Task-Conditioned Variational Autoencoders for Learning Movement Primitives

Michael Noseworthy, Rohan Paul, Subhro Roy, Daehyung Park, and Nicholas Roy
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{mnosew, rohanp, sroy, daehyung, nickroy}@csail.mit.edu

Abstract: Consider a task such as pouring liquid from a cup into a container. Some parameters, such as the location of the pour, are crucial to task success, while others, such as the length of the pour, can exhibit larger variation. In this work, we propose a method that differentiates between specified *task* parameters and learned *manner* parameters. We would like to allow a designer to specify a subset of the parameters while learning the remaining parameters from a set of demonstrations. This is difficult because the learned parameters need to be interpretable and remain independent of the specified *task* parameters. To disentangle the parameter sets, we propose a *Task-Conditioned Variational Autoencoder* (TC-VAE) that conditions on the specified *task* parameters while learning the rest from demonstrations. We use an adversarial loss function to ensure the learned parameters encode no information about the *task* parameters. We evaluate our method on pouring demonstrations on a Baxter robot from the MIME dataset. We show that the TC-VAE can generalize to task instances unseen during training and that changing the learned parameters does not affect the success of the motion.

Keywords: Learning from Demonstration, Movement Primitives, Variational Inference, Representation Learning for Manipulation.

1 Introduction

Consider teaching a robot to perform a class of tasks such as pouring liquid from a cup into a container. There are many ways it can accomplish this task – for example, it could vary the length or the maximum angle of the pour. However, it is crucial that no matter how the task is performed, the pouring location remains the same as the location of the container. Our goal is to learn an interpretable representation for movement primitives that respects task constraints such as these.

In general, the *Movement Primitive* framework aims to recover a set of primitives, each represented by a low-dimensional parameter space, which can be learned given demonstrations of a task. The parameter space allows the movement primitive to be adapted to new instances of the task and can either be manually specified *a priori* [1, 2, 3] or learned from a set of demonstrations [4]. Importantly, to simplify the adaptation to new task instances, the parameters should be interpretable.

Motivated by the *pouring* example (see Figure 1), we differentiate between specified *task* (or context [2, 3]) parameters that determine *what* task to perform (e.g., the location of the container) and unspecified, learned *manner* parameters that describe *how* that task should be performed (e.g., the length of the pour). Note that the partitioning of *task* and *manner* parameters may differ depending on the action. Due to the relative importance of the *task* parameters, we manually specify them while discovering a set of interpretable *manner* parameters from data. The *manner* parameters should explain the types of variation that can occur independent of the task.

There are two main difficulties in discovering parameters for how a task is performed. The first is disentangling the *task* parameters from the learned *manner* parameters. Varying a learned *manner* parameter should not affect aspects of the motion related to the *task* parameters. For example, varying the length of the pour should not change the location. The difficulty arises from the fact that the final action is a complex function of both *task* and *manner* parameters.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

The second difficulty arises from the fact that we are learning the *manner* parameter space from noisy data. The data may exhibit variation we do not explicitly wish to control in the learned parameter space. In order for our learned representation to be interpretable, the parameters should be independent of each other.

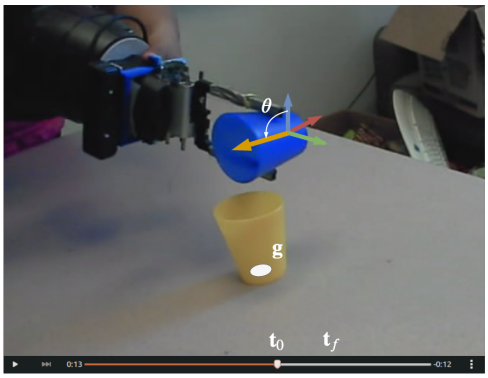

Figure 1: Example pouring demonstration from the MIME dataset annotated with specified *task* parameters (e.g., the pouring location, g) and *manner* parameters we aim to learn (e.g., maximum pouring angle, θ, pour start time, t_0, and pouring duration, $t_f - t_0$).

To address these problems, we propose a *Task-Conditioned Variational Autoencoder* (TC-VAE) to learn a representation for movement primitives given a set of demonstrations. The TC-VAE is conditioned on *task* information such as the location of the pouring container. As such, the latent space of the VAE does not need to encode aspects of the movement related to the *task* variables and can learn parameters that are independent of them. To enforce the independence between the specified *task* parameters and learned *manner* parameters, we adversarially train the latent space to minimize the information encoded about task variables [5, 6]. We show that this adversarial training is important in successfully disentangling the parameter sets.

To evaluate our model, we learn a movement primitive for *pouring* using the MIME dataset [7]. We augment the dataset to include the location of each pour. We show that the learned movement can generalize to task instances unseen during training and that changing the learned *manner* parameters does not affect the success of the motion. Finally, we inspect and visualize the learned latent space to show it captures semantically coherent axes of variation.

2 Task-Conditioned Variational Autoencoders

Our goal is to learn a generative model of action trajectories that explicitly differentiates between specified *task* parameters and learned *manner* parameters. We refer to the learned parameters as *manner* parameters as they will describe the variation that exists when performing specific tasks. Each action class (e.g., pouring) will have its own model. The input to this problem are the *task* parameters, $\mathbf{w}$. The output is a distribution over trajectories of length T that describe all the ways in which this task can be performed: $\mathrm{p}(\mathbf{x}|\mathbf{w})$, where $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$. Each $\mathbf{x}_t \in \mathbb{R}^m$ represents the m-dimensional state of the robot at time t (in our case, the end-effector pose).

We begin by describing the probabilistic model we propose to represent actions in Section 2.1 and the variational inference procedure used to fit it in Section 2.2. To enforce independence between the *task* and *manner* parameters, we augment the training procedure with an *adversarial information objective* [6]. This objective ensures no information about the specified *task* parameters is encoded into the learned latent space (Section 2.3).

2.1 Probabilistic Model

We propose a latent variable model to represent how a task can be performed. Specifically, we introduce a set of latent variables, $\mathbf{h}$, to capture the variation that can occur independent of the task, $\mathbf{w} \in \mathbb{R}^K$ where K is the number of task variables. We place a Normal prior on $\mathbf{h}$, $p(\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. $\mathbf{x}$ can then be modeled as:

$$p(\mathbf{x}|\mathbf{w}) = \int p(\mathbf{x}|\mathbf{w}, \mathbf{h}) p(\mathbf{h}) d\mathbf{h} = \int \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{w}, \mathbf{h}) p(\mathbf{h}) d\mathbf{h}. \tag{1}$$

The trajectory likelihood, $p(\mathbf{x}_t|\cdot) \sim \mathcal{N}(\mu_\theta^t(\cdot), \Sigma_\theta^t(\cdot))$, is a Gaussian distribution parameterized by a neural network that will be described in the following section.

As mentioned in the introduction, it is important that the learned *manner* parameters, $\mathbf{h}$, are interpretable. This will permit easy adaptation in downstream tasks. To achieve this goal, we use the β-VAE objective [8] which we will discuss in more detail in the following section.

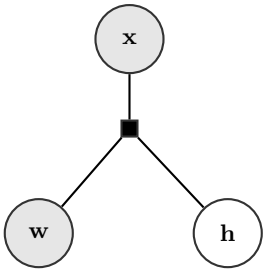

Figure 2: Factor graph of the *Task-Conditioned VAE* model. The *task* parameters, $\mathbf{w}$, are observed and the latent *manner* parameters, $\mathbf{h}$, are learned. Both are combined to produce a trajectory, $\mathbf{x}$.

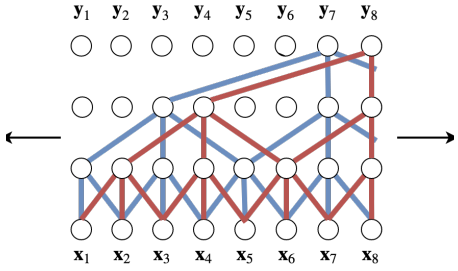

Figure 3: A *Temporal Convolution Network* uses stacked dilated convolutions to map an input sequence, $\mathbf{x}_{1:T}$ to an output sequence $\mathbf{y}_{1:T}$. We use a kernel size of 3.

Importantly, we require the specified *task* parameters, $\mathbf{w}$, and the learned *manner* parameters, $\mathbf{h}$, to be independent of each other. That is, when we vary one parameter, it should not change aspects the trajectory described by the others. To enforce this disentanglement, we adversarially train the latent space to not carry any information about the task-specific variables. We describe this process in more detail in Section 2.3. Overall, our model decomposes as (see Figure 2 for a factor graph):

$$p(\mathbf{x}|\mathbf{w}) = \int p(\mathbf{x}|\mathbf{w},\mathbf{h})p(\mathbf{h})d\mathbf{h}. \tag{2}$$

2.2 Variational Inference

Performing inference on the model described by Equation 2 is intractable and we resort to approximate variational inference [9]. We introduce a variational posterior distribution (or encoder), $q(\mathbf{h}|\mathbf{x})$. We parameterize $q(\mathbf{h}|\cdot)$, and the model, $p(\mathbf{x}|\cdot)$ (or decoder), by neural networks:

$$q_\phi(\mathbf{h}|\mathbf{x}) = \mathcal{N}\left(\mu_\phi\left(\mathbf{x}\right), \Sigma_\phi\left(\mathbf{x}\right)\right) \tag{3}$$

$$p_\theta(\mathbf{x}|\mathbf{w},\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{w},\mathbf{h}), \Sigma_\theta(\mathbf{w},\mathbf{h})) \tag{4}$$

The networks parameterized by θ and ϕ will be described below. We use *Stochastic Variational Inference*[1] to minimize a β-VAE ELBO loss function with the Adam optimizer [12]:

$$\mathcal{L}_{ELBO} = \beta D_{KL}(\underbrace{q_\phi(\mathbf{h}|\mathbf{x})}_{encoder}||p(\mathbf{h})) - \mathbb{E}_q[\log \underbrace{p_\theta(\mathbf{x}|\mathbf{h},\mathbf{w})}_{decoder}] \tag{5}$$

Note that this loss function uses a hyper-parameter, β, that trades off between reconstruction error and independence constraints of the latent variables [8]. The Normal prior helps to encourage interpretability of the latent variables.

Recognition Network As we are using variational inference to learn the trajectory model, we need a family of approximate posterior distributions. We use a recognition network to represent the posterior distribution of the latent variables as Gaussians with diagonal covariance: $q_\phi(\mathbf{h}|\mathbf{x}) = \mathcal{N}\left(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})\right)$. The recognition network uses a *Temporal Convolution Network* (TCN) [13] (see below) to model the trajectory and predict the mean and covariance:

$$\{\mathbf{o}_t\}_{t=1}^T = TCN^{recog}(\mathbf{x}) \tag{6}$$

$$\bar{o} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{o}_t \tag{7}$$

$$\mu_\phi = MLP(\bar{o}) \quad \mathbf{\Sigma}_\phi = diag\left(\exp\left(MLP(\bar{o})\right)\right) \tag{8}$$

Trajectory Decoder We model the trajectory as a function of the latent variables and task parameters: $\mathbf{x} \sim \mathcal{N}\left(\mu_\theta(\mathbf{h},\mathbf{w}), \Sigma_\theta(\mathbf{h},\mathbf{w})\right)$. First, we encode the *task* and *manner* parameters using two separate 2-layer MLPs. Since we want to output a trajectory, we also input a phase variable

[1] We implement our model in PyTorch [10] using the Pyro library [11].

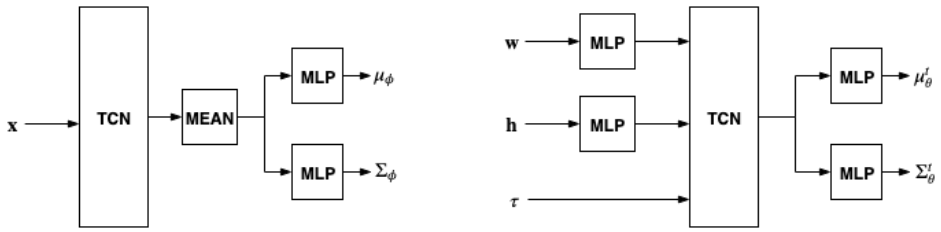

Figure 4: The neural network architecture for the encoder (left) and decoder (right). The encoder predicts the latent variables for a given trajectory while the decoder takes in task parameters, $\mathbf{w}$, the latent variables, $\mathbf{h}$, and a time variable, τ, to construct the trajectory.

$\tau = [\tau_1, \ldots, \tau_T]$, where $\tau_t = t/T$. We concatenate the encoded task and manner variables at each timestep and use this as input to a TCN. Finally, an MLP predicts the mean and diagonal covariance [14, 15] of the trajectory at each timestep. See Figure 4 for an architecture diagram.

$$\mathbf{h}^{enc} = MLP(\mathbf{h}) \quad \mathbf{w}^{enc} = MLP(\mathbf{w}) \tag{9}$$

$$\{\mathbf{i}_t\}_{t=1}^T = [\mathbf{h}^{enc}; \mathbf{w}^{enc}; \tau_t] \tag{10}$$

$$\{\mathbf{o}_t\}_{t=1}^T = TCN(\{\mathbf{i}_t\}_{t=1}^T) \tag{11}$$

$$\mu_\theta^t = MLP(\mathbf{o}_t) \quad \Sigma_\theta^t = \exp(MLP(\mathbf{o}_t)) \tag{12}$$

We found that concatenating the unencoded latent variables, $\mathbf{h}$, to the input of each layer of the decoder helps to prevent latent variable collapse by increasing the mutual information between the latent variables and the trajectory [16].

Temporal Convolution Networks In this work, we frequently use *Temporal Convolution Networks* (TCNs) [13, 17] to model trajectories. TCNs are neural network architectures that have shown success in sequence modeling domains such as language modeling and speech synthesis. TCNs produce an output sequence of length T given an input sequence of the same length by applying stacked dilated convolutions to the input sequence. The dilation provides an exponentially large receptive field, and can be tuned based on the scale of patterns we wish to capture in the data. We use a kernel size of three and a dilation size of two. See Figure 3 for a visualization of a TCN.

2.3 Adversarially Enforcing Independence

Although the model in Equation 2 expresses the independence structure we wish to capture, it is often difficult to enforce the independence in practice. For example, the encoder may encode information about the task parameters into $\mathbf{h}$ instead of forcing the decoder to use $\mathbf{w}$ directly. This is undesirable, as it means modifying the latent variables would also change the task.

To enforce the independence between $\mathbf{w}$ and $\mathbf{h}$, we train the latent space to contain no information about the *task* parameters, $\mathbf{w}$. We accomplish this by adversarially training the encoder, ϕ, with an auxilliary network, f_W (a 3-layer MLP), used to predict $\mathbf{w}$ from the latent space $\mathbf{h}$:

$$\min_W \max_\phi \mathcal{L}_{aux} = \min_W \max_\phi \text{L1}(f_W(\mu_\phi(\mathbf{x})), \mathbf{w}) \tag{13}$$

This objective will encourage the learned latent space to be non-informative about the task parameters and force the decoder to use the provided $\mathbf{w}$ variables [6]. Jointly optimizing the ELBO and adversarial losses leads to an alternating optimization problem:

$$\min_{\theta,\phi} \mathcal{L}_{ELBO} - \alpha\mathcal{L}_{aux} \qquad \min_W \mathcal{L}_{aux} \tag{14}$$

Note that the max operator of the adversarial loss has been incorporated into the ELBO loss with a hyperparameter α. It is important to choose α high enough so that no information about $\mathbf{w}$ is encoded in $\mathbf{h}$. In practice, we do this using a validation set.

3 Experiments

Our goal is to show that the *Task-Conditioned VAE* can learn an interpretable latent space that is independent of the specified *task* parameters. We evaluate our method on both a synthetic drawing

	Arc Drawing	MIME Pouring		
		Real	*Aug.*	*Total*
Train	8000	124	6652	6776
Val	1000	42	462	504
Test	1000	42	N/A	42

Table 1: Dataset sizes for the arc and pouring domains. Each entry consists of the $(\mathbf{x}, \mathbf{w})$-tuple. Note that we only augment the MIME training and validation sets.

Figure 5: Example arcs from the synthetic arc-drawing domain. Each color is a different example.

domain and a tabletop manipulation domain. We use the synthetic domain as an exposition of our method while the robotic domain shows the feasibility of the method with real data. This section describes each dataset (Sections 3.1 and 3.2) and the metrics we use for evaluation (Section 3.3).

3.1 Synthetic Arcs

For the synthetic dataset, we choose an *arc-drawing* domain where trajectories consist of points from an arc drawn in the 2d-plane. We choose this domain as it has clear *task* parameters (the center of the circle the arc is from) and *manner* parameters (e.g., circle radius, starting angle, arclength).

Data Generation To generate data, we first sample the center of the circle from a unit-square, $\mathbf{w} = (x_c, y_c)$. We then sample parameters of the arc such as the radius, the starting angle, and the arclength. To get a trajectory from the arc, we take (x, y) locations at a fixed interval starting at one end: $\mathbf{x} = \{(x_t, y_t)\}_{t=1}^{T}$. This is repeated to sample the training, validation and test sets, each consiting of $(\mathbf{w}, \mathbf{x})$ tuples. See example datapoints in Figure 5.

3.2 MIME Pouring Dataset

To evaluate our method on real data with higher dimensional state spaces, we adapt the *Multiple Interactions Made Easy* (MIME) [7] dataset to our problem. MIME provides a large-scale dataset of manipulation demonstrations on the Baxter platform such as *pouring*, *stirring*, and *stacking*, among others. As our method applies to a single action class where the *task*-parameters are shared, we focus on the pouring action but note that the method remains applicable to other action classes where we can specify a subset of the parameters. We convert each joint-space trajectory to 6-dimensional poses of the robot's end-effector. See Figure 1 for a snapshot from one of the pouring demonstrations.

Preprocessing Our method requires data in the form $(\mathbf{w}, \mathbf{x})$. As MIME does not include any task-specific information, we extract $\mathbf{w}$ algorithmically. Specifically, we choose $\mathbf{w}$ to be the tabletop location of the pour, (x_c, y_c), and the location of the source container, (x_s, y_s). We say the robot is in a pouring state if the angle between the cup it is holding and the world's z-axis is greater than 70 degrees. To extract the location of the pour, we simply average the locations where the robot is in a pouring state. As the MIME dataset has pouring from both arms and allows multiple pours, we restrict our dataset to focus on left-handed trajectories with a single pour and mirror the right-handed pours in the xz-plane to augment the amount of data we have. So that we only capture variation relevant to the pour, we segment each trajectory to only include portions between picking up the source container and completing the pour.

Data Augmentation Due to the small size of the MIME dataset when limited to the pouring action, we programmatically augment the dataset to make training more robust. Specifically, we use *Dynamic Movement Primitives* (DMPs) [1] to alter the goal location of trajectories currently in the training dataset. Note that the augmentation is meant to provide a greater diversity of *task* parameters which will help the model learn to disentangle these from the learned *manner* parameters. For each training trajectory, we fit a DMP and generate new trajectories with between 0-5cm added to the pouring location. We perturb the weights of the DMP by a small amount to introduce noise into the trajectories. We also found that randomly rotating and translating the entire trajectory was a useful form of data augmentation. The number of augmented trajectories can be seen in Table 1. Note that we augment the validation set so that it has a similar distribution to the training set but do not augment the test set so that we only evaluate on real data.

3.3 Evaluation Metrics

To evaluate our model, we need to show that: (1) The learned *manner* parameters are independent of the specified *task* parameters and (2) the latent space interpretably controls axes of variation we care about. A useful notion to help evaluate (1) are the *empirical task-parameters*, $\hat{\mathbf{w}}$. Given any trajectory, $\mathbf{x}$, we provide functions, $\hat{\mathbf{w}} = G(\mathbf{x})$ that extract what the task parameters must have been to generate that trajectory. For example, in the pouring domain, $G^{pour}(\cdot)$ returns the location where the angle between the z-axis and the cup is greater than 70 degrees. For the *arc*-drawing domain, $G^{arc}(\cdot)$ extracts the center of a trajectory representing an arc. We propose the following metrics to capture these performance criteria:

Task Success For each dataset, we would like to capture whether the trajectories generated by our model can successfully complete the task specified by the *task* parameters. To do this, we autoencode the trajectories in our test set to get $\hat{\mathbf{x}}$, the trajectory that would be generated using the most likely latent variables according to the learned posterior distribution. We then check if the empirical *task*-parameters are within a threshold, λ, of the true *task* parameters:

$$\textsc{Success}(\mathbf{x}, \mathbf{w}) = \mathbb{I}(d(G(\hat{\mathbf{x}}), \mathbf{w}) < \lambda) \tag{15}$$

Robustness Intuitively, the *learned manner* parameters should not change aspects of the motion related to the *task* parameters. To measure if the *learned* parameters are independent of the *task* parameters, for each $\mathbf{w}$ in the test set, we sample a set of K *learned* parameters from the prior. We then check if the empirical *task*-parameters are the same as the true parameters for all these samples (recall $\mu_\theta(\cdot, \cdot)$ is the decoder):

$$\{\mathbf{h}_k\}_{k=1}^K \sim p(\mathbf{h}) \quad \textsc{Robustness}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K} \textsc{Success}(\mu_\theta(\mathbf{w}, \mathbf{h}_k), \mathbf{w}) \tag{16}$$

We report the expected success and robustness on the test-set in Table 2 for various thresholds, λ.

We compare our model to *Task-Parameterized Gaussian Mixture Models* (TP-GMMs) [3] as a baseline that generalizes to new task parameters but does not offer a means of controlling the remaining captured variation. More details on the TP-GMM experimental setup can be found in the Appendix. We further compare our model to ablated versions to analyze the importance of our contributions: conditioning on task information and the adversarial loss. Specifically, we train a model without the adversarial loss, TC-VAE (no $\mathcal{L}_{aux}$). We also train a VAE that does not condition on any task information as a baseline that will have trouble separating task and manner parameters.

4 Results

In this work, we focus on learning a task-independent parameter space for trajectory generation. In this section, we show: (1) The learned latent space is independent of the *task* parameters and (2) the latent variables are interpretable and control semantically meaningful axes of variation.

4.1 Task Disentanglement

We first evaluate if the learned *manner* parameters are independent of the specified *task* parameters. We do this using the ROBUSTNESS score – can the model perform the same task with different settings of the learned parameters? In Table 2, we can see that the model that includes the adversarial loss has the highest robustness score across thresholds and datasets. This indicates that varying the latent space does not affect the success of the task being performed and that our model can properly disentangle the *task* parameters from the *manner* parameters.

Figures 6 and 7 show how the *task* parameters do not change as we sample new *manner* parameters. Specifically, we see how the arc centers do not change across latent variable settings in Figure 6. In Figure 7, we see that the pouring location remains within a 2cm radius of the specified pouring location when varying the learned parameters.

Incorporating the adversarial information loss increases the ROBUSTNESS score for each dataset (see Table 2). The model without this auxiliary training objective fails to perform the main task more frequently in both domains and with lower thresholds. The intuition behind this performance

Model	Synthetic Arcs					
	SUCCESS			ROBUSTNESS		
	$\lambda = 0.025$	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.025$	$\lambda = 0.05$	$\lambda = 0.1$
VAE	0.20 ($\pm$0.02)	0.92 ($\pm$0.02)	**0.99** ($\pm$0.01)	0.0003 ($\pm$0.01)	0.0048 ($\pm$0.01)	0.02 ($\pm$0.01)
TC-VAE (no $\mathcal{L}_{aux}$)	**0.66** ($\pm$0.03)	**0.98** ($\pm$0.01)	**0.99** ($\pm$0.01)	0.07 ($\pm$0.01)	0.27 ($\pm$0.01)	0.39 ($\pm$0.01)
TC-VAE	0.50 ($\pm$0.03)	**0.97** ($\pm$0.01)	**0.99** ($\pm$0.01)	**0.43** ($\pm$0.01)	**0.89** ($\pm$0.01)	**0.94** ($\pm$0.01)

Model	MIME Pouring					
	SUCCESS			ROBUSTNESS		
	$\lambda = 0.01m$	$\lambda = 0.02m$	$\lambda = 0.05m$	$\lambda = 0.01m$	$\lambda = 0.02m$	$\lambda = 0.05m$
TP-GMM [3]	**0.45** ($\pm$0.15)	**0.68** ($\pm$0.14)	**0.88** ($\pm$0.10)	N/A	N/A	N/A
VAE	0.0 ($\pm$0.03)	0.07 ($\pm$0.08)	0.31 ($\pm$0.14)	0.0 ($\pm$0.01)	0.01 ($\pm$0.01)	0.04 ($\pm$0.02)
TC-VAE (no $\mathcal{L}_{aux}$)	**0.29** ($\pm$0.14)	**0.64** ($\pm$0.15)	**0.86** ($\pm$0.10)	0.25 ($\pm$0.06)	0.59 ($\pm$0.07)	0.93 ($\pm$0.03)
TC-VAE	**0.36** ($\pm$0.15)	**0.81** ($\pm$0.12)	**0.83** ($\pm$0.11)	**0.47** ($\pm$0.07)	**0.80** ($\pm$0.05)	**1.0** ($\pm$0.01)

Table 2: Performance of our full model (TC-VAE) compared to ablated baselines that do not use the adversarial loss (no $\mathcal{L}_{aux}$) or the task information (VAE). SUCCESS is the fraction of the test set where the model can complete the specified task (within threshold λ). ROBUSTNESS measures if changing the learned parameters affects task success. TP-GMM does not have manner parameters to calculate robustness. 95% C.I.s in parentheses.

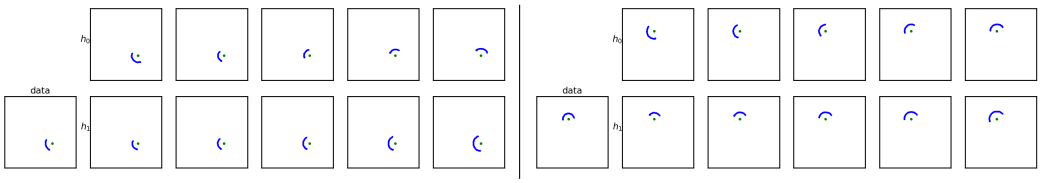

Figure 6: Visualization of the learned latent space for two examples from the arc-drawing dataset. Each example has a fixed *task* parameter (arc-center). Each row corresponds to changing a specific latent variable between (-1, 1). The first row, h_0, corresponds to the start angle of the arc, while the second row, h_1, corresponds to the arc length. The image on the left of each panel represents a trajectory from the dataset with the same center.

difference is that even with the TC-VAE structure (but not adversarial loss), the model still encodes information about the task into the latent space. The adversarial loss forces the decoder to rely only on the provided *task* parameters when generating a trajectory.

Both TC-VAE models perform as well as the TP-GMM baseline under the SUCCESS metric but have the added benefit that they have recovered additional parameters that control the remaining variation. We note that all models generally perform well with respect to the SUCCESS metric, which uses the most likely parameters of the posterior distribution of a trajectory. However, to achieve our goal of learning a task-disentangled parameter space, the models must perform well on *both* SUCCESS and ROBUSTNESS metrics (note that TP-GMM does not have a method to control the variation captured by the model and thus does not permit a ROBUSTNESS score). The VAE model does not perform well on either metric on the MIME dataset. We attribute this to the fact that VAE does not have access to the *task* parameters and has trouble extracting them from the noisy data. All models that have access to the *task* parameters outperform the VAE baseline that does not, showing the effectiveness of conditioning.

4.2 Latent Space Visualization

We next analyze the interpretability of the learned *manner* parameters. We desire parameters that are interpretable so that the learned parameter space can be easily controlled by a human. In Figures 6 and 7, we visualize how the trajectory changes while separately varying each of the learned parameters within the range $[-\sigma, \sigma]$ of the prior (while keeping the *task* parameters constant).

For the arc dataset (Figure 6), we see that the first latent dimension controls the starting angle of the arc and varying it causes the arc to rotate. The second learned parameter controls the arc length with a higher value corresponding to longer arcs. Note that in both examples, no matter the setting of the learned *manner* parameters, the location of the arc's center (green dot) remains the same.

For the MIME dataset (Figure 7), to visualize quantities of interest, we plot the angle between the cup and the z-axis (extracted from the TC-VAE output in 6-dimensional end-effector space) as we vary the second and fourth latent dimensions. We also draw a pouring threshold line where we consider the robot to be pouring for any angle above this line (70 degrees). We see that the second latent dimension controls how early the pouring occurs within each trajectory. Increasing this parameter

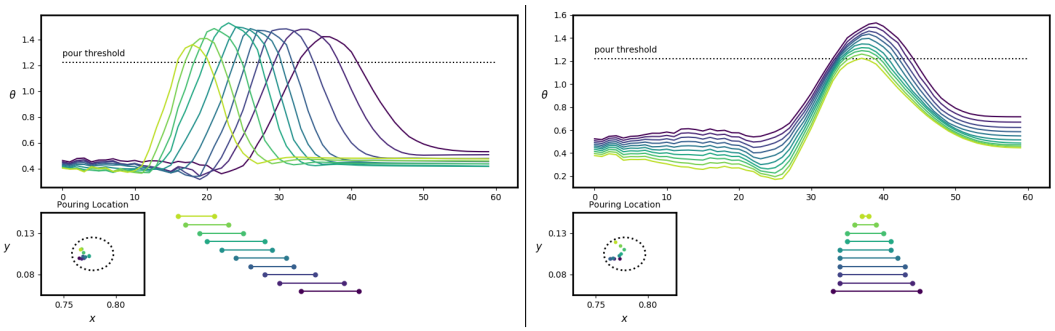

Figure 7: Visualization of the learned manner parameters for the *pouring* data and a fixed *task* parameter. Each plot shows the angle between the cup and the z-axis for the trajectory while varying a single learned parameter (h_2 on the left, and h_4 on the right). Each trajectory corresponds to different settings of that learned parameter in the range (-1, 1). The lines underneath show the corresponding length of the pour for the respective trajectory. h_2 can be seen to control how early the pour occurs and h_4 the duration of the pour. The bottom left plots show the pouring location of each trajectory and a $2cm$ radius around the specified pouring location.

causes the pour to occur later. The fourth latent dimension varies the the duration of the pour. Note that the second and fourth latent dimensions are independent of each other. That is, changing h_4 does not change the time the pour begins. Additional visualizations can be found in the Appendix.

5 Related Work

There has been much work on developing movement primitives that are easily transferable to new tasks. *Dynamic Movement Primitives* (DMPs) [1] can adapt the endpoint of a movement or its duration to those seen outside of training. DMPs have been widely extended to handle visual coupling [18] and support adaptation via reinforcement learning [19]. *Probabilistic Movement Primitives* (ProMPs) [20] and their extensions [2, 21] use a probabilistic representation where modulating the trajectory is performed by conditioning the trajectory distribution. *Latent Manifold Probabilistic Movement Primitives* (LMProMPs) [4] extend this approach to extract low-dimensional control parameters for the policy. Another framework that focuses on generalizing movement primitives to new tasks is the *Task Parameterized Gaussian Mixture Model* (TP-GMM) which parameterizes a task by specifying task-specific coordinate frames [3, 22, 23]. Although the experiments in this work use task parameters that can be interpreted in such a way, there is no limitation preventing us from using more abstract task parameterizations. We follow in this line of work and allow both specified and learned parameters by ensuring the parameters sets are independent of each other.

Much of the recent literature on deep generative modeling has focused on learning disentangled, interpretable representations [8, 24]. *Fader Networks* [5] and *Adversarial Information Factorization* [6] employ an adversarial loss to learn how to modify image attributes without changing other aspects of the image. Our work uses a similar method to modify trajectories. This technique has also been used in the fairness literature to censor sensitive information from learned representations [25].

Our work is related to the *Reinforcement Learning* literature that focuses on learning interpretable policies. *Independently Controllable Factors* [26] and *InfoGAIL* [27] both encourage learning meaningful representations of the environment. *Contextual Policy Search* [28] uses context variables to adapt a policy to new tasks. Finally, several efforts have addressed robotic pouring [29, 30, 31, 32, 33]. Whereas many of these methods focus on precision pouring, we learn an interpretable movement primitive from demonstrations.

6 Conclusion

In this work, we proposed the *Task-Conditioned Variational Autoencoder* (TC-VAE) – a generative model that explicitly differentiates between *what* task to perform and *how* that task should be performed. We showed that by utilizing an adversarial information objective, this model can learn a low dimensional latent-space that is independent of the task and generate trajectories that incorporate both types of parameters. This enables demonstrations to be modified in coherent ways.

Acknowledgments

We gratefully acknowledge funding support in part by the Honda Research Institute, the U.S. Army Research Laboratory under the Robotics Collaborative Technology Alliance (RCTA) Program, Lockheed Martin Co., and the Toyota Research Institute Award LP-C000765-SR.

References

[1] S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[2] A. Colomé, G. Neumann, J. Peters, and C. Torras. Dimensionality reduction for probabilistic movement primitives. In *International Conference on Humanoid Robots*. IEEE, 2014.

[3] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, 2016.

[4] E. Rueckert, J. Mundo, A. Paraschos, J. Peters, and G. Neumann. Extracting low-dimensional control variables for movement primitives. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[5] G. Lample, N. Zeghidour, N. Usunier, A. Bordes, L. Denoyer, et al. Fader networks: Manipulating images by sliding attributes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[6] A. Creswell, Y. Mohamied, B. Sengupta, and A. A. Bharath. Adversarial information factorization. *arXiv preprint arXiv:1711.05175*, 2017.

[7] P. Sharma, L. Mohan, L. Pinto, and A. Gupta. Multiple interactions made easy (mime): Large scale demonstrations data for imitation. In *Conference on Robot Learning (CoRL)*, 2018.

[8] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations (ICLR)*, 2017.

[9] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[11] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *arXiv preprint arXiv:1810.09538*, 2018.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[13] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[14] K. Liu, K. Ok, W. Vega-Brown, and N. Roy. Deep inference for covariance estimation: Learning gaussian noise models for state estimation. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[15] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[16] A. B. Dieng, Y. Kim, A. M. Rush, and D. M. Blei. Avoiding latent variable collapse with generative skip models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[17] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[18] J. Kober, B. Mohler, and J. Peters. Learning perceptual coupling for motor primitives. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ.

[19] J. Kober and J. Peters. Learning motor primitives for robotics. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2009.

[20] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems (NeurIPS)*, 2013.

[21] A. Colomé and C. Torras. Demonstration-free contextualized probabilistic movement primitives, further enhanced with obstacle avoidance. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2017.

[22] M. J. Zeestraten, S. Calinon, and D. G. Caldwell. Variable duration movement encoding with minimal intervention control. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

[23] J. Silvério, S. Calinon, L. Rozo, and D. G. Caldwell. Learning task priorities from demonstrations. *IEEE Transactions on Robotics*, 35(1):78–94, 2018.

[24] E. Mathieu, T. Rainforth, S. Narayanaswamy, and Y. W. Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning (ICML)*, 2019.

[25] H. Edwards and A. Storkey. Censoring representations with an adversary. In *International Conference on Learning Representations (ICLR)*, 2016.

[26] V. Thomas, E. Bengio, W. Fedus, J. Pondard, P. Beaudoin, H. Larochelle, J. Pineau, D. Precup, and Y. Bengio. Disentangling the independently controllable factorsof variation by interacting with the world. In *NIPS workshop on Learning Disentangled Representations*, 2017.

[27] Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[28] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

[29] A. Yamaguchi, C. G. Atkeson, S. Niekum, and T. Ogasawara. Learning pouring skills from demonstration and practice. In *International Conference on Humanoid Robots*. IEEE-RAS, 2014.

[30] Z. Pan, C. Park, and D. Manocha. Robot motion planning for pouring liquids. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[31] M. Kennedy, K. Schmeckpeper, D. Thakur, C. Jiang, V. Kumar, and K. Daniilidis. Autonomous precision pouring from unknown containers. *IEEE Robotics and Automation Letters*, 4(3): 2317–2324, 2019.

[32] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[33] C. Do and W. Burgard. Accurate pouring with an autonomous robot using an rgb-d camera. In *International Conference on Intelligent Autonomous Systems*. Springer, 2018.

Appendix

A TC-VAE Hyperparameters

We fit the TC-VAE model to each dataset using 4 latent variables (the VAE baseline uses 10 latent variables as it needs to encode richer information). The TCN uses 3-stacked convolutions with kernel size 3, and dilation size 2. β was set to 4 for the arc dataset and 8 for the pouring dataset. The hyperparameters were chosen using the validation sets.

B TP-GMM

Task Parameterized Gaussian Mixture Models (TP-GMMs) use coordinate frames as task parameters and fit a GMM to a trajectory represented by the coordinate frames [3]. In the MIME dataset, we use two coordinate frames given by the location of the pouring container (x_c, y_c) and the location of the source container (x_s, y_s). We assume the same orientation as the world frame and height of the table for the z axis of each coordinate frame. Before fitting the model, we align the trajectories using dynamic time warping. We fit the unaugmented training trajectories to a TP-GMM with 25 Gaussians and evaluate the model using *Gaussian Mixture Regression* to generate trajectories for the task parameters in the test set. We thank the authors for the code provided in [3].

C Additional Visualizations

Figures 8, 9, and 10 visualize the learned manner parameters for the TC-VAE, TC-VAE (no $\mathcal{L}_{aux}$), and VAE models respectively. Note the first two models each used h of dimensions 4 (we only visualize the first 4 dimensions of the VAE model).

Figure 8 shows all dimensions of the learned parameters for the TC-VAE model. While h_2, h_3, and h_4 correspond to semantically meaningful variations (starting time, maximum pouring angle, and duration respectively), h_1 does not capture much variation relating to the pour angle. Note that h_3 and h_4 both capture variation in the maximum pouring angle and thus are not semantically independent from each other.

All sampled trajectories from the TC-VAE model remain within 2cm of the desired pouring location. The ablated models are not as robust to changes in the learned manner parameters. For the TC-VAE (no $\mathcal{L}_{aux}$) model, the pouring locations are often more than 2cm from the desired locations (see the pouring location plots in Figure 9). In the VAE, most trajectories do not come close to the desired pouring location as the latent space is forced to encode task-specific information in this model (see Figure 10).

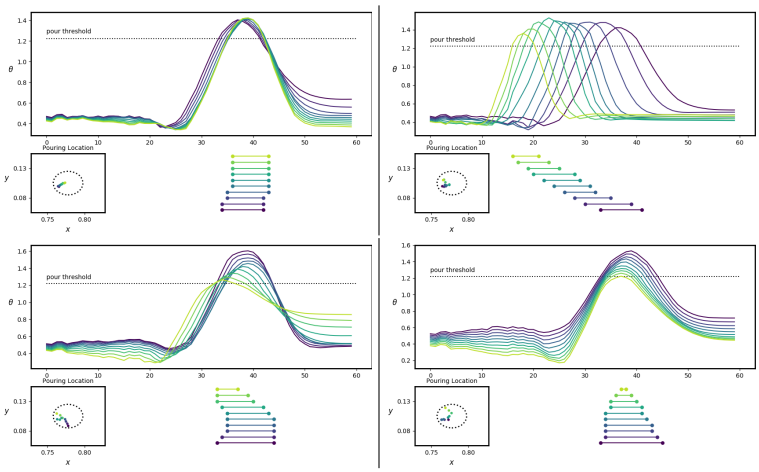

Figure 8: Visualization of the learned manner parameters for the TC-VAE model. Each plot shows the angle between the cup and the z-axis for the trajectory while varying a single learned parameter (h_1 on the top left to h_4 on the bottom right). Each trajectory corresponds to different settings of that learned parameter in the range (-1, 1). The lines underneath show the length of the pour for the respective trajectory. The pouring locations for each trajectory and a 2cm radius around the desired location are shown in the bottom left of each plot.

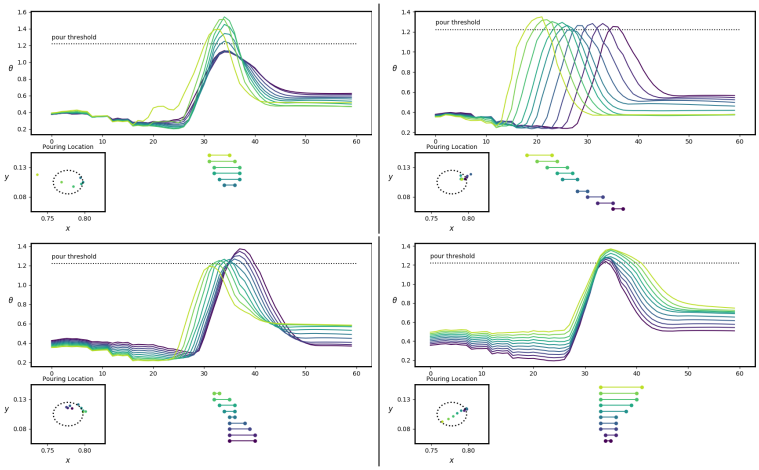

Figure 9: Visualization of the learned manner parameters of the TC-VAE (no $\mathcal{L}_{aux}$) model. Notice how the pouring locations (bottom left plots) are farther from the desired location than in the TC-VAE case.

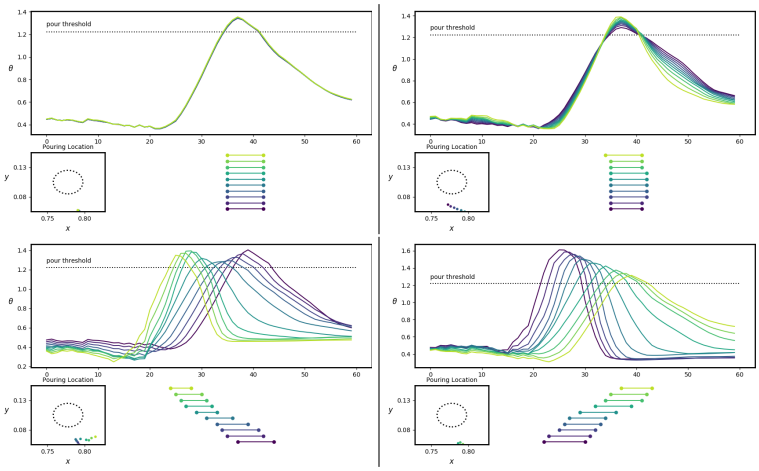

Figure 10: Visualization of the learned manner parameters of the VAE model. Note that since the VAE encodes task information in the latent space, most pours are far from the desired location.