Joint Embedding Predictive Architectures Focus on Slow Features

Vlad Sobal[1] **Jyothir S V**[1] **Siddhartha Jalagam**[1] **Nicolas Carion**[2]
Kyunghyun Cho[1, 3, 4] **Yann LeCun** [1, 2]
[1]New York University [2]Meta AI [3]Prescient Design, Genentech [4]CIFAR Fellow
`{us441,jyothir, scj9994, carion.nicolas, kyunghyun.cho}@nyu.edu`
`yann@cs.nyu.edu`

Abstract

Many common methods for learning a world model for pixel-based environments use generative architectures trained with pixel-level reconstruction objectives. Recently proposed Joint Embedding Predictive Architectures (JEPA) [20] offer a reconstruction-free alternative. In this work, we analyze performance of JEPA trained with VICReg and SimCLR objectives in the fully offline setting without access to rewards, and compare the results to the performance of RSSM, the widely used generative architecture. We test the methods in a simple environment with a moving dot with various background distractors, and probe learned representations for the dot's location. We find that JEPA methods perform on par or better than RSSM when distractor noise changes every time step, but fail when the noise is fixed. Furthermore, we provide a theoretical explanation for the poor performance of JEPA-based methods with fixed noise, highlighting an important limitation.

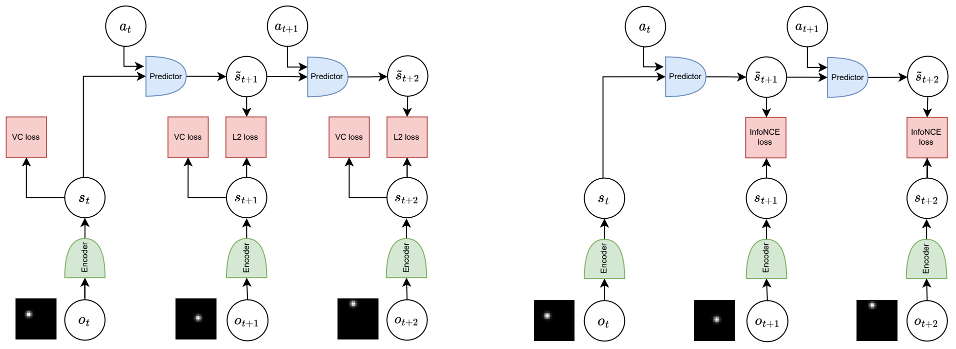
1 Introduction

Currently, the most common approach to learning world models is to use reconstruction objectives [11, 13, 12, 23]. However, reconstruction objectives suffer from object-vanishing, as the objectives by design do not distinguish important objects in the scene [24]. The framework of Joint Embedding Predictive Architectures presented in [20] may offer a possible alternative to reconstruction-based objectives, as has been shown in [30, 29, 24]. In this paper, we implement[1] JEPA for learning from image and action sequences with VICReg [3] and SimCLR [5] objectives (section 2 and code). We then test JEPA, reconstruction-based, and inverse dynamics modeling methods by training on image sequences of one moving dot and probing the learned representations for dot location in the presence of various noise distractors (section 3). We observe that JEPA methods can learn to ignore distractor noise that changes every time step, but fail when distractor noise is static.

2 Method

We focus on learning a world model from a fixed set of state-action sequences. We consider a Markov Decision Process (MDP) $M = (O, A, P, R)$. O is a set of possible observations (in our case these are images), A represents the set of possible actions, $P = Pr(o_t|o_{t-1}, a_{t-1})$ represents transition probabilities, and $R : O \times A \to \mathbb{R}$ is the reward function. In the offline setting we focus on, we do not have access to the MDP directly, we only have a pre-recorded set of sequences of observations $o_t \in O$ and actions $a_t \in A$. Our goal is to use these offline sequences to learn an encoder that converts observations o_t to D-dimensional representations s_t, $g_\phi(o_t) = s_t, g_\phi : O \to \mathbb{R}^D$, and the

[1]Code is available at `https://github.com/vladisai/JEPA_SSL_NeurIPS_2022`



(a) **VICReg-based architecture** (b) **SimCLR-based architecture**

Figure 1: **JEPA-based methods' training diagrams. (a)** JEPA with VICReg loss. VC here denotes variance and covariance losses from [3]. **(b)** JEPA with InfoNCE loss.

forward model $f_\theta(s_t, a_t) = \tilde{s}_{t+1}, f_\theta : \mathbb{R}^D \times A \rightarrow \mathbb{R}^D$. During pre-training we do not have access to the evaluation task, therefore we aim to learn representations that capture in D-dimensional vectors as much information about the environment and its dynamics as possible.

During both training and evaluation, given the initial observation o_1 and a sequence of actions of length T, $a_1 \ldots a_T$, we encode the first observation $\tilde{s}_1 = g_\phi(o_1)$, then auto-regressively apply the forward model $\tilde{s}_t = f_\theta(\tilde{s}_{t-1}, a_{t-1})$, obtaining representations for a sequence of length $T+1$: $\tilde{s}_1 \ldots \tilde{s}_{T+1}$. During testing, we probe the representations using a single linear layer that is trained, with frozen encoder and predictor, to recover some known properties of states $q(\tilde{s}_t) : \mathbb{R}^D \rightarrow \mathbb{R}^Q$, where Q is the dimension of the target value. This protocol is related to the one used in [1], but we probe not only the encoder output, but also the predictor output. Fore more details on probing, see appendix A.4. We compare multiple ways of training the encoder and predictor. In all the approaches, the gradients are propagated through time, and no image augmentations are used. Encoder and predictor architectures are fixed (see appendix A.7). We test the following methods:

VICReg (figure 1a) We take inspiration from [20], and adopt VICReg [3] objective for training a Joint Embedding Predictive Architecture (JEPA). We apply variance and covariance losses described in [3] to representations separately at each step, and apply the prediction loss to make the forward model output $\tilde{s}_t$ close to encoder output s_t. For a more detailed description, see appendix A.3.1.

SimCLR (figure 1b) In this case, we train JEPA, but instead of VICReg we utilize SimCLR objective [5]. We apply InfoNCE loss [25], with the forward model output and the encoder output for the same time step as positive pairs. For a more detailed description, see appendix A.3.2.

Recurrent state-space model (RSSM) This approach is directly taken from [12]. RSSM introduces a decoder $d_\xi(\tilde{s}_t) = \tilde{o}_t$, and utilizes a reconstruction objective $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \|o_t - \tilde{o}_t\|_2^2$ to train the encoder and predictor. See appendix A.3.4 for more details.

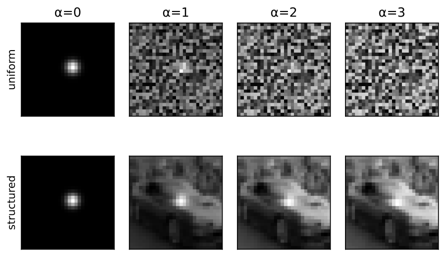
Inverse Dynamics Modeling (IDM) We add a linear layer that, given the encoder's outputs at two consecutive steps $g_\phi(o_t), g_\phi(o_{t+1})$, predicts a_t. The forward model is trained by predicting the encoder's output at the next time step. For more details, see appendix A.3.3.

Supervised All components are trained end-to-end by propagating the error from the probing function to both the encoder and decoder. This should give us the lower bound on probing error.

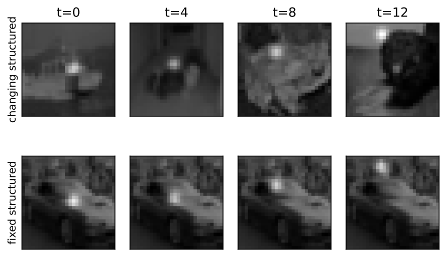
Random In this case, the probing is run with fixed random weights of the encoder and predictor.

2.1 Spurious correlation

VICReg and SimCLR JEPA methods may fall prey to a spurious correlation issue: the loss can be easily minimized by paying attention to noise that does not change with time, making the system ignore all other information. Intuitively, the objectives make the model focus on 'slow features' [37]. When the slowest features in the input belong to fixed background noise, the representation will only contain the noise. To demonstrate that, we show a trivial but plausible solution. In the presence of normally distributed distractor noise that does not change with time, the model can extract features by directly copying the values of the noise from the input : $g_\phi(o_t) = s \sim \mathcal{N}(0, \sigma^2 I),\ s \in \mathbb{R}^D$. Since



(a) **Different noise levels**



(b) **Changing and fixed structured noise** ($\alpha = 1$)

Figure 2: **Dataset examples. (a)** We introduce distractors to our moving dot dataset by adding either uniform or structured noise with different brightness coefficient α. **(b)** Temporally, the noise either changes every frame (top row), or remains fixed throughout the video (bottom row). In the fixed case, the noise is still re-sampled for each new sequence.

the noise is persistent through time, $g_\phi(o_t) = g_\phi(o_{t+1}) = s$. We assume that the forward model has converged to identity: $f_\theta(s, a) = s$. We denote a batch of encoded representations at step t with a matrix $S_t \in \mathbb{R}^{N \times D}$ where N is batch size; and batches of actions and observations with A_t and O_t respectively. Then the VICReg losses are:

$$\mathcal{L}_{\text{prediction}} = \frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} \|f_\theta(S_{t,i}, A_{t,i}) - g_\phi(O_{t+1,i})\|_2^2$$

$$= \frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} \|S_{t,i} - S_{t+1,i}\|_2^2 = \frac{1}{TN} \sum_{t=2}^{T+1} \sum_{i=1}^{N} \|S_{t,i} - S_{t,i}\|_2^2 = 0 \tag{1}$$

$$\text{Var}(s_t) = \frac{1}{N-1} \sum_{i=1}^{N} (s_i - \bar{s})^2 = \sigma^2 \qquad\qquad \text{As } s \sim \mathcal{N}(0, \sigma^2 I) \tag{2}$$

$$\mathcal{L}_{\text{variance}} = \frac{1}{(T+1)D} \sum_{t=1}^{T+1} \sum_{j=1}^{D} \max\left(0, \gamma - \sqrt{\text{Var}(S_{t,:,j}) + \epsilon}\right) = 0 \tag{3}$$

$$\mathcal{L}_{\text{covariance}} = \frac{1}{(T+1)(N-1)} \sum_{t=1}^{T+1} \sum_{i=1}^{D} \sum_{j=i+1}^{D} (S_t S_t^\top)_{i,j} = 0 \tag{4}$$

Equation 3 holds for large enough σ, 4 holds because the noise variables are independent across episodes. The total sum of the loss components is then 0 for the described trivial solution.

In SimCLR case, as shown by Wang and Isola [36] in their theorem 1, the InfoNCE loss is minimized in the infinite limit of negative samples if the positive pairs are perfectly aligned and the encoder output is perfectly uniform on the unit sphere. Both conditions are satisfied for the trivial solution described above, therefore SimCLR objective is also susceptible to fixed distractor noise problem.

3 Experiments

In order to verify whether the proposed JEPA-based methods indeed focus on fixed background noise, we introduce a simple moving dot dataset. The sequences of images contain a single dot on a square with sides of length 1, and the action denotes the delta in the dot's coordinates from the previous time step. We fix the length of the episode to 17 time steps (16 actions). After pre-training, we probe the representation by training a linear layer to recover the dot's position for all time steps. For pre-training, we use 1 million sequences; for training the prober we use 300,000 sequences; for evaluation, we use 10,000 sequences. We introduce two types of distractor noise to the background: structured and uniform. We generate structured noise by overlaying CIFAR-10 [19] images. Temporally, the noise can be changing, i.e., each time step the background is resampled; or fixed, i.e., the background does not change with time, but still changes between sequences. For examples, see figure 2. The coefficient α controls noise brightness relative to the dot. We tune hyperparameters of all methods separately for each noise level and type. For more details about the dataset, see appendix A.2.

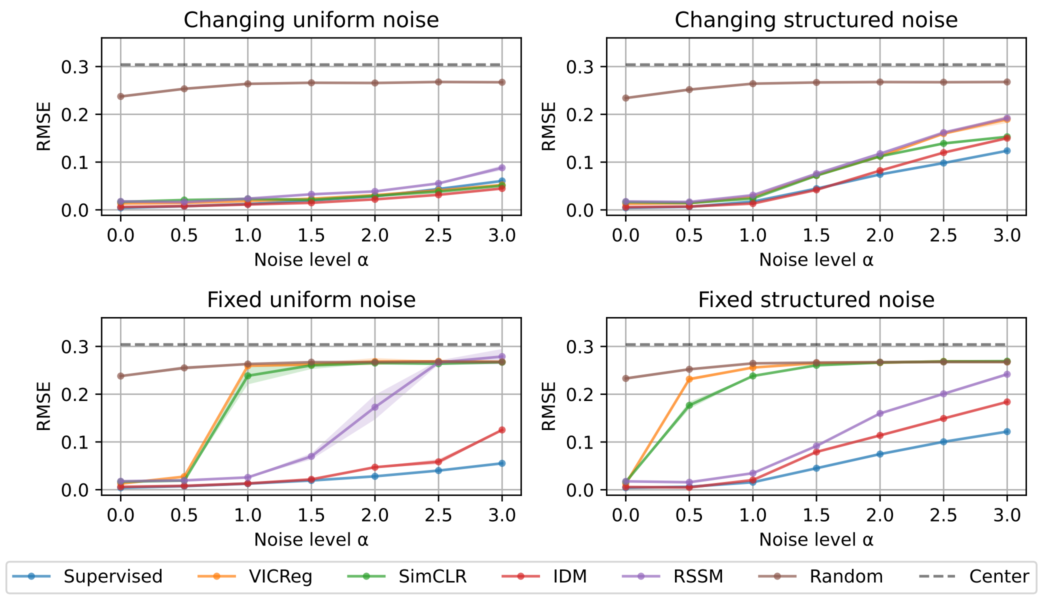


Figure 3: **Performance of the compared methods with different types and levels of noise.** We tune hyperparameters individually for each model, noise level, and type. We show results without tuning in figure 5 in the appendix. The dots represent the mean RMSE across 17 time steps. The shaded area represents the standard deviation calculated by running 3 random seeds for each experiment.

3.1 Results

We compare the approaches described in section 2 with different types of noise described above. We also add a baseline called 'Center', which corresponds to always predicting the dot's location to be in the center. 'Center' and 'Supervised' baselines should be upper and lower bounds on the error. The results are shown in figure 3. All methods perform well when there are no distractors. RSSM performs well in all settings with $\alpha \leq 1.5$, while JEPA-based methods fail in the presence of fixed noise, both structured and uniform. We hypothesize that, as described in section 2, these methods focus on background noise and ignore the dot. We observe a similar drop in performance when an extra static dot is introduced instead of distractor noise (see appendix A.6.2 for more details). All methods work well when the noise is changing every frame. Additionally, we find that JEPA-based methods do not require hyperparameter tuning to adapt to higher levels of changing noise, while RSSM performs much worse with untuned hyperparameters (see appendix A.5). Inverse dynamics modeling has great performance in all cases, but this method may be unsuitable for pre-training as it only learns representations that capture the agent, and fail if there is additional useful information to be captured, as we demonstrate with additional experiments with 3 dots in appendix A.6.

4 Conclusion

We demonstrate that JEPA-based methods offer a possible way forward for reconstruction-free forward model learning and are capable of ignoring unpredictable noise well even without additional hyperparameter tuning. However, these methods fail when slow features are present, even with a large pre-training dataset and hyperparameter tuning. We only demonstrate this with a toy dataset, but we hypothesize that this may happen in more complex problems. For example, when pre-training a forward-model for self-driving with JEPA on dash-cam videos, the model may focus on cloud patterns that are easily predictable, rather than trying to learn the traffic participants' behavior. This drawback of JEPA may be addressed by using image differences or optical flow as input to the model, although these input modalities will ignore potentially useful background and may still contain fixed noise. We believe that the way to learn representations that capture both fast and slow features is by adding hierarchy to the architecture (see HJEPA in [20]) or by changing the objective to impose an additional constraint that prevents the representations from being constant across time.

References

[1] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm. Unsupervised state representation learning in atari. *arXiv:1906.08226 [cs, stat]*, Nov 2020. URL `http://arxiv.org/abs/1906.08226`. arXiv: 1906.08226.

[2] K. Arulkumaran. Planet. `https://github.com/Kaixhin/PlaNet/`, 2021.

[3] A. Bardes, J. Ponce, and Y. LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv:2105.04906 [cs]*, May 2021. URL `http://arxiv.org/abs/2105.04906`. arXiv: 2105.04906.

[4] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv:2006.09882 [cs]*, Jan 2021. URL `http://arxiv.org/abs/2006.09882`. arXiv: 2006.09882.

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[6] T. Chen, C. Luo, and L. Li. Intriguing properties of contrastive losses. (arXiv:2011.02803), Oct 2021. doi: 10.48550/arXiv.2011.02803. URL `http://arxiv.org/abs/2011.02803`. arXiv:2011.02803 [cs, stat].

[7] X. Chen and K. He. Exploring simple siamese representation learning. (arXiv:2011.10566), Nov 2020. URL `http://arxiv.org/abs/2011.10566`. arXiv:2011.10566 [cs].

[8] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. (arXiv:1409.1259), Oct 2014. URL `http://arxiv.org/abs/1409.1259`. arXiv:1409.1259 [cs, stat].

[9] D. Gordon, K. Ehsani, D. Fox, and A. Farhadi. Watching the world go by: Representation learning from unlabeled videos. (arXiv:2003.07990), May 2020. URL `http://arxiv.org/abs/2003.07990`. arXiv:2003.07990 [cs].

[10] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv:2006.07733 [cs, stat]*, Sep 2020. URL `http://arxiv.org/abs/2006.07733`. arXiv: 2006.07733.

[11] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv*, Dec 2019. URL `http://arxiv.org/abs/1912.01603`.

[12] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. (arXiv:1811.04551), Jun 2019. URL `http://arxiv.org/abs/1811.04551`. arXiv:1811.04551 [cs, stat].

[13] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. *Mastering atari with discrete world models*. 2020.

[14] T. Han, W. Xie, and A. Zisserman. Video representation learning by dense predictive coding. (arXiv:1909.04656), Sep 2019. URL `http://arxiv.org/abs/1909.04656`. arXiv:1909.04656 [cs].

[15] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv:1911.05722 [cs]*, Mar 2020. URL `http://arxiv.org/abs/1911.05722`. arXiv: 1911.05722.

[16] R. D. Hjelm and P. Bachman. Representation learning with video deep infomax. (arXiv:2007.13278), Jul 2020. URL `http://arxiv.org/abs/2007.13278`. arXiv:2007.13278 [cs].

[17] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. (arXiv:1808.06670), Feb 2019. URL `http://arxiv.org/abs/1808.06670`. arXiv:1808.06670 [cs, stat].

[18] L. Jing, P. Vincent, Y. LeCun, and Y. Tian. Understanding dimensional collapse in contrastive self-supervised learning. *arXiv:2110.09348 [cs]*, Oct 2021. URL `http://arxiv.org/abs/2110.09348`. arXiv: 2110.09348.

[19] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[20] Y. LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. 2022.

[21] G. LORRE, J. RABARISOA, A. ORCESI, S. AINOUZ, and S. CANU. Temporal contrastive pretraining for video action recognition. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, page 651–659, Mar 2020. doi: 10.1109/WACV45572.2020.9093278.

[22] C. Lu, P. J. Ball, T. G. J. Rudner, J. Parker-Holder, M. A. Osborne, and Y. W. Teh. Challenges and opportunities in offline reinforcement learning from visual observations. (arXiv:2206.04779), Jun 2022. URL `http://arxiv.org/abs/2206.04779`. arXiv:2206.04779 [cs, stat].

[23] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. *arXiv:1507.08750 [cs]*, Dec 2015. URL `http://arxiv.org/abs/1507.08750`. arXiv: 1507.08750.

[24] M. Okada and T. Taniguchi. *Dreaming: Model-based Reinforcement Learning by Latent Imagination without Reconstruction*.

[25] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. (arXiv:1807.03748), Jan 2019. URL `http://arxiv.org/abs/1807.03748`. arXiv:1807.03748 [cs, stat].

[26] T. Pan, Y. Song, T. Yang, W. Jiang, and W. Liu. Videomoco: Contrastive video representation learning with temporally adversarial examples. (arXiv:2103.05905), Mar 2021. URL `http://arxiv.org/abs/2103.05905`. arXiv:2103.05905 [cs].

[27] R. Qian, T. Meng, B. Gong, M.-H. Yang, H. Wang, S. Belongie, and Y. Cui. Spatiotemporal contrastive video representation learning. (arXiv:2008.03800), Apr 2021. URL `http://arxiv.org/abs/2008.03800`. arXiv:2008.03800 [cs].

[28] J. Robinson, L. Sun, K. Yu, K. Batmanghelich, S. Jegelka, and S. Sra. Can contrastive learning avoid shortcut solutions? (arXiv:2106.11230), Dec 2021. URL `http://arxiv.org/abs/2106.11230`. arXiv:2106.11230 [cs].

[29] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman. Data-efficient reinforcement learning with self-predictive representations. *arXiv:2007.05929 [cs, stat]*, May 2021. URL `http://arxiv.org/abs/2007.05929`. arXiv: 2007.05929.

[30] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, D. Hjelm, P. Bachman, and A. Courville. Pretraining representations for data-efficient reinforcement learning. *arXiv:2106.04799 [cs]*, Jun 2021. URL `http://arxiv.org/abs/2106.04799`. arXiv: 2106.04799.

[31] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine. Time-contrastive networks: Self-supervised learning from video. (arXiv:1704.06888), Mar 2018. URL `http://arxiv.org/abs/1704.06888`. arXiv:1704.06888 [cs].

[32] T. Silva. Pytorch simclr: A simple framework for contrastive learning of visual representations. `https://github.com/sthalles/SimCLR`, 2021.

[33] A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv:2004.04136 [cs, stat]*, Sep 2020. URL `http://arxiv.org/abs/2004.04136`. arXiv: 2004.04136.

[34] A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *arXiv:2009.08319 [cs, stat]*, May 2021. URL `http://arxiv.org/abs/2009.08319`. arXiv: 2009.08319.

[35] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola. What makes for good views for contrastive learning? (arXiv:2005.10243), Dec 2020. URL `http://arxiv.org/abs/2005.10243`. arXiv:2005.10243 [cs].

[36] T. Wang and P. Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. (arXiv:2005.10242), Aug 2022. URL `http://arxiv.org/abs/2005.10242`. arXiv:2005.10242 [cs, stat].

[37] L. Wiskott and T. J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, Apr 2002. ISSN 0899-7667, 1530-888X. doi: 10.1162/089976602317318938.

[38] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. (arXiv:2103.03230), Jun 2021. URL `http://arxiv.org/abs/2103.03230`. arXiv:2103.03230 [cs, q-bio].

[39] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. (arXiv:2006.10742), Apr 2021. URL `http://arxiv.org/abs/2006.10742`. arXiv:2006.10742 [cs, stat].

[40] W. Zhang, A. GX-Chen, V. Sobal, Y. LeCun, and N. Carion. Light-weight probing of unsupervised representations for reinforcement learning. (arXiv:2208.12345), Aug 2022. URL `http://arxiv.org/abs/2208.12345`. arXiv:2208.12345 [cs].

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See appendix A.8.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix A.9
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See appendix A.10
   (b) Did you mention the license of the assets? [Yes] See appendix A.10
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Related work

Joint Embedding methods In recent years, multiple new joint-embedding methods for pre-training image classification systems were introduced [10, 38, 3, 5, 4, 15, 17, 7]. These methods heavily rely

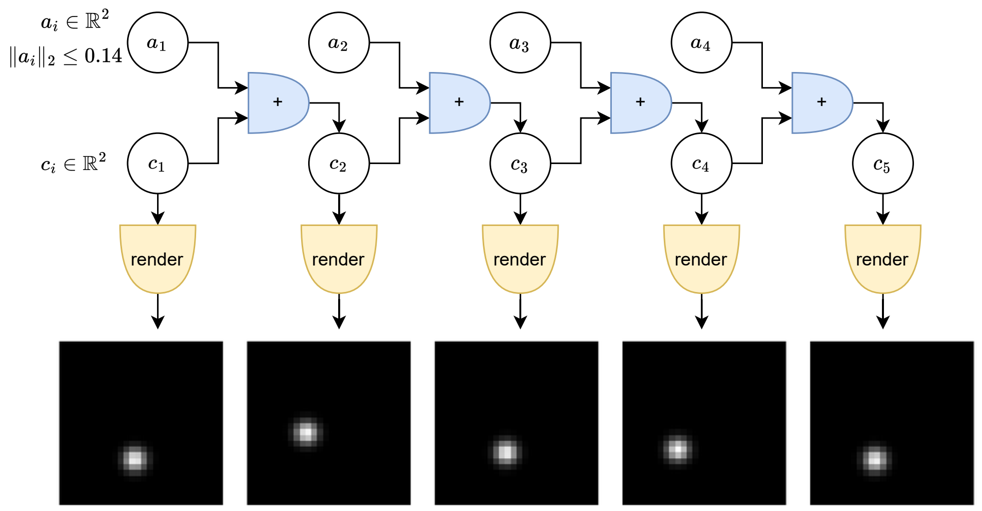


Figure 4: **Diagram of data generation process.** Initial position c_1 and action sequence $a_1 \dots a_T$ are first generated. Then, the actions are sequentially added to the initial location. The resulting coordinates $c_1 \dots c_{T+1}$ are then rendered.

on well-designed image augmentations to prevent collapse [35, 18]. Very closely related works of [6, 28] also investigate the tendency of contrastive losses to focus on easy features, although they do not study the application to videos.

Representation learning from video Many of the ideas from self-supervised pre-training for classification have also been applied to learning from video: [27] applies InfoNCE loss [25] to learn encodings of video segments, while [21] modify CPC architecture [25] and include samples from the same video clip as negative examples. [14] also propose a modification of CPC [25] for training from video and use predicted representations and encoder outputs as positive pair, and construct negative examples using both different videos and different fragments of the same video. [9] also use videos as an alternative to image augmentations. [31] use a triplet loss with single negative example instead of InfoNCE. [16] adopt Deep InfoMax objective [17] for videos, while [26] use MoCo [15].

Forward model learning Many methods using reconstruction objectives as main training signal have been proposed [23, 11, 13, 12]. [24] proposes a reconstruction-free version of Dreamer [11, 13] using contrastive learning. [39] proposes a method to learn representations using bisimulation objective, pushing states that lead to same rewards with same trajectories to have the same representations. In contrast, we focus on the setting where the reward is unknown during pretraining. Notably, [39] also explores the performance of reconstruction and CPC [25] objectives in the presence of distractor noise, but the noise is not fixed in time. [22] explore various aspects of model-based and model-free offline RL, and test RSSM with various perturbations, concluding that it performs quite well even with noisy data, which we confirm in our experiments as well.

Self-supervised learning for reinforcement learning Objectives used in self-supervised learning for images have been actively used in reinforcement learning from pixels to improve sample complexity of training [29, 33], or for pre-training the encoder [30, 34, 1, 40]. These methods focus on pre-training the encoder, and do not evaluate the trained forward model, even if it is present in the system [30].

A.2 Dataset

We consider the problem of capturing the location of an object in a video sequence. To this end, we introduce a simple environment with just one dot inside a square. The dot cannot leave the square, and is always visible on the screen. We denote dot's coordinates at time t as $c_t = (c_t^x, c_t^y)$. We assume the square size is 1 by 1, therefore $c_t^x, c_t^y \in [0, 1]$. At each step, the dot takes an action

$a_t = (a_t^x, a_t^y), a_t \in \mathbb{R}^2$. The norm of a_t is restricted to be less than or equal to a maximum step size D: $\|a_t\|_2 \leq D$. In our experiments $D = 0.14$.

The dot moves continuously around the square by moving by a vector specified by the action a_t with clipping to prevent the dot from going outside the square, i.e. $c_{t+1} = \max(0, \min(c_t + a_t, 1))$.

In order to generate one dataset example, we randomly generate a starting location c_1, and a sequence of actions $a_{1...T}$, where T is the sequence length. In our experiments $T = 16$. The actions in a sequence a_t are generated by first sampling vector directions using a Von Mises distribution with a randomly chosen direction $\omega \sim \text{Uniform}(0, 2\pi)$, $(u_t^x, u_t^y) = u_t \sim \text{VonMises}(\omega, 1.3)$. This prevents the dot from staying in one place in expectation, as would be the case with uniform random sampling. Then we multiply direction vectors with uniformly sampled step size $d_t \sim \text{Uniform}(0, D)$: $a_t = d_t u_t$. We then generate locations by adding the action vectors to the initial location. The length of the action sequence $a_{0...T-1}$ is then T, while the generated locations sequence is of length $T+1$ (in our case it is 17). A diagram of the process is shown in figure 4.

Once the actions and locations are generated, we obtain images o_t by rendering the dots at the generated locations c_t applying a Gaussian blur with $\sigma = 0.05$ to the image with the value set to 1 at the location c_t. In our experiments, the resolution of o_t is 28×28.

We introduce distractors to the dataset by overlaying noise onto the dot images. We consider two noise types: random and structured; and two temporal settings: fixed and changing. Random noise images Z are images of the same dimension as o_t where each pixel sampled from a uniform distribution, while structured noise images are loaded from CIFAR-10 dataset.

In all cases, we add noise Z with a coefficient β: $\hat{o}_t = o_t + \alpha Z_t$. Both noise image and dot image have values between 0 and 1, so the coefficient represents how many times stronger the brightest pixel in the noise is compared to the brightest pixel in the dot image.

A.3 Training methods details

We denote the batch of training video sequences as a tuple of observation and action sequences $(\mathcal{O}, \mathcal{A}), \mathcal{O} = (O_1 \dots O_{T+1}), O_t \in \mathbb{R}^{N \times H \times W}; \mathcal{A} = (A_1 \dots A_T), A_i \in \mathbb{R}^{N \times M}$. Here, T is the episode length, $H \times W$ is the resolution of observation image, N is batch size, M is the dimension of the action. For all algorithms we test, the observations are processed using an encoder to obtain representations for each time step $S_i = g_\phi(O_i), S_i \in \mathbb{R}^{N \times d}$, and the forward model is unfolded from the first observation with the given actions: $\tilde{S}_t = f_\theta(\tilde{S}_{t-1}, A_{t-1}); \tilde{S}_1 = S_1$. We use $\mathcal{S}, \tilde{\mathcal{S}}$ to denote encodings and predictions for all time steps: $\mathcal{S} = (S_1, \dots, S_{T+1}), \tilde{\mathcal{S}} = (\tilde{S}_1, \dots, \tilde{S}_{T+1})$.

A.3.1 VICReg

VICReg objective was originally used for image classification [3]. We follow ideas described in [20] and adopt the objective to learning from video. We consider representations at each time step separately for calculating variance and covariance losses, while the representation loss becomes the prediction error between forward model and encoder outputs. Total loss and its components are:

$$\mathcal{L}_{\text{VICReg}} = \alpha \mathcal{L}_{\text{prediction}} + \beta \mathcal{L}_{\text{variance}} + \mathcal{L}_{\text{covariance}} \tag{5}$$

$$\mathcal{L}_{\text{prediction}} = \frac{1}{NT} \sum_{t=1}^{T} \sum_{i=1}^{N} \|f_\theta(S_{t,i}, A_{t,i}) - g_\phi(O_{t+1,i})\|_2^2 = \frac{1}{NT} \sum_{t=2}^{T+1} \sum_{i=1}^{N} \|\tilde{S}_{t,i} - S_{t,i}\|_2^2 \tag{6}$$

$$\text{Var}(v) = \frac{1}{N-1} \sum_{i=1}^{N} (v_i - \bar{v})^2 \tag{7}$$

$$\mathcal{L}_{\text{variance}} = \frac{1}{T+1} \sum_{t=1}^{T+1} \frac{1}{D} \sum_{j=1}^{D} \max\left(0, \gamma - \sqrt{\text{Var}(S_{t,:,j}) + \epsilon}\right) \tag{8}$$

$$\mathcal{L}_{\text{covariance}} = \frac{1}{T+1} \sum_{t=1}^{T+1} \frac{1}{N-1} \sum_{i=1}^{D} \sum_{j=i+1}^{D} (S_t S_t^\top)_{i,j} \tag{9}$$

A.3.2 SimCLR

In case of SimCLR adaptation to JEPA, again, we use the same strategy as with VICReg, and treat each time step prediction separately. We apply SimCLR's InfoNCE loss [5] as follows:

$$\text{expsim}(u, v) = \exp\left(\frac{u^\top v}{\tau \|u\| \|v\|}\right) \tag{10}$$

$$\text{InfoNCE}(S_t, \tilde{S}_t) = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\text{expsim}(S_{t,i}, \tilde{S}_{t,i})}{\sum_{k=1}^{N} \text{expsim}(S_{t,i}, \tilde{S}_{t,i})) + \mathbb{1}_{k \neq i} \text{expsim}(S_{t,i}, S_{t,k})} \tag{11}$$

$$\mathcal{L}_{\text{SimCLR}} = \frac{1}{2T} \sum_{t=2}^{T+1} \text{InfoNCE}(S_t, \tilde{S}_t) + \text{InfoNCE}(\tilde{S}_t, S_t) \tag{12}$$

A.3.3 Inverse Dynamics Modeling (IDM)

We add a linear layer that, given the encoder's outputs at two consecutive steps $g_\phi(o_t), g_\phi(o_{t+1})$, predicts a_t. This should make the encoder pay attention to the parts of the observation that are affected by the action. The predictor is trained by predicting encoder output. We denote inverse dynamics model with $h_\xi(s_t, s_{t+1}) = \tilde{a}_t$. The loss components are then:

$$\mathcal{L} = \mathcal{L}_{\text{prediction}} + \mathcal{L}_{\text{IDM}} \tag{13}$$

$$\mathcal{L}_{\text{prediction}} = \frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} \|f_\theta(S_{t,i}, A_{t,i}) - g_\phi(O_{t+1,i})\|_2^2 \tag{14}$$

$$\mathcal{L}_{\text{IDM}} = \frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} \|h_\xi(S_{t,i}, S_{t+1,i}) - A_{t,i}\| \tag{15}$$

A.3.4 Recurrent State-Space Model (RSSM)

RSSM is a pixel reconstrution based method. It separates the state s_t into deterministic and stochastic parts: $s_t = (s_t^{\text{det}}, s_t^{\text{stoch}})$, the predictor then only predicts the deterministic part, while the prior and posterior models sample the stochastic part. Deterministic state model predicts the deterministic part of the state: $\tilde{s}_t^{\text{det}} = f_\theta(\tilde{s}_{t-1}^{\text{det}}, s_{t-1}^{\text{stoch}}, a_{t-1})$. Stochastic state model learns a distribution of stochastic states given the deterministic state: $\tilde{s}_t^{\text{stoch}} \sim p(\tilde{s}_t^{\text{stoch}} | \tilde{s}_t^{\text{det}})$. We also train a posterior model that has access to the last observation: $q(s_t^{\text{stoch}} | \tilde{s}_t^{\text{det}}, s_t)$. We train a decoder with the architecture that mimics the encoder to represent $p(o_t | s_t)$. Then, the total loss is:

$$\begin{aligned} \mathcal{L}_{\text{RSSM}} &= \mathcal{L}_{\text{reconstruction}} + \mathcal{L}_{\text{KL-divergence}} \\ &= \sum_{t=1}^{T} \left(\mathbb{E}_{q(\tilde{s}_t | o_{\leq t})}[\ln p(o_t | \tilde{s}_t)]) - \mathbb{E}[KL[q(s_t^{\text{stoch}} | \tilde{s}_t^{\text{det}}, s_t) || p(\tilde{s}_t^{\text{stoch}} | \tilde{s}_t^{\text{det}})]] \right) \end{aligned} \tag{16}$$

A.4 Probing details

In order to test whether representations contain the desired information, we train a probing function $q(s) : \mathbb{R}^D \to \mathbb{R}^Q$ which we represent with one linear layer. D is the representation size, Q is the target value dimension. In our case, the target value is the dot's location, so $Q = 2$. When training the prober, we follow a similar protocol to the pre-training. Given the initial observation o_1 and a sequence of actions of length T, $a_1 \dots a_T$, we encode the first observation $\tilde{s}_1 = g_\phi(o_1)$, then auto-regressively apply the forward model $\tilde{s}_t = f_\theta(\tilde{s}_{t-1}, a_{t-1})$, obtaining representations for a sequence of length $T + 1$: $\tilde{s}_1 \dots \tilde{s}_{T+1}$. We denote the location of the dot in the i-th batch element at time step t as $C_{t,i}$. Then, the loss function we apply to train q is:

$$\mathcal{L}_{\text{probing}} = \frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} \|q(\tilde{S}_{t,i}) - C_{t,i}\|_2^2 \tag{17}$$

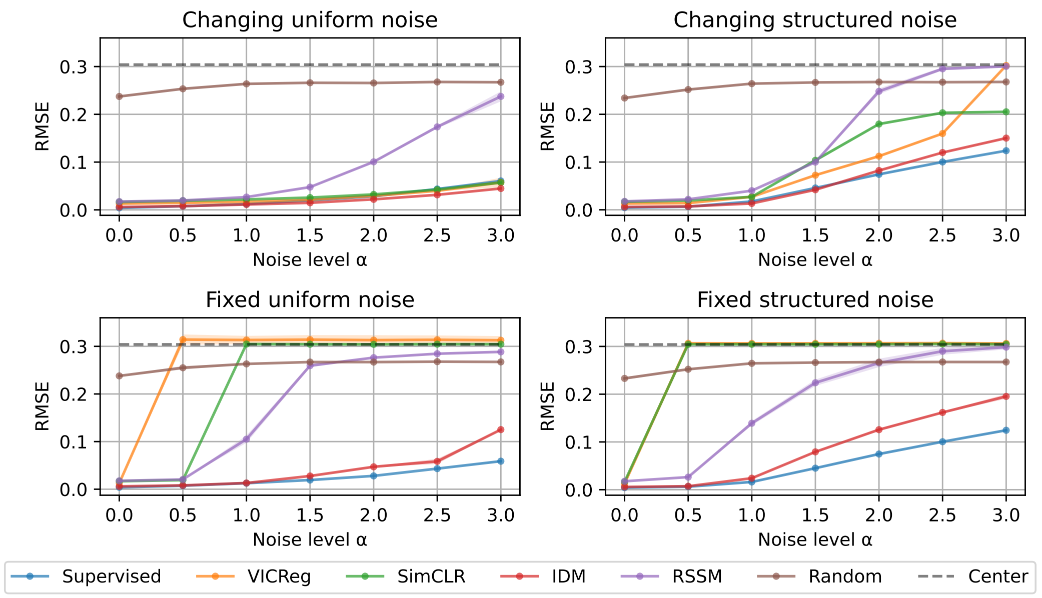


Figure 5: **Performance of compared methods with different types and levels of noise.** Hyperparameters for each model are chosen for no-noise setting and, in contrast to figure 3 are not re-tuned for each type or level of noise. The dots represent the mean RMSE across 17 time steps. The shaded area shows standard deviation calculated by running 3 random seeds for each experiment.

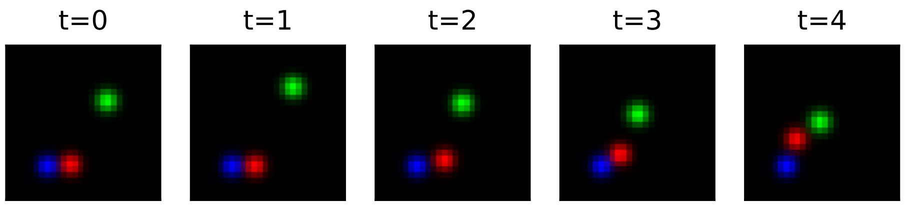


Figure 6: **Example sequence of 3 dots dataset.** Red, the first channel, corresponds to the action controlled dot. Green corresponds to the randomly moving dot with unknown actions. Blue corresponds to the stationary dot.

A.5 Additional results

In figure 5 we show results for the same experiments as in figure 3, with one difference that we do not tune hyperparameters for each model and noise setting. We pick the best hyperparameters for noise-free setting, and fix them for all levels of noise. We see that SimCLR and VICReg JEPA methods, compared to tuned performance, fail with lower levels of noise, while RSSM fails with changing uniform noise without tuning.

A.6 3 dots dataset

A.6.1 Environment

In this dataset, there are three dots on the three channels of the image: action-controlled dot, uncontrollable dot, and stationary dot. The first dot is action controlled similar to the single dot environment without any noise. The second dot is similar to the first dot, but the actions are unknown, making it impossible to predict its movements. The third dot is stationary across all the frames of the episode, but is at different positions in different episodes. To generate a sample of the three dot dataset, we concatenate these dots along different channels of image. These fixed channels ensure that our model can distinguish between the dots. We show an example sequence in figure 6. The goal

Table 1: **Results for 3-dots dataset.** All numbers denote RMSE across 17 steps. We run 3 seeds for each experiment to obtain standard deviations. Cells are colored according to the values, with higher values shown in red, and lower values shown in blue.

Method	Average	Action	Random	Stationary
VICReg	0.229±0.031	0.277±0.041	0.273±0.044	0.066±0.026
SimCLR	0.158±0.001	0.193±0.001	0.193±0.002	0.025±0.001
IDM	0.234±0.001	0.035±0.000	0.298±0.002	0.272±0.000
Supervised	0.104±0.000	0.010±0.001	0.180±0.001	0.005±0.000
RSSM	0.107±0.000	0.021±0.001	0.182±0.000	0.026±0.002
Random	0.260±0.001	0.235±0.002	0.278±0.002	0.265±0.004
Center	0.299±0.000	0.304±0.000	0.304±0.000	0.289±0.000

is to learn a representation that captures the locations of the action-controlled and stationary dots, while ignoring the random dot.

A.6.2 Results

We show performance of the compared methods on 3-dots dataset in table A.6.2. Hypeparameters were tuned for each model to obtain the best average RMSE. VICReg and SimCLR based methods focus only on the stationary dot, and fail to capture the other two dots. Again, we hypothesize that JEPA methods capture 'slow features' [37], and with the stationary dot containing the slowest features, the other dots are ignored. IDM manages to capture the action-controlled dot, but ignores the stationary dot, as it is irrelevant to inverse dynamics. Reconstruction-based RSSM captures both the stationary and action-controlled dots.

A.7 Model architectures

The encoder consists of 3 convolutional layers with ReLU and BatchNorm after each layer, and average pooling with kernel size of 2 by 2 and stride of 2 at the end. The first convolution layer has kernel size 5, stride 2, padding 2, and output dimension of 32. The second layer is the same as the first, except the output dimension is 64. The final layer has kernel size 3, stride 1, padding 1, and output dimension of 64. After average pooling, a linear layer is applied, with output dimension of 512.

The predictor is represented by a single-layer GRU [8] with hidden representation size of 512, and input size of 2. Hidden representation is initialized at the first step of predictions with encoder output $g_\phi(o_1)$. The inputs at each time step are actions.

RSSM model, apart from encoder and predictor, also has prior and posterior networks, which we represent with 2-layer neural network. The decoder is represented by a model symmetric to the encoder, with convolutions in reverse order and upsampling.

A.8 Limitations

The limitation of the current experiments is the use of the simple toy dataset. It is unclear that the same will hold for more complicated video datasets and bigger models. Another limitation is that we only check VICReg and SimCLR losses for JEPA methods, while there are many more objectives, e.g., [10, 38].

A.9 Computational resources

All experiments were run using AMD MI50 or Nvidia RTX 8000 GPUs. For each noise type and level, 100 random hyperparameters were run, and the best one was run for 3 seeds. Each individual experiment takes less than one hour of GPU time.

A.10 Code license

To implement RSSM, we used parts of the implementation of [2] distributed under MIT license. Implementing InfoNCE loss, we used the code from [32], which is also distributed under MIT license.

A.11 Acknowledgements

This material is based upon work supported by the National Science Foundation under NSF Award 1922658.