SR-Reward: Taking The Path More Traveled

Anonymous authors
Paper under double-blind review

Abstract

In this paper, we introduce a method for learning a reward function solely from offline demonstrations. Unlike inverse reinforcement learning (IRL), our reward function is learned independently of the policy. This removes the need for an adversarial relationship between the two and provides a more stable training process. Our reward function, *SR-Reward*, is based on successor representation (SR). Taking advantage of the nature of SR, it is learned using the Bellman equation and can be trained alongside most reinforcement learning (RL) methods without requiring modification to the training pipeline. We describe our design decisions and the training procedure and show how such a reward function can be trained in combination with off-the-shelf offline RL algorithms. Additionally, we introduce a negative sampling strategy that reduces the reward value for out-of-distribution data, effectively combating overestimation errors and resulting in a more robust reward function for such data. When applied to the reward function, this strategy introduces an inherent conservatism into the RL algorithms that utilize it. We evaluate our algorithm on D4RL and find competitive performance compared to offline RL algorithms with access to the true reward as well as imitation learning (IL) algorithms such as behavioral cloning. Furthermore, our ablation studies over data size and data quality provide insights into the strengths and limitations of SR-Reward as a proxy to the true reward.

1 Introduction

Imitation learning (IL) from expert demonstrations is one of the most popular avenues for tackling sequential decision-making tasks. There are two categories of methods that make use of expert demonstrations. The first focuses on learning a policy that resembles the expert behavior, e.g. using Behavioral Cloning Pomerleau (1991). Another set of methods, known as inverse reinforcement learning (IRL) (Ng & Russell, 2000), first infer a reward function that explains the expert behavior and then learn a policy derived from that reward function. In popular application domains of IL, such as robotics or medicine, interacting with the environment during training is not always possible due to risk and safety concerns, making it crucial to be able to learn from only the limited, previously collected expert demonstrations.

In this work, we focus on the offline inverse reinforcement learning setting, where the agent neither has access to the reward function nor can query the expert for any feedback. Furthermore, the transition dynamics of the environment are unknown and the agent is provided with limited data in the form of expert demonstrations. Our first contribution, *SR-Reward*, is a reward function based on Successor Representations (SR), that is learned offline from expert demonstrations. Unlike adversarial schemes popular with IRL methods, our reward function is decoupled from the policy that is being learned. Decoupling the reward from the policy eliminates the instabilities associated with adversarial training and enables the use of a wide range of RL and offline RL algorithms that were previously unusable due to the inaccessibility of the reward function. Moreover, hand-engineering a reward function for complex real-world tasks, such as robot manipulation, is both challenging and error-prone (Wu et al., 2022; Singh et al., 2009). In contrast, demonstrating the desired behavior, while potentially costly, is generally more straightforward. In such scenarios, the ability to learn a dense reward function from demonstrations can be incredibly valuable. Leveraging the SR structure allows SR-Reward to be learned via the Bellman equation. Consequently, it can be integrated into existing training pipelines alongside other RL methods based on temporal difference (TD) learning with minimal modifications. By incorporating bootstrapped values in TD learning, the reward function gains a long-term perspective on the task, unlike simple imitation learning algorithms such as behavioral cloning (BC), which often struggle with distribution shift due to their short-sighted optimization objective.

Function approximation can lead to a significant overestimation of values for out-of-distribution data (Thrun & Schwartz, 1999). Our second contribution is a negative sampling strategy designed to counteract the overestimation error in our reward function for out-of-distribution states and actions. This is accomplished by augmenting the Bellman loss for SR-Reward so that reward estimates for out-of-distribution states and actions decrease based on their distance from expert demonstrations. Incorporating negative sampling not only enhances the robustness of the reward function but also introduces a natural conservatism into the value functions and policies that rely on it.

This paper is organized as follows: section 2 discusses similarities and differences between our proposed reward model and other methods of learning or modifying reward functions. Notation and background information used for developing our method is presented in section 3. Our method, *SR-Reward*, is presented in detail in section 4, including the motivation, design decisions, and implementation details. Section 4.3 describes our novel negative-sampling method used to increase our reward function's robustness. Finally, section 5 shows the performance of our method on D4RL environments. We use our reward function in combination with different offline RL algorithms and compare their performance against offline RL with access to the true environment reward. In addition, we compare the performance of our reward in combination with offline RL against IL methods such as BC.

2 Related Work

Learning to perform a task from offline data has been extensively studied under the IL and IRL umbrella (Abbeel & Ng, 2004; Ho & Ermon, 2016; Fu et al., 2018; Garg et al., 2021; Kostrikov et al., 2020; Kalweit et al., 2020; Pomerleau, 1991). One common approach is methods based on behavioral cloning (Pomerleau, 1991) which reduce imitation learning to a supervised learning problem, i.e., learning a mapping from environment states to expert actions. They aim to increase the probability of expert actions for the states seen in the demonstrations. Although this approach can work in simple environments with large amounts of data, it is inherently myopic and fails to reason about the consequences of its selected actions. Consequently, such greedy approaches suffer from compounding errors due to covariant shift (Ross et al., 2010) when the agent deviates from the demonstrated states.

In contrast, IRL methods incorporate information about the environment dynamics into the decision-making process by imitating the expert actions as well as the visited states (Abbeel & Ng, 2004). Many IRL methods, such as GAIL (Ho & Ermon, 2016) and its extensions, simultaneously estimate the reward function that best explains the expert behavior and its associated policy. This optimization is done using an adversarial scheme with the discriminator trying to distinguish between the expert trajectories and ones generated by the learned policy. Simultaneously, the discriminator's error is used as the reward signal for training the policy. The adversarial nature of the training strategy makes such algorithms prone to training instabilities(Goodfellow et al., 2014; Kostrikov et al., 2019). Additionally, they require further interactions with the environment during training to create a dataset of non-expert trajectories for training the discriminator. Furthermore, there are no theoretical guarantees that show adversarial training to lead to a better performance than a two-step process, which first infers a reward function from demonstrations, followed by learning a policy using the previously inferred reward (Liu et al., 2021). In this work, we part from adversarial training and so decouple the learning process of reward function and policy while training both simultaneously.

Already there have been efforts in bypassing the adversarial optimization. Kalweit et al. (2020) derived an analytical solution for the reward based on the assumption that expert policy follows a Boltzmann distribution. Their formulation applies to continuous states but is limited to discrete action spaces. While sharing a similar objective to ValueDICE (Kostrikov et al., 2020), Garg et al. (2021) removes the need for adversarial training by formulating the reward function in terms of the value functions and maximizing them. Their objective function implicitly reduces a distance measure, such as χ^2-divergence, between the occupancy measure of the expert and the one of the policy being trained. This approach does not yield an explicit reward model, but a reward value can be extracted from the learned value function and policy.

Without optimizing for the optimal reward function Reddy et al. (2020) uses a simple binary indicator as the reward which distinguishes between expert demonstrations and online interactions. Our reward function can be seen as the continuous version of SQIL (Reddy et al., 2020) because the SR value of states and actions that are visited by the expert, will be naturally higher than the rest.

There is also a series of works that modify the existing environment reward to gain improvements during training. Vieillard et al. (2020) suggests adding $\log(\pi(a|s))$ of the policy π that is being learned to the reward in temporal difference (TD) learning. The authors argue that the logarithm of the policy is a strong learning signal as it is available even in a sparse reward setting. Since its value is close to zero for optimal actions under optimal policy this does not conflict with the optimal control objective. Other works such as (Moskovitz et al., 2022; Machado et al., 2020) modify the reward using the SR of the policy that is being learned. Machado et al. (2020) showed that the norm of the SR vector can act as the proxy for the state visitation count. They modify the reward from the environment by adding the inverse of this state visitation count during training. Moskovitz et al. (2022) suggest a modification to SR to only consider the first visitation of a state, hence learning the expected discounted time to reach successor states. Similar to Machado et al. (2020), they also make use of the inverse of their modified SR norm and show improved performance, especially in scenarios where the reward in a given state will be depleted after the first visit.

Our work is similar to the one of Machado et al. (2020) in the sense that we are also viewing the norm of SR vector as a proxy to state visitation count. However, we are working in an offline IRL setting where there is no other reward available and the dataset is fixed. We show that in the absence of a reward signal from the environment, one can use the norm of the SR vector directly as the reward. Additionally, we extend the SR vector to continuous actions and employ a negative sampling procedure to lower the value of our SR-based reward for state-action pairs that were not present in the demonstrations dataset, hence combating the extrapolation error and creating a more robust reward function for offline RL algorithms.

3 Background

We first introduce the notation and provide a more detailed review of concepts from imitation learning and successor representation.

3.1 Notation

We consider settings where the environment is represented by a Markov Decision Process (MDP) and is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ represent the continuous state and continuous action spaces respectively. $\mathcal{T}(s'|s,a)$ represents the state transition dynamics, $r(s,a)$ represents the reward function and $\gamma \in (0,1]$ is the discount factor. In the offline inverse reinforcement learning setting, we only have access to a limited set of expert demonstrations of the form $\mathcal{D} = \{(s_0, a_0, s_1, a_1, ...s_T)^i\}_{i=0}^{N}$. In this paper, we are focusing on a minimal setting where neither the transition dynamics $\mathcal{T}(s'|s,a)$ nor the reward function $r(s,a)$ are known. The goal is to learn a reward function $r_\theta(s,a)$ from expert demonstrations such that its corresponding policy $\pi_\phi(a|s)$ performs similarly to that of the expert.

3.2 Imitation Learning via Distribution Matching

Methods like behavioral cloning (BC), which directly learn a policy $\pi(a|s)$ mapping states to actions, are straightforward and effective when ample data is available. However, they are prone to distribution shift because they only match the observed action distribution. During inference, as the distribution of encountered states deviates from those seen during training, the accuracy of action predictions diminishes. This leads to accumulating errors that the policy cannot correct.

Distribution matching methods, and related approaches (Ke et al., 2020; Kostrikov et al., 2020; Nachum et al., 2019; Ho & Ermon, 2016; Fu et al., 2018; Ghasemipour et al., 2019), are more robust to distribution shifts since they aim to match both the state and action distributions encountered during training. This helps keep the policy close to the states observed in demonstrations.

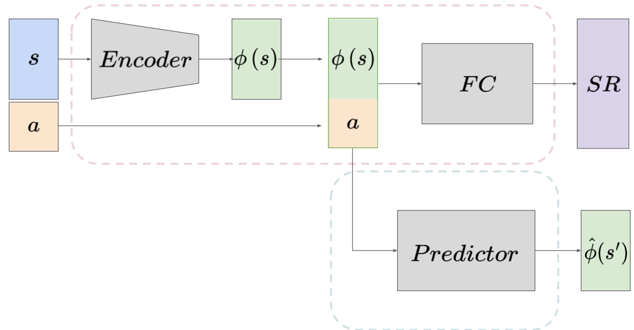


Figure 1: The architecture of the SR networks. The output of the *Encoder* is concatenated with the action to produce $\phi(s,a) = \binom{\phi(s)}{a}$ in Equation 3. The result passes through a fully connected network (FC) to create the $SR(s,a)$ vector. The *Predictor* network is used for an auxiliary task to help train the *Encoder*. It predicts $\hat{\phi}(s')$, an estimate of the true encoded next state $\phi(s')$, from $\binom{\phi(s)}{a}$.

Formally, the occupancy measure of a state-action pair under policy π can be defined as

$$\rho^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}(s_t = s, a_t = a)\right],$$

where $\mathbb{I}$ is the indicator function, which equals one if the condition is met and zero otherwise. This is closely related to the state-action distribution $d^\pi(s,a) = (1-\gamma)\rho^\pi(s,a)$. As shown by Puterman (1994), there is a one-to-one correspondence between the state-action distribution and the policy.

Distribution matching methods aim to indirectly learn a policy by minimizing the divergence between d^{Expert} and d^π. A common choice is KL-Divergence, and minimizing $D_{KL}(d^\pi || d^{Expert})$ can be viewed as maximizing the RL objective

$$\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t \log\frac{d^{Expert}(s,a)}{d^\pi(s,a)}\right],$$

where the reward is given by the log ratio of the state-action distributions between the expert policy and the learned policy π. Since the state-action distribution is often unavailable, efforts are typically focused on estimating the ratio of the two distributions (Ho & Ermon, 2016; Nachum et al., 2019).

In this paper, we propose a method to directly estimate a proxy for the expert's state-action distribution and use it as the reward for downstream RL algorithms.

3.3 Successor Representations

Successor Representation (SR) was originally introduced as a method to generalize the value function across different rewards (Dayan, 1993). SR is defined as the cumulative discounted probability of visiting future states when following a specific policy, effectively representing the current state (and action) in terms of potential future states (and actions).

For any given pair of states s, s' and actions a, a', the SR is expressed as:

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

where the expectation is taken over the policy $\pi(a|s)$ and the environment's transition dynamics $\mathcal{T}(s'|s,a)$. Similar to the Q-function, SR can be estimated using the recursive Bellman equation:

$$M(s_t,a_t,s',a') = \mathbb{I}(s_t = s', a_t = a') + \gamma\mathbb{E}\left[M(s_{t+1},a_{t+1},s',a')\right]. \tag{2}$$

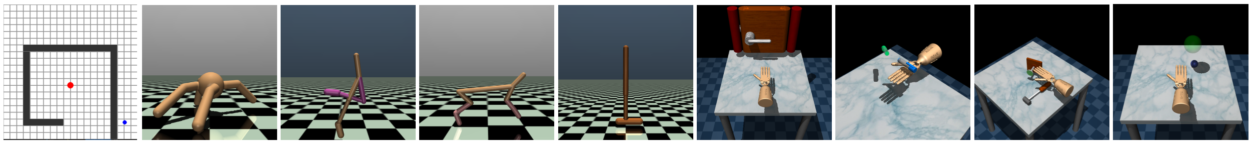


Figure 2: Environments used for our experiments. From left to right: 2D Toy Maze, MuJoCo environments: [Ant, Walker2D, HalfCheetah, Hopper], Adroit Hand environments: [Door, Pen, Hammer, Relocate]

This recursive formulation is particularly useful when learning SR alongside other temporal difference (TD) methods. Our SR-based reward function leverages this recursive approach, allowing the reward network to be trained in parallel with the actor and critic networks, with minimal changes to the existing training pipeline.

However, directly estimating SR using these formulations becomes computationally intractable as the number of states and actions increases, or when transitioning from discrete to continuous domains. To address this, previous research (Kulkarni et al., 2016; Machado et al., 2020; Zhang et al., 2017) has extended SR to continuous state and action spaces using Successor Features Representation (SF). SF is expressed in terms of state and action features $\phi(s, a)$:

$$M(s_t, a_t) = \phi(s_t, a_t) + \mathbb{E}\left[M(s_{t+1}, a_{t+1})\right]. \tag{3}$$

The choice of feature extractor ϕ is a design decision that depends on the environment. Most existing work focuses on extracting features only from the state, not the actions. In this scenario, $\phi(s, a)$ can be represented as $\binom{\phi(s)}{a}$, which is a concatenation of state features and actions.

In this work, we adopt this approach and use a feature extractor network to derive features from the state only.

3.3.1 Relationship to State-Action Visitation

SR implicitly captures the state-action visitation. Many density-based IL methods, such as GAIL (Ho & Ermon, 2016), use state-action distribution or occupancy measure for their distribution matching techniques. In Appendix:A, we derive the following close relationship between the occupancy measure and the successor representation

$$\rho(s', a') = \mathbb{E}_{s_0 \sim \mu_0, s \sim \mathcal{T}, a \sim \pi}\left[M(s, a, s', a')\right]. \tag{4}$$

The occupancy measure $\rho(s', a')$ can be seen as the expectation of successor representations $M(s, a, s', a')$ with respect to the probability of all state-action pairs (s, a) that preceded (s', a'). We learn successor features representation (SF) from the expert demonstrations using function approximation and use it as a proxy to the occupancy measure of the expert.

4 Technical Approach

4.1 Architecture

We use the architecture shown in Figure 1 to estimate the SR vector in continuous state and action settings. Our architecture is built upon the works of Machado et al. (2020), Kulkarni et al. (2016), and Borsa et al. (2019) with a few notable changes. First, our SR network extends the previous works to include the action when estimating the SR. This is important as our SR-based reward function $r(s, a)$ is a function of both the state and the action and needs to distinguish the reward values of different actions. Second, it is common to use an auxiliary task when learning the encoder from scratch. Kulkarni et al. (2016) use the reconstruction of the state as the auxiliary task, while Machado et al. (2020) opt for a prediction task in which the next state is predicted from the encoded state and the action. Inspired by the results of Ni et al. (2024), we use the prediction of the next encoded state as our auxiliary task. Given the encoding of the current state $\phi(s)$ and its corresponding action in the dataset a, we predict the encoded next state $\phi(s')$. We use the l^2 loss for this auxiliary task. Finally, our encoder consists of fully connected layers with ReLU activation layer as the final layer. We normalize the feature vector to ensure that all features are in the same range, such that $\|\phi(s)\|_1 = 1$ as suggested by Machado et al. (2020). If the environment dynamics are not fully Markovian one can use a history of states as s and replace the fully connected layers of the encoder with LSTM layers as proposed by Borsa et al. (2019).

4.2 SR to Reward

Machado et al. (2020) shows that the norm of SR implicitly counts the state visitation. Motivated by this result, we use the l^2-norm of the SR vector as our reward function. Intuitively, each element i of the SR vector, estimated using Equation 3, is the expected discounted sum of feature i of the state according to the policy that created the demonstration dataset. Hence aggregating all the elements of the SR vector in our offline setting can be seen as a visitation count of the state-action pairs when following the demonstration policy. If the demonstrations are created by an expert, $\|SR(s, a)\|_2$ represents how often the expert has visited (s, a) while performing a task. Taken as the reward for offline RL, we set out to find a policy that maximizes the state-action visitation of the expert. We empirically show that we can learn competitive policies using this reward function.

4.3 Negative Sampling

Neural networks tend to overestimate the value of out-of-distribution data points (Thrun & Schwartz, 1999; Ball et al., 2023; Fujimoto et al., 2019; 2018). The overestimation error is especially concerning in our setup because an overestimated value of the reward for unseen states and actions will encourage the value networks and subsequently the policy to diverge from the expert demonstrations. Motivated by the idea of conservative value function via negative sampling Luo et al. (2020), we develop our negative sampling strategy to combat the overestimation error of our SR network. Similar to Luo et al. (2020) we create our negative samples $\hat{s}$ and $\hat{a}$ by adding a small Gaussian noise to states and actions from our expert trajectories. However, instead of subtracting the l^2-norm of the difference vector $\|s - \hat{s}\|_2$ we decay the values using a Gaussian

$$\exp\left(-\frac{\|s - \hat{s}\|_2}{\sigma^2}\right)$$

with σ controlling the strength of the decay. Furthermore, we apply negative sampling not to the space of value functions but to the space of rewards. This is possible in our setting where a reward function is estimated and used for learning a policy. Having control over the reward function in this setting provides the opportunity to build conservatism directly into the value functions and subsequently the policy by modifying the reward instead of forcing the value function or the policy to act conservatively Kumar et al. (2020); Fujimoto et al. (2019; 2018). Figure 4 shows the effect of our negative sampling strategy on a toy environment. We train an SR network using ten demonstrations with and without negative sampling and evaluate the rewards over the grid space for each one of the cardinal directions. The mean value plot in Figure 4 shows how negative sampling during training prevents overestimation error for the rewards of the state-action pairs not seen in the demonstrations. Plots for the four main directions show higher reward estimates for the corresponding direction of movement. For example, the `Left` plot shows the reward for moving left at every grid point and as expected this reward is higher for the upper portion of the trajectories where the expert has moved left.

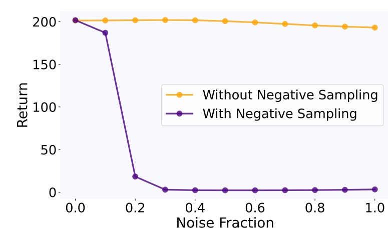
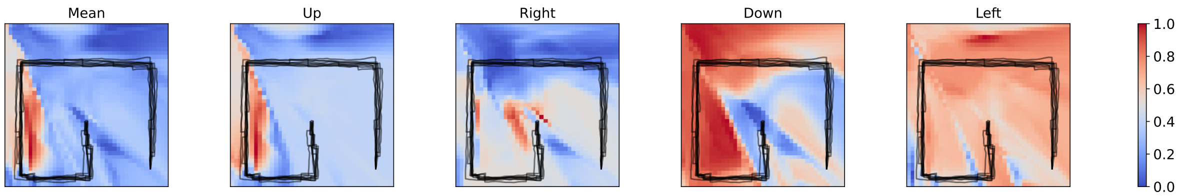
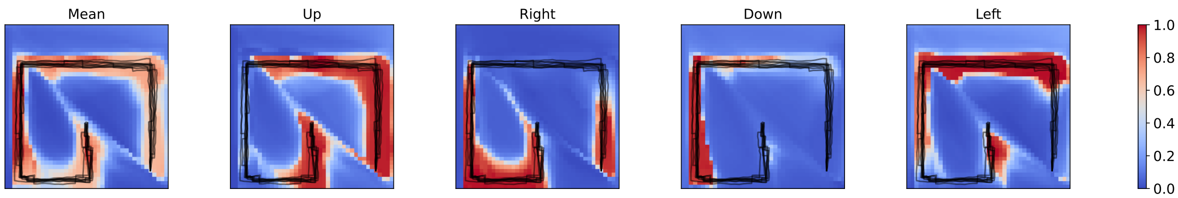


Figure 3: Effect of negative sampling on returns of perturbed expert demonstrations for Relocate environment. Negative sampling significantly reduces the reward for states and actions further away from the expert demonstrations.

To further illustrate the effects of negative sampling on preventing over-estimation error, Figure 3 compares the mean return estimated by SR-Reward trained with or without negative sampling. Expert trajectories are corrupted by varying levels of Gaussian noise and their episodic return is estimated using SR-Reward. The model trained with negative sampling produces significantly smaller returns for corrupted trajectories. This effect is shown by an initial drop in the returns when the trajectories are corrupted by Gaussian noise with $\sigma = 0.1$ and a much steeper drop when the noise is increased further. The model trained without negative



(a) Without Negative Sampling



(b) With Negative Sampling

Figure 4: The plots show the effect using negative sampling for a 2D Toy Maze environment (Figure 2)with continuous states and actions. The black lines represent the trajectories of the expert starting near the bottom-right and moving counter-clockwise towards the goal near the center. The mean of the SR-Reward over four directions [Up, Right, Down, Left] and the SR-Reward associated with each direction is plotted. Using negative sampling significantly reduces the extrapolation error for out-of-distribution state-action pairs.

sampling shows similar return values for expert and corrupted trajectories. The returns for this model show a small drop only after adding a significant amount of noise. Different environments may require different levels of sensitivity to noise and out-of-distribution data. Our negative sampling strategy can be configured for each environment by tuning the level of perturbation noise added to the expert trajectories as well as σ for the decay rate in Equation 8.

4.4 Training

We employ several loss functions to train our SR network. As mentioned in Section 3.3, we can estimate the SR using the Bellman equation in a continuous state-action setting. The reward for the Bellman target in Equation 3 is replaced with $\phi(s,a) = \binom{\phi(s)}{a}$ which is the concatenation of the encoded state and the action. Note that $\phi(s)$ and $M(s_{t+1}, s_{t+1})$ are calculated without the gradient. We use the l^2-loss to minimize the Bellman error:

$$\mathcal{L}_{Bellman} = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}\left[(M(s,a) - (\phi(s,a) + \gamma M(s',a')))^2\right] \tag{5}$$

To help train the encoder we use an auxiliary prediction task that predicts the next encoded state $\phi(s')$ from the current encoded state $\phi(s)$ and action a. We compute the l^2-loss as

$$\mathcal{L}_{Prediction} = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[(\phi(s') - Predictor(\phi(s),a))^2\right] \tag{6}$$

We have added an extra loss to penalize the magnitude of the reward for values greater than 1. This loss was found to stabilize the training and create a soft upper bound for the reward. As explained in Section 4.2 we use the l^2-norm of the SR vector as our reward.

$$\mathcal{L}_{Magnitude} = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\max(Reward(s,a) - 1, 0)^2\right] \tag{7}$$

Finally, we add a negative sampling loss to improve the robustness of the reward function for out-of-distribution state-action pairs. Similar to Luo et al. (2020) we create negative samples $\tilde{s}$ and $\tilde{a}$ by perturbing states and actions from the demonstrations with noise. There might be concerns that the negative samples

Algorithm 1 SR-Reward + RL

Given: $\mathcal{D} : [(s, a, s', a')^i]_{i=0}^N, \gamma, \beta, \sigma$
Initialize: Encoder: **Enc**, Fully Connected MLP:**FC**, Predictor: **Pred**
for each training step **do**
  Sample $(s, a, s', a') \sim \mathcal{D}$
  $\phi(s) \leftarrow Enc(s)$
  $\phi(s') \leftarrow Enc(s')$
  $SR \leftarrow FC(\phi(s), a)$
  $SR_{target} \leftarrow \binom{\phi(s)}{a} + \gamma FC(\phi(s'), a')$
  $\boldsymbol{\mathcal{L}_{Bellman}} \leftarrow MSE(SR, SR_{target})$
  $\phi_{pred}(s') \leftarrow Pred(\phi(s), a)$
  $\boldsymbol{\mathcal{L}_{Prediction}} \leftarrow MSE(\phi_{pred}(s'), \phi(s'))$
  $r \leftarrow \|SR\|_2$
  $\boldsymbol{\mathcal{L}_{Magnitude}} \leftarrow (max(r - 1, 0))^2$
  $\tilde{s} \leftarrow s + \mathcal{N}(0, \beta)$
  $\tilde{a} \leftarrow a + \mathcal{N}(0, \beta)$
  $\phi(\tilde{s}) \leftarrow Enc(\tilde{s})$
  $\alpha_{decay} \leftarrow exp(\frac{-\|[\phi(s),a]-[\phi(\tilde{s}),\tilde{a}]\|_2}{\sigma^2})$
  $\tilde{SR} \leftarrow FC(\phi(\tilde{s}), \tilde{a})$
  $\tilde{r} \leftarrow \|\tilde{SR}\|_2$
  $\boldsymbol{\mathcal{L}_{Neg.Sample}} \leftarrow MSE(\tilde{r}, \alpha_{decay} * r)$
  $\boldsymbol{\mathcal{L}_{Total}} \leftarrow \mathcal{L}_{Bellman} + \mathcal{L}_{Prediction} + \mathcal{L}_{Magnitude} + \mathcal{L}_{Neg.Sample}$
  $s \leftarrow \binom{s}{\tilde{s}}, a \leftarrow \binom{a}{\tilde{a}}, s' \leftarrow \binom{s'}{s'}, r \leftarrow \binom{r}{\tilde{r}}$
  $\boldsymbol{RL}(s, a, r, s')$
end for

will fall into the same distribution as the demonstrations and so harm the estimation of the SR. However, as discussed by Luo et al. (2020), the demonstrations cover only a small subset of the space, hence the negative samples are with high probability orthogonal to the demonstrations, an effect that increases with the state and action dimensions of the environment. We use isotropic Gaussian noise $\mathcal{N}(0, \beta)$ to create the negative samples. The hyperparameter β controls the standard deviation of the Gaussian noise. Intuitively, we want perturbed state-action pairs $(\tilde{s}, \tilde{a})$ to have lower reward values proportional to the distance from their counterpart (s, a) from the dataset. Since SR estimates the visitation count based on $\phi(s, a) = \binom{\phi(s)}{a}$, we measure the distance between the negative samples and their original counterparts in the space of features and actions $\binom{\phi(s)}{a}$. We calculate the decay factor using an exponential kernel as

$$\alpha_{decay} = \exp\left(\frac{-\|\phi(s, a) - \phi(\tilde{s}, \tilde{a})\|_2}{\sigma^2}\right). \tag{8}$$

σ can also be adjusted as a hyperparameter. Higher values of σ will produce a softer decay for the reward of negative samples. The l^2-loss is used to correct the estimation of SR for negative samples:

$$\mathcal{L}_{Neg.Sample} = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[(Reward(\tilde{s}, \tilde{a}) - \alpha_{decay} * Reward(s, a))^2\right] \tag{9}$$

We train our SR network using the summation of all losses as our total loss:

$$\mathcal{L}_{Total} = \mathcal{L}_{Bellman} + \mathcal{L}_{Prediction} + \mathcal{L}_{Magnitude} + \mathcal{L}_{Neg.Sample} \tag{10}$$

Algorithm 1 shows the pseudocode for training the SR-Reward and the offline RL in the same loop. The sampled transitions used for training the SR networks have the form (s, a, s', a') which is different from the ones typically used for RL due to the addition of the next action a'. This form of transition, however, can be

Table 1: Normalized return from different algorithms trained on D4RL datasets (Fu et al., 2020). Offline RL algorithms equipped with SR-Reward perform as well or better than BC and their counterparts using the ground truth rewards from the environment.

		f-DVL		**sparseQL**	
Env	**BC**	True Reward	SR-Reward (Ours)	True Reward	SR-Reward (Ours)
Ant	85.71 ± 32.48	83.84±32.63	81.64±32.12	**87.73±32.51**	82.31±32.86
Hopper	108.64 ± 13.88	**111.28±9.17**	108.69±14.73	110.64±10.86	109.69±11.19
Halfcheetah	105.55 ± 12.09	104.92±7.61	103.76±13.64	**106.23±5.52**	106.03 ± 5.20
Walker2d	73.22± 42.41	84.57±38.40	79.64±38.73	83.00±38.69	**85.57 ± 32.09**
Door	76.76 ± 35.13	96.40 ± 21.59	98.24 ± 17.44	78.14 ± 39.26	**104.02 ± 8.60**
Hammer	114.21 ± 29.66	90.67 ± 52.51	111.64 ± 35.91	69.90 ± 56.70	**117.50 ± 21.26**
Pen	101.74 ± 62.93	106.67 ± 61.82	94.92 ± 64.53	**107.11 ± 61.78**	103.52 ± 62.27
Relocate	**92.92 ± 26.67**	92.57 ± 25.48	88.21 ± 32.44	80.86 ± 35.57	91.58 ± 25.91

easily produced with access to a set of demonstrations $\mathcal{D}$. We warm-start the training loop by pre-training the SR networks for 10,000 steps before using its SR-Reward to train the RL agent.

5 Experiments

5.1 Experimental Setup

To evaluate our proposed SR-Reward, we integrate it with two different offline RL algorithms: f-DVL (Sikchi et al., 2023) and SparseQL (Haoran Xu, 2023). These recent algorithms, which build upon ideas from IQL (Kostrikov et al., 2022) and XQL (Garg et al., 2023), demonstrate improved stability and high performance. We train our reward function in tandem with these offline RL algorithms, replacing the rewards in the offline dataset with those generated by SR-Reward.

For our evaluation, we use the popular MuJoCo-based (Todorov et al., 2012) environments for locomotion tasks and the Adroit hand (Rajeswaran et al., 2018) environments to assess performance on more realistic tasks with hand-engineered rewards. Figure 2 illustrates the environments used in our evaluations.

Our agents are trained for one million gradient steps, using three seeds for each task. Similar hyperparameters are applied across all environments, with the most critical ones listed in Table 2. We utilize the offline datasets provided by D4RL (Fu et al., 2020) and adhere to their normalization procedure, using their reported scores for random and expert demonstrators.

It is our aim to answer the following questions through our experiments:

1) How does using SR-Reward + RL compare against BC with sufficient data?

2) How does using SR-Reward + RL compare against offline RL with the ground truth reward?

3) How does the performance change with a decrease in data size?

4) How does the performance change with a decrease in data quality?

5.2 SR-Reward v.s. BC

Behavioral Cloning is a simple yet effective imitation learning algorithm if a large amount of expert data is available. D4RL datasets include more than 1000 expert demonstrations for each MuJoCo task and 5000 demonstrations for Ardoit hand environments. This data is sufficient for training a competitive BC agent. Table 1 shows that using the combination of SR-Reward + RL rivals BC agent trained on large datasets. RL agents trained using SR-Reward often outperformed BC in both MuJoCo and Adroit environments. Unlike reward-free IL algorithms, offline RL algorithms depend on a good reward function. Our results suggest that

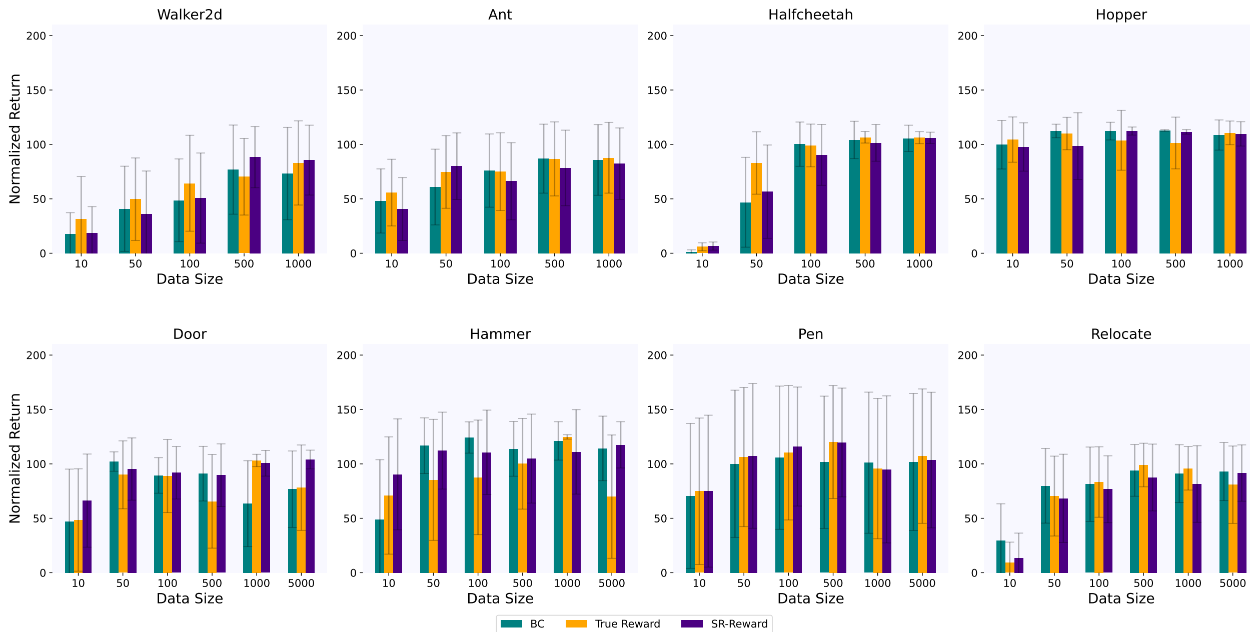


Figure 5: Effect of data size on performance. RL agents (SparseQL) using SR-Reward show competitive performance compared to BC and the RL agents that use the true reward.

SR-Reward provides an informative reward signal for offline RL agents, allowing them to achieve competitive results.

In addition, our findings highlight SR-Reward as an independent module that can be combined with any offline RL agent. This modular approach opens the door to applying offline RL algorithms in tasks such as robotics, where creating demonstrations comes more naturally than engineering a dense reward function for each task or where a sparse reward is insufficient.

5.3 SR-Reward v.s. True Reward

Since we use SR-Reward as a proxy for the true reward, we compare the performance of agents trained with SR-Reward to those trained with the true reward provided by the environment. As shown in Table 1, the performance of SR-Reward combined with RL in MuJoCo environments closely matches that of offline RL agents trained with the true reward. This suggests that the dense reward generated by SR-Reward is as informative as the environment's native reward.

The Adroit hand environments pose greater challenges for offline RL algorithms due to their narrower distribution of trajectories and the discontinuous, hand-engineered rewards provided for each task. Similar to the MuJoCo experiments, we train the offline RL agents using the hand-engineered rewards from the D4RL datasets, which consist of multiple sub-rewards and thresholds. These discontinuous rewards underscore the difficulties of hand-engineering reward functions and highlight the value SR-Reward offers.

As shown in Table 1, agents trained with SR-Reward often outperform those trained with the dense reward from the environment. This demonstrates the effectiveness of SR-Reward in scenarios where only sparse rewards are available or where hand-engineered reward functions fail to provide a sufficiently informative reward signal.

5.4 SR-Reward v.s. Data Size

To investigate the impact of data size on SR-Reward, we trained each algorithm using varying numbers of expert trajectories from the D4RL dataset. We used sparseQL as our offline RL algorithm. Specifically, we

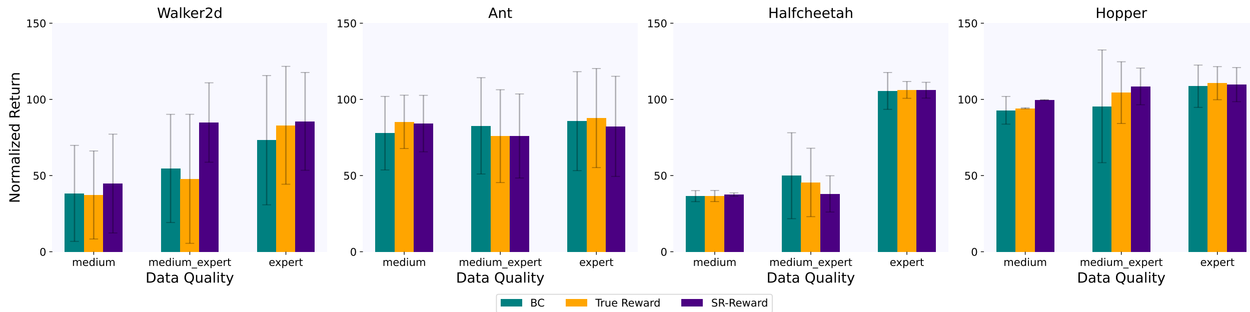


Figure 6: Effect of data quality on performance. RL agents (SparseQL) using SR-Reward show similar performance to the baselines. SR-Reward is robust to the mixing of sub-optimal demonstrations (medium-expert) as there is no significant drop in performance compared to the agents that were trained on the true reward.

evaluated performance using [10, 50, 100, 500, 1000] demonstrations for MuJoCo and [10, 50, 100, 500, 1000, 5000] demonstrations for Adroit hand environments. As shown in Figure 5, agents trained with true reward slightly outperform those trained with SR-Reward across different data sizes in all MuJoCo environments, with the latter catching up as the number of demonstrations is increased.

It's important to note that MuJoCo environments provide dense and continuous rewards based on the agents' velocity along the X-axis. The straightforward nature of these rewards presents a challenge for any reward-learning algorithm attempting to outperform them. In contrast, agents trained with SR-Reward often achieve higher returns in most Adroit hand environments than those trained with the true reward, indicating that SR-Reward can learn a more informative reward function than the hand-engineered rewards offered by the environment.

As the number of demonstrations decreases, performance declines for all agents, regardless of the reward function used. This trend suggests that informative rewards can still be learned even with limited data. Therefore, the performance drop observed with fewer demonstrations likely reflects the data inefficiency of the offline RL algorithms rather than a significant decline in the quality of the learned reward.

5.5 SR-Reward v.s. Data Quality

Depending on the environment, creating a set of high-quality expert demonstrations can quickly become a cumbersome task. Therefore, it is important to know the effect of sub-optimal demonstrations when used for training the SR-Reward. We conduct our experiments on MuJoCo environments using three datasets with different quality demonstrations from D4RL with the "medium-expert" dataset being a combination of both expert and medium demonstrations. Figure 6 shows that agents using SR-Reward have similar or better performance than the ones trained using true environment reward. The mixing of expert and medium datasets does not show a significant negative impact on agents trained with SR-Reward. In fact including the sub-optimal trajectories results in higher returns, especially for the more difficult Walker2D environment which can benefit from larger datasets. Having low sensitivity to sub-optimal demonstrations is a desirable attribute of SR-Reward since collecting expert demonstrations can be tedious and error-prone, which increases the possibility of including sub-optimal demonstrations.

6 Conclusion

We introduced SR-Reward, a reward function based on successor representation, which is learned from offline expert demonstrations. This reward function assigns high rewards to state-action pairs frequently visited by expert demonstrators. SR-Reward is independent of both policy and value functions but can be trained concurrently with them, enabling easy integration with various RL algorithms without requiring significant modifications to the training pipeline.

Table 2: Most important Hyperparameters used in the experiments.

Hyperparameter	Value
Noise β (MuJoCo)	1.0
Noise σ (MuJoCo)	3.0
Noise β (Adroit)	0.1
Noise σ (Adroit)	0.3
LR (Critic, Value)	0.0003
LR (Actor, SR-Reward)	0.0001
Encoder MLP	[256, 128]
SRNet (FC) MLP	[128]
Predictor MLP	[128, 32]
Critic MLP	[256, 256]
Actor MLP	[128, 128]
ValueNet MLP	[128, 128]
Batch Size	128
Training Steps	1000000

Additionally, we implemented a negative sampling strategy to encourage a pessimistic estimation of rewards for out-of-distribution state-action pairs, thereby making the reward function more resistant to overestimation errors. Our empirical results demonstrate that SR-Reward can effectively serve as a proxy for the true reward in scenarios where no reward function is available or where the complexity of the task makes it difficult to hand-engineer sufficiently informative reward functions.

7 Limitations and Future Work

Our experiments focused exclusively on state-based demonstrations. Extending these methods to visual domains is possible by substituting the encoder in Figure 1 with one capable of extracting meaningful representations from image data. Designing an effective visual encoder introduces a new set of engineering challenges that must be addressed when expanding to visual domains.

One limitation of SR-Reward is the assumed availability of a dataset of optimal trajectories. Although our empirical results indicate a degree of robustness when combining optimal and sub-optimal datasets (Figure 6), the presence of sub-optimal demonstrations can negatively impact SR-Reward since the training process treats optimal and sub-optimal demonstrations equally. Enhancing the ability to control the influence of demonstrations based on their quality could lead to higher-quality rewards and more data-efficient learning, offering a promising direction for future research.

Given that the successor representation is closely linked to occupancy measures and state-action distributions, the SR-Reward function proposed here can be employed to approximate the state-action distributions of both expert and non-expert actors. This paves the way for developing new algorithms in imitation learning (IL) and inverse reinforcement learning (IRL), enabling the direct matching of distributions using an approximate model of state-action distributions. We consider this an exciting direction for further exploration and future research.

Broader Impact Statement

The ability to learn from demonstrations enables users without technical knowledge to program agents such as industrial or household robots. This technology can have great potential for automating tasks where the industry is facing a labor shortage or where the safety of humans is of concern. On the other hand, such technology can accelerate the loss of jobs due to automation, a trend that raises concerns for many. So far automation in the physical world especially outside of repetitive motions of industrial robots has been limited, however, this can change through further development and deployment of systems that can easily and flexibly learn from a handful of demonstrations.

References

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1577–1594. PMLR, 23–29 Jul 2023.

Diana Borsa, André Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Rémi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Comput.*, 5(4):613–624, 1993.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, Conference Track Proceedings*. OpenReview.net, 2018.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2052–2062. PMLR, 2019.

Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, 2021, virtual*, pp. 4028–4039, 2021.

Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent RL without entropy. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, 2023*. OpenReview.net, 2023.

Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods, 2019.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

Jianxiong Li Zhuoran Yang Zhaoran Wang Victor Wai Kin Chan Xianyuan Zhan Haoran Xu, Li Jiang. Offline rl with no ood actions: In-sample learning via implicit value regularization. In *International Conference on Learning Representations*, 2023.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. NIPS'16, pp. 4572–4580, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Gabriel Kalweit, Maria Hügle, Moritz Werling, and Joschka Boedecker. Deep inverse q-learning with constraints. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization, 2020.

Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Hyg-JC4FDr`.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=68n2s9ZJWF8`.

Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. Deep successor reinforcement learning. *CoRR*, abs/1606.02396, 2016.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Minghuan Liu, Tairan He, Minkai Xu, and Weinan Zhang. Energy-based imitation learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pp. 809–817, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073.

Yuping Luo, Huazhe Xu, and Tengyu Ma. Learning self-correctable policies and value functions from demonstrations with negative sampling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5125–5133, 2020.

Ted Moskovitz, Spencer R Wilson, and Maneesh Sahani. A first-occupancy representation for reinforcement learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=JBAZe2yN6Ub`.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections, 2019.

Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In Pat Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 663–670. Morgan Kaufmann, 2000.

Tianwei Ni, Benjamin Eysenbach, Erfan Seyedsalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive rl, 2024.

Dean Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Comput.*, 3(1):88–97, 1991. doi: 10.1162/NECO.1991.3.1.88.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

Siddharth Reddy, Anca D. Dragan, and Sergey Levine. SQIL: imitation learning via reinforcement learning with sparse rewards. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

Harshit Sikchi, Qinqing Zheng, Amy Zhang, and Scott Niekum. Dual rl: Unification and new methods for reinforcement and imitation learning, 2023.

Satinder Singh, R. Lewis, and A. Barto. Where do rewards come from? 01 2009.

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. 1999. URL `https://api.semanticscholar.org/CorpusID:1115058`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4235–4246, 2020.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *Conference on Robot Learning*, 2022.

Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments, 2017.

A Occupancy Measure and Successor Representations

We will restrict ourselves to the occupancy measure of the state only (instead of state and action). The extension to state and action is trivial via a second summation over the actions.

The expectation is with respect to starting state distribution μ_0, the policy that is followed π, and the transition dynamics of the environment $\mathcal{T}$.

We can write the definitions of occupancy measures $\rho(s)$ and successor representations $M(s, s')$ in terms of probabilities p.

$$\begin{aligned} M(s,s') &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s')|s_0 = s] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathbb{I}(s_t = s')|s_0 = s] \\ &= \sum_{t=0}^{\infty} \gamma^t p(s_t = s'|s_0 = s) \end{aligned} \tag{11}$$

and similarly for the occupancy measure $\rho(s)$:

$$\begin{aligned} \rho(s) &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s)] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathbb{I}(s_t = s)] \\ &= \sum_{t=0}^{\infty} \gamma^k p(s_t = s) \end{aligned} \tag{12}$$

Below we show that $\rho(s') = \sum_s p(s) M(s, s')$:

$$\begin{aligned} \rho(s') &= \sum_{t=0}^{\infty} \gamma^t p(s_t = s') \\ &= \sum_{t=0}^{\infty} \gamma^t \sum_s p(s) p(s_t = s'|s_0 = s) \\ &= \sum_s p(s) \sum_{t=0}^{\infty} \gamma^t p(s_t = s'|s_0 = s) \\ &= \sum_s p(s) M(s, s') \end{aligned} \tag{13}$$