Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

Unsupervised Visuomotor Control through Distributional Planning Networks

Tianhe Yu, Gleb Shevchuk, Dorsa Sadigh, Chelsea Finn
Stanford University
Email: {tianheyu,glebs,dorsa,cbfinn}@stanford.edu

***Abstract*—While reinforcement learning (RL) has the potential to enable robots to autonomously acquire a wide range of skills, in practice, RL usually requires manual, per-task engineering of reward functions, especially in real world settings where aspects of the environment needed to compute progress are not directly accessible. To enable robots to autonomously learn skills, we instead consider the problem of reinforcement learning without access to rewards. We aim to learn an unsupervised embedding space under which the robot can measure progress towards a goal for itself. Our approach explicitly optimizes for a metric space under which action sequences that reach a particular state are optimal when the goal is *the final state reached.* This enables learning effective and control-centric representations that lead to more autonomous reinforcement learning algorithms. Our experiments on three simulated environments and two real-world manipulation problems show that our method can learn effective goal metrics from unlabeled interaction, and use the learned goal metrics for autonomous reinforcement learning.**

I. Introduction

Reinforcement learning (RL) is a promising approach for enabling robots to autonomously learn a breadth of visuomotor skills such as grasping [36, 24], object insertion and placement tasks [31], and non-prehensile manipulation skills [16, 15, 7]. However, reinforcement learning relies heavily on a reward function or metric that indicates progress towards the goal. In the case of vision-based skills, specifying such a metric is particularly difficult for a number of reasons. First, object poses are not readily accessible and pre-trained object detectors struggle without fine-tuning with data collected in the robot's domain [39]. Second, even when fine-tuned object detectors are available, the location of objects may not be a sufficient representation to identify success for some tasks, while a more suitable representation would require task-specific engineering. For example, if our goal is to manipulate a rope into a particular shape, the corresponding reward function would need to detect the shape of the rope. In many ways, such task-specific engineering of rewards defeats the point of autonomous reinforcement learning in the first place, as the ultimate goal of RL is to eliminate such manual and task-specific efforts.

Motivated by this problem, one appealing alternative approach is to provide an example image of a desired goal state [10, 48, 15, 43, 34, 11], and derive a reward function using the goal image. While such goal observations are applicable to a variety of goal-centric tasks and often easy for a user to provide, they do not solve the problem of rewards entirely: naïve distances to the goal image, such as mean squared error in pixel space, do not provide a suitable metric space for reinforcement learning as they are sensitive to small changes in lighting, differences in camera exposure, and distractor objects. In this paper, our goal is to leverage *autonomous, unlabeled interaction data* to learn an underlying informative metric that can enable the robot to achieve a variety of goals with access to only a single image of the task goal. This capability would enable reinforcement learning of such tasks to be significantly more autonomous.

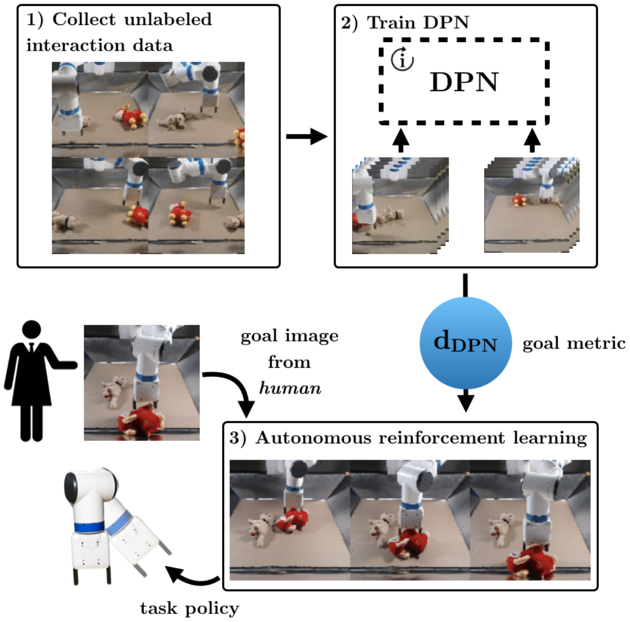

1) Collect unlabeled interaction data
2) Train DPN
DPN
goal image
from
human
d_{DPN}
goal metric
3) Autonomous reinforcement learning
task policy

Fig. 1. General overview of our method. Our method, DPN, enables autonomous reinforcement learning, without human-provided reward functions, on vision-based manipulation problems.

To approach this problem, we aim to learn an embedding space that imposes a metric with respect to a goal image, without using human supervision. One natural option is to use unsupervised representation learning methods [48, 15, 34]. However, these models are largely trained as density estimators, meaning that they will pay attention to the most salient aspects of the images rather than the ones that are relevant for control. Instead, our goal is to learn a *control-centric* representation that takes into account how a sequence of actions leads to a particular observation and ignores other changes in the observation space that are not caused by actions.

Our key insight is that any sequence of actions is optimal under the binary reward function of reaching the final state resulting from those actions. Further, we can use this property to explicitly optimize for control-centric metric spaces from unsupervised interactions. In particular, we propose to explicitly optimize for a metric such that the sequences of actions that lead to a given goal image have high-likelihood when optimizing with respect to the metric. Our approach can be viewed as a generalization of universal planning networks [43] to distributions of actions, while critically showing that such models can be trained from real-world unsupervised interaction rather than simluated expert demonstration data. Our experiments on three simulated domains and two real-world domains demonstrate that our approach can effectively enable robots to learn reaching, object pushing, and rope manipulation tasks from raw pixel observations without human reward feedback and with minimal engineering.

II. Related Work

Our work aims to enable a robot to learn a variety of skills without human supervision, hence falling under the category of self-supervised robotic learning [36, 2, 13, 6, 32]. We specifically approach this problem from the perspective of representation learning, using the learned embedding as a goal metric for reinforcement learning for reaching goal images. Prior works have aimed to learn representations for control through auto-encoding [27, 48, 15, 16, 34], pre-trained supervised features [40], spatial structure [15, 16, 23], and viewpoint invariance [41]. However, unlike these works, we build a metric that specifically takes into account how actions lead to particular states, leading to control-centric representations that capture aspects of the observation that can be controlled, while discarding other elements.

Previous approaches to building control-centric goal representations include using inverse models [2, 35] and mutual information estimation [47]. Unlike our approach, these methods will not necessarily encode all of the aspects of an observation needed to reach a goal. Further, inverse models are susceptible to local optimum when planning greedily. Other methods have built effective reward functions or goal metric spaces using either expert demonstration data [1, 50, 14, 43], pre-trained goal-conditioned policies [17], or other forms of supervision [8, 12]. Our approach, on the other hand, does not use supervision and requires only unlabeled interaction data.

Related to learning *goal* representations, a number of prior works have considered the problem of learning control-centric *state* representations from interaction data [38, 5, 22, 30, 45, 29], for use with planning or reinforcement learning under a known reward or cost. Other works have combined auxiliary representation learning objectives with reinforcement learning [9, 20, 42]. Unlike all of these methods, we focus on representations that induce accurate and informative goal metrics and do not assume access to any reward functions or metrics on state observations.

III. Preliminaries

Our method builds upon universal planning networks (UPN) [43], which learn abstract representations for visuomotor control tasks using expert demonstration trajectories. The representation learned from UPN provides an effective metric that can be used as a reward function to specify new goal state from images in model-free reinforcement learning.

To learn such representations, the UPN is constructed as a model-based planner that performs gradient-based planning in a latent space using a learned forward dynamics model. UPN encodes the initial image of a particular control task into the latent space, and then iteratively executes plans to reach the latent representation of the goal image. The plans are selected via gradient descent on the latent distance between the predicted terminal state and the encoding of the actual target image. Simultaneously, an outer imitation objective ensures that the learned plans match the sequences of actions of the training demonstrations. Consequentially, UPNs learn a latent distance metric by directly optimizing a plannable representation with which gradient-based planning leads to the desired actions.

Concretely, given initial and goal observations $\mathbf{o}_t$ and $\mathbf{o}_g$, e.g., as seen in the two images in Fig. 2, the model uses an encoder f to encode the images into latent embeddings:

$$\mathbf{x}_t = f(\mathbf{o}_t; \theta_{\text{enc}}) \qquad \mathbf{x}_g = f(\mathbf{o}_g; \theta_{\text{enc}}),$$

where $f(\cdot, \theta_{\text{enc}})$ is a convolutional neural network. After encoding, the features $\mathbf{x}_t$ and $\mathbf{x}_g$ are fed into a gradient descent planner (GDP), which outputs a predicted plan $\hat{\mathbf{a}}_{t:t+T}$ to reach $\mathbf{x}_g$ from $\mathbf{x}_t$. The GDP is composed of a forward dynamics model g with parameters θ_{dyn} where $\hat{\mathbf{x}}_{t+1} = g(\mathbf{x}_t, \hat{\mathbf{a}}_t; \theta_{\text{dyn}})$. The learned plan is initialized randomly from a uniform distribution $\hat{\mathbf{a}}^{(0)}_{t:t+T} \sim \mathcal{U}(-1, 1)$ and is updated iteratively via gradient descent as follows:

$$\hat{\mathbf{a}}^{(i+1)}_{t:t+T} = \hat{\mathbf{a}}^{(i)}_{t:t+T} - \alpha \nabla_{\hat{\mathbf{a}}^{(i)}_{t:t+T}} \mathcal{L}^{i}_{plan},$$

where α is the gradient descent step size and $\mathcal{L}^{i}_{plan} = \|\hat{\mathbf{x}}^{(i)}_{t+T+1} - \mathbf{x}_g\|^2_2$. In practice, we find the Huber loss is more effective than the ℓ_2 loss for $\mathcal{L}_{plan}$. After computing the predicted plan, UPN updates the planner by imitating the expert actions $\mathbf{a}^*_{t:t+T}$ in the outer loop. The imitation objective is computed as $\mathcal{L}_{imitation} = \|\hat{\mathbf{a}}_{t:t+T} - \mathbf{a}^*_{t:t+T}\|^2_2$ and is used to update the parameters of the encoder and forward dynamics $\theta := \{\theta_{\text{enc}}, \theta_{\text{dyn}}\}$ respectively:

$$\theta \leftarrow \theta - \beta \nabla_\theta \mathcal{L}_{imitation}$$

where β is the step size for the outer gradient update.

Srinivas et al. [43] applied the learned latent metric as a reward function for model-free reinforcement learning to a range of visuomotor control tasks in simulation and showed that the robot can quickly solve new tasks with image-based goals using the latent metric. However, in order to learn effective representations for new tasks, UPNs require access to optimal expert demonstrations, which are difficult and time-consuming to collect, making it difficult to extend to a variety of tasks, in particular, real-world tasks. In response, we will show a key extension to UPNs that can effectively learn such latent representations without using expert demonstrations in the next section.

IV. Unsupervised Distributional Planning Networks

Our end goal is to enable a robot to use reinforcement learning to reach provided goal images, without requiring manually-provided or hand-engineered rewards. To do so, we will derive an approach for learning a metric space on image observations using only unsupervised interaction data. Universal planning networks (UPNs) [43] show how we can learn such a metric from demonstration data. Further, Ghosh et al. [17] observe that one can learn such a metric with access to a goal-conditioned policy by optimizing for a goal metric that reflects the number of actions needed to reach a particular goal. However, if our end-goal is to learn a policy, we are faced with a "chicken-and-egg" problem: *does the goal-conditioned policy come first or the goal metric?* To solve this problem, we propose to learn both at the same time. Our key observation is that a sequence of any actions, even random actions, is optimal under the binary reward function of reaching the final state resulting from those actions. Specifically, we can use random interaction data to optimize for a metric such that the following is true: when we find a sequence of actions that minimizes the distance metric between the final predicted embedding and the embedded goal image, the true sequence of actions has high probability. Concretely, consider interaction data consisting of an initial image $\mathbf{o}_1$, a sequence of actions $\mathbf{a}_{1:t-1}$ executed by the robot, and the resulting image observation $\mathbf{o}_t$. We can use data like this to optimize for a latent space $\mathbf{x} = f(\mathbf{o}; \theta_{\text{enc}})$ such that when we plan to reach $\mathbf{x}_t = f(\mathbf{o}_t; \theta_{\text{enc}})$, we have high probability of recovering the actions taken to get there, $\mathbf{a}_{1:t-1}$.

So far, this computation is precisely the same as the original UPN optimization, except that we perform the optimization over randomly sampled interaction data, rather than demonstrations. In particular, we surpass the need for expert demonstrations because of the observation that random interaction data can also be viewed as "expert" behavior with respect to the cost function of reaching a particular goal observation at the last timestep of a trajectory (whereas the original UPN was optimizing with respect to the cost function of reaching a goal state with a sequence of optimally short actions). Once we have a representation that can effectively measure how close an observation is to a goal observation, we can use it as an objective that allows us to optimize for reaching a goal observation quickly and efficiently, even though the data that was used to train the network did not reach goals quickly. While not all robotic tasks can be represented as reaching a particular goal observation, goal reaching is general to a wide range of robotic control tasks, including object arrangement such as setting a table, deformable object manipulation such as folding a towel, and goal-driven navigation, such as navigating to the kitchen.

However, note that, unlike in the case of expert demonstration data, we are no longer optimizing for a unique solution for the sequence of actions: there are multiple sequences of actions that lead to the same goal. Hence, we need to model all of these possibilities. We do so by modeling the distribution of action sequences that could achieve the goal, in turn training the UPN as a stochastic neural network to sample different action sequences. Interestingly, universal planning networks are already stochastic neural networks, since the initial action sequence is randomly sampled before each planning optimization. However, as described in Section III, they are trained with a mean-squared error objective, which encourages the model to represent uncertainty by averaging over possible outcomes. To more effectively model the multiple possible sequences of actions that can lead to a potential goal observation, we extend universal planning networks by enabling them to sample from the distribution of potential action sequences. To do so, we introduce latent variables into the UPN model and build upon advances in amortized variational inference [26, 21] to train the model, which we will discuss next.

A. Distributional Planning Network Model

To extend universal planning networks towards modeling distributions over actions, we introduce latent variables into the model. We thus consider the following distribution,

$$\begin{aligned} &p(\mathbf{a}_{t:t+T}|\mathbf{o}_t, \mathbf{o}_{t+T+1}) \\ &\quad = \int p(\mathbf{a}_{t:t+T}|\mathbf{z}_{t:t+T}, \mathbf{o}_t, \mathbf{o}_{t+T+1}) p(\mathbf{z}_{t:t+T}) d\mathbf{z}_{t:t+T}, \end{aligned}$$

by introducing latent variables $\mathbf{z}_t$ for each timestep t. We model the prior over each timestep independently, and model each marginal as a standard Gaussian:

$$p(\mathbf{z}_{t:t+T}) = \prod_{t'=t}^{t+T} p(\mathbf{z}_{t'}) \qquad\qquad p(\mathbf{z}_t) = \mathcal{N}(\mathbf{0}, I)$$

We model $p(\mathbf{a}_{t:t+T}|\mathbf{z}_{t:t+T}, \mathbf{o}_t, \mathbf{o}_{t+T+1})$ using a neural network with parameters θ with two components. The first component is a deterministic gradient descent action planner with respect to latent action representations $\mathbf{z}'_{t:t+T}$, with gradient descent initialized at $\mathbf{z}_{t:t+T}$. The second component is a feedforward decoder that maps from an individual latent action representation $\mathbf{z}'_t$ to a probability distribution over the corresponding action $\mathbf{a}_t$. We will next describe these two components in more detail before discussing how to train this model.

Concretely, the gradient-based planner component consists of:

(a) an encoder $f(\cdot; \theta_{\text{enc}})$, which encodes the current and goal observation $\mathbf{o}_t, \mathbf{o}_g$ into the latent state space $\mathbf{x}_t, \mathbf{x}_g$,

(b) a latent dynamics model $\hat{\mathbf{x}}_{t+1} = g(\mathbf{x}_t, \mathbf{z}'_t; \theta_{\text{dyn}})$ that now operates on latent actions $\mathbf{z}'_t$ rather than actions $\mathbf{a}_t$, and

(c) a gradient descent operator on $\mathbf{z}'_{t:t+T}$ that is initialized at a sample from the prior $\mathbf{z}'^{(0)}_{t:t+T} = \mathbf{z}_{t:t+T}$, and runs n_p steps of gradient descent to produce $\mathbf{z}'^{(n_p)}_{t:t+T}$, using learned step size α_i for step $i = 1, ..., n_p$.

Like before, the gradient descent operator computes gradients with respect to the planning loss $\mathcal{L}_{plan}$, which corresponds to the Huber loss between the predicted $\hat{\mathbf{x}}_{t+T+1}$ and the encoded goal observation, $\mathbf{x}_g$.

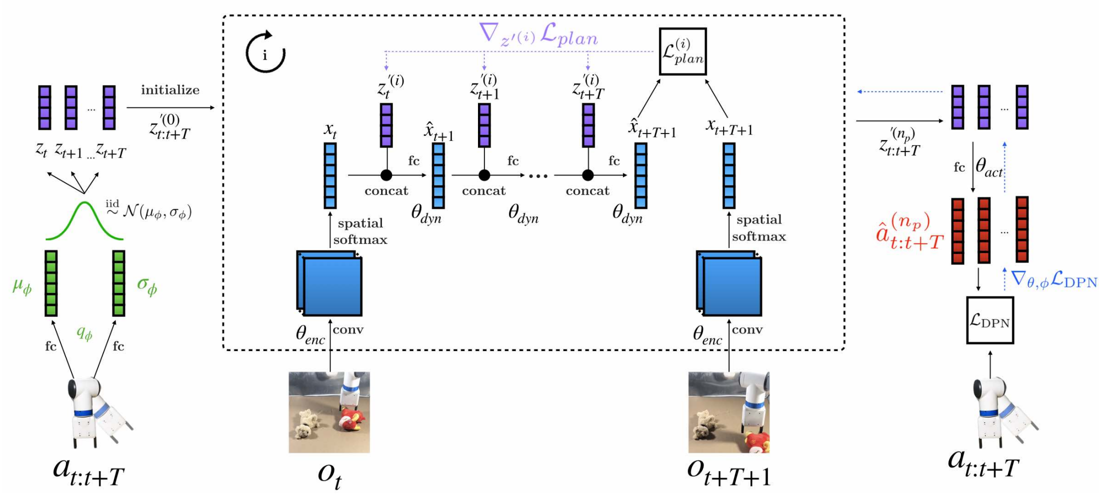


Fig. 2. Diagram of our distributional planning networks model. Our model enables learning a representation $\mathbf{x}$ that induces a control-centric goal metric on images $\mathbf{o}$ from unlabeled interaction data. It does so by explicitly training for a metric under which gradient-based planning leads to the a sequence of actions that reach the final image. To effectively model the many action sequences that might lead to a goal after T timesteps, we introduce latent variables $\mathbf{z}_{t:t+T}$ and train the model using amortized variational inference.

Once we have computed a sequence of latent actions $\mathbf{z}'^{(n_p)}_{t:t+T}$ using the planner, we need to decode the latent values into actions. We do so using a learned action decoder $h(\mathbf{a}_t|\mathbf{z}'_t;\theta_{\text{act}})$. This feedforward neural network outputs the mean of a Gaussian distribution over the action with a fixed constant variance. Overall, the parameters of our model are $\theta = \{\theta_{\text{enc}}, \theta_{\text{dyn}}, \theta_{\text{act}}, \alpha_i\}$. The architectures for each of these components are described in detail in Appendix A. We next describe how to train this model.

B. Distributional Planning Network Training

Since we are training on random interaction data, there are many different sequences of actions that may bring the robot from one observation to another. To effectively learn from this data, we need to be able to model the distribution over such action sequences. To do so, we train the above model using tools from amortized variational inference. We use an inference network to model the the variational distribution, which is factorized as

$$q_\phi(\mathbf{z}_{t:t+T}|\mathbf{a}_{t:t+T}) = \prod_{t'=t}^{t+T} q(\mathbf{z}_{t'}|\mathbf{a}_{t'};\phi),$$

where $q(\mathbf{z}_t|\mathbf{a}_t;\phi)$ outputs the parameters of a conditional Gaussian distribution $\mathcal{N}(\mu_\phi(\mathbf{a}_t), \sigma_\phi(\mathbf{a}_t))$. Following Kingma and Welling [26], we use this estimated posterior to optimize the variational lower bound on the likelihood of the data:

$$\begin{aligned}\mathcal{L}_{\text{DPN}}(\theta,\phi) = &-\mathbb{E}_{\mathbf{z}_{t:t+T}\sim q_\phi}\left[\log p_\theta(\mathbf{a}_{t:t+T}|\mathbf{o}_t, \mathbf{o}_{t+T+1})\right] \\ &+ \beta D_{\text{KL}}(q_\phi(\mathbf{z}_{t:t+T}|\mathbf{a}_{t:t+T}) \mid\mid p(\mathbf{z}_{t:t+T})). \quad (1)\end{aligned}$$

A value of $\beta = 1$ corresponds to the correct lower bound. As found in a number of prior works (e.g. [19, 4]), we find that using a smaller value of β leads to better performance. We compute this objective using random interaction data that is autonomously collected by the robot, and optimize it with respect to the model parameters θ and the inference network parameters ϕ using stochastic gradient descent. Minibatches are sampled by sampling a trajectory from the dataset, $\mathbf{o}_1, \mathbf{a}_1, ...,$ and selecting a length-T segment at random within that trajectory: $\mathbf{o}_t, \mathbf{a}_t, ..., \mathbf{a}_{t+T}, \mathbf{o}_{t+T+1}$. We compute the objective using these sampled trajectory segments by first passing the executed action sequence into the inference network to produce a distribution over $\mathbf{z}_{t:t+T}$. The second term in the objective operates on this Gaussian distribution directly, while the first term is computed using samples from this distribution. In particular, we compute the first term by passing observations $\mathbf{o}_t, \mathbf{o}_{t+T+1}$ and the samples $\mathbf{z}_{t:t+T}$ as input to the gradient descent planner, running gradient descent for n_p timesteps, and decoding the result into $\hat{\mathbf{a}}_{t:t+T}$ to produce the distribution $p_\theta(\mathbf{a}_{t:t+T}|\mathbf{o}_t, \mathbf{o}_{t+T+1}) = \mathcal{N}(\hat{\mathbf{a}}_{t:t+T}, I)$. See Figure 2 for a summary of the model and training.

C. RL with the Learned Goal Metric

Training the distributional planning network provides us with several key components. Most importantly, the encoder $f(\cdot;\theta_{\text{enc}})$ of the DPN provides an embedding of images under which distances to goal images accurately reflect whether or not a sequence of actions actually reached the goal. The combination of the image encoder f, latent dynamics g, and action decoder h serves as a policy that can optimize over a sequence of actions that will reach an inputted goal image.

One easy way to use the DPN model is directly as a goal-conditioned policy. In particular, consider a human-provided goal image $\mathbf{o}_g$ for a desired task. We can compute a sequence of actions to reach this goal image $\hat{\mathbf{a}}_{t:t+T}$ by running a forward pass through the DPN with $\mathbf{o}_t$ and $\mathbf{o}_g$ as input and gradient descent initialized at a sample from the prior $\mathbf{z}'^{(0)}_{t:t+T} \sim p(\mathbf{z}_{t:t+T})$. However, note that the model outputs a distribution over *all* action sequences that might reach the final state after T actions. This means that we can expect the action sequence produced by DPN to reach the goal, but may not do so in a timely manner. In turn, the DPN encoder represents a true metric of whether or not an embedded image $\mathbf{x}$ has reached the same state as another embedded image $\mathbf{x}'$, as it is minimizing for an action sequence that reaches the correct image. As a result, we can alternatively use this metric space with reinforcement learning to optimize for efficiently reaching a goal image.

Thus, after training the DPN model on autonomous, unlabeled interaction, we discard most of the DPN model, only keeping the encoder $f(\cdot;\theta_{\text{enc}})$: this encoder provides a goal metric on images. To enable a robot to autonomously learn new tasks specified by a human, we assume a human can provide an image of the goal $\mathbf{o}_g$, from the perspective of the robot. We then run reinforcement learning, without hand-tuned reward functions, by deriving a reward function from this goal metric. We derive rewards according to the following equation:

$$r(\mathbf{o}_t;\mathbf{o}_g) = -\exp(\mathcal{L}_\delta(\mathbf{o}_t,\mathbf{o}_g))$$

where $\mathcal{L}_\delta$ corresponds to the Huber loss:

$$\mathcal{L}_\delta(\mathbf{o}_t,\mathbf{o}_g) = \|d_{\text{DPN}}(f(\mathbf{o}_t;\theta_{\text{enc}}) - f(\mathbf{o}_g;\theta_{\text{enc}}),\delta)\|_1$$

where for the i-th entry $\mathbf{x}_i$ of some vector $\mathbf{x}$,

$$d_{\text{DPN}}(\mathbf{x},\delta)_i = \begin{cases} \frac{1}{2}\mathbf{x}_i^2 & \text{for } |\mathbf{x}_i| \le \delta, \\ \delta\,|\mathbf{x}_i| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Following Srinivas et al. [43], we use $\delta = 0.85$. We then use the soft actor-critic (SAC) algorithm [18] for running reinforcement learning with respect to this reward function.

V. Experiments

The goal of our experiments is to test our primary hypothesis: *can DPN learn an accurate and informative goal metric using only unlabeled experience?* We design several simulated and real world experiments in order to test this hypothesis with both synthetic and real images in multiple domains, ranging from simple visual reaching tasks to more complex object arrangement problems. In all cases, the objective is to reach the goal state which is illustrated to the agent by a goal image. We will release our code upon publication, and you can find videos of all results at the following link[1].

To quantify the performance of DPN, we compare our method to leading prior approaches for learning goal metrics from unlabeled interaction data. In particular, we compare to the following approaches:

- We train a multi-step **inverse model** to predict the intermediate actions $\mathbf{a}_{t:t+T}$ given two observations $\mathbf{o}_t, \mathbf{o}_{t+T+1}$. Following Agrawal et al. [2] and Pathak et al. [35], we use a siamese neural network that first embeds the two observations and then predicts the actions from the concatenated embeddings. We include a forward-consistency loss as a regularizer of the inverse model suggested in Pathak et al. [35]. We use the embedding space as a goal metric space.
- We train a variational autoencoder (**VAE**) [26], and use distances in the latent space as a goal metric, as done by prior work [34].

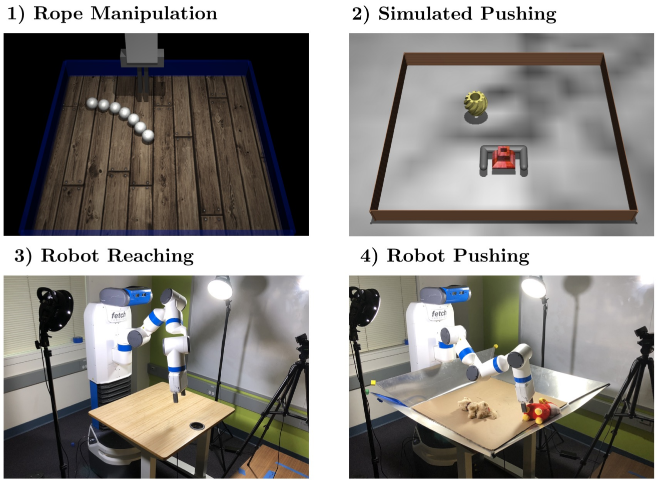


Fig. 3. We conduct experiments on several different vision-based manipluation domains, including simluated rope manipulation, simulated pushing, robot reaching, and robot pushing in the real world.

- We lastly evaluate ℓ_2 distance in **pixel space** as the goal metric.

All of the above approaches are trained on the same unlabeled datasets for each problem domain, except for the pixel distance metric, which is not learned. Because inverse models have a tendency to only pay attention to the most prominent features of an image to identify the actions, we expect the inverse models to work best in situations where the robot's embodiment is a critical aspect of the goal, such as reaching tasks, rather than tasks involving objects. However, even in situations such as reaching, the metric underlying the learned embedding may not correspond to true distances between states in a meaningful way. On the other hand, because VAEs learn an embedding space by reconstructing the input, we expect VAEs to work best in situations where the goal involves particularly salient parts of the image. Finally, we do not expect pixel error to work well, as matching pixels exactly is susceptible to local minima and sensitive to lighting variation and sharp textures.

Finally, we use the soft actor-critic (SAC) [18] with default hyperparameters as the RL algorithm for training all experiments with all four metrics respectively.

We provide full hyperparameters, architecture information, and experimental setup details in Appendix A in the supplemental material.

A. Simulation Experiments

We evaluate our approach starting from simulated experiments using the MuJoCo physics engine [46]. For all simulated experiments, the inputs $\mathbf{o}_t$ and $\mathbf{o}_g$ are 100×100 RGB images.

Simulated Reaching. The first experimental domain is a planar reaching task, where the goal of the task is for a 2-link arm to reach a colored block. We collect 30000 videos of unlabeled physical interactions and train DPN, the inverse model, and the VAE on the random dataset. Note that since this is a fully observed setup with no object interaction, we do not

[1] The supplementary website is at https://sites.google.com/view/dpn-public

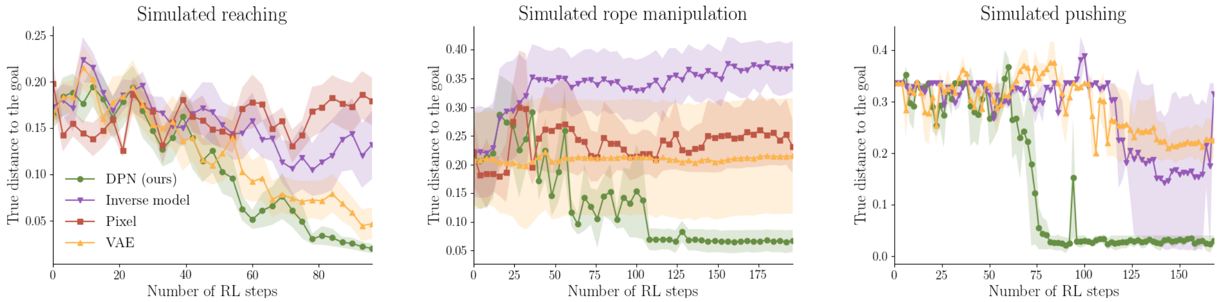


Fig. 4. Quantitative simulation results that evaluate the effectiveness of the goal metrics induced by each method by measuring the true distance to the goal state when running reinforcement learning with the reward derived from the learned goal metric. Performance is averaged across multiple tasks and error bars indicate standard error. Each RL step requires 20 samples from the environment.

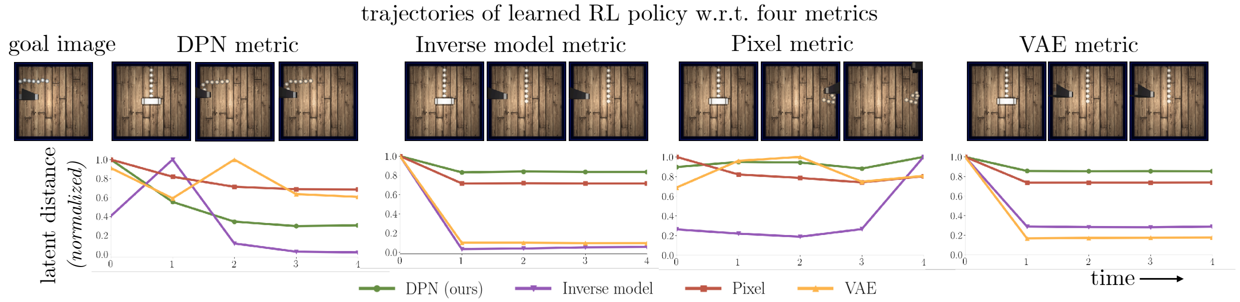


Fig. 5. Comparisons of normalized latent distance to the goal determined by four approaches for the simulated rope manipulation task. We evaluate each latent metric on the trajectories (from a top-down view) of RL policy with respect to DPN, inverse model, VAE, and pixel space, shown above from left to right. Note in the leftmost plot that, though the metric learned by the inverse model achieves a lower normalized latent distance than the DPN metric, it goes to around 0 once the gripper moves closer to its corresponding position in the goal image without touching the rope as shown in the second and fourth plot from the left. This suggests that the inverse model metric fails to capture the actual goal of task, which is directing the rope to the right form.

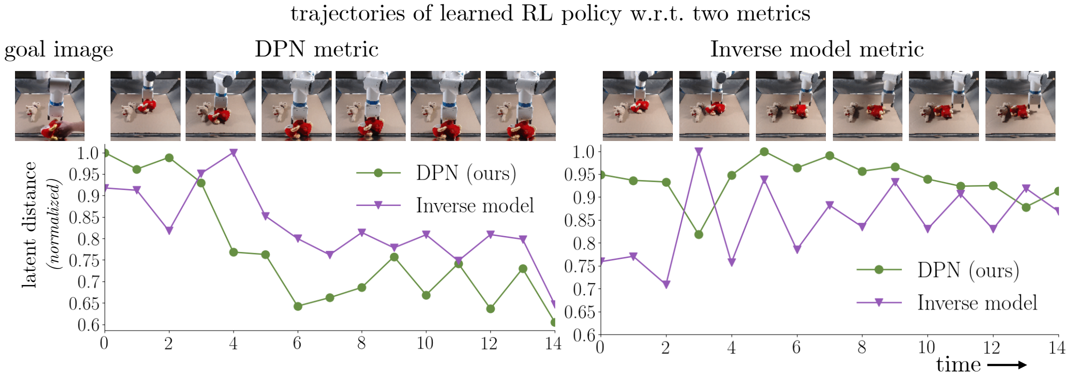


Fig. 6. Comparisons of normalized latent distance to the goal determined by DPN and inverse model for the real-world pushing task. We evaluate each latent metric on the trajectories of RL policy with respect to DPN and inverse model respectively, shown in the images above from left to right.

use the multi-step inverse model with recurrence and instead use a one-step inverse model with feed-forward networks, as suggested by Pathak et al. [35].

We evaluate all learned metrics along with the pixel space metric by running SAC on 10 different tasks where the target block is at a different position for each task. The input to the RL policy is the joint angles of the robot. We summarize the comparison in Figure 4. As shown in the plot, our method is able to reach the goal within 0.05cm in 60 RL steps and gets closer to 0 after 100 steps (see Figure 7). The RL policies with metrics learned by VAE and the inverse model are also able to get to the proximity of the goal but are less accurate. This is reasonable since VAEs usually pay attention to the most salient part of the image while inverse models usually pay attention to objects that correlate most with the actions, i.e. the robot arm in this domain. Tracking the arm movement is sufficient to solving reaching tasks. The pixel space distance, meanwhile, struggles to find the goal as expected since pixel-wise distance is susceptible to minor environmental changes.

Simulated Rope Manipulation. The goal of the second experiment is to manipulate a rope of 7 pearls into various shapes using a parallel-jaw gripper, where the setup is shown in Figure 3. In this experiment, we aim to test if our method can focus on the shape of the rope rather than the position of the gripper since, unlike simulated reaching, only encoding the movement of the gripper into the latent space would lead

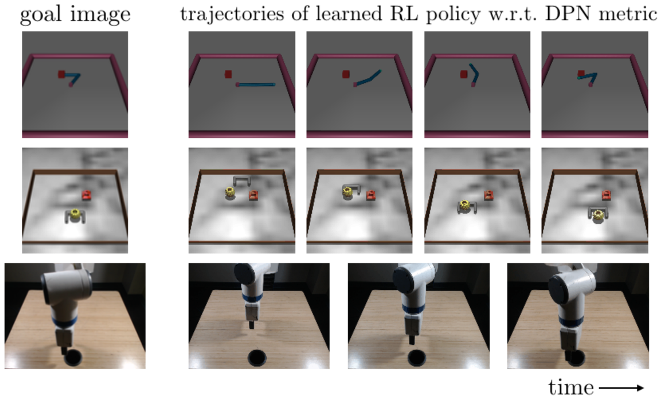


Fig. 7. Roll-outs of learned RL policy using the DPN metric of simulated reaching, simulated pushing, and robot reaching experiments from top to bottom.

to ignoring the actual goal of the task: manipulating the rope.

We collect 20000 10-frame random videos. Similar to simulated reaching, we then train a one-step inverse model for rope manipulation.

We evaluate the four metrics by running SAC on 4 tasks where the rope is displaced to a different shape in each task. The input to the RL policy are the end-effector positions and velocities of the gripper. For evaluation, we define the true distance to goal following Xie et al. [49], measuring the average distance of corresponding pearls in the rope. As seen from the results in Figure 4, our method does substantially better than the other three approaches, achieving around 0.05cm on average to the shapes shown in the goal images. The other three approaches fail to lead to effective RL training. To conduct a more direct comparison of all the latent metrics, we plot the latent distance to the goal of four approaches when rolling out the trajectories of learned RL policy with all the four metrics as reward functions respectively in Figure 5. Notice that the metrics learned by the inverse model and the VAE go to around 0 once the gripper goes to its corresponding position in the goal image but completely disregards the rope. In contrast, the DPN metric only decreases when the rope is manipulated to the target shape. This experiment demonstrates that DPN is able to produce a more informative metric without collapsing to the most salient feature in the image.

Simulated Pushing. For our third simulated experiment, we perform a simulated pushing task where a robot arm must push a target object to a particular goal position amid one distractor. In order to make this environment more realistic, we use meshes of a vase and a strainer from thingiverse.com with different textures for the two objects and a marble texture for the table (see Figure 3).

For this task, we collect 3000 random 16-frame videos and train a multi-step recurrent inverse model for comparison.

Based on the previous two experiments, the pixel distance does not serve as a good metric, so we drop it and only evaluate DPN, inverse models and VAEs. For this task, knowing only the hand's position is not sufficient to solve the task. Hence, in addition to end-effector positions and velocities of the robot hand, a latent representation of the current image extracted from the inverse model is also provided as an input the RL policy for all methods (see Appendix D for more details). As seen in Figure 4, DPN learns to push both objects toward the goal position with a distance close to 0cm (see Figure 7 for an example). With the multi-step inverse model, the robot only pushs one of the objects to goal, as indicated by the large standard error in Figure 4. Under the VAE metric, the robot cannot quite learn how to push both objects and RL training does not make significant progress.

B. Real World Robot Experiments

In order to ascertain how well our approach works in the real world with real images, we evaluate on two robot domains. Similar to the simulated tasks, a robot, a Fetch Manipulator, needs to reach a goal image using images from a forward-facing camera by applying continuous end effector velocities. Both setups are shown in Figure 3.

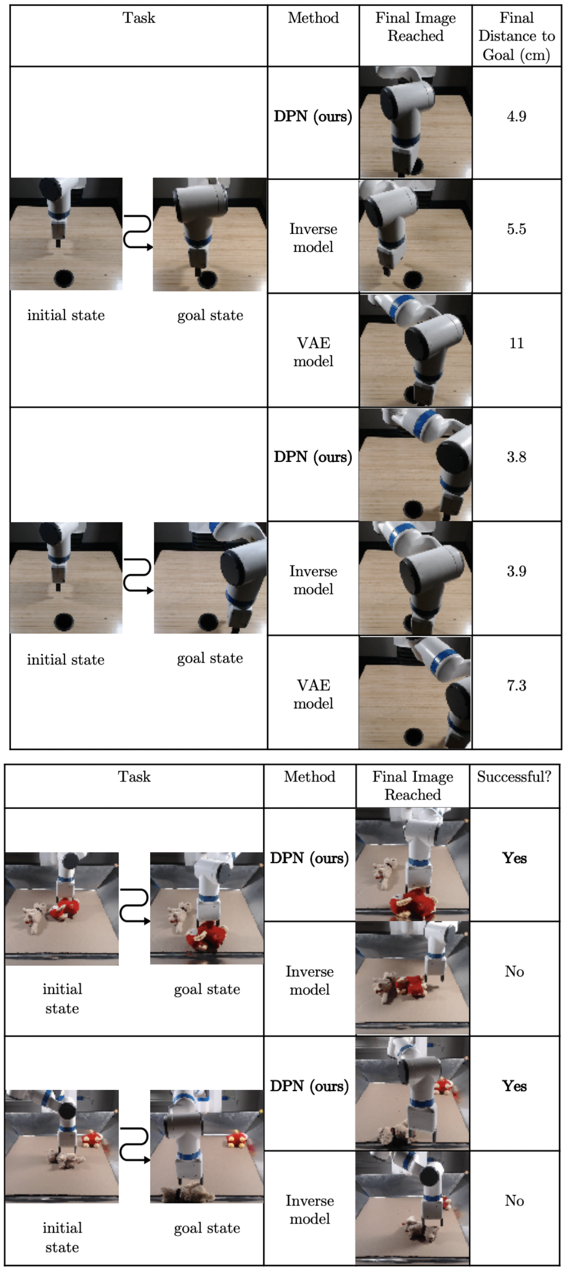
Robot Reaching. First, we evaluate our method for a simple environment, where the robot must learn to move its end-effector along the xy plane to a certain position above a table. We collect unlabeled interaction data by having the arm randomly move above the table. For this task, we capture 5000 episodes, which corresponds to approximately 28 hours of capture.

We train DPN on 4000 samples and use the remaining 1000 samples for validation. Seeing that the inverse and VAE models are the next best-performing in simulation experiments, we also train both on this dataset using the same procedures described earlier. We also use the one-step inverse model for this experiment with the same reason discussed in the simulated reaching section.

For the first task, the goal image consists of the arm hovering above a hole at the front-middle of the table. For the second task, the goal image consists of the arm hovering at the top left corner of the table. For both tasks, we run reinforcement learning for approximately 300 episodes, corresponding to around 3 hours, for each of the DPN, the inverse model, and VAE metrics. To evaluate performance, we roll out the learned policy 3 times at the checkpoint that achieved the highest training reward, according to the learned metric, and compute the average true distances to goal position at each final timestep (see Figure 7). As shown in Figure 8, we find that the policies that use our DPN metric and the inverse metric performs fairly similarly, while the agent that used the VAE metric performs considerably worse.

Across these two tasks, the VAE model may have performed worse due to slight changes in lighting and camera angle, as previously discussed. Furthermore, as discussed in the simulated reaching example, both DPN and the inverse model likely performed similarly because they both focused on the parts of the videos that correlate most heavily with the actions, i.e. the location of the arm.

Robot Pushing. Seeing that DPN and the inverse model performed similarly on the reaching task, we construct a



Task	Method	Final Image Reached	Final Distance to Goal (cm)
initial state goal state	**DPN (ours)**		4.9
	Inverse model		5.5
	VAE model		11
initial state goal state	**DPN (ours)**		3.8
	Inverse model		3.9
	VAE model		7.3

Task	Method	Final Image Reached	Successful?
initial state goal state	**DPN (ours)**		**Yes**
	Inverse model		No
initial state goal state	**DPN (ours)**		**Yes**
	Inverse model		No

Fig. 8. Results for the real world reaching and pushing tasks. Our approach is able to learn a metric on real images that enables successful autonomous RL for both reaching and object pushing, whereas prior methods do not consistently lead to successful reinforcement learning.

harder object pushing task that requires the two metrics to pay attention to smaller features in the environment and to take longer-horizon interactions into account.

For this task, the robot has to maneuver a given object from an initial position to a goal position on the table. This task is complicated by a distractor object next to the actual object. We choose two stuffed animals as our given objects and use an aluminum hopper to prevent them from falling off the table during data collection and RL. Data collection occurs in the same manner as in the reaching experiment. In total, we collect 5000 episodes, taking approximately one day on one robot.

Since our simulated pushing results indicated that the VAE metric performed poorly, we only compare DPN with the inverse model for this experiment. Again, we trained an agent with each of the DPN and inverse metrics for approximately 400 episodes and roll the highest-reward policy out 3 times. As shown in Figure 8, the agent trained with DPN is able to successfully push the stuffed animal to the goal position, while the agent trained with the inverse model is at best only able to push it to the middle of the table. The two metric curves shown in Figure 6 give some intuition as to why this is the case. When we plot the metrics for the policy learned w.r.t. the DPN metric, we see that both metrics correctly decrease, although the DPN metric is better at recognizing the similarity of later images, and is thus smaller. Meanwhile, when we plot them for the policy learned w.r.t. the inverse model, we see that the inverse metric incorrectly associates a small latent distance with the earlier images and thus rewards the RL agent for doing nothing, making it difficult for the RL agent to meaningfully move the object. These results seem to match what we saw in the simulated pushing experiment and suggest that DPN does better at distinguishing smaller features that are important for a goal, while the inverse model ignores them and over-prioritizes arm placement.

VI. Discussion

Summary. In this paper, we presented an approach for unsupervised learning of a control-centric metric space on images that allows a robot to evaluate its progress towards a specific goal. Our approach proposes more effective and autonomous reinforcement learning while only having access to the goal image by leveraging the learned metric. We then evaluated our method on simulated and real-world tasks, including reaching, pushing, and rope manipulation. Our results suggest that the DPN metric enables RL to perform well on these robotics tasks while converging faster than state-of-the-art techniques for unsupervised goal representations.

Limitations and Future Work. While we are able to show very good performance for an interesting set of robotics tasks, our method is limited to goal-reaching tasks, which leaves out a number of other interesting RL tasks. Despite this limitation, we believe learning a control-centric metric using our approach may be applicable to a wider range of settings by applying the cost function towards tracking an entire trajectory rather than simply a final state. We are excited to take our approach beyond the current tasks and consider learning control-theoretic metrics in this wider range of settings.

Beyond vanilla RL, which we study in this paper, a number of other methods rely heavily on effective distance metrics in state space, which have so far limited their application to non-vision domains due to the lack of suitable metrics. This includes goal relabeling for multi-goal RL [3, 37], planning with learned models [28, 33], and automatic curriculum generation [44]. In future work, we hope to explore the use of our metric in combination with these methods.

Acknowledgments

We thank Aravind Srinivas and Sergey Levine for helpful discussions. This work has been partially supported by JD.com American Technologies Corporation (”JD”) under the SAIL-JD AI Research Initiative. This article solely reflects the opinions and conclusions of its authors and not JD or any entity associated with JD.com.

References

[1] Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[4] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

[5] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011.

[6] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control. *arXiv preprint arXiv:1710.00489*, 2017.

[7] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *International Conference on Machine Learning*, pages 703–711, 2017.

[8] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014. doi: 10.15607/RSS.2014.X.031.

[9] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

[10] Koichiro Deguchi and Isao Takahashi. Image-based simultaneous control of robot and target object motions by direct-image-interpretation method. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 1, pages 375–380. IEEE, 1999.

[11] Ashley D Edwards. *Perceptual Goal Specifications for Reinforcement Learning*. PhD thesis, Georgia Institute of Technology, 2017.

[12] Ashley D Edwards, Srijan Sood, and Charles L Isbell Jr. Cross-domain perceptual reward functions. *arXiv preprint arXiv:1705.09045*, 2017.

[13] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[14] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*, 2016.

[15] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *International Conference on Robotics and Automation (ICRA)*, 2016.

[16] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2351–2358. IEEE, 2017.

[17] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

[18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[19] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[20] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[21] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

[22] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

[23] Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *arXiv preprint arXiv:1705.09805*, 2017.

[24] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen,

Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[25] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[27] Sascha Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[28] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[29] Timothée Lesort, Mathieu Seurin, Xinrui Li, Natalia Díaz Rodríguez, and David Filliat. Unsupervised state representation learning with robotic priors: a robustness benchmark. *arXiv preprint arXiv:1709.05185*, 2017.

[30] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 2018.

[31] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[32] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[33] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[34] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Steven Bahl, Shikhar ans Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[35] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018.

[36] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[37] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[38] Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proceedings of the twenty-first international conference on Machine learning*, page 88. ACM, 2004.

[39] Amir Rosenfeld, Richard Zemel, and John K Tsotsos. The elephant in the room. *arXiv preprint arXiv:1808.03305*, 2018.

[40] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[41] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *arXiv:1704.06888*, 2017.

[42] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

[43] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *International Conference of Machine Learning (ICML)*, 2018.

[44] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[45] Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable features. *arXiv preprint arXiv:1708.01289*, 2017.

[46] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[47] David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

[48] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

[49] Annie Xie, Avi Singh, Sergey Levine, and Chelsea Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, 2018.

[50] Brian Ziebart, Andrew Maas, Andrew Bagnell, and Anind Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

Appendix

In this appendix, we summarize our DPN approach in Algorithm 1. We also provide additional details about our implementation details and experimental setup.

A. Architecture and Hyperparameters

For all approaches except the ℓ_2 distance in pixel space, we use a 4-layer convolutional neural networks with 64 5×5 filters each layer and spatial soft-argmax [15] after the last convolution layer to represent the encoder $f(\cdot; \theta_{\text{enc}})$, which encodes images into latent space. For DPN, we represent the latent dynamics $g(\mathbf{x}_t, \hat{\mathbf{a}}_t; \theta_{\text{dyn}})$ as 2-layer fully-connected neural networks with 128 hidden units. The inference network $q(\mathbf{z}_t|\mathbf{a}_t; \phi)$ is modeled as a 2-layer fully-connected neural network with hidden units 16 where the last layer has two heads that output the mean $\mu_\phi(\mathbf{a}_t)$ and the standard deviation $\sigma_\phi(\mathbf{a}_t)$. The action decoder $h(\mathbf{a}_t|\mathbf{z}'_t; \theta_{\text{act}})$ is also a 2-layer neural network with 16 hidden units. We use $\beta = 0.5$ as the KL constraint value in $\mathcal{L}_{\text{DPN}}$. We use $n_p = 20$ as the number of gradient descent steps that update $\mathbf{z}'_{t:t+T}$ and the learned size α_i's are initialized with 0.05. For all experiments, we train DPN for approximately 12 hours on an NVIDIA Titan X GPU. For the inverse model, we represent the latent multi-step inverse model as a recurrent neural network with 128 units. For VAE, while the architecture of the encoder is the same across all methods as mentioned above, the decoder consists of 4 deconvolutional layers with 64 5×5 filters except that the last layer has 3 5×5 filters in order to reconstruct the image. All three approaches are trained with Adam optimizer [25] with learning rate 0.0005.

B. Simulated Reaching

We collect 3000 videos of the unlabeled interaction by sampling torques uniformly from $[-5, 5]$, where each video consists of 10 images. Along with each video, we also store a sequence of torques as actions and robot joint angles. For training all methods, we use a training set that contains 28500 videos and a validation set that has 1500 videos.

The length of each RL step is 1000 and the maximum path length is 50.

C. Simulated Rope Manipulation

For collecting 20000 10-frame random videos, we randomly sample positions of gripper at each time step. We use 19000 videos for training and the remaining 1000 for validation.

We set the maximum path length to be 5 and the length of one RL step to be 500.

D. Simulated Pushing

At each time step of the random data collection, we apply a random end-effector velocity sampled uniformly from $[-20, 20]$. In this way, we collect 3000 videos, where 2850 videos are used for training and 150 are used for validation.

Each RL step consists of 2000 timesteps and the maximum path length is 100 timesteps. As mentioned in Section V, the input to the RL policy is the position of the robot hand along with the latent representation of the current image extracted from the inverse model. We find that the state representation learned by the inverse model is the most accurate across all methods, and hence adopt it for training RL policies w.r.t. all metrics. However, DPN learns the most effective goal representations that produce the best RL performance as shown in Fig 4. Such results suggest an interesting point that good state representations might not necessarily be suitable for goal representations. DPN optimizes for the distance metric induced by the goal representation, and thus it might discover the best goal but probably not the best state representations.

Algorithm 1 Distributional Planning Networks

```
Require: random dataset $\{(\mathbf{o}_{t:t+T+1}, \mathbf{a}_{t:t+T})\}$
Require: KL constraint value $\beta$, outer step size $\gamma$
  Initialize $\alpha_{0:n_p-1}$
  Define the prior $p(\mathbf{z}_{t:t+T}) = \mathcal{N}(\mathbf{0}, I)$
  while training do
    Sample a batch of random data $\mathbf{o}_t$, $\mathbf{o}_{t+T+1}$, $\mathbf{a}_{t:t+T}$
    Sample latent actions $\mathbf{z}_{t:t+T} \sim q_\phi(\mathbf{z}_{t:t+T}|\mathbf{a}_{t:t+T})$
    Initialize $\mathbf{z}'^{(0)}_{t:T} = \mathbf{z}_{t:t+T}$
    for $i = 0, 1, ..., n_p - 1$ do
      Encode $\mathbf{x}_t = f(\mathbf{o}_t; \theta_{\text{enc}})$, $\mathbf{x}_{t+T+1} = f(\mathbf{o}_{t+T+1}; \theta_{\text{enc}})$
      Set $\hat{\mathbf{x}}^{(i)}_t = \mathbf{x}_t$
      for $j = 0, 1, \ldots, T$ do
        $\hat{\mathbf{x}}^{(i)}_{t+j+1} = g(\hat{\mathbf{x}}^{(i)}_{t+j}, \mathbf{z}'^{(i)}_{t+j}; \theta_{\text{dyn}})$
      end for
      Compute $\mathcal{L}^{(i)}_{plan} = \mathcal{L}_\sigma(\hat{\mathbf{x}}^{(i)}_{t+T+1}, \mathbf{x}_{t+T+1})$
      Update $\mathbf{z}'^{(i+1)}_{t:t+T} = \mathbf{z}'^{(i)}_{t:t+T} - \alpha_i \nabla_{\mathbf{z}'^{(i)}_{t:t+T}} \mathcal{L}^i_{plan}$
    end for
    Compute $\hat{\mathbf{a}}_{t:t+T} = h(\mathbf{z}'^{(n_p-1)}_{t:t+T}; \theta_{\text{act}})$
    Compute $\mathcal{L}_{\text{DPN}} = \log p_\theta(\mathbf{a}_{t:t+T}|\mathbf{o}_t, \mathbf{o}_{t+T+1}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}_{t:t+T}|\mathbf{a}_{t:t+T}) \;||\; p(\mathbf{z}_{t:t+T}))$
    Compute $\nabla_\theta \mathcal{L}_{\text{DPN}}$ and $\nabla_\phi \mathcal{L}_{\text{DPN}}$
    Update $\theta \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}_{\text{DPN}}$
    Update $\phi \leftarrow \phi - \gamma \nabla_\phi \mathcal{L}_{\text{DPN}}$
  end while
  Return $\theta, \phi$
```

E. Robot Setup

The initial state of the robot is with its arm above the middle of a table. The arm is constrained to move in a square region that is approximately 0.16m^2 in area. In both tasks, we collect 100×100 RGB images from a front facing camera alongside joint information, and use continuous end effector velocities as actions, which are normalized to be between $[-1, 1]$. Side lights are used to enable nighttime data collection.

F. Robot Reaching

During unlabeled interation data collection, each episode consists of 20 timesteps starting from a fixed initial state. At each timestep, we uniformly sample an action to apply to the arm and keep the arm within bounds by applying an inward end effector velocity at table edges.

We use the learned metrics for reinforcement learning of two reaching tasks, where the only reward exposed to the reinforcement learning agent is derived from the latent metric. Because we have access to the true end effector position, we use ℓ_2 distance between the final position and goal position to evaluate final performance. This distance is not provided to the robot.

In order to make the first task more challenging, we limit the policy to normalized actions between -0.1 and 0.1. This makes it harder for the arm to reach the goal if it initially chooses an incorrect direction of motion. This task, in turn, is designed to see how well each metric works at the start of an episode, where the arm was farther from its goal.

Meanwhile, in the second task, we leave the policy's actions unscaled, making it more likely that the arm will violate bounds and be pushed back inward by the correcting inward velocity once it comes close to an edge. Therefore, this task is meant to show how well each metric performs at the later timesteps of a policy, where the arm is closer to its goal.

The length of each RL step is 20 and the maximum path length is 20.

G. Robot Pushing

Each episode for random data collection consists of 15 timesteps. In order to vary the initial positions of the objects, we use a mix of scripted shuffling methods and manual rearrangement when objects got stuck in a corner, leading to data collection that is nearly entirely autonomous

For the reinforcement learning task, the arm had to learn to push the either the red or the tan stuffed animal from the middle of the table to a point at the front of the table depending on the goal image (see Fig 8).

The length of each RL step is 20 and the maximum path length is 20.