Codebase

Understanding Effectiveness of Learning Behavioral *Metrics* in Deep Reinforcement *Learning*

Ziyan "Ray" Luo, Tianwei Ni, Pierre-Luc Bacon, Doina Precup, Xujie Si





The artifact is available at https://github.com/Rayluo-mila/understanding-metric-learning

Motivation: State Abstraction in RL

Scaling RL to **high-dimensional, distraction-rich** domains remains challenging

(sensors' output, torque control parameters)

Proprioceptive states

Observations

Distracting DMC: Over **90%** pixels are **task-irrelevant**

=>

=>

[1.15,
2.53,
3.45,
-2.02,
...]

a **compact state**

What is a Good State Abstraction in RL?

- **Denoise**: Capturing task-relevant information
- **Benefits**: computation efficiency, sample efficiency, generalization/robustness, better value estimation, …
- But how to achieve this abstraction?

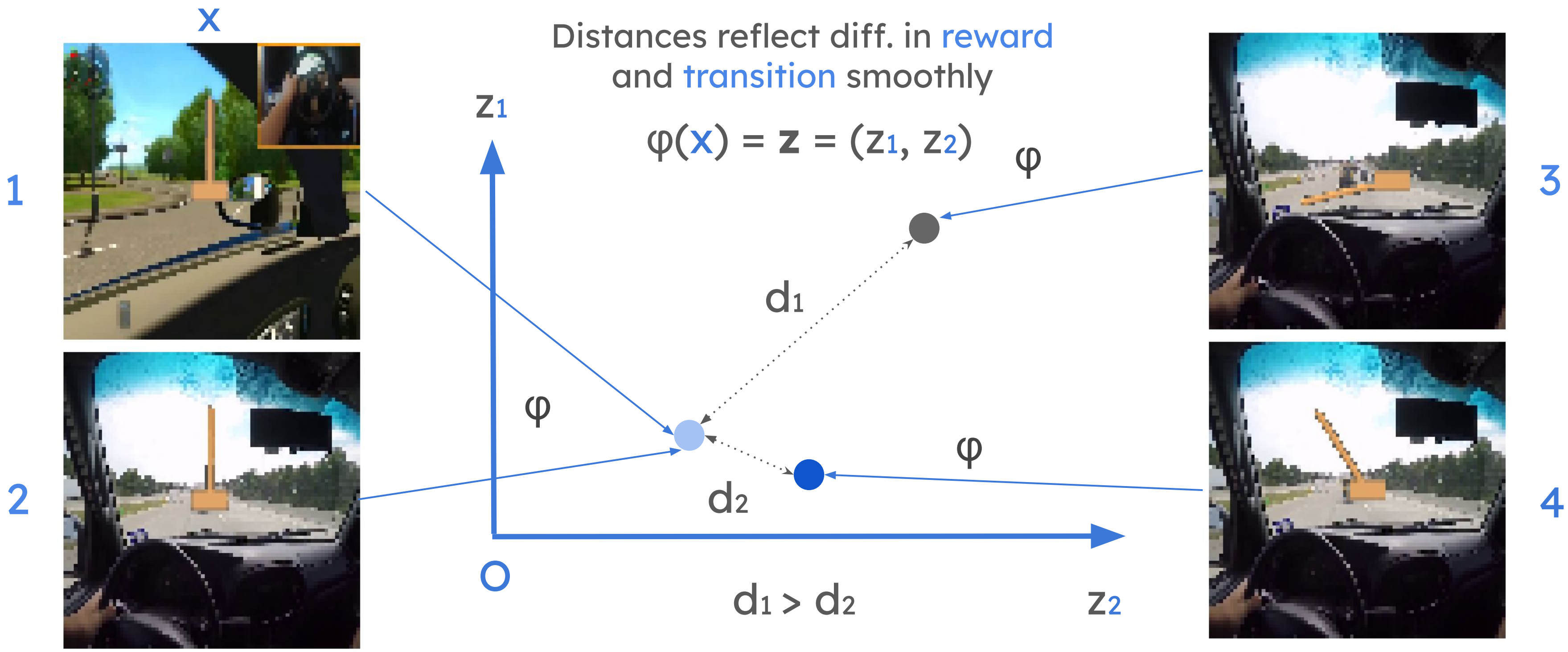
Behavioral metric: a **distance** that quantifies state similarity based on differences in **R & P**

- **Behaviorally similar** states have closer distances

Behavioral **Metric** (distance) & Denoising **Representation**

Behaviorally *similar* states should have *close* representations, vice versa

Described by behavioral metric



Isometric Embedding: connect and unify prior work

$$d_{\mathcal{X}}(x_1, x_2) = d_{\Psi}(\phi(x_1), \phi(x_2))$$

where $d_{\mathcal{X}}$ is the **target metric** and d_{Ψ} is the **representational metric**.

Target Metric $d_{\mathcal{X}}$: a **desired** behavioral distance between states

$$J_M(\phi) = \ell\left(d_{\Psi}(\phi(x_1), \phi(x_2)) - \hat{d}_{\mathcal{X}}(x_1, x_2)\right)$$

Use an auxiliary (regression) metric loss ℓ to approximate

Target Metric $d_{\mathcal{X}}$ - Examples, as design choices

PBSM (Castro, 2020; Zhang et al., 2020; Kemertas and Aumentado-Armstrong, 2021):



$$d^{\pi}(x_1, x_2) = \underbrace{c_R |\mathcal{R}^{\pi}(x_1) - \mathcal{R}^{\pi}(x_2)|}_{d_R} + \underbrace{c_T\, \mathcal{W}_1(d^{\pi})(\mathcal{P}^{\pi}(x_1), \mathcal{P}^{\pi}(x_2))}_{d_T}.$$

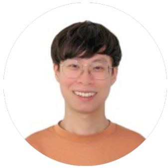
MICo (Castro et al., 2021):

$$u^{\pi}(x_1, x_2) = c_R |\mathcal{R}^{\pi}(x_1) - \mathcal{R}^{\pi}(x_2)| + c_T\, \mathbb{E}_{\substack{x_1' \sim \mathcal{P}^{\pi}(\cdot|x_1) \\ x_2' \sim \mathcal{P}^{\pi}(\cdot|x_2)}} [u^{\pi}(x_1', x_2')].$$

SimSR (Zang et al., 2022):

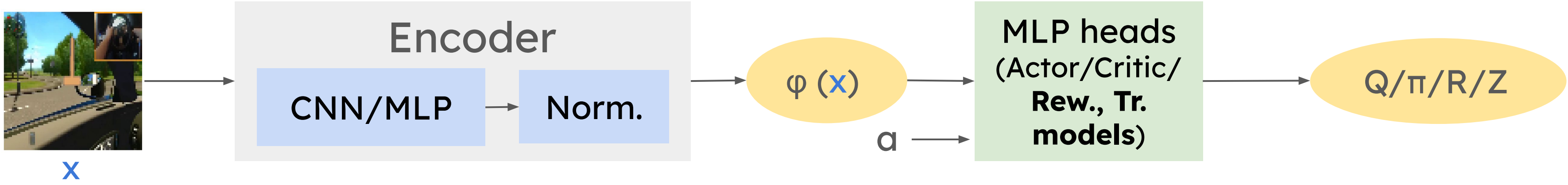
d_T uses learned $\hat{\mathcal{P}}^{\pi}$, $\quad u^{\pi}(x_1, x_2) = d_{\psi}(\phi(x_1), \phi(x_2)) = 1 - \cos(\phi(x_1), \phi(x_2)).$

RAP (Chen and Pan, 2022):



$$d_R(x_1, x_2)^2 = \mathbb{E}_{a_1 \sim \pi, a_2 \sim \pi}[(\mathcal{R}(x_1, a_1) - \mathcal{R}(x_2, a_2))^2] - \mathrm{Var}[r_{x_1}] - \mathrm{Var}[r_{x_2}].$$

Design choices in Metric Learning Objectives



- Self-prediction (**ZP**) loss

$$J_{\mathrm{ZP}}(\phi, \nu) = -\log P_\nu(\bar{\phi}(x') \mid \phi(x), a)$$

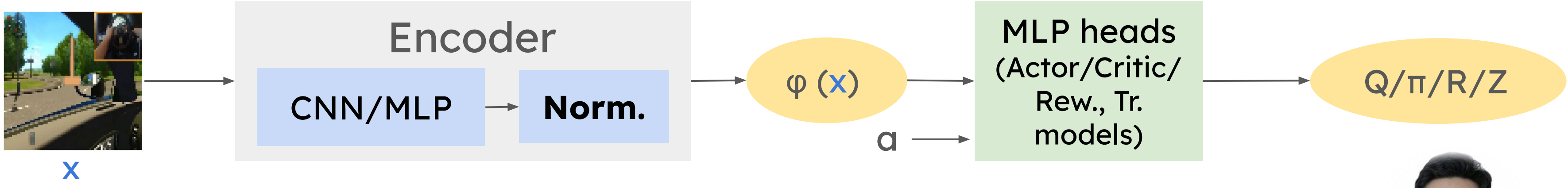
- Reward prediction (**RP**) loss

$$J_{\mathrm{RP}}(\phi, \kappa) = (R_\kappa(\phi(x), a) - r)^2$$

- Metric loss function ℓ (regression loss, e.g., MSE/Huber)

$$J_M(\phi) = \ell\Big(d_\Psi(\phi(x_1), \phi(x_2)) - \hat{d}_{\mathcal{X}}(x_1, x_2)\Big)$$

Design choices in Normalizing Representations



- L2 normalization $\mathrm{L2Norm}(\psi) = \frac{\psi}{\|\psi\|_2}.$ (from SimSR)
- MaxNorm: adjust the diameter of the vector so that d_Ψ is bounded theoretically

$$d_\Psi(\phi(x_1), \phi(x_2)) = d_\mathcal{X}(x_1, x_2) \leq \frac{c_R}{1 - c_T}(\max_{x,a} \mathcal{R}(x, a) - \min_{x,a} \mathcal{R}(x, a)) := C.$$

$$\mathrm{MaxNorm}(\psi) := \begin{cases} \psi, & \text{if } \|\psi\|_p < \frac{C}{2}, \\ \frac{C}{2}\frac{\psi}{\|\psi\|_p}, & \text{otherwise.} \end{cases}$$

(from DBC-normed)

- LayerNorm (as default design choice in CNNs in prior work)

$$\mathrm{LayerNorm}(\psi) = \alpha \odot \frac{\psi - \mu(\psi)}{\sqrt{\sigma^2(\psi) + \epsilon}} + \beta$$

Two baselines, five metric learning methods

Table 1: **Summary of key implementation choices for the benchmarked methods.**

Method	$\hat{d}_R$	$\hat{d}_T$	d_Ψ	Metric Loss	Target Trick	Other Losses	Transition Model	Normali -zation
SAC (Haarnoja et al., 2018)	—	—	—	—	—	—	—	—
DeepMDP (Gelada et al., 2019)	—	—	—	—	—	RP + ZP	Probabilistic	—
DBC (Zhang et al., 2020)	Huber	W_2 closed-form	Huber	MSE	—	RP + ZP	Probabilistic	—
DBC-normed (Kemertas & Aumentado-Armstrong, 2021)	Huber	W_2 closed-form	Huber	MSE	—	RP + ZP	Deterministic	MaxNorm
MICo (Castro et al., 2021)	Abs.	Sample-based	Angular	Huber	✓	—	—	—
RAP (Chen & Pan, 2022)	RAP	W_2 closed-form	Angular	Huber	—	RP + ZP	Probabilistic	—
SimSR (Zang et al., 2022)	Abs.	Sample-based	Cosine	Huber	—	ZP	Prob. ensemble	L2Norm

But…

how does the metrics help denoising?

Promising returns are reported.
But are they driven by better denoising through metric learning?

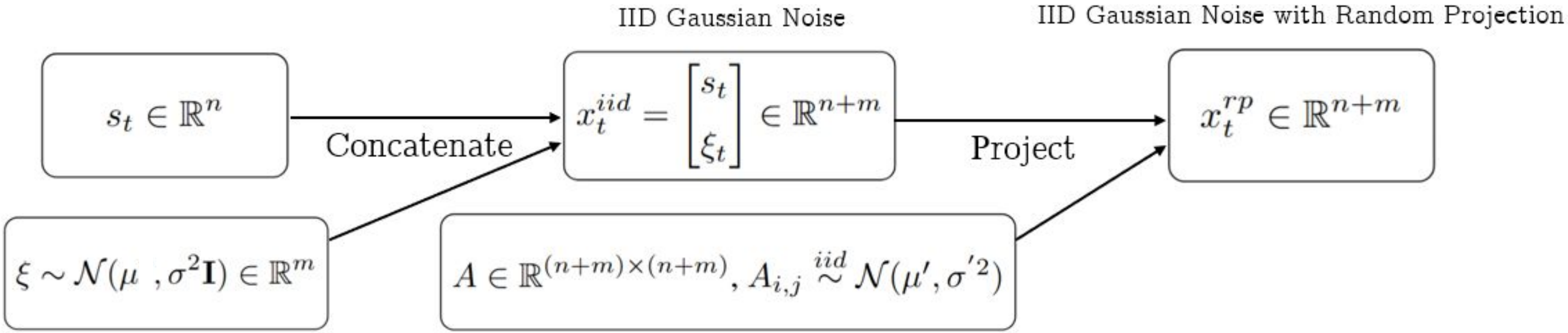
Study Design on Denoising

- ❑ Task (Noise) Diversity
- ❑ Generalization Evaluation
- ❑ Evaluation Measure
- ❑ Loss Attribution

Noise Settings

State-based environments:

- IID Gaussian Noise (dims and stds can be varied)
- IID Gaussian Noise with *Random Projection*

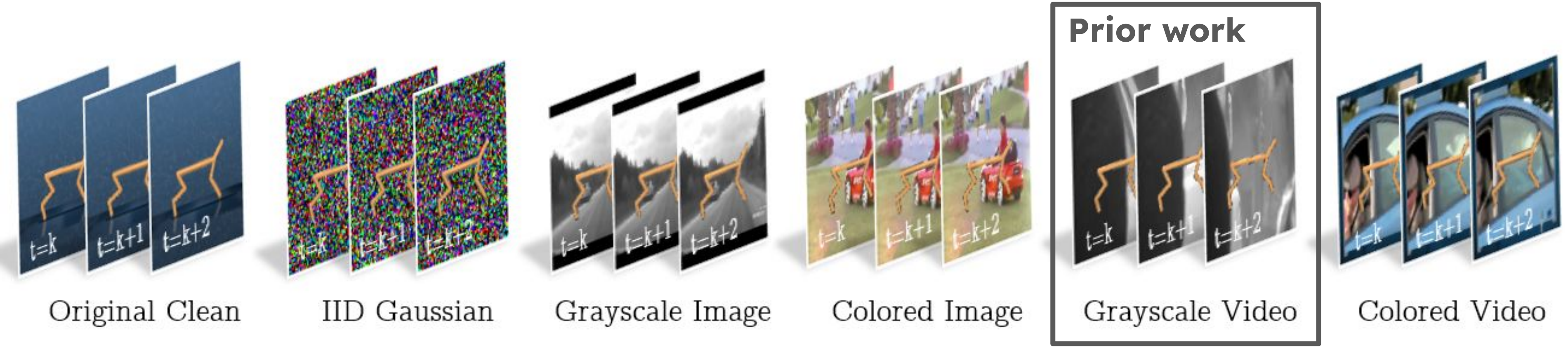


One run, one *A* sampled

Noise Settings

Pixel-based (backgrounds can be grayscale or colored):

- **IID Gaussian Noise** applied per-pixel
- **Natural Images**: clean background -> **one** randomly selected image [visual complexity only]
- **Natural videos**: clean background -> videos [temporally dependent]

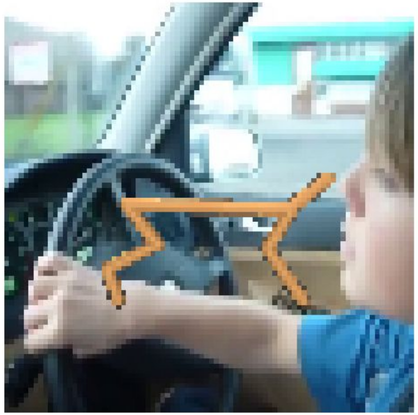


In-distribution (ID) vs. Out-of-distribution (OOD) Generalization

Always needed for excelling training reward

Training

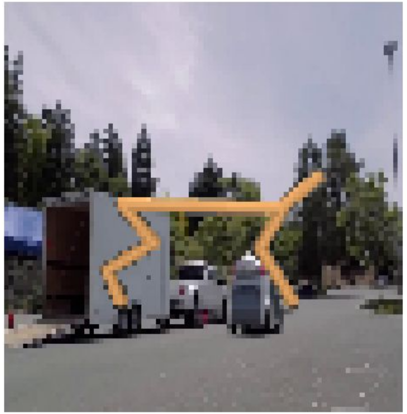
Evaluation



ID Generalization

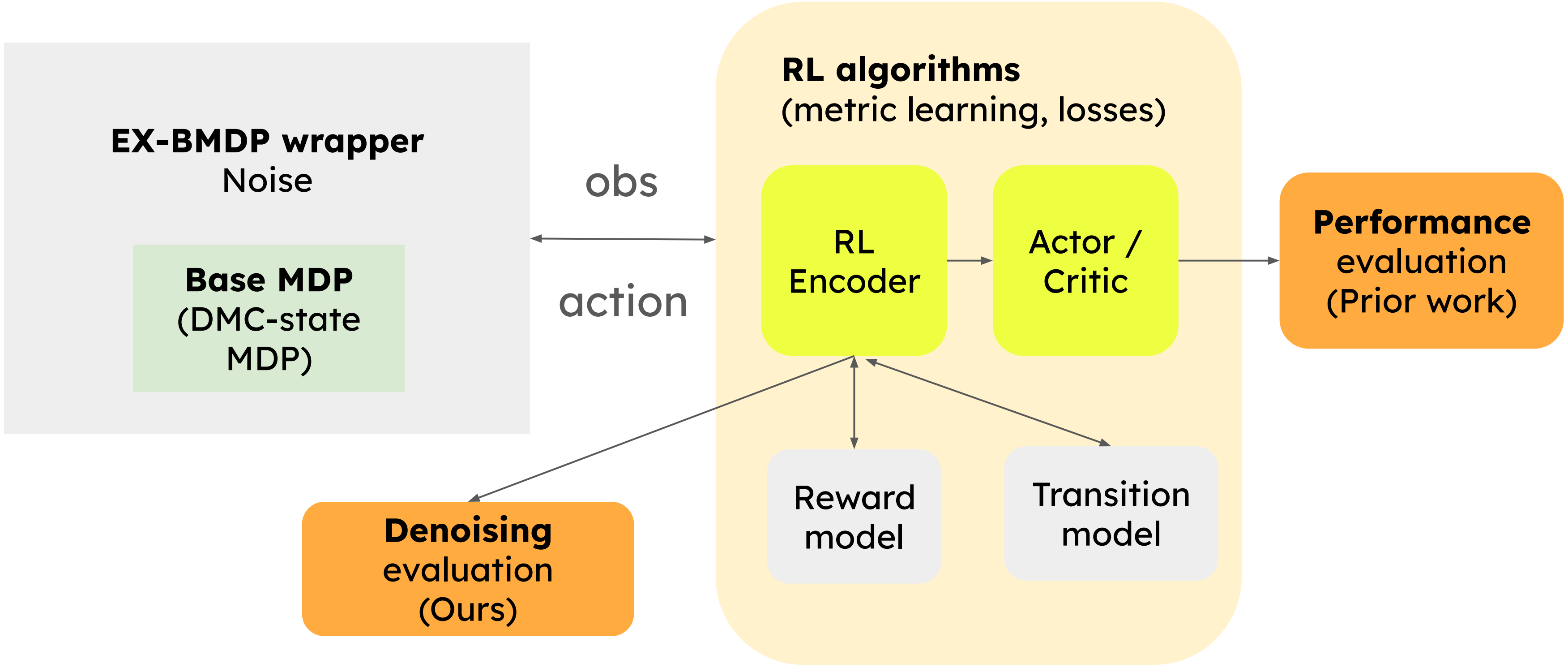
Training

Evaluation



OOD Generalization (prior work)

RL Performance + Encoder Denoising Evaluation



Quantifying Denoising

We introduce the *denoising factor* (DF), a measure that **quantifies** an encoder’s ability to **denoise**.

Positive examples x+

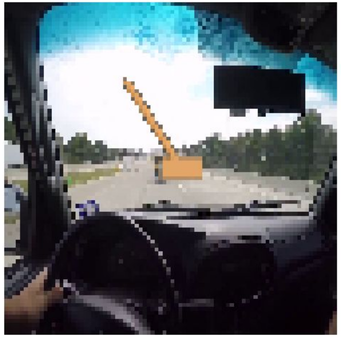
With same underlying states

Anchor x

Visited in evaluation

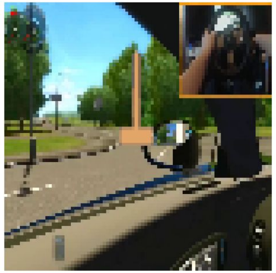
Negative examples x-

Any randomly sampled observations



Positive Score

Average distance from an anchor to its positive examples

$\mathrm{Pos} := \mathbb{E}\left[d_\Psi\left(\phi\left(\right.\right.\right.$  $\left.\left.\right), \phi\left(\right.$  $\left.\left.\left.\right)\right)\right]$

Positive examples x+

With same underlying states

Negative Score

Average distance from an anchor to its negative examples

$\mathrm{Neg} := \mathbb{E}\big[d_\Psi\big(\phi($  $), \phi($  $)\big)\big]$

Negative examples x+

Any randomly sampled
observations

Denoising Factor (DF)

$$\mathrm{DF}^{\pi}_{d_\Psi}(\phi) := \frac{\mathrm{Neg} - \mathrm{Pos}}{\mathrm{Neg} + \mathrm{Pos}} \in [-1, 1].$$

A **normalized** difference between Pos and Neg

Large DF, better denoising

Isolated Metric Estimation Setting

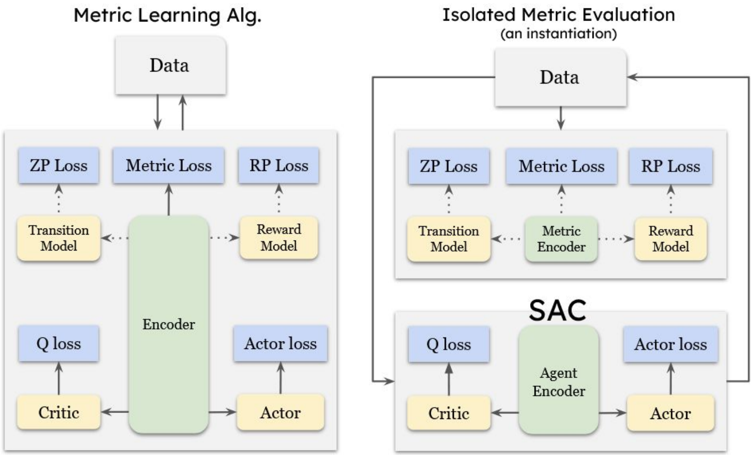
Running a SAC agent (or other baselines) with an Isolated metric encoder

- Optimized by **metric(-related) losses**
- Only used to **evaluate DF**

Ensure fair comparison by:

√ Remove other losses on representation from analysis

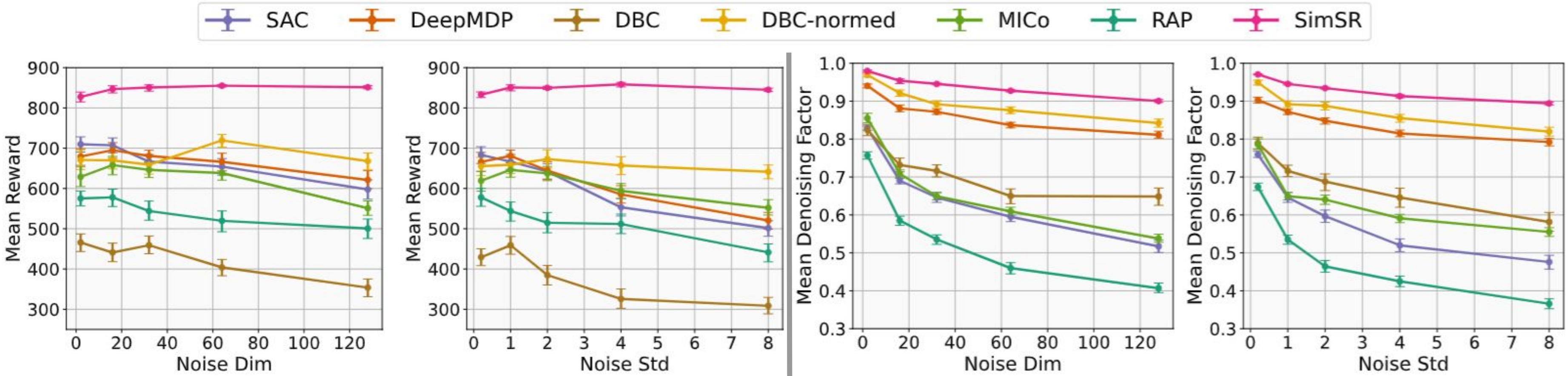
√ Ensure a fixed data collection (π)



Experiment

- **Benchmarking** result on various tasks and noise settings
  - Understanding task difficulty and agent’s performance on aggregate (~300 settings)
- **Case study**: What matters in metric (and representation) learning?
  - Identifying key design choices that lead to performance gain
- **Isolated Metric Evaluation Setting**: Does Metric Learning Help with Denoising?
  - Understanding the connection between metric learning and denoising
- OOD Generalization Evaluation on Pixel-based Tasks
  - The setting of interest in previous work

State-based benchmarking result: IID Gaussian Noise



Return

Denoising Factor (DF)

20 DMC tasks aggregated

- **SimSR** perform the best (but why?)
- Increasing noise dim/std -> moderate reward drop
- Well-performing methods are robust to noise variations

Case study: **with (R') / w/o (R)** LayerNorm on representation

6 representative easy-to-hard tasks are sampled for later analysis.

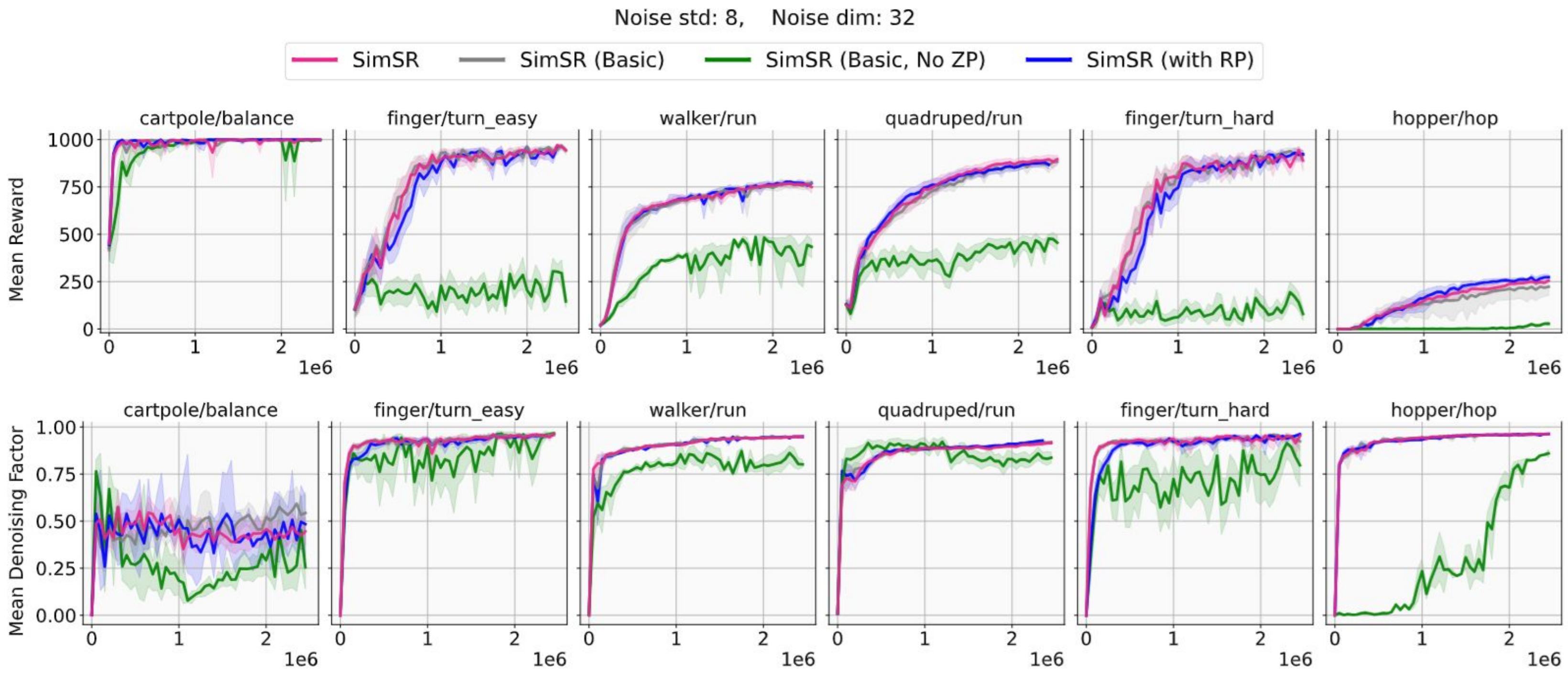
Task	Return	SAC	DeepMDP	DBC	DBC-normed	MICo	RAP	SimSR
		Methods						
cartpole/balance	R	967.5±12.3	928.7±32.3	814.1±86.6	973.7±12.4	966.6±9.2	950.3±71.2	999.5±0.5
	R'	979.5±20.1	994.6±3.6	943.6±24.1	975.5±19.9	936.1±29.8	981.7±19.1	980.2±19.3
finger/turn_easy	R	592.9±176.6	327.3±88.5	201.9±38.5	619.0±35.1	419.0±75.9	240.6±36.4	926.8±10.9
	R'	770.6±65.8	955.0±7.1	193.7±22.2	577.5±33.7	745.3±47.6	412.8±39.3	934.6±16.0
walker/run	R	635.3±19.8	347.8±84.0	23.9±2.6	628.9±25.7	455.9±41.3	649.4±11.1	760.6±19.4
	R'	534.5±53.6	776.0±5.9	342.9±54.5	759.8±19.4	611.0±22.5	661.6±88.4	761.6±20.0
quadruped/run	R	233.8±59.0	381.1±64.9	219.5±63.5	433.3±47.3	417.9±44.2	441.1±93.7	847.4±21.7
	R'	483.8±6.0	891.1±17.8	291.3±55.0	509.5±35.4	467.4±21.8	687.3±59.8	832.9±63.4
finger/turn_hard	R	177.6±66.1	168.3±50.4	97.9±11.8	414.7±49.5	207.2±53.8	110.8±17.0	885.4±24.5
	R'	495.7±53.1	925.8±14.7	95.9±12.4	473.4±39.9	335.1±42.6	201.1±26.3	917.1±13.9
hopper/hop	R	0.1±0.0	31.3±16.7	0.3±0.3	51.1±13.4	0.4±0.3	0.8±0.5	233.9±22.6
	R'	12.4±4.9	195.4±19.9	6.2±4.8	125.8±22.3	1.8±2.0	1.0±0.3	207.4±36.4

Red: **R' > R** **Blue**: **R' < R**

Most methods benefit from LayerNorm in the representation space

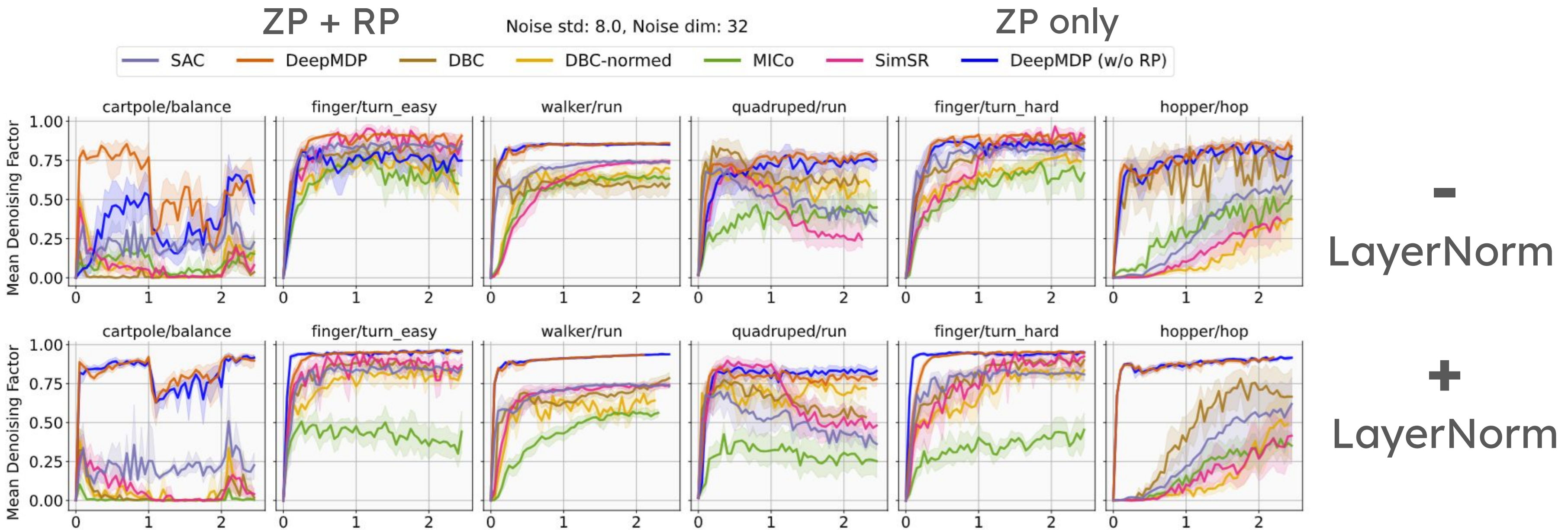
DeepMDP (RP+ZP) + LayerNorm ≈ SimSR

Case study: ZP's effectiveness in SimSR



- RP does not matter too much here

Isolated Metric Evaluation



Learned metric denoises,

but not better than by optimizing ZP (with LayerNorm)

Takeaways

Come to Poster #1 at 3pm!

Code

Blog

Paper

- Evaluate **simple, controlled settings first** to build foundational insight
- Support metric-learning claims via **direct measure**
- **Self-prediction (ZP) loss** & **Normalization** truly matters
- Examine when metric learning offers **unique benefit**

Knowledge Map