## A. Detailed Proofs.

### A.1. Observation 1

**Observation 1** *The objective $J(\hat{r})$ is minimized when $\hat{r}(s,a) = \frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}$ for all state-action pairs $(s,a)$.*

Where

$$\min_{\hat{r}(s,a)\forall(s,a)} J(\hat{r}) := \frac{1}{2}\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\hat{r}(s,a)^2\right] - (1-\gamma)\mathbb{E}_{s_0,a_0}\left[\hat{Q}^\pi(s_0,a_0)\right] \tag{1}$$

$$= \frac{1}{2}\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\hat{r}(s,a)^2\right] - \mathbb{E}_{(s,a)\sim d^\pi}\left[\hat{r}(s,a)\right]. \tag{2}$$

*Proof.*

Take the partial derivative of $J(\hat{r})$ with respect to $\hat{r}(s,a)$:

$$\frac{\partial}{\partial\hat{r}(s,a)}\left(\frac{1}{2}\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\hat{r}(s,a)^2\right] - \mathbb{E}_{(s,a)\sim d^\pi}\left[\hat{r}(s,a)\right]\right) = d^\mathcal{D}(s,a)\hat{r}(s,a) - d^\pi(s,a). \tag{3}$$

Then setting $\frac{\partial J(\hat{r})}{\partial\hat{r}(s,a)} = 0$, we have that $J(\hat{r})$ is minimized when $\hat{r}(s,a) = \frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}$ for all state-action pairs $(s,a)$.

■

### A.2. Theorem 1

**Theorem 1** *Equation (5) is the optimal solution to Equation (4) and is equal to $\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}$.*

Where

$$\min_{\hat{r}(s,a)\forall(s,a)} J_\Psi(\hat{r}) := \frac{1}{2}\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\hat{r}(s,a)^2\right] - (1-\gamma)\mathbb{E}_{s_0}\left[\sum_s \Psi^\pi(s|s_0)\mathbb{E}_{a\sim\pi}\left[\hat{r}(s,a)\right]\right]. \tag{4}$$

and

$$\hat{r}^*(s,a) = (1-\gamma)\frac{|\mathcal{D}|}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)}\mathbb{E}_{s_0}[\pi(a|s)\Psi^\pi(s|s_0)]. \tag{5}$$

*Proof.*

First we consider the relationship between Equation (4) and Equation (5). Take the gradient of Equation (4):

$$\nabla_{\hat{r}(s,a)}J_\Psi(\hat{r}) := d^\mathcal{D}(s,a)\hat{r}(s,a) - (1-\gamma)\mathbb{E}_{s_0}[\Psi^\pi(s|s_0)\pi(a|s)]. \tag{6}$$

Where we can replace $d^\mathcal{D}(s,a)$ with $\frac{1}{|\mathcal{D}|}\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)$:

$$\nabla_{\hat{r}(s,a)}J_\Psi(\hat{r}) := \frac{1}{|\mathcal{D}|}\left(\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)\right)\hat{r}(s,a) - (1-\gamma)\mathbb{E}_{s_0}[\Psi^\pi(s|s_0)\pi(a|s)]. \tag{7}$$

Setting the above gradient equal to 0 and solving for $\hat{r}(s,a)$ we have the optimizer of $J_\Psi(\hat{r})$.

$$\hat{r}^*(s,a) = (1-\gamma)\frac{|\mathcal{D}|}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)}\mathbb{E}_{s_0}[\pi(a|s)\Psi^\pi(s|s_0)]. \tag{8}$$

Now consider the relationship between Equation (5) and $\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}$. Recall the definition of the state occupancy $d^\pi(s,a)$:

$$d^\pi(s,a) = (1-\gamma)\sum_{t=0}^\infty \gamma^t \int_{s_0} d_0(s_0)p_\pi(s_0\to s,t)\pi(a|s)ds_0 \tag{9}$$

$$= (1-\gamma)\mathbb{E}_{s_0}\left[\sum_{t=0}^\infty \gamma^t p_\pi(s_0\to s,t)\pi(a|s)\right]. \tag{10}$$

Now consider the SR:

$$\Psi^\pi(s|s_0) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{1}(s_t = s)|s_0\right] \tag{11}$$

$$= \sum_{t=0}^{\infty}\gamma^t p_\pi(s_0 \to s, t). \tag{12}$$

It follows that the SR and the state occupancy share the relationship:

$$d^\pi(s,a) = (1-\gamma)\mathbb{E}_{s_0}[\Psi^\pi(s|s_0)\pi(s,a)]. \tag{13}$$

Finally recall that $d^{\mathcal{D}}(s,a) = \frac{1}{|\mathcal{D}|}\sum_{(s',a')\in\mathcal{D}} \mathbb{1}(s'=s, a'=a)$. Note in this case, the relationship is exact and does not rely on an expectation. It follows that:

$$\hat{r}^*(s,a) = (1-\gamma)\frac{|\mathcal{D}|}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)}\mathbb{E}_{s_0}[\pi(a|s)\Psi^\pi(s|s_0)] \tag{14}$$

$$= \frac{d^\pi(s,a)}{d^{\mathcal{D}}(s,a)}. \tag{15}$$

■

### A.3. Theorem 2

**Theorem 2** *Let $\bar{r}(s,a)$ be the average reward in the dataset $\mathcal{D}$ at the state-action pair $(s,a)$. Let $\hat{\Psi}$ be any approximate SR. The direct SR estimator $(1-\gamma)\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\sum_{s\in\mathcal{S}}\hat{\Psi}(s|s_0)\sum_{a\in\mathcal{A}}\pi(a|s)\bar{r}(s,a)$ of $R(\pi)$ is identical to the MIS estimator $\frac{1}{|\mathcal{D}|}\sum_{(s,a)\in\mathcal{D}} r^*(s,a)r(s,a)$.*

*Proof.*

First we will derive the SR estimator for $R(\pi)$. Define $\bar{r}(s,a)$ as the average of all $r(s,a)$ in the dataset $\mathcal{D}$:

$$\bar{r}(s,a) = \begin{cases} \sum_{r(s,a)\in\mathcal{D}} \frac{r(s,a)}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)} & \text{if } (s,a)\in\mathcal{D} \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Recall by definition $V^\pi(s) = \sum_{s'}\Psi^\pi(s'|s)\mathbb{E}_{a'\sim\pi}[r(s',a')]$. It follows that $\sum_{s'}\Psi^\pi(s'|s)\sum_{a'\in\mathcal{A}}\pi(a'|s')\bar{r}(s',a')$ is an unbiased estimator of $V^\pi(s)$. It follows that estimating $R(\pi) = (1-\gamma)\mathbb{E}_{s_0}[V^\pi(s_0)]$ with the SR would be computed by

$$(1-\gamma)\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\sum_{s\in\mathcal{S}}\Psi^\pi(s|s_0)\sum_{a\in\mathcal{A}}\pi(a|s)\bar{r}(s,a). \tag{17}$$

Now consider the SR-DICE estimator for $R(\pi)$. By expanding and simplifying we arrive at the SR estimator for $R(\pi)$:

$$\frac{1}{|\mathcal{D}|}\sum_{(s,a,r(s,a))\in\mathcal{D}} r^*(s,a)r(s,a) \tag{18}$$

$$= \frac{1}{|\mathcal{D}|}\sum_{(s,a,r(s,a))\in\mathcal{D}}(1-\gamma)\frac{|\mathcal{D}|}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)}\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\pi(a|s)\Psi^\pi(s|s_0)r(s,a) \tag{19}$$

$$= (1-\gamma)\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\sum_{(s,a,r(s,a))\in\mathcal{D}}\Psi^\pi(s|s_0)\pi(a|s)\frac{1}{\sum_{(s',a')\in\mathcal{D}}\mathbb{1}(s'=s,a'=a)}r(s,a) \tag{20}$$

$$= (1-\gamma)\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\sum_{s\in\mathcal{S}}\Psi^\pi(s|s_0)\sum_{a\in\mathcal{A}}\pi(a|s)\bar{r}(s,a). \tag{21}$$

■

## A.4. Derivation of $\mathbf{w}^*$

Recall the optimization objective $J(\mathbf{w})$:

$$\min_{\mathbf{w}} J(\mathbf{w}) := \frac{1}{2|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} \left[(\mathbf{w}^\top \phi(s,a))^2\right] - (1-\gamma)\frac{1}{|\mathcal{D}_0|} \sum_{s_0\in\mathcal{D}_0, a_0} \pi(a_0|s_0)\mathbf{w}^\top \psi^\pi(s_0,a_0). \tag{22}$$

Let:

- $\Phi$ be a $|\mathcal{D}| \times F$ matrix where each row is the feature vector $\phi(s,a)$ with $F$ features.
- $\Psi$ be a $|\mathcal{D}_0||\mathcal{A}| \times F$ matrix where each row is $\pi(a_0|s_0)\psi^\pi(s_0,a_0)$, the SR weighted by its probability under the policy.
- $\mathbf{1}$ be a $|\mathcal{D}_0||\mathcal{A}|$ dimensional vector of all $1$.

We can reformulate Equation (22) as the following:

$$\frac{1}{2|\mathcal{D}|}(\Phi\mathbf{w})^\top \Phi\mathbf{w} - (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi\mathbf{w}. \tag{23}$$

Now take the gradient:

$$\begin{aligned} &\nabla_{\mathbf{w}} \left(\frac{1}{2|\mathcal{D}|}(\Phi\mathbf{w})^\top \Phi\mathbf{w} - (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi\mathbf{w}\right) &(24)\\ &= \frac{1}{|\mathcal{D}|}\mathbf{w}^\top \Phi^\top \Phi - (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi &(25)\end{aligned}$$

And set it equal to 0 to solve for $\mathbf{w}^*$:

$$\begin{aligned} \frac{1}{|\mathcal{D}|}\mathbf{w}^\top \Phi^\top \Phi &- (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi = 0 &(26)\\ \frac{1}{|\mathcal{D}|}\mathbf{w}^\top \Phi^\top \Phi &= (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi &(27)\\ \Phi^\top \Phi\mathbf{w} &= (1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{D}_0|}\Psi^\top \mathbf{1} &(28)\\ \mathbf{w}^* &= (1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{D}_0|}(\Phi^\top \Phi)^{-1}\Psi^\top \mathbf{1}. &(29)\end{aligned}$$

■

## A.5. Theorem 3

**Theorem 3** *If the deep SR is exact, such that* $(1-\gamma)\mathbb{E}_{s_0,a_0}\left[\psi^\pi(s_0,a_0)\right] = \mathbb{E}_{(s,a)\sim d^\pi}[\phi(s,a)]$*, and the support of* $d^\pi$ *is contained in the dataset* $\mathcal{D}$*, then the optimizer* $\mathbf{w}^*$ *of Equation (30), as defined by Equation (31), is the least squares estimator of* $\sum_{(s,a)\in\mathcal{D}}\left(\mathbf{w}^\top \phi(s,a) - \frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}\right)^2$.

Where

$$\min_{\mathbf{w}} J(\mathbf{w}) := \frac{1}{2|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} \left[(\mathbf{w}^\top \phi(s,a))^2\right] - (1-\gamma)\frac{1}{|\mathcal{D}_0|} \sum_{s_0\in\mathcal{D}_0, a_0} \pi(a_0|s_0)\mathbf{w}^\top \psi^\pi(s_0,a_0). \tag{30}$$

and (as derived in Subsection A.4)

$$\mathbf{w}^* = (1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{D}_0|}(\Phi^\top \Phi)^{-1}\Psi^\top \mathbf{1}_{|\mathcal{D}_0||\mathcal{A}|}. \tag{31}$$

*Proof.*

Let:

- $\Phi$ be a $|\mathcal{D}| \times F$ matrix where each row is the feature vector $\phi(s,a)$ with $F$ features.
- $\Psi$ be a $|\mathcal{D}_0||\mathcal{A}| \times F$ matrix where each row is the SR weighted by its probability under the policy $\pi(a_0|s_0)\psi^\pi(s_0,a_0)$.
- $\mathbf{1}_x$ be a $x$ dimensional vector of all $1$.
- $d^\pi$ and $d^\mathcal{D}$ be diagonal $|\mathcal{D}| \times |\mathcal{D}|$ matrices where the diagonal entries contain $d^\pi(s,a)$ and $d^\mathcal{D}(s,a)$ respectively, for each state-action pair $(s,a)$ in the dataset $\mathcal{D}$.

First note the least squares estimator of $\sum_{(s,a)\in\mathcal{D}} \left(\mathbf{w}^\top\phi(s,a) - \frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}\right)^2$ is $(\Phi^\top\Phi)^{-1}\Phi^\top \frac{d^\pi}{d^\mathcal{D}}\mathbf{1}_{|\mathcal{D}|}$, where the division is element-wise.

By our assumption on the deep SR, we have that:

$$\begin{aligned}(1-\gamma)\mathbb{E}_{s_0,a_0}\left[\psi^\pi(s_0,a_0)\right] &= \mathbb{E}_{(s,a)\sim d^\pi}\left[\phi(s,a)\right] && (32)\\ &= \mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}\phi(s,a)\right]. && (33)\end{aligned}$$

and therefore:

$$(1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top_{|\mathcal{D}_0||\mathcal{A}|}\Psi = \frac{1}{|\mathcal{D}|}\mathbf{1}^\top_{|\mathcal{D}|}\frac{d^\pi}{d^\mathcal{D}}\Phi. \tag{34}$$

It follows that we can simplify $\mathbf{w}^*$:

$$\begin{aligned}\mathbf{w}^* &= (1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{D}_0|}(\Phi^\top\Phi)^{-1}\Psi^\top\mathbf{1}_{|\mathcal{D}_0||\mathcal{A}|} && (35)\\ &= |\mathcal{D}|(\Phi^\top\Phi)^{-1}\left((1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top_{|\mathcal{D}_0||\mathcal{A}|}\Psi\right)^\top && (36)\\ &= |\mathcal{D}|(\Phi^\top\Phi)^{-1}\left(\frac{1}{|\mathcal{D}|}\mathbf{1}^\top_{|\mathcal{D}|}\frac{d^\pi}{d^\mathcal{D}}\Phi\right)^\top && (37)\\ &= (\Phi^\top\Phi)^{-1}\Phi^\top\frac{d^\pi}{d^\mathcal{D}}\mathbf{1}_{|\mathcal{D}|}. && (38)\end{aligned}$$

■

## A.6. Theorem 4

**Theorem 4** *Given the least squares estimator $\mathbf{w}_{SR}$ of $\sum_{(s,a)\in\mathcal{D}}\left(\mathbf{w}^\top\phi(s,a) - r(s,a)\right)^2$ and the optimizer $\mathbf{w}^*$ of Equation (39), as defined by Equation (40), then the traditional SR estimator $\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0}\mathbf{w}_{SR}^\top\psi^\pi(s_0,a_0)$ of $R(\pi)$ is identical to the SR-DICE estimator $\frac{1}{|\mathcal{D}|}\sum_{(s,a,r(s,a))\in\mathcal{D}}\mathbf{w}^{*\top}\phi(s,a)r(s,a)$ of $R(\pi)$.*

Where

$$\min_\mathbf{w} J(\mathbf{w}) := \frac{1}{2|\mathcal{D}|}\sum_{(s,a)\in\mathcal{D}}\left[(\mathbf{w}^\top\phi(s,a))^2\right] - (1-\gamma)\frac{1}{|\mathcal{D}_0|}\sum_{s_0\in\mathcal{D}_0,a_0}\pi(a_0|s_0)\mathbf{w}^\top\psi^\pi(s_0,a_0). \tag{39}$$

and (as derived in Subsection A.4)

$$\mathbf{w}^* = (1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{S}_0|}(\Phi^\top\Phi)^{-1}\Psi^\top\mathbf{1}. \tag{40}$$

*Proof.*

Let:

- $\Phi$ be a $|\mathcal{D}| \times F$ matrix where each row is the feature vector $\phi(s,a)$ with $F$ features.
- $\Psi$ be a $|\mathcal{D}_0||\mathcal{A}| \times F$ matrix where each row is the SR weighted by its probability under the policy $\pi(a_0|s_0)\psi^\pi(s_0,a_0)$.
- $\mathbf{1}$ be a $|\mathcal{D}_0||\mathcal{A}|$ dimensional vector of all $1$.
- $R$ be the $\mathcal{D}$ dimensional vector of each reward $r(s,a)$ in the dataset $\mathcal{D}$.

First note the least squares solution for direct SR, where $\mathbf{w}_{\text{SR}}$ is optimized to reduce the mean squared error between $\mathbf{w}_{\text{SR}}\phi(s,a)$ and $r(s,a)$:

$$\mathbf{w}_{\text{SR}} = (\Phi^\top \Phi)^{-1}\Phi^\top R. \tag{41}$$

It follows that the direct SR solution to $R(\pi)$ is:

$$(1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi \mathbf{w}_{\text{SR}} = (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top \Psi (\Phi^\top\Phi)^{-1}\Phi^\top R. \tag{42}$$

Now consider the SR-DICE solution to $R(\pi)$:

$$\begin{aligned}
\frac{1}{|\mathcal{D}|}(\Phi\mathbf{w}^*)^\top R &= \frac{1}{|\mathcal{D}|}\mathbf{w}^{*\top}\Phi^\top R && (43)\\
&= \frac{1}{|\mathcal{D}|}\left((1-\gamma)\frac{|\mathcal{D}|}{|\mathcal{D}_0|}(\Phi^\top\Phi)^{-1}\Psi^\top\mathbf{1}\right)^\top \Phi^\top R && (44)\\
&= (1-\gamma)\frac{1}{|\mathcal{D}_0|}\mathbf{1}^\top\Psi(\Phi^\top\Phi)^{-1}\Phi^\top R. && (45)
\end{aligned}$$

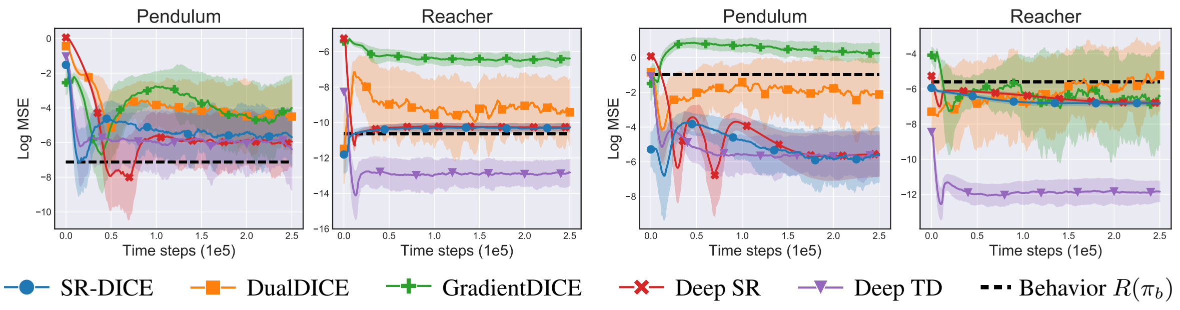
■

## B. Additional Experiments

In this section, we include additional experiments and visualizations, covering extra domains, additional ablation studies, run time experiments and additional behavior policies in the Atari domain.

### B.1. Extra Continuous Domains

Although our focus is on high-dimensional domains, the environments, Pendulum and Reacher, have appeared in several related MIS papers (Nachum et al., 2019; Zhang et al., 2020a). Therefore, we have included results for these domains in Figure 1. All experimental settings match the experiments in the main body, and are described fully in Appendix E.



(a) *Easy* setting (500k time steps and $\sigma_b = 0.133$)

(b) *Hard* setting (50k time steps, $\sigma_b = 0.2$, random actions with $p = 0.2$)

Figure 1: Off-policy evaluation results for Pendulum and Reacher. The shaded area captures one standard deviation across 10 trials. Even on these easier environment, we find that SR-DICE outperforms the baseline MIS methods.

### B.2. Representation Learning & MIS

SR-DICE relies a disentangled representation learning phase where an encoding $\phi$ is learned, followed by the deep successor representation $\psi^\pi$ which are used with a linear vector $\mathbf{w}$ to estimate the density ratios. In this section we perform some experiments which attempt to evaluate the importance of representation learning by comparing their influence on the baseline MIS methods.

**Alternate representations.** We examine both DualDICE (Nachum et al., 2019) and GradientDICE (Zhang et al., 2020c) under four settings where we pass the representations $\phi$ and $\psi^\pi$ to their networks, where both $\phi$ and $\psi^\pi$ are learned in identical fashion to SR-DICE.

| | | |
|---|---|---|
| (1) Input encoding $\phi$, | $f(\phi(s,a))$, | $w(\phi(s,a))$. |
| (2) Input SR $\psi^\pi$, | $f(\psi^\pi(s,a))$, | $w(\psi^\pi(s,a))$. |
| (3) Input encoding $\phi$, linear networks, | $f^\top \phi(s,a)$, | $w^\top \phi(s,a)$. |
| (4) Input SR $\psi^\pi$, linear networks, | $f^\top \psi^\pi(s,a)$, | $w^\top \psi^\pi(s,a)$. |

See Appendix D for specific details on the baselines. We report the results in Figure 2. For GradientDICE, no benefit is provided by varying the representations, although using the encoding $\phi$ matches the performance of vanilla GradientDICE regardless of the choice of network, providing some validation that $\phi$ is a reasonable encoding. Interestingly, for DualDICE, we see performance gains from using the SR $\psi^\pi$ as a representation: slightly as input, but significantly when used with linear networks. On the other hand, as GradientDICE performs much worse with the SR, it is clear that the SR cannot be used as a representation without some degree of forethought.

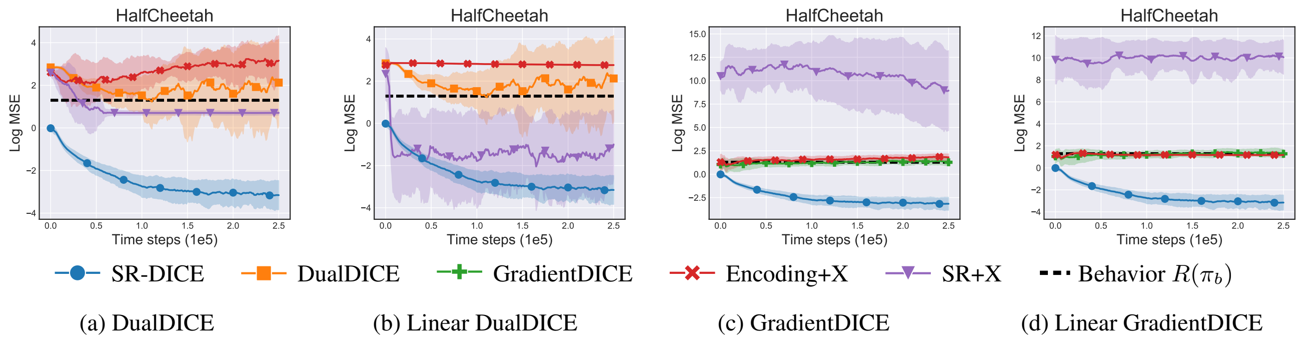


Figure 2: Off-policy evaluation results on HalfCheetah examining the value of differing representations added to the baseline MIS methods. The experimental setting corresponds to the *hard* setting from the main body. The shaded area captures one standard deviation across 10 trials. We see that using the SR $\psi^\pi$ as a representation improves the performance of DualDICE. On the other hand, GradientDICE performs much worse when using the SR, suggesting it cannot be used naively to improve MIS methods.

**Increased capacity.** As SR-DICE uses a linear function on top of a representation trained with the same capacity as the networks in DualDICE and GradientDICE, our next experiment examines if this additional capacity provides benefit to the baseline methods. To do, we expand each network in both baselines by adding an additional hidden layer. The results are reported in Figure 3. We find there is a very slight decrease in performance when using the larger capacity networks. This suggests the performance gap from SR-DICE over the baseline methods has little to do with model size.

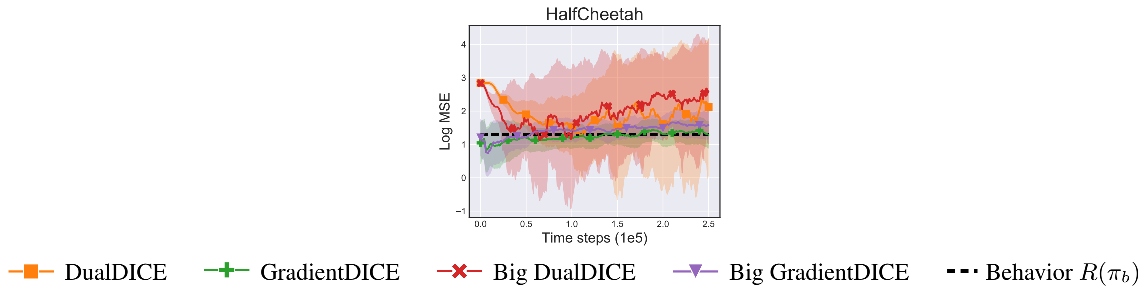


Figure 3: Off-policy evaluation results on HalfCheetah evaluating the performance benefits from larger network capacity on the baseline MIS methods. "Big" refers to the models with an additional hidden layer. The experimental setting corresponds to the *hard* setting from the main body. The shaded area captures one standard deviation across 10 trials. We find that there is no clear performance benefit from increasing network capacity.

## B.3. Toy Domains

We additional test the MIS algorithms on a toy random-walk experiment with varying feature representations, based on a domain from (Sutton et al., 2009).

**Domain.** The domain is a simple 5-state MDP $(x_1, x_2, x_3, x_4, x_5)$ with two actions $(a_0, a_1)$, where action $a_0$ induces the transition $x_i \to x_{i-1}$ and action $a_1$ induces the transition $x_i \to x_{i+1}$, with the state $x_1$ looping to itself with action $a_0$ and $x_5$ looping to itself with action $a_5$. Episodes begin in the state $x_1$.

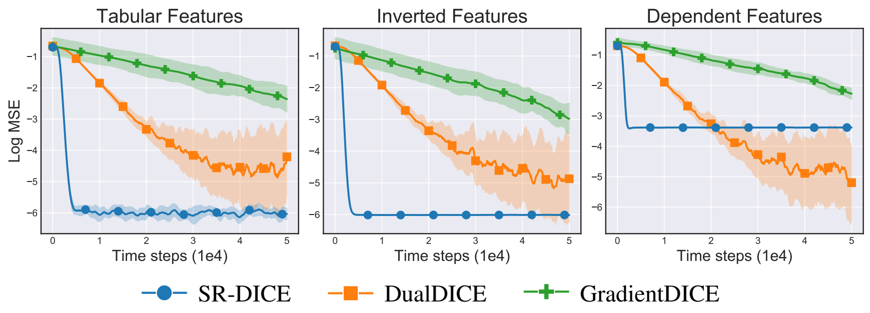
**Target.** We evaluate policy $\pi$ which selects actions uniformly, i.e. $\pi(a_0|x_i) = \pi(a_1|x_i) = 0.5$ for all states $x_i$. Our dataset $\mathcal{D}$ contains all 10 possible state-action pairs and is sampled uniformly. We use a discount factor of $\gamma = 0.99$. Methods are evaluated on the average MSE between their estimate of $\frac{d^\pi}{d^\mathcal{D}}$ on all state-action pairs and the ground-truth value, which is calculated analytically.

**Hyper-parameters.** Since we are mainly interested in a function approximation setting, each method uses a small neural network with two hidden layers of 32, followed by tanh activation functions. All networks used stochastic gradient descent with a learning rate $\alpha$ tuned for each method out of $\{1, 0.5, 0.1, 0.05, 0.01, 0.001\}$. This resulted in $\alpha = 0.05$ for DualDICE, $\alpha = 0.1$ for GradientDICE, and $\alpha = 0.05$ for SR-DICE. Although there are a small number of possible data points, we use a batch size of 128 to resemble the regular training procedure. As recommended by the authors we use $\lambda = 1$ for GradientDICE (Zhang et al., 2020c), which was not tuned. For SR-DICE, we update the target network at every time step $\tau = 1$, which was not tuned.

Since there are only 10 possible state-action pairs, we use the closed form solution for the vector $\mathbf{w}$. Additionally, we skip the state representation phase of SR-DICE, instead learning the SR $\psi^\pi$ over the given representation of each state, such that the encoding $\phi = x$. This allows us to test SR-DICE to a variety of representations rather than using a learned encoding. Consequently, with these choices, SR-DICE has no pre-training phase, and therefore, unlike every other graph in this paper, we report the results as the SR is trained, rather than as the vector $\mathbf{w}$ is trained.

**Features.** To test the robustness of each method we examine three versions of the toy domain, each using a different feature representation over the same 5-state MDP. These feature sets are again taken from (Sutton et al., 2009).

- Tabular features: states are represented by a one-hot encoding, for example $x_2 = [0, 1, 0, 0, 0]$.
- Inverted features: states are represented by the inverse of a one-hot encoding, for example $x_2 = [\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$.
- Dependent features: states are represented by 3 features which is not sufficient to cover all states exactly. In this case $x_1 = [1, 0, 0]$, $x_2 = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0]$, $x_3 = [\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}]$, $x_4 = [0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$, $x_5 = [0, 0, 1]$. Since our experiments use neural networks rather than linear functions, this representation is mainly meant to test SR-DICE, where we skip the state representation phase for SR-DICE and use the encoding $\phi = x$, limiting the representation of the SR.



(a) Tabular Features (b) Inverted Features (c) Dependent Features

Figure 4: Results measuring the log MSE between the estimated density ratio and the ground-truth on a simple 5-state MDP domain with three feature sets. The shaded area captures one standard deviation across 10 trials. Results are evaluated every 100 time steps over 50k time steps total.

**Results.** We report the results in Figure 4. We remark on several observations. SR-DICE learns significantly faster than the baseline methods, likely due to its use of temporal difference methods in the SR update, rather than using an update similar to residual learning, which is notoriously slow (Baird, 1995; Zhang et al., 2020b). GradientDICE appears to still be improving, although we limit training at 50k time steps, which we feel is sufficient given the domain is deterministic and only has $5$ states. Notably, GradientDICE also uses a higher learning rate than SR-DICE and DualDICE. We also find the final performance of SR-DICE is much better than DualDICE and GradientDICE in the domains where the feature representation is not particularly destructive, highlighting the easier optimization of SR-DICE. In the case of the dependent features, we find DualDICE outperforms SR-DICE after sufficient updates. However, we remark that this concern could likely be resolved by learning the features and that SR-DICE still outperforms GradientDICE. Overall, we believe these results demonstrate that SR-DICE's strong empirical performance is consistent across simpler domains as well as the high-dimensional domains we examine in the main body.

## B.4. Run Time Experiments

In this section, we evaluate the run time of each algorithm used in our experiments. Although SR-DICE relies on pre-training the deep successor representation before learning the density ratios, we find each marginalized importance sampling (MIS) method uses a similar amount of compute, due to the reduced cost of training $\mathbf{w}$ after the pre-training phase.

We evaluate the run time on the HalfCheetah environment in MuJoCo (Todorov et al., 2012) and OpenAI gym (Brockman et al., 2016). As in the main set of experiments, each method is trained for 250k time steps. Additionally, SR-DICE and Deep SR train the encoder-decoder for 30k time steps and the deep successor representation for 100k time steps before training $\mathbf{w}$. Run time is averaged over 3 seeds. All time-based experiments are run on a single GeForce GTX 1080 GPU and a Intel Core i7-6700K CPU. Results are reported in Figure 5.

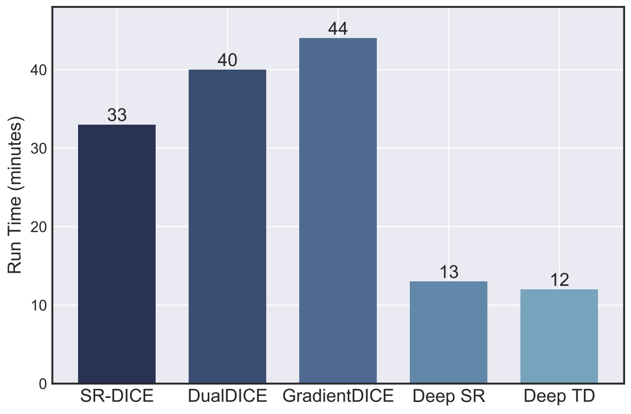


Figure 5: The average run time of each off-policy evaluation approach in minutes. Each experiment is run for 250k time steps and is averaged over 3 seeds. SR-DICE and Deep SR pre-train encoder-decoder for 30k time steps and the deep successor representation 100k time steps.

We find the MIS algorithms run in a comparable time, regardless of the pre-training step involved in SR-DICE. This can be explained as training $\mathbf{w}$ in SR-DICE involves significantly less compute than DualDICE and GradientDICE which update multiple networks. On the other hand, the deep reinforcement learning approaches run in about half the time of SR-DICE.

## B.5. Atari Experiments

To better evaluate the algorithms in the Atari domain, we run two additional experiments where we swap the behavior policy. We observe similar trends as the experiments in the main body of the paper. In both experiments we keep all other settings fixed. Notably, we continue to use the same target policy, corresponding to the greedy policy trained by Double DQN (Van Hasselt et al., 2016), the same discount factor $\gamma = 0.99$, and the same dataset size of $1$ million.

**Increased noise.** In our first experiment, we attempt to increase the randomness of the behavior policy. As this can cause destructive behavior in the performance of the agent, we adopt an episode-dependent policy which selects between the noisy policy or the deterministic greedy policy at the beginning of each episode. This is motivated by the offline deep reinforcement learning experiments from (Fujimoto et al., 2019). As a result, we use an $\epsilon$-greedy policy with $p = 0.8$ and the deterministic greedy policy (the target policy) with $p = 0.2$. $\epsilon$ is set to $0.2$, rather than $0.1$ as in the experiments in the main body of the paper. Results are reported in Figure 6.

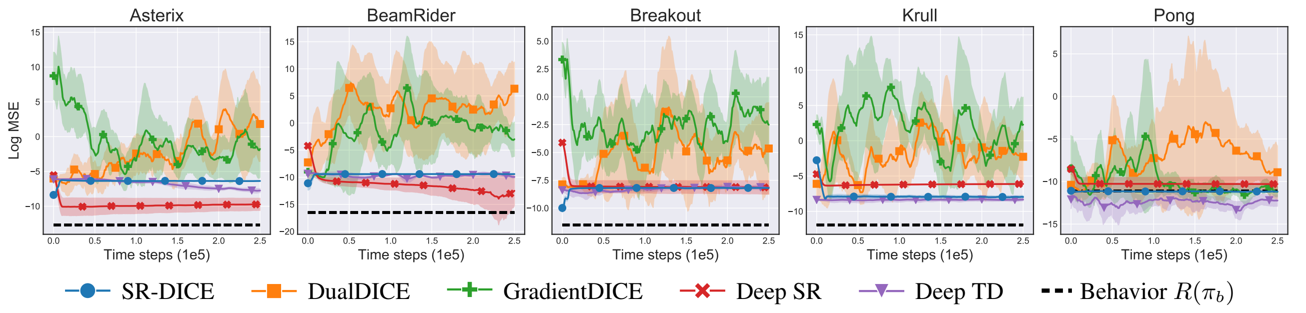


Figure 6: We plot the log MSE for off-policy evaluation in the image-based Atari domain, using an episode-dependent noisy policy, where $\epsilon = 0.2$ with $p = 0.8$ and $\epsilon = 0$ with $p = 0.2$. This episode-dependent selection ensures sufficient state-coverage while using a stochastic policy. The shaded area captures one standard deviation across 3 trials. Markers are not placed at every point for visual clarity.

We observe very similar trends to the original set of experiments. Again, we note DualDICE and GradientDICE perform very poorly, while SR-DICE, Deep SR, and Deep TD achieve a reasonable, but biased, performance. In this setting, we still find the behavior policy is the closest estimate of the true value of $R(\pi)$ .

**Separate behavior policy.** In this experiment, we use a behavior which is distinct from the target policy, rather than simply adding noise. This behavior policy is derived from an agent trained with prioritized experience replay and Double DQN (Schaul et al., 2016; Fujimoto et al., 2020). Again, we use a $\epsilon$-greedy policy, with $\epsilon = 0.1$. We report the results in Figure 7.

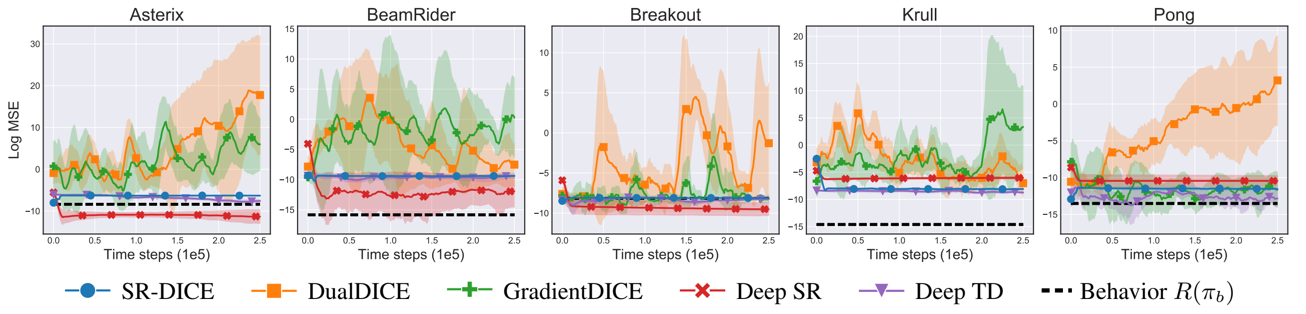


Figure 7: We plot the log MSE for off-policy evaluation in the image-based Atari domain, using a distinct behavior policy, trained by a separate algorithm, from the target policy. This experiment tests the ability to generalize to a more off-policy setting. The shaded area captures one standard deviation across 3 trials. Markers are not placed at every point for visual clarity.

Again, we observe similar trends in performance. Notably, in the Asterix game, the performance of Deep SR surpasses the behavior policy, suggesting off-policy evaluation can outperform the naïve estimator in settings where the policy is sufficiently "off-policy" and distinct.

## C. SR-DICE Practical Details

In this section, we cover some basic implementation-level details of SR-DICE. Note that code is provided for additional clarity.

SR-DICE uses two parametric networks, an encoder-decoder network to learn the encoding $\phi$ and a deep successor representation network $\psi^\pi$. Additionally, SR-DICE uses the weights of a linear function $\mathbf{w}$. SR-DICE begins by pre-training the encoder-decoder network and the deep successor representation before applying updates to $\mathbf{w}$.

**Encoder-Decoder.** This encoder-decoder network encodes $(s, a)$ to the feature vector $\phi(s, a)$, which is then decoded by several decoder heads. For the Atari domain, we choose to condition the feature vector only on states $\phi(s)$, as the reward is generally independent of the action selection. This change applies to both SR-DICE and Deep SR. Most design decisions

**Algorithm 1** SR-DICE

```
Input: dataset $\mathcal{D}$, target policy $\pi$, number of iterations $T_1$, $T_2$, $T_3$, mini-batch size $N$, target_update_rate.

# Train the encoder-decoder.
for $t = 1$ to $T_1$ do
    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{D}$.
    $\min_{\phi, D_{s'}, D_a, D_r} \lambda_{s'}(D_{s'}(\phi(s,a)) - s')^2$
        $+\lambda_a(D_a(\phi(s,a)) - a)^2 + \lambda_r(D_r(\phi(s,a)) - r)^2$.
end for

# Train the successor representation.
for $t = 1$ to $T_2$ do
    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{D}$.
    Sample $a' \sim \pi(s')$.
    $\min_{\psi^\pi} (\phi(s,a) + \gamma\psi'(s',a') - \psi^\pi(s,a))^2$.
    If $t$ mod target_update_rate $= 0$: $\psi' \leftarrow \psi$.
end for

# Learn $\mathbf{w}$.
for $t = 1$ to $T_3$ do
    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{D}$.
    Sample mini-batch of $N$ start states $s_0$ from $\mathcal{D}$.
    Sample $a_0 \sim \pi(s_0)$.
    $\min_{\mathbf{w}} \frac{1}{2}(\mathbf{w}^\top \phi(s,a))^2 - (1-\gamma)\mathbf{w}^\top \psi^\pi(s_0, a_0)$.
end for
```

are inspired by prior work (Machado et al., 2017; 2018a).

For continuous control, given a mini-batch transition $(s, a, r, s')$, the encoder-decoder network is trained to map the state-action pair $(s, a)$ to the next state $s'$, the action $a$ and reward $r$. The resulting loss function is as follows:

$$\min_{\phi, D_{s'}, D_a, D_r} \mathcal{L}(\phi, D) := \lambda_{s'}(D_{s'}(\phi(s,a)) - s')^2 + \lambda_a(D_a(\phi(s,a)) - a)^2 + \lambda_r(D_r(\phi(s,a)) - r)^2. \tag{46}$$

We use $\lambda_{s'} = 1$, $\lambda_a = 1$ and $\lambda_r = 0.1$.

For the Atari games, given a mini-batch transition $(s, a, r, s')$, the encoder-decoder network is trained to map the state $s$ to the next state $s'$ and reward $r$, while penalizing the size of $\phi(s)$. The resulting loss function is as follows:

$$\min_{\phi, D_{s'}, D_r} \mathcal{L}(\phi, D) := \lambda_{s'}(D_{s'}(\phi(s)) - s')^2 + \lambda_r(D_r(\phi(s)) - r)^2 + \lambda_\phi \phi(s)^2. \tag{47}$$

We use $\lambda_{s'} = 1$, $\lambda_r = 0.1$ and $\lambda_\phi = 0.1$.

**Deep Successor Representation.** The deep successor representation $\psi^\pi$ is trained to estimate the accumulation of $\phi$. The training procedure resembles standard deep reinforcement learning algorithms. Given a mini-batch of transitions $(s, a, r, s')$ the network is trained to minimize the following loss:

$$\min_{\psi^\pi} \mathcal{L}(\psi^\pi) := (\phi(s,a) + \gamma\psi'(s',a') - \psi^\pi(s,a))^2, \tag{48}$$

where $\psi'$ is the target network. A target network is a frozen network used to provide stability (Mnih et al., 2015; Kulkarni et al., 2016) in the learning target. The target network is updated to the current network $\psi' \leftarrow \psi^\pi$ after a fixed number of time steps, or updated with slowly at each time step $\psi' \leftarrow \tau\psi^\pi + (1-\tau)\psi^\pi$ (Lillicrap et al., 2015).

**Marginalized Importance Sampling Weights.** As described in the main body, we learn $\mathbf{w}$ by optimizing the following objective:

$$\min_{\mathbf{w}} J(\mathbf{w}) := \frac{1}{2}\mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}\left[(\mathbf{w}^\top\phi(s,a))^2\right] - (1-\gamma)\mathbb{E}_{s_0, a_0\sim\pi}\left[\mathbf{w}^\top\psi^\pi(s_0,a_0)\right]. \tag{49}$$

This is achieved by sampling state-action pairs uniformly from the dataset $\mathcal{D}$, alongside a mini-batch of start states $s_0$, which are recorded at the beginning of each episode during data collection.

We summarize the learning procedure of SR-DICE in Algorithm 1.

## D. Baselines

In this section, we cover some of the practical details of each of the baseline methods.

### D.1. DualDICE

Dual stationary DIstribution Correction Estimation (DualDICE) (Nachum et al., 2019) uses two networks $f$ and $w$. The general optimization problem is defined as follows:

$$\begin{aligned}\min_f \max_w J(f,w) := \; & \mathbb{E}_{(s,a)\sim d^{\mathcal{D}},a'\sim\pi,s'}\left[w(s,a)(f(s,a)-\gamma f(s',a')) - 0.5w(s,a)^2\right] \\ & -(1-\gamma)\mathbb{E}_{s_0,a_0}[f(s_0,a_0)].\end{aligned} \tag{50}$$

In practice this corresponds to alternating single gradient updates to $f$ and $w$. The authors suggest possible alternative functions to the convex function $0.5w(s,a)^2$ such as $\frac{2}{3}|w(s,a)|^{\frac{3}{2}}$, however in practice we found $0.5w(s,a)^2$ performed the best.

### D.2. GradientDICE

Gradient stationary DIstribution Correction Estimation (GradientDICE) (Zhang et al., 2020c) uses two networks $f$ and $w$, and a scalar $u$. The general optimization problem is defined as follows:

$$\begin{aligned}\min_w \max_{f,u} J(w,u,f) := \; & (1-\gamma)\mathbb{E}_{s_0,a_0}[f(s_0,a_0)] + \gamma\mathbb{E}_{(s,a)\sim d^{\mathcal{D}},a'\sim\pi,s'}[w(s,a)f(s',a')] \\ & -\mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}[w(s,a)f(s,a)] + \lambda\left(\mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}[uw(s,a)-u] - 0.5u^2\right).\end{aligned} \tag{51}$$

Similarly to DualDICE, in practice this involves alternating single gradient updates to $w$, $u$ and $f$. As suggested by the authors we use $\lambda = 1$.

### D.3. Deep SR

Our Deep SR baseline is a policy evaluation version of deep successor representation (Kulkarni et al., 2016). The encoder-decoder network and deep successor representation are trained in exactly the same manner as SR-DICE (see Section C). Then, rather than train $\mathbf{w}$ to learn the marginalized importance sampling ratios, $\mathbf{w}$ is trained to recover the original reward function. Given a mini-batch of transitions $(s,a,r,s')$, the following loss is applied:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) := (r - \mathbf{w}^\top \phi(s,a))^2. \tag{52}$$

### D.4. Deep TD

Deep TD, short for deep temporal-difference learning, takes the standard deep reinforcement learning methodology, akin to DQN (Mnih et al., 2015), and applies it to off-policy evaluation. Given a mini-batch of transitions $(s,a,r,s')$ the Q-network is updated by the following loss:

$$\min_{Q^\pi} \mathcal{L}(Q^\pi) := (r + \gamma Q'(s',a') - Q^\pi(s,a))^2, \tag{53}$$

where $a'$ is sampled from the target policy $\pi(\cdot|s')$. Similarly, to training the deep successor representation, $Q'$ is a frozen target network which is updated to the current network after a fixed number of time steps, or incrementally at every time step.

## E. Experimental Details

All networks are trained with PyTorch (version 1.4.0) (Paszke et al., 2019). Any unspecified hyper-parameter uses the PyTorch default setting.

**Evaluation.** The marginalized importance sampling methods are measured by the average weighted reward from transitions sampled from a replay buffer $\frac{1}{N}\sum_{(s,a,r)} w(s,a)r(s,a)$, with $N = 10$k, while the deep RL methods use $\frac{(1-\gamma)}{M}\sum_{s_0} Q(s_0, \pi(a_0))$, where $M$ is the number of episodes. Each OPE method is trained on data collected by some behavioral policy $\pi_b$. We estimate the "true" normalized average discounted reward of the target and behavior policies from 100 roll-outs in the environment.

## E.1. Continuous Action Environments

Our agents are evaluated via tasks interfaced through OpenAI gym (version 0.17.2) (Brockman et al., 2016), which mainly rely on the MuJoCo simulator (mujoco-py version 1.50.1.68) (Todorov et al., 2012). We provide a description of each environment in Table 1.

Table 1: Continuous action environment descriptions.

| Environment | State dim. | Action dim. | Episode Horizon | Task description |
|---|---|---|---|---|
| Pendulum-v0 | 3 | 1 | 200 | Balance a pendulum. |
| Reacher-v2 | 11 | 2 | 50 | Move end effector to goal. |
| HalfCheetah-v3 | 17 | 6 | 1000 | Locomotion. |
| Hopper-v3 | 11 | 3 | 1000 | Locomotion. |
| Walker2d-v3 | 17 | 6 | 1000 | Locomotion. |
| Ant-v3 | 111 | 8 | 1000 | Locomotion. |
| Humanoid-v3 | 376 | 17 | 1000 | Locomotion. |

**Experiments.** Our experiments are framed as off-policy evaluation tasks in which agents aim to evaluate $R(\pi) = \mathbb{E}_{(s,a)\sim d^\pi, r}[r(s,a)]$ for some target policy $\pi$. In each of our experiments, $\pi$ corresponds to a noisy version of a policy trained by a TD3 agent (Fujimoto et al., 2018), a commonly used deep reinforcement learning algorithm. Denote $\pi_d$, the deterministic policy trained by TD3 using the author's GitHub `https://github.com/sfujim/TD3`. The target policy is defined as: $\pi + \mathcal{N}(0, \sigma^2)$, where $\sigma = 0.1$. The off-policy evaluation algorithms are trained on a dataset generated by a single behavior policy $\pi_b$. The experiments are done with two settings *easy* and *hard* which vary the behavior policy and the size of the dataset. All other settings are kept fixed. For the *easy* setting the behavior policy is defined as:

$$\pi_b = \pi_d + \mathcal{N}(0, \sigma_b^2), \sigma_b = 0.133, \tag{54}$$

and 500k time steps are collected (approximately 500 trajectories for most tasks). The *easy* setting is roughly based on the experimental setting from Zhang et al. (2020a). For the *hard* setting the behavior policy adds an increased noise and selects random actions with $p = 0.2$:

$$\pi_b = \begin{cases} \pi_d + \mathcal{N}(0, \sigma_b^2), \sigma_b = 0.2 & p = 0.8, \\ \text{Uniform random action} & p = 0.2, \end{cases} \tag{55}$$

and only 50k time steps are collected (approximately 50 trajectories for most tasks). For Pendulum-v0 and Humanoid-v3, the range of actions is $[-2, 2]$ and $[-0.4, 0.4]$ respectively, rather than $[-1, 1]$, so we scale the size of the noise added to actions accordingly. We set the discount factor to $\gamma = 0.99$. All continuous action experiments are over 10 seeds.

**Pre-training.** Both SR-DICE and Deep SR rely on pre-training the encoder-decoder and deep successor representation $\psi$. These networks were trained for 30k and 100k time steps respectively. As noted in Section B.4, even when including this pre-training step, both algorithm have a lower running time than DualDICE and GradientDICE.

**Architecture.** For fair comparison, we use the same architecture for all algorithms except for DualDICE. This a fully connected neural network with 2 hidden layers of 256 and ReLU activation functions. This architecture was based on the network defined in the TD3 GitHub and was not tuned. For DualDICE, we found tanh activation functions improved stability over ReLU.

For SR-DICE and SR-Direct we use a separate architecture for the encoder-decoder network. The encoder is a network with a single hidden layer of 256, making each $\phi(s, a)$ a feature vector of 256. There are three decoders for reward, action, and next state, respectively. For the action decoder and next state decoder we use a network with one hidden layer of 256.

The reward decoder is a linear function of the encoding, without biases. All hidden layers are followed by ReLU activation functions.

**Network hyper-parameters.** All networks are trained with the Adam optimizer (Kingma & Ba, 2014). We use a learning rate of $3\text{e}{-4}$, again based on TD3 for all networks except for GradientDICE, which we found required careful tuning to achieve a reasonable performance. For GradientDICE we found a learning rate of $1\text{e}{-5}$ for $f$ and $w$, and $1\text{e}{-2}$ for $u$ achieved the highest performance. For DualDICE we chose the best performing learning rate out of $\{1\text{e}-2, 1\text{e}-3, 3\text{e}-4, 5\text{e}-5, 1\text{e}-5\}$. SR-DICE, Deep SR, and Deep TD were not tuned and use default hyper-parameters from deep RL algorithms. For training $\psi^\pi$ and $Q^\pi$ for the deep reinforcement learning aspects of SR-DICE, Deep SR, and Deep TD we use a mini-batch size of 256 and update the target networks using $\tau = 0.005$, again based on TD3. For all MIS methods, we use a mini-batch size of 2048 as described by (Nachum et al., 2019). We found SR-DICE and DualDICE succeeded with lower mini-batch sizes but did not test this in detail. All hyper-parameters are described in Table 2.

Table 2: Continuous action environment training hyper-parameters.

| Hyper-parameter | SR-DICE | DualDICE | GradientDICE | Deep SR | Deep TD |
|---|---|---|---|---|---|
| Optimizer | Adam | Adam | Adam | Adam | Adam |
| $\psi^\pi$, $Q^\pi$ Learning rate | $3\text{e}-4$ | - | - | $3\text{e}-4$ | $3\text{e}-4$ |
| $\mathbf{w}$ Learning rate | $3\text{e}-4$ | - | - | $3\text{e}-4$ | - |
| $f$ Learning rate | - | $5\text{e}-5$ | $1\text{e}-5$ | - | - |
| $w$ Learning rate | - | $5\text{e}-5$ | $1\text{e}-5$ | - | - |
| $u$ Learning rate | - | - | $1\text{e}-2$ | - | - |
| $\psi^\pi$, $Q^\pi$ Mini-batch size | 256 | - | - | 256 | 256 |
| $\mathbf{w}$, $f$, $w$, $u$, Mini-batch size | 2048 | 2048 | 2048 | 2048 | - |
| $\psi^\pi$, $Q^\pi$ Target update rate | 0.005 | - | - | 0.005 | 0.005 |

**Visualizations.** We graph the log MSE between the estimate of $R(\pi)$ and the true $R(\pi)$, where the log MSE is computed as $\log 0.5(X - R(\pi))^2$. We smooth the learning curves over a uniform window of 10. Agents were evaluated every 1k time steps and performance is measured over 250k time steps total. Markers are displayed every 25k time steps with offset for visual clarity.

**Randomized reward experiments.** The randomized reward experiment uses the MIS ratios collected from the *hard* setting from the continuous action environments. 1000 reward functions were generated by a randomly initialized neural network. The neural network architecture is two hidden layers of 256 with ReLU activation functions after each hidden layer and a sigmoid activation function after the final layer. Weights were sampled from the normal distribution and biases were set to 0. The ground truth value was estimated from 100 on-policy trajectories. If the ground truth value was less than 0.1 or greater than 0.9, (where the range of possible values is $[0, 1]$), the reward function was considered redundant and removed. To reduce variance across reward functions, we normalize both the estimated $R(\pi)$ and ground truth $R(\pi)$ by dividing by the average reward for all state-action pairs in the dataset.

## E.2. Atari

We interface with Atari by OpenAI gym (version 0.17.2) (Brockman et al., 2016), all agents use the NoFrameskip-v0 environments that include sticky actions with $p = 0.25$ (Machado et al., 2018b).

**Pre-processing.** We use standard pre-processing steps based on Machado et al. (2018b) and Castro et al. (2018). We base our description on (Fujimoto et al., 2019), which our code is closely based on. We define the following:

- Frame: output from the Arcade Learning Environment.
- State: conventional notion of a state in a MDP.
- Input: input to the network.

The standard pre-processing steps are as follows:

- Frame: gray-scaled and reduced to $84 \times 84$ pixels, tensor with shape $(1, 84, 84)$.
- State: the maximum pixel value over the 2 most recent frames, tensor with shape $(1, 84, 84)$.
- Input: concatenation over the previous 4 states, tensor with shape $(4, 84, 84)$.

The notion of time steps is applied to states, rather than frames, and functionally, the concept of frames can be abstracted away once pre-processing has been applied to the environment.

The agent receives a state every $4$th frame and selects one action, which is repeated for the following $4$ frames. If the environment terminates within these $4$ frames, the state received will be the last $2$ frames before termination. For the first $3$ time steps of an episode, the input, which considers the previous $4$ states, sets the non-existent states to all 0s. An episode terminates after the game itself terminates, corresponding to multiple lives lost (which itself is game-dependent), or after 27k time steps (108k frames or 30 minutes in real time). Rewards are clipped to be within a range of $[-1, 1]$.

Sticky actions are applied to the environment (Machado et al., 2018b), where the action $a_t$ taken at time step $t$, is set to the previously taken action $a_{t-1}$ with $p = 0.25$, regardless of the action selected by the agent. Note this replacement is abstracted away from the agent and dataset. In other words, if the agent selects action $a$ at state $s$, the transition stored will contain $(s, a)$, regardless if $a$ is replaced by the previously taken action.

**Experiments.** For the main experiments we use a behavior and target policy derived from a Double DQN agent (Van Hasselt et al., 2016), a commonly used deep reinforcement learning algorithm. The behavior policy is an $\epsilon$-greedy policy with $\epsilon = 0.1$ and the target policy is the greedy policy (i.e. $\epsilon = 0$). In Section B.5 we perform two additional experiments with a different behavior policy. Otherwise, all hyper-parameters are fixed across experiments. For each, the dataset contains $1$ million transitions and uses a discount factor of $\gamma = 0.99$. Each experiment is evaluated over 3 seeds.

**Pre-training.** Both SR-DICE and Deep SR rely on pre-training the encoder-decoder and deep successor representation $\psi$. Similar to the continuous action tasks, these networks were trained for 30k and 100k time steps respectively.

**Architecture.** We use the same architecture as most value-based deep reinforcement learning algorithms for Atari, e.g. (Mnih et al., 2015; Van Hasselt et al., 2016; Schaul et al., 2016). This architecture is used for all networks, other than the encoder-decoder network, for fair comparison and was not tuned in any way.

The network has a 3-layer convolutional neural network (CNN) followed by a fully connected network with a single hidden layer. As mentioned in pre-processing, the input to the network is a tensor with shape $(4, 84, 84)$. The first layer of the CNN has a kernel depth of 32 of size $8 \times 8$ and a stride of 4. The second layer has a kernel depth of 32 of size $4 \times 4$ and a stride of 2. The third layer has a kernel depth of 64 of size $3 \times 3$ and a stride of 1. The output of the CNN is flattened to a vector of 3136 before being passed to the fully connected network. The fully connected network has a single hidden layer of 512. Each layer, other than the output layer, is followed by a ReLU activation function. The final layer of the network outputs $|\mathcal{A}|$ values where $|\mathcal{A}|$ is the number of actions.

The encoder-decoder used by SR-DICE and SR-Direct has a slightly different architecture. The encoder is identical to the aforementioned architecture, except the final layer outputs the feature vector $\phi(s)$ with 256 dimensions and is followed by a ReLU activation function. The next state decoder uses a single fully connected layer which transforms the vector of 256 to 3136 and then is passed through three transposed convolutional layers each mirroring the CNN. Hence, the first layer has a kernel depth of 64, kernel size of $3 \times 3$ and a stride of 1. The second layer has a kernel depth of 32, kernel size of $4 \times 4$ and a stride of 2. The final layer has a kernel depth of 32, kernel size of $8 \times 8$ and a stride of 4. This maps to a $(1, 84, 84)$ tensor. All layers other than the final layer are followed by ReLU activation functions. Although the input uses a history of the four previous states, as mentioned in the pre-processing section, we only reconstruct the succeeding state without history. We do this because there is overlap in the history of the current input and the input corresponding to the next time step. The reward decoder is a linear function without biases.

**Network hyper-parameters.** Our hyper-parameter choices are based on standard hyper-parameters based largely on (Castro et al., 2018). All networks are trained with the Adam optimizer (Kingma & Ba, 2014). We use a learning rate of $6.25\text{e}{-5}$. Although not traditionally though of has a hyper-parameter, in accordance to prior work, we modify $\epsilon$ used by Adam to be $1.5\text{e}{-4}$. For $\mathbf{w}$ we use a learning rate of $3\text{e}{-4}$ with the default setting of $\epsilon = 1\text{e}{-8}$. For $u$ we use $1\text{e}{-3}$. We use a mini-batch size of 32 for all networks. SR-DICE, Deep SR, and Deep TD update the target network every 8k time steps. All hyper-parameters are described in Table 3.

**Visualizations.** We use identical visualizations to the continuous action environments. Graphs display the log MSE between the estimate of $R(\pi)$ and the true $R(\pi)$ of the target policy, where the log MSE is computed as $\log 0.5(X - R(\pi))^2$. We smooth the learning curves over a uniform window of 10. Agents were evaluated every 1k time steps and performance is measured over 250k time steps total. Markers are displayed every 25k time steps with offset for visual clarity.

Table 3: Training hyper-parameters for the Atari domain.

| Hyper-parameter | SR-DICE | DualDICE | GradientDICE | Deep SR | Deep TD |
|---|---|---|---|---|---|
| Optimizer | Adam | Adam | Adam | Adam | Adam |
| $\psi^\pi$, $Q^\pi$ Learning rate | $6.25\text{e}-5$ | - | - | $6.25\text{e}-5$ | $6.25\text{e}-5$ |
| $\psi^\pi$, $Q^\pi$, $f$, $w$ Adam $\epsilon$ | $1.5\text{e}{-4}$ | $1.5\text{e}{-4}$ | $1.5\text{e}{-4}$ | $1.5\text{e}{-4}$ | $1.5\text{e}{-4}$ |
| $\mathbf{w}$, $u$ Adam $\epsilon$ | $1\text{e}{-8}$ | - | $1\text{e}{-8}$ | $1\text{e}{-8}$ | - |
| $\mathbf{w}$ Learning rate | $3\text{e}-4$ | - | - | $3\text{e}-4$ | - |
| $f$ Learning rate | - | $6.25\text{e}-5$ | $6.25\text{e}-5$ | - | - |
| $w$ Learning rate | - | $6.25\text{e}-5$ | $6.25\text{e}-5$ | - | - |
| $u$ Learning rate | - | - | $1\text{e}-3$ | - | - |
| $\psi^\pi$, $Q^\pi$ Mini-batch size | 32 | - | - | 32 | 32 |
| $\mathbf{w}$, $f$, $w$, $u$, Mini-batch size | 32 | 32 | 32 | 32 | - |
| $\psi^\pi$, $Q^\pi$ Target update rate | 8k | - | - | 8k | 8k |

## References

Baird, L. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pp. 30–37. Elsevier, 1995.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80, pp. 1587–1596. PMLR, 2018.

Fujimoto, S., Conti, E., Ghavamzadeh, M., and Pineau, J. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

Fujimoto, S., Meger, D., and Precup, D. An equivalence between loss functions and non-uniform sampling in experience replay. *Advances in Neural Information Processing Systems*, 33, 2020.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*, 2017.

Machado, M. C., Bellemare, M. G., and Bowling, M. Count-based exploration with the successor representation. *arXiv preprint arXiv:1807.11622*, 2018a.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018b.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nachum, O., Chow, Y., Dai, B., and Li, L. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. In *Advances in Neural Information Processing Systems*, pp. 2315–2325, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, pp. 993–1000. ACM, 2009.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. IEEE, 2012.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Zhang, R., Dai, B., Li, L., and Schuurmans, D. Gendice: Generalized offline estimation of stationary values. *arXiv preprint arXiv:2002.09072*, 2020a.

Zhang, S., Boehmer, W., and Whiteson, S. Deep residual reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1611–1619, 2020b.

Zhang, S., Liu, B., and Whiteson, S. Gradientdice: Rethinking generalized offline estimation of stationary values. *arXiv preprint arXiv:2001.11113*, 2020c.