[Re] Better transfer learning with inferred successor maps

Matthew Slivinski[1], **Alana Jaskir**[2], **Aaron Traylor**[1]
[1]Department of Computer Science
[2]Department of Cognitive, Linguistic and Psychological Sciences
Brown University
Providence, RI 02842
`{matthew_slivinski,alana_jaskir,aaron_traylor}@brown.edu`

Abstract

We reproduced findings of the original paper that proposed an algorithm for multitask learning by combining the successor representation with task evaluation through non-parametric inference over various domains.

1 Introduction

Madarasz (2019) [7] introduces a framework that focuses on combining generalized abstractions and transfer learning into reinforcement learning (RL) agents through successor representations (SR) with memory-based approaches. They propose to formulate this learning system based on neurobiologically grounded phenomena that parallel predictive representations, which have been found in the amygdala and hippocampus [3, 6], along with observations clustering of experiences into latent features [5]. Their work aims to balance these core ideas into compressed, SR maps that efficiently process experiences within sequential decision making.

The proposed algorithm, Bayesian Successor Representation (BSR), was tested on several reward changing domains where they showed that BSR outperforms previous methods such as single SR (SSR), GPI (SFQL) [2], and an agent that knew the quadrant of the goal within the domain (KQ). In this work[1], we implemented SSR and BSR agents in tabular grid-world navigation tasks to reproduce Fig. 2a and 2c from Madarasz (2019) [7] in Python (v3.7.5) with the support of simple_rl [1].

2 Methods

2.1 Single Successor Representation

Model A Markov Decision Process (MDP) can be formally represented as a tuple (S, A, R, T, γ), where S is a set of states, A is the action space, R is the reward function, T is the transition probabilities that describes state transition dynamics, and $\gamma \in (0, 1]$ is the discount factor [8]. For the specific transfer learning problem, where the reward function changes, we consider MDPs where the reward locations are determined by some stochastic process. In the tabular setting, the proposed SSR framework [4] computes SR maps M of expected discount sums over future state occupancies, where the rewards are linear in the state representation $\phi(s)$ of state $s \in S$, given a policy π at discrete time steps t (Eq. 1):

$$M_t(s, a, \cdot) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \cdot \phi\left(s_{t+k+1}\right) \middle| s_t = s, a_t = a\right] \tag{1}$$

[1]Code can be found in `src` at `https://github.com/attraylor/reproducing_bsr`

Given the current reward weights $\boldsymbol{w}$ that satisfy $\phi\left(s'\right)$ for next state $s' \in S$ then the action-value function $Q(s,a)$ for current action $a \in A(s)$ is represented as (Eq. 2):

$$Q_t(s,a) = \sum_{s'} M_t\left(s,a,s'\right) \cdot \boldsymbol{w}_{s'} \tag{2}$$

Furthermore, the following Bellman updates can be applied (Eq. 3,4):

$$M_{t+1}\left(s_t,a_t,\cdot\right) \leftarrow M_t\left(s_t,a_t,\cdot\right) + \alpha\left(\phi\left(s_{t+1}\right) + \gamma \cdot M_t\left(s_{t+1},a^*_{t+1},\cdot\right) - M_t\left(s_t,a_t,\cdot\right)\right) \tag{3}$$

$$a^*_{t+1} = \max_{a_{t+1}}\left[M_t\left(s_{t+1},a_{t+1},\cdot\right) \cdot \boldsymbol{w}\right] \tag{4}$$

Implementation As SSR is one component of the more complex BSR algorithm, we first ensured its reproduciblility. This code then served as a skeleton for implementing the BSR algorithm to isolate any difficulty in BSR's reproduction from issues in the SSR. Using the simple_rl framework, we created a new successor map agent (`agents/SuccessorMapAgentClass.py`), which tracked both the discounted visits of future states given a state-action as described (Eq. 1) and updated (Eq. 3,4) in the paper. The agent also tracked the immediate reward received at each state, represented by the vector $\boldsymbol{w}$. For each experience transition and reward received from its actions, $< s,a,r,s' >$, the agent updated $\boldsymbol{w}$ as follows (Eq. 5):

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}}\left[r - \phi(s')^T\boldsymbol{w}\right]\phi(s') \tag{5}$$

More simply expressed, any reward only updated the state reward weight that the agent transitioned into (Eq. 6):

$$\boldsymbol{w}_{s'} \leftarrow \boldsymbol{w}_{s'} + \alpha_{\boldsymbol{w}}\left(r - \boldsymbol{w}_{s'}\right) \tag{6}$$

Notably, as reward was deterministic (though reward location was not), the author's use of $\alpha_{\boldsymbol{w}} = 1$ for both SSR and BSR, upon reflection, is sensible. There is no need for the agent to average $\boldsymbol{w}$ over long horizons as it is not learning average future reward (as in more classic TD frameworks), but *immediate* reward experienced at a state. However, it does restrict the claims of the author to a deterministic reward feedback environments, as this assumption is baked into the parameter selection.

One difficulty encountered when reproducing SSR behavior was handling the corner case when the action selected lead to a terminal, or goal, state. As Eq. 3 shows, the Bellman update of the successor maps uses future discounted maps. To account for this, we assumed this term was dropped for terminal states. Below, we assume $T(s_t,a_t,s_{goal}) = 1$.

$$M_{t+1}\left(s_t,a_t,\cdot\right) \leftarrow M_t\left(s_t,a_t,\cdot\right) + \alpha\left(\phi\left(s_{t+1}\right) - M_t\left(s_t,a_t,\cdot\right)\right) \tag{7}$$

2.2 Bayesian Successor Representation

Model The proposed model uses multiple SR maps where an agent maintains a belief distribution of weighted values ω over these maps and samples an SR map at every time step with respect to ω. To transfer policies between environments where similar rewards are close in proximity, BSR evaluates reward similarity using normalized, average convolved reward (CR) values $\boldsymbol{v}^{cr}\left(s_{t-f}\right)$. For each SR map, a reward kernel is used to calculate $\boldsymbol{v}^{cr}$, delayed by window f to start the kernel from time t, the curren time step. The CR values are defined in the original text as the current rewards $\boldsymbol{r}_{delay}(s) = [r_{t-f},\ldots,r_{t+f}]$, delayed by f at t, by the exponential discounted factors $\boldsymbol{K}_\gamma = [\gamma^{-f},\gamma^{-f-1},\ldots,1\ldots,\gamma^{f-1},\gamma^f]^T$, all normalized (Eq. 8).

$$\boldsymbol{v}^{cr}\left(s_{t-f}\right) = \frac{\boldsymbol{r}_{delay} \cdot \boldsymbol{K}_\gamma}{\sum \boldsymbol{K}_\gamma} \tag{8}$$

Inference These CR values are utilized to perform inference over a distrubution G that is sampled from a non-parametric Dirchlet process (DP) mixture model where $\alpha \in \mathbb{R}^+$ and H is a base mixing distribution (Eq. 9).

$$G \sim \mathrm{Dir}(\alpha, H) \tag{9}$$

Each CR map is represented by the weights $\boldsymbol{w}_t^{CR}$ sampled from G (Eq. 10).

$$\boldsymbol{w}_t^{cr} \sim G \tag{10}$$

These mixture components determine the likelihoods of the CR values denoted as $\boldsymbol{L}^{cr}(s_t)$ using each CR map from a Gaussian distribution of linear basis $\phi(s)$ (Eq. 11).

$$\boldsymbol{L}^{cr}(s_t) \sim \mathcal{N}\left(\phi(s) \cdot \boldsymbol{w}_t^{cr}, \sigma_{cr}^2\right) \tag{11}$$

A Sequential Monte Carlo (SMC) method is used, namely particle filtering, to handle this multi-task RL setup with dynamically changing reward locations. The particle filer samples from a Chinese Restaurant Process (CRP) of each particle for the i-th partition $\boldsymbol{c}^i = \left[c_1^i, \ldots, c_t^i\right]$ where m_k is the number of observations assigned to cluster k (Eq. 12):

$$\Pr\left(c_t^i = k \middle| \boldsymbol{c}_{1:t-1}^i\right) = \begin{cases} \frac{\alpha}{t-1+\alpha} & \text{if } k \text{ is a new cluster} \\ \frac{m_k}{t-1+\alpha} & \text{otherwise} \end{cases} \tag{12}$$

The function of the particle filter is to tractably infer the appropriate context (i.e. successor map and weights), given previous cluster proposals for the episode. The particle filter process allows the agent to transfer learning to new goal locations by inferring its similarity to previously seen contexts, and according to popularity of the context. It also allows the agent to account for uncertainty in its cluster assignment earlier in the episode.

Implementation To implement the BSR algorithm, we extended the functioning SSR to track multiple successor maps (`agents/BayesianSRAgentClass.py`). The map used for action selection was determined probabilistically according to weights, ω.

The main flow of our reproduced code went as follows:

Algorithm 1 BSR Logic Flow (per episode)

```
 1: procedure BSR($\gamma, k, f, \epsilon, c_{ws}, \alpha_{ws}, \alpha_{sr}, \alpha_{cr}, \alpha_w$)
 2:     $t \leftarrow 0$, initial state $s \leftarrow s_0$, $\omega \leftarrow 1/k$
 3:     while $t <$ episode length limit or $s \neq$ terminal do
 4:         ▷ Sample new context, $i$, according to $\omega$
 5:         ▷ Select successor map $M_i$ according to sampled context
 6:         for all clusters do
 7:             ▷ Add optimism bonus to each state $\boldsymbol{w}$ according to $c_{ws}$ ($\alpha_{ws}$)
 8:             ▷ Add CR of state ($\boldsymbol{w}^{cr}$) to each state $\boldsymbol{w}$ ($\alpha_{ws}$)
 9:         ▷ $\epsilon$-greedy exploration, using $M_i \cdot \boldsymbol{w}_i$, to get next state $s'$ and respective reward
10:         for all clusters ($\alpha_w$) do
11:             ▷ Update reward weights $\boldsymbol{w}$ for $s'$ (Eq. 5)
12:         ▷ Update $M_i$ successor map (Eq. 3 or 7) ($\alpha_{sr}, \gamma$)
13:         if $t \geq f$ then
14:             ▷ Calculate CR for state visited at $t - f$, $\boldsymbol{v}^{cr}(s_{t-f})$ (Eq. 8) ($\alpha, \gamma$)
15:             ▷ Use a particle filter to infer cluster identity of $s_{t-f}$ to update $\omega$.
16:                 ▷ Returns most likely cluster, $i^*$
17:             ▷ Use $v^{cr}(s_{t-f})$ to update CR of most likely cluster, $\boldsymbol{w}_{i^*}^{cr}(s_{t-f})$
18:         $s_{t+1} \leftarrow s'$
19:         if $s_{t+1}$ is terminal then
20:             ▷ Update $\boldsymbol{w}^{cr}$ for all remaining visited states in episode
21:                 ▷ Repeat lines 14-17, adjusting $\boldsymbol{K}_\gamma$ accordingly
```

While the author provided an extensive supplementary section of the particle filter theory, aspects of the implementation remained unclear. After many different iterations, the pseudo-code in Algorithm 2 represents our best approximation of the implementation.

Algorithm 2 Particle Filter

```
1:  procedure FILTER($\sigma_{cr}$)
2:      for all rows (particle/history of observations) do
3:          ▷ Generate a new cluster, or "proposal" $c^p$, according to CRP (Eq. 12)
4:      for all unique clusters proposed in CRP do
5:          ▷ Calculate the likelihood of clusters ($l$) given the observed (Eq. 11)
6:              ▷ $v^{cr}(s_{t-f}) \sim N(\boldsymbol{w}_j^{cr}(s_t), \sigma_{cr}^2)$
7:          ▷ Assign likelihood ($l^p$) to each particle according to its proposed cluster $c^p$
8:      ▷ Normalize to calculate $c^p$ (particle proposal) importance weights ($\boldsymbol{w}$)
9:          ▷ $\boldsymbol{w}^p$ for each particle, $\boldsymbol{w}^p = \frac{l^p}{\sum_{p'} l^{p'}}$
10:     ▷ Resample a new column for the particle matrix $P$ from particle proposals ($\boldsymbol{c}$) according to
    their importance $\boldsymbol{w}$
11:     ▷ Remove first (oldest) column of $P$
12:     ▷ $\omega(i) = \sum_{p \text{ if } c^p = i} w^p, \forall i$ clusters
13:     Return $\max_i(l^i)$, cluster with highest likelihood
```

At times the pseudo-code and supporting text had mismatches in notation and were hard to decipher. As such, we made the following assumptions and changes to produce our code:

- As the value of k was not clearly defined, we assumed a value of 4 or 6 as written in the Figure 2c of the original paper.
- For the convolved reward kernel, $\boldsymbol{K}_\gamma$ notation in pseudo-code and text had $\boldsymbol{K}_\gamma = [\gamma^{-f}, \gamma^{-f-1}, \ldots, 1, \ldots, \gamma^{f-1}, \gamma^f]$. However, upon further inspection, elsewhere in the text and in Figure 1c suggests that the kernel should be symmetric about the center. Therefore, our current implementation uses $\boldsymbol{K}_\gamma = [\gamma^f, \gamma^{f-1}, \ldots, 1, \ldots, \gamma^{f-1}, \gamma^f]$.
- For the convolved reward kernel, $\boldsymbol{K}_\gamma$, we also assumed γ was the same as that used in the discounting future successor maps (Eq. 3).
- At line 23 of the original pseudo-code, we assumed cr_t meant the calculated convolved value (line 21), as the term was undefined otherwise.
- Line 16 has been modified from the pseudo-code of line 23 from the original text. While in the original text, $v_{s_{t-f}}^{cr}$ says to update CR of current state (s_t) for current cluster, $\boldsymbol{w}_i^{cr}(s_t)$, from other sections of the text we inferred that $v_{s_{t-f}}^{cr}$ should update the CR of the state $t-f$ (center of filter) of the most likely cluster at time $t-f$.
- In order to properly implement the convolved reward, we had to keep a short history of recent rewards and states visited. While this is similar to the memory buffer described, memory buffers outlined in the text are kept separately for each cluster. However, the convolved reward as described needs $< r, s' >$ independent of the chosen maps.

Implementation differences and challenges Part of the original implementation methods were consciously not reproduced:

- We forewent annealing as described in the paper for all models. The only parameter annealed for the models reproduced, as documented in the original paper, was α_{cr} in the grid-world. However, in Table S1 of the original work, the parameter was contradictorily reported as fixed. We therefore forewent annealing for our simulations.
- For sake of time for replication, we forewent the use of memory buffers and replay in the original paper. In the related work section of the original text, it is suggested that results are not dependent on this feature.

It should be noted that, although the CRP is a non-parametric method for determining cluster labels, by providing a maximum number of clusters it is a parametric method here, contradicting what is written in the paper's introduction. The method does allow the agent to reuse "forgotten" labels. This had been an initial source of confusion during early stages of reproduction concerning k

3 Experiments & Results

The supplementary material from Madarasz (2019) [7] provided algorithmic details along with instructions on how to setup the domains and experiments.

3.1 Excluded experiments

For the grid-world experiment, the GPI and KQ methods was not implemented. For the puddle-world experiment, GPI-4, GPI-6, nor the equal weights (EW) methods were implemented. Additionally, since the results from the original work for BSR of 6 clusters were almost equivalent to BSR of 4 clusters (where the exploration offset of a constant plus $\boldsymbol{w}^{cr}$ had a greater outcome), we only ran experiments with BSR of 4 clusters with only an exploration offset of a constant plus $\boldsymbol{w}^{cr}$. The proposed improvements to a navigation task in a continuous state-maze was not implemented. Lastly, the work done in the neural data analysis of hippocampal flickering during navigation and splitter cells was not reproduced.

3.2 Hyper-parameter setup

BSR For our simulations, we used the reported best performing parameters. Below, we provide the parameter, description, line in this pseudo-code, value, and page in original paper where the value was reported. For the grid-world experiment, we have the following parameters in Table 1:

Table 1: Grid-world parameters for BSR.

Parameter	Description	pseudo-code	Value	Page
k	Max number of clusters		4	16
c_{ws}	Optimism offset	Alg 1.7	1	15
α_{ws}	Optimism offset ($\boldsymbol{w}$) learning rate	Alg 1.7,1.8	0.01	15
ϵ	ϵ-greedy	Alg 1.9	0	4
α_w	Reward weight ($\boldsymbol{w}$) learning rate	Alg 1.11	1	15
α_{sr}	Successor map (M) learning rate	Alg 1.12	0.005	16
γ	Discount factor: SR maps & CR Kernel	Alg 1.12,1.14	0.99	16
f	Filter delay	Alg 1.13	3	15
α_{cr}	CR learning rate	Alg 1.16	0.15	15
α	CRP concentration	Alg 2.3	2	14
σ_{cr}	Std. Deviation of CR distr.	Alg 2.5	1.6	15

For the puddle-world experiment, we have the following change in parameters in Table 2 for different number of maximum clusters:

Table 2: Puddle-world parameters for BSR.

Parameter	Description	pseudo-code	Value	Page
k	Max number of clusters		4	16
ϵ	ϵ-greedy	Alg 1.9	0.05	16
α_{sr}	Successor map (M) learning rate	Alg 1.12	0.05	16
k	Max number of clusters		6	16
ϵ	ϵ-greedy	Alg 1.9	0.1	16
α_{sr}	Successor map (M) learning rate	Alg 1.12	0.05	16

SSR It was assumed that other parameters for SSR not reported were the same value as for BSR (eg. $\gamma = 0.99$). The only change in parameters noted are reported in Tables 3, 4.

For grid-world experiments:

Table 3: Grid-world parameters for SSR.

Parameter	Description	Value	Page
ϵ	ϵ-greedy	0.1	16
α_{sr}	Successor map (M) learning rate	0.001	16

For puddle-world experiments (with constant offset/optimism bonus):

Table 4: Puddle-world parameters for SSR.

Parameter	Description	Value	Page
ϵ	ϵ-greedy	0.25	16
α_{sr}	Successor map (M) learning rate	0.005	16

We assumed, like BSR, $c_{ws} = 1$ and $\alpha_{ws} = 0.01$ for the offset.

Q-learner Since we added a Q-learning agent to the mix, it's parameters in Table 5 are by default from simple_rl.

Table 5: Parameters for Q-learner.

Parameter	Description	Value
ϵ	ϵ-greedy	0.1
α	Learning rate	0.1
γ	Discount factor	0.99

3.3 Included experiments

In order to perform the original experiments, the simple_rl framework needed to be extended as follows:

- A domain class that was able to construct both grid-world and puddle-world domains (`domains/RC_GridWorldMDPClass`).
- SSR and BSR agents described previously.
- Procedures to run agents on a changing MDP (`RC_run_experiments.py`).
- Procedures to build and run the experiments (`reproduction.py`).

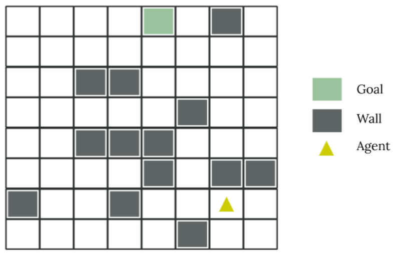

Figure 1: Grid-world domain.

Grid-world The grid-world experiment was performed on a 8×8 maze with a changing goal, of a reward of 10, and initial start state every 20 episodes with fixed wall locations (Fig. 1) for a total of 4500 episodes. Each episode terminates when the goal has been reached or after 75 steps. Each instance of this experiment ran for 10 times where the results from the instances were averaged. Refer to the Hyper-parameter setup section for extra parameter details.

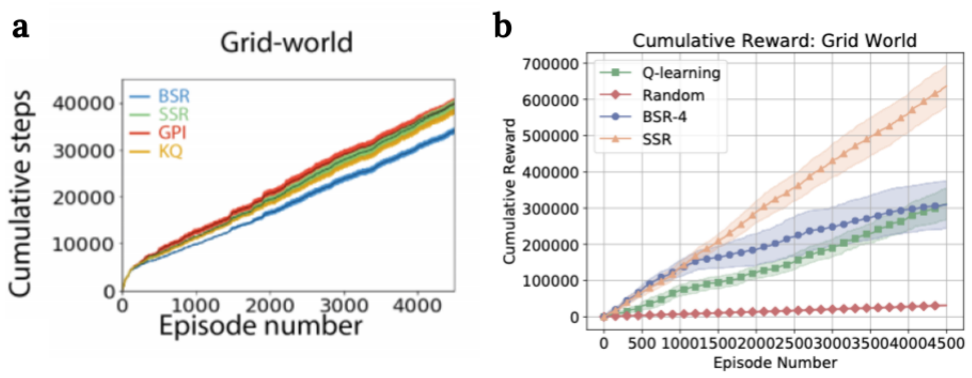

Figure 2: Grid-world experiment. (a) Original work (b) Reproduction

Our work took advantage of the simple_rl's provided output of cumulative reward graphs to analyze our reproduction. While this metric does differ from that of the original (Fig. 2a), where the author compares performance of models by cumulative steps, we argue that these are comparable metrics, even if not perfectly matched. Presumably, an agent with lower cumulative reward also has higher cumulative steps; as it is not reaching as many rewards, its episode and step length should be higher. We settled with the approximation of the metric in order to focus on attempts to faithfully reconstruct the nuances of the BSR.

In the original graph (Fig. 2a), BSR demonstrated lower cumulative steps in comparison to baselines, including SSR. In our reproduction (Fig. 2b), we see a divergence from this pattern. Interestingly, SSR shows the best performance in the grid-world domain, followed by our best-performing implementation of BSR. Prior attempts at implementing BSR resulted in either chance or below-chance performance of the model. This implementation as described above, notably outperforms, for various episodes, a standard Q-learning model (though one that is not tuned to the domain). These curves do converge in later episode number. We saw that this performance benefit of BSR did not hold in the puddle domain.

Furthermore, while Q-learning and SSR appear to have relatively linear performance curves, BSR appears to learn quickly and then decelerates. This may point to potential implementation reproduction issues on the inference step of the algorithm, despite careful inspection of the provided supplementary materials and pseudo-code.

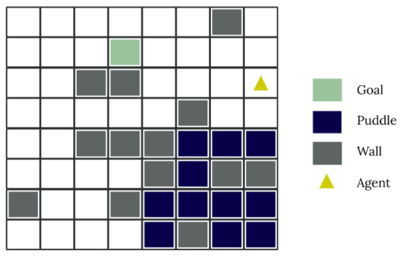

Figure 3: Puddle-world domain.

Puddle-world The puddle-world experiment inherits the almost equivalent setup as the grid-world experiment except that there are added puddles, of a penalty of -1, that fill the opposite 4×4 quadrant of the goal location (where the puddles are not terminal) and that the goal, puddle, and initial start states change every 30 episodes (Fig. 3).

We tested the reported best model versions in the original (Fig. 4a): BSR-4 with constant + $\boldsymbol{w}^{cr}$ offset and SSR with a constant offset. While it appears in Fig. 4a that SSR with an offset of constant + $\boldsymbol{w}^{cr}$ outperforms SSR with a constant offset, Table S2 and supporting text of the original work assert SSR with a constant offset is the outperforming model.

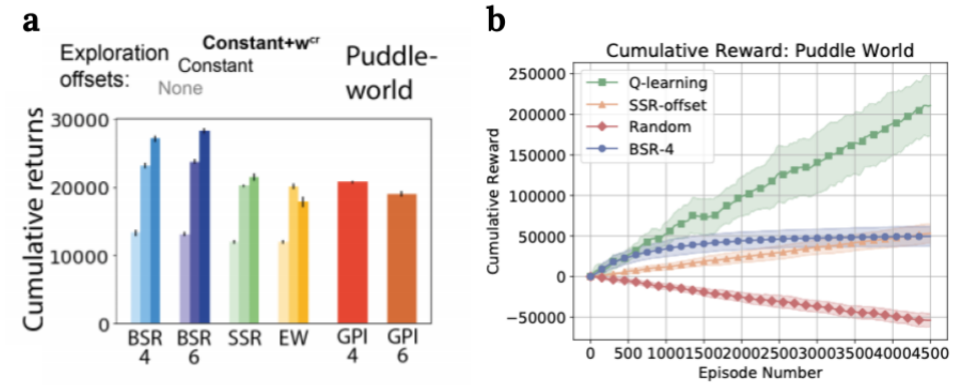

Figure 4: Puddle-world experiment. (a) Original work (b) Reproduction (BSR-4 with a constant + $\boldsymbol{w}^{cr}$ offset and SSR with a constant offset)

Again, we took advantage of simple_rl's provided output of cumulative reward graphs to analyze our reproduction. We equate the cumulative returns in the original figure (Fig. 4a) to the last episode in the cumulative return curves (Fig. 4b). We see that BSR-4 initially outperforms SSR-offset but by the final episode cumulative returns had converged. Again noting the slowing of the BSR-4 curve, discrepency between Fig. 4a and 4b may be due to difficulty in the re-implementing the BSR inference step. This was despite iteratively building up from the SSR code and careful review of the supplementary material and pseudo-code.

Curiously, despite implementation differences and relative performance differences between curves, the models in the reproduction appear to have overall higher cumulative rewards. It is not immediately obvious where this difference arises, especially since our reproduction, if anything, should perform worse without the addition of memory buffers and replay. Finally, of particular note is Q-learning's dominant performance despite changing puddle and goal locations. While not in the original figure, it provides another baseline that should have been outperformed by BSR given the original figure.

4 Conclusion

We analyzed the reproducibility of the methods within the work "Better transfer learning with inferred successor maps" Madarasz (2019) [7]. There were several incongruities between the text of the paper, the supplementary material, and the pseudo-code. We resolved these based on our interpretation. This lack of clarity may have led to a failure to replicate the results BSR for both grid-world and puddle-world experiments in Madarasz (2019) [7]. Furthermore, the Q-learning baseline without a hyperparameter search was comparable to or exceed the performance of the BSR algorithm.

Overall, we were unable to satisfactorily reproduce the findings solely from the provided supplementary materials and pseudocode. We do not suggest that this derives from an inability to replicate. Rather, the clarity issues and inconsistencies in the text during this reproduction process had been a major factor. We look forward to maintaining a dialogue with the authors of the original work about the differences between our implementations.

Acknowledgments

We would like to thank Rex Liu for his detailed explanation on particle filtering.

References

[1] David Abel. simple_rl: Reproducible reinforcement learning in python. 2019.

[2] Andre Barreto, Remi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. 06 2016.

[3] A. C. Courville, N. D. Daw, G. J. Gordon, and D. S. Touretzky. Model uncertainty in classical conditioning. Technical report, 2004.

[4] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993. doi: 10.1162/neco.1993.5.4.613.

[5] Samuel Gershman, David Blei, and Yael Niv. Context, learning, and extinction. *Psychological review*, 117: 197–209, 05 2010. doi: 10.1037/a0017808.

[6] Tamas Madarasz, Lorenzo Diaz-Mataix, Omar Akhand, Edgar Ycu, Joseph Ledoux, and Joshua Johansen. Evaluation of ambiguous associations in the amygdala by learning the structure of the environment. *Nature Neuroscience*, 19, 05 2016. doi: 10.1038/nn.4308.

[7] Tamas J. Madarasz. Better transfer learning with inferred successor maps. 2019.

[8] Andrew G. Barto Richard S. Sutton. *Reinforcement learning: An introduction*. MIT press, 2018.