol{\beta}$: β_j is a basis radius of receptive field in j_{th} dimension of the state space.
Output: learned $\boldsymbol{\theta}$-values and adjusted $\boldsymbol{\sigma}$-values

```
Procedure Main()
    for each episode do
        Initialize s
        Choose a in s based on policy derived from θ
        for each step of episode do
            Take action a, observe r, s'
            p_activated = the set of prototypes activated by s'
            M = the size of p_activated
            if M ≥ N then
                Choose a' in s' based on p_activated and ε-greedy approach
                Update θ values with (3)
            else
                p_activated = AdaptiveAdjacency(σ, p, s', p_activated)
                Choose a' in s' based on p_activated and ε-greedy approach
                Update θ values with (3)
            Set s = s' and a = a'
        Until s is terminal
Procedure AdaptiveAdjacency(σ, p, s', p_activated)
    Initialize the factor k to 0
    for each round of adjusting radii of receptive field do
        Increase k by 1
        for each prototype p_i in p do
            if p_i is not in p_activated and σ_i has not been modified more than L
             times then
                Increase σ_i by a multiplicative factor, (1 + kφ), where φ ∈[0,1]
                if p_i is activated by its newly increased radii of receptive field
                 then
                    Add p_i to p_activated
                    M = the size of p_activated
                    if M ≥ N then
                        Return
                else
                    Reset σ_i to its previously not-yet-increased value
```

and a prototype p_i are said to be *adjacent* if $d(s, p_i) \leq \left\lfloor \frac{n}{2} \right\rfloor$. We also say that a prototype is *activated* by a state if the prototype is adjacent to the state.

The estimates of the Q-values and the updates to the θ-values use the same definitions described in (2) and (3), respectively. In (5), parameter σ_i plays an important role in adjusting the radius of the receptive field in each dimension, giving each prototype the ability to adjust its sensitivity to visited states. The

ability of each prototype to generalize can now be adjusted, based on information gathered during the learning process.

3.2 Adaptively Changing Adjacencies

Our approach begins with a set of randomly-selected prototypes from the state space. Each prototype p_i has a corresponding factor σ_i whose value allows for on-line adjustments that directly change the radii of the receptive field of prototype p_i. A parameter N (denoted as *active parameter*), the minimum number of adjacent prototypes for a visited state, is used to control the amount of generalization that is provided.

For each encountered state, the algorithm adjusts prototypes' receptive fields so that state is adjacent to at least N prototypes. More specifically, when encountering a state s, if N or more prototypes are adjacent to s, we perform value updates using the regular SDMs routine. However, if only M prototypes, where $0 \leq M < N$, are adjacent to s, our approach enlarges the receptive fields of certain nearby prototypes enough to allow these prototypes to be adjacent to s. This allows s to meet the adjacency requirement.

A detailed description of our adaptive adjacency approach is presented in Algorithm 1. Starting from line 20, in each round of increments, the algorithm provisionally adjusts the boundary of the receptive field for each prototype p_i that is not activated by the current state by increasing σ_i by a multiplicative factor that is > 1. This process continues through additional rounds of increments until the state is adjacent to at least N prototypes. The algorithm then fixes the receptive field boundaries for all newly activated prototypes, and resets the receptive field boundaries for the remaining prototypes that failed to be activated back to the values held before the increments began. In each round, the multiplicative factor used to augment the receptive fields increases.

Line 23 of Algorithm 1 implements a constraint that ensures that the radii of receptive field of a particular prototype can only be adjusted a limited number of times, i.e., at most L times. This rule guarantees a fine-grained generalization for the set of prototypes and can avoid over-generalization.

4 Experimental Evaluation

We first present an experimental evaluation of our adaptive adjacency Kanerva coding (AdjacencyK) algorithm with Sarsa on the Mountain Car domain [18], a classic RL benchmark. The problem has a continuous state space with 2 state variables: the car position and velocity. We then present an experimental evaluation using a more complex benchmark, a variant of the Hunter-Prey domain that has a continuous, much large state space with 5 state variables (see details in Sect.4.2). We compare the performance of AdjacencyK algorithm with that of popular CMACs algorithm, pure Kanerva coding (also called traditional SDMs, denoted as PureK) as well as one commonly used dynamic Kanerva coding algorithm (an approach that selectively deletes and generates certain expected prototypes proposed and utilized in [1,21], denoted as DynamicK).

4.1 Evaluate Performance with Mountain Car Domain

In a Mountain Car task, the learning agent attempts to learn a policy to drive an underpowered car up a steep hill. Each task consists of a sequence of episodes, and each episode ends and resets to the start state if the goal is achieved, i.e., the car reaches the top position of the hill, or if the agent-environment interaction exceeds the maximal number of time steps allowed in an episode.

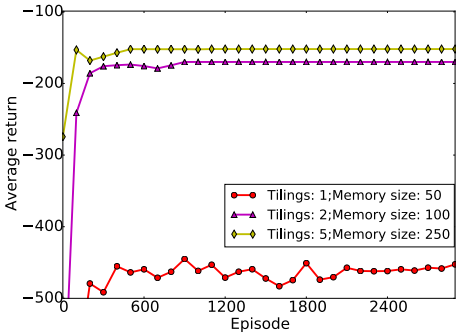
Tile coding (or CMACs) has been successful in helping RL algorithm learn policies for the Mountain Car domain [18]. The performance of CMACs relies largely on obtaining an effective memory layout on the state space. In CMACs, a *tiling* exhaustively partitions the whole state space into *tiles* and an input state sample falls within the receptive field of exactly one tile of a tiling. The receptive field of a tile indicates its generalization ability. To avoid over-generalization and to increase the resolution of needed approximations, typically a collection of overlapped and slightly offset tilings is used.

We ran experiments of our proposed AdjacencyK algorithm on the standard Mountain Car domain and then analyzed its performance over CMACs, PureK and DynamicK algorithms. All our experiments start with the same initial state in which the car is at rest at the bottom of the valley and its velocity is 0. Each prototype's receptive field radius in each dimension in AdjacencyK, PureK and DynamicK is set equal to the size of a tile in corresponding dimension in CMACs. Since the number of tilings in CMACs greatly affects the agent's ability to distinguish states in the state space, our experiments test different number of tilings to present a complete set of empirical results. We also explore the differences in performance when using various values of parameter N in proposed AdjacencyK algorithm.

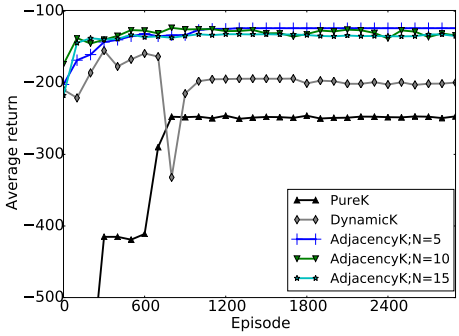
The PureK and DynamicK have the same initial set of randomly selected prototypes as the one used in our AdjacencyK approach and the initial radii of the receptive fields are also the same. Note that the radii in PureK and DynamicK are unable to change during learning.

The results of applying CMACs, PureK, DynamicK and AdjacencyK algorithms to the Mountain Car domain are shown in Fig. 1. The results are averaged on 8 repeated runs and each episode starts with the same initial state. The best selected RL parameters are set as: $\alpha = 0.5$, $\epsilon = 0.0$, and $\gamma = 1.0$. We set ϕ to 0.5 and L to 1 in AdjacencyK algorithm. The results show that the AdjacencyK approach outperforms CMACs, PureK and DynamicK both in terms of learning quality and memory usage. As demonstrated by Fig. 1a and 1b, the adaptive adjacency approach learns the task faster and converges to a higher return in a shorter time than CMACs, PureK and DynamicK algorithms.

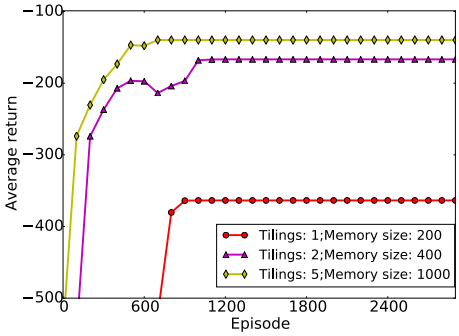
We also evaluate the sensitivity of the performance of considered algorithms to variations in the settings of important parameters. In Fig. 1b, we show that different values of active parameter N, i.e., 5, 10, 15, in proposed AdjacencyK algorithm have only a small impact on average returns and therefore do not need carefully tuned. Furthermore, the two graphs in the top row of Fig. 1 show data assuming the same size base receptive field (or tile size), and the two graphs in the bottom row show data assuming a smaller size base receptive field (or tile size).



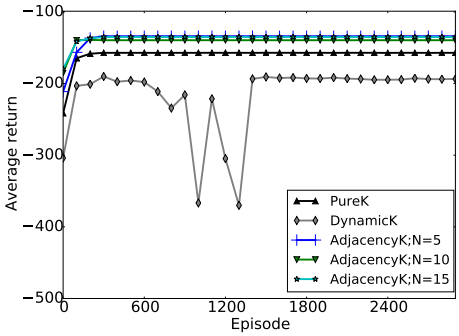
(a) CMACs algorithm with the tile size $\langle 0.34, 0.014\rangle$. Memory sizes are 50, 100 and 250 respectively.



(b) Comparisons of PureK, DynamicK and AdjacencyK. Each algorithm sets base radii $\langle 0.34, 0.014\rangle$ and memory size 50.



(c) CMACs algorithm with the tile size $\langle 0.17, 0.007\rangle$. Memory sizes are 200, 400 and 1000 respectively.

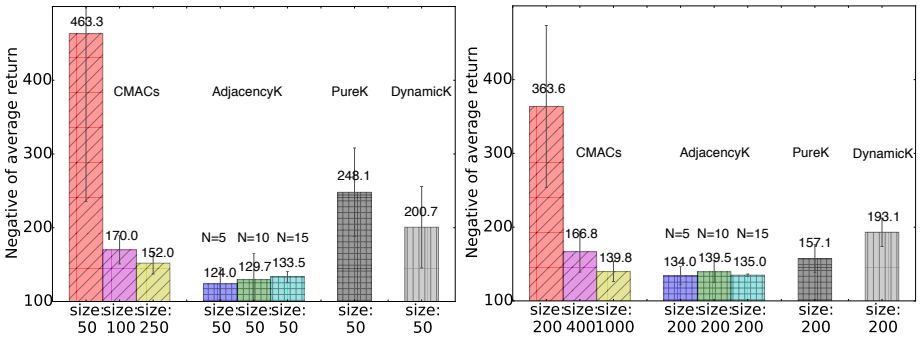


(d) Comparisons of PureK, DynamicK and AdjacencyK. Each algorithm sets base radii $\langle 0.17, 0.007\rangle$ and memory size 200.

Fig. 1: Comparisons of average returns of 4 algorithms with different sizes of tiles and radii of receptive fields in the Mountain Car domain

The results in Fig. 1c and 1d demonstrate that the adaptive adjacency approach still outperforms CMACs, PureK and DynamicK algorithms both in terms of learned policies and memory usage as the sizes of base radii of receptive fields (or tiles) decrease from $\langle 0.34, 0.014\rangle$ to $\langle 0.17, 0.007\rangle$. Interestingly, based on the new settings on the radii and memory size, Fig. 1d shows that the performance of PureK was largely improved, e.g., PureK with memory size of 200 learned a slightly better policy than CMACs with memory size of 400.

Figure 2 demonstrates the memory savings of our proposed adaptive adjacency approach (AdjacencyK) with two different settings of the sizes of base radii of receptive fields. With each setting, we show the negative of average returns of CMACs with various memory sizes (refer to the number of tiles), AdjacencyK with a fixed memory size (refer to the number of prototypes) but various values of N, and PureK as well as DynamicK with the same fixed memory size as Ad-



(a) Base radii $\langle 0.34, 0.014 \rangle$ (b) Base radii $\langle 0.17, 0.007 \rangle$

Fig. 2: Memory savings of AdjacencyK with two different sizes of base radii

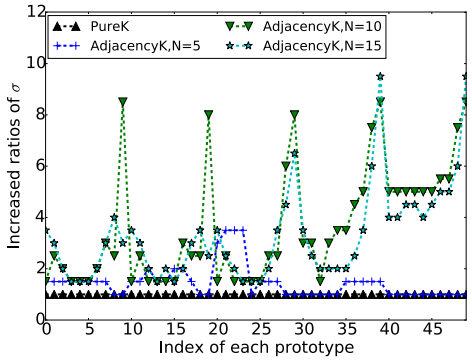
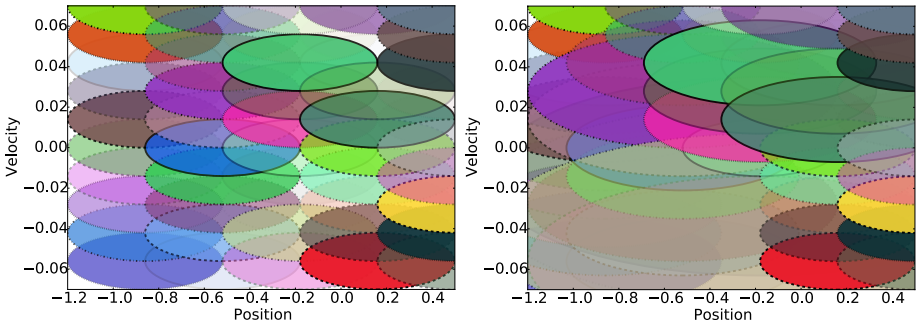


Fig. 3: Changes of the receptive field of each prototype with PureK and AdjacencyK in the Mountain Car task. The base radii are $\langle 0.34, 0.014 \rangle$. The allocations of prototypes are exactly same for both algorithms when the learning starts.

jacencyK after 1500 episodes of learning in our experiments. When testing with one setting of base radii of receptive fields (or tiles), i.e., $\langle 0.34, 0.014 \rangle$, the left three bars in Fig. 2a show that the performance of CMACs is poor and unstable when the memory size is small (i.e., 50), and its performance improves significantly when the memory size is relatively large (i.e. 250) which is reasonable. And as demonstrated by the three clustered bars of AdjacencyK algorithm, our adaptive adjacency approach can achieve even better performance than CMACs when only using 20% of the memories. At the same time, it can be observed in Fig. 2a that given the same initial set of prototypes, our approach outperforms both Kanerva-based algorithms, PureK and DynamicK, with the same memory usages. And generally, these observations are also true for results shown in Fig. 2b that has a different setting of base radii of receptive fields (or tiles).



(a) Before learning (b) After learning

Fig. 4: Changes to the coverages of receptive fields for all prototypes with the AdjacencyK approach in a Mountain Car task

We argue that the improved performance in our proposed approach results from changes in the sizes of receptive fields of prototypes during learning. Figure 3 shows these changes to the sizes of receptive fields across all prototypes. As shown in Fig. 3, initialized with the same set of prototypes and base radii, PureK algorithm does not change the sizes of receptive fields during RL while our proposed algorithm allows the receptive fields appropriately changed (see the rises of the curves in the graph) and thus all prototypes' generalization capabilities are flexibly adjusted so that required approximation complexity is guaranteed for input state samples. In other words, during learning, each input state would be adjacent to a sufficient number of prototypes and each prototype has the chance to be activated if being adjusted to have a required size of receptive field. Note that a larger value of N means that input states expect to have higher complexity of approximations, i.e., have more adjacent prototypes for function predictions. Therefore, a larger N implies that the receptive fields of prototypes will need to be increased more so that their generalization abilities are enhanced in order to be more easily activated by input states. As shown in Fig. 3, the learning agent with a bigger N value, i.e., $N = 10$ or $N = 15$, increases its prototypes' radii of receptive fields more obviously than that with $N = 5$.

In our experiments, the adaptive adjacency approach gave similar good learning results when setting N to 5, 10 or 15. We argue that these good results are a result of having a collection of prototypes with necessarily adjusted sizes of receptive fields that overlap with visited input states sufficiently. Figure 4 shows the changes to the sizes of receptive fields of 50 uniformly distributed prototypes when performing the adaptive adjacency approach to a Mountain Car task. Note that all the data depicting the radii changes in this figure came from one of our experiments that learned a good policy. The detailed results of this experiment are already shown in Fig. 1b and Fig. 3 in which the base radii are $\langle 0.34, 0.014 \rangle$, the memory size is 50 and the active parameter N is 5.

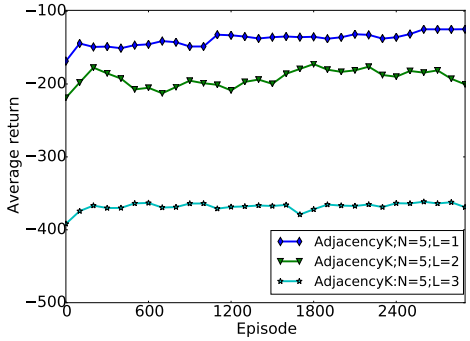


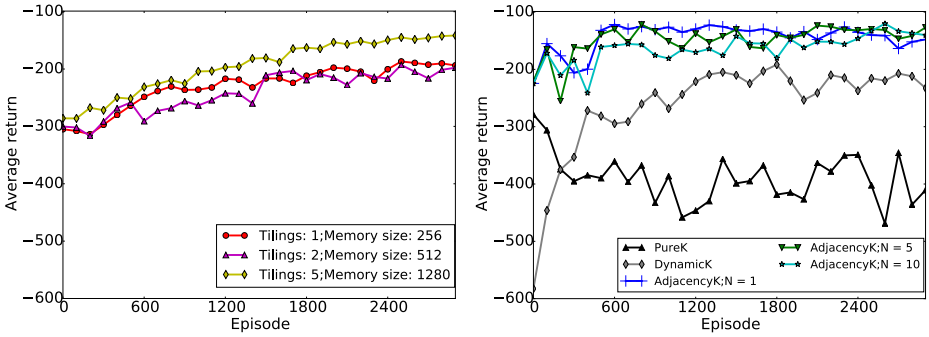
Fig. 5: Average returns of adaptive adjacency approach with different settings of L. Base radii for each prototype is $\langle 0.34, 0.014 \rangle$, and memory size is 50.

The constraint L defined in our algorithm that limits the number of changes to the adjacency factor σ can have a large impact on the learning performance. Figure 5 shows the results of applying our algorithm with 3 different L values to a Mountain Car task. It shows that our algorithm achieves better learning results when using a smaller L. The reason is that a smaller L allows certain distant prototypes to have opportunities to change their receptive fields and thus provides a collection of prototypes with receptive fields of much varying sizes that give finer-grained generalization on the state space. Therefore, in order to achieve best learning quality, we set L to 1 in all our experiments.

4.2 Evaluate Performance with Hunter-Prey Domain

We also evaluate the performance of proposed adaptive adjacency approach on a variant of Hunter-Prey task introduced in [15]. In our experiments, we use one prey and two hunters where the hunters work cooperatively to capture the prey and the prey needs to learn a policy to avoid being captured, e.g., the prey can kill a hunter if only this alone hunter is close enough to the prey. The state space in this task consists of 4 continuous state variables in which the position of each hunter is described by 2 continuous variables, i.e., a radial coordinate and an angular coordinate in a polar coordinate system with the prey as the reference point, and one extra integer variable indicating the number of alive hunters.

To capture the prey, both hunters should approach close enough to the prey, i.e., within 5 units, and the two angles between both hunters need to be $\leq \pi + 0.6$ radians. Each hunter's movements follow a predefined stochastic strategy similar to a ϵ-greedy approach. For example, in the radial direction, each hunter moves 5 units towards the prey if this movement would not get it killed (otherwise, it moves 5 units away from the prey), during all but some fraction ϵ of the time when each hunter makes a 0.2 radians' movement clockwise or counterclockwise. We set ϵ to 0.7 in our experiments to make the hunters less greedy to approach to the prey, otherwise the prey is very likely to kill an individual hunter even



(a) CMACs algorithm with the tile size $\langle 125, 1.57\rangle^2$. Memory sizes are 256, 512 and 1280 respectively.

(b) Comparisons of PureK, DynamicK and AdjacencyK. Each algorithm sets base radii $\langle 125, 1.57\rangle^2$ and memory size 256.

Fig. 6: Comparisons of average returns of 4 algorithms in the Hunter-Prey domain

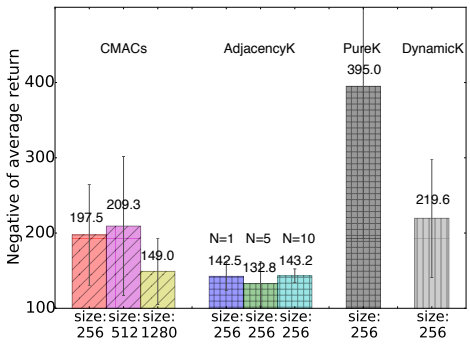


Fig. 7: Memory savings of the AdjacencyK algorithm

with random movements, giving an unexpected high return. The prey moves 5 units per time step in 4 available directions: up, right, down, left in the polar coordinate system. The episode starts with an initial state in which each hunter is randomly placed on a cycle of radius 500 units centered on the prey. If the prey kills one hunter (reward of 1) or gets captured (reward of -200), the episode ends, otherwise, the interaction between prey and hunters continues (reward of -1 per time step).

We show experimental results of our AdjacencyK algorithm and compare its performance with CMACs, PureK and DynamicK algorithms in Fig. 6 and Fig. 7. The experimental configuration is similar to that of the Mountain Car task and RL parameters are also optimally selected. The ϕ is set to 0.3 and L is set to 1 in AdjacencyK algorithm. The AdjacencyK approach obviously learns better policies than CMACs, PureK and DynamicK algorithms. PureK algorithm has

the worst average returns. It cannot learn the task well with a memory size of 256. Although DynamicK has a better performance than PureK, it still cannot beat CMACs. The CMACs algorithm learns slower than AdjacencyK and if their memory sizes are both 256, the learned average return of AdjacencyK ($N = 1$) is 32.8% better than that of CMACs. Note that the CMACs algorithm needs 5-times of the memories, i.e., 1280, to obtain a comparable performance with our algorithm. In other words, the memory usage of our algorithm is 80% smaller than that of CMACs while maintaining a similar performance.

5 Conclusion

In this paper, we extended the architecture of SDMs by using the adaptive adjacency approach as a practical function approximator, and aimed to use RL with this new function approximator to solve complex tasks with continuous, large state spaces. Our new approach enables the generalization capabilities of prototypes to be dynamically adjusted to provide needed complexities of approximations for all input states. This approach supports flexible shaping of the receptive fields of those preselected memory locations in order to cover the areas of interest in the state space. On the one hand, limited memory resources are more efficiently utilized to accomplish qualified value functions' storage and retrieval. On the other hand, performance is less sensitive to the sizes of the receptive fields as well as memory allocations and reallocations.

Our experimental studies demonstrate that our adaptive adjacency approach learns better policies in RL than the popular CMAC algorithm and other existing algorithms in SDMs over two benchmarks. Moreover, this approach has a concise mechanism that is easy to implement, is flexible and scalable in various domains, and does not require expert tunings or prior knowledge about the state space. Finally, the approach works very well when memory resources are constrained.

References

1. Allen, M., Fritzsche, P.: Reinforcement learning with adaptive kanerva coding for xpilot game ai. In: 2011 IEEE Congress of Evolutionary Computation (CEC). pp. 1521–1528. IEEE (2011). https://doi.org/10.1109/CEC.2011.5949796
2. Chernova, S., Veloso, M.: Tree-based policy learning in continuous domains through teaching by demonstration. In: Proceedings of Workshop on Modeling Others from Observations (MOO 2006) (2006)
3. Chiariotti, F., D'Aronco, S., Toni, L., Frossard, P.: Online learning adaptation strategy for dash clients. In: Proceedings of the 7th International Conference on Multimedia Systems. p. 8. ACM (2016). https://doi.org/10.1145/2910017.2910603
4. Forbes, J.R.N.: Reinforcement learning for autonomous vehicles. University of California, Berkeley (2002)
5. Frommberger, L.: Qualitative Spatial Abstraction in Reinforcement Learning. Springer Science & Business Media (2010). https://doi.org/10.1007/978-3-642-16590-0

6. Geist, M., Pietquin, O.: Algorithmic survey of parametric value function approximation. IEEE Transactions on Neural Networks and Learning Systems **24**(6), 845–867 (2013). https://doi.org/10.1109/TNNLS.2013.2247418
7. Hausknecht, M., Khandelwal, P., Miikkulainen, R., Stone, P.: Hyperneat-ggp: A hyperneat-based atari general game player. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp. 217–224. ACM (2012). https://doi.org/10.1145/2330163.2330195
8. Kanerva, P.: Sparse distributed memory and related models, vol. 92. NASA Ames Research Center, Research Institute for Advanced Computer Science (1992)
9. Keller, P.W., Mannor, S., Precup, D.: Automatic basis function construction for approximate dynamic programming and reinforcement learning. In: Proc. of Intl. Conf. on Machine Learning (2006). https://doi.org/10.1145/1143844.1143901
10. Li, L., Baker, T.E., White, S.R., Burke, K.: Pure density functional for strong correlations and the thermodynamic limit from machine learning. Phys. Rev. B **94**(24), 245129 (2016)
11. Li, W., Zhou, F., Meleis, W., Chowdhury, K.: Learning-based and data-driven tcp design for memory-constrained iot. In: 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS). pp. 199–205. IEEE (2016). https://doi.org/10.1109/DCOSS.2016.8
12. Lin, S., Wright, R.: Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning. In: Abstraction, Reformulation, and Approximation (2010)
13. Mao, H., Netravali, R., Alizadeh, M.: Neural adaptive video streaming with pensieve. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 197–210. ACM (2017). https://doi.org/10.1145/3098822.3098843
14. Munos, R., Moore, A.: Variable resolution discretization in optimal control. Machine learning **49**(2), 291–323 (2002)
15. Ratitch, B., Precup, D.: Sparse distributed memories for on-line value-based reinforcement learning. In: Proc. of the European Conf. on Machine Learning (2004). https://doi.org/10.1007/978-3-540-30115-8_33
16. Santamaría, J.C., Sutton, R.S., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive behavior **6**(2), 163–217 (1997)
17. Smart, W.D., Kaelbling, L.P.: Practical reinforcement learning in continuous spaces. In: ICML. pp. 903–910 (2000)
18. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. Bradford Books (1998)
19. Whiteson, S., Taylor, M.E., Stone, P., et al.: Adaptive tile coding for value function approximation. Computer Science Department, University of Texas at Austin (2007)
20. Wu, C., Li, W., Meleis, W.: Rough sets-based prototype optimization in kanerva-based function approximation. In: Web Intelligence and Intelligent Agent Technology (WI-IAT), 2015 IEEE/WIC/ACM International Conference on. vol. 2, pp. 283–291. IEEE (2015). https://doi.org/10.1109/WI-IAT.2015.179
21. Wu, C., Meleis, W.: Adaptive kanerva-based function approximation for multi-agent systems. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3. pp. 1361–1364. International Foundation for Autonomous Agents and Multiagent Systems (2008). https://doi.org/10.1145/1402821.1402872