Continual Model-Based Reinforcement Learning with Hypernetworks

Yizhou Huang **Kevin Xie** **Homanga Bharadhwaj** **Florian Shkurti**
Department of Computer Science
University of Toronto, Canada
University of Toronto Robotics Institute
philipyizhou.huang@mail.utoronto.ca
{kevincxie, homanga, florian}@cs.toronto.edu

Abstract

Effective planning in model-based reinforcement learning (MBRL) and model-predictive control (MPC) relies on the accuracy of the learned dynamics model. In many instances of MBRL and MPC, this model is assumed to be stationary and is periodically re-trained from scratch on state transition experience collected from the beginning of environment interactions. This implies that the time required to train the dynamics model – and the pause required between plan executions – grows linearly with the size of the collected experience. We argue that this is too slow for lifelong robot learning and propose `HyperCRL`, a method that continually learns the encountered dynamics in a sequence of tasks using task-conditional hypernetworks. Our method has three main attributes: first, it enables constant-time dynamics learning sessions between planning and only needs to store the most recent fixed-size portion of the state transition experience; second, it uses fixed-capacity hypernetworks to represent non-stationary and task-aware dynamics; third, it outperforms existing continual learning alternatives that rely on fixed-capacity networks, and does competitively with baselines that remember an ever increasing coreset of past experience. We show that `HyperCRL` is effective in continual model-based reinforcement learning in robot locomotion and manipulation scenarios, such as tasks involving pushing and door opening. Our project website with videos is at this link `rvl.cs.toronto.edu/blog/2020/hypercrl/`

1 Introduction

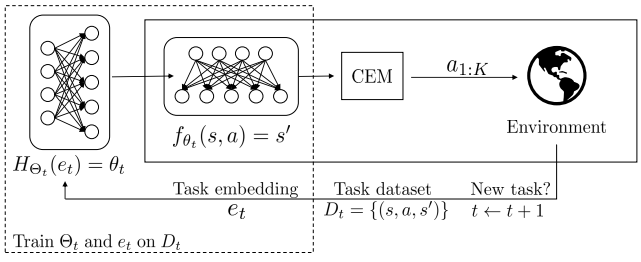


Figure 1: Overview of our proposed solution

Lifelong model-based robot learning is predicated upon continual adaptation to the dynamics of new tasks. For example, robots need to learn to manipulate unseen objects with various mass distributions, walk on new types of terrains with different friction, elasticity, and other physical properties, or even learn to adapt to different tasks, such as walking, running, or climbing stairs. This presents at least two challenges for many model-based reinforcement learning (MBRL) and model-predictive control (MPC) formulations, which typically comprise of a dynamics learning phase followed by a planning/policy optimization and execution phase. First, these methods are not scalable because the time required to train the dynamics model grows linearly with the size of the collected experi-

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

ence. Second, as the robot learner encounters and adapts to new tasks, it has to avoid catastrophic forgetting of the dynamics of old tasks.

In this work, we propose to extend the task-aware continual learning approach based on hypernetworks in [1] to adapt to changing environment dynamics and to address the scalability and catastrophic forgetting challenges mentioned above in a reinforcement learning setting. We use task-conditional hypernetworks, which are neural network models that accept a learned task encoding as an input, and output the weights of another (target) network. In our case, the output is the dynamics model for that task. No additional information other than task transition boundary is needed. We show that continual learning with hypernetworks leads to effective model-based reinforcement learning, while having constant-time dynamics model updates and preventing catastrophic forgetting.

We consider the setting where task boundaries are known in order to simplify the problem, although there are continual learning methods that have addressed the task-agnostic setting using Bayesian non-parametric methods [2]. In addition, we focus on fixed-capacity hypernetworks and target networks that can handle a sequence of tasks without adding new neurons or layers to the network, unlike many related works [3, 4] in which every new task adds capacity to the dynamics model. We argue that the fixed-capacity setting, together with the constant-time dynamics learning session, is more realistic and scalable for lifelong robot learning applications compared to approaches in which training time or the size of the weights scale linearly with the size of collected experience.

Our work makes the following contributions: we show that task-aware continual learning with hypernetworks is an effective and practical way to adapt to new tasks and changing dynamics for model-based reinforcement learning without keeping state transitions from old tasks nor adding capacity to the dynamics model. We evaluate our method on locomotion and manipulation scenarios, where we show that our method outperforms related continual learning baselines.

2 Related Works

Continual Learning of Neural Networks Continual learning studies the problem of incrementally learning from a sequential stream of data with only a small portion of the data available at once [5]. A simple yet effective approach is finetuning, which directly tunes the trained source task network on the target task [6]. The efficacy of this approach for continual learning suffers from the well-established phenomenon of catastrophic forgetting [4]. Sequential Bayesian posterior updates are a principled way to perform continual learning and naturally avoids the forgetting problem since the exact posterior fully incorporates all previous data but in practice approximations have to be made that may be prone to forgetting. Elastic Weight Consolidation (EWC) uses a Laplace approximation to the posterior, storing previous tasks' empirical fisher matrices and regularizing future task weight deviations under their induced norms [7]. Other works have also employed variational mean-field approximations [8] or block-diagonal Kronecker factored Laplace approximations [9]. Synaptic Intelligence (SI) forgoes an obvious approximate Bayesian interpretation but operates similarly to EWC in that it computes a relative parameter importance measure, but through a linear approximation to the contribution in loss reduction due to each parameter on previous tasks [10]. Coreset methods prevent catastrophic forgetting by choosing and storing a significantly smaller subset of the previous task's data, which is used to rehearse the model during or after finetuning [11, 12, 13]. Similarly, the inducing points used in sparse Gaussian Process (GP) formulations, which can be seen as a type of coreset, has been used for continual learning [14, 15]. Another type of approach learns separate task-specific network components. The most common version of this are multi-head networks that learn and switch between separate output layers depending on the task [16]. Progressive Neural Networks (PNN) [4] are an extreme version of this approach, in which an entirely new copy of the network is appended for each task, thereby eliminating any forgetting. These methods can incur significant memory and compute cost especially for larger models and many tasks.

Model-based RL Model-based reinforcement learning approaches incorporate model learning of the environment in solving the control task. Traditionally, the model is trained to approximate the stationary dynamics and/or reward of the environment from all collected samples. Various choices for the model have been proposed. Many non-parametric models, such as commonly used GPs, rely on storing and making inferences with past data [17], although in many cases the amount of data needed to be stored can be drastically reduced through sparse variational inference [18]. Nonlinear

parametric models typically do not admit efficient sequential update rules and therefore usually train on all past data as well [19]. Typically, the trained model is then used to simulate imagined trajectories, either for the purpose of online planning [20, 21] or as training data for an amortized policy [22]. In non-stationary environments, the dynamics can change over time. In this setting, a lot of works focus on quickly adapting to the change in dynamics to minimize online regret, as opposed to retaining performance on previously experienced dynamics [23, 24, 25]. Meta-learning is a popular tool in this paradigm where a global dynamics model is "meta-trained" to quickly adapt to the true online dynamics from only a few samples. However, the meta-learning process typically requires updating the meta-model with data from a diverse set of dynamics that is obtained by storing previous experiences in a buffer and/or being able to simultaneously interact with many different environments [26].

Relationship to Meta-Learning Our work is different from meta-learning approaches in MBRL like [27, 28] in two ways. First, we focus on preventing catastrophic forgetting and do not explicitly train a model prior across multiple tasks for fast adaptation. This means that we do not need to design a set of tasks for meta-training and in principle, our work can continuously learn to perform new tasks from scratch. Second, we do not require the use of a replay buffer that grows linearly with the number of tasks or total length of collected state-transition pairs. We emphasize that this is consistent with the theme of continual learning, where storing past data is limited.

Continual RL Memory-efficient continual learning methods in the reinforcement learning setting have also been proposed. PNN was used in an on-policy actor-critic method and was demonstrated on sequential discrete action Atari games. The authors of [29] build on top of PNN and continual policy compression methods [30] by compressing the extended model from PNN into a fixed size network after each task. For the compression stage, they propose a more scalable online EWC algorithm that eschews the linear cost of storing past fisher matrices. The use of coresets has also been explored in this setting [31]. In [27], a mixture model of separate task-specific neural networks is used for the environment model that requires adding a new model each time a task is instantiated. A similar approach was also demonstrated using a mixture of GPs using an online clustering algorithm [32]. Our work is also related to MPC interpretations as a reduction to online learning [33].

Robust and Adaptive Control Existing literature on control theory provides many related classes of approaches that handle changes in dynamics: adaptive control methods handle unknown parameters of a dynamics model by estimating them over time so as to execute a given trajectory, and robust control methods provide stability guarantees as long as the parameter or model disturbance is within bounds [34]. The setting we study in this work differs in that we learn the dynamics model from scratch, without assuming a reference model, and it may change over tasks without imposing any bounds on particular parameters.

3 Preliminaries

Hypernetworks for Continual Learning A hypernetwork [35, 36] is a network that generates the weights of another neural network. The hypernetwork $H_\Theta(e) = \theta$ with weights Θ can be conditioned on an embedding vector e to output the weights θ of the main (target) network $f_\theta(x) = f(x;\theta) = f(x; H_\Theta(e))$ by varying the embedding vector e. The hypernetwork is typically smaller with respect to the number of trainable parameters in comparison to the main network. Hypernetworks have been shown to be useful in the continual learning setting [1] for classification and generative models. This has been shown to alleviate some of the issues of *catastrophic forgetting*. They have also been used to enable gradient-based hyperparameter optimization [37].

Planning with CEM and MPC The Cross-Entropy Method (CEM) [38] is a widely used online planning algorithm that samples action sequences from a time-evolving distribution which is usually considered to be a diagonal Gaussian $a_{1:h} \sim \mathcal{N}(\mu_{1:h}, \texttt{diag}(\sigma^2_{1:h}))$, where h is the planning horizon. Action sequences are iteratively re-sampled and evaluated under the currently learned dynamics model, and the sampling distribution parameters $\mu_{1:h}, \sigma_{1:h}$ are re-fitted to the top percentile of trajectories. CEM for planning in MBRL has been successfully used in a number of previous approaches [20, 21], as it alleviates exploitation of model bias compared to purely gradient based optimizations [39] and can better adapt to varying dynamics as compared to fully amortized policies [22].

4 The Proposed Approach

4.1 Problem Setting and Method Overview

We consider the following **problem setting**: A robot interacts with the environment to solve a sequence of T goal-directed tasks, each of which brings about different dynamics while having the same state-space $\mathcal{S}$ and action space $\mathcal{A}$. The robot is exposed to the tasks sequentially online without revisiting data collected in a previous task.

The robot also has finite memory and is not allowed to maintain a full history of state transitions for the purpose of re-training. Since the distribution of tasks changes over time and the agent does not know about it a priori, it must continually adapt to the streaming data of observations that it encounters, while trying to solve each task. The robot knows when a task switch occurs.

Algorithm 1: Continual Reinforcement Learning via Hypernetworks (`HyperCRL`)

Input: T tasks, each with its own dynamics $\mathcal{S} \times \mathcal{A} \to \mathcal{S}'$, reward $r(s, a)$. Learning rates α_Θ, α_e, and planning horizon h.
Randomly initialize hypernetwork weights Θ_1
for task t = 1, 2, ..., T **do**
  Initialize task-specific replay buffer $\mathcal{D}_t = \{\}$
  Randomly initialize task embedding e_t
  Collect P episodes of trajectories τ using a random policy and add it to $\mathcal{D}_t$
  for episode m = 1, 2, ..., M **do**
    (Optionally) Reset the environment; Observe s_0
    Generate target network weights $\theta_t = H_{\Theta_t}(e_t)$
    // `The dynamics model for the current episode is` $f_{\theta_t}(\cdot)$
    for step k = 1, 2, ..., K **do**
      Optimize action sequence $a_{k:k+h}$ using CEM with $f_{\theta_t}(\cdot)$ and known reward
      Execute first action a_k, observe next state s_{k+1}, add (s_k, a_k, s_{k+1}) to $\mathcal{D}_t$
    end for
    // `Update hypernetwork and task embedding`
    for s = 1, 2, ..., S **do**
      Sample a batch $\mathcal{B}$ of state-action pairs (s_k, a_k, s_{k+1}) from $\mathcal{D}_t$ and compute $\mathcal{L}_t$
      $\Theta_t \leftarrow \Theta_t - \alpha_\Theta \nabla_{\Theta_t} \mathcal{L}_t(\Theta_t, e_t)$
      $e_t \leftarrow e_t - \alpha_e \nabla_{e_t} \mathcal{L}_t(\Theta_t, e_t)$
    end for
  end for
  $\Theta_{t+1} = \Theta_t$
end for

We consider the **solution setting** of MBRL with a learned dynamics model, the parameters of which are inferred through a task-conditioned hypernetwork. Given learned task embeddings e_t and parameters Θ_t of the hypernetwork $H(\cdot)$, we infer parameters θ_t of the dynamics neural network $f_{\theta_t}(\cdot)$. Using this dynamics model, we perform CEM optimization to generate action sequences and execute them in the environment for K time-steps with MPC. We store the observed transitions in the replay dataset and update the parameters of the hypernetwork Θ_t and task-embeddings e_t (off-policy optimization). We repeat this for M episodes per task, and for each of the T tasks sequentially.

4.2 Training Procedure

Dynamics Learning The learned dynamics model is a feed-forward neural network whose parameters vary across tasks. One way to learn a dynamics network $f_\theta(\cdot)$ across tasks is to update it sequentially as training progresses. However, since our problem setting is such that the agent is not allowed to retain state-transition data from previous tasks in the replay buffer, adapting the weights of a single network sequentially across tasks is likely to lead to catastrophic forgetting [1]. In order to alleviate issues of catastrophic forgetting while trying to adapt the weights of the network, we learn a hypernetwork that takes task embeddings as input, and outputs parameters for the dynamics network corresponding to every task, learning different dynamics networks $f_{\theta_t}(\cdot)$ for each task t.

We assume that the agent has finite memory and does not have access to state-transition data across tasks. So, the task-specific replay buffer $\mathcal{D}_t$ is reset at the start of every task t. For the current

episode, the agent generates a dynamics network f_{θ_t} using $\theta_t = H_{\Theta_t}(e_t)$. Then, for $k = 1...K$ timesteps and planning horizon h, the agent optimizes action sequences $a_{k:k+h}$ using CEM, and executes the first action a_k (MPC). $\mathcal{D}_t$ is augmented by a tuple (s_k, a_k, s_{k+1}), where s_k is the current state, a_k is the executed action, and s_{k+1} is the next observed state under task t.

The parameters Θ_t of the hypernetwork and the task embeddings e_t are updated by backpropagating gradients with respect to the sum of a dynamics loss $\mathcal{L}_{\text{dyn}}$ and a regularization term. We define the dynamics loss as $\mathcal{L}_{dyn}(\Theta_t, e_t) = \sum_{\mathcal{D}_t} ||\hat{s}_{k+1} - s_{k+1}||_2$, where the predicted next states are $\hat{s}_{k+1} = f_{\theta_t}(s_k, a_k)$ and $\theta_t = H_{\Theta_t}(e_t)$. In practice, we infer the difference Δ_{k+1} through the dynamics network ($\Delta_{k+1} = f_{\theta_t}(s_k, a_k)$) such that $\hat{s}_{k+1} = s_k + \Delta_{k+1}$ for stable training. In addition, inputs to the f_{θ_t} network are normalized, following the procedure in previous works [20]. Similarly, a new e_t is initialized at the start of every task and updated every episode during the task by gradient descent. Older task embeddings ($e_{1:t-1}$) are kept fixed.

Regularizing the Hypernetwork To alleviate catastrophic forgetting, we regularize the output of the hypernetwork for all previous task embeddings $e_{1:t-1}$. After training for task $t-1$, a snapshot of the hypernetwork weights is saved as Θ_{t-1}. For each of the past tasks $i = 1...t-1$, we use a regularization loss to keep the outputs of the snapshot $H_{\Theta_{t-1}}(e_i)$ and the current output $H_{\Theta_t}(e_i)$. This approach sidesteps the need to store all past data across tasks, preserves the predictive performance of dynamic networks f_{θ_t}, and only requires a single point in the weight space (a copy of the hypernetwork) to be stored. The task embeddings are differentiable vectors learned along with parameters of the hypernetwork. The overall loss function for updating Θ_t and e_t is given by the sum of the dynamics loss $\mathcal{L}_{\text{dyn}}(\cdot)$, which is evaluated on the data collected from task t and the regularization term $\mathcal{L}_{\text{reg}}(\cdot)$:

$$\mathcal{L}_t(\Theta_t, e_t) = \mathcal{L}_{\text{dyn}}(\Theta_t, e_t) + \mathcal{L}_{\text{reg}}(\Theta_{t-1}, \Theta_t, e_{1:t-1}) \quad (1)$$

$$= \mathcal{L}_{\text{dyn}}(\Theta_t, e_t) + \frac{\beta_{\text{reg}}}{t-1} \sum_{i=1}^{t-1} \|H_{\Theta_{t-1}}(e_i) - H_{\Theta_t}(e_i)\|_2^2 \quad (2)$$

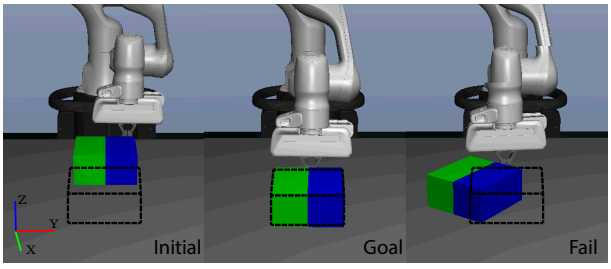
The planning objective for CEM optimization of action sequences is given by the sum of rewards obtained by executing the sequence of actions $a_{k:k+h}$ under the learned dynamics model $f_{\theta_t}(\cdot)$ for the task. The reward function $r(s, a)$ is assumed to be known, but nothing precludes learning it from data under our current framework.

5 Experiments

We perform multiple robot simulation experiments to answer the following questions: (a) How does `HyperCRL` compare with existing continual learning baselines in terms of overall performance across tasks? (b) How effective is `HyperCRL` in preventing catastrophic forgetting across tasks?

5.1 Pushing a block of non-uniform mass

First, we look at an intuitively simple experiment simulated using Surreal Robotics Suite [40], in which a Panda robot tries to push a non-uniform-density block across the table (Figure 2). The objective is to push the block to the goal position while **maintaining its initial orientation**. We vary the densities of the left and right parts of the block across different tasks ($T = 5$), changing



(a) Initial and goal poses of the block.

Task	Green	Blue
1	500	500
2	100	500
3	500	100
4	500	250
5	250	500

(b) Density of the block in kg/m^3

Figure 2: **Setup of Pusher Environment**. The robot needs to solve 5 pushing tasks sequentially, each involving a block of different mass distribution. The objective is to push the block to the goal pose (indicated by the dotted frame) without changing its orientation.

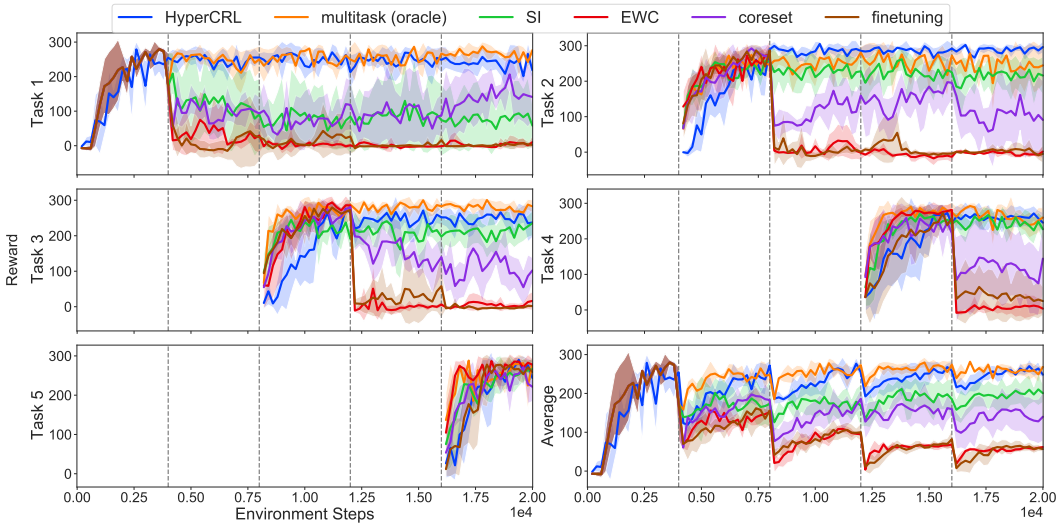


Figure 3: **Reward on Pusher Environment**. Shown are episodic rewards evaluated during training. Results are averaged across four random seeds, and the shaded areas represent one standard deviation. Each task is trained for 4k steps, summing to 20k steps in total. Vertical dotted lines indicate task switches. The bottom-right subplot shows the average reward across all task $\leq t$ seen so far.

the center of mass and moment of inertia. The robot needs to learn to maneuver the position of its end-effector on the side of the block to correct for orientation deviations while pushing the block forward. Each episode is 100-step long, which is 10 seconds of simulator time. At the start of each episode, we initialize the robot end-effector to a fixed position behind the block.

The state is represented as a concatenated vector $(x_{\text{ee}}, x_1, x_2, x_3, x_4)$. Here (x_1, x_2, x_3, x_4) represents the xy positions of the four corners and x_{ee} the xy position of the end-effector with respect to the block. The input action vector, δx_{ee}, specifies a desired offset with respect to the current position x_{ee}. The robot is actuated using an operational space controller [41]. It calculates the joint torques given an action that is updated at 10Hz. The reward function is set to minimize the sum of distances between the current and goal pose of the block following $r(s,a) = \sum_{i=1}^{4}(1 - tanh(10\ ||x_i - g_i||)) - 0.25||a||$, where g_i is the goal position of the i^{th} corner. The reward function consists of one term per corner within the range [0, 1], and to maximize reward the robot should minimize the corner distances to the goal pose.

Baselines We compare the hypernetwork to the following baselines: (i) Multi-task learning with access to a buffer of all data from all previous tasks (an oracle) (ii) Coreset that remembers one percent of the state-action transition data per task, sampled randomly (iii) Synaptic Intelligence [10] (iv) Elastic Weight Consolidation [7] (v) Fine-tuning, where we optimize $f_{\theta_t}(\cdot)$ on each task's data without regularization. All the baseline models resemble the target model in our hypernetwork setup,

Table 1: **Performance Retention on Pusher and Door Envs**. We measure performance retained at the end of training all 5 tasks compared to the end of training on task t. Results show the mean and one std. deviation, evaluated across 4 seeds and 10 episodes per seed. Greater than 100 indicates positive backward transfer, otherwise it indicates forgetting.

% Retention in terms of Reward (Pusher)					
Task	**1**	**2**	**3**	**4**	**Average**
Multi-task (Oracle)	**142 ± 74**	91 ± 25	96 ± 9	91 ± 11	**106 ± 20**
HyperCRL	99 ± 10	**107 ± 9**	**98 ± 13**	**103 ± 15**	102 ± 6
SI ($c = 0.1$)	40 ± 54	88 ± 21	95 ± 21	92 ± 14	79 ± 16
EWC ($\lambda = 10^5$)	7 ± 15	13 ± 8	6 ± 8	0 ± 3	4 ± 5
Coreset (1% Data)	87 ± 66	32 ± 46	39 ± 13	61 ± 43	55 ± 23
Finetuning	0 ± 2	-2 ± 6	1 ± 3	13 ± 16	3 ± 4
% Retention in terms of Normalized Reward (Door, see Figure 5)					
Task	**1**	**2**	**3**	**4**	**Average**
Multi-task (Oracle)	37 ± 70	81 ± 88	59 ± 60	**93 ± 129**	68 ± 45
HyperCRL	**113 ± 75**	**100 ± 76**	**99 ± 36**	86 ± 68	**100 ± 26**
SI ($c = 0.1$)	-17 ± 27	11 ± 61	6 ± 37	16 ± 60	4 ± 24
EWC ($\lambda = 10^5$)	-6 ± 33	17 ± 59	4 ± 25	16 ± 50	7 ± 22
Coreset (1% Data)	-5 ± 28	26 ± 53	21 ± 34	45 ± 68	22 ± 24
Finetuning	-14 ± 28	11 ± 58	3 ± 23	16 ± 65	4 ± 23

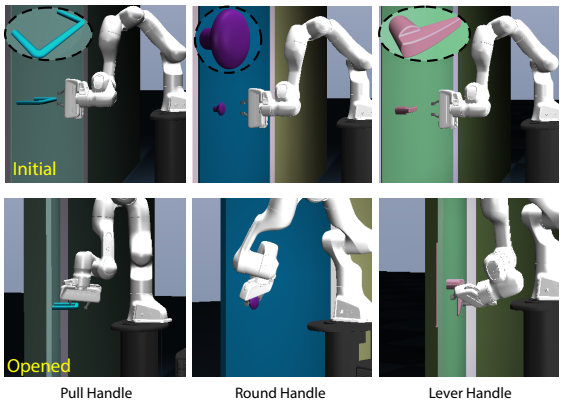


Figure 4: **Setup of Door Environment**. For round and lever handles, the robot must first rotate the handle *clockwise* before pulling the door open. Additionally, we introduce two more tasks by flipping the rotational direction to *counter-clockwise* for the round and lever handles.

except the multi-head output layer with one head per task. For coreset or multi-task learning, an additional batch of past data is sampled from the coreset or oracle every update step, and contributes to the total dynamics loss.

Results `HyperCRL` is able to learn to push all 5 of the blocks across the table with minimal forgetting (Table 1), and even shows signs of positive backward transfer. The average performance of our method is on par with the multitask learning baseline (Figure 3), which has access to the entire history of data. `HyperCRL` also outperforms other continual learning baselines, either regularization-based (SI, EWC), or replay-based (Coreset). Simple finetuning, however, catastrophically forgets and is unable to perform pushing for all 5 types of block at the end.

5.2 Door Opening

Next, we experiment on a more complex task to better demonstrate the flexibility of `HyperCRL`. We choose a door opening experiment, where the Panda robot has to open doors with different types of handles (Figure 4). Unlike the pushing example, the dynamic switches between tasks are much more discontinuous and the tasks require different motions to solve, due to the rotational joints imposed by some of the handles. A total of $T = 5$ tasks need to be solved sequentially, each involving a different handle type. Tasks 2 and 4 (similarily tasks 3 and 5) both have a round (lever) handle, but with different turning directions. The environment is modified from the DoorGym environment [42] and simulated in the Surreal Robotics Suite.

We model the environment as follow: (i) ϕ_d represents the angle of the door (ii) ϕ_k represents the angle of the door handle (iii) $x \in \mathbb{R}^3$ is the position, $q \in \mathbb{R}^4$ is the quaternion representation of the rotation of the robot end-effector with respect to the handle (iv) $d \in \mathbb{R}$ is the state of the gripper (v) $q_j \in \mathbb{R}^7$ is the angle of the joints of the Panda arm. Concatenating all the above elements and their time derivative gives the full state vector $(\phi_d, \dot{\phi}_d, \phi_k, \dot{\phi}_k, x, q, d, q_j, \dot{q}_j) \in \mathbb{R}^{26}$. Similar to the pushing environment, the robot is actuated with operational space control updated at 10Hz. The input action is a vector $(\delta x, \delta q, \delta d)$, which specifies a translation and rotational movement of the end-effector in the world frame. Finally, the reward function has five components as follows: $r(s,a) = -||x||_2 - \log(||x||_2 + \epsilon) - ||q_o||_2 + 50\,\phi_d + 20\,\phi_k$, where q_o is the difference between the orientation of the current end-effector and the required pose for opening the door.

Results `HyperCRL` outperforms all continual learning baselines on the door opening task, and even the multi-task baseline with oracle. Figure 5 shows the learning curves of all our evaluated methods, compared to a single-task baseline trained from scratch on each task. `HyperCRL` shows high reward across all tasks and virtually sees no performance degradation in terms of reward (Table 1).

One observation here is that the multi-head multitask baseline underperforms `HyperCRL` (Figure 5). We hypothesize that the hypernetwork architecture might be more effective for learning the task since it allows modelling multiple distinct dynamics in one model. We consider another multi-task baseline, `HyperCRL-MT`, that shares the same architecture of the hypernetwork used in `HyperCRL`, but is trained without regularization and has access to the entire replay buffer similar to the mul-

Method	**Final Average Reward**
HyperCRL-MT	1.25 ± 0.39
HyperCRL	1.08 ± 0.31
Multitask (Oracle)	0.73 ± 0.36

Table 2: Final performance comparison for the door opening task. Results are normalized episodic reward, averaged over all tasks.

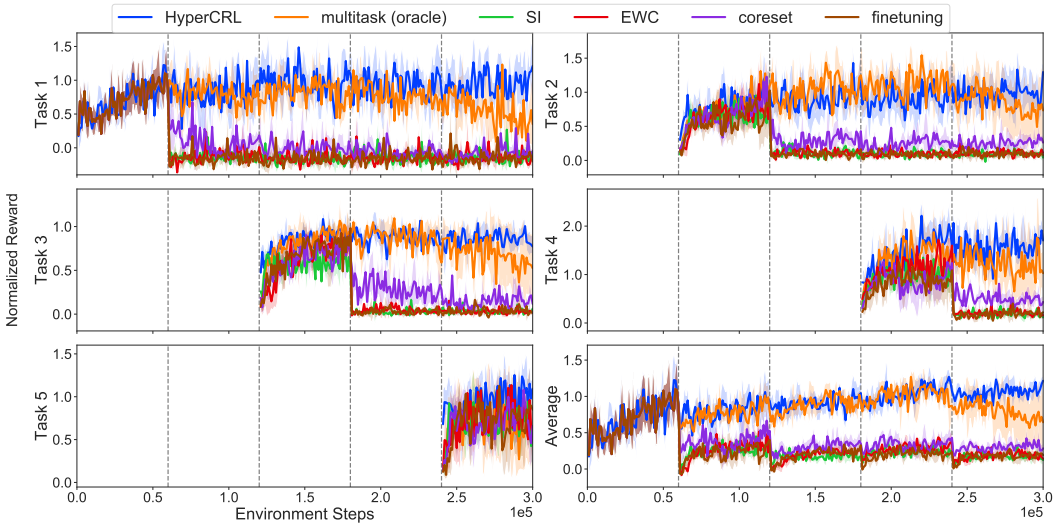


Figure 5: **Normalized Reward on Door Environment** during training. Results are averaged across four random seeds, and the error bars represent one standard deviation. Each task is trained for 60k steps, totaling 300k steps. The reward is normalized with respect to a model trained from scratch separately on each task.

titask baseline. We show that this baseline outperforms the multitask baseline while sharing the same training procedure, in Table 2. Thus, we infer that the hypernetwork architecture is what makes the difference.

5.3 Training details

For both experiments, we model the target network as a multi-layer perceptron (MLP) with two hidden layers (four for door opening) of 200 neurons each and ReLU non-linearity. Each task embedding is initialized as a $10d$ standard normal vector. The hypernetwork is also a MLP with two hidden layers of 50 neurons each (256 neurons for door opening). The parameters of the hypernetwork are initialized with the Xavier initialization [43]. We use Adam with a learning rate of 0.0001 to optimize $\mathcal{L}_t$. During planning, we run CEM for 5 iterations to optimize the actions for a horizon of $h = 20$ (10 for door opening) steps. Each iteration, we sample 500 (2000 for door opening) action sequences to maximize the sum of rewards. For SI and EWC, we use implementations from [3].

6 Limitations and Future Work

While our proposed approach shows good results against the baselines we tested, there are many interesting prospects for future work. First, better techniques to train hypernetworks will significantly aid this line of work. Currently, the size of our hypernetwork is at least an order of magnitude larger than the target network, since it directly outputs all the weights. Hypernetworks are often sensitive to the choice of random seeds and architecture. Second, extending our method to image-based RL environments is worth investigating, as it will enable higher-capacity target networks. Finally, `HyperCRL` is not task agnostic, nor can it automatically detect task switching that happens continuously with no clear task boundaries. A possible direction is to use probabilistic inference models (i.e. Bayesian non-parametrics [2]) or changepoint detection methods to perform task identification.

7 Conclusion

In this paper we described `HyperCRL`, a task-aware method for continual model-based reinforcement learning using hypernetworks. In all of our experiments, we have demonstrated that `HyperCRL` consistently outperforms alternative continual learning baselines in terms of overall performance at the end of training, as well as in terms of retaining performance on previous tasks. By allowing the entire network to change between tasks, rather than just the output head layer of the dynamics network, `HyperCRL` is more effective at accurately representing different dynamics, even with significant discontinuity between them, while only requiring a fixed-size hypernetwork and constant time updates to generate task-conditional dynamics for planning.

References

[1] J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2019.

[2] M. Xu, W. Ding, J. Zhu, Z. Liu, B. Chen, and D. Zhao. Task-agnostic online reinforcement learning with an infinite mixture of gaussian processes, 2020.

[3] G. M. van de Ven and A. S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

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

[5] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *arXiv: Computer Vision and Pattern Recognition*, 2020.

[6] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36, 2012.

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[8] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[9] H. Ritter, A. Botev, and D. Barber. Online structured laplace approximations for overcoming catastrophic forgetting, 2018.

[10] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. *Proceedings of machine learning research*, 70:3987, 2017.

[11] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017.

[12] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem, 2018.

[13] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato. Continual learning with tiny episodic memories. 2019.

[14] M. K. Titsias, J. Schwarz, A. G. de G. Matthews, R. Pascanu, and Y. W. Teh. Functional regularisation for continual learning using gaussian processes. *ArXiv*, abs/1901.11356, 2020.

[15] S. Kapoor, T. Karaletsos, and T. D. Bui. Variational auto-regressive gaussian processes for continual learning, 2020.

[16] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[17] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[18] D. Burt, C. E. Rasmussen, and M. van der Wilk. Explicit rates of convergence for sparse variational inference in gaussian process regression. In *Symposium on advances in approximate Bayesian inference*, volume 1, 2018.

[19] T. Raiko and M. Tornio. Variational bayesian learning of nonlinear hidden state-space models for model predictive control. *Neurocomputing*, 72(16-18):3704–3712, 2009.

[20] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[21] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019.

[22] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[23] E. W. Basso and P. M. Engel. Reinforcement learning in non-stationary continuous time and space scenarios. In *Artificial Intelligence National Meeting (Enia)*, volume 7, pages 1–8, 2009.

[24] S. Padakandla, S. Bhatnagar, et al. Reinforcement learning in non-stationary environments. *arXiv preprint arXiv:1905.03970*, 2019.

[25] A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[26] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

[27] A. Nagabandi, C. Finn, and S. Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018.

[28] I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt: Meta-learning for model-based control. *ArXiv*, abs/1803.11347, 2018.

[29] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

[30] G. Berseth, K. Xie, P. Cernek, and M. van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *ArXiv*, abs/1802.04765, 2018.

[31] D. Abel, D. Arumugam, L. Lehnert, and M. Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 10–19, 2018.

[32] A. Abdollahi, R. Allamraju, and G. Chowdhary. Adaptive-optimal control of nonstationary dynamical systems. In *Proceedings of the 2015 European Guidance, Navigation, and Control Conference*, 2015.

[33] N. Wagener, C. Cheng, J. Sacks, and B. Boots. An online learning approach to model predictive control. *CoRR*, abs/1902.08967, 2019. URL `http://arxiv.org/abs/1902.08967`.

[34] K. J. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 1994. ISBN 0201558661.

[35] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[36] J. U. Schmidhuber. Learning to control fast-weight memories: an alternative to dynamic recurrent networks. 1991.

[37] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. B. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *CoRR*, abs/1903.03088, 2019. URL `http://arxiv.org/abs/1903.03088`.

[38] R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

[39] H. Bharadhwaj, K. Xie, and F. Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. *Learning for Dynamics and Control*, 2020.

[40] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, 2018.

[41] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[42] Y. Urakami, A. Hodgkinson, C. Carlin, R. Leu, L. Rigazio, and P. Abbeel. Doorgym: A scalable door opening environment and baseline agent. *arXiv preprint arXiv:1908.01887*, 2019.

[43] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[44] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek. Benchmark environments for multitask learning in continuous domains. *arXiv preprint arXiv:1708.04352*, 2017.

Appendix

A Additional Test Results

The objective of a continual learning algorithm is to continuously learn new knowledge on the current task, while maintaining its performance on past tasks. To achieve this, there are two components that needs to be quantified: 1) **backward transfer** and 2) **forward transfer**. We formally define them in the context of continual reinforcement learning as follow:

Backward Transfer First, let $r_{i,j}$ denote the test episodic (normalized) reward on task i after training on task j. Retention for each task is then given by $f_i = r_{i,T}/r_{i,i}$ for all $i < T$. The average retention is $f = \frac{1}{T-1}\sum_{i=1}^{T-1} f_i$. If this measure is larger than 100%, it indicates *positive backward transfer*, which means the performance on previously seen tasks benefit from more recent experiences. If this measure of retention is lower than 100%, this means that the robot forgets some knowledge about older tasks and we call this *negative backward transfer* or *forgetting*.

Forward Transfer Forward Transfer measures how the robot adapts to new tasks after training on older task(s). Specifically, let r_i^* denote the test episodic reward on task i for a single-task model trained from scratch using CEM planning. This single-task model serves as a baseline for performance. We then calculate forward transfer as $I_i = r_{i,i}/r_i^*$, and the average is $I = \frac{1}{T-1}\sum_{i=2}^{T} I_i$. If this measure is greater than 100%, it indicates *positive forward transfer*, which means that the robot benefits from older experiences when learning a new task. If this measure is lower than 100%, it means that the robot is unable to take advantage of past experience to solve new tasks, also known as *negative forward transfer*.

To better illustrate the continual learning capability of `HyperCRL`, we provide an evaluation of the positive transfer capabilities defined above. Note that `HyperCRL` is not explicitly designed for improving forward transfer or fast adaptation. We show the result on both the pusher and door opening experiments in Table 3.

In the pusher experiment, `HyperCRL` achieves an average forward transfer of $100 \pm 11\%$. This means that our proposed method can adapt to new tasks and reach similar level of performances compared to a single-task baseline trained from scratch. In the door experiment, `HyperCRL` beats all baseline in terms of forward transferring capabilities. A note of caution is that some results have very large variance across different random seeds. Better techniques to train hypernetworks or plan with learned dynamics model would improve the reliability of our approach, and we leave this to future work.

B Additional Experiment on HalfCheetah

We ran another experiment using `HyperCRL` following the environment setup from [44], changing the body of the robot. We modified body size (torso, leg) on the HalfCheetah environment to create

Table 3: **Forward Transfer on Pusher and Door Environment**. Shown is performance at the end of training on task t compared to training a single-task baseline from scratch. Results show the mean and one standard deviation, evaluated across four seeds and 10 episodes per seed. Greater than 100 indicates positive forward transfer.

% Forward Transfer in terms of Task Reward (Pusher)					
Task	**2**	**3**	**4**	**5**	**Average**
Multi-task (Oracle)	118 ± 23	106 ± 12	$\mathbf{105 \pm 9}$	110 ± 21	109 ± 9
HyperCRL	127 ± 22	99 ± 12	94 ± 12	107 ± 20	107 ± 9
SI ($c = 0.1$)	114 ± 26	84 ± 13	98 ± 15	101 ± 20	99 ± 10
EWC ($\lambda = 10^5$)	125 ± 23	$\mathbf{109 \pm 11}$	105 ± 10	$\mathbf{110 \pm 21}$	$\mathbf{112 \pm 9}$
Coreset	126 ± 20	99 ± 13	90 ± 17	106 ± 21	105 ± 9
Finetuning	$\mathbf{138 \pm 22}$	108 ± 9	94 ± 17	107 ± 23	112 ± 9
% Forward Transfer in terms Task Normalized Reward (Door)					
Task	**2**	**3**	**4**	**5**	**Average**
Multi-task (Oracle)	103 ± 74	$\mathbf{94 \pm 34}$	122 ± 160	78 ± 78	96 ± 49
HyperCRL	$\mathbf{106 \pm 76}$	91 ± 31	$\mathbf{168 \pm 203}$	$\mathbf{102 \pm 67}$	$\mathbf{117 \pm 57}$
SI ($c = 0.1$)	69 ± 57	56 ± 40	107 ± 137	66 ± 66	75 ± 42
EWC ($\lambda = 10^5$)	71 ± 62	78 ± 32	125 ± 151	71 ± 61	86 ± 44
Coreset	94 ± 80	83 ± 34	113 ± 145	57 ± 68	84 ± 46
Finetuning	73 ± 83	90 ± 43	101 ± 145	83 ± 77	87 ± 47

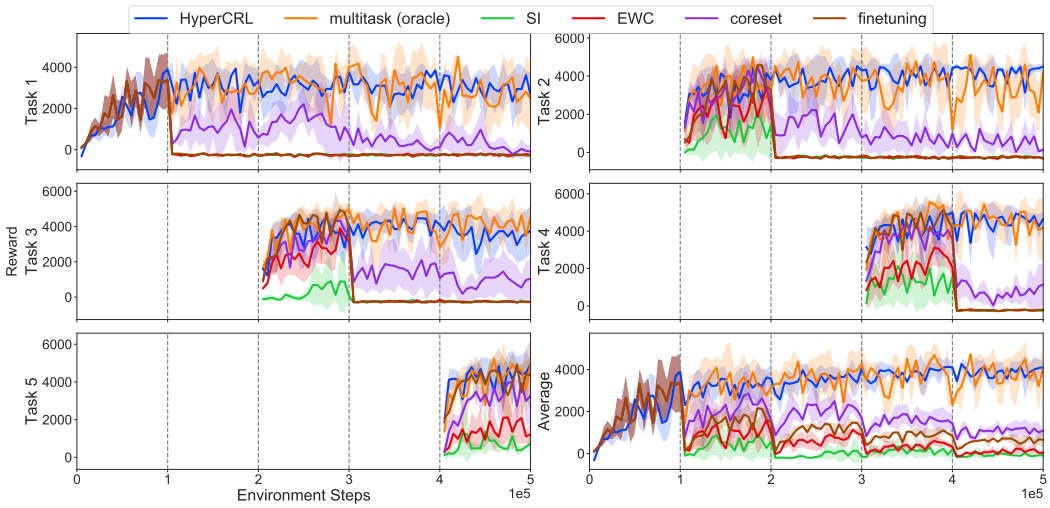


Figure 6: **Performance on HalfCheetah Environment** from [44]. Each task is trained for 100k steps, totaling 500k steps. Results are averaged across four random seeds, and shaded area marks one standard deviation.

a continual learning scenario with $T = 5$ tasks. Our method outperforms other continual learning baselines (SI, EWC, coreset) and performs similarly to multi-task learning (Figure 6).

C More Training Details

In this section, we provide more details about the hyperparameters used during the experiments in Table 4.

Table 4: Hyper-parameter values of the proposed algorithm `HyperCRL` for all the environments.

	P	M	K	S	α_H	$\alpha_{\mathbf{e}}$	$\mathcal{B}$	β_{reg}	hnet non-linearity
Pusher	10	20	200	2000	0.0001	0.0001	100	0.05	ELU
Door	10	300	200	200	0.0001	0.0001	100	0.5	ReLU
Half Cheetah	10	100	1000	2000	0.0001	0.0001	100	0.05	ReLU