Continual learning from demonstration of robotics skills

Sayantan Auddy [a,*], Jakob Hollenstein [a], Matteo Saveriano [b], Antonio Rodríguez-Sánchez [a], Justus Piater [a,c]

[a] *Department of Computer Science, University of Innsbruck, Austria*
[b] *Department of Industrial Engineering, University of Trento, Italy*
[c] *Digital Science Center (DiSC), University of Innsbruck, Austria*

ARTICLE INFO

ABSTRACT

Methods for teaching motion skills to robots focus on training for a single skill at a time. Robots capable of learning from demonstration can considerably benefit from the added ability to learn new movement skills without forgetting what was learned in the past. To this end, we propose an approach for continual learning from demonstration using hypernetworks and neural ordinary differential equation solvers. We empirically demonstrate the effectiveness of this approach in remembering long sequences of trajectory learning tasks without the need to store any data from past demonstrations. Our results show that hypernetworks outperform other state-of-the-art continual learning approaches for learning from demonstration. In our experiments, we use the popular LASA benchmark, and two new datasets of kinesthetic demonstrations collected with a real robot that we introduce in this paper called the *HelloWorld* and *RoboTasks* datasets. We evaluate our approach on a physical robot and demonstrate its effectiveness in learning real-world robotic tasks involving changing positions as well as orientations. We report both trajectory error metrics and continual learning metrics, and we propose two new continual learning metrics. Our code, along with the newly collected datasets, is available at https://github.com/sayantanauddy/clfd.

1. Introduction

Robots deployed in unstructured real-world environments will face new tasks and challenges over time, requiring capabilities that cannot be fully anticipated at the beginning. These robots need to learn continually, which implies that they should be able to acquire new capabilities without forgetting the previously learned ones. Furthermore, a continual learning robot should be able to do this without the need to store and retrain on the training data of all the previously learned skills.

Continual learning can be effective in expanding a robot's repertoire of skills and in increasing the ease of use for non-expert human users. However, apart from a few approaches for robotics [1,2], the current continual learning research mostly focuses on vision-based tasks such as incrementally learning classification of new image categories [3–5].

Continually acquiring perceptive skills is important for a robot that interacts with its environment, but equally important is the ability to incrementally learn new movement skills. Learning from demonstration (LfD) [6] is a popular and tangible way to impart motion skills to robots, for instance via kinesthetic

teaching, where a human user teaches new skills by guiding the robot. A trend that recently received increased attention in LfD is encoding observations into a vector field [7–12]. These methods, like many other works in the field, focus on learning a single motion. To naively learn multiple motion skills, one would need to train a different model for each skill, or jointly train on the demonstrations for all skills.

In this paper, we propose an approach to continual learning from demonstration in which a robot learns individual motions sequentially *without retraining* on past demonstrations. The motion demonstrations are recorded in the robot's task space and consist of either trajectories of the end-effector position, or trajectories of the end-effector position and orientation (the robot is free to rotate in all 3 rotation axes). The skills learned from demonstrations of different tasks are incorporated into a common unified model, after which our robot can reproduce all the trajectories it has learned in the past (Fig. 1). To the best of our knowledge, this is the first work on continual trajectory learning from demonstrations.

More specifically, we show that a Hypernetwork [5,13], that generates the parameters of a Neural Ordinary Differential Equation (NODE) solver [14], remembers a long sequence of motion skills equally well as when learning each task with a separate NODE. The number of parameters of the hypernetwork grows

* Corresponding author.
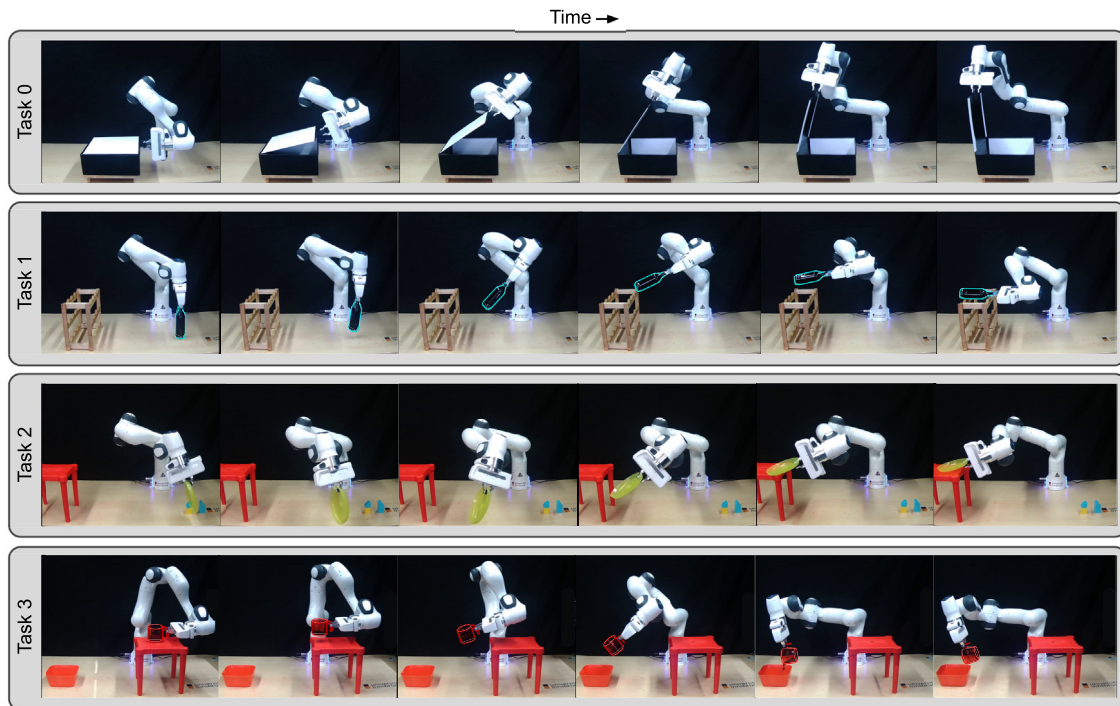   *E-mail address:* sayantan.auddy@uibk.ac.at (S. Auddy).

Fig. 1. A robot, continually trained using learning from demonstration to perform real-world tasks involving changing positions and orientations of the end-effector, can reproduce all the tasks that it has learned in the past with negligible parameter growth and without retraining on past tasks. The order of tasks during sequential training is 0: *box opening*, 1: *bottle shelving*, 2: *plate stacking*, 3: *pouring*. Video of the robot performing these tasks is available at https://youtu.be/0gdIImIBnXc. Further details can be found in Section 5.3.3.

by a negligible amount for each new task, making it suitable for potential deployment on resource-constrained, non-networked robotic platforms. We also demonstrate the effectiveness of chunked hypernetworks [5] which can be even smaller in size than the NODEs they generate. Our results show how using a time index as an additional, direct input to a NODE increases its prediction accuracy for complex trajectories. We first evaluate our approach on the popular LASA trajectory learning benchmark [8], where our model learns a long sequence of 26 tasks without forgetting. Due to the lack of datasets containing real robot data useful for continual learning from demonstration, we introduce two new datasets, named *HelloWorld* and *RoboTasks*. The HelloWorld dataset consists of two-dimensional trajectories of the robot's position collected with a Franka Emika Panda robot. RoboTasks is a dataset of real-world robot tasks containing trajectories of the robot's position as well as its orientation in 3D space (Fig. 1). These two datasets serve as additional benchmarks to evaluate our approach, both quantitatively and qualitatively on a real robot. For all three datasets, we extensively compare our hypernetwork-based continual LfD approach against methods from all major continual learning families (*dynamic architecture*, *replay*, and *regularization* [15]) and report multiple metrics, both for accuracy of the reproduced motion skills, as well as the continual learning performance. Finally, we introduce two new easily-computable metrics that offer additional insights into the continual learning performance: *Time Efficiency* (TE) and *Final Model Size* (FS). TE captures training time changes while learning multiple tasks, and FS measures the relative model size after learning all tasks.

In summary, our primary contributions are:

- We propose an approach to learning from demonstration with hypernetworks and NODEs for continually learning new tasks without reusing training data of previous tasks. We show that this approach can be used for learning robot tasks in the real world.

- We release 2 new datasets suitable for continual LfD: a dataset containing 7 tasks of planar motions, and a dataset of 4 tasks of motions involving both position and orientation. Both datasets are collected with a real robot using kinesthetic teaching.

2. Related work

Robot *learning from demonstration* (LfD) is a means for humans to teach motion skills to robots without explicitly programming them [6], which allows even users without expertise in robotics to train robots. It is also known by other names such as *programming by demonstration* or *imitation learning* [6,16]. The demonstrations required for training robots via LfD can be obtained by different means, some of which are: (i) using a motion-tracking system to record human motions, (ii) using teleoperation to operate the robot while recording the robot's state, or (iii) using kinesthetic teaching where a human user physically guides the robot to perform a motion task [6,17–19]. Once the demonstrations are available, there exist several different algorithmic approaches for learning from this data [6]. Supervised learning has been used to learn from either a single demonstration [20] or a collection of demonstrations [21]. LfD has also been used in conjunction with reinforcement learning (RL) [22] where RL is used to refine the skills learned with LfD. Another approach is to learn a cost function from demonstrations and then to train a model predictive controller to reproduce the skills through inverse RL [23] or through constrained optimization [24]. In addition to demonstrations which show the robot the motion it has to perform, negative demonstrations have also been shown to be advantageous [25]. We refer the reader to [6,17,18] for a comprehensive overview of LfD.

In this paper, we focus on a subfield of LfD: *trajectory-based* learning methods that use a supervised approach for acquiring motion skills. These methods can be broadly categorized into two

groups [19]: some methods use *generative models* to fit a distribution from the training data [7–9], while other methods exploit *function approximators* like neural networks to fit the training data [10,11]. In both groups, training data can be used to learn a *static mapping* (time input → desired position) or a *dynamic mapping* (input position → desired velocity). A dynamic mapping generates vector fields where input quantities are transformed into their time derivatives, and different strategies have been proposed to ensure convergence of the vector field to a given target [9,10,26,27]. Among others, the *Imitation Flow* (iFlow) approach of Urain et al. [9] leverages the representational power of neural networks and normalizing flows to learn vector fields from demonstrations. Another neural network based approach for learning vector fields is *Neural Ordinary Differential Equation solvers* (NODEs) [14]. Although NODEs have not been exploited for LfD, in our experiments we found that their empirical performance is comparable to that of iFlow. Crucially, the time needed to train a NODE to convergence is significantly less than that required for an iFlow model with an equivalent parameter size.

Trajectory-based LfD is a mature research field, but most methods assume that different tasks are encoded in different representations, i.e., one has to fit a new model for each task the robot has to execute [6]. In this paper, we take the continual learning perspective on learning by demonstration and propose an approach capable of continuously learning new tasks without needing to store and use the training data from past tasks.

Continual Learning approaches in the current literature mostly address the problem of continual image classification. Popular strategies include growing the network architecture [28], replaying data from past tasks, or regularizing trainable parameters to avoid catastrophic forgetting [15]. *Replay*-based methods cache samples of real data from past tasks [29], or use generative models to create pseudo-samples of past data [3], which are interleaved with the current task's data during training. *Regularization*-based methods [4,30] add a regularization term to the learning objective to minimize changes to parameters important for solving previous tasks. Refer to [15,31] for in-depth surveys on continual learning methods.

Continual learning has also been successfully applied to robotics, though the number of such studies is relatively few. Gao et al. [1] present an approach for continual imitation learning that relies on deep generative replay [3] and action-conditioned video prediction to generate state and action trajectories of past tasks. This pseudo-data is interleaved with demonstrations of the current task to train a policy network that controls the robot's actions. The authors note that the generation of high-quality video frames can be problematic for a long sequence of tasks.

Xie and Finn [32] propose a method for lifelong robotic reinforcement learning that seeks to improve the forward transfer performance while learning a new current task by pre-training on the entire experience collected from all previous tasks. The problem of catastrophic forgetting is not considered.

Our continual LfD approach shares some similarities with the work of Huang et al. [2] who continually learn a dynamics model for reinforcement learning. In their work, a task-conditioned hypernetwork generates the parameters of the dynamics model for reinforcement learning tasks such as opening doors or pushing blocks. In contrast, we use hypernetworks for generating parameters for a trajectory learning NODE in a setup for learning from demonstration. We follow a supervised approach and do not need to rely on robot simulators. Compared to [2], we evaluate on much longer sequences of tasks and also investigate the effectiveness of chunked hypernetworks [5]. In addition, we qualitatively evaluate our approach on a physical robot.

LfD involves learning the entire vector field of a robot's motion from only a handful of demonstration trajectories. This makes it a more challenging problem than typical supervised regression or classification scenarios where the amount of training data is usually much greater. To continually learn such vector fields of different kinds of motion demonstrations, we need to tackle the challenges of LfD as well as the problem of *catastrophic forgetting* associated with neural networks. As far as we know, ours is the first work that demonstrates that continual LfD is feasible for sequences of real-world robot tasks.

3. Background

In this paper, we utilize Neural Ordinary Differential Equation (NODE) solvers [14] to learn from demonstrations and different state-of-the-art continual learning approaches [4,5,30] to alleviate catastrophic forgetting [15] when the NODEs learn a sequence of multiple tasks.

3.1. Learning from demonstration

In the simplest case, task space demonstrations provided to a robot can consist of trajectories of only the end-effector positions. Since these position trajectories reside in Euclidean space, we can directly use a NODE to learn them using maximum likelihood estimation. However, the more general situation is when demonstrations contain both the position and orientation (which are commonly expressed in terms of unit quaternions). Trajectories of unit quaternions do not reside in Euclidean space and require additional processing, as discussed in Section 3.1.2.

3.1.1. Neural ODE solver

Consider a set of N observed trajectories $\mathcal{D} = \{\mathbf{y}_{0:T-1}^{(0)}, \ldots, \mathbf{y}_{0:T-1}^{(N-1)}\}$, where each trajectory $\mathbf{y}_{0:T-1}^{(i)}$ is a sequence of T observations $\mathbf{y}_t^{(i)} \in \mathbb{R}^d$. Each observation $\mathbf{y}_t^{(i)}$ is a perturbation of an unknown true state $\mathbf{x}_t^{(i)}$ generated by an unknown underlying vector field $\mathbf{f}_{\text{true}}$ [33]:

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \mathbf{f}_{\text{true}}(\mathbf{x}_\tau) \, \mathrm{d}\tau, \tag{1}$$

where $\mathbf{x}_0$ is the true starting state of the trajectory. The goal of a Neural Ordinary Differential Equation (NODE) solver [14] is to learn a neural network $\mathbf{f}_\theta$ parameterized by θ that approximates the true underlying dynamics of the observed system such that $\mathbf{f}_\theta \approx \mathbf{f}_{\text{true}}$. As we do not have access to $\mathbf{f}_{\text{true}}$ but only to the noisy observed trajectories, we compute the loss $\mathcal{L}$ based on the difference of the forward simulated states of the NODE $\hat{\mathbf{y}}_t$ and the observations $\mathbf{y}_t$:

$$\mathcal{L} = \frac{1}{2} \sum_t \parallel \mathbf{y}_t - \hat{\mathbf{y}}_t \parallel_2^2 \tag{2}$$

where $\hat{\mathbf{y}}_t = \hat{\mathbf{y}}_0 + \int_0^t \mathbf{f}_\theta(\hat{\mathbf{y}}_\tau) \, \mathrm{d}\tau$

When the set of observed trajectories $\mathcal{D}$ consists of only positions, then the demonstrations can be learned directly with the help of Eq. (2).

3.1.2. Learning orientation trajectories

When demonstration trajectories of the robot's orientation are expressed as unit quaternions, it is not possible to directly utilize Eq. (2) due to the unit norm constraint [34]. Following Ude et al. [35] and Huang et al. [34], we project quaternions into the tangent space which we can consider a local Euclidean space. These transformed trajectories can then be learned with Eq. (2). For inference, the Euclidean trajectories predicted by the NODE are transformed back into quaternions and then passed to the robot.

Consider a set of N observed orientation trajectories $\mathcal{D}_q = \{\mathbf{q}_{0:T-1}^{(0)}, \ldots, \mathbf{q}_{0:T-1}^{(N-1)}\}$, where the ith trajectory $\mathbf{q}_{0:T-1}^{(i)}$ is a sequence of T quaternions $\mathbf{q}_t^{(i)} = \begin{bmatrix} v_t^{(i)} \\ \mathbf{u}_t^{(i)} \end{bmatrix}$, where $\mathbf{q}_t^{(i)} \in \mathbb{S}^3$, $v_t^{(i)} \in \mathbb{R}$ and $\mathbf{u}_t^{(i)} \in \mathbb{R}^3$. We convert $\mathbf{q}_t^{(i)}$ into a rotation vector $\mathbf{r}_t^{(i)} \in \mathbb{R}^3$ using

$$\mathbf{r_t^{(i)}} = \text{Log}(\bar{\mathbf{q}}_t^{(i)} * \mathbf{q}_{T-1}^{(i)}) \tag{3}$$

where $\bar{\mathbf{q}}$ indicates the conjugate of the quaternion $\mathbf{q}$ and the $*$ operator denotes quaternion multiplication, and $\text{Log}(\cdot) : \mathbb{S}^3 \mapsto \mathbb{R}^3$ is the *logarithmic map* [36], defined as

$$\text{Log}(\mathbf{q}) = \begin{cases} \arccos(v)\frac{\mathbf{u}}{\|\mathbf{u}\|} & \text{if } \|\mathbf{u}\| > 0, \\ [0, 0, 0]^T & \text{otherwise.} \end{cases} \tag{4}$$

Here, $\mathbf{q}_{T-1}^{(i)}$ is the final quaternion in the sequence $\mathbf{q}_{0:T-1}^{(i)}$ and $\bar{\mathbf{q}}_t^{(i)}$ is the conjugate of $\mathbf{q}_t^{(i)}$. By applying Eq. (3) on the trajectories in $\mathcal{D}_q$, we obtain a set of Euclidean trajectories $\mathcal{D}_r$, which can then be learned using Eq. (2). After learning is complete, the trajectories $\hat{\mathcal{D}}_r = \{\hat{\mathbf{r}}_{0:T-1}^{(0)}, \ldots, \hat{\mathbf{r}}_{0:T-1}^{(N-1)}\}$ predicted by the NODE can be converted into quaternion trajectories $\hat{\mathcal{D}}_q$ using

$$\hat{\mathbf{q}_t^{(i)}} = \mathbf{q}_{T-1}^{(i)} * \text{Exp}(\hat{\mathbf{r}}_t^{(i)}) \tag{5}$$

where $\text{Exp}(\cdot) : \mathbb{R}^3 \mapsto \mathbb{S}^3$ is the *exponential map* [36], defined as

$$\text{Exp}(\mathbf{r}) = \begin{cases} \left[\cos(\|\mathbf{r}\|), \sin(\|\mathbf{r}\|)\frac{\mathbf{r}^T}{\|\mathbf{r}\|}\right]^T & \text{if } \|\mathbf{r}\| > 0, \\ [1, 0, 0, 0]^T & \text{otherwise.} \end{cases} \tag{6}$$

Once we obtain $\hat{\mathcal{D}}_q$, it can be directly compared against the ground truth demonstrations $\mathcal{D}_q$. However, we assume that the input domain of $\text{Log}(\cdot)$ is restricted to $\mathbb{S}^3$ except for $[1, 0, 0, 0]^T$ and the input domain of $\text{Exp}(\zeta)$ is constrained to satisfy $\|\zeta\| < \pi$ [34].

3.2. Continual learning

Existing continual learning approaches can be broadly categorized into a few groups, the most prominent among which are methods based on *dynamic architectures*, methods based on *replaying* (or *pseudo-rehearsing* on) training data of past tasks, and methods based on *regularization* [15]. In this paper, we consider continual learning methods (CL methods) from all of these categories.

3.2.1. Dynamic architectures
Progressive Networks: One of the early continual learning approaches, proposed by Rusu et al. [28], involves the addition of a new network for each new task, while reusing feature-mapping knowledge from previous tasks through lateral layer-wise connections from the networks of previous tasks. Although this approach eliminates catastrophic forgetting by design, it does not scale well to a large number of tasks due to the unconstrained growth of parameters and it also has the problem of a gradual slowing down of inference due to the increasing number of lateral connections.

3.2.2. Replay
Some continual learning approaches are based on the idea of replaying the training data (or some part of the data, or a compressed version of the data) of previous tasks while learning a new task [29]. While learning task k, the most naive way to do this is to combine the data from older tasks $\{\mathcal{D}_0, \ldots, \mathcal{D}_{k-1}\}$ with the data of the current task $\mathcal{D}_k$ to get a combined dataset $\{\mathcal{D}_0, \ldots, \mathcal{D}_k\}$. Thus, the network has access to all the training data from every task. However, this approach is not scalable due to the linear storage requirements with the number of tasks.

3.2.3. Regularization
Synaptic Intelligence: Synaptic Intelligence (SI) [30] is a regularization-based continual learning approach. Each neural network parameter is assigned an importance measure based on its contribution to the change in the loss. The loss for the mth task is defined as:

$$\tilde{\mathcal{L}}^m = \mathcal{L}^m + c \sum_k \Omega_k^m \left(\theta_k^* - \theta_k\right)^2, \tag{7}$$

where c is the regularization constant which trades off between learning a new task and remembering previously learned tasks, θ_k^* denotes the value of the kth parameter before starting to learn the mth task, and θ_k is the current value of the kth parameter. The per-parameter regularization strength Ω_k^m [30] is given by

$$\Omega_k^m = \sum_{l < m} \frac{\omega_k^l}{(\Delta_k^l)^2 + \xi}, \tag{8}$$

where ω_k^l is the importance of parameter k for learning task l, Δ_k^l is the change in parameter k, and ξ is a damping constant.
Memory Aware Synapses: Memory Aware Synapses (MAS) [4] is also a regularization-based continual learning approach. The loss for the mth task for MAS has the same form as SI (Eq. (7)). MAS differs from SI in the way Ω_k^m is computed: the importance of a trainable parameter depends on the gradient of the squared L_2 norm of the network's output, i.e.,

$$\Omega_k^m = \frac{1}{N} \sum_{n=1}^N \|g_k(\mathbf{x}_n)\| = \frac{1}{N} \sum_{n=1}^N \left\| \frac{\partial L_2^2(\mathbf{f_\theta}(\mathbf{x}_n))}{\partial \theta_k} \right\|. \tag{9}$$

The above summation is performed over N input data points.
Hypernetworks: A hypernetwork [5,13] is a meta-model that generates the parameters of a target network that solves the task we are interested in. It uses a trainable task embedding vector as an input to generate the network parameters for a task. Though the parameters $\mathbf{h}$ of the hypernetwork $\mathbf{f_h}$ are regularized, the parameters θ^{m+1} produced by a hypernetwork for the $(m + 1)$th task can be arbitrarily far away in parameter space from the parameters θ^m produced for the previous mth task. Intuitively, this gives a hypernetwork more freedom to find good solutions for both the mth and $(m+1)$th tasks than other regularization-based approaches [4,30].

A two-step optimization process is used for training a hypernetwork [5]. First, a candidate change $\Delta\mathbf{h}$ for the hypernetwork parameters is computed which minimizes the task-specific loss $\mathcal{L}^m$ for the (current) mth task with respect to θ^m:

$$\mathcal{L}^m = \mathcal{L}^m(\theta^m, \mathbf{y}^m) \text{ where } \theta^m = \mathbf{f_h}(\mathbf{e}^m, \mathbf{h}) \tag{10}$$

Here $\mathbf{e}^m$ is the task embedding vector and $\mathbf{y}^m$ is the data for the mth task. Next, $\Delta\mathbf{h}$ is considered to be fixed and the actual change for the hypernetwork parameters $\mathbf{h}$ is learned [5] by minimizing the regularized loss $\tilde{\mathcal{L}}^m$ with respect to $\theta^m = \mathbf{f_h}(\mathbf{e}^m, \mathbf{h})$:

$$\tilde{\mathcal{L}}^m = \mathcal{L}^m(\theta^m, \mathbf{y}^m)$$
$$+ \frac{\beta}{m-1} \sum_{l=0}^{m-1} \left\| \mathbf{f_h}(\mathbf{e}^l, \mathbf{h}^*) - \mathbf{f_h}(\mathbf{e}^l, \mathbf{h} + \Delta\mathbf{h}) \right\|^2 \tag{11}$$

Here $\mathbf{h}^*$ denotes the hypernetwork parameters before learning the mth task, and β is a hyperparameter that controls the regularization strength. To calculate the second part of Eq. (11), the stored task embedding vectors $\{\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^l, \ldots, \mathbf{e}^{m-1}\}$ for all tasks before the mth task are used. In each learning step, the current task embedding vector $\mathbf{e}^m$ is also updated to minimize the task-specific loss $\mathcal{L}^m$ [5]. Note that the parameters of the target network θ^m for the mth task are simply the hypernetwork outputs and are not directly trainable.
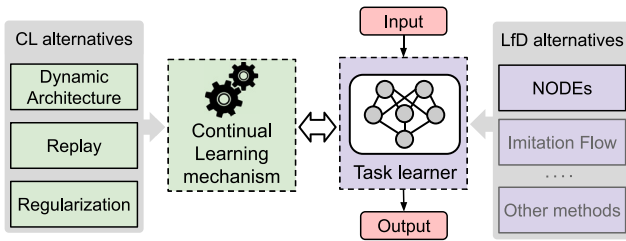
Fig. 2. Overview of a continual learning system. The task learner performs the task we are interested in (purple dashed box) and the continual learning mechanism (green dashed box) mitigates the effect of catastrophic forgetting. The continual learning (CL) mechanism can be one of many alternatives: dynamic architecture-based, replay-based, or regularization-based. For learning from demonstration (LfD), the task learner can be any trajectory learning approach, but we use NODEs for LfD (reasons are detailed in Section 4.1). Methods depicted with gray text are not considered for continual LfD in this paper. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Chunked hypernetworks [5] produce the parameters of the target network in segments known as *chunks*. A regular hypernetwork has a very high-dimensional output, but a chunked hypernetwork's output is of a much smaller dimension, leading to a lower hypernetwork parameter size. A chunked hypernetwork requires additional inputs in the form of trainable *chunk embedding vectors*. While each task has its dedicated task embedding vector, chunk embedding vectors are shared across tasks and are regularized in the same way as the hypernetwork parameters. The task embedding vector and all the chunk embedding vectors are combined in a batch and fed into the hypernetwork to produce the target network parameters for a task in one forward pass [5].

4. Methods

As depicted in Fig. 2, any continual learning system in general consists of 2 components: (i) a task learner, and (ii) a continual learning mechanism. The task learner performs the task we are interested in; e.g. for continual learning of image classification, the task learner can be a CNN classifier. In this paper the task that we are interested in is *learning from demonstration* (LfD), and the task learner in our case is a NODE as described in Section 3.1.1. The continual learning mechanism mitigates the effects of catastrophic forgetting as the task learner is trained on a sequence of tasks. This mechanism can act directly on the parameters of the task learner and can also contain other sub-components to achieve this goal. In this section, we describe in detail our proposed approach for continual learning from demonstration, as well as the other models used in our experiments. We also discuss the design choices made in this paper regarding the task learner and the continual learning mechanism.

4.1. Learning from demonstration

Along with the basic NODE $\mathbf{f_\theta}(\hat{\mathbf{y}}_t)$ described in Section 3.1.1, we use another variant where the NODE neural network $\mathbf{f_\theta}(\hat{\mathbf{y}}_t, t)$ is a function of both the state $\hat{\mathbf{y}}_t$ and the normalized time t of the state (t scales linearly from $t = 0.0$ for the starting state to $t = N/f$ for the Nth state of the trajectory, where f is the recording frequency). This explicit time input results in the NODE learning a time-evolving vector field. We show empirically that this improves the accuracy of predicted trajectories, especially for those containing loops. We refer to this time-dependent NODE as NODE^T, and to the time-independent one as NODE^I.

There exist many approaches for LfD [9,26], and in principle, any such method can be used as the *task learner* (in Fig. 2). In this paper we focus on neural network-based continual learning. Hence methods such as *Stable Estimator of Dynamical Systems* (SEDS) [26] are not considered as the constrained parameters (covariance matrices in case of SEDS) required by such methods cannot be produced by a neural network in a straightforward way. Alternative options for the task learner can be deep learning methods such as *Imitation Flow* (iFlow) [9] or NODEs [14]. We performed an experimental comparison between NODEs and iFLow (details in Appendix A.5) and found that the empirical performance of NODEs is better than iFlow and their training converges much quicker. Consequently, we use NODEs as the task learner in all the models used in our experiments.

4.2. Baselines and continual learning models

We enable continual learning for NODEs by adapting several state-of-the-art continual learning approaches for LfD. We consider all major families of continual learning (dynamic architecture, replay, and regularization) and describe the details of each continual learning method (CL method) used in our experiments (Fig. 3).

4.2.1. Dynamic architecture
Single NODE per task (SG): A simple way to learn M tasks is to use a dedicated, newly-initialized NODE to learn a task and to freeze it afterwards. At the end we get M NODEs, from which we can pick one at prediction time to reproduce the desired trajectory (Fig. 3(a)). In this setting, which acts as an upper-performance baseline, catastrophic forgetting is eliminated because the parameters of a NODE trained on a task are not affected when a new NODE is trained on the next task. However, this also means that we end up with M times the number of parameters of a single NODE. This setup acts as a simplified version of the *Progressive Networks* approach [28], but without the layer-wise lateral connections to the networks of previous tasks. This keeps the inference time constant as new tasks are added.

4.2.2. Optimizing only for the current task
Finetuning (FT): A single NODE is sequentially finetuned on M tasks. To tell the NODE which task it should reproduce, i.e. to make it task-conditioned, we use an additional input in the form of a trainable vector known as a task embedding vector. This is similar to the approach followed for hypernetworks. After the mth task is learned, the trained task embedding vector $\mathbf{e}^m$ for that task is saved. To reproduce the trajectory for the mth task, we pick the corresponding task embedding vector and use it as the additional network input. The NODE parameters are finetuned to minimize the loss on the current task without any mechanism for avoiding catastrophic forgetting. In this setting (Fig. 3(b)), we would expect the NODE to only remember the latest task and so this acts as a lower-performance baseline.

4.2.3. Replaying training data
Replay (REP): For most continual learning scenarios *replaying* training data from previous tasks is a trivial exercise of combining the data from disparate tasks and then training a model with the composite dataset. However, for learning from demonstration, such a simple approach does not suffice as our NODE models need to be task conditioned. If they are trained using a jumbled up combination of demonstration trajectories from different tasks, they will be unable to learn any of the tasks. To overcome this issue we propose to use a task selector, as shown in Fig. 3(c). REP stores the dataset of each task it learns, and similar to FT, also maintains separate task embeddings for each task. In
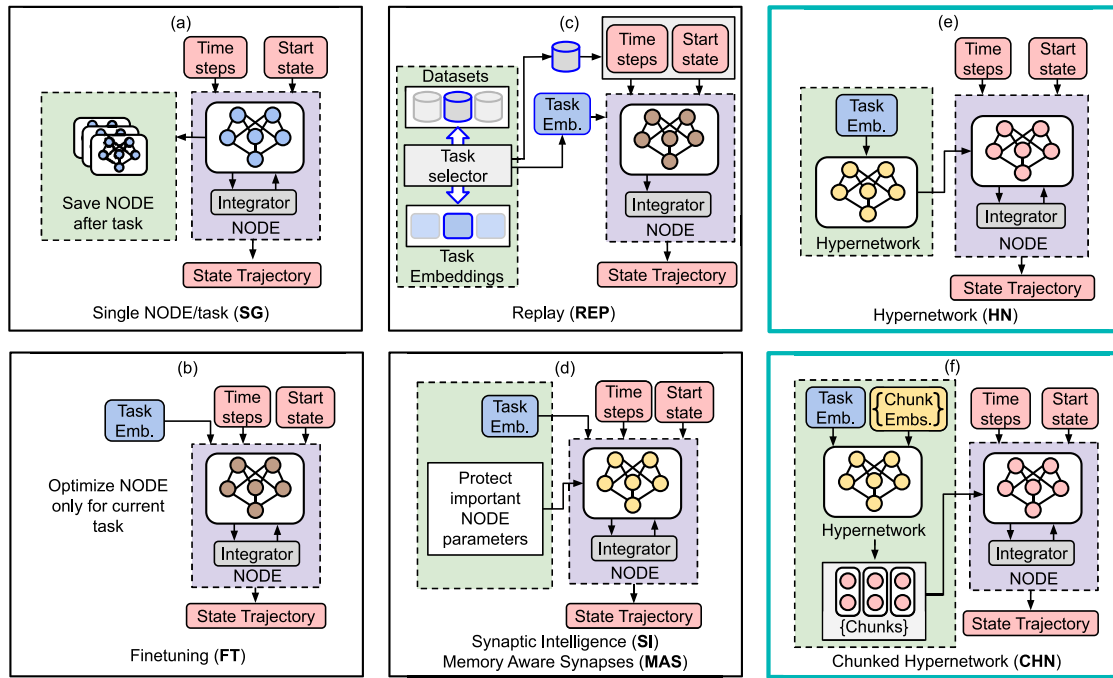
Fig. 3. Continual learning models used in our experiments are depicted in (a)–(f). Non-regularized trainable parameters that are initialized before and saved after each task are shown with ▩. Non-regularized parameters that are initialized only once before learning the first task are shown with ▩. Regularized trainable parameters that are protected from catastrophic forgetting while learning a sequence of tasks are shown with ▩. Other inputs and outputs are not trainable (shown with ▩). Given a start state and time steps, a NODE generates the state trajectory for the time steps. (a) **Single NODE per task (SG)**: A separate NODE learns only a single task. This forms the upper-performance baseline. (b) **Finetuning (FT)**: The NODE from the previous task is optimized for the current task by finetuning. (c) **Replay (REP)**: The training data from previous tasks is combined with the data of the current task. (d) **Synaptic Intelligence (SI)** and Memory Aware Synapses (**MAS**): NODE parameters are regularized to prevent changes to parameters important for solving previous tasks. (e) **Hypernetworks (HN)**: A hypernetwork produces all the NODE parameters using a task embedding vector. (f) **Chunked Hypernetworks (CHN)**: Chunk embedding vectors together with a task embedding vector are used to produce the NODE parameters in segments called chunks. HN and CHN (highlighted in turquoise) are our proposed solutions for continual learning from demonstration.

each training iteration during learning task m, the task selector randomly selects 1 out of the m previous tasks, and then passes the selected dataset and task embedding to the NODE, which optimizes its parameters for the selected task in that training iteration. When the REP model is trained for a moderately high number of iterations, each past task is sampled approximately uniformly and the NODE is able to learn continually. For learning new tasks, instead of scaling the number of iterations linearly with the number of tasks, we use a fixed number of training iterations, irrespective of the task being learned so that the run time does not explode while learning a long sequence of tasks.

4.2.4. Regularization

Synaptic Intelligence (SI): To learn M tasks, the NODE parameters are regularized with the SI loss $\tilde{\mathcal{L}}^m$ (Eq. (7)). The task-specific part ($\mathcal{L}^m$) of $\tilde{\mathcal{L}}^m$ corresponds to the NODE loss (Eq. (2)). Similar to FT, we make the SI NODE task-conditioned using a trainable task embedding vector, as shown in Fig. 3(d).

Memory Aware Synapses (MAS): For MAS [4], we also follow the architecture in Fig. 3(d). The NODE parameters are learned using Eq. (7) and we use Eq. (9) to compute Ω_k^m. As before, we apply Eq. (2) as the task-specific loss $\mathcal{L}^m$. The MAS NODE is also made task-conditioned with a trainable task embedding vector.

Both SI and MAS seek to change the plasticity/elasticity of the parameters of the NODE in an attempt to prevent changes to parameters that are important for solving past tasks.

Hypernetworks (HN): We use a hypernetwork to generate the parameters of a NODE. We first compute the candidate change $\triangle\mathbf{h}$ for the hypernetwork parameters by minimizing the NODE loss (Eq. (2)), which acts as our task-specific loss $\mathcal{L}^m$:

$$\mathcal{L}^m = \mathcal{L}^m(\boldsymbol{\theta}^m, \mathbf{y}^m) = \frac{1}{2}\sum_t \parallel \mathbf{y}_t^m - \hat{\mathbf{y}}_t^m \parallel_2^2 \tag{12}$$

where $\boldsymbol{\theta}^m = \mathbf{f_h}(\mathbf{e}^m, \mathbf{h})$, $\hat{\mathbf{y}}_t^m = \hat{\mathbf{y}}_0^m + \int_0^t \mathbf{f}_{\boldsymbol{\theta}^m}(\hat{\mathbf{y}}_\tau^m)\,\mathrm{d}\tau$

As in Eqs. (2) and (10), $\mathbf{y}_t^m$ and $\hat{\mathbf{y}}_t^m$ denote the ground truth observation and the prediction for the tth timestep of task m respectively, and $\boldsymbol{\theta}^m$ denotes the NODE parameters for task m generated by the hypernetwork $\mathbf{f_h}$ (with parameters $\mathbf{h}$) using the embedding vector $\mathbf{e}^m$ as input. We use Eq. (11) for training the hypernetwork in the second optimization step. The structure of HN is shown in Fig. 3(e).

Chunked Hypernetworks (CHN): We use a chunked hypernetwork to generate the parameters of a NODE (Fig. 3(f)). For this, Eqs. (12) and (11) are employed as the loss functions in the 2-step optimization process.

To reproduce a particular task with HN or CHN at test time, we input the corresponding task embedding vector to the hypernetwork which generates the NODE for that task. Thereafter, we simply provide the NODE with the starting state of the trajectory (and the timesteps in the case of NODE$^\mathrm{T}$) and it predicts the entire trajectory for the desired task.

For the comparative baselines (SG, REP, FT, SI, MAS), we included representative methods from all the major groups of continual learning methods. The choice of hypernetworks for our proposed approach to continual LfD (HN, CHN) was motivated by the following: (i) Hypernetworks have exhibited remarkable performance in multiple domains [2,5]. (ii) A robot capable of learning multiple tasks from demonstration needs to be told which task it should execute. Hypernetworks provide a natural way of task-conditioning using task embedding vectors, and thus are a very good fit for the LfD setup. (iii) Hypernetworks do not need to retrain on the data of past tasks. (iv) The chunked version of hypernetworks can be smaller than the NODEs they generate (model compression).

5. Experiments

We first describe the datasets used in our experiments and the metrics we report. This is followed by the details of our experimental results.

5.1. Datasets

We evaluate the performance of all models described in Section 4 on three different datasets with different numbers of tasks and degrees of difficulty: (i) *LASA* [8]: A dataset that is used frequently for comparisons and benchmarking in the area of LfD [9,37,38]; (ii) *HelloWorld*: A dataset of trajectories recorded while a robot was shown how to write letters containing loops; (iii) *RoboTasks*: A dataset of trajectories collected while the robot was shown how to perform tasks which need both the positions as well as orientations of the end-effector to be varied. Of these three datasets, *HelloWorld* and *RoboTasks* are introduced by us in this paper.

5.1.1. LASA

LASA [8] is a widely-used benchmark [9,27,37,38] for evaluating motion generation algorithms. It contains 30 patterns of handwritten motions, each with 7 similar demonstrations (see Fig. 7 for examples). We refer to each pattern $\mathcal{D}_m$ as a task. Of the 30 tasks, we use the first 26 tasks: $\mathcal{D}_{\text{LASA}} = \{\mathcal{D}_{0:25}\}$. We omit the last 4 tasks, each of which contains 2 or 3 dissimilar patterns merged together. Each demonstration of a task is a sequence of 1000 2-D points. We arrange the 26 tasks alphabetically in the order of their names and train sequentially models of SG, FT, REP, SI, MAS, CHN and HN on each task. During training only REP has access to the data of older tasks while the other methods are trained with the data of only the current task. We evaluate on the LASA dataset because of its wide adoption by the LfD community [9,27,37,38], and because it contains a large number of diverse tasks which can be used to gauge the continual learning ability of our models.

5.1.2. HelloWorld

We further evaluate our approach on a dataset of demonstrations we collected using the Franka Emika Panda robot. The *x* and *y* coordinates of the robot's end-effector were recorded while a human user guided it kinesthetically to write the 7 lower-case letters *h,e,l,o,w,r,d* one at a time on a horizontal surface. The *HelloWorld* dataset $\mathcal{D}_{\text{HW}} = \{\mathcal{D}_{0:6}\}$ consists of 7 tasks, each containing 8 slightly varying demonstrations of a letter. Each demonstration is a sequence of 1000 2-D points. After training on all the tasks, the objective is to make the robot write the words "*hello world*". Our motivation for using this dataset is to test our approach on trajectories containing loops and to show that it also works on kinesthetically recorded demonstrations using a real robot. This dataset is available at https://github.com/sayantanauddy/clfd.

5.1.3. RoboTasks

To evaluate our approach on robotic tasks that can be expected in the real world, we collect a dataset of 4 tasks. For each task, a human user kinsethetically guides the robot's end-effector while varying both the position in 3D space and also the orientation in all three rotation axes. The tasks of this dataset are: (i) *box opening*: the lid of a box is lifted to an open position; (ii) *bottle shelving*: a bottle in a vertical position is transferred to a horizontal position on a shelf; (iii) *plate stacking*: a plate in a vertical position is transferred to a horizontal position on an elevated platform while orienting the arm so as to avoid the blocks used for holding the plate in its initial vertical position; (iv) *pouring*: a cup full of coffee beans is taken from an elevated platform and

the contents of the cup are emptied into a container. Thus, the *RoboTasks* dataset $\mathcal{D}_{\text{Robot}} = \{\mathcal{D}_{0:3}\}$ consists of 4 tasks, where each task contains 9 demonstrations, each of 1000 steps. In each step we record the position $\mathbf{p} \in \mathbb{R}^3$ and the orientation as a unit quaternion $\mathbf{q} \in \mathbb{S}^3$. Compared to $\mathcal{D}_{\text{LASA}}$ and $\mathcal{D}_{\text{HW}}$, $\mathcal{D}_{\text{Robot}}$ contains a lot more variability in the demonstrations and presents a more difficult learning challenge.

We use this dataset because we want to evaluate our approach in a real-world setup involving changing positions and orientations, and also because we did not find any existing datasets that are suitable for our purpose. Datasets such as RoboNet [39] and Meta-World [40] are designed for reinforcement learning and do not contain the demonstrations in terms of task-space trajectories. See Fig. 1 for a visual example of the tasks in $\mathcal{D}_{\text{Robot}}$. This dataset is also available at https://github.com/sayantanauddy/clfd.

5.2. Metrics

5.2.1. Trajectory metrics

To evaluate the end-effector position trajectories, we report the following widely used metrics: *Dynamic Time Warping error* (DTW) [9,41], *Swept Area error* [26], and *Frechet distance* [9], which measure how close the predicted trajectories are to the ground-truth demonstrations. For orientation, we report the commonly used quaternion error [35,36] defined as

$$e_q(\mathbf{q}_1, \mathbf{q}_2) = 2\text{Log}(\mathbf{q}_1 * \bar{\mathbf{q}}_2) \in \mathbb{R}^3 \tag{13}$$

where $\mathbf{q}_1$ and $\mathbf{q}_2$ are quaternions. Given a ground truth trajectory $\mathbf{q}_{0:T-1}$ and a predicted trajectory $\hat{\mathbf{q}}_{0:T-1}$ (both containing T quaternions), we compute the error between the two trajectories using

$$E_q = \frac{1}{3T} \sum_{t=0}^{T-1} \|e_q(\mathbf{q}_t, \hat{\mathbf{q}}_t)\|_1 \tag{14}$$

5.2.2. Continual learning metrics

We report *Accuracy* (ACC), *Remembering* (REM), *Model Size Efficiency* (MS) and *Sample Storage Size Efficiency* (SSS) [42]. ACC is a measure of the average accuracy for the current and past tasks. REM measures how well past tasks are remembered. MS measures how much the size of a model grows compared to its size after learning the first task. SSS measures how the storage requirements of a model grows due to the storage of training data from older tasks.

Additionally, we introduce two new easy-to-compute continual learning metrics: *Time Efficiency* (TE) and *Final Model Size* (FS). TE measures the increase in training duration with the number of tasks, relative to the training time for the first task. TE only needs the training times to be logged, and it reflects the extra effort needed in the training loop (e.g. due to extra regularization steps) with an increase in the number of tasks. TE is similar to the *Computational Efficiency* metric (CE) proposed in [42], but while CE is based on the number of addition and multiplication operations, TE is based on the observed wall clock time. When a neural network is trained on a GPU (as is most often the case), many neural network operations are performed in parallel. Hence, simply counting the number of operations does not provide an accurate estimate of the increase in training effort with increasing number of tasks. A time-based measure such as TE is more suited to this task. For a meaningful interpretation of this metric, all experiments need to be run on near-identical hardware.

For M tasks, TE is defined as

$$\text{TE} = \min\left\{1, \frac{\mathcal{T}_0}{M} \sum_{i=0}^{M-1} \frac{1}{\mathcal{T}_i}\right\}, \tag{15}$$

where $\mathcal{T}_i$ is the time required to learn task *i*.

FS is a measure of the *absolute* parameter size, which contrasts with MS which only measures the parameter growth relative to the size after learning the first task. A model which has a large number of parameters for the first task and adds a relatively small number of parameters for subsequent tasks will achieve a high score for MS, but will fare worse in terms of FS if models of other compared methods have a smaller absolute size. FS is defined as

$$FS = 1 - \mathrm{Mem}_{norm}(\boldsymbol{\theta}^{M-1}) \tag{16}$$

$\mathrm{Mem}_{norm}(\boldsymbol{\theta}^{M-1})$ is the parameter size after learning M tasks, normalized by the size of the largest compared model among SG, FT, REP, SI, MAS, HN and CHN. With these 6 metrics, we compute the overall continual learning metrics [42]:

$$CL_{score} = \sum_{i}^{n(\mathcal{C})} c_i, \tag{17}$$

$$CL_{stability} = 1 - \sum_{i}^{n(\mathcal{C})} \mathrm{STDEV}(c_i), \tag{18}$$

where $\mathcal{C} = \{ACC, REM, MS, TE, FS, SSS\}$. All the continual learning metrics lie in the range 0 (worst) to 1 (best).

5.3. Results

We present next the results of our experiments on three different datasets: *LASA*, *HelloWorld*, and *RoboTasks*. Details of our hardware setup (Appendix A.1), experiment hyperparameters (Appendix A.4), results of the experimental comparison of NODEs and iFlow (Appendix A.5), and additional results (Appendix A.6) can be found in Appendix A.

5.3.1. LASA dataset

We train each model (SG, FT, REP, SI, MAS, CHN and HN) on the 26 tasks of $\mathcal{D}_{LASA}$ sequentially. Fig. 4 shows the median DTW errors of the predictions for tasks $\mathcal{D}_0$–$\mathcal{D}_m$ after training on task $\mathcal{D}_m$ (using NODE^T) for $m = 0, 1, \ldots, 25$. For example, the value for task 10 denotes the median of the evaluation errors for all the predicted trajectories from tasks 0 to 10 after training on task 10. We provide the same information using the other trajectory error metrics (Frechet distance and Swept Area in Fig. A.2 in Appendix A).

Fig. 4 (top) shows a drastic increase in the error for FT as the number of learned tasks increases, since FT optimizes its parameters *only* for the current task. After the first task, the errors for SI and MAS also increase steeply. Fig. 4 (bottom) shows a zoomed-in version showing the methods which perform well in more detail (SG, REP, CHN and HN). The performance of REP for task 0 is the same as FT, but the performance does not deteriorate as more tasks are learned, since REP has access to the data from prior tasks. Till task 7, CHN performs better than REP, after which it starts exhibiting catastrophic forgetting as can be seen from the upward trend in its error plot. Overall, SG and HN exhibit the best performance. They do not suffer from catastrophic forgetting, and their error plots overlap with each other (red and blue lines in Fig. 4 (bottom)).

An overall picture of the continual learning performance of the different methods may be obtained from Fig. 5, which shows the overall errors of the predicted trajectories during the course of learning all the tasks. It can be seen that SG, REP, CHN and HN perform much better than FT, SI, and MAS, and that HN and SG are the best performers. Note that in Fig. 5 the trajectory metrics are plotted in the $\log_{10}$ scale to accommodate the high errors for FT, SI, and MAS in the same plot as SG, REP, HN and CHN.

Note that a regular HN model produces all the parameters of the target NODE directly from its final layer. This implies a much larger number of HN parameters compared to the produced
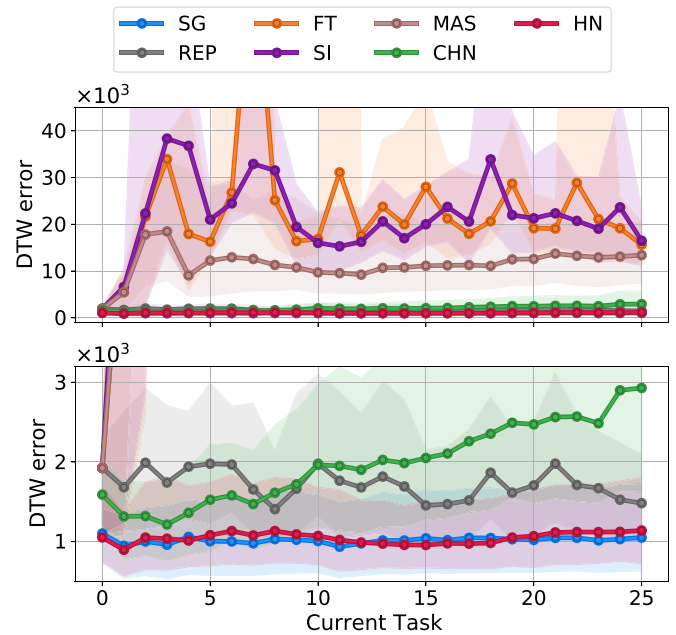

Fig. 4. DTW errors of trajectories predicted for the LASA dataset (lower is better). The *x*-axis shows the *current* task. After learning a task (using NODE^T), all current and previous tasks are evaluated. HN performs as well as SG but with negligible growth in parameter size. Lines show medians and shaded regions denote the lower and upper quartiles of the errors over 5 independent seeds. (top) The DTW errors for all methods. (bottom) A zoomed-in view of the methods that perform well. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
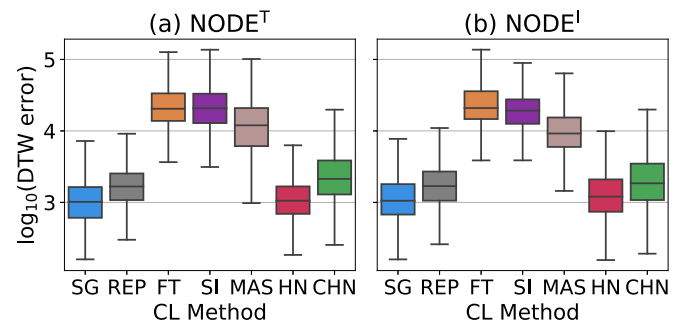

Fig. 5. DTW errors (lower is better) of all trajectory predictions during the learning of all tasks of the LASA dataset. Results are obtained using 5 independent seeds. The hypernetwork approach (HN, CHN) performs substantially better than regularization-based continual learning methods (SI, MAS) and on par with the upper baseline SG and REP.

target network parameters. To keep the overall size of the HN comparable to the other models, the target NODE for HN has one-tenth the number of units in each layer as the NODEs used by the other models (SG, FT, REP, SI, MAS, CHN). We refer the reader to Table A.3 in Appendix A for details.

Although HN's performance after learning 26 tasks is very similar to that of SG, its parameter size is 4.3×10^6 compared to SG's combined size of 52.2×10^6 parameters, as shown in Fig. 6. CHN's final parameter size is 1.9×10^6. Also, the parameter count for SG grows by 2.1×10^6 per task, whereas CHN and HN grow at a much smaller rate of 256 parameters per task, same as SI, MAS, FT and REP (Fig. 6 does not include REP as it has the same model size as FT). Thus, CHN and especially HN perform similar to the upper baseline SG, while their parameter size is close to the lower baseline FT. Fig. 7 shows examples of trajectories predicted
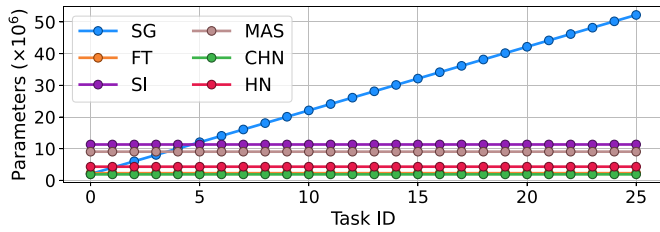
Fig. 6. Growth of parameter size with new tasks for the LASA dataset (using NODE^T). SG has a high rate of growth since it uses a separate network for each task. All other models grow by only 256 parameters for each new task. Plots for CHN and FT overlap with each other. The plot for REP is not shown since its size is identical to FT.
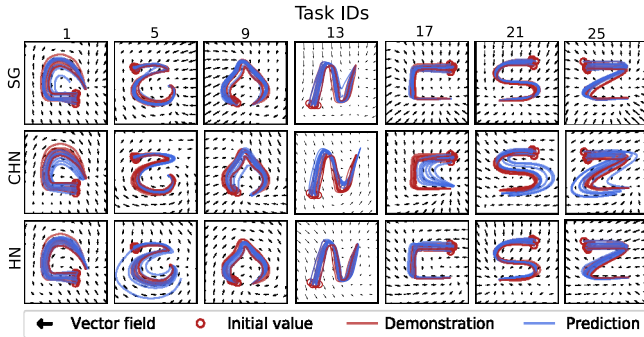

Fig. 7. Example of trajectories predicted by SG, CHN and HN using NODE^T for a selection of LASA tasks after learning the last task. HN can remember all past tasks, while CHN exhibits some forgetting.

Table 1

Continual learning metrics for the LASA dataset (median over 5 seeds). Values range from 0 (worst) to 1 (best).

(a) NODE^T

MET	ACC	REM	MS	TE	FS	SSS	CL$_{sco}$	CL$_{stab}$
SG	**0.87**	**1.00**	0.15	0.85	0.00	**1.00**	0.64	0.55
FT	0.06	0.20	**1.00**	0.89	**0.96**	**1.00**	0.68	0.57
REP	0.67	**1.00**	**1.00**	0.88	**0.96**	0.48	0.83	0.79
SI	0.04	0.38	**1.00**	**0.98**	0.78	**1.00**	0.70	0.60
MAS	0.02	0.86	**1.00**	0.83	0.83	**1.00**	0.76	0.63
HN	0.86	0.97	**1.00**	0.51	0.92	**1.00**	**0.88**	**0.81**
CHN	0.51	0.80	**1.00**	0.53	**0.96**	**1.00**	0.80	0.77

(b) NODE^I

MET	ACC	REM	MS	TE	FS	SSS	CL$_{sco}$	CL$_{stab}$
SG	**0.81**	**1.00**	0.15	0.85	0.00	**1.00**	0.64	0.56
FT	0.06	0.22	**1.00**	0.88	**0.96**	**1.00**	0.69	0.57
REP	0.65	**1.00**	**1.00**	0.83	**0.96**	0.48	0.82	0.79
SI	0.04	0.38	**1.00**	0.91	0.78	**1.00**	0.69	0.61
MAS	0.03	0.84	**1.00**	0.87	0.83	**1.00**	0.76	0.63
HN	0.76	0.97	**1.00**	0.55	0.92	**1.00**	**0.87**	**0.82**
CHN	0.57	0.86	**1.00**	0.52	**0.96**	**1.00**	0.82	0.78

Table 2

Robustness to changes in regularization hyperparameters for the LASA dataset (5 configurations for each method).

Method	CL$_{score}$		CL$_{stability}$	
	Median	IQR	Median	IQR
HN	0.8578	0.0011	0.8324	0.0050
CHN	0.7939	0.0098	0.8126	0.0022
MAS	0.7104	0.0019	0.6562	0.0062
SI	0.6047	0.0065	0.6403	0.0011

by SG, CHN, and HN for a selection of past tasks after learning the last task.

To compute the continual learning metrics [42], each predicted trajectory needs to be marked as accurate or inaccurate based on its difference to the ground truth. Since there is no preexisting procedure for this, we adopt the following approach: We set a threshold on the DTW error, such that predictions with an error less than the threshold are considered accurate. As each task has multiple ground truth demonstrations, we first compute the DTW error between all pairs of demonstrations for each task. We then find the maximum value from this list and multiply it by 3. The multiplicative factor 3 was chosen empirically to allow some room for error such that a predicted trajectory with the same general shape as its demonstration (discerned visually) is considered accurate. Doing so, we arrive at a DTW threshold of 2191 for $\mathcal{D}_{LASA}$, and use it to evaluate the continual learning metrics shown in Table 1.

For both NODE variants, HN outperforms all the compared models in terms of CL$_{score}$. For NODE^T, HN performs close to the upper baseline SG in terms of both ACC and REM. The additional regularization needed for training hypernetworks leads to a comparatively lower score for the time efficiency metric TE for CHN and HN (actual wall-clock training times can be found in Table A.1(a) in Appendix A). A very high parameter growth rate for SG results in poor scores for MS and FS. The replay method REP achieves a low score for SSS since it is the only method that needs to store the training data from previous tasks. The direct time input in NODE^T also leads to better ACC scores for SG, REP and HN, compared to NODE^I. Fig. 8 shows how the different methods measure up against each other in the different aspects of continual learning performance.

To test the sensitivity of the regularization-based continual learning methods (SI, MAS, HN and CHN) to changes in the regularization hyperparameters, we create sets of 5 hyperparameters

each for SI, MAS, HN and CHN by drawing independently and uniformly from the following ranges: (SI) $c \in [0.1, 0.5], \xi \in [0.1, 0.5]$, (MAS) $c \in [0.1, 0.5]$, (CHN) $\beta \in [10^{-3}, 10^{-2}]$, (HN) $\beta \in [10^{-3}, 10^{-2}]$ resulting in 20 different configurations. We then repeat the LASA experiment with NODE^T for all these configurations. In terms of CL$_{score}$ we observe that all configurations of HN outperform all configurations of CHN, which in turn are better than all configurations of MAS, followed by SI. This trend is reflected in the medians and inter-quartile ranges (IQR) of the overall continual learning metrics CL$_{score}$ and CL$_{stability}$ for each method (over its 5 configurations) shown in Table 2. It can be seen that HN and CHN perform better than the other methods and the variability in terms of IQR is very small, thereby showing that they are robust to changes in the regularization hyperparameter β. For calculating CL$_{score}$ and CL$_{stability}$ in this experiment, we do not consider the SSS metric since none of the regularization-based methods need to cache training data from prior tasks.

5.3.2. HelloWorld

For $\mathcal{D}_{HW}$, which comprises 7 tasks, we perform the same experiments as $\mathcal{D}_{LASA}$. Fig. 9 shows the errors in the predicted trajectories for all past and current tasks, as new tasks are learned. The median errors for CHN, HN and REP stay nearly unchanged and are similar to the upper baseline SG, although CHN and HN do not need to store the training data of past tasks like REP. As before, FT, SI, and MAS exhibit severe catastrophic forgetting. Due to fewer tasks in $\mathcal{D}_{HW}$, CHN's performance does not deteriorate even after learning all tasks. Fig. A.3 in Appendix A shows the other trajectory error metrics (Frechet distance and Swept Area), which exhibit the same trend as DTW.

Fig. 10 shows examples of trajectories predicted by SG, CHN and HN for past tasks after being trained sequentially on all $\mathcal{D}_{HW}$ tasks. All models exhibit superior performance when using the additional time input in NODE^T (Fig. 10(a)), without which even
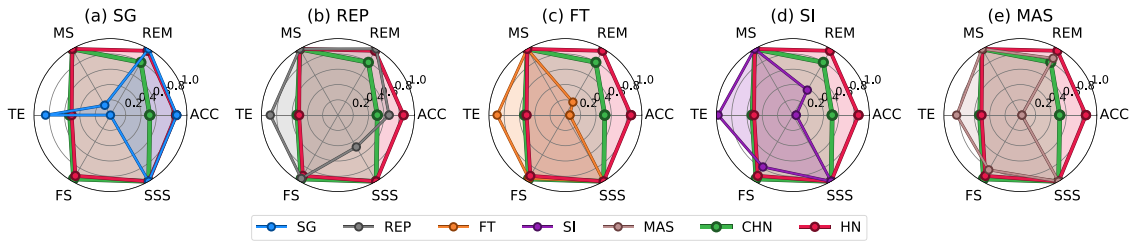
Fig. 8. Continual learning metrics (0: worst, 1: best) for the different methods (with NODE^T) on the LASA dataset. In (a)–(e), each of the baseline methods is compared against HN and CHN to illustrate the different tradeoffs made by the methods. HN shows good performance for almost all metrics (apart from TE, in which it performs moderately). CHN is similar to HN in terms of TE but scores lower in REM and ACC. REP also performs well but has a low score for SSS, and the other methods fail in multiple metrics.
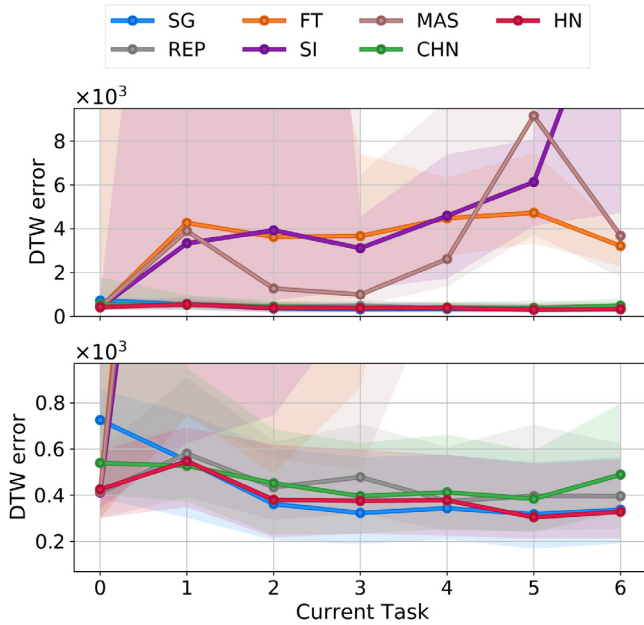

Fig. 9. DTW errors of trajectories predicted for the HelloWorld dataset (lower is better). The *x*-axis shows the *current* task. After learning a task (using NODE^T), all current and previous tasks are evaluated. Lines show medians and shaded regions denote the lower and upper quartiles of the errors over 5 independent seeds. (top) The DTW errors for all methods. (bottom) A zoomed-in view of the methods that perform well. The performance of SG, REP, HN and CHN are nearly identical and much better than FT, SI and MAS.

SG is unable to learn trajectories with loops. This can be seen from the errors for the letters *e*, *r* and *d* in Fig. 10(b). This is also evident in Fig. 11 which shows the errors for all predictions during the course of learning all the tasks. Apart from FT, all methods have higher median errors when using NODE^I (Fig. 11(b)) compared to NODE^T (Fig. 11(a)).

Using the same threshold computation approach we followed for $\mathcal{D}_{LASA}$, we compute a DTW threshold value of 1821 for $\mathcal{D}_{HW}$. With this, we compute the continual learning metrics shown in Table 3. The advantage of using NODEs with a time input is clear from the higher values of ACC for NODE^T compared to NODE^I for all the methods. In terms of ACC or REM, there is very little to choose between SG, REP, HN and CHN. However, when all the CL metrics are considered together (see the CL$_{sco}$ column in Table 3), CHN exhibits the best performance on account of its small size and the fact that it does not need to cache training data from past tasks. HN shows the second-best overall performance. Fig. 12 illustrates that the hypernetwork methods (HN and CHN) perform well in all continual learning metrics, while the other methods perform well in only some aspects of continual learning but fail in others.
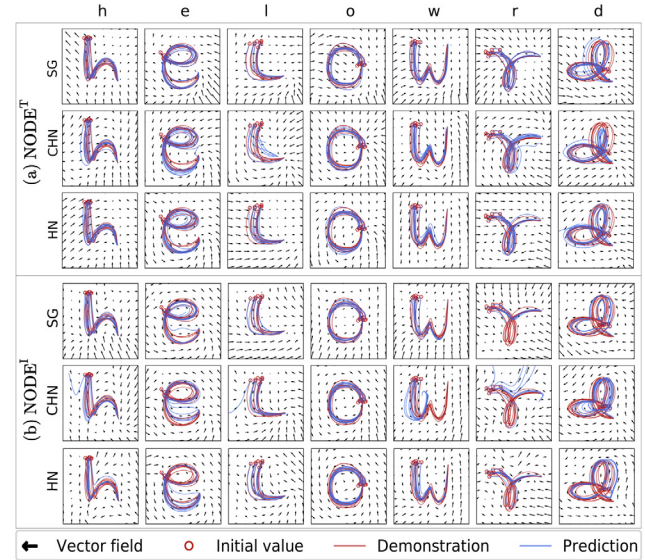

Fig. 10. Example of trajectories predicted by SG, CHN and HN using NODE^T for all past HelloWorld tasks after learning the last task. With NODE^T, even trajectories with loops can be learned. HN and CHN (with NODE^T) can remember all past tasks.
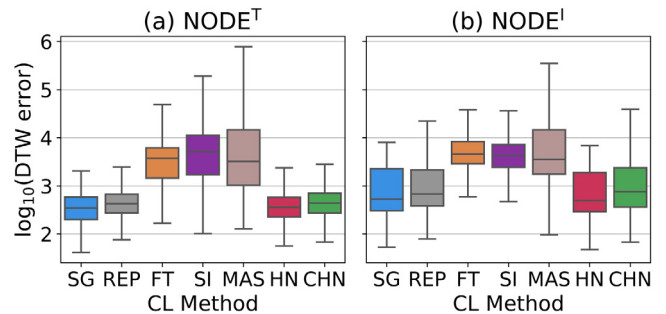
Fig. 11. DTW errors (lower is better) of all trajectory predictions during the learning of all tasks of the HelloWorld dataset. Results are obtained using 5 independent seeds. HN, CHN perform as well as the upper baseline SG.

We qualitatively evaluate how the trajectories predicted by HN can be reproduced with a physical robot. For this, we use the same Franka Emika Panda robot that was used for recording the demonstrations for $\mathcal{D}_{HW}$. The HN model trained on the 7 tasks of $\mathcal{D}_{HW}$ is queried to produce the letters *h*, *e*, *l*, *l*, *o*, *w*, *o*, *r*, *l*, *d* by using the appropriate task embedding vectors in sequence. The trajectory of each letter is scaled and translated by a constant amount and provided to the robot, which then follows this path with its end-effector. The *z*-coordinate and orientation of the

Fig. 12. Continual learning metrics (0: worst, 1: best) for the different methods (with NODE^T) on the HelloWorld dataset. In (a)–(e), each of the baseline methods is compared against HN and CHN to illustrate the different tradeoffs made by the methods. The hypernetwork methods (HN,CHN) show good performance for all metrics whereas the other methods perform well only in some metrics and poorly for others.
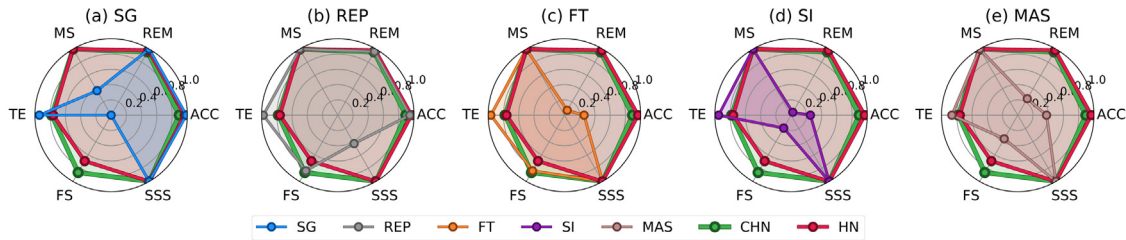
Table 3
Continual learning metrics for the HelloWorld dataset (median over 5 seeds). Values range from 0 (worst) to 1 (best).

(a) NODE^T

MET	ACC	REM	MS	TE	FS	SSS	CL_{sco}	CL_{stab}
SG	**0.99**	**1.00**	0.37	0.94	0.00	**1.00**	0.72	0.57
FT	0.26	0.07	**1.00**	0.97	0.84	**1.00**	0.69	0.59
REP	0.94	0.96	**1.00**	**0.98**	0.84	0.43	0.86	0.78
SI	0.25	0.04	**1.00**	0.95	0.19	**1.00**	0.57	0.55
MAS	0.38	0.25	**1.00**	0.86	0.36	**1.00**	0.64	0.65
HN	0.97	0.98	**1.00**	0.76	0.69	**1.00**	0.90	0.86
CHN	0.89	0.95	**1.00**	0.78	**0.87**	**1.00**	**0.92**	**0.91**

(b) NODE^I

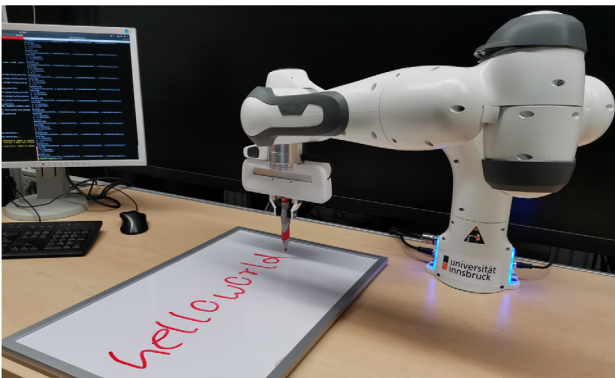| MET	ACC	REM	MS	TE	FS	SSS	CL_{sco}	CL_{stab}
SG	0.71	**1.00**	0.37	0.93	0.00	**1.00**	0.67	0.59
FT	0.17	0.24	**1.00**	0.93	0.84	**1.00**	0.70	0.62
REP	0.71	**1.00**	**1.00**	**0.94**	0.84	0.43	0.82	0.78
SI	0.19	0.29	**1.00**	**0.94**	0.19	**1.00**	0.60	0.59
MAS	0.25	0.47	**1.00**	0.86	0.36	**1.00**	0.66	0.66
HN	**0.74**	0.99	**1.00**	0.75	0.69	**1.00**	0.86	0.85
CHN	0.68	0.96	**1.00**	0.76	**0.87**	**1.00**	**0.88**	**0.87**

Fig. 13. After being continually trained using learning from demonstration to write single letters, the robot can reproduce all the trajectories that it has learned in the past with a single HN network and without having access to training data from past tasks. Video is available at https://youtu.be/0gdIImIBnXc.

end-effector are fixed. Fig. 13 shows the letters written by the robot.

5.3.3. RoboTasks

$\mathcal{D}_{Robot}$ comprises 4 tasks, where each task involves learning the vector fields of the position as well as the orientation of the end-effector of the robot. For this experiment, the methods we evaluate are SG, FT, REP, CHN and HN. We omit SI and MAS because of their poor performance on the previous datasets. We use only NODE^T for this experiment since in the previous experiments it has been shown to be better than NODE^I. As before, SG acts as an upper baseline and FT serves as a lower performance

baseline. For each method, we train separate models for learning the position and orientation trajectories. The order of tasks for this experiment is: *box opening*, *bottle shelving*, *plate stacking*, and *pouring*.

Fig. 14 shows the position and orientation errors of the predicted trajectories for all past and current tasks, as new tasks are learned. For both positions and orientations, FT is unable to predict correct trajectories for past tasks as soon as the second task is learned. The performance of the other methods is comparable to each other. For the end-effector position trajectories (Fig. 14(a) (bottom)), the performance of SG, REP and HN are nearly identical, while CHN is slightly worse. However this difference does not result in significant qualitative differences in the robot's movement. SG, REP, CHN and HN are equally good at predicting the orientation trajectories, as can be seen in Fig. 14(b) (bottom). The slight upward trend in the plots of SG, REP, CHN and HN in Fig. 14 can be attributed to the fact that task 2 (*plate stacking*) is more difficult than the other tasks due to very diverse ground truth demonstrations. It can be observed that CHN shows a more pronounced upward trend in the position error plot (Fig. 14(a) (bottom)). This is probably due to the fact that CHN has far fewer parameters than HN and thus has lower representation and remembrance capabilities.

Similar to the results for $\mathcal{D}_{LASA}$ and $\mathcal{D}_{HW}$, it can be seen that for continually learning real-world robot tasks involving both position and orientation, the hypernetwork-based methods CHN and HN perform similar to SG and REP, but without needing significant growth in parameters and without requiring to be retrained on the data from prior tasks.

Fig. 15 shows an example of the position trajectories and Fig. 16 shows an example of the orientation trajectories predicted by SG, CHN and HN for tasks 0–4 after the last task has been learned (here we show the orientation in terms of Euler angles and the corresponding results in terms of quaternion elements can be seen in Fig. A.4). In both these qualitative examples, CHN and HN produce trajectories that closely mimic the ground truth demonstration. Fig. 17 shows the overall prediction errors during the course of training. Images of the robot performing the 4 tasks of $\mathcal{D}_{Robot}$ after learning all the tasks sequentially with CHN can be seen in Fig. 1.

Next, we report the continual learning metrics for $\mathcal{D}_{Robot}$ in Table 4. For this we again need to set thresholds on the position and orientation errors to categorize predicted trajectories as accurate or inaccurate. This time we follow a slightly different approach for setting these thresholds compared to $\mathcal{D}_{LASA}$ and $\mathcal{D}_{HW}$. For positions, we first compute the DTW error between all pairs of demonstrations for each task. Since the ground truth demonstrations are already quite diverse, we use the maximum value from this list as the position error threshold. This threshold DTW value is 7131. For setting the orientation error threshold, we simply use a threshold of 10 degrees, since an absolute error of 10 degrees (averaged across the 3 rotation axes) results in roughly the same orientation as the desired one.
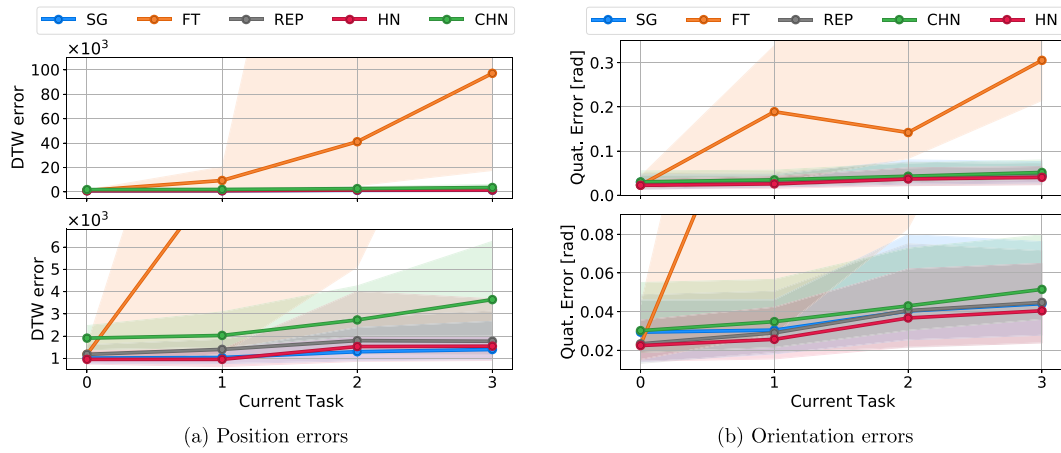
Fig. 14. Position (a) and orientation (b) errors of trajectories predicted for the RoboTasks dataset (lower is better). The *x*-axis shows the *current* task. After learning a task (using NODE^T), all current and previous tasks are evaluated. Lines show medians and shaded regions denote the lower and upper quartiles of the errors over 5 independent seeds. (top row) The errors for all methods are plotted. (bottom row) A detailed view of the methods that perform well. The performance of HN is comparable to the upper baseline SG.
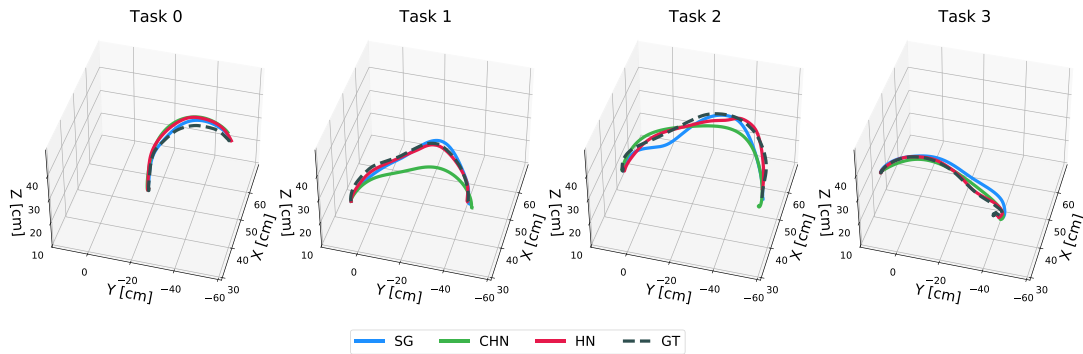
Fig. 15. Example of end-effector position trajectories predicted for all past tasks after sequentially learning all the tasks of the RoboTasks dataset. All methods employ NODE^T as the trajectory learning method. Hypernetworks (HN and CHN) remember all past tasks and mimic the demonstrations accurately.
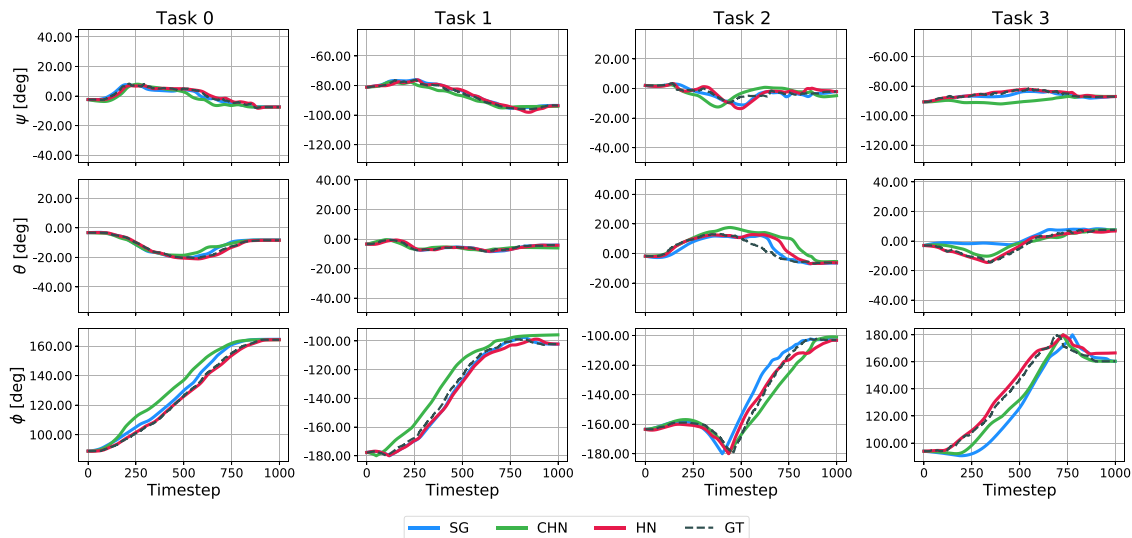
Fig. 16. Example of end-effector orientation trajectories predicted for all past tasks after sequentially learning all the tasks of the RoboTasks dataset. All methods employ NODE^T as the trajectory learning method. The ground truth demonstrations as well as the predictions of the orientation consist of a sequence of unit quaternions. Here, we convert each quaternion in the sequence to its corresponding Euler angle representation ($z - y - x$ convention) for easier interpretation. Columns show the predictions for each task, and rows show the different Euler angles in degrees. A figure with the original quaternion sequences can be seen in Fig. A.4. Hypernetworks (HN and CHN) remember all past tasks and mimic the ground truth orientations accurately.

For position as well as orientation, SG, REP, CHN and HN are comparable to each other in terms of ACC and REM. However, when all the metrics are considered together, CHN is the best amongst all the methods on account of its small near-constant

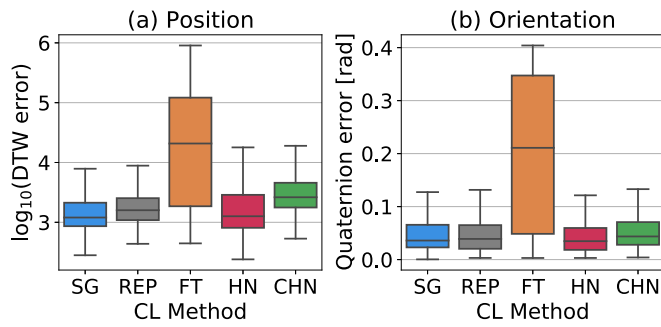**Fig. 17.** Prediction errors (lower is better) showing DTW position errors (a) and Quaternion orientation errors (b) of all trajectory predictions during the learning of all tasks of the RoboTasks dataset. Results are obtained using 5 independent seeds. Overall, HN and CHN produce low errors which are comparable to the performance of the upper baseline SG.

Table 4
Continual learning metrics for the RoboTasks dataset (median over 5 seeds). Values range from 0 (worst) to 1 (best).

(a) Position

MET	ACC	REM	MS	TE	FS	SSS	CL_{sco}	CL_{stab}
SG	**0.95**	**1.00**	0.52	**0.93**	0.00	**1.00**	0.73	0.60
FT	0.39	0.01	**1.00**	0.84	0.69	**1.00**	0.66	0.61
REP	0.94	0.94	**1.00**	0.84	0.69	0.38	0.80	0.77
HN	0.88	0.98	**1.00**	0.72	0.45	**1.00**	0.84	0.78
CHN	0.90	0.90	**1.00**	0.75	**0.74**	**1.00**	**0.88**	**0.89**

(b) Orientation

MET	ACC	REM	MS	TE	FS	SSS	CL_{sco}	CL_{stab}
SG	0.98	**1.00**	0.52	0.99	0.00	**1.00**	0.75	0.59
FT	0.49	0.15	**1.00**	0.99	0.66	**1.00**	0.68	0.62
REP	**0.99**	**1.00**	**1.00**	**1.00**	0.66	0.38	0.84	0.74
HN	**0.99**	**1.00**	**1.00**	0.77	0.20	**1.00**	0.83	0.68
CHN	0.98	0.97	**1.00**	0.78	**0.79**	**1.00**	**0.92**	**0.90**

parameter size and its non-dependence on the demonstrations of previous tasks. This is also corroborated by Fig. 18, which shows that CHN performs well in all the continual learning metrics, while the other methods perform poorly in one or more metrics.

We also evaluate the robustness of the predicted trajectories when the robot starts from novel initial positions that are not present in the demonstrations used for training. To this end, we compare the predictions made by upper-baseline SG and CHN for 100 randomly selected starting positions on each of the 4 tasks of the RoboTasks dataset. Please refer to Appendix A.7 for details. Our results indicate that SG is much more sensitive to small changes in the starting position despite being the best at predicting trajectories that have the same starting position as the demonstrations. The continual learning process for CHN appears to reduce overfitting to the demonstrations and makes the model more robust to changes in the initial state of the robot.

6. Discussion

To be functional in the real world, a robot must be able to acquire multiple skills over time, without forgetting the skills that were learned in the past and also without needing to be retrained on past skills. In this paper, we have taken a further step in this direction by presenting the first work on continual trajectory learning from demonstrations.

The results from our experiments on 3 different datasets show that the hypernetwork model (HN) performs on par with the upper baseline (SG) for all datasets, and the smaller chunked hypernetwork (CHN) performs on par with SG on the robot datasets
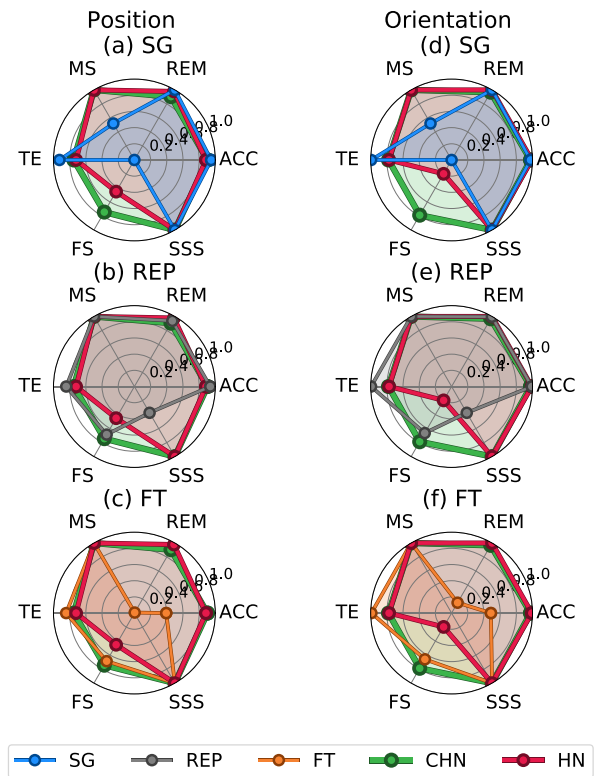

Fig. 18. Continual learning metrics (0: worst, 1: best) for the different methods (with $NODE^T$) on the RoboTasks dataset. In each individual plot, one of the baseline methods is compared against HN and CHN to illustrate the different tradeoffs made by the methods. The first column (a)–(c) shows the results for position, and the second column (d)–(f) shows the results for orientation. Across all metrics, CHN shows good performance for both position and orientation. HN exhibits a similar performance but due to its bigger size, achieves a low score for FS for orientation. REP's score for SSS is low due to its requirement of storing training data of all tasks. SG and FT achieve low scores in multiple metrics.

(HelloWorld and RoboTasks). Compared to the growth of SG for each new task, the hypernetwork models (HN, CHN) scale much more efficiently as the growth per task of HN and CHN is negligible (Section 5.3).

Using continual learning metrics (e.g. accuracy of predictions, how well past tasks are remembered, model size growth, storage of training data from past tasks, etc.), we compared the hypernetwork-based approach to methods from all continual learning families and found that empirically, hypernetworks perform best. Specifically, HN is the best among the compared models for the LASA dataset. On the remaining 2 robot datasets, the hypernetwork-based methods outperform the other approaches. CHN, due to its memory efficiency, is empirically the best, followed by HN.

We also improve the trajectory learning NODE by introducing an additional time input to create the $NODE^T$ model. In Section 5.3.2, we show how $NODE^T$ improves performance and enables the models to learn trajectories with loops.

To facilitate future research in the area of continual-LfD, we release 2 new datasets of trajectories collected kinesthetically with a real robot: *HelloWorld* (Section 5.3.2) and *RoboTasks* (Section 5.3.3). The latter includes tasks involving changing positions and orientations. After continually learning all the tasks from these datasets using our hypernetwork-based approach, we verified that the real robot can successfully perform all past tasks. Further, we verified that the order of tasks during training does

not affect the performance of these models (refer to Appendix A.8 for details).

The code used in this paper is made publicly available to aid reproducibility. Although the HelloWorld and RoboTasks datasets used in our experiments were recorded kinesthetically, our implementation only requires the training data in terms of the robot's trajectories and does not depend on how this data is collected. Similar to [19], our implementation can potentially also be used for trajectories recorded using other means such as shadowing or teleoperation [17].

A limitation of our current hypernetwork-based approach is the increase in training time with an increasing number of tasks because an additional regularization term is added for each new task (e.g. HN needs 620 s to learn task 0 and 1609 s to learn task 25 of the LASA dataset, further details in Table A.1). As proposed by von Oswald et al. [5], this can be overcome by using an approximate sample-based regularization instead of the full regularization process. We leave the experimental verification of this solution to future work.

In this paper, our focus is on continual learning and we use NODEs as a simple trajectory learning method because of its good empirical performance and fast convergence during training, which facilitates running multiple experiments on long sequences of tasks (Section 4.1). However, NODEs do not have a mechanism for enforcing the stability of the predicted trajectories. To overcome this, it is possible to substitute the simple NODE with a stable alternative in the future [9,10]. Interestingly, we found that even without stability guarantees, NODEs generated by a chunked hypernetwork (CHN) produce trajectories that are much more robust to novel initial conditions than the upper-baseline SG, which uses separate networks to learn each individual task (refer to Appendix A.7 for details).

Further interesting research directions in continual LfD may include: improving the ability of the chunked hypernetworks to remember long sequences of tasks (e.g., via architectural changes), using continual reinforcement learning [2] to refine predicted trajectories [22], and the inclusion of visual cues to improve and adapt trajectories [43].

7. Conclusion

In this paper, we presented the first study on continual learning of trajectories from demonstrations, thereby expanding the field of continual learning in robotics to LfD. We adapted continual learning (CL) methods from all CL families, and compared these against our hypernetwork-based approaches on three different datasets. Two of these three datasets were collected using kinesthetic teaching with the real robot and are released by us in this paper. We have also verified that our approach works for real-world tasks involving changing positions and orientations by evaluating it on a real robot. Compared to other continual learning approaches considered in this paper, hypernetworks achieve equal or better accuracy, but scale much better in terms of other criteria such as growth in model size, and storage of past training data. Our results suggest that a regular hypernetwork (HN) is capable of learning many LfD tasks and is empirically the method of choice for a long sequence of tasks. For a shorter task sequence, the smaller chunked hypernetwork (CHN) is empirically the best choice for continual-LfD.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to our open source git repository containing our code and datasets has been shared in the paper.

Appendix A

A.1. Hardware setup

We run all our experiments on a shared computing cluster with 125 GB of RAM and 16 nodes, with each node having 1 AMD Ryzen 2950X 16-Core processor and 4 GeForce RTX 2070 GPUs. Each experiment uses a limited amount of RAM and only a single GPU and can be easily run on a single GPU or even on a CPU-only system.

A.2. Training times

In Table A.1, we show the time (in seconds) needed to train each of the 7 continual learning methods presented in this paper (median values over 5 independent seeds). The hypernetwork-based methods (HN and CHN) show a small gradual increase in the training time as the number of tasks increase. This is due to the extra regularization that needs to be performed to prevent catastrophic forgetting. A possible way to attain a near constant training time (as proposed by von Oswald et al. [5]) is to randomly sample a fixed set of task embedding vectors to regularize with instead of using the embeddings of all tasks in each training iteration. We do not apply this process in this paper, but leave it for future work.

A.3. Inference times

The time taken by a network to produce predictions is critically important in a robotics scenario. The hypernetwork-based solutions for *learning from demonstration* are well suited for this task, as can be seen by the speed of making predictions reported in Table A.2. For this evaluation, the models which were originally trained on the GPU, were loaded either into GPU memory (Table A.2 (a)) or into the CPU memory (Table A.2 (b)). After loading the hypernetworks (HN or CHN), one of the task embedding vectors was used as an input to *generate* the parameters for a NODE and the speed of this forward pass is noted (column $\mathbf{f_h} \rightarrow \mathbf{f_\theta}$ in Table A.2 (a) and (b)). Note that this NODE generation step occurs only when the robot is required to perform a new task (task switch). Once a NODE for a task is generated, it can be reused for multiple predictions of that task. The generated NODE is then made to predict a trajectory and the speed of each step (column $\mathbf{f_\theta}(\mathbf{x}) = \dot{\mathbf{x}}$), as well as the time to integrate over a 1000 steps (column $\int \mathbf{f_\theta}(\mathbf{x}_\tau)d\tau$) is noted. This process is repeated 100 times and the median times and speeds are reported in milliseconds and Hz. As expected, using the GPU generally results in faster predictions. The time taken to integrate over the entire trajectory of 1000 steps is more than a thousand times the time for a single step (due to the integration operation) and in the majority of cases, this takes much less than 1 s. It can be seen from Table A.2 that hypernetworks are able to produce predictions very fast on both the GPU as well as the CPU. This makes them suitable for use in a robotics application.

A.4. Hyperparameters

The hyperparameters used for the LASA, HelloWorld and RoboTasks datasets are listed in Tables A.3, A.4 and A.5 respectively.

Table A.1

Wall clock training times (in seconds) of the models for continual learning from demonstration (median over 5 independent seeds).

(a) LASA (15×10^3 iterations/task)

Method	Task 0	Task 5	Task 9	Task 13	Task 17	Task 21	Task 25
SG	586	691	714	728	677	721	737
FT	665	722	767	752	673	747	735
REP	792	909	900	913	904	934	936
SI	694	699	707	702	647	713	686
MAS	603	705	753	697	712	723	705
HN	620	1087	1186	1253	1305	1510	1609
CHN	604	991	1119	1257	1384	1624	1743

(b) HelloWorld (40×10^3 iterations/task)

Method	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
SG	1896	2012	1872	1960	2030	2030	2281
FT	1954	2022	1973	2016	2050	2086	2385
REP	1926	1939	1950	1942	1957	1975	2016
SI	2019	2128	2093	2091	2110	2132	2268
MAS	1965	2281	2252	2302	2316	2362	2500
HN	1968	2337	2508	2702	2843	2930	3234
CHN	1948	2254	2341	2578	2786	2858	3150

(c) RoboTasks-position (50×10^3 iterations/task)

Method	Task 0	Task 1	Task 2	Task 3
SG	1869	2400	2328	1684
FT	1483	1985	1923	1732
REP	1486	1952	1901	1797
HN	1508	2387	2585	2214
CHN	1591	2484	2662	2141

(d) RoboTasks-orientation (50×10^3 iterations/task)

Method	Task 0	Task 1	Task 2	Task 3
SG	1809	1800	1800	1783
FT	2026	2068	2061	2058
REP	1396	1409	1409	1409
HN	1461	1903	2116	2303
CHN	1504	1840	2204	2511

Table A.2

Inference speeds of trained hypernetworks (HN and CHN) in milliseconds and Hz (median over 100 forward passes). $\mathbf{f_h} \to \mathbf{f_\theta}$ denotes the generation of a NODE by a hypernetwork, $\mathbf{f_\theta(x)} = \mathbf{\dot{x}}$ denotes a single step taken by the generated NODE, and $\int \mathbf{f_\theta(x_\tau)}d\tau$ denotes the generation of an entire trajectory from a NODE by integrating over 1000 steps.

(a) GPU

Dataset	Method	$\mathbf{f_h} \to \mathbf{f_\theta}$		$\mathbf{f_\theta(x)} = \mathbf{\dot{x}}$		$\int \mathbf{f_\theta(x_\tau)}d\tau$	
		(ms)	(Hz)	(ms)	(Hz)	(ms)	(Hz)
LASA	HN	0.73	1373.83	0.27	3765.08	584.27	1.71
LASA	CHN	0.61	1642.25	0.27	3771.86	569.71	1.76
HelloWorld	HN	0.72	1391.15	0.26	3856.83	562.79	1.78
HelloWorld	CHN	0.60	1662.43	0.27	3731.59	569.51	1.76
RoboTasks (pos)	HN	0.74	1360.24	0.27	3761.71	570.53	1.75
RoboTasks (pos)	CHN	0.60	1660.45	0.27	3728.27	570.28	1.75
RoboTasks (ori)	HN	0.74	1348.00	0.27	3685.68	595.44	1.68
RoboTasks (ori)	CHN	0.63	1597.83	0.28	3548.48	574.78	1.74

(b) CPU

Dataset	Method	$\mathbf{f_h} \to \mathbf{f_\theta}$		$\mathbf{f_\theta(x)} = \mathbf{\dot{x}}$		$\int \mathbf{f_\theta(x_\tau)}d\tau$	
		(ms)	(Hz)	(ms)	(Hz)	(ms)	(Hz)
LASA	HN	3.01	332.56	0.12	8422.30	261.13	3.83
LASA	CHN	5.23	191.04	0.61	1634.89	730.18	1.37
HelloWorld	HN	2.94	339.99	0.12	8456.26	262.61	3.81
HelloWorld	CHN	5.29	189.14	0.66	1511.73	750.89	1.33
RoboTasks (pos)	HN	2.87	348.16	0.17	5833.52	257.85	3.88
RoboTasks (pos)	CHN	5.78	172.94	0.57	1749.81	738.15	1.35
RoboTasks (ori)	HN	7.51	133.10	0.15	6579.30	265.23	3.77
RoboTasks (ori)	CHN	17.68	56.55	1.22	820.88	1499.81	0.67

Table A.3
Hyperparameters for the LASA dataset.

Hyperparameter	SG	FT	REP	SI	MAS	HN	CHN
Train iterations	15 000	15 000	15 000	15 000	15 000	15 000	15 000
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
NODE output dim.	2	2	2	2	2	2	2
NODE hidden layers	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[100] \times 3$	$[1000] \times 3$
NODE activation	elu	elu	elu	elu	elu	elu	elu
Task Emb. dim.	–	256	256	256	256	256	256
c (SI)	–	–	–	0.3	–	–	–
ξ (SI)	–	–	–	0.3	–	–	–
λ (MAS)	–	–	–	–	0.1	–	–
HN hidden layers	–	–	–	–	–	$[200] \times 3$	$[200] \times 3$
β (HN)	–	–	–	–	–	0.005	0.005
Chunk Emb. dim.	–	–	–	–	–	–	256
Chunk dim.	–	–	–	–	–	–	8192

Table A.4
Hyperparameters for the HelloWorld dataset.

Hyperparameter	SG	FT	REP	SI	MAS	HN	CHN
Train iterations	40 000	40 000	40 000	40 000	40 000	40 000	40 000
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
NODE output dim.	2	2	2	2	2	2	2
NODE hidden layers	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[100] \times 3$	$[1000] \times 3$
NODE activation	elu	elu	elu	elu	elu	elu	elu
Task Emb. dim.	–	256	256	256	256	256	256
c (SI)	–	–	–	0.3	–	–	–
ξ (SI)	–	–	–	0.3	–	–	–
λ (MAS)	–	–	–	–	0.1	–	–
HN hidden layers	–	–	–	–	–	$[200] \times 3$	$[200] \times 3$
β (HN)	–	–	–	–	–	0.005	0.005
Chunk Emb. dim.	–	–	–	–	–	–	256
Chunk dim.	–	–	–	–	–	–	8192

Table A.5
Hyperparameters for the RoboTasks dataset.

(a) Position						(b) Orientation					
Hyperparameter	SG	FT	REP	HN	CHN	Hyperparameter	SG	FT	REP	HN	CHN
Train iterations	50 000	50 000	50 000	50 000	50 000	Train iterations	50 000	50 000	50 000	50 000	50 000
Learning rate	0.00001	0.00001	0.00001	0.00001	0.00001	Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001
NODE output dim.	3	3	3	3	3	NODE output dim.	3	3	3	3	3
NODE hidden layers	$[1000] \times 3$	$[1000] \times 3$	$[1000] \times 3$	$[100] \times 3$	$[1000] \times 3$	NODE hidden layers	$[1500] \times 3$	$[1500] \times 3$	$[1500] \times 3$	150,150,150	$[1500] \times 3$
NODE activation	elu	elu	elu	elu	elu	NODE activation	elu	elu	elu	elu	elu
Task Emb. dim.	–	512	512	512	512	Task Emb. dim.	–	1024	1024	1024	1024
HN hidden layers	–	–	–	$[200] \times 3$	$[200] \times 3$	HN hidden layers	–	–	–	300,300,300	300,300,300
β (HN)	–	–	–	0.0005	0.0005	β (HN)	–	–	–	0.0005	0.0005
Chunk Emb. dim.	–	–	–	–	512	Chunk Emb. dim.	–	–	–	–	1024
Chunk dim.	–	–	–	–	8192	Chunk dim.	–	–	–	–	8192

The regularization hyperparameters for SI (c, ξ), MAS (λ) and HN/CHN (β) are based on values proposed by [4,5,30] respectively. Earlier, in Section 5.3 we have shown that HN and CHN are robust to changes in regularization hyperparameters. In all cases, NODEs use the smooth ELU activation function, and hypernetworks have ReLU activations. We use the Adam optimizer in all our experiments.

A.5. Imitation flow comparison

To compare NODEs against an alternative trajectory learning approach, we performed a small non-continual learning experiment using the LASA dataset. For each task, we trained a model of iFlow [9], NODE^I, and NODE^T and measured the DTW errors (Fig. A.1 shows the errors for all tasks taken together). The iFlow hyperparameters are shown in Fig. A.1 (right) and the NODE hyperparameters are given in Table A.3. Although for iFlow we used almost twice the number of training iterations as NODEs, the empirical performance of NODEs is still better. An iFlow model needs around 60 min to be trained to convergence for a single LASA task on our setup, whereas a NODE takes around 10 min.
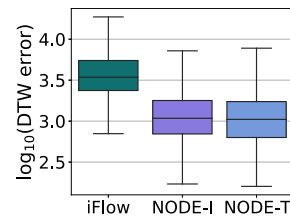
Hyperparameter	iFlow
Batch size	256
Depth	7
Learning rate	0.0001
Epochs	1000
Train iterations	27000
Output dim.	2

Fig. A.1. Comparison of NODE with iFlow. (left) DTW errors (lower is better) of all trajectory predictions for all tasks of the LASA dataset. Standalone models of iFlow, NODE^T, and NODE^I are used to learn each of the 26 LASA tasks. Results are obtained using 5 independent seeds. (right) Hyperparameters for iFlow. Hyperparameters for NODE are given in Table A.3. NODEs show better empirical performance and converge faster than iFlow.

A.6. Additional results

Figs. A.2 and A.3 report additional trajectory error metrics (*Frechet distance* and *Swept Area error*) for the LASA and HelloWorld datasets respectively. Fig. A.4 shows a qualitative example of orientation prediction in terms of quaternions for the RoboTasks dataset.
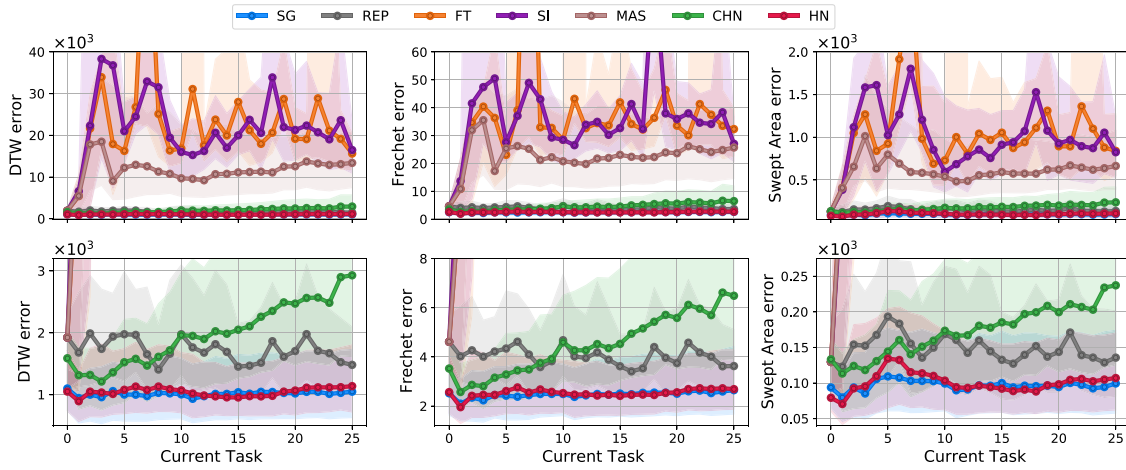
Fig. A.2. DTW, Frechet distance and Swept Area errors of trajectories predicted for the LASA dataset (lower is better). The *x*-axis shows the *current* task. After learning a task (using NODE^T), all current and previous tasks are evaluated. Plots for SG and HN overlap with each other. Lines show medians and shaded regions denote the lower and upper quartiles of the errors over 5 independent seeds. (top) The errors for all methods. (bottom) A zoomed-in view of the methods that perform well.
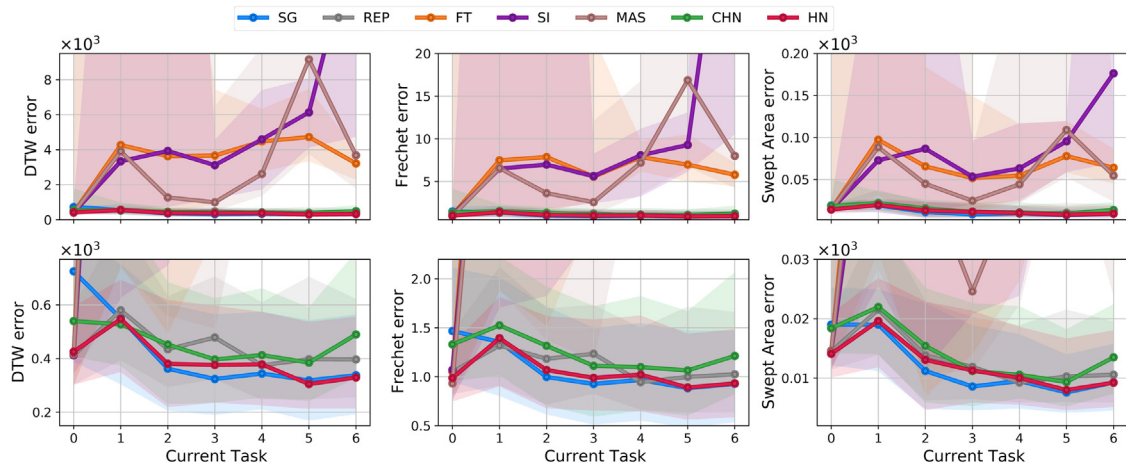

Fig. A.3. DTW, Frechet distance and Swept Area errors of trajectories predicted for the HelloWorld dataset (lower is better). The *x*-axis shows the *current* task. After learning a task (using NODE^T), all current and previous tasks are evaluated. Lines show medians and shaded regions denote the lower and upper quartiles of the errors over 5 independent seeds. (top) The errors for all methods. (bottom) A zoomed-in view of the methods that perform well.
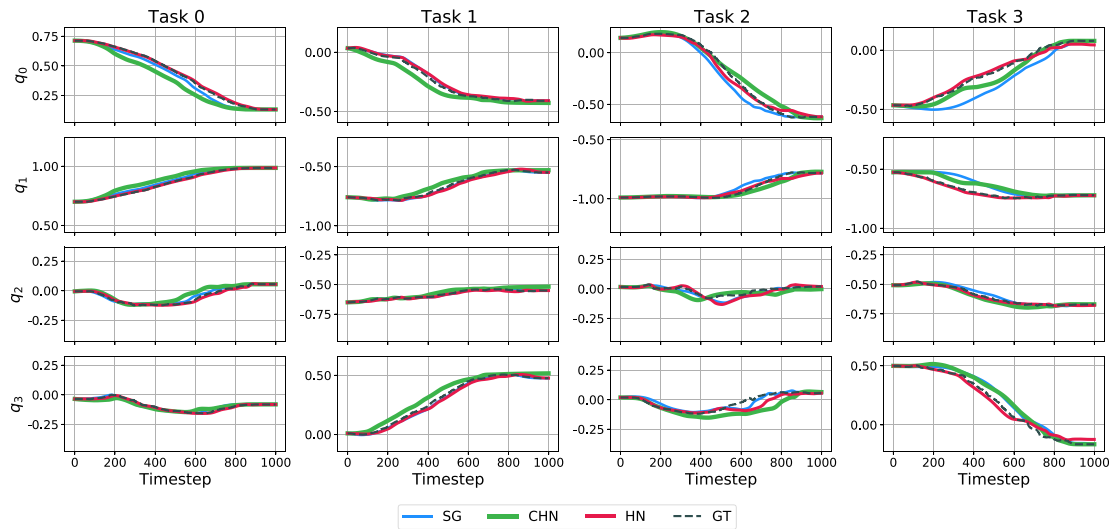

Fig. A.4. Example of end-effector orientation trajectories predicted for all past tasks after sequentially learning all the tasks of the RoboTasks dataset. All methods employ NODE^T as the trajectory learning method. The ground truth demonstrations as well as the predictions of the orientation consist of a sequence of unit quaternions. Columns show the predictions for each task, and rows show the different elements of the quaternions (scalar first convention).
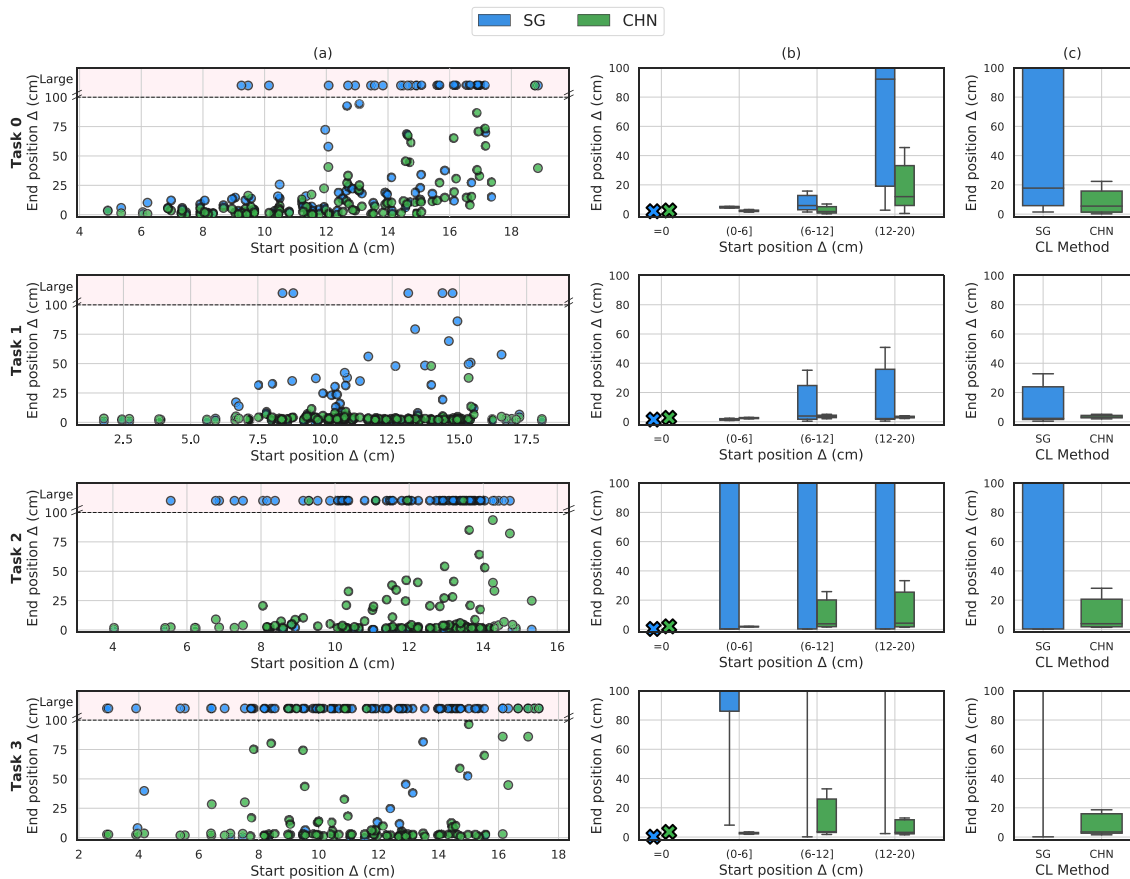
17

Fig. A.5. Robustness of SG and CHN on the RoboTasks dataset. Each row shows the results for 1 of the 4 tasks of this dataset. Columns (a,b) show that for small values of *start position* Δ, SG's trajectories often diverge far from the goal position but CHN's trajectories do not. For larger values of *start position* Δ, CHN gets worse but much less than SG. In column (a), values greater than *end position* $\Delta = 100$ cm are grouped together and shown in the shaded region marked "Large". Column (b) shows the divergence of SG and CHN by grouping the results based on the *start position* Δ. Here, the symbol $\times$ shows the value of *end position* Δ when *start position* $\Delta = 0$. Column (c) shows overall divergence of SG and CHN for novel initial positions, where CHN shows much less divergence. All models use NODE-T for learning.

A.7. Robustness to starting position

To test the robustness of the different CL methods to changes in the initial positions of the robot (i.e., when the robot starts at a position that is different from the start position of the demonstrated trajectory), we perform an analysis using the RoboTasks dataset.

For each of the four tasks of this dataset, we randomly select one ground truth demonstration. We then create a sphere with a radius of 20 cm (the maximum reach of the Franka Emika Panda robot is 85.5 cm) centered at the starting position of the ground truth trajectory. From within the volume of this sphere, we uniformly sample 100 random starting points. We then take the trained models of SG and CHN (trained on all the tasks of the dataset), and use these to predict trajectories for each of the 4 tasks and each of the 100 random starting points with the same timesteps as the ground truth demonstration. By sampling points from a sphere of radius 20 cm, we are asking the models to extrapolate from the demonstrations and start from areas for which no demonstration has been provided.

For each predicted trajectory, we measure (i) *start position* Δ: the distance between the random starting position and the starting position of the demonstration; (ii) *end position* Δ: the distance between the end point of the predicted trajectory (that starts from a random starting position) and the end position (goal)

of the demonstration. The results are shown in Fig. A.5. In the ideal case, the end position Δ would always be 0 irrespective of the start position Δ (every trajectory, irrespective of the starting position would always converge at the goal). In the absence of any stability guarantees, if the starting positions of the robot are very different from the ground truth, the predicted trajectory can be expected to diverge far from the ground truth goal, leading to a large end position Δ.

It can be seen that for all tasks, when the random starting position is up to 6 cm from the GT source, the CHN model produces trajectories that end up very close to the GT goal. This is not true for SG. Although SG performs almost perfectly when starting from the GT source position (see the $\times$ symbols in column (b) of Fig. A.5, and also Figs. 14 and 15), it produces trajectories that diverge far from the GT goal even when the random starting positions are quite close to the GT source position. For larger start position differences, the performance of CHN does degrade, but much less than that for SG, which most of the time produces trajectories that end up far beyond the reachable task space of the robot (in the scatter plots we club together all predictions that end up more than 100 cm from the goal). This shows that the continual learning process for CHN has a regularization effect which prevents the strong overfitting to demonstrations as exhibited by SG.
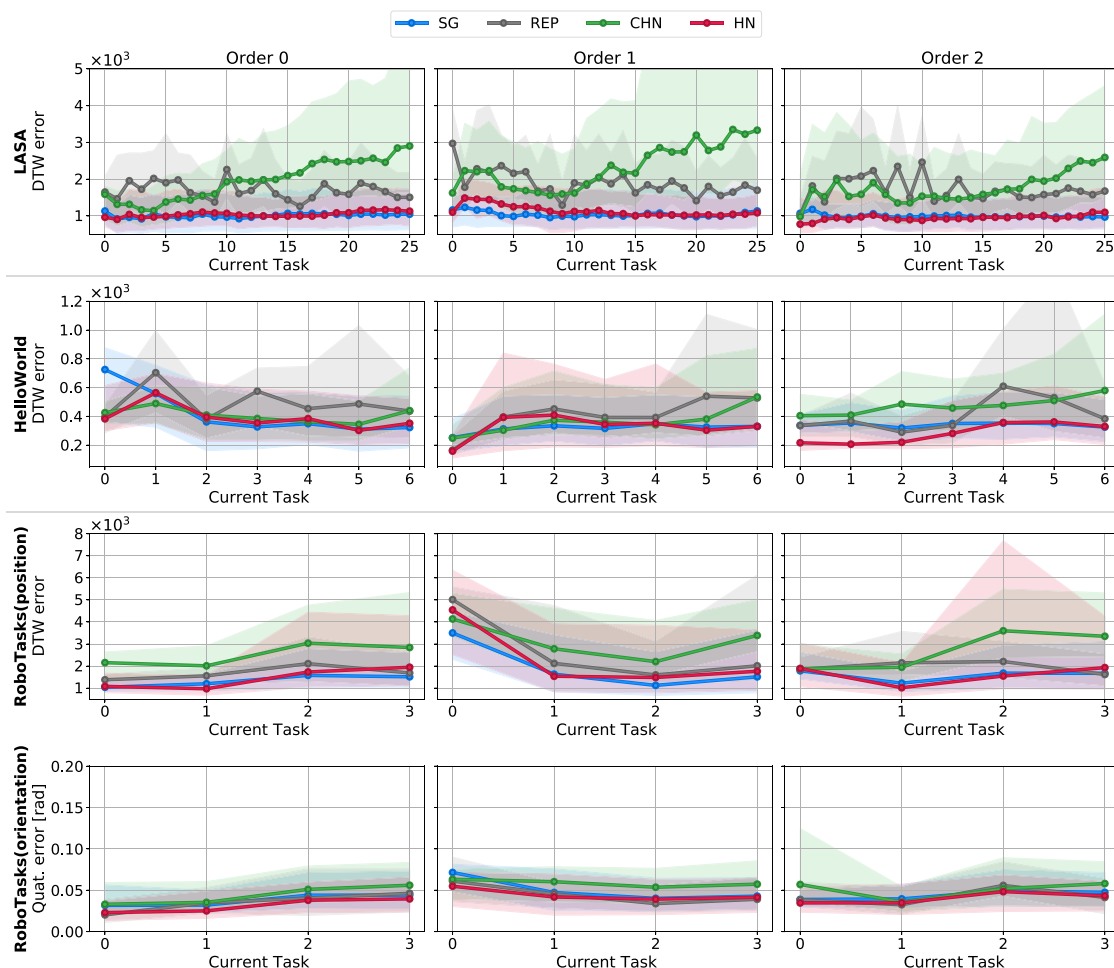
Fig. A.6. Robustness to task order during training. The models for SG, REP, CHN and HN are trained and evaluated with 3 different task orders for each dataset (each column of plots shows the result of 1 order). Overall the same trends are visible for all task orders. For LASA, HN is almost identical to SG, and CHN performs well for a small number of tasks but degrades afterwards. For HelloWorld, HN and CHN perform as well as the other baselines. For RoboTasks, again HN and CHN perform comparably to the other baselines. This shows that the continual learning performance is not affected by the order in which tasks are learned. The *x*-axis in the plots denotes the task number. To see the name of a task from a given order, please refer to Table A.6 (e.g., for LASA, task 0 from *Order 0* is 'Angle', while task 0 from *Order 1* is 'Spoon'). Also, note that the DTW metric is dependent on the dimension of the trajectory points. Thus, the DTW values for a 2D dataset such as HelloWorld are lower than those for RoboTasks. In all cases, the upper baseline SG provides a reference for comparison. For each model, each dataset and each task order, we train and evaluate with 3 independent seeds. Similar to Figs. 4, 9 and 14, in this plot lines show median values and shaded areas represent the IQR of the errors (lower is better).

A.8. Robustness to task order

We test the robustness of the best performing models (SG, REP, CHN and HN) against changes in the order in which tasks are presented during training. For each of the 3 datasets used in this paper, we create 3 different orders of tasks: *Order 0*, *Order 1*, and *Order 2*. The first order (Order 0) is the same as that used for the results reported in Section 5.3. The next 2 task orders for each dataset are created by randomly shuffling the first task order of the respective datasets. The task orders and the corresponding data file names used during training are listed in Table A.6.

We train and evaluate models for SG, REP, CHN, and HN (as they perform the best) each with 3 independent seeds on each task order of each dataset. The results for this evaluation are shown in Fig. A.6. Although the results are marginally different for the different task orders, the overall trends are similar across task orders for the LASA, HelloWorld and RoboTasks datasets.

For LASA (see the first row of plots in Fig. A.6), the performance of HN is almost identical to that of the upper baseline SG, while REP and CHN perform worse for all orders. CHN performs well up to a small number of tasks but its performance degrades as more tasks are learned.

The results for HelloWorld (second row of plots in Fig. A.6) also show similar trends for all 3 task orders: CHN performs similarly to SG, REP and HN, in spite of being much smaller in size than SG and HN, and without needing to store training data of past tasks like REP.

Similarly, for RoboTasks (last two rows of plots in Fig. A.6), the task order does not seem to affect the continual learning performance. Learning a difficult task is equally challenging for all the methods (e.g. the *plateslyingtostanding* task, which is task 2 in Order 0 and task 0 in Order 1, appears to be a difficult task judging from the relatively higher errors). However this does not affect the relative performance of the different methods. CHN performs almost equally well as the other methods, although other methods have more parameters (SG, HN) or store past training data (REP).

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2023.104427.

Table A.6
Task orders.

(a) LASA

Task ID	Order 0 file	Order 1 file	Order 2 file
0	Angle	Spoon	BendedLine
1	BendedLine	Sine	RShape
2	CShape	Zshape	Trapezoid
3	DoubleBendedLine	NShape	WShape
4	GShape	PShape	PShape
5	heee	heee	heee
6	JShape_2	Angle	Sshape
7	JShape	Khamesh	Line
8	Khamesh	Line	JShape
9	Leaf_1	LShape	DoubleBendedLine
10	Leaf_2	JShape_2	Sine
11	Line	WShape	GShape
12	LShape	Leaf_2	Sharpc
13	NShape	RShape	LShape
14	PShape	JShape	Saeghe
15	RShape	Sharpc	Khamesh
16	Saeghe	Snake	CShape
17	Sharpc	DoubleBendedLine	Zshape
18	Sine	Saeghe	JShape_2
19	Snake	Worm	Snake
20	Spoon	Leaf_1	Angle
21	Sshape	GShape	Worm
22	Trapezoid	BendedLine	Leaf_2
23	Worm	Sshape	Spoon
24	WShape	CShape	NShape
25	Zshape	Trapezoid	Leaf_1

(b) HelloWorld

Task ID	Order 0 file	Order 1 file	Order 2 file
0	h	l	o
1	e	d	w
2	l	h	l
3	o	o	h
4	w	w	e
5	r	r	d
6	d	e	r

(c) RoboTasks

Task ID	Order 0 file	Order 1 file	Order 2 file
0	openbox	platestandingtolying	pouring
1	bottle2shelf	openbox	openbox
2	platestandingtolying	bottle2shelf	platestandingtolying
3	pouring	pouring	bottle2shelf

References

[1] C. Gao, H. Gao, S. Guo, T. Zhang, F. Chen, CRIL: Continual robot imitation learning via generative and prediction model, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 6747–5754, http://dx.doi.org/10.1109/IROS51168.2021.9636069.

[2] Y. Huang, K. Xie, H. Bharadhwaj, F. Shkurti, Continual model-based reinforcement learning with hypernetworks, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 799–805.

[3] H. Shin, J.K. Lee, J. Kim, J. Kim, Continual learning with deep generative replay, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 2994–3003.

[4] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, T. Tuytelaars, Memory aware synapses: Learning what (not) to forget, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 139–154.

[5] J. von Oswald, C. Henning, J. Sacramento, B.F. Grewe, Continual learning with hypernetworks, in: International Conference on Learning Representations (ICLR), 2019.

[6] A. Billard, S. Calinon, R. Dillmann, Learning from humans, in: Springer Handbook of Robotics, second ed., 2016.

[7] M. Hersch, F. Guenter, S. Calinon, A. Billard, Dynamical system modulation for robot learning via kinesthetic demonstrations, IEEE Trans. Robot. 24 (6) (2008) 1463–1467.

[8] S.M. Khansari-Zadeh, A. Billard, Learning stable nonlinear dynamical systems with Gaussian mixture models, IEEE Trans. Robot. 27 (5) (2011) 943–957.

[9] J. Urain, M. Ginesi, D. Tateo, J. Peters, ImitationFlow: Learning deep stable stochastic dynamic systems by normalizing flows, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 5231–5237.

[10] J.Z. Kolter, G. Manek, Learning stable deep dynamics models, Advances in Neural Information Processing Systems 32 (2019) 11128–11136.

[11] A.J. Ijspeert, J. Nakanishi, S. Schaal, Movement imitation with nonlinear dynamical systems in humanoid robots, in: International Conference on Robotics and Automation (ICRA), 2002, pp. 1398–1403.

[12] M. Saveriano, F.J. Abu-Dakka, A. Kramberger, L. Peternel, Dynamic movement primitives in robotics: A tutorial survey, 2021, arXiv preprint arXiv:2102.03861.

[13] D. Ha, A.M. Dai, Q.V. Le, HyperNetworks, in: International Conference on Learning Representations, 2017, URL https://openreview.net/forum?id=rkpACe1lx.

[14] R.T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 6572–6583.

[15] G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: A review, Neural Netw. 113 (2019) 54–71.

[16] S. Calinon, Learning from demonstration (programming by demonstration), Encyclopedia Robot. (2018) 1–8.

[17] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (5) (2009) 469–483.

[18] H. Ravichandar, A. Polydoros, S. Chernova, A. Billard, Robot learning from demonstration: A review of recent advances, Annu. Rev. Control Robot. Auton. Syst. (2019).

[19] S.R. Ahmadzadeh, S. Chernova, Trajectory-based skill learning using generalized cylinders, Front. Robot. AI 5 (2018) 132.

[20] Y. Wu, Y. Demiris, Towards one shot learning by imitation for humanoid robots, in: 2010 IEEE International Conference on Robotics and Automation, IEEE, 2010, pp. 2889–2894.

[21] B.D. Argall, B. Browning, M.M. Veloso, Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot, Robot. Auton. Syst. 59 (3–4) (2011) 243–255.

[22] S. Calinon, P. Kormushev, D.G. Caldwell, Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning, Robot. Auton. Syst. 61 (4) (2013) 369–379.

[23] N. Das, S. Bechtle, T. Davchev, D. Jayaraman, A. Rai, F. Meier, Model-based inverse reinforcement learning from visual demonstrations, in: Conference on Robot Learning, PMLR, 2021, pp. 1930–1942.

[24] P. Englert, N.A. Vien, M. Toussaint, Inverse KKT: Learning cost functions of manipulation tasks from demonstrations, Int. J. Robot. Res. 36 (13–14) (2017) 1474–1488.

[25] A. Kalinowska, A. Prabhakar, K. Fitzsimons, T. Murphey, Ergodic imitation: Learning from what to do and what not to do, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 3648–3654.

[26] S.M. Khansari-Zadeh, A. Billard, Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions, Robot. Auton. Syst. 62 (6) (2014) 752–765.

[27] M. Saveriano, An energy-based approach to ensure the stability of learned dynamical systems, in: IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 4407–4413.

[28] A.A. Rusu, N.C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, R. Hadsell, Progressive neural networks, 2016, arXiv preprint arXiv:1606.04671.

[29] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, C.H. Lampert, Icarl: Incremental classifier and representation learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2001–2010.

[30] F. Zenke, B. Poole, S. Ganguli, Continual learning through synaptic intelligence, in: International Conference on Machine Learning, PMLR, 2017, pp. 3987–3995.

[31] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, T. Tuytelaars, A continual learning survey: Defying forgetting in classification tasks, IEEE Trans. Pattern Anal. Mach. Intell. (2021).

[32] A. Xie, C. Finn, Lifelong robotic reinforcement learning by retaining experiences, 2021, arXiv preprint arXiv:2109.09180.

[33] M. Heinonen, C. Yildiz, H. Mannerström, J. Intosalmi, H. Lähdesmäki, Learning unknown ODE models with Gaussian processes, in: International Conference on Machine Learning, PMLR, 2018, pp. 1959–1968.

[34] Y. Huang, F.J. Abu-Dakka, J. Silvério, D.G. Caldwell, Toward orientation learning and adaptation in cartesian space, IEEE Trans. Robot. 37 (1) (2020) 82–98.

[35] A. Ude, B. Nemec, T. Petrić, J. Morimoto, Orientation in cartesian space dynamic movement primitives, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2014, pp. 2997–3004.

[36] M. Saveriano, F. Franzel, D. Lee, Merging position and orientation motion primitives, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 7041–7047.

[37] H.C. Ravichandar, I. Salehi, A.P. Dani, Learning partially contracting dynamical systems from demonstrations, in: CoRL, 2017, pp. 369–378.

[38] C. Blocher, M. Saveriano, D. Lee, Learning stable dynamical systems using contraction theory, in: 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), IEEE, 2017, pp. 124–129.

[39] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, C. Finn, RoboNet: Large-scale multi-robot learning, 2019, http://dx.doi.org/10.48550/ARXIV.1910.11215, URL https://arxiv.org/abs/1910.11215.

[40] T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, S. Levine, Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2019, http://dx.doi.org/10.48550/ARXIV.1910.10897, URL https://arxiv.org/abs/1910.10897.

[41] C.F. Jekel, G. Venter, M.P. Venter, N. Stander, R.T. Haftka, Similarity measures for identifying material parameters from hysteresis loops using inverse analysis, Int. J. Mater. Form. 12 (3) (2019) 355–378.

[42] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, D. Maltoni, Don't forget, there is more than forgetting: new metrics for Continual Learning, 2018, arXiv preprint arXiv:1810.13166.

[43] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, S. Levine, Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 3758–3765.

Sayantan Auddy is a Ph.D. student in the Intelligent and Interactive Systems group of the department of Computer Science at the University of Innsbruck (Austria). He received his M.Sc. degree in Intelligent Adaptive Systems from the University of Hamburg (Germany) in 2018, and his B.Tech. degree in Computer Science and Engineering from Haldia Institute of Technology (India) in 2009. Prior to starting his masters studies, he has worked as a Software Engineer in the Information Technology industry for 5 years. His research interests include continual learning for robotics, reinforcement learning and deep learning.

Jakob Hollenstein is a Ph.D. student in the Intelligent and Interactive Systems group of the department of Computer Science at the University of Innsbruck (Austria). He received his B.Sc. degree in Computer Engineering and his Dipl.-Ing. degree in Computational Intelligence from the Vienna University of Technology (Austria) in 2010 and 2014. He worked as a software engineer in the Information Technology industry for 3 years before joining the Intelligent Interactive Systems group. His research interests include robot learning, deep learning and reinforcement learning.

Matteo Saveriano received his B.Sc. and M.Sc. degree in automatic control engineering from University of Naples, Italy, in 2008 and 2011, respectively. He received his Ph.D. from the Technical University of Munich in 2017. Currently, he is an assistant professor at the Department of Industrial Engineering (DII), University of Trento, Italy. Previously, he was an assistant professor at the University of Innsbruck and a post-doctoral researcher at the German Aerospace Center (DLR). He is an Associate Editor for RA-L. His research activities include robot learning, human–robot interaction, understanding and interpreting human activities. Webpage: https://matteosaveriano.weebly.com/

Antonio José Rodríguez Sánchez is currently an Associate Professor in the Intelligent and Interactive Systems group of the department of Computer Science at the Universität Innsbruck (Austria). He completed his Ph.D. at the Center for Vision Research (York University, Toronto, Canada) on modeling attention and intermediate areas of the visual cortex under the supervision of Prof. John K. Tsotsos (Canada Research Chair) in 2010. He obtained the degree of M.Sc. in Computer Science at the Universidade da Coruña (Spain) in 1998. He received his B.Sc. in Computer Science at Universidad de Córdoba (Spain) with Honors and did his Bachelor Thesis at the Université de La Rochelle (France). His current research interests include different areas of artificial intelligence: Explainable AI, computational neuroscience, (deep) neural networks, computer vision, machine learning and robotics.

Justus Piater is a professor of computer science at the University of Innsbruck, Austria, where he leads the Intelligent and Interactive Systems group and serves as the founding director of the interdisciplinary Digital Science Center. He earned his Ph.D. degree at the University of Massachusetts Amherst, USA, and was a professor of computer science at the University of Liège, Belgium, and a visiting researcher at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany. His research interests focus on learning and inference in sensorimotor systems. He has published more than 200 papers, several of which have received best-paper awards.