Counterfactual Data Augmentation using Locally Factored Dynamics

Silviu Pitis [1 2] **Elliot Creager** [1 2] **Animesh Garg** [1 2]

Abstract

Many dynamic processes, including common scenarios in robotic control and reinforcement learning (RL), involve a set of interacting subprocesses. Though the subprocesses are not independent, their interactions are sparse, and the dynamics at any given time step can often be decomposed into *locally independent* causal mechanisms. Such local causal structures can be leveraged to improve the sample efficiency of sequence prediction and off-policy RL. We formalize this by introducing local causal models (LCMs), which are induced from a global causal model by conditioning on a subset of the state space. We propose an approach to inferring these structures given an object-oriented state representation, as well as a novel algorithm for model-free Counterfactual Data Augmentation (CoDA). We find that CoDA significantly improves the performance of RL agents in locally factored tasks, including the batch-constrained and goal-conditioned settings. Code available at `https://github.com/spitis/mrl`.

1. Introduction

High-dimensional dynamical systems are often composed of simple subprocesses that affect one another through sparse interaction. If the subprocesses *never* interacted, an agent could realize significant gains in sample efficiency by globally factoring the dynamics (Guestrin et al., 2003). In most cases, however, the subprocesses do *eventually* interact and so in practice we model the entire process using an unfactored model. In this paper, we take advantage of the observation that *locally—in the time between interactions—the subprocesses are causally independent*. By locally factoring dynamic processes in this way, we can capture the benefits of factorization even in presence of global interactions.

[1]University of Toronto [2]Vector Institute. Correspondence to: Silviu Pitis <spitis@cs.toronto.edu>, Elliot Creager <creager@cs.toronto.edu>.

Workshop on Inductive Biases, Invariances and Generalization in RL (BIG) at ICML 2020.

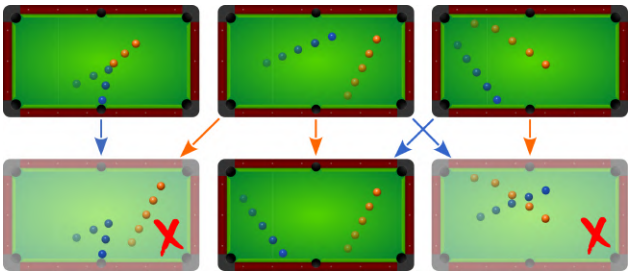


Figure 1. **Counterfactual Data Augmenation (CoDA)**. Given factual samples (top row), the local causal structure lets us mix and match factored subprocesses to form counterfactual samples. The first proposal is rejected because one of its factual sources (the blue ball) is not locally factored. The third proposal is rejected because it is not itself factored. The second proposal is accepted, and can be used as additional training data for an RL agent.

Consider a billiards game, where each ball is a separate physical subprocess. Predicting the opening break is difficult because all balls are mechanically coupled by their initial placement. So a dynamics model with dense coupling amongst balls may seem sensible, as each ball has a nonzero chance of colliding with the others. But at any given timestep, interactions between balls are usually sparse.

One way to leverage sparse interactions between otherwise disentangled entities is to use a structured state representation together with a graph neural network or other message passing transition model (Kipf et al., 2019). In many cases, however, the underlying processes are difficult to model, and model-free (Wang et al., 2019) or task-oriented model-based (Farahmand et al., 2017; Oh et al., 2017) approaches are less biased and exhibit superior performance. In this paper we argue that *knowledge of whether or not local interactions occur is useful in and of itself*, and can be used to generate causally-valid counterfactual data even in absence of a dynamics model. We observe that if two trajectories have the same local factorization in their transition dynamics we can, under mild conditions, produce new counterfactual data using our proposed **Counterfactual Data Augmentation (CoDA)** technique, wherein factorized subspaces of observed trajectory pairs are swapped (Figure 1). This lets us sample from a counterfactual distribution in an essentially model-free way by stitching together subsamples of factual trajectories. In the sequel, we formalize CoDA and show how it can improve results in locally factored tasks.

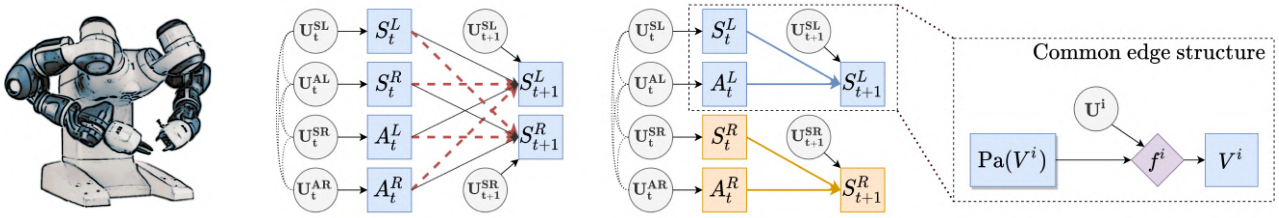


Figure 2. A two-armed robot (**left**) might be modeled as an MDP whose S, A decompose into left and right subspaces: $\mathcal{S} = \mathcal{S}^L \oplus \mathcal{S}^R, \mathcal{A} = \mathcal{A}^L \oplus \mathcal{A}^R$. Because the arms can touch, the global causal model (**center left**) between time steps is fully connected, even though left-to-right and right-to-left connections (dashed red edges) are rarely active. By restricting our attention to the subspace of states in which left and right dynamics are independent we get a local causal model (**center right**) with two components that can be considered separately.

2. Local Causality in MDPs

Problem Setup We consider a Markov Decision Process (MDP), described by tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ consisting of the state space, action space, transition function, reward function, and discount factor, respectively (Sutton & Barto, 2018). We denote individual states and actions by lowercase $s \in \mathcal{S}$, $a \in \mathcal{A}$, and variables by uppercase S, A (e.g., $s \in \text{range}(S) \subseteq \mathcal{S}$). In most non-trivial cases, the state $s \in \mathcal{S}$ can be described as an object hierarchy together with global context. In this paper we consider MDPs with a single, known top-level decomposition of the state space $\mathcal{S} = \mathcal{S}^1 \oplus \cdots \oplus \mathcal{S}^n$ for fixed n. $\mathcal{A}$ might be similarly decomposed: $\mathcal{A} = \mathcal{A}^1 \oplus \cdots \oplus \mathcal{A}^m$.

Given such decompositions, we may model time slice $(t, t+1)$ using a structural causal model $\mathcal{M}_t = \langle V_t, U_t, \mathcal{F} \rangle$ (Pearl, 2009) with directed acyclic graph (DAG) $\mathcal{G}$, where:
- $V_t = \{V^i_{t[+1]}\}_{i=0}^{2n+m} = \{S^1_t \dots S^n_t, A^1_t \dots A^m_t, S^1_{t+1} \dots S^n_{t+1}\}$ are the nodes (variables) of $\mathcal{G}$.
- $U_t = \{U^i_{t[+1]}\}_{i=0}^{2n+m}$ is a set of noise variables, one for each V^i. We assume that noise variables at time $t+1$ are independent from other noise variables. We use $u = (u^1, u^2, \dots, u^{2n+m})$ to denote an instance of U_t.
- $\mathcal{F} = \{f^i\}_{i=0}^{2n+m}$ is a set of functions ("structural equations") that map from $U^i_{t[+1]} \times \text{Pa}(V^i_{t[+1]})$ to $V^i_{t[+1]}$, where $\text{Pa}(V^i_{t[+1]}) \subset V_t \setminus V^i_{t[+1]}$ are the parents of $V^i_{t[+1]}$ in $\mathcal{G}$; hence each f^i is associated with the set of incoming edges to node $V^i_{t[+1]}$ in $\mathcal{G}$; see, e.g., Figure 2 (center).

While V_t, U_t, and $\mathcal{M}_t$ are indexed by t, the structural equations $f^i \in \mathcal{F}$ and causal graph $\mathcal{G}$ represent the global P and apply at all times t. To reduce clutter, we drop the subscript t on V, U, and $\mathcal{M}$ when no confusion can arise.

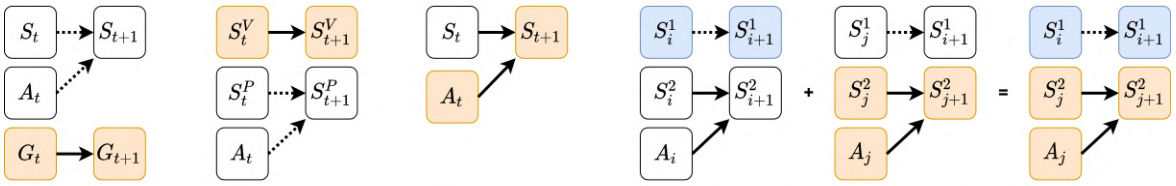
Assumption (Minimality)**.** *$V^j \in Pa(V^i)$ iff there exists $\{u^i, v^{-ij}\}$ with $u^i \in range(U^i), v^{-ij} \in range(V \setminus \{V^i, V^j\})$ and pair (v^j_1, v^j_2) with $v^j_1, v^j_2 \in range(V^j)$ such that $v^i_1 = f^i(\{u^i, v^{-ij}, v^j_1\}) \neq f^i(\{u^i, v^{-ij}, v^j_2\}) = v^i_2$.*

Critically, we require that the domain of each f^i and set of edges in $\mathcal{G}$ is *minimal* (Peters et al. (2017), Remark 6.6). Intuitively, minimality says that V^j is a parent of V^i if and only if setting the value of V^j can have an effect on the child V^i through structural equation f^i. Given minimality, we can think of edges in $\mathcal{G}$ as representing global causal dependence. The distribution of S^i_{t+1} is fully specified by its parents $\text{Pa}(S^i_{t+1})$ and its noise variable U_i and we have $S^i_{t+1} \perp\!\!\!\perp V^j \,|\, \text{Pa}(S^i_{t+1})$ for nodes $V^j \notin \text{Pa}(S^i_{t+1})$. We call an MDP with this structure a *factored MDP* (Kearns & Koller, 1999). When edges in $\mathcal{G}$ are sparse, factored MDPs admit more efficient solutions (Guestrin et al., 2003).

Limitations of Global Models Unfortunately, even if states and actions are disentangled, in most practical scenarios the DAG $\mathcal{G}$ is fully connected (or nearly so): since the f^i apply globally, so too does minimality, and edge (S^i_k, S^j_{k+1}) at time k is present so long as there is a single instance—at any time t, no matter how unlikely—in which S^i_t influences S^j_{t+1}. As a result, the factorized model $\mathcal{M}_t$ representing global dynamics rarely offers an advantage over a simpler model that treats states and actions as monolithic entities.

Local models Our key insight is that for each pair of nodes (V^i_t, S^j_{t+1}) with $V^i_t \in \text{Pa}(S^j_{t+1})$ in $\mathcal{G}$, there often exists a large subspace $\mathcal{S}^{(j \perp\!\!\!\perp i)} \subset \mathcal{S} \times \mathcal{A}$ for which $S^j_{t+1} \perp\!\!\!\perp V^i_t \,|\, \text{Pa}(S^j_{t+1}) \setminus V^i_t, (s_t, a_t) \in \mathcal{S}^{(j \perp\!\!\!\perp i)}$. E.g., in case of a two-armed robot (Figure 2), there is a large subspace in which the two arms are too far apart to influence each other physically. Thus, if we restrict our attention to $(s_t, a_t) \in \mathcal{S}^{(j \perp\!\!\!\perp i)}$, we can consider a *local* model $\mathcal{M}^{(j \perp\!\!\!\perp i)}_t$ whose local DAG $\mathcal{G}^{(j \perp\!\!\!\perp i)}$ is sparser than the global DAG $\mathcal{G}$, as the minimality assumption applied to $\mathcal{G}^{(j \perp\!\!\!\perp i)}$ implies there is no edge from V^i_t to S^j_{t+1}. More generally, for any $\mathcal{L} \subseteq S \times A$, we can induce the Local Causal Model (LCM) $\mathcal{M}^{\mathcal{L}}_t = \langle V^{\mathcal{L}}_t, U^{\mathcal{L}}_t, \mathcal{F}^{\mathcal{L}} \rangle$ with DAG $\mathcal{G}^{\mathcal{L}}$ from global $\mathcal{M}_t$ as:
- $V^{\mathcal{L}}_t = \{V^{\mathcal{L},i}_{t[+1]}\}_{i=0}^{2n+m}$, $P(V^{\mathcal{L},i}_{t[+1]}) = P(V^i_{t[+1]} | (s_t, a_t) \in \mathcal{L})$.
- $U^{\mathcal{L}}_t = \{U^{\mathcal{L},i}_{t[+1]}\}_{i=0}^{2n+m}$, $P(U^{\mathcal{L},i}_{t[+1]}) = P(U^i_{t[+1]} | (s_t, a_t) \in \mathcal{L})$.
- $\mathcal{F}^{\mathcal{L}} = \{f^{\mathcal{L},i}\}_{i=0}^{2n+m}$, where $f^{\mathcal{L},i} = f^i|_{\mathcal{L}}$. Due to minimality, the signature of $f^{\mathcal{L},i}$ may shrink.

Leveraging LCMs To answer the counterfactual question, "*what might the transition at time t have looked like if*



(a) Goal relabeling **(b)** Visual augmentation **(c)** Dyna (model-based) **(d)** Counterfactual Data Augmentation by swapping locally independent subspaces **(ours)**

Figure 3. Four instances of CoDA. **First:** Goal relabeling (Kaelbling, 1993) augments transitions with counterfactual goals. **Second:** Visual feature augmentation (Andrychowicz et al., 2020) changes visual features S_t^V that the designer knows do not impact the physical state S_{t+1}^P. **Third:** Dyna (Sutton, 1991) augments real states with new actions and resamples the next state using a dynamics model. **Fourth (ours):** Given two transitions that share local causal structures, we propose to swap connected components to form new transitions.

component S_t^i had value x instead of value y?", we would ordinarily apply Pearl's do-calculus to global causal model $\mathcal{M}$ to obtain submodel $\mathcal{M}_{\text{do}(S_t^i=x)} = \langle V, U, \mathcal{F}_x \rangle$, where $\mathcal{F}_x = \mathcal{F} \setminus f^i \cup \{S_t^i = x\}$ and incoming edges to S_t^i are removed from $\mathcal{G}_{\text{do}(S_t^i=x)}$ (Pearl, 2009). The component distributions at time $t+1$ can be computed by reevaluating each function f^j that depends on S_t^i. When S_t^i has many children (as is often the case in the global $\mathcal{G}$), this requires one to estimate many structural equations $\{f^i\}$. But if both the original value of S_t (with $S_t^i = y$) and its new value (with $S_t^i = x$) are in the set $\mathcal{L}$, the intervention is "within the bounds" of local model $\mathcal{M}^{\mathcal{L}}$ and we can instead work directly with local submodel $\mathcal{M}^{\mathcal{L}}_{\text{do}(S_t^i=x)}$ (defined accordingly). The validity of this follows from the definitions: since $f^{\mathcal{L},j} = f^j|_{\mathcal{L}}$ for all of S_t^i's children, the nodes V_t^k for $k \neq i$ at time t are held fixed, and the noise variables at time $t+1$ are unaffected, the distribution at time $t+1$ is the same under both models. When S_t^i has fewer children in $\mathcal{M}^{\mathcal{L}}$ than in $\mathcal{M}$, this reduces the number of structural equations that need to be considered.

3. Counterfactual Data Augmentation

Although LCMs are a general formalism that may have several applications, we focus our present effort on improving off-policy learning in RL by exploiting causal independence in local models for **Counterfactual Data Augmentation (CoDA)**. CoDA augments real data by making counterfactual modifications to a subset of the causal factors at time t, leaving the rest of the factors untouched. Following the logic outlined in the Subsection 2, this can understood as manufacturing "fake" data samples using the counterfactual model $\mathcal{M}^{[\mathcal{L}]}_{\text{do}(S_t^{i\ldots j}=x)}$, where we modify $S_t^{i\ldots j}$ and resample their children. While this is always possible using a model-based approach if we have good models of the structural equations, it is particularly nice when the causal mechanisms are *independent*, as we can use environment samples for model-free counterfactual reasoning.

Definition. *The causal mechanisms of $\mathcal{G}_i, \mathcal{G}_j \subset \mathcal{G}$ are **independent** when $\mathcal{G}_i$ and $\mathcal{G}_j$ are disconnected in $\mathcal{G}$.*

Global independence relations have been used for CoDA by past work on goal relabeling (Kaelbling, 1993) and visual feature augmentation (Laskin et al., 2020); see Figure 3.

We propose a novel form of model-free CoDA that uses knowledge of *local* independence relationships. In particular, we observe that whenever an environment transition is within the bounds of some local model $\mathcal{M}^{\mathcal{L}}$ whose graph $\mathcal{G}^{\mathcal{L}}$ has the locally independent causal mechanism $\mathcal{G}_i$ as a disconnected subgraph, that transition contains an unbiased sample from $\mathcal{G}_i$. Thus, given two transitions in $\mathcal{L}$, we may mix and match samples of $\mathcal{G}_i$ to generate counterfactual data, so long as the resulting transitions are also in $\mathcal{L}$.

Remark 3.1. *If we have n independent samples from subspace $\mathcal{L}$ whose graph $\mathcal{G}^{\mathcal{L}}$ has m connected components, we have n choices for each of the m components, for a total of n^m CoDA samples—an **exponential** increase in data!*

Implementing Model-Free CoDA We implement CoDA as a function of two factual transitions and a mask function $M(s_t, a_t) : |S| \times |A| \to \{0,1\}^{(|S|+|A|)\times|S|}$ that returns the adjacency matrix of the sparsest local causal graph $\mathcal{G}^{\mathcal{L}}$ such that $\mathcal{L}$ is a neighborhood of (s_t, a_t). We apply M to each transition to obtain local masks m_1 and m_2, compute their connected components, and swap independent components $\mathcal{G}_i$ and $\mathcal{G}_j$ between transitions to get a CoDA proposal. We then apply M to counterfactual $(\tilde{s}_t, \tilde{a}_t)$ to validate the proposal—if counterfactual $\tilde{m}$ shares graph partitions with m_1 and m_2, we accept the proposal as a CoDA sample. See Appendix A for pseudocode and more technical exposition.

Inferring local factorization While the ground truth mask M may be available in rare cases as part of a simulator, the general case either requires a domain expert to specify an approximate causal model or requires the agent to learn the local factorization from data. To learn the masking function, we consider a single-head set transformer architecture (Lee et al., 2018), which is *trained* to model forward dynamics using an L2 prediction loss, but used by the agent to *infer* local factorization (not to directly sample future states as is typical in model-based RL). We found this

$\lvert\mathcal{D}\rvert$ ($1000s$)	Real data 1_R	MBPO $1_R{:}1_M$	Ratio of Real:CoDA [:MBPO] data (ours) $1_R{:}1_C$	$1_R{:}3_C$	$1_R{:}5_C$	$1_R{:}3_C{:}1_M$
25	13.2 ± 0.7	18.5 ± 1.5	43.8 ± 2.8	40.9 ± 2.5	38.4 ± 4.9	$\mathbf{46.8} \pm 3.1$
50	22.8 ± 3.0	36.6 ± 4.3	66.6 ± 3.8	64.4 ± 3.1	62.5 ± 3.5	$\mathbf{70.4} \pm 3.8$
75	43.2 ± 4.9	46.0 ± 4.7	73.4 ± 2.8	$\mathbf{76.7} \pm 2.6$	75.0 ± 3.4	74.6 ± 3.2
100	63.0 ± 3.1	66.4 ± 4.9	77.8 ± 2.0	$\mathbf{82.7} \pm 1.5$	76.6 ± 3.0	73.7 ± 2.9
150	77.4 ± 1.2	72.6 ± 5.6	82.2 ± 1.8	$\mathbf{85.8} \pm 1.4$	84.2 ± 1.0	79.7 ± 3.6
250	78.2 ± 2.7	77.9 ± 2.4	85.0 ± 2.9	$\mathbf{87.8} \pm 1.8$	87.0 ± 1.0	78.3 ± 4.9

Table 1. **Batch RL** (10 seeds): Mean success ($\pm$ standard error, estimated using 1000 bootstrap resamples) on `Pong` environment. CoDA with learned masking function more than doubles the effective data size, resulting in a 3x performance boost at smaller data sizes.

produced reasonable results in the tested domains (below). Once trained, the local network mask $M(s_t, a_t)$ is found by computing the matrix product of the attention masks at each layer: $M(s_t, a_t) = \Pi_{\ell=1}^{L} M_\ell(s_t, a_t)$, which generalizes the global network masks of MADE (Germain et al., 2015) and GraN-DAG (Lachapelle et al., 2019). See Appendix D for details and an empirical evaluation of our approach.

4. Experiments

We evaluate CoDA in the batch and goal-conditioned settings, in each case finding that CoDA significantly improves agent performance as compared to non-CoDA baselines. See Appendix C for specific details and additional results.

Batch RL A natural setting for CoDA is batch RL, where an agent has access to an existing transition-level dataset, but cannot collect more data via exploration (Fujimoto et al., 2018a). This makes any additional, high quality data invaluable. Another reason why CoDA is attractive in this setting is that there is no *a priori* reason to prefer the given batch data distribution to a counterfactual one. We use a continuous control `Pong` environment where the agent must hit the ball past the opponent, receiving reward of +1 when the ball is behind the opponent's paddle, -1 when the ball is behind the agent's paddle, and 0 otherwise. Since our transformer model performed poorly when used as a dynamics model, our Dyna baseline adopts a state-of-the-art architecture (Janner et al., 2019) that employs a 7-model ensemble (MBPO). We collect datasets of up to 250,000 samples from an pretrained policy with added noise. For each dataset, we train both mask and reward functions (and in case of MBPO, the dynamics model) on the provided data and use them to generate different amounts of counterfactual data. We train the same TD3 agent on the expanded datasets in batch mode for 500,000 optimization steps. The results in Table 1 show that with only 3 state factors (two paddles and ball), CoDA approximately doubles the effective data size.

Goal-conditioned RL As HER (Andrychowicz et al., 2017) is an instance of prioritized CoDA that greatly improves sample efficiency in sparse-reward tasks, can our unprioritized CoDA algorithm further improve HER agents? We use HER to relabel goals on real data only, relying on random CoDA-style goal relabeling for CoDA data. We obtain state-of-the-art results in `FetchPush-v1` (Plappert et al., 2018) and show that CoDA also accelerates learning in a novel and significantly more challenging `Slide2` environment, where the agent must slide two pucks onto their targets (Figure 4). For this experiment, we specified a heuristic mask using domain knowledge ("objects are disentangled if more than 10cm apart") that worked in both `FetchPush` and `Slide2` despite different dynamics.

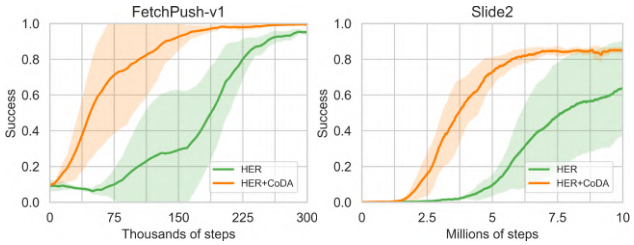


Figure 4. **Goal-conditioned RL** (5 seeds): In `FetchPush` and the challenging `Slide2` environment, a HER agent whose dataset has been enlarged with CoDA achieves twice the sample efficiency.

5. Future work

There are several interesting avenues for future work. First, we have applied CoDA in an unprioritized fashion, but past work (Schaul et al., 2015b) suggests there is significant benefit to prioritization. Second, we have applied CoDA model-free, but our LCM formalism allows for mixing model-free and model-based reasoning, which might further improve results. Third, we would like to deploy CoDA in partially observable, entangled settings (e.g. RL from pixels) with multiple dynamic, multi-level decompositions (Higgins et al., 2018; Esmaeili et al., 2019). Unsupervised learning of factorized latent representations is an active area of research (Dundar et al., 2020; Locatello et al., 2018), and it would be interesting to combine these methods with CoDA. Finally, it would be interesting to explore applications of our LCM framework to other areas such as interpretability (Lyu et al., 2019), exploration (Wang & Hayden, 2019), and off-policy evaluation (Thomas & Brunskill, 2016).

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in neural information processing systems*, pp. 4349–4357, 2016.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Buesing, L., Weber, T., Zwols, Y., Racaniere, S., Guez, A., Lespiau, J.-B., and Heess, N. Woulda, coulda, shoulda: Counterfactually-guided policy search. *International Conference on Learning Representations*, 2019.

Diuk, C., Cohen, A., and Littman, M. L. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247, 2008.

Dundar, A., Shih, K. J., Garg, A., Pottorf, R., Tao, A., and Catanzaro, B. Unsupervised disentanglement of pose, appearance and background from images and videos. *arXiv preprint arXiv:2001.09518*, 2020.

Esmaeili, B., Wu, H., Jain, S., Bozkurt, A., Siddharth, N., Paige, B., Brooks, D. H., Dy, J., and Meent, J.-W. Structured disentangled representations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2525–2534, 2019.

Farahmand, A.-m., Barreto, A., and Nikovski, D. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pp. 1486–1494, 2017.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018a.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Garg, S., Perot, V., Limtiaco, N., Taly, A., Chi, E. H., and Beutel, A. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 219–226, 2019.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Hallak, A., Schnitzler, F., Mann, T., and Mannor, S. Off-policy model-based learning under unknown factored dynamics. In *International Conference on Machine Learning*, pp. 711–719, 2015.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Higgins, I., Sonnerat, N., Matthey, L., Pal, A., Burgess, C. P., Bošnjak, M., Shanahan, M., Botvinick, M., Hassabis, D., and Lerchner, A. SCAN: Learning hierarchical compositional visual concepts. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rkN2Il-RZ`.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pp. 12498–12509, 2019.

Kaelbling, L. P. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Kearns, M. and Koller, D. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, pp. 740–747, 1999.

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

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

Klimov, O. and Schulman, J. Roboschool, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lachapelle, S., Brouillard, P., Deleu, T., and Lacoste-Julien, S. Gradient-based neural dag learning. *arXiv preprint arXiv:1906.02226*, 2019.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. *arXiv preprint arXiv:1810.00825*, 2018.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Li, R., Jabri, A., Darrell, T., and Agrawal, P. Towards practical multi-object manipulation using relational reinforcement learning. *arXiv preprint arXiv:1912.11032*, 2019.

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.

Locatello, F., Bauer, S., Lucic, M., Rätsch, G., Gelly, S., Schölkopf, B., and Bachem, O. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pp. 6379–6390, 2017.

Lu, C., Schölkopf, B., and Hernández-Lobato, J. M. Deconfounding reinforcement learning in observational settings. *arXiv preprint arXiv:1812.10576*, 2018.

Lyu, D., Yang, F., Liu, B., and Gustafson, S. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

Madumal, P., Miller, T., Sonenberg, L., and Vetere, F. Explainable reinforcement learning through a causal lens. *arXiv preprint arXiv:1905.10958*, 2019.

Mandel, T., Liu, Y.-E., Levine, S., Brunskill, E., and Popovic, Z. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, pp. 1077–1084, 2014.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, pp. 6118–6128, 2017.

Park, J. H., Shin, J., and Fung, P. Reducing gender bias in abusive language detection. *arXiv preprint arXiv:1808.07231*, 2018.

Pearl, J. Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146, 2009.

Perez, L. and Wang, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Peters, J., Janzing, D., and Schölkopf, B. *Elements of causal inference: foundations and learning algorithms*. MIT press, 2017.

Pitis, S., Chan, H., and Ba, J. Protoge: Prototype goal encodings for multi-goal reinforcement learning. *The 4th Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM2019)*, 2019.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Rezende, D. J., Danihelka, I., Papamakarios, G., Ke, N. R., Jiang, R., Weber, T., Gregor, K., Merzic, H., Viola, F., Wang, J., et al. Causally correct partial models for reinforcement learning. *arXiv preprint arXiv:2002.02836*, 2020.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015a.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Thomas, P. and Brunskill, E. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pp. 2139–2148, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, M. Z. and Hayden, B. Y. Monkeys are curious about counterfactual outcomes. *Cognition*, 189:1–10, 2019.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P., and Lerchner, A. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

Weber, T., Heess, N., Buesing, L., and Silver, D. Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*, 2019.

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. Variance reduction for policy gradient with action-dependent factorized baselines. *arXiv preprint arXiv:1803.07246*, 2018.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Zhu, G., Huang, Z., and Zhang, C. Object-oriented dynamics predictor. In *Advances in Neural Information Processing Systems*, pp. 9804–9815, 2018.

Algorithm 1 Mask-based Counterfactual Data Augmentation (CoDA)

function CODA(transition t1, transition t2):
  s1, a1, s1′ ← t1
  s2, a2, s2′ ← t2
  m1, m2 ← MASK(s1, a1), MASK(s2, a2)
  D1 ← COMPONENTS(m1)
  D2 ← COMPONENTS(m2)
  d ← random sample from (D1 ∩ D2)
  s̃, ã, s̃′ ← copy(s1, a1, s1′)
  s̃[d], ã[d], s̃′[d] ← s2[d], s2[d], s2′[d]
  D̃ ← COMPONENTS(MASK(s̃, ã))
  return (s̃, ã, s̃′) **if** d ∈ D̃ **else** ∅

function MASK(state s, action a):
  Returns $(|\mathcal{S}| + |\mathcal{A}|) \times (|\mathcal{S}|)$ matrix indicating if the next state components (columns) locally depend on the state and action components (rows).

function COMPONENTS(mask m):
  Using the mask as the adjacency matrix for $\mathcal{G}^{\mathcal{L}}$ (with dummy columns for next action), finds the set of connected components $C = \{C_j\}$, and returns the set of independent components
  $D = \{\mathcal{G}_i = \bigcup_k \mathcal{C}_k^i \,|\, \mathcal{C}^i \subset \text{powerset}(C)\}$.

A. Additional details on CoDA Algorithm

The pseudo-code for our mask-based CoDA algorithm is in Algorithm 1. We apply this to different transition pairs to produce novel counterfactual samples each time.

Note that masks m_1, m_2 and the counterfactual mask $\tilde{m}$ correspond to different neighborhoods $\mathcal{L}_1, \mathcal{L}_2$ and $\tilde{\mathcal{L}}$, so it is not clear that we are "within the bounds" of any model $\mathcal{M}^{\mathcal{L}}$ as was required in Section 2 for valid counterfactual reasoning. To correct this discrepancy we use the following proposition and additionally require the causal mechanisms for the independent components $\mathcal{G}_i$ and $\mathcal{G}_j$ to share structural equations in each local neighborhood: $f^{\mathcal{L}_1,i} = f^{\mathcal{L}_2,i} = f^{\tilde{\mathcal{L}},i}$ (and similarly for j). To see why this is not trivially true, imagine there are two rooms, one of which is icy. In either room the ground conditions are locally independent of movement dynamics, but not so if we consider their union. Using Proposition 1 below, this makes our reasoning valid in the local subspace $\mathcal{L}^* = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \tilde{\mathcal{L}}$.

Lemma 1. *If $V^j \in Pa^{\mathcal{L}}(V^i)$ in DAG $G^{\mathcal{L}}$ of (local) causal model $\mathcal{M}^{\mathcal{L}}$, and $\mathcal{L} \subset \mathcal{X}$, then $V^j \in Pa^{\mathcal{X}}(V^i)$ in DAG $G^{\mathcal{X}}$ corresponding to causal model $\mathcal{M}^{\mathcal{X}}$.*

Proof. By minimality, there exist $\{u^i, v^{-j}, v_1^j\}$ and $\{u^i, v^{-j}, v_2^j\}$ with $v^{-j} \in \text{Pa}^{\mathcal{L}}(V^i) \setminus V^j$ for which $f^i(\{u^i, v^{-j}, v_1^j\}) \neq f^i(\{u^i, v^{-j}, v_2^j\})$. Expand $\{v^{-j}, v_1^j\}$ and $\{v^{-j}, v_2^j\}$ to $(s_1, a_1), (s_2, a_2) \in \mathcal{L}$ (with any values of other components). But $\mathcal{L} \subset \mathcal{X}$, so $(s_1, a_1), (s_2, a_2) \in \mathcal{X}$ and it follows from minimality in $\mathcal{X}$ that $V^j \in \text{Pa}^{\mathcal{X}}(V^i)$. □

Corollary 1. *If $\mathcal{L} \subset \mathcal{X}$, $G^{\mathcal{L}}$ is sparser (has fewer edges) than $G^{\mathcal{X}}$.*

Proposition 1. *The mechanisms represented by $\mathcal{G}_i, \mathcal{G}_j \subset \mathcal{G}$ are independent in $G^{\mathcal{L}_1 \cup \mathcal{L}_2}$ iff $\mathcal{G}_i$ and $\mathcal{G}_j$ are independent in both $\mathcal{G}^{\mathcal{L}_1}$ and $\mathcal{G}^{\mathcal{L}_2}$, and $f^{\mathcal{L}_1,i} = f^{\mathcal{L}_2,i}$, $f^{\mathcal{L}_1,j} = f^{\mathcal{L}_2,j}$.*

Proof. ($\Rightarrow$) If $\mathcal{G}_i$ and $\mathcal{G}_j$ are independent in $G^{\mathcal{L}_1 \cup \mathcal{L}_2}$, independence in $\mathcal{G}^{\mathcal{L}_1}$ and $\mathcal{G}^{\mathcal{L}_2}$ follows from Corollary 1. That $f^{\mathcal{L}_1,i} = f^{\mathcal{L}_2,i}$ (and $f^{\mathcal{L}_1,j} = f^{\mathcal{L}_2,j}$), on their shared domain, follows since each is a restriction of the same function $f^{\mathcal{L}_1 \cup \mathcal{L}_2,i}$ (or $f^{\mathcal{L}_1 \cup \mathcal{L}_2,j}$).

($\Leftarrow$) Suppose $\mathcal{G}_i$ and $\mathcal{G}_j$ are independent in $\mathcal{G}^{\mathcal{L}_1}$ and $\mathcal{G}^{\mathcal{L}_2}$ but not $G^{\mathcal{L}_1 \cup \mathcal{L}_2}$. By the definition of independence applied to $G^{\mathcal{L}_1 \cup \mathcal{L}_2}$, we have that, without loss of generality, there is a $V_i \in \mathcal{G}_i, V_j \in \mathcal{G}_j$ with $V_j \in \text{Pa}^{\mathcal{L}_1 \cup \mathcal{L}_2}(V_i)$. Then, from the definition of minimality, it follows that there exist $(s_1, a_1), (s_2, a_2) \in \mathcal{L}_1 \cup \mathcal{L}_2$ that differ only in the value of V_j, and $u_i \in \text{range}(U_i)$ for which $f^i(s_1, a_1, u_i) \neq f^i(s_2, a_2, u_i)$.

Clearly, if (s_1, a_1) and (s_2, a_2) are both in $\mathcal{L}_1$ (or $\mathcal{L}_2$), there will be an edge from V_j to V_i in $\mathcal{G}^{\mathcal{L}_1}$ (or $\mathcal{G}^{\mathcal{L}_2}$) and the claim follows by contradiction. Thus, the only interesting case is when, without loss of generality, $(s_1, a_1) \in \mathcal{L}_1$ and $(s_2, a_2) \in \mathcal{L}_2$. The key observation is that (s_1, a_1) and (s_2, a_2) differ only in the value of node $V_j \notin \mathcal{G}_i$: since $\mathcal{G}_i$ is an independent causal mechanism in both $\mathcal{G}^{\mathcal{L}_1}$ and $\mathcal{G}^{\mathcal{L}_2}$ and the parents of V_i take on the same values in each, we have that $f^i(s_1, a_1, u_i) = f^{i,\mathcal{L}_1}(s_1, a_1, u_i) = f^{i,\mathcal{L}_2}(s_2, a_2, u_i) = f^i(s_2, a_2, u_i)$ and the claim follows by contradiction. □

B. Related Work

Factored MDPs (Guestrin et al., 2003; Hallak et al., 2015; Weber et al., 2019) consider MDPs where state variables are only influenced by a fixed subset of "parent" variables at the previous timestep. Object-oriented and relational approaches to RL and sequence prediction (Diuk et al., 2008; Goyal et al., 2019; Kipf et al., 2019; Li et al., 2019; Zambaldi et al., 2018; Zhu et al., 2018) represent the dynamics as a set of interacting entities. Factored action spaces and policies have been used to formulate dimension-wise policy gradient baselines in both standard and multi-agent settings (Foerster et al., 2018; Lowe et al., 2017; Wu et al., 2018).

A growing body of work applies causal reasoning the RL setting to improve sample efficiency, interpretability and learn better representations (Lu et al., 2018; Madumal et al., 2019; Rezende et al., 2020). Particularly relevant is the work by Buesing et al. (2019), which improves sample efficiency by using a causal model to sample counterfactual trajectories, thereby reducing variance of off-policy gradient estimates

in a guided policy search framework. These counterfactuals use coarse-grained representations at the trajectory level, while our approach uses factored representations within a single transition.

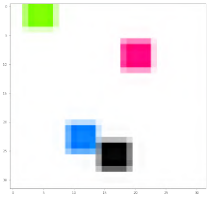
Batch RL (Levine et al., 2020; Fujimoto et al., 2018a; Mandel et al., 2014) and more generally off-policy RL (Watkins & Dayan, 1992; Munos et al., 2016) are counterfactual by nature, and are particularly important when it is costly or dangerous to obtain on-policy data (Thomas & Brunskill, 2016). The use of counterfactual goals to accelerate learning goal-conditioned RL (Kaelbling, 1993; Schaul et al., 2015a; Plappert et al., 2018) is what inspired our local CoDA algorithm.

Data augmentation is also widely used in supervised learning, and is considered a required best practice in high dimensional problems (Krizhevsky et al., 2012; Laskin et al., 2020; Perez & Wang, 2017). Heuristics for data augmentation often encode a causal invariance statement with respect to certain perturbations on the inputs, which is relevant to applications where *fairness* is a concern, as counterfactuals can be used to achieve robust performance and debias data (Garg et al., 2019; Park et al., 2018; Bolukbasi et al., 2016).

C. Additional Experiment Details

This section provides training details for the experiments discussed in Section 4 as well as some additional results.

Standard online RL We extend `Spriteworld` (Watters et al., 2019) to construct a "bouncing ball" environment (right), that consists of multiple objects (sprites) that move and collide within a confined 2D canvas. We use tasks of varying difficulty, where the agent must navigate $N \in \{1, 2, 3, 4\}$ of 4 sprites to their fixed target positions. The agent receives reward of $1/N$ for each of the N sprites placed; e.g., the hardest task (Place 4) gives $1/4$ reward for each of 4 sprites placed. For each task, we use CoDA to expand the replay buffer of a TD3 agent (Fujimoto et al., 2018b) by about 8 times. We compare CoDA with a ground truth masking function (available via the `Spriteworld` environment) and learned masking function to the base TD3 agent, as well as a Dyna agent that generates additional training data by sampling from a model. For fair comparison, we use the same transformer used for CoDA masks for Dyna, which we pretrain using approximately 42,000 samples from a random policy. As in HER, we assume access to the ground truth reward function to relabel the rewards. The results in Figure 5 show that both variants of CoDA significantly improve sample complexity over the baseline. By contrast, the Dyna agent suffers from model bias, even though it uses the same model as CoDA. Additional details are included below.



Implementation We work with the original TD3 codebase, architecture, and hyperparameters (except batch size; see below), and focus our efforts solely on modifying the agent's training distribution.

Environment We extend the base Spriteworld framework (Watters et al., 2019) with (1) basic collisions (to induce a local factorization / so that a global factorization will not work), (2) a new continuous action space (2-dimensional), (3) a disentangled state renderer that returns the position and velocity of each sprite (a total of 16-dimensions in tasks with four sprites), (4) a mask renderer that returns the ground truth masking function (allows us to evaluate our masking function in Appendix D), and (5) a suite of partial and sparse reward tasks that we use for experiments. These extensions will included with the release of our code.

Data augmentation Every 1000 environment steps, we sample 2000 pairs of random transitions from the agent's replay buffer, and apply CoDA to produce a maximum of 5 unique CoDA samples per pair. We apply two forms of CoDA, using (1) an oracle / ground truth mask function that we back out of the simulator, and (2) the mask of a pre-trained transformer model (see Section D). The mask function was trained using approximately 42,000 samples from a random policy (5/6 of 50,000, with the rest of the data used for validation). CoDA samples are added to a second CoDA replay buffer. For purposes of this experiment both buffers are have effectively infinite capacity (they are never filled). During training, the agent's batches are sampled proportionally from the real and CoDA replay buffers (this means that approximately 7/8 of the data that the agent trains on is counterfactual).

Baselines In addition to the base TD3 agent and CoDA, we also use the transformer model that is used as a mask function to generate data by performing forward rollouts with a random policy, as in Dyna (Sutton, 1991). So that this baseline produces approximately the same number of samples as CoDA with the learned mask, we roll the model out for 5 steps from 1500 random samples from the replay buffer, again every 1000 environment steps. This produces 7500 model-based samples every 1000 environment steps.

Use of the transformer mask function requires setting the threshold value τ, which we do by monitoring accuracy and F1 scores for sparsity prediction (as discussed in Appendix D) on validation data, ultimately using the value $\tau = 0.05$.

Batch Size Since CoDA samples are plentiful we increase the agent's batch size from 256 to 1000 to allow it to train on

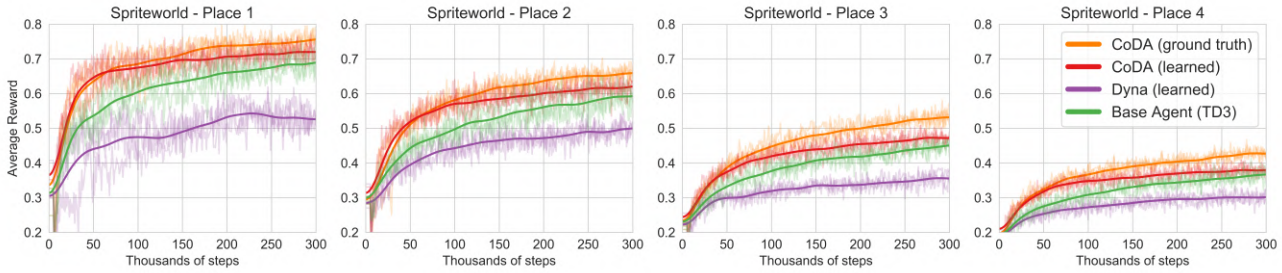


Figure 5. **Standard RL** (3 seeds, partial reward): CoDA with the ground truth mask performs best, validating our basic idea. CoDA with a pretrained model also offers a significant early boost in sample efficiency and maintains its lead over the base TD3 agent throughout training. Using the same model to generate data directly (a la Dyna, Sutton (1991)) performs poorly, suggesting significant model bias.

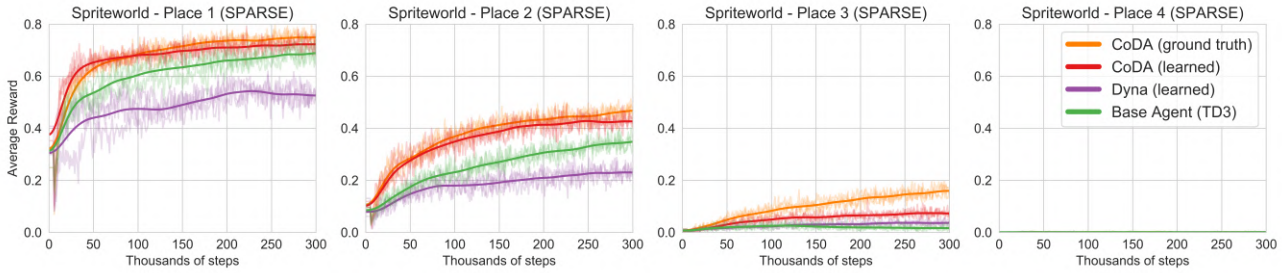


Figure 6. **Standard RL** (3 seeds, dense reward): As in the partial reward case (Figure 5), we observe that CoDA agents outperform the other agents (except in Place 4, where no agent achieves any reward).

more environment samples in the same number of training steps. We found that this slightly improved the performance of the base TD3 agent. An increase in batch size also allows the agent to see more of its own on-policy data in the face of many typically off-policy CoDA samples.

Additional results in sparse reward task variants In addition to the partial reward tasks described above, we also tested CoDA in four sparse reward tasks of varying difficulty. These are the same as the partial reward tasks, except that a sparse reward of 1 is granted only when *all* N sprites are in their target locations. While these tasks were much harder (and perhaps impossible in the case of Place 4 due to moving sprites), as shown in Figure 6, the CoDA agents maintain a clear advantage.

C.1. Batch RL

Here we detail the procedure used in the Batch RL experiments in the `Pong` environment.

Implementation This experiment works with a different codebase than the Spriteworld experiment, in order to simplify the use of CoDA in a Batch RL setting. Our experiment first builds the agent's dataset (consisting of real data, dyna data and/or CoDA data), then instantiates a TD3 agent by filling its replay buffer with the dataset. The replay buffer is always expanded to include the entire enlarged dataset (for the 5x CoDA ratio at 250,000 data size this means the buffer has 1.5E6 experiences). The agent is run for 500,000 optimization steps.

Hyperparameters We used similar hyperparameters to the original TD3 codebase, with the following differences:

- We use a discount factor of $\gamma = 0.98$ instead of 0.99.
- Since Pong is a sparse reward task with $\gamma = 0.98$, we clip critic targets to $(-50, 50)$.
- We use networks of size $(128, 128)$ instead of $(256, 256)$.
- We use a batch size of 1000.

Environment We base our `Pong` environment on `RoboSchoolPong-v1` (Klimov & Schulman, 2017). The original environment allowed the ball to teleport back to the center after one of the players scored, offered a small dense reward signal for hitting the ball, and included a stray "timeout" feature in the agent's state representation. We fix the environment so that the ball does not teleport, and instead have the episode reset every 150 steps, and also 10 steps after either player scores. The environment is treated as continuous and never returns a done signal that is not

also accompanied by a `TimeLimit.truncated` indicator (Brockman et al., 2016). We change the reward to be strictly sparse, with reward of ± 1 given when the ball is behind one of the players' paddles. Finally, we drop the stray "timeout" feature, so that the state space is 12-dimensional, where each set of 4 dimensions is the x-position, y-position, x-velocity, and y-velocity of the corresponding object (2 paddles and one ball).

Training the CoDA model Without access to a ground truth mask, we needed to train a masking function D to identify local disentanglement. We also forewent the ground truth reward, instead training our own reward classifier. In each case we used the batch dataset given, and so we trained different models for each random seed. For our masking model, we stacked two single-head transformer blocks (without positional encodings) and used the product of their attention masks as the mask. Each block consists of query Q, key K, and value V networks that each have 3-layers of 256 neurons each, with the attention computed as usual (Vaswani et al., 2017; Lee et al., 2018). The transformer is trained to minimize the L2 error of the next state prediction given the current state and action as inputs. The input is disentangled, and so has shape `(batch_size, num_components, num_features)`. In each row (component representation) of each sample, features corresponding to other components are set to zero. The transformer is trained for 2000 steps with a batch size of 256, Adam optimizer (Kingma & Ba, 2014) with learning rate of 3e-4 and weight decay of 1e-5. For our reward function we use a fully-connected neural network with 1 hidden layer of 128 units. The reward network accepts an (s, a, s') tuple as input (not disentangled) and outputs a softmax over the possible reward values of $[-1, 0, 1]$. It is trained for 2000 steps with a batch size of 512, Adam optimizer (Kingma & Ba, 2014) with learning reate of 1e-3 and weight decay of 1e-4. All hyperparameters were rather arbitrary (we used the default setting, or in case it did not work, the first setting that gave reasonable results, as was determined by inspection). To ensure that our model and reward functions are trained appropriately (i.e., do not diverge) for each seed, we confirm that the average loss of the CoDA model is below 0.005 at the training and that the average loss of the reward model is below 0.1, which values were found by inspection of a prototype run. These conditions were met by all seeds.

When used to produce masks, we chose a threshold of $\tau = 0.02$ by inspection, which seemed to produce reasonable results. A more principled approach would do cross-validation on the available data, as we did for Spriteworld (Appendix D).

Tested configurations We considered the following configurations:

1. **Real data only**.
2. **CoDA + real data**: after training the CoDA model, we expand the base dataset by either 2, 4 or 6 times.
3. **Dyna (using CoDA model)**: after training the CoDA model, we use it as a forward dynamics model instead of for CoDA; we tried 1-step and 5-step rollouts with random actions from random states in the given dataset to expand the dataset by 2x. We found that this hurt performance at all data sizes (not shown).
4. **Dyna (using MBPO model)**: as the CoDA model exhibits significant model bias when used as a forward dynamics model, we replicate the state-of-the-art model-based architecture used by MBPO (Janner et al., 2019) and use it as a forward dynamics model for Dyna; we experimented with 1-step and 5-step rollouts with random actions from random states in the given dataset to expand the dataset by 2x. This time we found that the 5-step rollouts do better, which we attribute to the lower model bias together with the ability to create a more diverse dataset (1-step not shown). The MBPO model is described below. We use the same reward model as CoDA to relabel rewards for the MBPO model, which only predicts next state.
5. **MBPO + CoDA**: as MBPO improved performance over the baseline (real data only) at lower dataset sizes, we considered using MBPO together with CoDA. We use the base dataset to train the MBPO, CoDA, and reward models, as described above. We then use the MBPO model to expand the base dataset by 2x, as described above. We then use the CoDA model to expand *the expanded dataset* by 3x the original dataset size. Thus the final dataset is 5x as large as the original dataset (1 real : 1 MBPO : 3 CoDA).

All configurations alter only the training dataset, and the same agent architecture/hyperparameters (reported above) are used in each case.

MBPO model Since using the CoDA model for Dyna harms rather than helps, we consider using a stronger, state-of-the-art model-based approach. In particular, we adopt the model used by Model-Based Policy Optimization (Janner et al., 2019). This model consists of a size 7 ensemble of neural networks, each with 4 layers of 200 neurons. We use ReLU activations, Adam optimizer (Kingma & Ba, 2014) with weight decay of 5e-5, and have each network output a the mean and (log) diagonal covariance of a multi-variate Gaussian. We train the networks with a maximum likelihood loss. To sample from the model, we choose an ensemble member uniformly at random and sample from its output distribution, as done by MBPO.

C.2. Goal-conditioned RL

Here we detail the procedure used in the Goal-conditioned RL experiments on the `Fetchpush-v1` and `Slide2` environments.

Implementation This experiment uses the same codebase as our Batch RL, which provides state-of-the-art baseline HER agents and will be released with the paper.

Hyperparameters For `Fetchpush-v1` we use the default hyperparameters from the codebase, which outperform the original HER agents of (Andrychowicz et al., 2017; Plappert et al., 2018) and follow-up works. We do not tune the CoDA agent (but see additional CoDA hyperparameters below). They are as follows:

- Off-policy algorithm: DDPG (Lillicrap et al., 2015)
- Hindsight relabeling strategy: `futureactual_2_2` (Pitis et al., 2019), using exclusively `future` (Andrychowicz et al., 2017) relabeling for the first 25,000 steps
- Optimizer: Adam (Kingma & Ba, 2014) with default hyperparameters
- Batch size: 2000
- Optimization frequency: 1 optimization step every 2 environment steps after the 5000th environment step
- Target network updates: update every 10 optimization steps with a Polyak averaging coefficient of 0.05
- Discount factor: 0.98
- Action l2 regularization: 0.01
- Networks: 3x512 layer-normalized (Ba et al., 2016) hidden layers with ReLU activations
- Target clipping: (-50, 0)
- Action noise: 0.1 Gaussian noise
- Epsilon exploration (Plappert et al., 2018): 0.2, with an initial 100% exploration period of 10,000 steps
- Observation normalization: yes
- Buffer size: 1M

On `Slide2` we tried to tune the baseline hyperparameters somewhat, but note that this is a fairly long experiment (10M timesteps) and so only a few settings were tested. In particular, we considered the following modifications:

- Expanding the replay buffer to 2M (effective)
- Reducing the batch size to 1000 (effective)* (used for results)
- Using the `future_4` strategy (agent fails to learn in 10M steps)
- Reducing optimization step frequency to 1 step every 4 environment steps (about the same performance)

We tried similar adjustments to our CoDA agent, but found the default hyperparameters (used for results) performed well. We found that the CoDA agent outperforms the base HER agent on all tested settings.

For CoDA, we used the additional hyperparameters:

- CoDA buffer size: 3M
- Make CoDA data every: 250 environment steps
- Number of source pairs from replay buffer used to make CoDA data: 2000
- Number of CoDA samples per source pair: 2
- Maximum ratio of CoDA:Real data to train on: 3:1

Environment On `FetchPush-v1` the standard state features include the relative position of the object and gripper, which entangles the two. While this could be dealt with by dynamic relabeling (as used for HER's reward), we simply drop the corresponding features from the state.

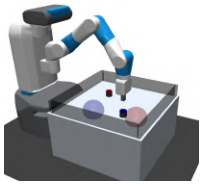


Figure 7. `Slide2`.

`Slide2` has two pucks that slide on a table and bounce off of a solid railing. Observations are 40-dimensional (including the 6-dimensional goal), and actions are 4-dimensional. Initial positions and goal positions are sampled randomly on the table. During training, the agent gets a sparse reward of 0 (otherwise -1) if *both* pucks are within 5cm of their ordered target. At test time we count success as having both picks within 7.5cm of the target on the last step of the episode. Episodes last 75 steps and there is no done signal (this is intended as a continuous task).

CoDA Heuristic For these experiments we use a hand-coded heuristic designed with domain knowledge. In particular, we assert that the action is always entangled with the gripper, and that gripper/action and objects (pucks or blocks) are disentangled whenever they are more than 10cm apart. This encodes independence due to physical separation, which we hypothesize is a very generally heuristic that humans implicitly rely on all the time. The pucks have a radius of 2.5cm and height of 4cm, and the blocks are 5cm x 5cm x 5cm, so this heuristic is quite generous / suboptimal. Despite being suboptimal, it demonstrates the ease with which domain knowledge can be injected via the CoDA mask: we need only a high precision (low false positive rate) heuristic—the recall is not as important. It is likely that an agent could learn a better mask that also takes into account velocity.

D. Inferring Local Factorization

Here we present several approaches to inferring the local factorization of subspaces, a crucial subroutine of CoDA. We note that in many cases where domain knowledge is available, simple heuristics may suffice, e.g. in our Goal-conditioned RL experiments discussed in Section 4 where a simple distance-based indicator function in the state space was used. However, as such heuristics may not be universally available, the question of whether data-driven approaches can successfully infer local factorization is of general interest. We note that the performance of CoDA will improve alongside future improvements in this inference task (motivating future work in this area), and that inferring the local factorization in general is an easier task than learning the environment dynamics.

We begin by presenting two methods for inferring local factorization, derived from variants of a next-state prediction task, which we here refer to as SANDy for Sparse Attention Neural Dynamics. To verify the merits of SANDy in the local factorization inference task, we evaluate two SANDy variants in controlled settings where the ground truth factorization is known: first in a synthetic Markov process (MDP without actions), and second in Spriteworld. This label information is used only to evaluate performance, and not to train the SANDy parameters. The SANDy-Transformer model was used for the Online RL experiments presented in Section 4.

D.1. Methods

We propose two Sparse Attention for Neural Dynamics (SANDy) models. In each case we seek to learn a function (or mask) $M(s,a) \to \{0,1\}^{(|S|+|A|)\times|S|}$ whose output represents the adjacency matrix of the local causal graph, conditioned on the state and action. We note that $M(s,a)_{i,j} = 0$ alone is insufficient, in general, to determine the local subspace $\mathcal{L} \subset \mathcal{S} \times \mathcal{A}$, since there may be multiple disconnected subspaces $\mathcal{L}_k$ with $M_k(s,a)_{i,j} = 0$ whose union $\bigcup_k \mathcal{L}_k$ has $M_{\cup k}(s,a)_{i,j} = 1$. This can be resolved by our Proposition 1 if we also force the relevant structural equations to be the same. For now, we assume the mask determines the local subspace, and leave exploration of this possibility to future work. Empirically, we will see that our assumption is reasonable.

SANDy-Mixture The first model is a mixture-of-MLPs model with an attention mechanism that is computed from the current state. Each component of the mixture is a neural dynamics model with sparse local dependencies. For a given component, the key idea is to train a neural dynamics model to predict the next state $h(s_t, a_t) \approx s_{t+1}$ and approximate the masking function by thresholding the (transpose of the) network Jacobian of h, $[\mathbf{J}(s,a)]_{i,j} = \frac{\delta}{\delta[s,a]_j}[h(s,a)]_i$. Intuitively, we can think of $\mathbf{J}$ as providing the first-order element-wise dependencies between the predicted next state and the network input. We then derive the local factorization by thresholding the absolute Jacobian

$$M_\tau(s,a) = \mathbb{1}(|\mathbf{J}(s,a)| > \tau), \tag{1}$$

where $\mathbb{1}(\cdot)$ represents the indicator function and τ is a threshold hyperparameter.

To estimate the network Jacobian, we note that for standard activation functions (sigmoid, tanh, relu), it can be bound from above by the matrix product of its weight matrices. To see this, let h_θ be an L-layer MLP parameterized by $\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \cdots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)})$ with activation σ with bounded derivative $\sigma'(x) \leq 1$, and note that for each layer $\boldsymbol{h}^{(\ell)} = \sigma(\mathbf{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \mathbf{b}^{(\ell)})$ we have:

$$\left|\frac{d\boldsymbol{h}_j^{(\ell)}}{d\boldsymbol{h}_i^{(\ell-1)}}\right| \leq \left|\mathbf{W}_{ji}^{(\ell)}\right|.$$

Then, using the chain rule, triangle inequality, and the identity $|ab| = |a||b|$, we can compute:

$$\begin{aligned}\left|\frac{d\boldsymbol{h}_j^{(\ell)}}{d\boldsymbol{h}_i^{(\ell-2)}}\right| &\leq \left|\frac{d}{d\boldsymbol{h}_i^{(\ell-2)}}\mathbf{W}_{j\cdot}^{(\ell)}\boldsymbol{h}^{(\ell-1)}\right| = \left|\mathbf{W}_{j\cdot}^{(\ell)} \cdot \frac{d\boldsymbol{h}^{(\ell-1)}}{d\boldsymbol{h}_i^{(\ell-2)}}\right| \\ &\leq \sum_k \left|\mathbf{W}_{jk}^{(\ell)}\frac{d\boldsymbol{h}_k^{(\ell-1)}}{d\boldsymbol{h}_i^{(\ell-2)}}\right| \leq \left|\mathbf{W}_{j\cdot}^{(\ell)}\right| \cdot \left|\mathbf{W}_{\cdot i}^{(\ell-1)}\right|.\end{aligned}$$

Expanding this out to multiple layers, we see that $\left|\mathbf{J}_\theta(s,a)\right| \leq \prod_{\ell\in[L]} |\mathbf{W}^{(\ell)}|$, as desired. We use this upper bound to approximate the network Jacobian of an MLP by setting $\hat{\mathbf{J}} = \prod_{\ell\in[L]} |\mathbf{W}^{(\ell)}|$. A similar idea is used by (Germain et al., 2015; Lachapelle et al., 2019), among others, to control element-wise input-output dependencies.

We use this static approximation $\hat{\mathbf{J}}$ to facilitate learning a sparse dynamic mask by specifying a mixture model $h(s,a) = \sum_i \alpha_\phi^{(i)}(s,a) h_\theta^{(i)}(s,a)$ over the environment dynamics (with $\sum_i \alpha_\phi^{(i)}(s,a) = 1 \; \forall \; (s,a)$), where each component $h_\theta^{(i)}$ is an MLP as specified above with a sparsity prior on its Jacobian bound $\hat{\mathbf{J}}^{(i)}$ to encourage sparse (i.e. well-factorized) local solutions. The dynamic mask is computed by first approximating the Jacobian, $\hat{\mathbf{J}}(s,a) = \sum_i \alpha_\phi^{(i)}(s,a)\hat{\mathbf{J}}^{(i)}$, then thresholding by τ as in Equation 1.

Note that we assume the mixture components α are a function of the current state and action alone; in other words the factorization (captured by the network Jacobian of each component) can be *locally inferred*. The next-state prediction is given by $\hat{\mathbf{s}}_{t+1} = (h_\theta^{(1)}(\mathbf{s}_t, \mathbf{a}_t), \cdots, h_\theta^{(N)}(\mathbf{s}_t, \mathbf{a}_t))^T \boldsymbol{\alpha}_\phi(\mathbf{s}_t, \mathbf{a}_t)$. To train the model, we optimize the objective:

$$\underset{\theta,\phi}{\text{minimize}} \; \frac{1}{|\mathcal{D}|}\Big(\sum_{(\mathbf{s}_t, \mathbf{a}_t \mathbf{s}_{t+1}) \in \mathcal{D}} ||\mathbf{s}_{t+1} - h(\mathbf{s}_t, \mathbf{a}_t)||_2^2 \Big) + \lambda_1 S(\theta) + \lambda_2 R(\phi) + \lambda_3 ||(\theta, \phi)||_2 \tag{2}$$

where $S(\theta) = \frac{1}{K}\sum_i |\hat{\mathbf{J}}_\theta^{(i)}|_1$ puts an ℓ_1 prior on each mixture component to induce sparsity, and $R(\phi)$ encourages high entropy in the attention probabilities

$$R(\phi) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{s} \in \mathcal{D}} \sqrt{\frac{1}{N} \sum_{j \in [N]} [A_\phi(\mathbf{s}, \mathbf{a})]_j}.$$

We note that more sophisticated methods of computing Jacobians of neural networks–including architectural changes and optimization strategies (Arjovsky et al., 2017)—have been proposed, and could in principle be used here as well.

SANDy-Transformer As an alternative model, we use a transformer-like architecture that applies self-attention between a set of (potentially multi-dimensional) inputs (Vaswani et al., 2017; Lee et al., 2018). Our architecture is composed of a stacked self-attention blocks. Each block accepts a set of inputs $X = \{x_i \in \mathbb{R}^n\}$ and composes three functions of each input: query $Q : \mathbb{R}^n \to \mathbb{R}^d$, key $K : \mathbb{R}^n \to \mathbb{R}^d$, and value $V : \mathbb{R}^n \to \mathbb{R}^m$. The block returns a set of outputs $\{y_i \in \mathbb{R}^m\}$ of size $|X|$, each of which is computed as: $y_i = A_i^T V(X)$, where $A_i = \text{softmax}\big(\sum_j (Q(x_i)_j K(x_j)_i)\big)$ (note that $V(X)$ is a matrix of size $|X| \times m$). We approximate the block mask (approximate Jacobian) as $A = [A_1, A_2, \ldots, A_{|X|}] \in \mathbb{R}^{|X| \times |X|}$, and the full network mask as the product of the block masks (as in the SANDy-Mixture). We used two-layer MLPs for each function K, Q, V in Spriteworld and three-layer MLPs in Pong.

To apply this architecture to our problem, we first embed each state and action component (single feature, or group of features) into $\mathbb{R}^n$ to produce a set of inputs X and pass this through each stacked self-attention block to obtain a set of outputs Y. We then discard any output components that correspond to the action features to obtain the next state prediction and mask. The network h is trained to minimize mean squared error:

$$\underset{\theta}{\text{minimize}} \; \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t \mathbf{s}_{t+1}) \in \mathcal{D}} ||\mathbf{s}_{t+1} - h(\mathbf{s}_t, \mathbf{a}_t)||_2^2. \tag{3}$$

Unlike the SANDy-Mixture, no sparsity regularizers are applied, as we found the sparsity induced by the softmax attention mechanism to be sufficient.

D.2. Evaluation environments

Synthetic Markov Processes We investigate the capacity of the SANDy-Mixture to learn simple factorized transition dynamics under a globally factored Markov Process (StationaryMP). Unlike the general MDP setting, no agent/policy is considered. However the ability to train an unconstrained dynamics model to approximate factorized environment dynamics is an important subtask within our overall approach. Assuming a spherical Gaussian prior over the initial states $\mathbf{s}^0 \in \mathbb{R}^9$, the StationaryMP is entirely specified by transition distribution $p(\mathbf{s}^{t+1}|\mathbf{s}^t)$. We assume that the state transitions factorize (globally) into three parts. Denoting by $\mathbf{s}^t_{n\cdots m}$ the n-th through m-th dimensions of the time t state $\mathbf{s}^t$, we have:

$$p(\mathbf{s}^{t+1}_{1\cdots 9}|\mathbf{s}^t_{1\cdots 9}) = p(\mathbf{s}^{t+1}_{1\cdots 4}|\mathbf{s}^t_{1\cdots 4}) p(\mathbf{s}^{t+1}_{5\cdots 7}|\mathbf{s}^t_{5\cdots 7}) p(\mathbf{s}^{t+1}_{8,9}|\mathbf{s}^t_{8,9}).$$

In other words, we have a block-diagonal transition matrix comprising three blocks with sizes 4, 3, and 2, respectively. In our case, all transition factors are deterministic non-linear mappings, e.g. $p(\mathbf{s}^{t+1}_{1\cdots 4}|\mathbf{s}^t_{1\cdots 4}) = \delta(g_{1\cdots 4}(\mathbf{s}^t_{1\cdots 4}))$, with

$g_{1\cdots4} : \mathbb{R}^4 \to \mathbb{R}^4$ is a randomly-initialized single-hidden-layer neural network with 32 hidden units and GELU nonlinearity (Hendrycks & Gimpel, 2016). In this case, we can alternatively express the deterministic dynamics via the transition function

$$\begin{aligned}\mathbf{s}^{t+1} &= (\mathbf{s}^{t+1}_{1\cdots4}, \mathbf{s}^{t+1}_{5\cdots7}, \mathbf{s}^{t+1}_{8,9}) \\ &= (g_{1\cdots4}(\mathbf{s}^{t}_{1\cdots4}), g_{5\cdots7}(\mathbf{s}^{t}_{5\cdots7}), g_{8,9}(\mathbf{s}^{t}_{8,9})).\end{aligned} \tag{STATIONARYMP}$$

We now turn to a more sophisticated MP with locally factored dynamics (ϵ-NONSTATIONARYMP), and investigate whether the SANDy-Mixture can learn to recognize local disentanglement.

We refer to the component functions $g_{1\cdots4} : \mathbb{R}^4 \to \mathbb{R}^4$, $g_{5\cdots7} : \mathbb{R}^3 \to \mathbb{R}^3$, and $g_{8,9} : \mathbb{R}^2 \to \mathbb{R}^2$ used in the STATIONARYMP as *local* transitions. We now introduce *global* interactions via the global transition functions, $G_{1\cdots4} : \mathbb{R}^4 \to \mathbb{R}^9$, $G_{5\cdots7} : \mathbb{R}^4 \to \mathbb{R}^9$, and $G_{8,9} : \mathbb{R}^4 \to \mathbb{R}^9$, which respectively map the local state factors onto the global state space[1]. This allows us to extend the STATIONARYMP by adding global state transitions to the local state transitions whenever the norm of the local state factors exceeds the value of a hyperparameter ϵ (lower ϵ indicates more global interaction). Denoting by $\mathbb{1}(\cdot)$ the indicator function, we have

$$\begin{aligned}\mathbf{s}^{t+1} &= \left[(\mathbf{s}^{t+1}_1, \mathbf{s}^{t+1}_2, \mathbf{s}^{t+1}_3, \mathbf{s}^{t+1}_4), (\mathbf{s}^{t+1}_5, \mathbf{s}^{t+1}_6, \mathbf{s}^{t+1}_7), (\mathbf{s}^{t+1}_8, \mathbf{s}^{t+1}_9)\right] \\ &= \left[g_{1\cdots4}(\mathbf{s}^{t}_{1\cdots4}), g_{5\cdots7}(\mathbf{s}^{t}_{5\cdots7}), g_{8,9}(\mathbf{s}^{t}_{8,9})\right] \\ &\quad + G_{1\cdots4}(\mathbf{s}^{t}_{1\cdots4})\mathbb{1}(||\mathbf{s}^{t}_{1\cdots4}||_2 > \epsilon) \\ &\quad + G_{5\cdots7}(\mathbf{s}^{t}_{5\cdots7})\mathbb{1}(||\mathbf{s}^{t}_{5\cdots7}||_2 > \epsilon) \\ &\quad + G_{8,9}(\mathbf{s}^{t}_{8,9})\mathbb{1}(||\mathbf{s}^{t}_{8,9}||_2 > \epsilon).\end{aligned} \tag{ϵ-NONSTATIONARYMP}$$

Spriteworld Since we extended `Spriteworld` with a ground truth mask renderer, we are able to directly evaluate our SANDy models in `Spriteworld` as well. See the main text and Appendix C for a description of the `Spriteworld` environment.

D.3. Results

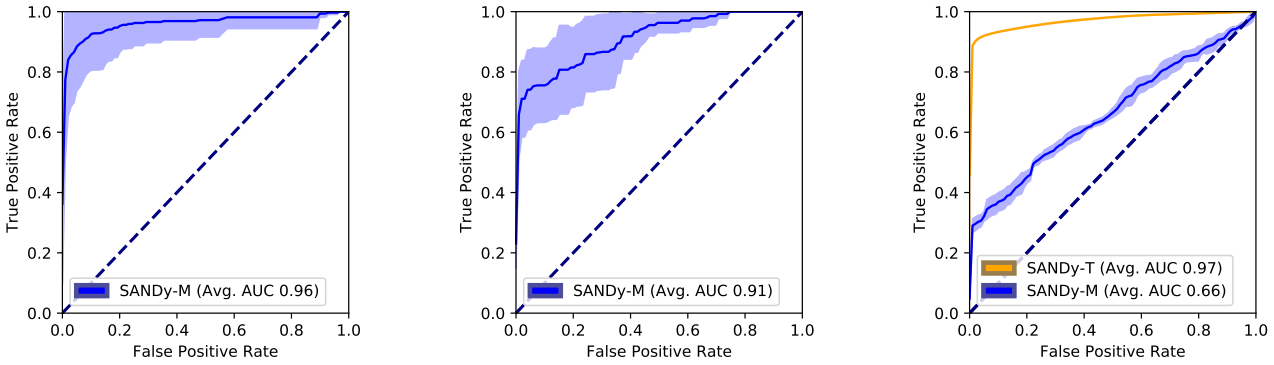
In this Subsection we measure ability of the proposed SANDy algorithm (in its two variants) to correctly infer local factorization. At each transition we can query the environment for the ground-truth connectivity pattern of the local causal graph: given $|S| + |A|$ dimensions of current state and action and $|S|$ dimensions of next state, this corresponds an adjacency matrix $\mathbf{Y} \in \{0, 1\}^{|S|+|A|\times|S|}$.We note that accessing these evaluation labels—which are not used to train SANDy—requires a controlled synthetic environment like the ones we consider, and we leave design of an evaluation protocol suitable for real-world environments to future work.

We learn the SANDy network parameters using a training dataset of $40,000$ transitions, with an additional validation dataset of $10,000$ transitions used for early stopping and hyperparameter selection. We used the Adam optimizer with learning rate of 0.001 and default hyperparameters. In the ϵ-NONSTATIONARYMP, we set $\epsilon = 1.5$, while in the Spriteworld setting we collect training trajectories by deploying a random-action agent in the environment, and randomly resetting the environment with 5% probability at every step to increase diversity of experiences. We then evaluate the SANDy models by computing local factorizations $M_\tau(s, a) : |S| \times |A| \to \{0, 1\}^{(|S|+|A|)\times|S|}$ as a function of the threshold τ for each transition in a held-out test dataset of $10,000$ trajectories. We compute true and false positive rates for various values of τ to produce ROC plots.

Figure 8 shows that while the SANDy-Mixture is sufficient to solve the simpler synthetic MP settings (with avg AUC of 0.96 and 0.91 for the stationary and non-stationary variants), it scales poorly to the Spriteworld environment. While the modest inductive bias of sparse local connections and high-entropy mixture components in SANDy-Mixture makes it widely applicable, we hypothesize that its sensitivity to hyperparameters makes it difficult to tune in complex settings. Fortunately, SANDy-Transformer, performs favorably in Spriteworld by incorporating a stronger inductive bias about the state subspace structure. Thus we use SANDy-Transformer to perform local factorization inference in the remaining experiments.

Figure 9 provides some qualitative intuition as to how the two variants of SANDy differ in their attention strategy in the Spriteworld environment.

[1]Like the local transition functions, global transition functions are implemented as randomly-initialized single-hidden-layer neural networks.



(a) StationaryMP (b) ϵ-NonstationaryMP (c) `Spriteworld`

Figure 8. ROC plots for correct sparsity patter prediction on the three environments. On held-out test transitions we derive the ground truth local connectivity per step—label information that is *not* used to train the attention model—and measure (over 5 runs; 1 std. dev. shaded) true and false positive rates while sweeping the mask threshold τ over its allowable range. An accurate model generates an Area Under the Curve (AUC) close to 1. We observe that while SANDy-Mixture is sufficient for (nearly) solving the simpler synthetic MP environments, it underperforms in Spriteworld. SANDy-Transfomer, which has a stronger inductive bias, is sufficient to (nearly) solve Spriteworld.

E. Fitting dynamics models to Spriteworld

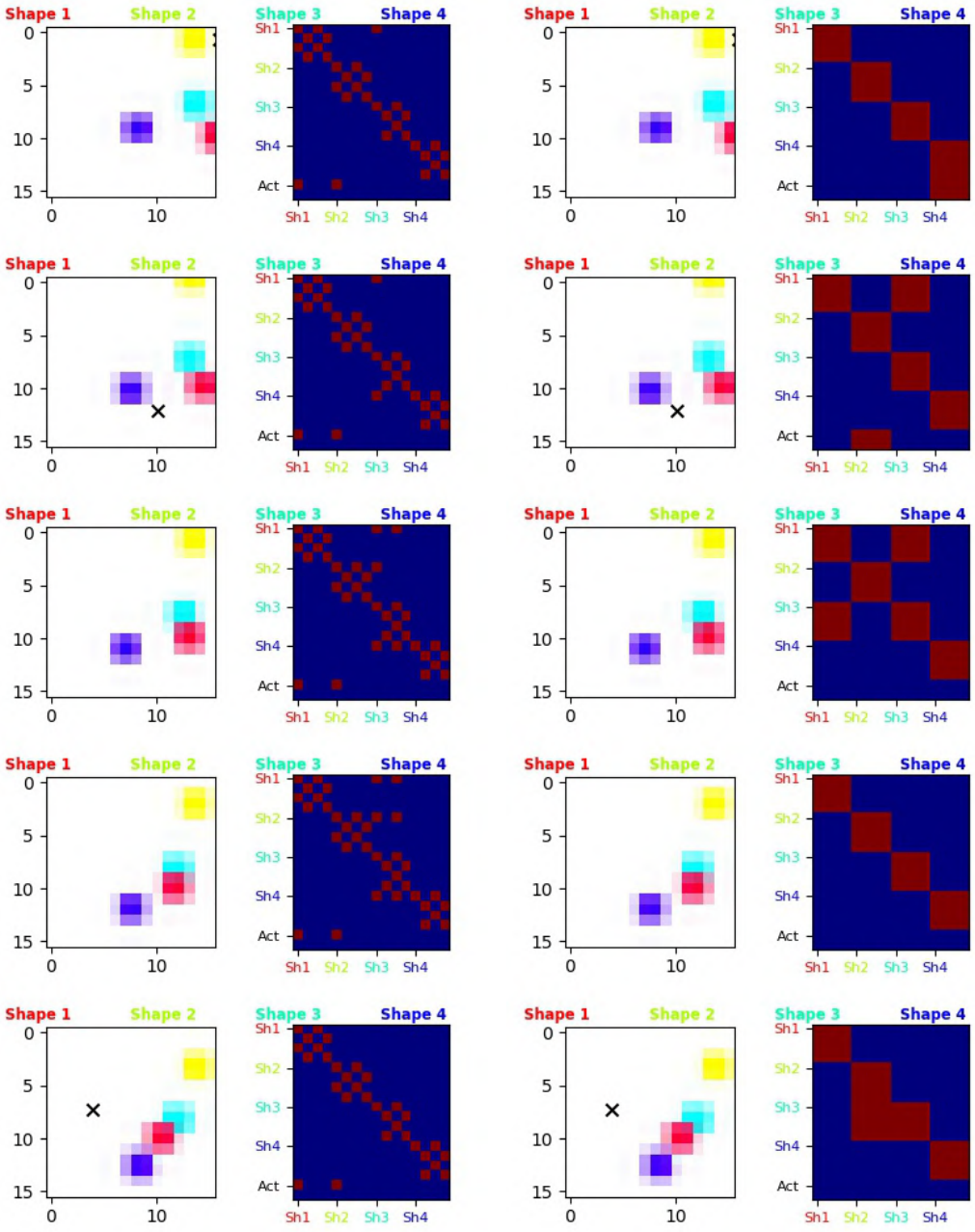
Since CoDA is a data augmentation strategy, it is reasonable to consider an alternative approach to augmenting the experience buffer: sampling from a dynamics model as in model-based RL. Here we present some qualitative results from our efforts in fitting dynamics models to the Spriteworld environment. We found that while dynamics models achieve a decent error in the next-state prediction task, they fail to produce a diverse set of trajectories when used as autoregressive samplers. In particular, the autoregressive sampling did not model collisions well and often produced trajectories where sprites converged to fixed points in space after a short number of steps. Figure 10 shows trajectories sampled autoregressively from Linear, MLP, and LSTM-based dynamics models, alongside the ground truth trajectory. Note that all dynamics models were trained to minimize error in next-state prediction given the current state and action. In other words the LSTM auto-regressively predicts successive dimensions of the next state rather than modeling multiple time steps of the trajectory. Nevertheless the environment is truly Markov because instantaneous velocities are observed, so this information should be sufficient in theory to capture the environment dynamics.

F. Sample efficient dynamics modeling with CoDA

If we had access to the ground truth local factorization, even for a few samples (e.g., we could have humans label them), how much more efficient would it be to train a dynamics model? In Figure 11, we sample 2000 transitions from a random policy in Spriteworld and use the data to train a forward dynamics model using MSE loss. The validation loss throughout training is plotted. Our baseline uses only the initial dataset, and quickly overfits the training set, showing increasing error after the initial few epochs. The same applies to a "random" CoDA strategy, that does CoDA using an identity attention mask ($M(s, a) = I \; \forall \; (s, a)$) to randomly relabel the components. The random strategy does a bit better than the no CoDA strategy, since the randomness acts as a regularizer. Finally, we train a model using an additional 35,000 unique counterfactual CoDA transitions, and find that it significantly improves validation loss and prevents the model from overfitting. Note that we could have generated many more CoDA samples: from 2000 base transitions, if 80% of them do not involve collisions and there are 4 connected components in each, we could generate as many as 1600^4 (6.5 trillion!) counterfactual samples.

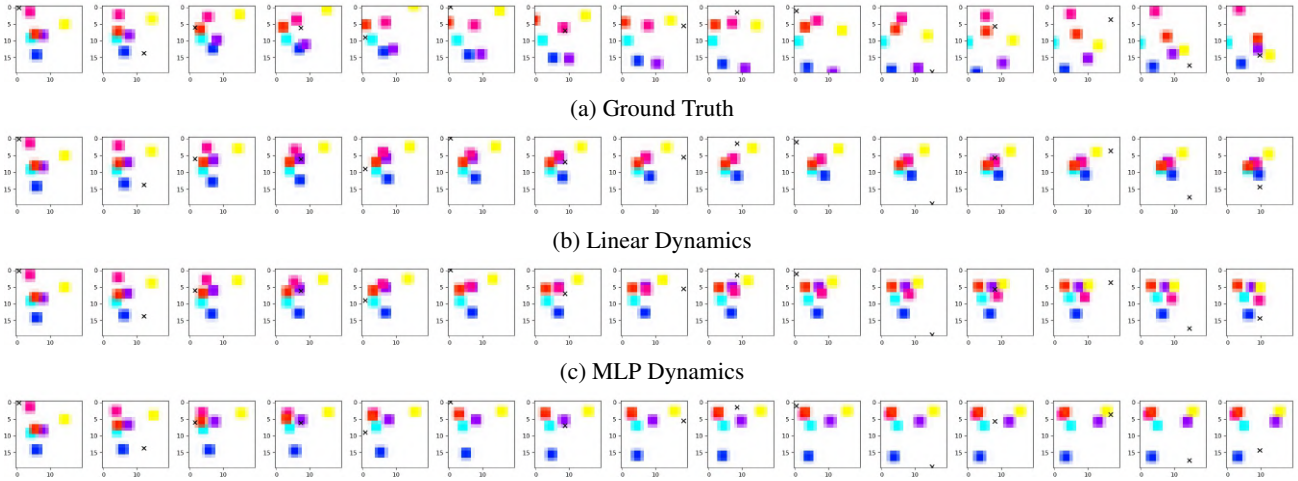
G. Compute Infrastructure

Experiments were run on a mix of local machines and a compute cluster, with a mix of GTX 1080 Ti, Titan XP, and Tesla P100 GPUs. This was solely to run jobs in parallel, and all experiments can be run locally (GPU optional for `Spriteworld` and `Pong`, but recommended for `Fetch` experiments).



(a) SANDy-Mixture (b) SANDy-Transformer

Figure 9. Qualitative comparison of two attention mechanisms on the same `Spriteworld` trajectory. **SANDy-Mixture** (left) has a weaker inductive bias as it relies on sparsity regularization alone. Accordingly, it can learn a more compact subspace (e.g. grid patterns within a shape indicate that x and y coordinates move independently), but is less reliable in attending to collisions between shapes, and completely fails to attend to the action's affect on shapes. **SANDy-Transformer** has a stronger inductive bias and can more reliably infer the local interaction pattern between the five subspaces (four shapes and one action).



(a) Ground Truth

(b) Linear Dynamics

(c) MLP Dynamics

(d) LSTM Dynamics

Figure 10. Auto-regressive model-based rollouts for a variety of dynamics models fit to Spriteworld. While the dynamics models were able to achieve relatively low error in the next-state prediction task, they fail to capture collisions and long-term dependencies in the sampled trajectories, and thus were omitted as baselines in the RL experiments. All trajectories share the same initial state.

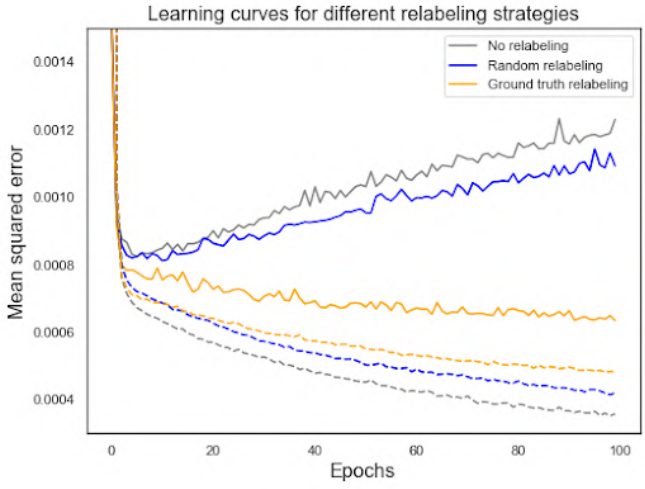


Figure 11. Learning curves for training a forward dynamics model using data from a random policy. Dotted lines indicate training performance, whereas solid lines indicate validation performance. We see that using the ground truth mask prevents overfitting and allows us to achieve much better validation performance.