Continual Task Allocation in Meta-Policy Network via Sparse Prompting

Yijun Yang 1 2  Tianyi Zhou 3  Jing Jiang 2  Guodong Long 2  Yuhui Shi 1

Abstract
How to train a generalizable meta-policy by continually learning a sequence of tasks? It is a natural human skill yet challenging to achieve by current reinforcement learning: the agent is expected to quickly adapt to new tasks (plasticity) meanwhile retaining the common knowledge from previous tasks (stability). We address it by “Continual Task Allocation via Sparse Prompting (CoTASP)”, which learns over-complete dictionaries to produce sparse masks as prompts extracting a sub-network for each task from a meta-policy network. CoTASP trains a policy for each task by optimizing the prompts and the sub-network weights alternatively. The dictionary is then updated to align the optimized prompts with tasks’ embedding, thereby capturing tasks’ semantic correlations. Hence, relevant tasks share more neurons in the meta-policy network due to similar prompts while cross-task interference causing forgetting is effectively restrained. Given a meta-policy and dictionaries trained on previous tasks, new task adaptation reduces to highly efficient sparse prompting and sub-network finetuning. In experiments, CoTASP achieves a promising plasticity-stability trade-off without storing or replaying any past tasks’ experiences. It outperforms existing continual and multi-task RL methods on all seen tasks, forgetting reduction, and generalization to unseen tasks. Our code is available at https://github.com/stevenyangyj/CoTASP

1. Introduction
Although reinforcement learning (RL) has demonstrated excellent performance on learning a single task, e.g., playing Go (Silver et al., 2016), robotic control (Schulman et al., 2017; Degrave et al., 2022), and offline policy optimization (Yu et al., 2020; Yang et al., 2022), it still suffers from catastrophic forgetting and cross-task interference when learning a stream of tasks on the fly (McCloskey & Cohen, 1989; Bengio et al., 2020) or a curated curriculum of tasks (Fang et al., 2019; Ao et al., 2021; 2022). So it is challenging to train a meta-policy that can generalize to all learned tasks or even unseen ones with fast adaptation, which however is an inherent skill of human learning. This problem has been recognized as continual or lifelong RL (Mendez & Eaton, 2022) and attracted growing interest in recent RL research.

A primary and long-standing challenge in continual RL is the plasticity-stability trade-off (Khetarpal et al., 2022): the RL policy on the one hand needs to retain and reuse the knowledge shared across different tasks in history (stability) while on the other hand can be quickly adapted to new tasks without interference from previous tasks (plasticity). Addressing this challenge is vital to improving the efficiency of
As illustrated by Fig. 1, given each task \( t \) when the number of tasks increases (Shin et al., 2017; Li & Hoiem, 2018). Furthermore, better plasticity indicates faster adaptation and generalization to new tasks.

In order to directly address the plasticity-stability trade-off and overcome the drawbacks of prior work, we study how to train a meta-policy network in continual RL. Then, given only a textual description of a previously learned or unseen task, a specific policy can be automatically and efficiently extracted from the meta-policy. This is in the same spirit of prompting in recent large language models (Li & Liang, 2021; Liu et al., 2021) but differs from existing methods that randomly select task policies (Rajesagar et al., 2019; Mirzadeh et al., 2020) or independently optimize a policy for each task from scratch (Serra et al., 2018; Kang et al., 2022). To this end, we propose to learn layer-wise dictionaries along with the meta-policy network to produce sparse prompts (i.e., binary masks) for each task, which extract a sub-network from the meta-policy to be the task-specific policy. We call this approach “Continual Task Allocation via Sparse Prompting (CoTASP)”.

As illustrated by Fig. 1, given each task \( t \), the prompt \( \alpha_t \) is generated by sparse coding of its task embedding \( e_t \) under dictionary \( D_t \) and used to allocate a policy sub-network \( \theta_t \) from the meta-policy. Then \( \alpha_t \) and \( \theta_t \) are optimized through RL. At the end of each task, the dictionary \( D_t \) is optimized (Mairal et al., 2009; Arora et al., 2015) for all learned tasks to provide a mapping from their task embedding to their optimized prompts/sub-networks, which exploits the task correlations in both the embedding and prompt spaces. This leads to efficient usage of the meta-policy network’s capacity and automatic optimization of the plasticity-stability trade-off, i.e., relevant tasks reuse skills by sharing more neurons (plasticity and fast adaptation) while the harmful interference between irrelevant tasks is avoided by sharing less or no neurons (stability and less forgetting). Moreover, due to the dictionary, CoTASP does not need to store or replay any previous tasks’ experiences and thus costs much less computation and memory than rehearsal-based methods (Rolnick et al., 2019; Wolczyk et al., 2022). Furthermore, the sparse prompting in CoTASP, as an efficient task adaptation method, can extract policy sub-networks for unseen tasks and thus leads to a more generalizable meta-policy.

On Continual World benchmarks (Wolczyk et al., 2021), CoTASP outperforms most baselines on all learned tasks, forgetting reduction, and generalization to unseen tasks (Ta-
ing stability and reducing catastrophic forgetting. Rehearsal-based methods such as CLEAR (Rolnick et al., 2019) and P&C (Schwarz et al., 2018) repeatedly replay buffered experiences from previous tasks but their buffer memory and computational cost linearly grow with the number of tasks (Kumari et al., 2022). Regularization-based methods such as EWC (Kirkpatrick et al., 2017) and PC (Kaplanis et al., 2019) alleviate forgetting without the replay buffer by adding extra regularizers when learning new tasks, which can bias the policy optimization and lead to sub-optimal solutions (Zhao et al., 2023). Finally, structure-based methods adopt different modules, i.e., sub-networks within a fixed-capacity policy network, for each task (Mendez & Eaton, 2022). We summarize two main categories of structure-based methods in the following. We also provide a more detailed discussion of related work in Appendix B and C.

Connection-level methods. This category includes methods such as PackNet (Mallya & Lazebnik, 2018), SupSup (Wortsman et al., 2020), and WSN (Kang et al., 2022). For task \( t \), the action \( a_t \) is drawn from \( a_t \sim \pi(s_t; \theta \otimes \phi_t) \) where \( s_t \) is the state and \( \phi_t \) is a binary mask applied to the model weights \( \theta \) in an element-wise manner (i.e., \( \otimes \)). PackNet generates \( \phi_t \) by iteratively pruning \( \theta \) after the learning of each task, thereby preserving important weights for the task while leaving others for future tasks. SupSup fixes a randomly initialized network and finds the optimal \( \phi_t \) for each task \( t \). WSN jointly learns \( \theta \) and \( \phi_t \) and uses Huffman coding (Huffman, 1952) to compress \( \phi_t \) for a sub-linear growing size of \( \{\phi_t\}_{t=1}^T \) with increasing tasks. However, these methods usually need to store the task-specific masks for each task in history, leading to additional memory costs (Lange et al., 2022). Moreover, their masks are seldom optimized for knowledge sharing across tasks, impeding the learned policy from being generalized to unseen tasks.

Neuron-level methods. Instead of extracting task-specific sub-networks by applying masks to model weights, the other category of methods (Fernando et al., 2017; Serrà et al., 2018; Ke et al., 2021; Sokar et al., 2021) produces sub-networks by applying masks to each layer’s neurons/outputs of a policy network. Compared to connection-level methods, they use layer-wise masking to achieve a more flexible and compact representation of sub-networks. But the generation of masks depends on either heuristic rules or computationally inefficient policy gradient methods (Gurbuz & Dovrolis, 2022; Serrà et al., 2018). By contrast, CoTASP generates masks by highly efficient sparse coding (solving a relatively small lasso problem).

3. Methods

In this section, we introduce the main steps and components of CoTASP (see Fig. 1). Specifically, Sec. 3.1 introduces the meta-policy network in the continual RL setting. Sec. 3.2 describes our proposed sparse prompting for task policy extraction. Finally, Sec. 3.3 provides the detailed optimization procedure for each component in CoTASP, including the prompt, task policy, and dictionary.

3.1. Continual RL with a Meta-Policy Network

As discussed in Sec. 2, finetuning all weights in \( \theta \) via the optimization in Eq. 1 without access to past tasks leads to harmful shift on some weights important to previous tasks and catastrophic forgetting of them. Structure-based methods address it by allocating a sub-network for each task and freezing its weights once completing the learning of the task so they are immune to catastrophic forgetting. Following prior work (Srivastava et al., 2014; Fernando et al., 2017; Serrà et al., 2018; Ke et al., 2021; Sokar et al., 2021), we represent such a sub-network by applying a binary mask to each neuron’s output. Specifically, given a meta-policy network with \( L \) layers, let \( l \in \{1, \ldots, L − 1\} \) index the hidden layers of the network as a superscript, e.g., \( y^{(l)} \) is the output vector of layer-\( l \) and \( \theta^{(l)} \) denotes layer-\( l \)’s weights. The output of the sub-network on its \( (l + 1) \)-th layer is

\[
y^{(l+1)} = f(\phi^{(l)}_t \otimes y^{(l)}; \theta^{(l+1)}),
\]

where \( \phi^{(l)}_t \) is a binary mask generated for task \( t \) and applied to layer-\( l \), and \( f \) is a stand-in for the neural operation, e.g., a fully-connected or convolutional layer. The element-wise product operator \( \otimes \) activates a portion of neurons in layer-\( l \) of the meta-policy network according to \( \phi^{(l)}_t \). These activated neurons over all layers extract a sub-network as a task-specific policy, which then interacts with the environment to collect training experiences. Training the sub-network avoids harmful interference with previous tasks on other neurons and meanwhile encourages their knowledge sharing on the shared neurons. However, allocating policies in the meta-policy network for a sequence of diverse tasks raises several challenges. For efficient usage of network capacity, each task policy should be a sparse sub-network with only a few neurons activated. Furthermore, the policy should selectively reuse neurons from previously learned policies, which can facilitate knowledge sharing between relevant tasks and reduce interference from irrelevant tasks. Inspired by prompting and in-context training for NLP (Rebuffi et al., 2017; Houlshby et al., 2019; Li & Liang, 2021; Liu et al., 2021), we propose “Sparse Prompting” to address the aforementioned challenge of continual task allocation, which can automatically and efficiently extract task-specific policies from the meta-policy network.

3.2. Task Policy Extraction via Sparse Prompting

In continual task allocation, the sub-network extracted for a new task is expected to reuse the knowledge learned from relevant tasks in the past and meanwhile to avoid harmful in-
Alternating Optimization of Task Policy and Prompt

To address the first concern, we update the weights selectively by only allowing updates of weights that have never been allocated for any previous task. For this purpose, we accumulate the binary masks for all learned tasks by $\phi_{t-1} = \bigvee_{i=1}^{t-1} \phi_{t-1}^{(i)}$ and update $\theta$ when learning task $t$ by

$$\theta \leftarrow \theta - \eta g_t$$

where $\eta$ is the learning rate and $g_t$ denotes negative gradients of the expected return w.r.t. $\theta$ for layer-$l$ on the task $t$. In Eq. 4, we modify each weight’s gradient according to the accumulated mask associated with its input and output layer. This effectively avoids overwriting the weights selected by policies of previous tasks and thus mitigates forgetting.

To address the second concern, we use the straight-through estimator (STE) (Bengio et al., 2013), i.e., clip($\alpha_i, 0, 1$), in the backward pass so that the $\alpha$ can be directly optimized using the same gradient descent algorithm applied to the meta-policy weights.

Dictionary Learning

Given the previous tasks’ optimized prompts $\alpha_{t}^{*}$ and their embedding, we further update the dictionary per layer so that sparse prompting will be improved to produce the optimized prompts for each previous task, i.e.,

$$D_{t}^{(i)} = \arg\min_{D \in \mathbb{R}^{m \times k}} \frac{1}{t} \sum_{i=1}^{t} \|e_i - D \alpha_{t}^{*} \|_2^2, \text{ s.t. } \|D[j]\|_2 \leq c$$

where we constrain the $\ell_2$ norm of each atom $D[j]$ to prevent the scale of $D$ from growing arbitrarily large, which would lead to arbitrarily small entries in $\alpha$.

To solve the optimization with inequality constraints in Eq. 5, we use block-coordinate descent with $D_{t-1}^{(i)}$ as warm restart, as described in Alg. 1. Specifically, we sequentially update each atom of $D_{t-1}^{(i)}$ under the constraint $\|D[j]\|_2 \leq c$.
while fixing the rest of atoms. Since this optimization admits separable constraints to the updated atoms, convergence to a global optimum is guaranteed (Bertsekas, 1997; Lee et al., 2006; Mairal et al., 2009). Moreover, $D_{t-1}^{(l)}$ as a warm start for $D_t^{(l)}$ significantly reduces the required optimization steps: we empirically found one step suffices. The complete training procedure of CoTASP is detailed in Alg. 2.

4. Experiments

4.1. Experimental Setups

Benchmarks. To evaluate CoTASP, we follow the same settings as prior work (Wolczyk et al., 2022) and perform thorough experiments. Specifically, we primarily use CW10, a benchmark in the Continual World (CW) (Wolczyk et al., 2021), which consists of 10 representative manipulation tasks from MetaWorld (Yu et al., 2019). To make the benchmark more challenging, we rank these tasks according to a pre-computed transfer matrix so that there is a high variation of forward transfer both in the whole sequence and locally. We also use CW20, which repeats CW10 twice, to measure the transferability of the learned policy when encountering the same task. For a fair comparison between different tasks, the number of environment interaction steps is limited to 1M per task.

Evaluation metrics. Following a widely-used evaluation protocol in continual learning literature (Lopez-Paz & Ranzato, 2017; Rolnick et al., 2019; Chaudhry et al., 2019; Wolczyk et al., 2021; 2022), we adopt three metrics. (1) Average Performance (higher is better): the average performance at time $t$ is defined as $\bar{P}(t) = \frac{1}{T} \sum_{i=1}^{T} p_i(t)$ where $p_i(t) \in [0, 1]$ denotes the success rate of task $i$ at time $t$. This is a canonical metric used in the continual learning community. (2) Forgetting (lower is better): it measures the average degradation across all tasks at the end of learning, denoted by $F = \frac{1}{T} \sum_{i=1}^{T} p_i(1) - p_i(T)$. (3) Generalization (lower is better): it equals to the average number of steps needed to reach a success threshold across all tasks. Note that we stop the training when the success rate in two consecutive evaluations reaches the threshold (set to 0.9). Moreover, the metric is divided by $\delta$ to normalize its scale to $[0, 1]$.

Comparing methods. We compare CoTASP with several baselines and state-of-the-art (SoTA) continual RL methods. According to (Lange et al., 2022), these methods can be divided into three categories: regularization-based, structure-based, and rehearsal-based methods. Concretely, regularization-based methods include L2, Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017), Memory-Aware Synapses (MAS) (Aljundi et al., 2018), and Variational Continual Learning (VCL) (Nguyen et al., 2018). Structure-based methods include PackNet (Mallya & Lazebnik, 2018), Hard Attention to Tasks (HAT) (Serrà et al., 2018), and TaDeLL (Rostami et al., 2020). Rehearsal-based methods include Reservoir, Average Gradient Episodic Memory (A-GEM) (Chaudhry et al., 2019), and ClonEx-SAC (Wolczyk et al., 2022). For completeness, we also include a naive sequential training method (i.e., Finetuning) and representative multi-task RL baselines (MTL (Yu et al., 2019) and MTLa+PopArt (Hessel et al., 2019)), which are usually regarded as the soft upper bound a continual RL method can achieve. For a fair comparison, we refer to the Continual World repository for implementation and hyper-parameter selection. We re-run these methods to ensure the best possible performance. In addition, we adopt author-reported results for ClonEx-SAC due to the lack of open-sourced implementation. An extended description and discussion of these methods are provided in Appendix C.

Training details. In order to ensure the reliability and comparability of our experiments, we follow the training details described in (Wolczyk et al., 2021; 2022) and implement all of the baseline methods based on Soft Actor-Critic (SAC) (Haarnoja et al., 2018), a SoTA off-policy actor-critic.

---

**Algorithm 2 Training Procedure of CoTASP**

1: initialize: replay buffer $\mathcal{B} = \emptyset$, meta-policy network $\pi_{\theta}$ with $L$ layers, critic $Q$, dictionaries $\{D_{t-1}^{(l)}\}_{l=1}^{L-1}$, $A_{t}^{(l)}$, $B_{t}^{(l)}$, $\phi_{t}^{(l)}$, $\psi_{t}^{(l)} \leftarrow 0, 0, 0$, and constant $c$ for Alg. 1
2: input: training budget $I_{\omega}$, $I_{\alpha}$, and step function $\sigma(\cdot)$
3: for $t = 1$ to $T$ do
4: $e_{t} = f_{S.}^{\text{BERT}}$ (textual description of task $t$)
5: Initialize $\{\sigma(e_{t}^{(l)})\}_{l=1}^{L-1}$ by solving Eq. 3
6: Extract task-specific $\pi$ by Eq. 2 with $\{\sigma(e_{t}^{(l)})\}_{l=1}^{L-1}$
7: for each iteration do
8: $\triangleright$ Learning task $t$ with SAC
9: for $i = 1$ to $I_{\omega}$ do
10: Collect $\tau = \{s_i, a_i, r_i, s'_i\}$ with $\pi$
11: Gradient descent on $Q$
12: Update $\theta$ by Eq. 4 with $\{\phi_{t-1}^{(l)}\}_{l=1}^{L-1}$
13: for $i = 1$ to $I_{\alpha}$ do
14: Collect $\tau = \{s_i, a_i, r_i, s'_i\}$ with $\pi$
15: Update $B$ and sample a mini-batch $\tau$
16: Gradient descent on $Q$
17: Gradient descent on $\{\alpha_{t}^{(l)}\}_{l=1}^{L-1}$ by STE
18: for $l = 1$ to $L - 1$ do
19: $\phi_{t}^{(l)} \leftarrow \phi_{t-1}^{(l)} \lor \sigma(\alpha_{t}^{(l)} T)$
20: $A_{t}^{(l)} \leftarrow A_{t-1}^{(l)} + \alpha_{t}^{(l)} \phi_{t}^{(l)} T$
21: $B_{t}^{(l)} \leftarrow B_{t-1}^{(l)} + e_{t} \alpha_{t}^{(l)} T$
22: Get updated $D_{t}^{(l)}$ by Alg. 1 with $D_{t-1}^{(l)}$
23: output: $\theta^*$ and $\{D_{t}^{(l)}\}_{l=1}^{L-1}$.
algorithm. The actor and the critic are implemented as two separate multi-layer perceptron (MLP) networks, each with 4 hidden layers of 256 neurons. For structure-based methods (PackNet, HAT) and our proposed CoTASP, a wider MLP network with 1024 neurons per layer is used as the actor. We refer to these hidden layers as the backbone and keep the constraints on computation, memory, number of samples, and neural network architecture constant.

In this case, CoTASP does not require selecting the appropriate head for each task and enables the reuse of parameters between similar tasks. According to (Wolczyk et al., 2021), regularizing the critic often leads to a decline in performance. Therefore, we completely ignore the forgetting issue in the critic network and retrain it for each new task. More details on the hyperparameters used in training can be found in the Appendix D.

### 4.2. Main Results

This section presents the comparison between CoTASP and ten representative continual RL methods on CW benchmarks. We focus on the stability (retain performance on seen tasks) and the plasticity (quickly adapt to unseen tasks) and keep the constraints on computation, memory, number of samples, and neural network architecture constant.

Table 1 summarizes our main results on CW10 and CW20 sequences. CoTASP consistently outperforms all the compared methods across different lengths of task sequences, in terms of both average performance (measures stability) and generalization (measures plasticity). We observe that when the hidden-layer size is the same as other structure-based methods (PackNet and HAT), CoTASP outperforms them by a large margin, especially in the generalization metric, indicating the advantage of CoTASP in improving the adaptation to new tasks. A more detailed analysis of the reasons for CoTASP’s effectiveness is presented in Sec. 4.3 and 4.4. Moreover, we find that most continual

---

**Table 1:** Evaluation (mean\̲std of 3 metrics over 5 random seeds) on Continual World. *-reported in previous work. Reg = Regularization-based, Struc = Structure-based, Reh = Rehearsal-based, MT = Multi-task, P = Average Performance, F = Forgetting, G = Generalization. A detailed description of baselines and metrics can be found in Sec. 4.1. The best result for each metric is highlighted.

**Figure 3:** Performance (mean\̲std over 5 random seeds) of all methods on CW20 sequence. CoTASP outperforms all the continual RL methods and all the multi-task RL baselines. We separately plot the curve per method in Fig. 9 of Appendix.
Continual Task Allocation in Meta-Policy Network via Sparse Prompting

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CW 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics</td>
<td>$P (\uparrow)$</td>
</tr>
<tr>
<td>CoTASP (ours)</td>
<td>$0.88 \pm 0.02$</td>
</tr>
<tr>
<td>with D frozen</td>
<td>$0.73 \pm 0.06$</td>
</tr>
<tr>
<td>with $\alpha$ frozen</td>
<td>$0.79 \pm 0.06$</td>
</tr>
<tr>
<td>with both frozen</td>
<td>$0.62 \pm 0.05$</td>
</tr>
<tr>
<td>lazily update D</td>
<td>$0.85 \pm 0.03$</td>
</tr>
<tr>
<td>EWC</td>
<td>$0.60 \pm 0.07$</td>
</tr>
<tr>
<td>PackNet</td>
<td>$0.78 \pm 0.07$</td>
</tr>
<tr>
<td>A-GEM</td>
<td>$0.08 \pm 0.02$</td>
</tr>
<tr>
<td>Finetuning</td>
<td>$0.05 \pm 0.00$</td>
</tr>
</tbody>
</table>

Table 2: Ablation study. Performance of CoTASP variants on CW20 sequence. Please refer to Sec. 4.3 for a detailed explanation.

RL methods fail to achieve positive backward transfer (i.e., $F < 0$) except for VCL, suggesting the ability to improve previous tasks’ performance by learning new ones is still a significant challenge. We leave this for future work. Finally, the results in Fig. 3 show that CoTASP is the only method performing comparably to the multi-task learning baselines on the first ten tasks of CW20 sequence, and it exhibits superior performance over these baselines after learning the entire CW20 sequence. One possible explanation is that the knowledge accumulated by CoTASP’s meta-policy network and dictionaries leads to improved generalization.

4.3. Ablation Studies

Effectiveness of core designs. To show the effectiveness of each of our components, we conduct an ablation study on four variants of CoTASP, each of which removes or changes a single design choice made in the original CoTASP. Table 2 presents the results of the ablation study on CW20 sequence, using two representative evaluation metrics. Among the four variants of CoTASP, “D frozen” replaces the learnable dictionary with a fixed, randomly initialized one; “$\alpha$ frozen” removes the prompt optimization proposed in Sec. 3.3; “both frozen” neither updates the dictionary nor optimizes the prompt; “lazily update D” stops the dictionary learning after completing the first ten tasks of CW20 sequence. According to the results in Table 2, we give the following conclusions: (1) The use of a fixed, randomly initialized dictionary degrades the performance of CoTASP on two evaluation metrics, highlighting the importance of the learnable dictionary in capturing semantic correlations among tasks. (2) The “$\alpha$ frozen” variant performs comparably to our CoTASP but outperforms the results achieved by EWC and PackNet. This indicates that optimizing the prompt can improve CoTASP’s performance but is not crucial to our appealing results. (3) The “both frozen” variant exhibits noticeable degradation in performance, supporting the conclusion that the combination of core designs proposed in CoTASP is essential for achieving strong results. (4) The “lazily update D” variant only slightly degrades from the original CoTASP on the performance but still outperforms all baselines by a large margin, indicating that the learned dictionary has accumulated sufficient knowledge in the first ten tasks so that CoTASP can achieve competitive results without updating the dictionary for repetitive tasks.

Effect of key hyperparameters. CoTASP introduces the sparsity parameter $\lambda$, a hyperparameter that controls the trade-off between the used network capacity and the performance of the resulting policy. A larger value of $\lambda$ results in a more sparse policy sub-network, improving the usage efficiency of the meta-policy network’s capacity. But the cost is decreased performance on each task due to the loss of expressivity of the over-sparse task policy. According to the results in Fig. 4, CoTASP with $\lambda=1e^{-3}$ or $1e^{-4}$ achieves better trade-off between performance and usage efficiency than other structure-based methods (HAT and PackNet) on CW10 sequence.

4.4. Why does CoTASP work? An Empirical Study

In this section, we answer the following questions based on the phenomena observed from our empirical results: (1) Is the learned dictionary generalizable? (2) Does the sparse prompts generated by CoTASP capture the semantic correlations between tasks?

To answer the first question, we measure the change of the learned dictionary generalizable? (2) Does the sparse prompts generated by CoTASP capture the semantic correlations between tasks?

Continual World, CW10

Figure 4: Model Capacity Usage. Comparison of CoTASP with different sparsity $\lambda$ and two baselines on the efficiency of model capacity usage, i.e., ratio of used parameters vs. performance.
Continual Task Allocation in Meta-Policy Network via Sparse Prompting

Figure 5: (a): Steps (mean±std over 5 random seeds) required by new task adaptation after learning \( t \) tasks (fewer is better) on CW20 sequence. CoTASP spends the least steps on the new task adaptation and the steps decrease for later tasks, indicating a benefit of knowledge reuse. (b): The change of dictionary (averaged over 5 random seeds) of each hidden layer on CW20 sequence. The change is computed by \( \frac{1}{n_{\text{D}}(i)} \| D^{(i)}_l - D^{(i)}_{l-1} \|_2^2 \). The black dash-dotted line splits the x-axis in two parts, the first CW10 sequence (left part) and the repeated one (right part). It shows a fast convergence of the dictionary learning. Please refer to Sec. 4.4 for a detailed discussion.

Figure 6: Evaluation (mean±std over 6 random seeds) in Meta-World environments (Yu et al., 2019). The complete results in 5 environments are reported in Fig. 11 of Appendix.

Figure 7: Prompt (sub-network mask) similarity between two tasks \( i \) and \( j \) computed by \( \frac{1}{n_{\text{D}}(i)} \sum \| \phi(\cdot, l)_{i} \wedge \phi(\cdot, l)_{j} \|_1 \| \phi(\cdot, 0)_{i} \|_1 \). A layer-wise version is provided in Fig. 10 of Appendix.

will produce a “good” initial prompts by sparse coding of its task embedding. This significantly reduces the number of training steps needed to reach the success threshold, as demonstrated by Fig 5(a). Furthermore, we compare CoTASP with a SoTA meta-RL algorithm, CoMPS (Berseth et al., 2022), to show its superiority. Specifically, we pre-train CoTASP and CoMPS on CW10 sequence and then use the learned meta-policy as the initial policy to fine-tune unseen tasks, e.g., reach-v1 and button-press-v1. According to the results shown in Fig 6, CoTASP performs comparably to CoMPS but significantly better than the continual RL baseline (HAT) in terms of adaptation cost. However, due to the lack of mechanism against catastrophic forgetting, CoMPS adapts to the new task while its performance on previous tasks quickly degrades, resulting in worse average return across all tasks.

To answer the second question, we visualize the similarity (i.e., the overlap between two binary prompts) of the prompts generated by CoTASP between every two tasks over CW10 sequence in Fig. 7. The blue heatmap summarizes the similarity values averaged over all hidden layers. Specifically, the element on row \( i \) and column \( j \) is the averaged similarity value computed between task \( i \) and task \( j \). For task 2 and task 7, their task descriptions share the same manipulation primitive, i.e., pushing a puck. Hence, the prompts generated by solving the lasso problem in Eq. 3 are highly correlated. By contrast, for task 2 and task 7 with irrelevant task descriptions, CoTASP produces different prompts, reducing cross-task interference and improving plasticity.
5. Conclusions

We propose CoTASP to address two key challenges in continual RL: (1) training a meta-policy generalizable to all seen and even unseen tasks, and (2) efficiently extracting a task policy from the meta-policy. CoTASP learns a dictionary to produce sparse masks (prompts) to extract each task’s policy as a sub-network of the meta-policy and optimizes the sub-network via RL. This encourages knowledge sharing/reusing among relevant tasks while reducing harmful cross-task interference that causes forgetting and poor new-task adaptation. Without any experience replay, CoTASP achieves a significantly better plasticity-stability trade-off and more efficient network capacity allocation than baselines. Its extracted policies outperform all baselines on both previous and new tasks.

Acknowledgements

Yijun Yang and Yuhui Shi are partially supported by the Science and Technology Innovation Committee Foundation of Shenzhen under the Grant No. JCYJ20200109141235597 and ZDSYS201703031748284, National Science Foundation of China under grant number 61761136008, Shenzhen Peacock Plan under Grant No. KQTD2016112514355531, and Program for Guangdong Introducing Innovative and Entrepreneurial Teams under grant number 2017ZT07X386. We would like to thank ICML area chairs and anonymous reviewers for their efforts in reviewing this paper and their constructive comments!

References


Arora, S., Ge, R., Ma, T., and Moitra, A. Simple, efficient, and neural algorithms for sparse coding. In COLT, 2015.


A. Continual World benchmark

We visualize all of the tasks in Continual World in Fig. 8, and provide a description of these tasks in Table 3.

![Image of Continual World tasks](image)

Figure 8: Continual World benchmark adopts robotic manipulation tasks developed by Meta-World (Yu et al., 2019). Presented above is CW10 sequence.

<table>
<thead>
<tr>
<th>Index</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hammer-v1</td>
<td>Hammer a screw on the wall.</td>
</tr>
<tr>
<td>2</td>
<td>push-wall-v1</td>
<td>Bypass a wall and push a puck to a goal.</td>
</tr>
<tr>
<td>3</td>
<td>faucet-close-v1</td>
<td>Rotate the faucet clockwise.</td>
</tr>
<tr>
<td>4</td>
<td>push-back-v1</td>
<td>Pull a puck to a goal.</td>
</tr>
<tr>
<td>5</td>
<td>stick-pull-v1</td>
<td>Grasp a stick and pull a box with the stick.</td>
</tr>
<tr>
<td>6</td>
<td>handle-press-side-v1</td>
<td>Press a handle down sideways.</td>
</tr>
<tr>
<td>7</td>
<td>push-v1</td>
<td>Push the puck to a goal.</td>
</tr>
<tr>
<td>8</td>
<td>shelf-place-v1</td>
<td>Pick and place a puck onto a shelf.</td>
</tr>
<tr>
<td>9</td>
<td>window-close-v1</td>
<td>Push and close a window.</td>
</tr>
<tr>
<td>10</td>
<td>peg-unplug-side-v1</td>
<td>Unplug a peg sideways.</td>
</tr>
</tbody>
</table>

Table 3: A list of all of the Continual-World tasks and a description of each task. Listed above is CW10 sequence. CW20 sequence contains tasks from CW10 repeated twice. Tasks are learned sequentially, and 1M environment interaction steps are allowed per task.

B. Detailed Related Work

A promising strategy to address the stability-plasticity trade-off is using different modules, i.e., sub-networks within a fixed-capacity policy network, for each task. The modules of a new task are flexible to learn (for high plasticity), while the modules of past tasks are fixed (for high stability). In addition, there may exist shared modules that are trained across similar tasks, encouraging knowledge transfer. We introduce existing continual RL methods that adopt the strategy in two groups: connection-level and neuron-level methods, and also discuss their limitations.

Connection-level methods. Continual RL methods in this category, such as PackNet (Mallya & Lazebnik, 2018), Piggyback (Mallya et al., 2018), SupSup (Wortsman et al., 2020), BatchE (Wen et al., 2020), WSN (Kang et al., 2022) and NISPA (Gurbuz & Dovrolis, 2022), sequentially learn and select an optimal sub-network for each task. Specifically, they alternately optimize the model weights and the binary masks of sub-networks associated with each task while attempting to select a sparse set of weights to be activated by reusing weights of the prior sub-networks. Formally, given the state $s_t$ from task $t$, the action $a_t$ is computed as $a_t \sim \pi(s_t, \theta \otimes \phi_t)$ where $\phi_t$ denotes the binary masks for task $t$, $\otimes$ denotes element-wise product. PackNet generates $\phi$ via iterative pruning after learning each task, which preserves important weights for current task while leaving others available for the future tasks. Piggyback learns task-specific binary masks on the weights given a pre-trained model. SupSup uses a randomly initialized, fixed network and finds the optimal $\phi$ for each task. BatchE learns a shared weight matrix on the first task and then produces only a rank-one element-wise $\phi$ for each new task. WSN jointly
C. An Extended Description of Compared Methods

We now provide a detailed description of those baseline methods compared with CoTASP. Most of them are developed in the supervised continual learning setting, and require task-specific adaption to the continual RL setting. We refer to a groundbreaking work (Wolczyk et al., 2021) for the implementation of these methods. Concretely,

**Regularization-based methods.** This line of work reduces forgetting by restricting policy parameters important for the learned tasks. The most basic method is L2. It applies a $\ell_2$ regularization to the objective function, which keeps the parameters close to the previously optimized ones. In this method, each parameter is considered to be equally important for the previous tasks. In effect, prior work demonstrates that only a few parameters are essential for retaining performance on previous tasks. Based on this observation, EWC (Kirkpatrick et al., 2017) adopts the Fisher information as a metric to select important parameters for previous tasks. MAS (Aljundi et al., 2018) uses a parameter-wise regularizer and computes the regularization weights for each parameter by estimating their impact on the output of the policy. VCL (Nguyen et al., 2018) interprets the above methods from a Bayesian perspective and adopts variational inference to minimize the KL divergence between the posterior (current distribution of parameters) and the prior (distribution of the previously optimized parameters).

**Structure-based methods** preserve a set of parameters (i.e., sub-network), which is important for previous tasks, as a task-specific policy from a large “super-network”. PackNet (Mallya & Lazebnik, 2018) preserves a single sub-network for each task via iterative pruning after learning the task. HAT (Serrà et al., 2018) jointly learns the policy parameters and the corresponding sub-network masks through a gradient-based optimization. TaDeLL (Rostami et al., 2020) first integrates task descriptors into continual RL and then uses coupled dictionary learning to model the inter-task relationships between the task descriptions and the task-specific policies. However, dictionary learning in TaDeLL is defined on and applied to the policy parameters instead of task embeddings in CoTASP. A linear combination of policy parameters does not hold for nonlinear neural networks (Mendez et al., 2020) and the high dimensionality of the parameters will make the dictionary learning highly inefficient. So TaDeLL is not practical (in terms of both the efficiency and effectiveness) for nonlinear policy networks.

**Rehearsal-based methods** repeatedly replay buffered experiences from previous tasks and use them for retraining or as
constraints to alleviate catastrophic forgetting. We choose a simple reservoir strategy-based baseline, which replays all the data from the past when learning a new task, to investigate whether the naive method can increase the performance of continual RL agents. Moreover, A-GEM (Chaudhry et al., 2019) projects policy gradients from the current experiences to the closest gradients guaranteeing the average performance at previous tasks does not decrease. ClonEX-SAC (Wolczyzk et al., 2022) retains some samples from previous tasks and performs behavior cloning based on them to reduce forgetting. It achieved SoTA results on Continual World benchmarks.

D. Hyperparameter Details

We carefully tune the hyperparameters for a JAX implementation of the SAC algorithm (Bradbury et al., 2018; Kostrikov, 2021), and they are common for all baseline methods. Moreover, we tune the hyperparameters used for CoTASP using the final average performance on CW10 sequence as the objective. The search space and selected hyperparameters are presented in Table 4.

Table 4: Hyperparameters of CoTASP for Continual World experiments

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Search Space</th>
<th>Selected Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAC Hyperparameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actor hidden size</td>
<td>{256, 512, 1024, 2048}</td>
<td>1024</td>
</tr>
<tr>
<td>Critic hidden size</td>
<td>{128, 256, 512, 1024}</td>
<td>256</td>
</tr>
<tr>
<td># of hidden layers</td>
<td>{2, 3, 4}</td>
<td>4</td>
</tr>
<tr>
<td>Activation function</td>
<td>{Tanh, ReLU, LeakyReLU}</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Batch size</td>
<td>{128, 256}</td>
<td>256</td>
</tr>
<tr>
<td>Discount factor</td>
<td>-</td>
<td>0.99</td>
</tr>
<tr>
<td>Target entropy</td>
<td>{-4.0, -2.0, 0.0}</td>
<td>-2.0</td>
</tr>
<tr>
<td>Target interpolation</td>
<td>-</td>
<td>5 × 10^{-3}</td>
</tr>
<tr>
<td>Replay buffer size</td>
<td>-</td>
<td>10^6</td>
</tr>
<tr>
<td>Exploratory steps</td>
<td>-</td>
<td>10^4</td>
</tr>
<tr>
<td>Optimizer</td>
<td>-</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>{3 × 10^{-4}, 1 × 10^{-3}}</td>
<td>3 × 10^{-4}</td>
</tr>
<tr>
<td>CoTASP-specific Hyperparameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparsity parameter λ</td>
<td>{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}}</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>Coding algorithm</td>
<td>-</td>
<td>LARS-Lasso</td>
</tr>
<tr>
<td>Constant c</td>
<td>-</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Figure 9: Average success rate over CW20 sequence for each tested method.
Continual Task Allocation in Meta-Policy Network via Sparse Prompting

Figure 10: The similarity of each layer’s prompts between every two tasks over CW20 sequence. Note that the task sequence 11-20 are repetitive CW10 sequence. We observe that the prompts generated by CoTASp are highly correlated among repetitive tasks (e.g., tasks 1 and 11), reflecting the inherent capabilities of CoTASP for knowledge transfer and task inference.

Figure 11: Results of different methods in environments from Meta-World (Yu et al., 2019). All curves are the average of 6 runs with different seeds, and the shaded areas are standard errors of the mean.