Deep Successor Reinforcement Learning For Maze Navigation

Wenchao Wang
ECE, UMass
wenchaowang@umass.edu

Nitin Kishore Sai Samala
CS, UMass
nsamala@cs.umass.edu

Xingjian Chen
ECE, UMass
xingjian@umass.edu

ABSTRACT

Successor Representations have been used in the field of neuroscience as an attractor network in a low–dimensional space and it has been observed that if the network is stimulated with a goal location it can generate a path to the goal. The brain's spatial representations are designed to support the reward maximization problem (RL); DSR generalizes Successor Representations (SR) within an end-to-end deep reinforcement learning framework. In our project, we explored Deep Successor Representations (DSR) and compared it with a traditional method, which is deep Q-network (DQN). SR is an alternative to deep reinforcement learning algorithms apart from model-free and model-based algorithms. The value function is decomposed into two components in SR -- a reward predictor mapping states to scalar rewards and a successor map representing the expected future state occupancy from any given state and the reward predictor. 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 analysed the efficacy of this new approach on a grid-world domain of MazeBase.

Keywords

Deep Successor Representations; Reinforcement learning; Successor Representations; Reward predictor; Successor map; Subgoals;

1. INTRODUCTION

In reinforcement learning (RL), the end game is to find an optimal policy that maximizes expected future discounted rewards (value). There are two main classes in RL methods: model-free algorithms, that learn cached value functions directly from sample trajectories and model-based algorithms that estimate transition and reward functions, from which values can be computed using tree-search or dynamic programming. Recently, a novel method based on successor representations (SR), that factors the value function into a predictive representation and a reward function, was put forward. As the name suggests, in this representation scheme each state is described by a prediction about the future occurrence of all other states under a fixed policy. The value function at a given state is the dot product between the vector of expected discounted future state occupancies and the immediate reward in each of those successor states. This alternative appraoch has several advantages. (1) It combines computational efficiency comparable to model-free algorithms with some of the flexibility of model-based algorithms so that it can adapt quickly to changes in distal reward. (2) It has the ability to extract bottleneck states (candidate subgoals) from the successor representation under a random policy [16]. In our project we used a powerful function approximation algorithm and architecture for the SR using a deep neural network, which is Deep Successor Reinforcement Learning (DSR).

There are two components in DSR architecture: (1) A reward feature learning component, constructed as a deep neural network, predicts intrinsic and extrinsic rewards to learn useful features from raw observations; (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 inner product of these two factored representations.

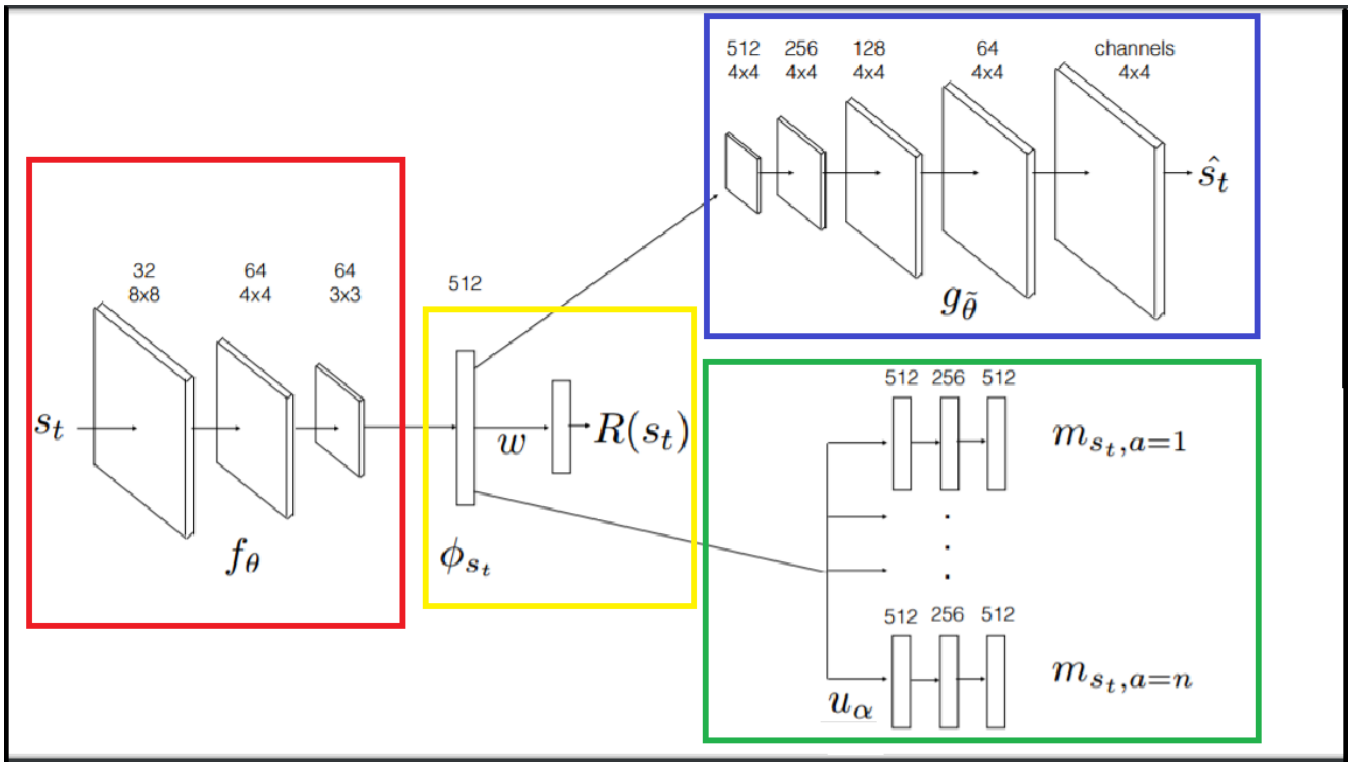
In our project, we devised many experimental runs and got the results: (1) For different games, efficiency of DSR approach is similar to DQN algorithm. (2) For the same game, the evaluated steps have an effect on the converging rate in DSR. (3) It is possible to extract plausible subgoals for hierarchical RL by performing normalized-cuts on the SR [15].

2. RELATED WORK

When it comes to successor representation, a lot of effort in previous research has been given. It is used in neuroscience as a model for describing different cognitive phenomena. [6] showed that the temporal context model [9], 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 demonstrated that 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 works (see for instance, [3, 5]) have also discussed utilizing the SR for subgoals and option discovery.

Models similar to the SR that have been been applied to other RL-related domains.[18] 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 [19], uses state occupancy function to build a general model of options.

3. MODEL ARCHITECTURE



DSR consists of: **(1)** feature branch f_θ(CNN) which takes in raw images and computes the features $\emptyset_{s_t}$, **(2)** successor branch u_α which computes the SR $m_{s_t,a}$ for each possible action $a \in A$, **(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: $Q^\pi(s,a) \approx m_{sa}.w$

3.1 Background

Consider an MDP with a set of states S, set of actions A, reward function $R: S \rightarrow Real\ numbers$, discount factor γ, and a transition distribution $T: SxA \rightarrow [0,1]$, the Q-value function for selecting action a in state s is defined as the expected future discounted return. The agent's goal is to find the optimal policy Q* which follows the Bellman equation.

$$Q^*(s,a) = R(s_t) + \gamma \max_{a'} E(Q(s_{t+1}, a'))$$

3.2 The Successor representation

SR is defined as the expected discounted future state occupancy:

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

where $\mathbf{1}[:] = 1$ when its argument is true and zero otherwise. This implicitly captures the state visitation count. We can express the SR in a recursive form as:

$$M(s, s', a) = \mathbf{1}[s_t = s'] + \gamma E[M(s_{t+1}, s', a_{t+1})]$$

The Q-value for selecting action a in state s can be expressed as the inner product of the immediate reward and the SR:

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

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 $f_\theta: S \rightarrow R^D$ parameterized by θ.
s

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: $m_{sa} \approx u_\alpha(\varphi_s, a)$. We also approximate the immediate reward for state s as a linear function of the feature vector φ_s: $R(s) \approx \varphi_s \cdot w$, where $w \in R^R$ is a weight vector. Since reward values can be sparse, we can also train an intrinsic reward predictor $R_i(s) = g_\theta(\varphi_s:)$. A good intrinsic reward channel should give dense feedback signal and provide features that preserve latent factors of variations in the. Putting these two pieces together, the Q-value function can be approximated as:

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

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

$$m_{sa} = \varphi_s + \gamma E\left[m_{s_{t+1}a'}\right]$$

where $a' = \text{argmax}_a(m_{s_{t+1}a}) \cdot \mathbf{w}$

3.4 Learning

The parameters $(\theta, \alpha, w, \hat{\boldsymbol{\theta}})$ can be learned online through stochastic gradient descent. The loss function for α is given by:

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

where $a' = \text{argmax}_a u_\alpha(\varphi_{s_{t+1}}, a) \cdot \mathbf{w}$ 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.

For learning w, the weights for the reward approximation function

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

Parameter θ is used for obtaining the φ(s), the shared feature representation for both reward prediction and SR approximation. An ideal φ(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:

$$L_t^a(\hat{\theta}, \theta) = (g_{\hat{\theta}}(\varphi_{s_t}) - s_t)^2$$

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

$$L_t(\theta, \alpha, w, \hat{\theta}) = L_t^m(\alpha, \theta) + L_t^r(w, \theta) + L_t^a(\hat{\theta}, \theta)$$

For optimizing the last equation, with respect to the parameters $(\theta, \alpha, w, \hat{\theta})$, we iteratively update α and $(\theta, w, \hat{\theta})$. That is, we learn a feature representation by minimizing$L_t^r(w, \theta) + L_t^a(\hat{\theta}, \theta)$; then given $(\theta^*, w^*, \hat{\theta}^*)$, 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 D of size 1e6 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 Learning algorithm for DSR

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

4. EXPERIMENTS

To demonstrate the properties of DSR on mazes we performed the following experiments on Mazebase, which is a grid-world environment. The observations are presented in the form of raw pixels to the agent. These were performed using Lua+Torch as it offers rapid prototyping of the game and is easy to connect to models that control the agent's behavior.

4.1 Environment

The Each game is played in a 2D rectangular grid. Each location in the grid can be empty, or may contain one or more items such as:

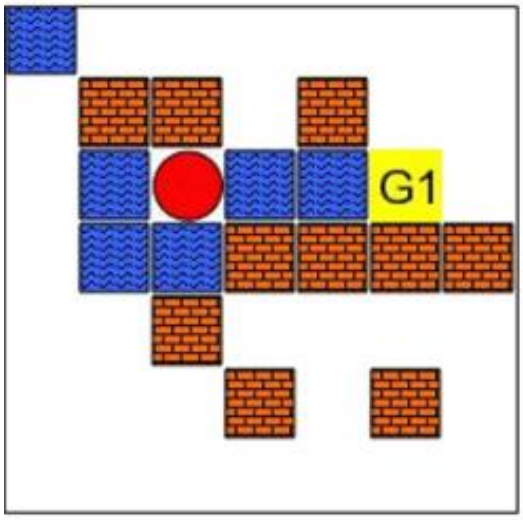


Figure 0. MazeBase Game Environment

- **Block:** an impassible obstacle that does not allow the agent to move to that grid location
- **Water:** the agent may move to a grid location with water, but incurs an additional cost of for doing so.
- **Switch:** a switch can be in one of M states, which we refer to as colors. The agent can toggle through the states cyclically by a toggle action when it is at the location of the switch .

- **Door:** a door has a color, matched to a particular switch. The agent may only move to the door's grid location if the state of the switch matches the state of the door.
- **Pushable-Block:** This block is impassable, but can be moved with a separate "push" actions. The block moves in the direction of the push, and the agent must be located adjacent to the block opposite the direction of the push.
- **Corner:** This item simply marks a corner of the board.
- **Goal:** depending on the game, one or more goals may exist, each named individually.
- **Info:** these items do not have a grid location, but can specify a or give information necessary for its completion.

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.

The environment presented to the agent as a raw pixel observations, describes an item in the game. The environments are generated randomly with some distribution on the various items. For example, we usually specify a uniform distribution over height and width, and a percentage of wall blocks and water blocks. Four the purpose of our project we limited the environment to only blocks and a goal and split tasks to four levels of difficulty:

1. Moving Goals,
2. Moving Goals Easy,
3. Moving Goals Medium,
4. Moving Goals Hard

4.2 Goal-directed behavior

We verified how our approach compares to DQN in four goal reaching tasks. Average trajectory of the reward over 100k steps for the grid-world maze in normal and easy tasks and 200k for the medium and hard tasks.

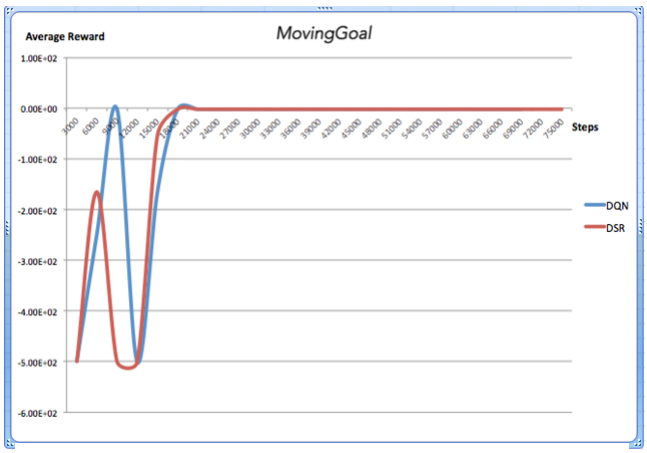


Figure 1. Average reward on game MovingGoal

Figure 2. Average reward on game MovingGoal Easy

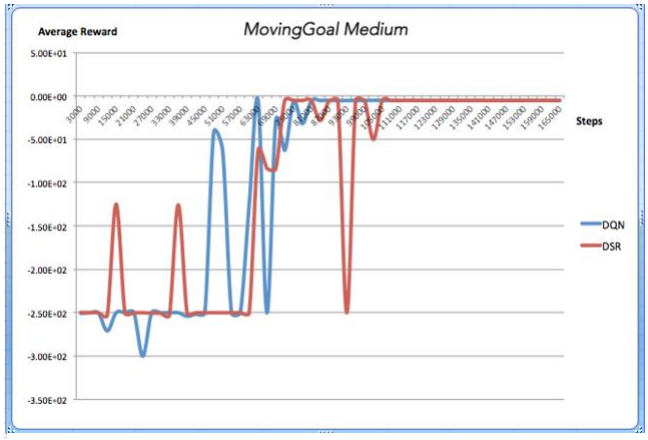


Figure 3. Average reward on game MovingGoal Medium

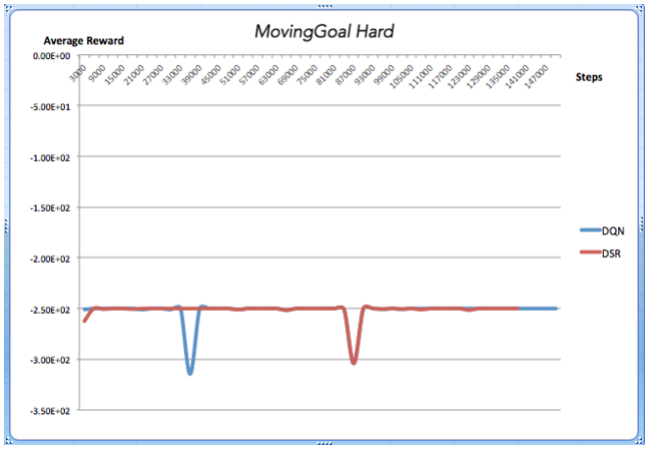


Figure 4. Average reward on game MovingGoal Hard

As the plots suggest, DSR performs on par with DQN. The agent is able to successfully navigate the environment to obtain the reward, and is competitive with DQN.

4.3 Sensitivity to evaluate steps changes

In order to explore the relationship between evaluated steps and convergence rate, we performed experiments to measure the adaptability of the value function to evaluated steps changes. Given the grid-world map, we can train the agent to solve the goal specified in the map. We tried evaluate step from 500 to 1500

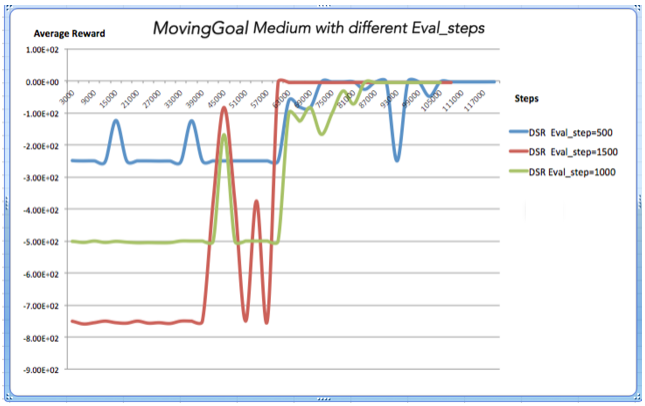


Figure 5. Average reward on evaluate steps changes

As shown in the plot, when we have lower evaluate steps, the initial reward tend to be lower, however, it converges more quickly.

4.4 Sensitivity to distal reward changes

We investigate the effect of modifications of distal reward on the Q-value function. 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, we can train the agent to solve the goal specified in the map. 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).

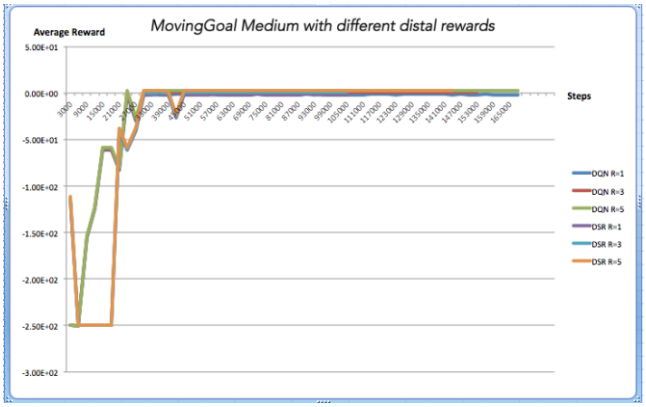


Figure 6. Average reward on distal reward changes

As shown, we confirm that the DSR is able to quickly adapt to the new value function by just updating w.

4.5 Extracting subgoals from DSR

Subgoals are states which separate different partitions of the environments under the normalized-cut algorithm. One inherent limitation of this approach is that due to the random policy, the subgoal candidates are often quite noisy.

Given a random policy, we train DSR until convergence and collect a large number of sample transitions and their corresponding successor representations.Applying a normalized cut-based algorithm on the SRs we obtained a partition of the environment as well as the bottleneck states (which correspond to goals). Additionally, this subgoal extraction algorithm is non-parametric and can handle flexible number of subgoals.

Given the successor representations in this environment, we obtained the following

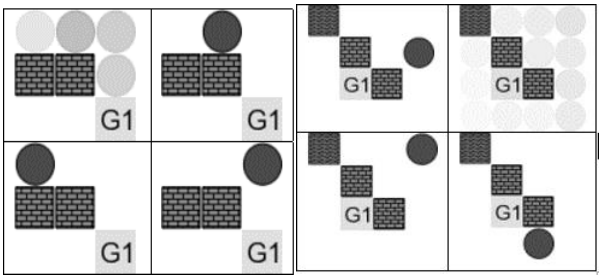


Figure 6. Subgoals extracted from DSR

Our subgoal extraction scheme is able to capture useful subgoals and clusters the environment into reasonable segments. Such a scheme can be run periodically within a hierarchical reinforcement learning framework to aid exploration.

5. CONCLUSION

We used 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.

Here are some conclusions we got from our experiments:

1) DSR achieves results competitive with DQN in four different level games
2) DSR converges quickly with higher evaluate step even with lower initial reward.
3) DSR has a higher sensitivity of the value function to distal reward changes
4) It is possible to extracting subgoals from the SR under a random policy.

For future work, first we would like to implement DSR in a more complex environment like Doom to get a better result for changing distal reward. Second, we plan to combine the DSR with hierarchical reinforcement learning. One of the major issues with the DSR is learning discriminative features. In addition to subgoals, using DSR for extracting other intrinsic motivation measures such as improvements to the predictive world model [13] or mutual information [12] is worth pursuing.

6. REFERENCES

[1] Kulkarni, T. D., Saeedi, A., Gautam, S., & Gershman, S. J. (2016). Deep Successor Reinforcement Learning. *arXiv preprint arXiv:1606.02396*.

[2] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003..

[3] Dane S Corneil and Wulfram 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] Sandeep Goel and Manfred Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In FLAIRS conference, pages 346–350, 2003.

[7] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Binding via reconstruction clustering. *arXiv preprint arXiv:1511.06418*, 2015.

[8] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[9] Marc W Howard and Michael J Kahana. A distributed representation of temporal context. *Journal of Mathematical Psychology*, 46(3):269–299, 2002.

[10] Marlos C Machado and Michael Bowling. Learning purposeful behaviour in the absence of rewards. *arXiv preprint arXiv:1605.07700*, 2016.

[11] 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.

[12] 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.

[13] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *Autonomous Mental Development, IEEE Transactions on*, 2(3):230–247, 2010.

[14] 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.

[15] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[16] Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. Design principles of the hip- pocampal cognitive map. In *Advances in neural information processing systems*, pages 2528–2536, 2014.

[17] Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

[18] 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

[19] 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.