

Deep Successor Reinforcement Learning

Tejas D. Kulkarni*
BCS, MIT
tejask@mit.edu

Ardavan Saeedi*
CSAIL, MIT
ardavans@mit.edu

Simanta Gautam
CSAIL, MIT
simanta@mit.edu

Samuel J. Gershman
Department of Psychology
Harvard University
gershman@fas.harvard.edu

Abstract

Learning robust value functions given raw observations and rewards is now possible with model-free and model-based deep reinforcement learning algorithms. There is a third alternative, called Successor Representations (SR), which decomposes the value function into two components – a reward predictor and a successor map. The successor map represents the expected future state occupancy from any given state and the reward predictor maps states to scalar rewards. The value function of a state can be computed as the inner product between the successor map and the reward weights. In this paper, we present DSR, which generalizes SR within an end-to-end deep reinforcement learning framework. DSR has several appealing properties including: increased sensitivity to distal reward changes due to factorization of reward and world dynamics, and the ability to extract bottleneck states (subgoals) given successor maps trained under a random policy. We show the efficacy of our approach on two diverse environments given raw pixel observations – simple grid-world domains (MazeBase) and the *Doom* game engine.²

1 Introduction

Many learning problems involve inferring properties of temporally extended sequences given an objective function. For instance, in reinforcement learning (RL), the task is to find a policy that maximizes expected future discounted rewards (value). RL algorithms fall into two main classes: (1) model-free algorithms that learn cached value functions directly from sample trajectories, and (2) model-based algorithms that estimate transition and reward functions, from which values can be computed using tree-search or dynamic programming. However, there is a third class, based on the successor representation (SR), that factors the value function into a predictive representation and a reward function. Specifically, the value function at a state can be expressed as the dot product between the vector of expected discounted future state occupancies and the immediate reward in each of those successor states.

Representing the value function using the SR has several appealing properties. It combines computational efficiency comparable to model-free algorithms with some of the flexibility of model-based algorithms. In particular, the SR can adapt quickly to changes in distal reward, unlike model-free algorithms. In this paper, we also highlight a feature of the SR that has been less well-investigated: the ability to extract bottleneck states (candidate subgoals) from the successor representation under a random policy [35]. These subgoals can then be used within a hierarchical RL framework. In this paper we develop a powerful function approximation algorithm and architecture for the SR using a deep neural network, which we call *Deep Successor Reinforcement Learning* (DSR). This enables learning the SR and reward function from raw sensory observations with end-to-end training.

*Authors contributed equally and listed alphabetically.

²Code and other resources – <https://github.com/Ardavans/DSR>

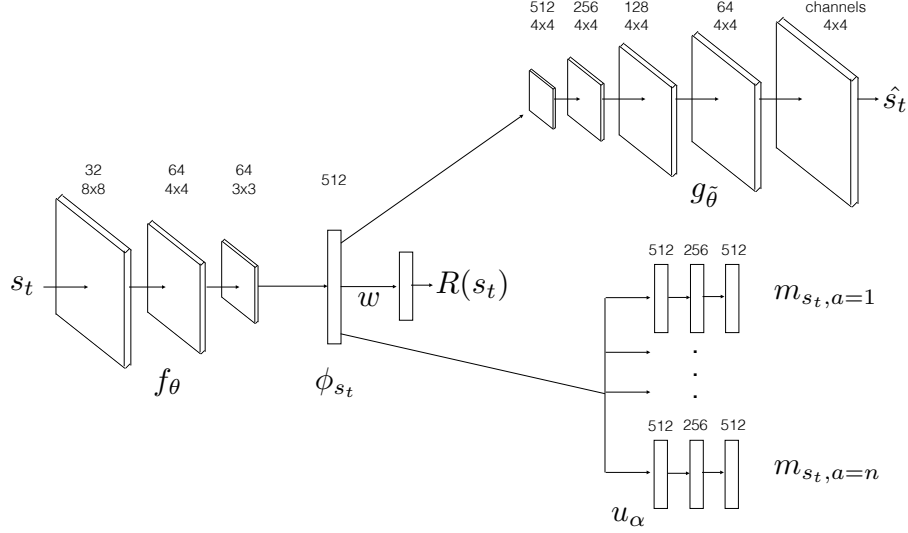


Figure 1: **Model Architecture:** DSR consists of: (1) feature branch f_θ (CNN) which takes in raw images and computes the features ϕ_{s_t} , (2) successor branch u_α which computes the SR $m_{s_t, a}$ for each possible action $a \in \mathcal{A}$, (3) a deep convolutional decoder which produces the input reconstruction \hat{s}_t and (4) a linear regressor to predict instantaneous rewards at s_t . The Q-value function can be estimated by taking the inner-product of the SR with reward weights: $Q^\pi(s, a) \approx m_{sa} \cdot \mathbf{w}$.

The DSR consists of two sub-components: (1) a reward feature learning component, constructed as a deep neural network, predicts intrinsic and extrinsic rewards to learn useful features from raw observations; and (2) an SR component, constructed as a separate deep neural network, that estimates the expected future “feature occupancy” conditioned on the current state and averaged over all actions. The value function can then be estimated as the dot product between these two factored representations. We train DSR by sampling experience trajectories (state, next-state, action and reward) from an experience replay memory and apply stochastic gradient descent to optimize model parameters. To avoid instability in the learning algorithm, we interleave training of the successor and reward components.

We show the efficacy of our approach on two different domains: (1) learning to solve goals in grid-world domains using the *MazeBase* game engine and (2) learning to navigate a 3D maze to gather a resource using the *Doom* game engine. We show the empirical convergence results on several policy learning problems as well as sensitivity of the value estimator given distal reward changes. We also demonstrate the possibility of extracting plausible subgoals for hierarchical RL by performing normalized-cuts on the SR [32].

2 Related work

The SR has been used in neuroscience as a model for describing different cognitive phenomena. [7] showed that the temporal context model [11], a model of episodic memory, is in fact estimating the SR using the temporal difference algorithm. [3] introduced a model based on SR for preplay and rapid path planning in the CA3 region of the hippocampus. They interpret the SR as an attractor network in a low-dimensional space and show that if the network is stimulated with a goal location it can generate a path to the goal. [35] suggested a model for tying the problems of navigation and reward maximization in the brain. They claimed that the brain’s spatial representations are designed to support the reward maximization problem (RL); they showed the behavior of the place cells and grid cells can be explained by finding the optimal spatial representation that can support RL. Based on their model they proposed a way for identifying reasonable subgoals from the spectral features of the SR. Other work (see for instance, [2, 4]) have also discussed utilizing the SR for subgoal and option discovery.

There are also models similar to the SR that have been applied to other RL-related domains. [31] introduced a model for evaluating the positions in the game of Go; the model is reminiscent of SR as it predicts the fate of every position of the board instead of the overall game score. Another reward-independent model, universal option model (UOM), proposed in [39], uses state occupancy function to build a general model of options. They proved that UOM of an option, given a reward function, can construct a traditional option model. There has also been a lot of work on option discovery in the tabular setting [14, 18–20, 34]. In more recent work, Machado et al. [17] presented an option discovery algorithm where the agent is encouraged to explore regions that were previously out of reach. However, option discovery where non-linear state approximations are required is still an open problem.

Our model is also related to the literature on value function approximation using deep neural networks. The deep-Q learning model [22] and its variants (e.g., [21, 24, 29, 33]) have been successful in learning Q-value functions from high-dimensional complex input states.

3 Model

3.1 Background

Consider an MDP with a set of states \mathcal{S} , set of actions \mathcal{A} , reward function $R : \mathcal{S} \rightarrow \mathbb{R}$, discount factor $\gamma \in [0, 1]$, and a transition distribution $T : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. Given a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, the Q-value function for selecting action a in state s is defined as the expected future discounted return:

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, a_0 = a \right], \quad (1)$$

where, s_t is the state visited at time t and the expectation is with respect to the policy and transition distribution. The agent’s goal is to find the optimal policy Q^* which follows the Bellman equation:

$$Q^*(s, a) = R(s) + \gamma \max_{a'} \mathbb{E} [Q(s', a')]. \quad (2)$$

3.2 The successor representation

The SR can be used for calculating the Q-value function as follows. Given a state s , action a and future states s' , SR is defined as the expected discounted future state occupancy:

$$M(s, s', a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[s_t = s'] | s_0 = s, a_0 = a \right],$$

where $\mathbb{1}[\cdot] = 1$ when its argument is true and zero otherwise. This implicitly captures the state visitation count. Similar to the Bellman equation for the Q-value function (Eq. 2), we can express the SR in a recursive form:

$$M(s, s', a) = \mathbb{1}[s = s'] + \gamma \mathbb{E}[M(s', s', a)]. \quad (3)$$

Given the SR, the Q-value for selecting action a in state s can be expressed as the inner product of the immediate reward and the SR [5]:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} M(s, s', a) R(s') \quad (4)$$

3.3 Deep successor representation

For large state spaces, representing and learning the SR can become intractable; hence, we appeal to non-linear function approximation. We represent each state s by a D -dimensional feature vector ϕ_s which is the output of a deep neural network $f_\theta : \mathcal{S} \rightarrow \mathbb{R}^D$ parameterized by θ .

For a feature vector ϕ_s , we define a feature-based SR as the expected future occupancy of the features and denote it by m_{sa} . We approximate m_{sa} by another deep neural network u_α parameterized by α : $m_{sa} \approx u_\alpha(\phi_s, a)$. We also approximate the immediate reward for state s as a linear function of the feature vector ϕ_s : $R(s) \approx \phi_s \cdot \mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^D$ is a weight vector. Since reward values

can be sparse, we can also train an intrinsic reward predictor $R_i(s) = g_{\tilde{\theta}}(\phi_s)$. A good intrinsic reward channel should give dense feedback signal and provide features that preserve latent factors of variations in the data (e.g. deep generative models that do reconstruction). Putting these two pieces together, the Q-value function can be approximated as (see 4 for closed form):

$$Q^\pi(s, a) \approx m_{sa} \cdot \mathbf{w}. \quad (5)$$

The SR for the optimal policy in the non-linear function approximation case can then be obtained from the following Bellman equation:

$$m_{sa} = \phi_s + \gamma \mathbb{E} [m_{s_{t+1}a'}] \quad (6)$$

where $a' = \operatorname{argmax}_a m_{s_{t+1}a} \cdot \mathbf{w}$.

3.4 Learning

The parameters $(\theta, \alpha, \mathbf{w}, \tilde{\theta})$ can be learned online through stochastic gradient descent.

The loss function for α is given by:

$$L_t^m(\alpha, \theta) = \mathbb{E}[(\phi(s_t) + \gamma u_{\alpha_{prev}}(\phi_{s_{t+1}}, a') - u_\alpha(\phi_{s_t}, a))^2],$$

where $a' = \operatorname{argmax}_a u_\alpha(\phi_{s_{t+1}}, a) \cdot \mathbf{w}$ and the parameter α_{prev} denotes a previously cached parameter value, set periodically to α . This is essential for stable Q-learning with function approximations (see [22]).

For learning \mathbf{w} , the weights for the reward approximation function, we use the following squared loss function:

$$L_t^r(\mathbf{w}, \theta) = (R(s_t) - \phi_{s_t} \cdot \mathbf{w})^2 \quad (7)$$

Parameter θ is used for obtaining the $\phi(s)$, the shared feature representation for both reward prediction and SR approximation. An ideal $\phi(s)$ should be: 1) a good predictor for the immediate reward for that state and 2) a good discriminator for the states. The first condition can be handled by minimizing loss function L_t^r ; however, we also need a loss function to help in the second condition. To this end, we use a deep convolutional auto-encoder to reconstruct images under an L2 loss function. This dense feedback signal can be interpreted as an intrinsic reward function. The loss function can be stated as:

$$L_t^a(\tilde{\theta}, \theta) = (g_{\tilde{\theta}}(\phi_{s_t}) - s_t)^2. \quad (8)$$

The composite loss function is the sum of the three loss functions given above:

$$L_t(\theta, \alpha, \mathbf{w}, \tilde{\theta}) = L_t^m(\alpha, \theta) + L_t^r(\mathbf{w}, \theta) + L_t^a(\tilde{\theta}, \theta) \quad (9)$$

For optimizing Eq. 9, with respect to the parameters $(\theta, \alpha, \mathbf{w}, \tilde{\theta})$, we iteratively update α and $(\theta, \mathbf{w}, \tilde{\theta})$. That is, we learn a feature representation by minimizing $L_t^r(\mathbf{w}) + L_t^a(\tilde{\theta})$; then given $(\theta^*, \mathbf{w}^*, \tilde{\theta}^*)$, we find the optimal α^* . This iteration is important to ensure that the successor branch does not back-propagate gradients to affect θ . We use experience replay memory \mathcal{D} of size $1e^6$ to store transitions, and apply stochastic gradient descent with a learning rate of $2.5e^{-4}$, momentum of 0.95, a discount factor of 0.99 and the exploration parameter ϵ annealed from 1 to 0.1 as training progresses. Algorithm 1 highlights the learning algorithm in greater detail.

4 Automatic Subgoal Extraction

Learning policies given sparse or delayed rewards is a significant challenge for current reinforcement learning algorithms. This is mainly due to inefficient exploration schemes such as ϵ -greedy. Existing methods like Boltzmann exploration and Thomson sampling [26, 36] offer significant improvements over ϵ -greedy, but are limited due to the underlying models functioning at the level of basic actions. Hierarchical reinforcement learning algorithms [1] such as the *options* framework [38, 39] provide a flexible framework to create temporal abstractions, which will enable exploration at different time-scales. The agent will learn options to reach the subgoals which can be used for intrinsic motivation. In the context of hierarchical RL, [8] discuss a framework for subgoal extraction using the structural aspects of a learned policy model. Inspired by previous work in subgoal discovery from

Algorithm 1 Learning algorithm for DSR

```
1: Initialize experience replay memory  $\mathcal{D}$ , parameters  $\{\theta, \alpha, \mathbf{w}, \tilde{\theta}\}$  and exploration probability  $\epsilon = 1$ .
2: for  $i = 1 : \#episodes$  do
3:   Initialize game and get start state description  $s$ 
4:   while not terminal do
5:      $\phi_s = f_\theta(s)$ 
6:     With probability  $\epsilon$ , sample a random action  $a$ , otherwise choose  $\operatorname{argmax}_a u_\alpha(\phi_s, a) \cdot \mathbf{w}$ 
7:     Execute  $a$  and obtain next state  $s'$  and reward  $R(s')$  from environment
8:     Store transition  $(s, a, R(s'), s')$  in  $\mathcal{D}$ 
9:     Randomly sample mini-batches from  $\mathcal{D}$ 
10:    Perform gradient descent on the loss  $L^r(\mathbf{w}, \theta) + L^a(\tilde{\theta}, \theta)$  with respect to  $\mathbf{w}$ ,  $\theta$  and  $\tilde{\theta}$ .
11:    Fix  $(\theta, \tilde{\theta}, \mathbf{w})$  and perform gradient descent on  $L^m(\alpha, \theta)$  with respect to  $\alpha$ .
12:     $s \leftarrow s'$ 
13:   end while
14:   Anneal exploration variable  $\epsilon$ 
15: end for
```

state trajectories [34] and the tabular SR [35], we use the learned SR to generate plausible subgoal candidates.

Given a random policy π_r ($\epsilon = 1$), we train the DSR until convergence and collect the SR for a large number of states $\mathcal{T} = \{m_{s_1, a_1}, m_{s_2, a_2}, \dots, m_{s_n, a_n}\}$. Following [32, 34], we generate an affinity matrix W given \mathcal{T} , by applying a radial basis function (with Euclidean distance metric) for each pairwise entry $(m_{s_i, a_i}, m_{s_j, a_j})$ in \mathcal{T} (to generate w_{ij}). Let D be a diagonal matrix with $D(i, i) = \sum_j w_{ij}$. Then as per [32], the second largest eigenvalue of the matrix $D^{-1}(D - W)$ gives an approximation of the minimum normalized cut value of the partition of \mathcal{T} . The states that lie on the end-points of the cut are plausible subgoal candidates, as they provide a path between a community of state groups. Given randomly sampled \mathcal{T} from π_r , we can collect statistics of how many times a particular state lies along the cut. We pick the top- k states as the subgoals. Our experiments indicate that it is possible to extract useful subgoals from the DSR.

5 Experiments

In this section, we demonstrate the properties of our approach on MazeBase [37], a grid-world environment, and the Doom game engine [13]. In both environments, observations are presented as raw pixels to the agent. In the first experiment we show that our approach is comparable to DQN in two goal-reaching tasks. Next, we investigate the effect of modifying the distal reward on the initial Q-value. Finally, using normalized-cuts, we identify subgoals given the successor representations in the two environments.

5.1 Goal-directed Behavior

Solving a maze in MazeBase We learn the optimal policy in the maze shown in Figure 2 using the DSR and compare its performance to the DQN [22]. The cost of living or moving over water blocks is -0.5 and the reward value is 1. For this experiment, we set the discount rate to 0.99 and the learning rate to $2.5 \cdot 10^{-4}$. We anneal the ϵ from 1 to 0.1 over 20k steps; furthermore, for training the reward branch, we anneal the number of samples that we use, from 4000 to 1 by a factor of 0.5 after each training episode. For all experiments, we prioritize the reward training by keeping a database of non-zero rewards and sampling randomly from the replay buffer with a 0.8 probability and 0.2 from the database. Figure 3 shows the average trajectory (over 5 runs) of the rewards obtained over 100k episodes. As the plot suggests, DSR performs on par with DQN.

Finding a goal in a 3D environment We created a map with 4 rooms using the ViZDoom platform [13]. The map is shown in Figure 2. We share the same network architecture as in the case of MazeBase. The agent is spawned inside a room, and can explore any of the other three rooms. The agent gets a per-step penalty of -0.01 and a positive reward of 1.0 after collecting an item from one

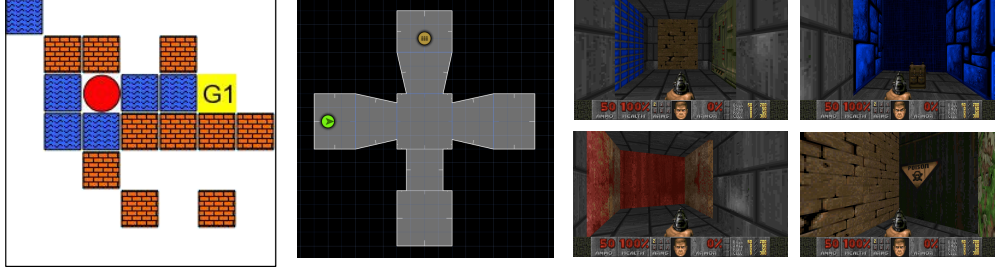


Figure 2: **Environments:** (left) MazeBase [37] map where the agent starts at an arbitrary location and needs to get to the goal state. The agent gets a penalty of -0.5 per-step, -1 to step on the water-block (blue) and $+1$ for reaching the goal state. The model observes raw pixel images during learning. (center) A *Doom* map using the VizDoom engine [13] where the agent starts in a room and has to get to another room to collect ammo (per-step penalty = -0.01 , reward for reaching goal = $+1$). (right) Sample screen-shots of the agent exploring the 3D maze.

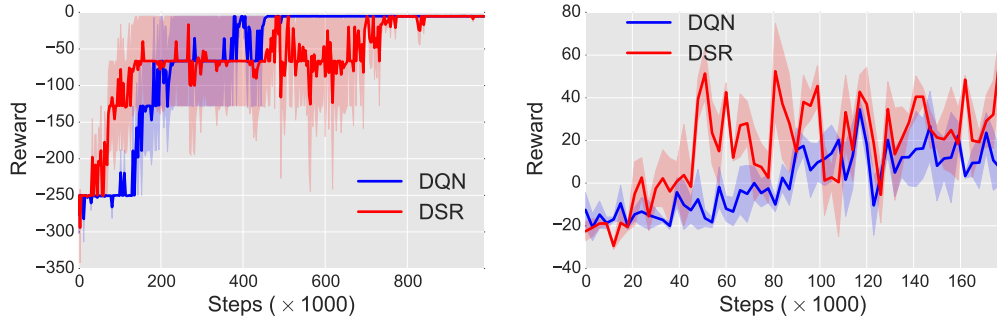


Figure 3: Average trajectory of the reward (left) over 100k steps for the grid-world maze. (right) over 180k steps for the Doom map over multiple runs.

of the room (highlighted in *red* in Figure2). As shown in Figure3, the agent is able to successfully navigate the environment to obtain the reward, and is competitive with DQN.

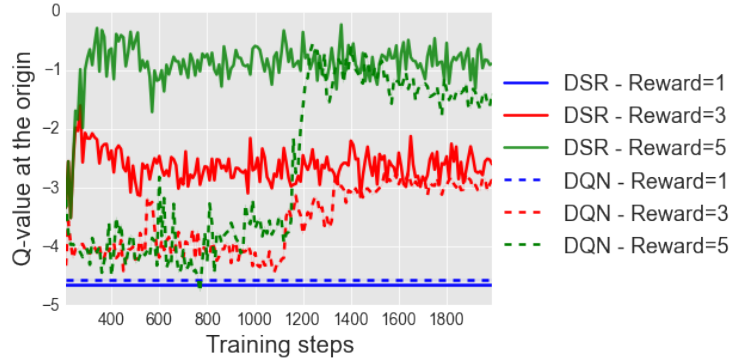


Figure 4: **Changing the value of the distal reward:** We train the model to learn the optimal policy on the *maze* shown in Figure 2. After convergence, we change the value of the distal reward and update the Q-value for the optimal action at the origin (bottom-left corner of the maze). In order for the value function to converge again, the model only needs to update the linear weights w given the new external rewards.

5.2 Value function sensitivity to distal reward changes

The decomposition of value function into SR and immediate reward prediction allows DSR to rapidly adapt to changes in the reward function. In order to probe this, we performed experiments to measure

the adaptability of the value function to distal reward changes. Given the grid-world map in Figure 2, we can train the agent to solve the goal specified in the map as highlighted in section 5.1. Without changing the goal location, we can change the reward scalar value upon reaching the goal from 1.0 to 3.0. Our hypothesis is that due to the SR-based value decomposition, our value estimate will converge to this change by just updating the reward weights w (SR remains same). As shown in Figure 4, we confirm that the DSR is able to quickly adapt to the new value function by just updating w .

5.3 Extracting subgoals from the DSR

Following section 4, we can also extract subgoals from the SR. We collect \mathcal{T} by running a random policy on both MazeBase and VizDoom. During learning, we only update SR (u_α) and the reconstruction branch ($g_{\hat{\theta}}$), as the immediate reward at any state is zero (due to random policy).

As shown in Figures 5 and 6, our subgoal extraction scheme is able to capture useful subgoals and clusters the environment into reasonable segments. Such a scheme can be ran periodically within a hierarchical reinforcement learning framework to aid exploration. One inherent limitation of this approach is that due to the random policy, the subgoal candidates are often quite noisy. Future work should address this limitation and provide statistically robust ways to extract plausible candidates. Additionally, the subgoal extraction algorithm should be non-parametric to handle flexible number of subgoals.

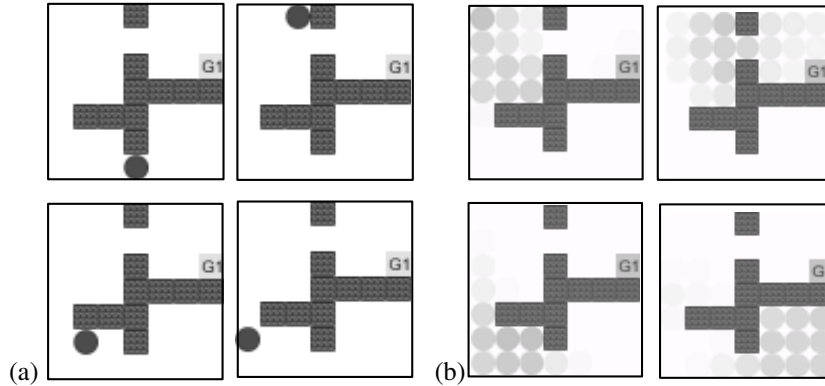


Figure 5: **Subgoal extraction on grid-world:** Given a random policy, we train DSR until convergence and collect a large number of sample transitions and their corresponding successor representations as described in section 4. We apply a normalized cut-based algorithm on the SRs to obtain a partition of the environment as well as the bottleneck states (which correspond to goals) (a) Subgoals are states which separate different partitions of the environments under the normalized-cut algorithm. Our approach is able to find reasonable subgoal candidates. (b) Partitions of the environment reflect latent structure in the environment.

6 Conclusion

We presented the DSR, a novel deep reinforcement learning framework to learn goal-directed behavior given raw sensory observations. The DSR estimates the value function by taking the inner product between the SR and immediate reward predictions. This factorization of the value function gives rise to several appealing properties over existing deep reinforcement learning methods—namely increased sensitivity of the value function to distal reward changes and the possibility of extracting subgoals from the SR under a random policy.

For future work, we plan to combine the DSR with hierarchical reinforcement learning. Learning goal-directed behavior with sparse rewards is a fundamental challenge for existing reinforcement learning algorithms. The DSR can enable efficient exploration by periodically extracting subgoals, learning policies to satisfy these intrinsic goals (skills), and subsequently learning hierarchical policy over these subgoals in an options framework [15, 28, 39]. One of the major issues with the DSR is learning discriminative features. In order to scale up our approach to more expressive environments, it will be crucial to combine various deep generative and self-supervised models [6, 9, 10, 12, 16, 25, 27, 40]

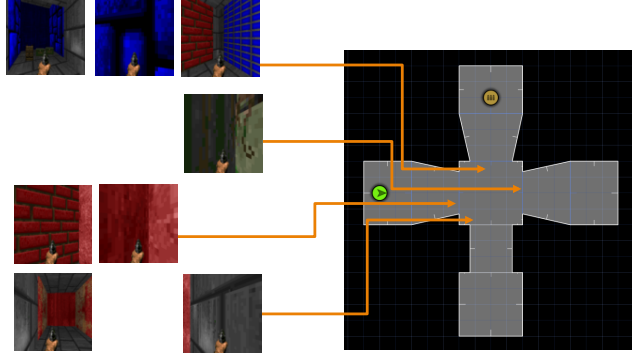


Figure 6: **Subgoal extraction on the Doom map** The subgoals are extracted using the **normalized cut-based algorithm on the SR**. The SR samples are collected based on a random policy. The subgoals mostly correspond to the rooms’ entrances in the common area between the rooms. Due to random policy, we sometimes observe high variance in the subgoal quality. Future work should address robust statistical techniques to obtain subgoals, as well as non-parametric approaches to obtaining flexible number of subgoals.

with our approach. In addition to subgoals, using DSR for extracting other intrinsic motivation measures such as improvements to the predictive world model [30] or mutual information [23] is worth pursuing.

References

- [1] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.
- [2] Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1655):20130480, 2014.
- [3] Dane S Corneil and Wolfram Gerstner. Attractor network dynamics enable preplay and rapid path planning in maze-like environments. In *Advances in Neural Information Processing Systems*, pages 1675–1683, 2015.
- [4] Nathaniel D Daw and Peter Dayan. The algorithmic anatomy of model-based evaluation. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1655):20130478, 2014.
- [5] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.
- [6] SM Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.
- [7] Samuel J Gershman, Christopher D Moore, Michael T Todd, Kenneth A Norman, and Per B Sederberg. The successor representation and temporal context. *Neural Computation*, 24(6):1553–1568, 2012.
- [8] Sandeep Goel and Manfred Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *FLAIRS conference*, pages 346–350, 2003.
- [9] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Binding via reconstruction clustering. *arXiv preprint arXiv:1511.06418*, 2015.
- [10] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.
- [11] Marc W Howard and Michael J Kahana. A distributed representation of temporal context. *Journal of Mathematical Psychology*, 46(3):269–299, 2002.
- [12] Jonathan Huang and Kevin Murphy. Efficient inference in occlusion-aware generative models of images. *arXiv preprint arXiv:1511.06362*, 2015.

- [13] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.
- [14] George Konidaris and Andre S Barreto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, pages 1015–1023, 2009.
- [15] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.
- [16] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2530–2538, 2015.
- [17] Marlos C Machado and Michael Bowling. Learning purposeful behaviour in the absence of rewards. *arXiv preprint arXiv:1605.07700*, 2016.
- [18] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.
- [19] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *Computer Science Department Faculty Publication Series*, page 8, 2001.
- [20] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002*, pages 295–306. Springer, 2002.
- [21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.
- [22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [23] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2116–2124, 2015.
- [24] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.
- [25] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *arXiv preprint arXiv:1603.09246*, 2016.
- [26] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.
- [27] Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. *arXiv preprint arXiv:1603.05106*, 2016.
- [28] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1312–1320, 2015.
- [29] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- [30] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *Autonomous Mental Development, IEEE Transactions on*, 2(3):230–247, 2010.
- [31] Nicol N Schraudolph, Peter Dayan, and Terrence J Sejnowski. Temporal difference learning of position evaluation in the game of go. *Advances in Neural Information Processing Systems*, pages 817–817, 1994.
- [32] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

- [33] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [34] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pages 816–823. ACM, 2005.
- [35] Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. Design principles of the hippocampal cognitive map. In *Advances in neural information processing systems*, pages 2528–2536, 2014.
- [36] Bradley C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.
- [37] Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.
- [38] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.
- [39] Csaba Szepesvari, Richard S Sutton, Joseph Modayil, Shalabh Bhatnagar, et al. Universal option models. In *Advances in Neural Information Processing Systems*, pages 990–998, 2014.
- [40] William F Whitney, Michael Chang, Tejas Kulkarni, and Joshua B Tenenbaum. Understanding visual concepts with continuation learning. *arXiv preprint arXiv:1602.06822*, 2016.