Supplementary Material

A Experimentation Details

A.1 Source code

Upon request, we will provide an anonymized version of our code in the rebuttal.

A.2 Decision Oriented Model Learning

We replicated our experiments using the codebase provided by Shah et al. [2022], which can be found at github. To ensure consistency, we used the same hyperparameters as mentioned in the code or article for the baselines. Our metric learning pipeline was added on top of their code, and thus we focused on tuning hyperparameters related to metric learning. The metric is parameterized as $\Lambda_\phi(x) = L_\phi(x)L_\phi^\top(x) + \epsilon_\phi \mathbb{I}_{n\times n}$, where ϵ_ϕ is a learnable parameter that explicitly controls the amount of Euclidean metric in the predicted metric. This helps ensure the stability of metric learning. We initialize the parameters in such a way that the predicted metric is close to the Euclidean metric. For each outer loop of metric update, we perform K inner updates to train the predictor. Following the methodology of Shah et al. [2022], we conducted 50 runs with different seeds for each of the experiments, where each method was evaluated on 10 different datasets, with 5 different seeds used for each dataset.

Table 3: Hyper-parameters for Decision Oriented Learning Experiments

Hyper-Parameter	Values
Λ_ϕ learning rate	10^{-3}
Λ_ϕ hidden layer sizes	`[200]`
Warmup steps	500
Inner Iterations (K)	100
Implicit derivative batchsize	10
Implicit derivative solver	Conjugate gradient on the normal equations (5 iterations)

A.3 Model-Based Reinforcement Learning

We consider the work of Nikishin et al. [2022] as the baseline for replicating the experiments, and we build upon their source code. Our metric learning is just one additional step to their method. We adopt exact same hyperparameters as their for dynamics learning and Action-Value function learning. We focus on exploring and tuning the hyper-parameters specific to the metric learning component of the method.

Table 4: Hyper-parameters for the CartPole experiments

Hyper-Parameter	Values
Λ_ϕ learning rate	10^{-3}
Λ_ϕ hidden layer sizes	`[32, 32]`
Warmup steps	5000
Inner iterations (K)	1
Implicit derivative batchsize	256
Implicit derivative solver	Conjugate gradient on the normal equations (10 iterations)

For the state distractor experiments, we parameterize the metric as an unconditinoal diagonal matrix, denoted as $\Lambda_\phi = \text{diag}(\phi)$ where $\phi \in \mathbb{R}^n$ and n is the dimension of the state space. In addition, we also consider a hyper-parameter of *metric parameterization*, for which we either take normalize or unnormliazed metric. When we refer to normalizing the metric, we mean constraining the norm of the ϕ vector to be equal to the L2 norm of an euclidean metric which is used by MSE method. This constraint the family of learnable metrics. To achieve this, we set $\phi := \frac{\phi}{\|\phi\|_2}\sqrt{n}$, ensuring

$\|\phi\|_2 = \|\mathbb{I}_{n\times n}\|_2 = \sqrt{n}$. We also used L1 regularization on the metric output, to induce sparsity in the metric. We sweep over three values of the regularization coefficient - $[0.0, 0.001, 0.1]$. We ran a sweep over the 6 combinations of hyperparameters - [unnormalized, normalized] $\times$ $[0.0, 0.001, 0.1]$ and choose the best hyper-parameter combination for each of the experiment. All the number reported in the experiments are calculated over 10 random seeds.

Our metric learning approach uses two implicit gradient steps. Firstly, we take the implicit derivative through action-value network parameters, approximating the inverse hessian to the identity, similar to Nikishin et al. [2022].Secondly, for the step through dynamics network parameters, we calculate the exact implicit derivative.

B Training time analysis

We also measure the training time of our method compared to MLE-based training, as using implicit derivatives may introduce additional computational overhead. It is important to note that the evaluation time is not affected by the use of metric learning, as the metric is only employed during the training phase. Consequently, the evaluation runtime of the agent remains the same for both methods.

Table 5: Time (sec) for one update step of the agent

Method	Time (s)
MLE	0.0240
TaskMet	0.0243

We measure the time for single update step of agent. The code is written in JAX and evaluated on the computer with GPU - Quadro GV100 and CPU - Intel Xeon Gold 6230 @ 2.10GHz. As shown in table 5, learning a metric using implicit derivative to train the predictor takes negligible extra time compared to MLE method.