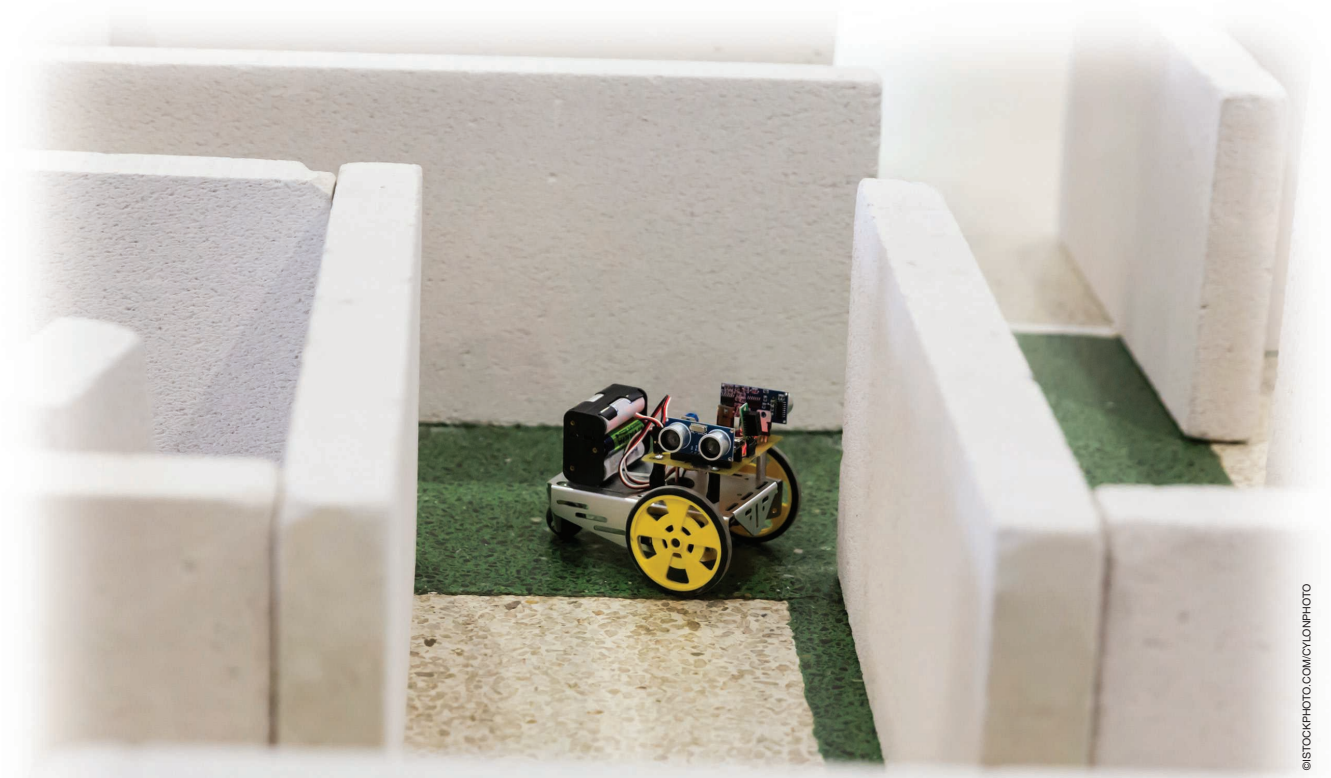
Movement Primitive Learning and Generalization

Using Mixture Density Networks



©ISTOCKPHOTO.COM/CYLONPHOTO

By You Zhou, Jianfeng Gao, and Tamim Asfour

Representing robot skills as movement primitives (MPs) that can be learned from human demonstration and adapted to new tasks and situations is a promising approach toward intuitive robot programming. To allow such adaptation, mapping between task parameters and MP parameters is needed, and different approaches have been proposed in the literature to learn such mapping. In human demonstrations, however, multiple modes and models exist, and these should be taken into account when learning these mappings and generalized MP representations.

Here, a challenging problem is mode or model collapse. To solve this problem, we propose using a mixture density network (MDN) that takes task parameters as input and provides a Gaussian mixture model (GMM) of the MP parameters. To avoid mode and model collapse during MDN training, we introduce an entropy cost to achieve a more balanced association of demonstrations to GMM components. Since it is often easier to collect failed examples using an underfitted MDN model instead of additional human demonstrations, we introduce a failure cost to reduce the occurrence of failures in future executions. We evaluated our approach in simulation and real robot experiments and showed that the method outperforms previous approaches.

Digital Object Identifier 10.1109/MRA.2020.2980591
Date of current version: 6 April 2020

Challenges of Imitation Learning

Over the past decades, robotics researchers have developed different approaches that enable robots to learn from humans, imitate human behavior, and autonomously improve their

1070-9932/20©2020IEEE

movements. As a replacement for manual robot programming, learning from human demonstrations, also called *imitation learning*, has been proven a promising and powerful technique for intuitive robot programming [1]. Inspired by neuropsychological findings [2], we use MPs as essential building blocks for describing robot motions. In this context, the question of how a generalizable and compact representation of MPs can be learned from demonstrations and adapted to new situations and changing task parameters is an active research area in robotics.

Different MP representations have been proposed, such as dynamic MP (DMP) [3], probabilistic MP (ProMP) [4], and task-parameterized GMM (TP-GMM) [5]. These representations can adapt to task parameters in the space in which we define these primitives. For example, a DMP adapts to a new start or goal; a ProMP adapts to intermediate via points; and TP-GMM adapts to changes of predefined local frames, where we describe the demonstrated trajectories. However, for different tasks, the parameters could have different meanings and are not directly associated with spatial or temporal requirements for motion trajectories. For example, a robot throwing a ball to a target specified in the Cartesian space by executing a motion learned and represented in the robot's joint space should be able to adapt the learned motion to new targets. None of these approaches can adapt a learned MP to new targets specified in a different space.

For such generalization problems, methods have been proposed in which task-specific mapping between the task and MP parameters is learned through regression models, such as locally weighted regression (LWR) [6], [7], Gaussian process regression (GPR) [8], or deep neural networks [9]. However, such methods cannot deal with the problems of mode and model collapse. Given the task of reaching a target while avoiding an obstacle (see Figure 1), the goal can be reached using two different modes, i.e., by trajectories passing the obstacle from the left or right. Thus, learning an MP for such a task should take multiple modes in human demonstrations into account to increase the diversity of motions, which is beneficial in the case of changing task constraints.

Even though there are no multiple modes for each individual task parameter query in human demonstrations, there might exist multiple models for different types of task parameters. As shown in Figure 1(b), the goals on the left and right sides of the obstacle are reached separately by two types of trajectories, resulting from two different models, to allow the obstacle to be passed from the left or right.

In many applications, human demonstrations might use multiple modes and models at the same time. To avoid both mode and model collapse, we propose learning a mapping from the task parameter query $\boldsymbol{q}$ to a GMM of the MP parameters $\boldsymbol{w}$ with MDNs. Each mixture component of the GMM represents either a mode for one specific task parameter query or a model for multiple task parameter queries. However, training an MDN with only the negative log-likelihood (NLL) cost, as is usually done in the original MDN, might still lead to mode and model collapse, especially when the set of demonstrations is relatively small, as shown in Figure 1. To further reduce the occurrence of both collapses, we introduce an entropy cost function for training the MDN (entropy MDN).

Figure 1(a) shows the human demonstrations (dashed curves) for obstacle avoidance with an imbalanced distribution of examples associated with the two different modes as well as the results obtained with different approaches for a new task parameter query (the green dot). As can be seen, our approach (entropy MDN) generates multiple solutions (here, five solutions are shown in Figure 1) that cover both demonstration modes (blue and green) for the same task parameter query. We achieve this result with the entropy cost function, which ensures a more balanced probability associated with

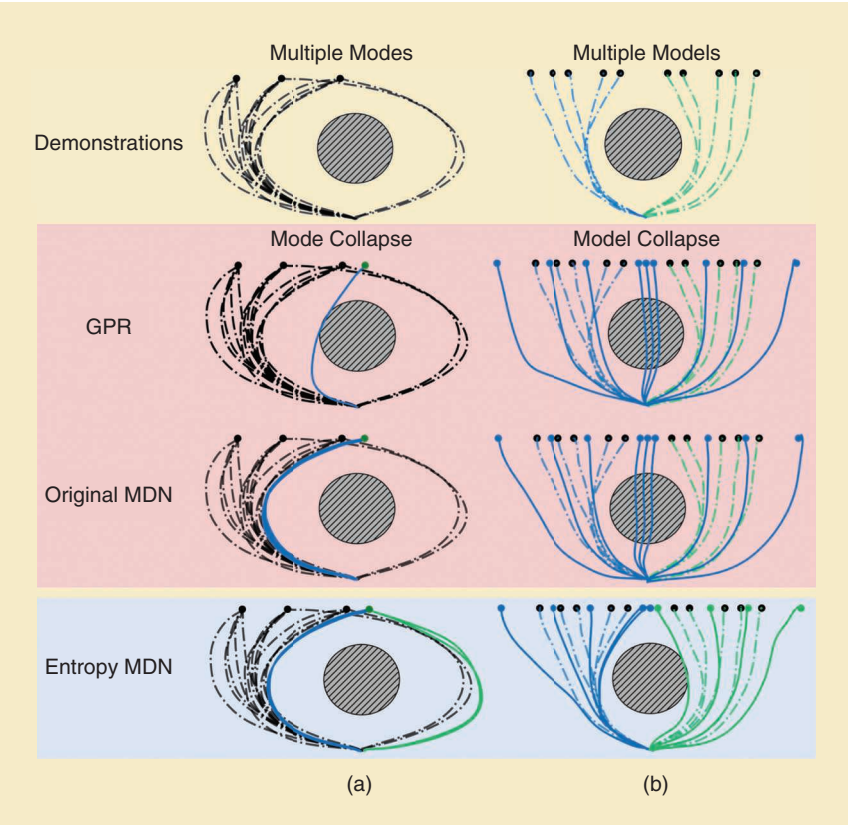


Figure 1. The (a) mode and (b) model collapse issues in the obstacle-avoidance problem are solved using the entropy MDNs. The dashed curves denote the demonstrations, and the solid curves show the generated trajectories.

the different modes. The original MDN generates multiple solutions, but these solutions (also five) reflect only one mode in the human demonstrations since it is associated with a high probability for the dominant mode. The GPR generates only one single solution that is close to the examples for the dominant mode.

Figure 1(b) shows the demonstrations (blue and green dashed curves) associated with two different models and the trajectories generated by different approaches (solid curves) for several task parameter queries (the colored dots). As shown at the bottom of Figure(b), our approach (entropy MDN) successfully learns the two models and generates the solution for the task parameter query correspondingly. We also achieve this result with the entropy cost function, which ensures that each model is associated with some human demonstrations. The original MDN has a higher chance of being stuck in the local minima of the NLL, where only one model is trained and overfitted for all task parameter queries. This fact leads to the failure of the task execution, i.e., collision with the obstacle, especially when the task parameter queries are on the boundary of two models. The GPR can learn only one model; thus, it suffers from the same problem as the original MDN.

For many tasks, it is often easier to collect failed samples with an underfitted MDN model instead of requesting additional human demonstrations. To further improve the MDN performance, we introduce a failure cost function that reduces the occurrence of the same failures for a given task parameter query.

Related Works

DMP is one of the popular approaches to representing robot motions. A DMP consists of a damped spring system and a nonlinear force term $f(x)$ such that

$$\begin{aligned} \tau\dot{v} &= K(g-y) - Dv + (g-y_0)f(x)x, \\ \tau\dot{y} &= v, \\ f(x) &= \frac{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)\,\boldsymbol{w}_i}{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)}, \end{aligned} \tag{1}$$

where v and y are the scaled velocity and position of the trajectory point, respectively; g, y_0, and τ, as the hyperparameters, represent the goal, start, and temporal factor separately; and x is the canonical variable, which goes from 1 to 0. The force term $f(x)$ is a linear regression model with N squared exponential kernels (SEKs) $\boldsymbol{\psi}_i(x) = \exp(-h_i(x-c_i)^2)$, where h_i, and c_i are fixed constants. $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N)^T$ is a vector representing the DMP parameters.

DMP generalization means replacing $\boldsymbol{w}$ with a parameterized function $\omega(\boldsymbol{q})$. As mentioned previously, this function can be represented and learned with different approaches, such as GPR [8], LWR [6], [7], support-vector regression (SVR) [10], or deep neural networks [9]. To train those models, the training data set is collected as M pairs of task parameter queries and MP parameters $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$, where the ith MP parameter $\boldsymbol{w}_i$ is learned from the ith demonstration and corresponds to the ith task parameter $\boldsymbol{q}_i$.

Those methods require two parameterized functions, $f(x)$ and $\omega(\boldsymbol{q})$. The output of $\omega(\boldsymbol{q})$ is the parameter $\boldsymbol{w}$ of $f(x)$; hence, they are called *two-step methods*. In [11], Stulp et al. proposed combining two functions into one single function $f(x, \boldsymbol{q})$, which was learned with LWR or GPR and extended to the GMM in [12]. Since these methods use only one single function, they are called *one-step methods*, and they are more compact than the two-step methods.

The idea of the TP-GMM, suggested in [5], is to observe human demonstrations from multiple perspectives (local frames). A global GMM represents the trajectory points in a global frame and maximizes the likelihood of demonstrations from different perspectives. With the transformation of the local frames, the TP-GMM achieves a better extrapolation performance than other methods. In [13], Pignat and Calinon extended the TP-GMM and learned the sensory data together with the motion trajectories. As mentioned before, however, the task parameters considered by the TP-GMM are limited.

The ProMP uses a linear regression model with kernel functions $\boldsymbol{\psi}(\cdot)$ to directly represent the motion trajectory $y(x) = \boldsymbol{\psi}(x)^T\boldsymbol{w}$. In [4], Parachos et al. assumed that $\boldsymbol{w}$ follows a Gaussian distribution. In [14], the Gaussian distribution was extended to a GMM. For human–robot interactions, the ProMP parameter $(\boldsymbol{w} = \{\boldsymbol{w}_o, \boldsymbol{w}_c\})$ is separated to encode both human $(\boldsymbol{w}_o)$ and robot motions $(\boldsymbol{w}_c)$. With the conditional probability, the robot MP parameter $\boldsymbol{w}_c$ is inferred based on the human MP parameter $\boldsymbol{w}_o$. Both methods in [13] and [14] learn a generative model and use the conditional probability to infer unknown variables based on known ones.

In this article, we use a via-points MP (VMP), an MP formulation presented in our previous work [15] and described in the "The VMP" section. Instead of learning a generative model, we learn an MDN to directly map from a task parameter $\boldsymbol{q}$ to a GMM of the MP parameters $\boldsymbol{w}$. As a GPR or SVR for DMP, our technique belongs to the two-step methods.

MP Generalization

The VMP

We use the VMP (see [15]) to represent robot motions, which consists of an elementary trajectory h and shape modulation f as follows:

$$y(x) = h(x) + f(x) = g + x(y_0 - g) + \boldsymbol{\psi}(x)^T\boldsymbol{w}, \tag{2}$$

where x is the canonical variable, which goes from 1 to 0 with a linear-decay canonical system. The elementary trajectory takes the form of a polynomial; here, it is a first-order polynomial, i.e., a line connecting the start and the goal. By changing y_0 and g as the hyperparameters, the VMP adapts to the new start and goal. As the nonlinear force term in DMP, the shape modulation is a linear regression model with the SEKs $\boldsymbol{\psi}$. $\boldsymbol{w}$ is the parameter vector of the VMP.

In [15], we assumed that the parameter $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ follows a Gaussian distribution and that the maximum likelihood estimation of $\boldsymbol{\mu}$ is the empirical mean of all $\boldsymbol{w}$s, each of which corresponds to one demonstration and is obtained by solving a least-square problem.

Here, we extend the Gaussian distribution to a GMM and consider task parameter queries $\boldsymbol{q}$ as inputs:

$$\boldsymbol{w} = \omega(\boldsymbol{q}) \sim \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{q}), \Sigma_k(\boldsymbol{q}))^{z_k}, \tag{3}$$

where z_k is an element of a K-dimensional binary variable $\boldsymbol{z} = (z_1, z_2, \ldots, z_K)^T$, with only one particular element being equal to one and all other elements being zero. The probability of the kth element of $\boldsymbol{z}$ being equal to one is

$$p(z_k = 1) = \pi_k(\boldsymbol{q}).$$

The probability of the result $\boldsymbol{w}$ given the task parameter $\boldsymbol{q}$ is

$$p(\boldsymbol{w}|\boldsymbol{q}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{q})\, \mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{q}), \boldsymbol{\Sigma}_k(\boldsymbol{q})). \tag{4}$$

We assume that the number of the mixture components K of the GMM is known and that $\{\pi_k(\cdot), \boldsymbol{\mu}_k(\cdot), \boldsymbol{\Sigma}_k(\cdot)\}_{k=1}^{K}$ are the functions to be learned.

The advantage of the VMP over DMP is its ability to adapt to intermediate via points by modifying the elementary trajectory $h(x)$ (see [15]). Apart from the via-points adaptation, extending the force term in a DMP to a GMM makes VMPs and DMPs exchangeable. Since the ProMP lacks the elementary trajectory and has no hyperparameters y_0 and g, the ProMP parameter function $\boldsymbol{w} = \omega(\boldsymbol{q})$ determines both the motion trajectory shape and its start and goal. In many tasks, the start and goal are a part of the task parameter queries. With a VMP, we reduce the learning complexity, because one requirement of the task, i.e., reaching a new goal, is directly satisfied by the hyperparameter g.

For example, in Figure 2, in contrast to a VMP, not all of the trajectories generated by the ProMP reach the goal if we use the entropy MDN and select the most probable parameter $\boldsymbol{w}$ from the output distribution. In some tasks, however, the goal is not a part of the task parameters and is necessary for the task execution. For these tasks, the VMP requires learning an additional mapping from the task parameters to the goals, whereas the ProMP provides a more compact solution. As shown in Figure 2, the entropy MDN suggested for VMP generalization also works for ProMP generalization.

In general, for MP generalization, M human demonstrations for different task parameter queries are collected. The purpose is to learn an MDN $[\omega(\cdot)]$ mapping from the task parameter query $\boldsymbol{q}$ to the parameter distribution of the MP parameter $\boldsymbol{w}$. For MDN, we have an assumption that the number of mixture components K of the output GMM is known.

The MDN

Using neural networks to learn the functions in (4) results in an MDN (see [16]). The mixing coefficients $\pi_k(\cdot)$, mean $\boldsymbol{\mu}(\cdot)$, and covariance $\boldsymbol{\Sigma}(\cdot)$ are represented by the network branches, as shown in Figure 3. These network branches

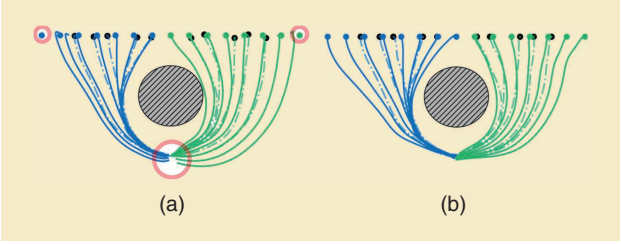


Figure 2. A comparison of (a) a ProMP and (b) a VMP. The dashed curves are demonstrations, and the solid curves are generated trajectories for different goals. The red circles indicate the starts and goals missed by the ProMP.

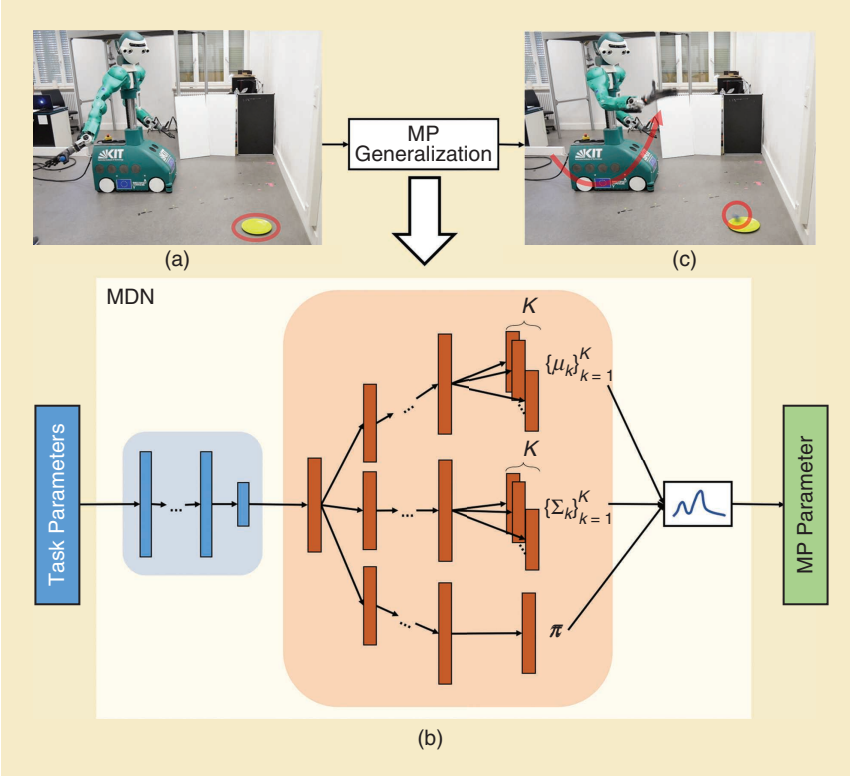


Figure 3. The MDN proposed for MP generalization. As an example, (a) with the target as the task parameter indicated by the red circle, the system (b) generates an MP parameter corresponding to (c) the motion of throwing the ball (see the red curves) on the target.

share a common network part (the blue box), which allows extracting common latent features.

We assume that the covariance is a diagonal matrix $\boldsymbol{\Sigma}$. According to [17], a GMM with a diagonal variance matrix approximates any given density function to arbitrary accuracy. The diagonal covariance output has the same dimension as the mean output. For K components, an MDN has K pairs of mean and variance outputs. Each pair corresponds to a mixture component of a GMM.

The dimension of the MP parameters, as the output of the MDN, determines the accuracy of the trajectory representation. With more SEKs, the MP represents the motion in a more accurate way. In the following experiments, we use 10 SEKs for each dimension. As an example, the output mean vector has 30 dimensions for a 3D motion trajectory, and there are a total of $K + K \times 30 \times 2$ values in the MDN output.

For one single demonstration, we calculate $\boldsymbol{w}$ by solving the least-square problem for (2). With M demonstrations, a training data set $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$ is collected. The NLL is written as

$$l_{\mathrm{NLL}}(\boldsymbol{\Theta}) = -\sum_{i=1}^{M} \log\left(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i;\boldsymbol{\Theta})\, \mathcal{N}(\boldsymbol{w}_i;\boldsymbol{\mu}(\boldsymbol{q}_i;\boldsymbol{\Theta}), \boldsymbol{\Sigma}(\boldsymbol{q}_i;\boldsymbol{\Theta}))\right), \tag{5}$$

where $\boldsymbol{\Theta}$ is the parameters of the network, and

$$\mathcal{N}(\boldsymbol{w}_i;\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}_i|^{1/2}} \cdot \exp\left(-\frac{1}{2}\sum_{j=1}^{d} \frac{(w_{i,j}-\mu_{i,j})^2}{\sigma_{i,j}^2}\right), \tag{6}$$

where d is the dimension of the output, $\boldsymbol{\mu}_i = \boldsymbol{\mu}(\boldsymbol{q}_i;\boldsymbol{\Theta})$, and $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}(\boldsymbol{q}_i;\boldsymbol{\Theta})$. A stochastic gradient descent method is used to minimize the NLL.

According to previous works [18], [19], training an MDN with the NLL suffers from mode collapse. In [18], to avoid the mode collapse and reduce learning complexity, Hjorth and Nabney suggest fixing the mean and variance on a grid defined in the output space and training only the model that outputs the mixing coefficients to reduce the NLL. If there are enough components regularly distributed in the output space, fixed means and variances do not reduce the representation capability. However, for large-dimensional outputs, such as MP parameters, this method leads to a sizeable intractable grid.

In [19], Makansi et al. used an MDN to predict the distribution of future car positions based on a current car position. To avoid mode collapse, they separated the MDN into two parts: a sampling and inference network. The sampling network takes the current car position as input and outputs a fixed number of hypotheses for future car positions. The authors trained the sampling network to place hypotheses to cover all of the observed outputs diversely. Based on these hypotheses, an inference network infers the parameters of the GMM. The MDN is a combination of the sampling and inference networks. The proposed method avoids mode collapse for the car-position prediction. However, for a high-dimensional output, such as MP parameters, the sampling network requires a large output dimension. Hence, it is difficult to apply both methods to our problem.

Entropy Costs for the MDN

Before introducing the entropy cost function, we first inspect the reasons that mode and model collapses occur when learning an MDN from demonstrations. Mode collapse occurs if the demonstrations associated with different modes for a task parameter query are imbalanced. For example, in Figure 1(a), only a small number of demonstrations take the path from the right side. By maximizing the likelihood of all demonstrations, the MDN tends to output a small mixing coefficient for the mixture component, which corresponds to the mode with fewer associated training data. In theory, it is correct to associate a small probability with an event that rarely happens in the observations. However, the reasons for the imbalance of the demonstrations in different modes, such as the habit of the demonstrator, can be meaningless for correct motion generation. It is often the case that we cannot collect enough demonstrations to cover all modes. Even if there are only a few demonstrations, where the human accomplishes the task with particular types of motions, the robot should learn these motions to increase motion diversity.

Model collapse occurs in particular when relatively few demonstrations are available. Several mixture components of the MDN, which are represented by neural networks, are powerful enough to overfit all demonstrations. After training of the MDN, instead of all K mixture components, it uses only a subset of them, which corresponds to the local minima of the NLL and results in poor performance of the MDN for some task parameter queries. As shown in Figure 1(b), one of the two models disappears with the original MDN, and the MDN performs similarly to the GPR. Compared to mode collapse, model collapse is more severe because it can lead to the failure of task execution.

To reduce the occurrence of mode and model collapses, we introduce the negative model entropy cost function as follows:

$$l_{\mathrm{model}}(\boldsymbol{\Theta}) = \sum_{k=1}^{K} p(m=k\,|\,\boldsymbol{D};\boldsymbol{\Theta}) \log p(m=k\,|\,\boldsymbol{D};\boldsymbol{\Theta}), \tag{7}$$

where

$$p(m=k\,|\,\boldsymbol{D};\boldsymbol{\Theta}) = \sum_{i=1}^{M} \pi_k(\boldsymbol{q}_i;\boldsymbol{\Theta})\, p(\boldsymbol{q}_i), \tag{8}$$

where m is the component index, and $p(\boldsymbol{q}_i) \propto M^{-1}$. By minimizing the cost, we increase the uncertainty of the model labels when considering all demonstrations $\boldsymbol{D}$. A high uncertainty of the model labels is equivalent to either equally distributed mixing coefficients for each task parameter query or equally distributed demonstrations to different models. In the former case, if all mixing coefficients for one specific task

parameter query are almost equal and close to $1/K$, each mode has the same probability of being selected to generate motions. Hence, the mode collapse does not occur. In the latter case, if each model is associated with some demonstrations, the corresponding mixture component, i.e., the network branch, is well trained. Hence, model collapse does not occur.

The objective function for training the entropy MDN is a weighted sum of the NLL and the entropy cost function: $w_{\text{NLL}} l_{\text{NLL}} + w_{\text{model}} l_{\text{model}}$. In the following experiments, the weights are empirically determined: $w_{\text{NLL}} = 1$, $w_{\text{model}} = 50$.

Improving the MDN With Failures

In many applications, the failed samples are easily collected with an underfitted MDN model. To reduce the occurrence of these failed MP parameters for similar task parameter queries, we introduce the failure cost function as follows:

$$l_{\text{neg}}(\boldsymbol{\Theta}) = \sum_{i=1}^{M^*} \log\left(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i; \boldsymbol{\Theta})\, \mathcal{N}(\boldsymbol{w}_i^*; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_{\text{neg}}) \right), \tag{9}$$

where the normal distribution has the same form as (6), but $\boldsymbol{\Sigma}_{\text{neg}} = \sigma_{\text{neg}} \boldsymbol{I}$. By minimizing this cost, the output mean vector $\boldsymbol{\mu}_i$ for a specific task parameter $\boldsymbol{q}_i$ is kept away from the failed MP parameters $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$.

If σ_{neg} is too small, the failure cost does not affect the results; on the other hand, a σ_{neg} that is too large can result in trajectories that are significantly different from demonstrations. Here, we determined σ_{neg} empirically with the smallest variance of all MP parameter components.

To train an MDN with the failure cost, we prepare an evaluation data set. After a certain number of training steps, we run the MDN on this evaluation data set and collect the failed samples in a failures data set $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$. In the next training steps, we calculate the failure cost function based on the failures data set. To avoid increasing the computational cost, we use a fixed data set size M^* and remove the earliest failed samples when new samples are collected.

To evaluate the MP generalization methods, we check whether the generated MPs accomplish the tasks with different task parameters. In learning from demonstrations, a successful task execution means that the generated motions are similar to the demonstrations and can accomplish the task with specific task parameters. In the proposed method, we meet this requirement by training the MDN with the NLL, which is related to the similarity between the collected and generated MP parameters. To check whether the trained model meets the latter requirement, we evaluate it with the success rate of task execution.

For task execution, we can only execute one motion after another. Hence, the MP parameter for the task must be determined based on the MDN output distribution in a subsequent step.

Generating Motion With the MDN

The purpose is to generate single motions for some task parameter queries. In the following experiments, we consider two strategies: selecting the most probable mode or choosing the best one from multiple samples. The most probable mode is the output mean vector of the mixture component that has the most significant mixing coefficient. If the Gaussian components of the output GMM have separated means, the most probable mode corresponds to the mode of the GMM.

For one specific task parameter query $\boldsymbol{q}$, the MDN outputs K Gaussian mixture components with their mixing coefficients $\{\pi_k\}_{k=1}^{K}$. The K modes of these Gaussian mixture components correspond to the K most probable motions of different types. However, not all of these K modes can accomplish the task. The most significant mixing coefficient indicates the mode that most likely succeeds. With this strategy, the MDN serves as a deterministic model; hence, we can compare it with previous deterministic methods. Selecting the most probable mode is the simplest way to generate motion from the MDN output. Moreover, this strategy works quite well in many tasks.

However, with the most probable mode, we ignore the information provided by the output variance $\boldsymbol{\Sigma}$ of the MDN. Each of its diagonal elements indicates how various the generated trajectories can be at the corresponding time for successful task execution. When we draw samples from the output GMM for a specific task parameter query, the variance matrix ensures that the samples have a high probability of success. In some tasks, the most probable mode does not work very well, such as in the experiment described in the "The Hit-the-Ball Experiment in MuJoCo" section. To improve the performance, we draw several MP parameters from the output distribution and execute one after another for the task until success. In this case, the success rate is also dependent on the number of samples.

Moreover, with the former strategy, the MDN always generates the same motion for one specific task parameter query. To demonstrate motion diversity and the fact that the MDN learns multiple modes, we also must draw multiple samples from the output distribution (see the "The Throw-the-Ball Experiment" section).

In the next section, we evaluate the proposed method with four experiments. In the first two investigations, we select the most probable mode and compare our method with previous deterministic methods. In the other two, we draw multiple samples from the MDN output to either improve the performance or show the motion diversity.

Experiments and Evaluations

Fitting Polynomials

In this experiment, we consider a fifth-order polynomial $y(x) = \sum_{k=1}^{5} a_k x^k$. The purpose is to learn a mapping from the coefficients a_k to the MP parameter $\boldsymbol{w}$ in (2). The error is the distance between the true fifth-order polynomials and the generated trajectories by the output VMP parameters. We evaluate different methods separately for the inputs with one dimension a_5 to all dimensions $(a_5, a_4, \ldots, a_0)^T$. Figure 4 shows the results for 60 experiments. In each experiment, 30 random coefficients are for training, and 20 are for testing.

The one-step approaches presented in [11] with SVR and GPR are denoted as "–1," while two-step methods are indicated with "–2." Notice that the dot-product kernel is the perfect assumption for this task because the polynomial value is, indeed, a dot product of the coefficients and bases. However, GPR-1 with dot-product kernels is worse than other methods when learning the mapping from $(a_5, a_4)^T$ to $\boldsymbol{w}$; this results because the time-dependent variable x is a part of the input, which loses the advantage of the correct dot-product assumption. In contrast, the two-step method GPR-2 with dot-product kernels perfectly reproduces the polynomial. On the other hand, however, SVR-1 performs better than SVR-2.

For the MDN, we select the most probable VMP parameter as the output. Except for the GPR with dot-product kernels, the MDN outperforms all other methods.

Random-Obstacles Avoidance

To show whether the methods can scale to a more complex task than the one shown in Figure 1, we randomly placed three obstacles in 2D space and asked for the collision-free trajectories with random starts and goals. To collect demonstrations, a person drew curves connecting random starts and goals without collisions with three randomly generated 2D balls on a tablet. Without any instructions, the human demonstrations show multiple modes and models.

The success of the task execution requires that the generated trajectory connects the start and goal without any collision with randomly placed obstacles. Due to the task complexity and existence of multiple modes and models, previous approaches cannot achieve acceptable results. With a data set with 100 demonstrations, the TP-GMM has a success rate of only approximately 45% with five local frames (three for the obstacles and two for the start and goal). Both one-step and two-step methods with SVR and GPR perform worse, with a success rate of less than 30%.

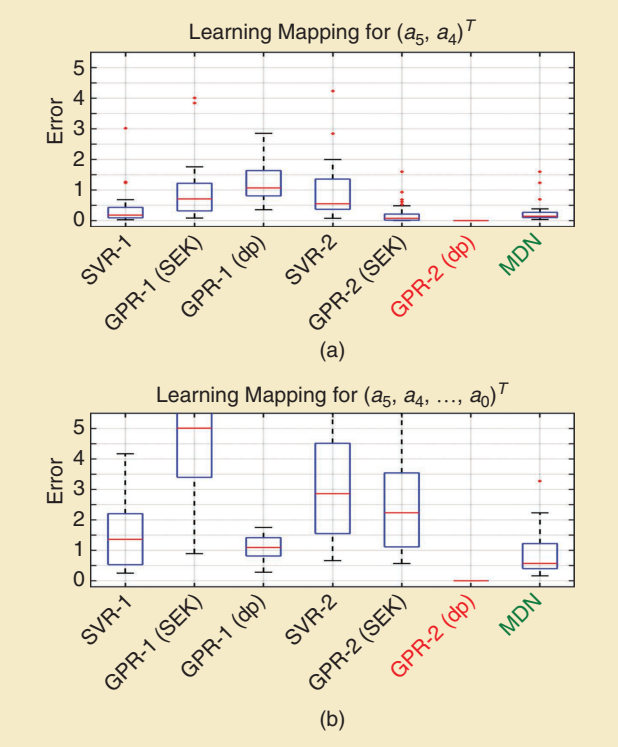


Figure 4. The models learned to fit a fifth-order polynomial, showing the results for learning a mapping from (a) a part of the coefficients to the MP parameters and (b) all coefficients to the MP parameters. dp: dot-product kernel.

For the MDN, we assume three mixture components: $K = 3$. To extract the latent feature (see the blue box in Figure 3), we introduce three separate network branches for three obstacles. Each branch takes the position of one obstacle and the start and goal of the trajectory as the input and outputs a hidden feature vector. The three hidden feature vectors are then concatenated for the rest of the MDN. The entire MDN is trained in an end-to-end manner.

During testing, we select the most probable mode, as mentioned previously. For each number of training data, 30 experiments are conducted for 30 different training data sets randomly chosen from the collected demonstrations. To utilize the failure cost function, after each 100 training steps, the MDN is evaluated on an evaluation data set to produce failed samples. To avoid increasing data, we consider only the most recent 3,000 failed samples.

As shown in Figure 5, with 100 demonstrations, the MDN with both the entropy and failure cost functions ($l_{\text{NLL}} + l_{\text{model}} + l_{\text{neg}}$) achieves a success rate of approximately 82%. The performance is improved further to 85% with 300 demonstrations. With 100 demonstrations, the entropy MDN with the failure cost function ($l_{\text{NLL}} + l_{\text{model}} + l_{\text{neg}}$) achieves the best result, and the entropy MDN ($l_{\text{NLL}} + l_{\text{model}}$) is better than the original MDN (l_{NLL}). Their difference decreases with an increasing number of demonstrations.

In Figure 6 [16], testing samples are shown. The colorized solid curves are generated by the most probable mode (MP parameter) given by the MDN, and the transparent dashed curves correspond to the other two modes, which are not selected by the MDN, with relatively small output mixing coefficients. Three different colors indicate three different modes: the green curves bend toward the top, red curves bend toward the bottom, and blue curves connect the start and goal directly. As can be seen, the MDN accomplishes the task with two steps. One step is to separately update each mixture component branch to increase the success rate of their modes. The other step is to adjust the mixing-coefficient output to select the mode that has the highest chance of accomplishing the task.

The Hit-the-Ball Experiment in MuJoCo

In this experiment, the robot hits the ball with its fist. After being hit, the ball slides on the table and stops at some locations. The final location of the ball on the table is the task parameter query. The purpose is to generate an appropriate robot motion to hit the ball and let the ball stop at a specific location. We conducted this experiment in the MuJoCo simulator [20] with the model of the humanoid robot ARMAR-6 [21].

For the demonstrations, we used a random trajectory generator based on a fifth-order polynomial with which the position and velocity at the end of the trajectory can be specified. The initial ball location on the table is fixed. The end velocities of the trajectories are randomly sampled from a uniform distribution. With different hitting velocities, the ball stops at different locations. We collected the final locations of the ball $\boldsymbol{q}$ and MP parameters $\boldsymbol{w}$ in a training data set.

As shown in Figure 7, the ball can bounce off the borders of the table, which realizes multiple modes for one specific target location in the collected demonstrations. The table measures 260×200 cm, and the ball has a radius of 5 cm. Successful task execution means that the ball is no more than 10 cm from the target location.

We use $K = 3$ mixture components and train the MDN with 50 random demonstrations and test it on 100 new ball locations. When selecting the most probable mode, we get an average success rate of approximately 15%. This poor performance might be because the task requires an accurate trajectory. With a randomly small perturbation of the MP parameters that correspond to the original 50 demonstrations, the success rate can drop rapidly. As mentioned previously to improve the task performance, we draw several samples from the output distribution. In this case, a successful MP generalization means that there exists at least one of these samples that leads to

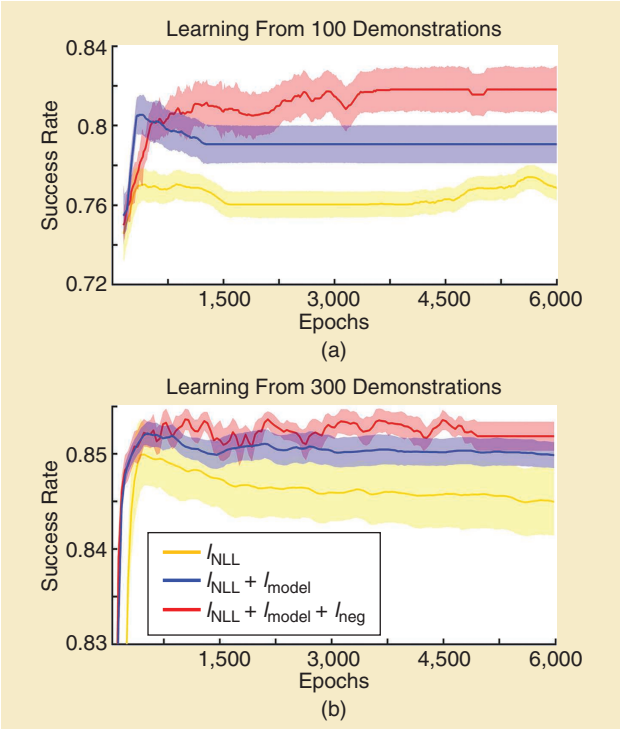


Figure 5. A comparison between the original and entropy MDNs, showing the results with (a) 100 and (b) 300 demonstrations.

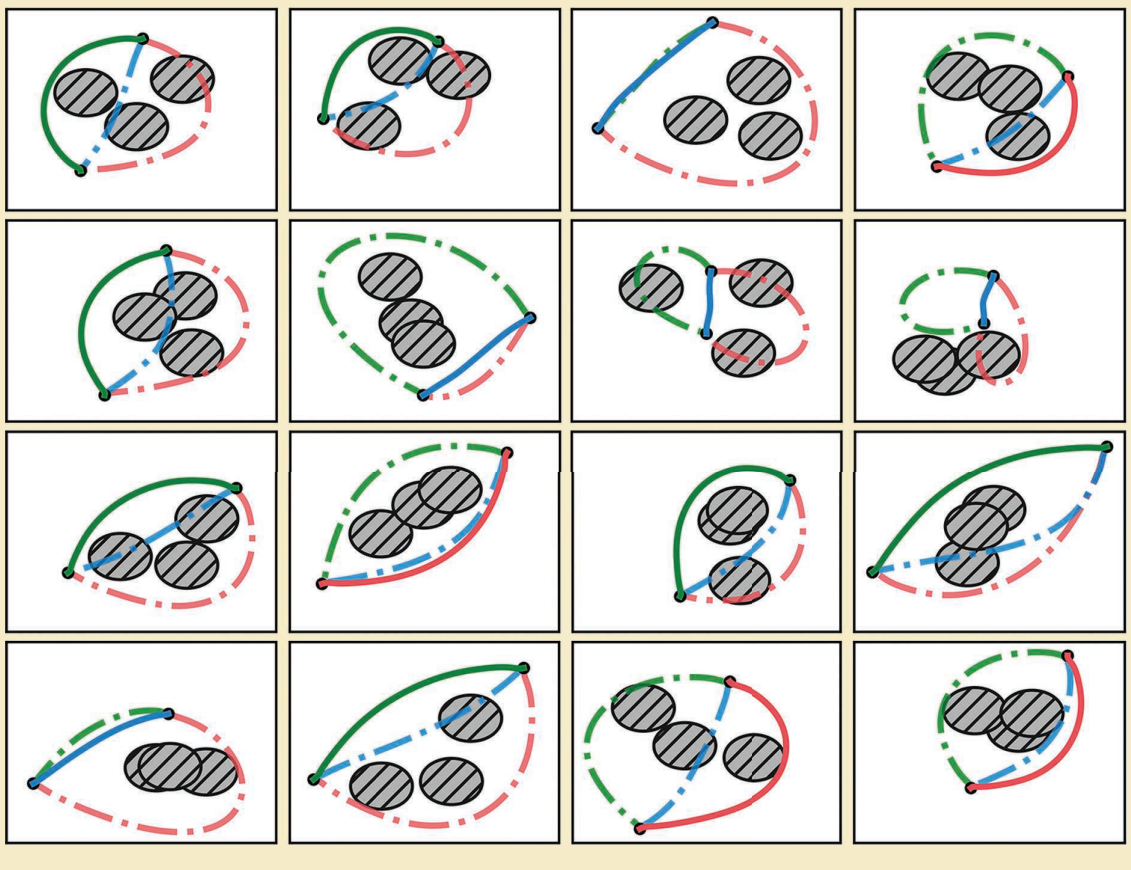


Figure 6. Three obstacles are randomly placed in the 2D space. The solid curves correspond to the most probable mode, and the dashed curves are generated by the two other modes that are not selected by the MDN.

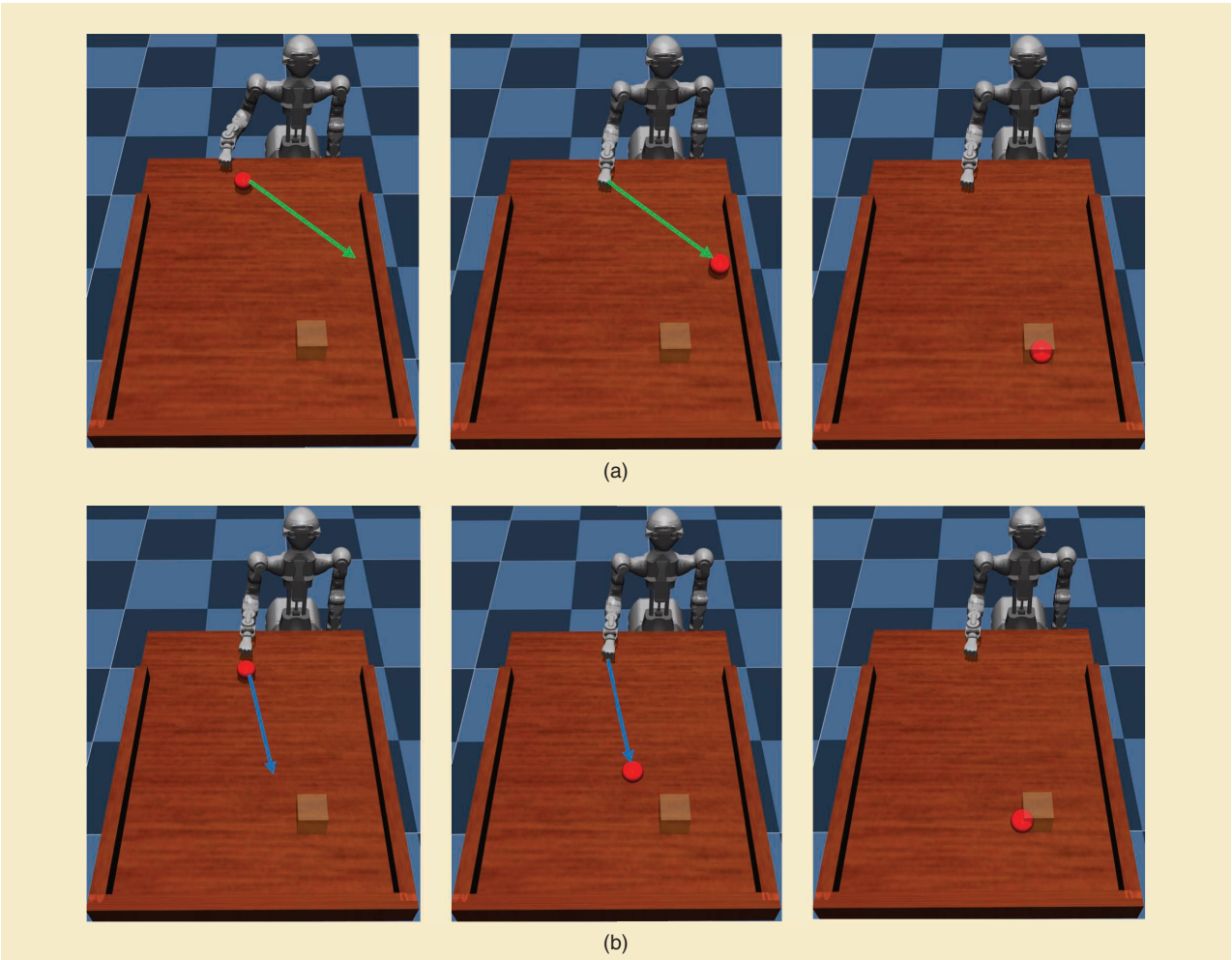


Figure 7. In the hit-the-ball experiment, the desired final ball location is the input of the MDN, which is denoted by a transparent box. (a) The robot hits the ball from its right side, and the ball bounces off the border and stops at the target. (b) The robot hits the ball directly toward the target.

successful task execution. As shown in Figure 8, increasing the number of samples improves the success rate.

In this specific task, the number of samples helps for two reasons. One trivial reason is the setup of the task, which allows successful task executions by chance: the VMP guarantees that the robot hits the ball, and the table borders limit the final ball locations. The other reason is that the MDN learns the correct distribution, which gives a high probability to the correct MP parameter, which is, unfortunately, not precisely the mode. Sampling from the correct distribution has a greater chance of finding the correct solution than directly selecting the most probable mode.

To prove that the latter exists with the MDN for this task, we consider a uniform distribution of the MP parameters as the baseline, whose interval is determined by the minimal and maximal components of the MP parameters, which correspond to the 50 demonstrations. In addition to the baseline, we construct Gaussian distributions by considering the GPR and SVR outputs as mean vectors and with a fixed-variance matrix ($\Sigma = 0.01\boldsymbol{I}$).

As shown in Figure 8, the MDN outperforms the others for all sample numbers. For the baseline, it coincides with the intuition that its success rate is almost proportional to the sample number because it does not learn from the demonstrations. GPR and SVR have better performances than the baseline because they draw the samples close to their output MP parameters. However, the samples drawn around their outputs are totally by chance because of the fixed-variance matrix. In contrast, the MDN learns a relatively correct distribution output. Hence, it already achieves a high success rate with a smaller sample number.

The Throw-the-Ball Experiment

To further evaluate our methods, we let the humanoid robot ARMAR-6 throw a ball at a specific target. The arms of ARMAR-6 have eight degrees of freedom (DoF) each. To simplify the task, we used only four of them without loss of generality (see the 4 DoF in Figure 9). The demonstrations were conducted by a human using kinesthetic teaching. After learning the corresponding MP parameter for each

demonstration, we speed up the motion to 1 s and set a fixed joint goal. The robot hand always opens at 0.55 s. Then, we record the location of the ball when it drops to the ground. By fixing the goals and speed of the motions, these hit-ground locations are dependent only on the shapes of the joint trajectories. In the experiment, we let the robot face the wall, and it can bounce the ball off the wall to the target.

We let the robot throw 50 times with different human demonstrations and randomly split the collected data into 30 for training and 20 for testing. We train an MDN ($K = 2$) on 30 demonstrations. For the testing, we use only the hit-ground locations of the other 20 demonstrations as task parameter queries, which guarantees that all of the hit-ground locations are reachable. During the testing, we place a plate on the ground to indicate the current query. Successful task execution is to throw the ball onto the plate either directly or by bouncing it off the wall. In the experiment, with 10 samples, the robot missed only two out of 20 target hit-ground locations. In Figure 9, for one specific task parameter query, we show how two of 10 MP parameters, which correspond to two different modes, result in different paths of the ball.

Discussion and Conclusions

This work addressed the problem of MP generalization to different tasks and was concerned with two aspects. First, to take the multiple modes and models of human demonstrations into account, we propose using an MDN for the mapping from the task parameter query to the MP parameter distribution. The experiments show that the MDN-based approach outperforms techniques used in previous works. Second, to further reduce the occurrence of mode and model collapse during training of the MDN, we propose the entropy cost function. Moreover, for some tasks, we introduce the failure cost to improve the performance of the MDN further. The comparison of different MDNs shows that the new cost functions perform better than the original one, especially when the set of demonstrations is relatively small.

What we did not consider here is the extrapolation of the method to areas outside the demonstration range. Since the MDN is learned fully from demonstrations, its extrapolation capability is limited. Current methods dealing with the extrapolation problem focus only on a specific set of task parameters, such as the TP-GMM and via-points adaptation of the VMP, described in [5] or our previous work [15]. The extrapolation of MP generalization to arbitrary task parameter queries is still unsolved.

Recent approaches, such as those in [13] and [14], also take task-relevant sensory inputs into account and learn them together with the robot motions. For human–robot

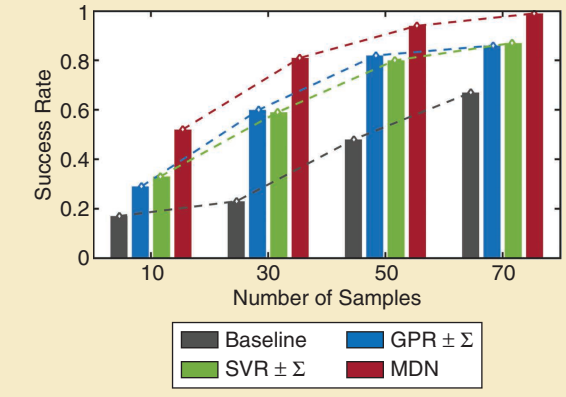


Figure 8. The results of the hit-the-ball experiment show that the MDN (red) outperforms the baseline (gray), GPR ± Σ (blue), and SVR ± Σ (green).

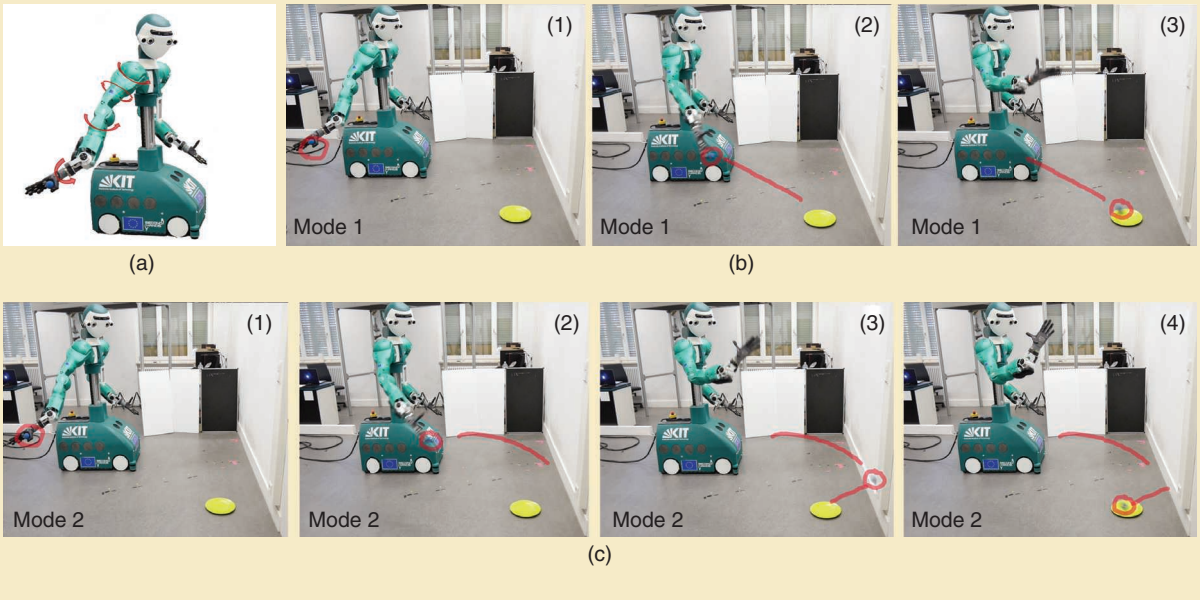


Figure 9. (a) There are 4 DoF used for throwing the ball. (b) The robot throws the ball directly to the target. (c) The robot bounces the ball to the target off the wall.

interactions, the robot motion is generated based on the observed human activity. With the human activity considered as a task parameter query, these methods also learn a mapping from the task to MP parameters. However, unlike the MDN, which directly learns this mapping, these approaches learn a generative model. In the future, we will explore the use of our proposed method for human–robot interaction tasks and compare these methods.

For the sampling strategy, selecting the most probable mode already solves many tasks. However, for some other tasks, the suggested method needs multiple samples to achieve better performance, such as that described in the "The Hit-the-Ball Experiment in MuJoCo" section. This fact requires multiple robot trials for each task parameter query. To solve this problem, we consider either using reinforcement learning to refine the mean vector given by the trained MDN with a small number of trials, as in [22], or training a discriminator that can predict success or failure based on both task and MP parameters.

Acknowledgments

The research leading to these results received funding from the German Federal Ministry of Education and Research under the project Organic Machine Learning (01IS18040A) and the European Union Horizon 2020 Research and Innovation program under grant agreement 643950 (SecondHands). We would like to thank Jonas Rothfuss for the fruitful discussions.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Robot Programming by Demonstration*. New York: Springer-Verlag, 2008, pp. 1371–1394.

[2] S. F. Giszter, F. A. Mussa-Ivaldi, and E. Bizzi, "Convergent force fields organized in the frog's spinal cord," *J. Neurosci., Official J. Soc. Neurosci.*, vol. 13, no. 2, pp. 467–491, 1993. doi: 10.1523/JNEUROSCI.13-02-00467.1993.

[3] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, Feb. 2013. doi: 10.1162/NECO_a_00393.

[4] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proc. Advances Neural Information Processing Systems 26*, 2013, pp. 2616–2624.

[5] S. Calinon, T. Alizadeh, and D. G. Caldwell, "On improving the extrapolation capability of task-parameterized movement models," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Nov. 2013, pp. 610–616. doi: 10.1109/IROS.2013.6696414.

[6] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Trans. Robot.*, vol. 26, no. 5, pp. 800–815, Oct. 2010. doi: 10.1109/TRO.2010.2065430.

[7] Y. Zhou and T. Asfour, "Task-oriented generalization of dynamic movement primitive," in *Proc. 2017 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2017, pp. 3202–3209. doi: 10.1109/IROS.2017.8206153.

[8] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robot. Auton. Syst.*, vol. 60, no. 10, pp. 1327–1339, 10 2012. doi: 10.1016/j.robot.2012.05.004.

[9] R. Pahic, A. Gams, A. Ude, and J. Morimoto, "Deep encoder-decoder networks for mapping raw images to dynamic movement primitives," in *Proc. 2018 IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 1–6. doi: 10.1109/ICRA.2018.8460954.

[10] B. da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," in *Proc. 29th Int. Conf. Machine Learning*, June 2012, pp. 1443–1450. doi: 10.5555/3042573.3042758.

[11] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, Oct. 2013, pp. 417–422. doi: 10.1109/HUMANOIDS.2013.7030008.

[12] A. Pervez and D. Lee, "Learning task-parameterized dynamic movement primitives using mixture of GMMs," *Intell. Service Robot.*, vol. 11, no. 1, pp. 61–78, Jan. 2018. doi: 10.1007/s11370-017-0235-8.

[13] E. Pignat and S. Calinon, "Learning adaptive dressing assistance from human demonstration," *Robot. Auton. Syst.*, vol. 93, no. C, pp. 61–75, July 2017. doi: 10.1016/j.robot.2017.03.017.

[14] M. Ewerton, G. Neumann, R. Lioutikov, H. Ben Amor, J. Peters, and G. Maeda, "Learning multiple collaborative tasks with a mixture of interaction primitives," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2015, pp. 1535–1542. doi: 10.1109/ICRA.2015.7139393.

[15] Y. Zhou, J. Gao, and T. Asfour, "Learning via-point movement primitives with inter- and extrapolation capabilities," in *Proc 2019 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 4301–4308. doi: 10.1109/IROS40897.2019.8968586.

[16] C. M. Bishop, "Mixture density networks," Dept. of Computer Science and Applied Mathematics, Aston Univ., Birmingham, NCRG/94/004, 1994.

[17] G. Mclachlan and K. Basford, *Mixture Models: Inference and Applications to Clustering* (Statistics, Textbooks and Monographs Series 84). New York: Marcel Dekker, 1988.

[18] L. U. Hjorth and I. T. Nabney, "Regularisation of mixture density networks," in *Proc. 9th Int. Conf. Artificial Neural Networks ICANN 99*, Sept. 1999, vol. 2, pp. 521–526. doi: 10.1049/cp:19991162.

[19] O. Makansi, E. Ilg, Ö. Çiçek, and T. Brox, "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7137–7146. doi: 10.1109/CVPR.2019.00731.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Proc 2012 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 5026–5033. doi: 10.1109/IROS.2012.6386109.

[21] T. Asfour et al., "Armar-6: A collaborative humanoid robot for industrial environments," in *Proc 2018 IEEE/RAS Int. Conf. Humanoid Robots (Humanoids)*, pp. 447–454. doi: 10.1109/HUMANOIDS.2018.8624966.

[22] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Learning Motor Skills: From Algorithms Robot Experiments*. Cham, Switzerland: Springer-Verlag, 2014, pp. 9–67.

You Zhou, Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Germany. E-mail: you.zhou@kit.edu.

Jianfeng Gao, Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Germany. E-mail: jianfeng.gao@kit.edu.

Tamim Asfour, Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Germany. E-mail: asfour@kit.edu.