Building a Subspace of Policies for Scalable Continual Learning

Jean-Baptiste Gaya[*]
Meta AI
CNRS-ISIR, Sorbonne University

Thang Doan[*]
McGill University, Mila

Lucas Caccia
McGill University, Mila

Laure Soulier
CNRS-ISIR, Sorbonne University

Ludovic Denoyer[†]
Meta AI

Roberta Raileanu
Meta AI

Abstract

The ability to continuously acquire new knowledge and skills is crucial for autonomous agents. However, existing methods are typically based on either fixed-size models that cannot capture many diverse behaviors, or growing-size models that scale poorly with the number of tasks. In this paper, we introduce Continual Subspace of Policies (CSP), a method that iteratively learns a subspace of policies in the continual reinforcement learning setting where tasks are presented sequentially. The subspace's high expressivity allows our method to strike a good balance between *stability* (*i.e.* not forgetting prior tasks) and *plasticity* (*i.e.* learning new tasks), while the number of parameters grows *sublinearly* with the number of tasks. In addition, CSP displays good *transfer*, being able to quickly adapt to new tasks including combinations of previously seen ones without additional training. Finally, CSP outperforms state-of-the-art methods on a wide range of scenarios in two different domains. An interactive visualization of the subspace can be found at `https://continual-subspace-policies-streamlit-app-gofujp.streamlitapp.com/`.

1 Introduction

Developing autonomous agents that can continuously acquire new knowledge and skills is a key open challenge in AI. This problem is referred to as continual reinforcement learning (CRL) and solving it is crucial for large-scale deployment of autonomous agents in non-stationary domains such as robotics or dialogue systems. In recent years, there has been a growing interest in this problem [38, 22, 40, 36]. As suggested in [40], the balance between **stability** (*i.e.* no forgetting of prior tasks), **plasticity** (*i.e.* positive transfer to new tasks including combinations of previously seen ones and the ability to express many diverse behaviors), and **scalability** (*i.e.* memory increases sublinearly with the number of tasks) is crucial for designing effective CRL methods.

While current methods perform well along some of these dimensions, they tend to suffer along others [32, 49]. One class of methods aims to alleviate forgetting by storing previously trained models [38, 9, 48] or maintaining replay buffers with trajectories from prior tasks [24, 18, 34, 36]. However, these methods scale poorly with memory and compute, especially as the number and complexity of the tasks increase, rendering them unfeasible for real-world applications. Another

[*]Equal contribution.
[†]Now at Ubisoft.
. Correspondance to jbgaya@fb.com

36th *Decision Awareness in Reinforcement Learning Workshop at the 39th International Conference on Machine Learning (ICML)*, Baltimore, Maryland, USA, 2022.

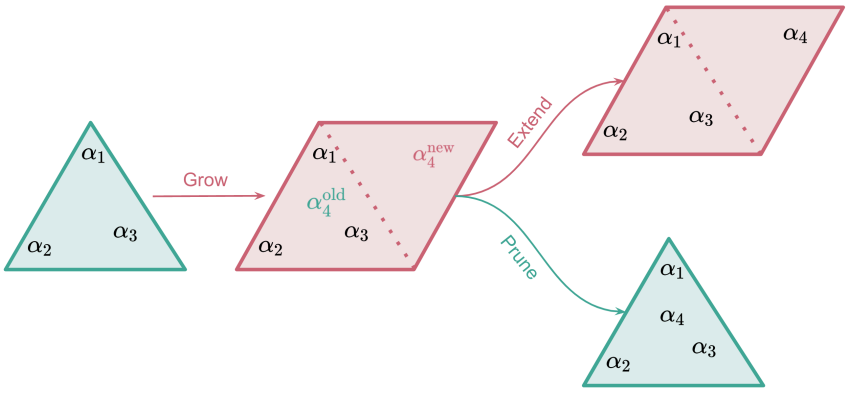


Figure 1: **Continual Subspace of Policies (CSP)** iteratively learns a subspace of policies in the continual RL setting. At every stage during training, the subspace is a simplex defined by a set of anchors (*i.e.* vertices). Any policy (*i.e.* point) in this simplex can be represented as a convex combination α of the anchor parameters. α_i defines the best policy in the subspace for task i. When the agent encounters a new task, CSP tentatively *grows* the subspace by adding a new anchor. If the new task i is very different from previously seen ones, a better policy α_i^{new} can usually be learned in the new subspace. In this case, CSP *extends* the subspace by keeping the new anchor. If the new task bear some similarities to previously seen ones, a good policy α_i^{old} can typically be found in the old subspace. In this case, CSP *prunes* the subspace by removing the new anchor.

body of work focuses on improving transfer to new tasks [17, 37, 23, 5, 20, 46], but these methods tend to forget previously learned skills. A few approaches attempt to find a better balance between transfer and forgetting [22, 40]. However, because they train a single model, these methods struggle to express a wide range of diverse behaviors.

In this paper, we take inspiration from the mode connectivity literature [13] to develop a novel CRL method by iteratively learning a subspace of policies [14]. Our method, **Continual Subspace of Policies** or **CSP** for short, aims to strike a good balance between stability, plasticity, and scalability. Instead of learning a single policy, CSP maintains an entire subspace of policies defined as a convex hull in parameter space. The vertices of this convex hull are called anchors, with each anchor representing the parameters of a policy. This subspace captures a large number of diverse behaviors which enables efficient training on a wide range of tasks. At every stage of the CRL process, the best found policy for a previously seen task is represented as a single point in the current subspace (*i.e.* unique convex combination of the anchors). This enables cheap storage and retrieval of prior solutions. If a new task is a combination of previously seen ones, a good policy can be found in the current subspace without increasing the number of parameters. On the other hand, if a new task is very different from previously seen ones, CSP extends the current subspace by adding another anchor, and learns a new policy in the extended subspace. In this case, the expressivity of the subspace will increase allowing our model to deal with more diverse tasks in the future, and thus ensuring good plasticity. See Figure 1 for an overview of our method.

To summarize, in this paper we make the following contributions: (i) we introduce CSP, a new CRL method based on adaptively building a subspace of policies which allows to cheaply store and retrieve solutions to prior tasks, provides positive transfer and grows sublinearly w.r.t the number of tasks, (ii) we design a number of continuous control environments to evaluate specific abilities of CRL agents, (iii) we illustrate our method's ability to learn diverse task sequences and transfer to compositional tasks, (iv) and we show that CSP outperforms state-of-the-art CRL methods on a suite of continuous control scenarios in both Brax [12] and Continual World [49].

2 Problem Setting

A continual learning problem is defined by a sequence of N tasks denoted $t_1, ..., t_N$. The task i is defined by a Markov Decision Process (MDP) $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i, r_i, \gamma \rangle$ with a set of states $\mathcal{S}_i$, a set of actions $\mathcal{A}_i$, a transition function $\mathcal{T}_i : \mathcal{S}_i \times \mathcal{A}_i \to \mathcal{P}(\mathcal{S}_i)$, and a reward function $r_i : \mathcal{S}_i \times \mathcal{A}_i \to \mathbb{R}$. For the sake of simplicity, we consider here that all the tasks have the same state space denoted $\mathcal{S}$ and action space $\mathcal{A}$. We also define a global policy $\Pi : [1..N] \times \mathcal{Z} \to (\mathcal{S} \to \mathcal{P}(\mathcal{A}))$ which takes as input

a task id i and a sequence of tasks $\mathcal{Z}$, and outputs the policy which should be used when interacting with task i. $\mathcal{P}(\mathcal{A})$ is a probability distribution over the action space.

Budgeted Constraints and Training. We consider that each task i is associated with a training budget denoted b_i. It means that, when facing task i, the learning algorithm can only execute b_i actions. Then the system will switch to task t_{i+1} and will no longer have access to transitions from the MDP $\mathcal{M}_i$. We do not assume access to a replay buffer of trajectories from past MDPs since this would lead to a significant increase in memory cost.

3 Continual Subspace of Policies (CSP)

3.1 Subspace of Policies

Our work builds on top of [14] by leveraging a subspace of policies. However, instead of using the subspace to train on a single environment and adapt to new ones at test time, we use it to efficiently learn tasks sequentially in the continual RL setting. This requires designing a new approach for learning the subspace, as detailed below.

Let $\theta \in \mathbb{R}^d$ represent the parameters of a policy, which we denote by $\pi(a|s, \theta)$. A subspace of policies is a simplex in $\mathbb{R}^d$ defined by a set of anchors $\Theta = \{\theta_1, ..., \theta_k\}$, where any policy in this simplex can be represented as a convex combination of the anchor parameters. Let the weight vector $\alpha \in \mathbb{R}^k_+$ with $\|\alpha\|_1 = 1$ denote the coefficients of the convex combination. Therefore, each value of α uniquely defines a policy with $\pi(a|s, [\alpha, \Theta]) = \pi(a|s, \sum \alpha_i \theta_i)$. Throughout the paper, we sometimes refer to the weight vector α and its corresponding policy $\pi(a|s, [\alpha, \Theta])$ interchangeably (since a one-to-one mapping is defined at each point in training).

3.2 Learning Algorithm

We propose to adaptively construct a subspace of policies for the continual RL setting in which an agent learns tasks sequentially. We call our method **Continual Subspace of Policies** or **CSP** for short.

Our model builds a sequence $\Theta_1, ..., \Theta_N$ of subspaces, one new subspace after each training task, with each subspace extending the previous one. Note that $|\Theta_j| \leq j, \ \forall \ j \in 1...N$ since the subspace grows sublinearly with the number of tasks (as explained below). Hence, the number of anchors m of a subspace Θ_j is not the same as the number of tasks j used to create Θ_j. The learned policies are represented as single points in these subspaces. At each stage, CSP maintains both a set of anchors defining the current subspace, as well as the weights α corresponding to the best policies found for all prior tasks. The (best found) policy for task i after training on the first j tasks is denoted as $\pi_i^j(a|s)$ and can be represented as a point in the subspace Θ_j with a weight vector denoted α_i^j such that $\pi_i^j(a|s) = \pi(a|s, [\alpha_i^j, \Theta_j])$.

Given a set of anchors Θ_j and a set of previously learned policies $\{\alpha_1^j, ..., \alpha_j^j\}$, updating our model on the new task t_{j+1} produces a new subspace Θ_{j+1} and a new set of weights $\{\alpha_1^{j+1}, ..., \alpha_{j+1}^{j+1}\}$. There are two possible cases. One possibility is that the current subspace already contains a good policy for the new task, so we just need to find the weight vector corresponding to a policy which performs well on the new task. The other possibility is that the current subspace does not contain a good policy for the new task. In this case, the algorithm produces a new subspace by adding one anchor to the previous one (see next section), and converts the previous policies to be compatible with this new subspace.

To achieve this, CSP operates in two phases:

1. *Grow* the current subspace by adding a new anchor and learning the best possible policy for task $j + 1$ in this subspace.
2. Compare the quality of this policy with the best possible policy expressed in the previous subspace. Based on this comparison, decide whether to *extend* the subspace to match the new one or *prune* it back to the previous one.

See Algorithm 17 for the pseudo-code of our method. We now describe in detail the two phases of the learning process, namely how we grow the subspace and how we decide whether to extend or prune the subspace.

Algorithm 1: Continual Subspace of Policies (CSP)

Input: $\theta_1,\dots,\theta_j$ (previous anchors), B (budget), ϵ (threshold)
Initialize: W_ϕ (subspace critic), $\mathcal{B}uf$ (replay buffer)

Define $\theta_{j+1} \leftarrow \frac{1}{j}\sum_{i=1}^{j}\theta_i$ (new anchor) ; `// Grow the Subspace`
for $i = 1, ..., B$ **do**
  Sample $\alpha \sim Dir\,(j+1)$
  Set policy parameters $\theta_\alpha \leftarrow \sum_{i=1}^{j+1}\alpha_i\theta_i$
  for $l = 1, ..., K$ **do**
    Collect and store (s, a, r, s', α) in $\mathcal{B}uf$ by sampling $a \sim \pi_{\theta_\alpha}(\cdot|s)$
  end
  if *time to update* **then**
    Update $\pi_{\theta_{j+1}}$ and W_ϕ using the SAC algorithm and the replay buffer $\mathcal{B}uf$
  end
end
Use $\mathcal{B}uf$ and W_ϕ to estimate: ; `// Extend or Prune the Subspace`

$$\alpha^{\text{old}} \leftarrow \underset{(\alpha,0)\text{ with }\alpha\in\mathbb{R}_+^m,\|\alpha\|_1=1}{\arg\max} W_\phi(\alpha)$$
$$\alpha^{\text{new}} \leftarrow \underset{\alpha\in\mathbb{R}_+^{m+1},\|\alpha\|_1=1}{\arg\max} W_\phi(\alpha)$$

if $W_\phi(\cdot,\alpha^{\text{new}}) > (1+\epsilon)\cdot W_\phi(\cdot,\alpha^{\text{old}})$ **then**
  Return: $\theta_1,\dots,\theta_j,\theta_{j+1},\alpha^{\text{new}}$; `// Extend`
else
  Return: $\theta_1,\dots,\theta_j,\alpha^{\text{old}}$; `// Prune`
end

3.3 Grow the Subspace

Given the current subspace Θ_j composed of $m \leq j$ anchors (with j being the number of tasks seen so far), a new subspace $\tilde{\Theta}_{j+1}$ is built as follows. First a new anchor denoted θ_{j+1} is added to the set of anchors such that $\tilde{\Theta}_{j+1} = \Theta_j \bigcup \{\theta_{j+1}\}$. With all previous anchors frozen, we train the new anchor by sampling α values from a Dirichlet distribution and derive updates from a similar objective as the one proposed in [14]:

$$\theta_{j+1} = \arg\max_{\theta} \mathbb{E}_{\alpha\sim Dir,\,\tau\sim\pi(a|s,[\alpha,\Theta_j\cup\{\theta\}])}[R_{j+1}(\tau)], \tag{1}$$

where $R_{j+1}(\tau)$ is the return obtained on task $j+1$ throughout trajectory τ which was generated using policy $\pi(a|s,[\alpha,\tilde{\Theta}_{j+1})]$. Note that the anchor is trained such that not only one but all possible values of α tend to produce a good policy. The resulting subspace thus aims at containing as many good policies as possible for task $j+1$.

3.4 Extend or Prune the Subspace

To decide if the new anchor is kept, we propose to simply compare the best possible policy (for task $j+1$) in the new subspace with the best possible policy for the same task in the previous subspace (*i.e.* without using the new anchor). Each policy could be evaluated via Monte-Carlo (MC) estimates by doing additional rollouts in the environment and recording the average performance. However, this typically requires a large number (*e.g.* millions) of interactions which may be impossible with a limited budget. Thus, we propose an alternative procedure to make this evaluation sample efficient.

For each task j and corresponding subspace Θ_j, our algorithm also learns a Q-function $Q(s,a,\alpha)$ which is trained to predict the expected return on task j for all possible states, actions, and all possible

α's in the corresponding subspace. Our algorithm is based on SAC [15], so for each task, we collect a replay buffer of interactions $\mathcal{B}uf$ which contains all states and actions seen by the agent during training. Thus, the Q-function $Q(s, a, \alpha)$ can help us directly estimate the quality $W(\alpha)$ of the policy represented by the weights α in the new subspace which can be computed as the average over all states and actions in the replay buffer:

$$W(\alpha) = \mathbb{E}_{s,a\sim\mathcal{B}uf} Q(s, a, \alpha) \tag{2}$$

It is thus possible to compute the value of α corresponding to the best policy in the extended subspace (denoted $\alpha^{\text{new}} \in \mathbb{R}_+^{m+1}, |\alpha^{\text{new}}\|_1 = 1$):

$$\alpha^{\text{new}} = \underset{\alpha\in\mathbb{R}_+^{m+1}, \|\alpha\|_1=1}{\arg\max} W(\alpha), \tag{3}$$

as well as the value of α corresponding to the best policy in the previous subspace (denoted $\alpha^{\text{old}} \in \mathbb{R}_+^m, \|\alpha^{\text{old}}\|_1 = 1$):

$$\alpha^{\text{old}} = \underset{(\alpha,0) \text{ with } \alpha\in\mathbb{R}_+^m, \|\alpha\|_1=1}{\arg\max} W(\alpha). \tag{4}$$

In practice, α^{new} and α^{old} are estimated by uniformly sampling a number of α's in the corresponding subspace as well as a number of states and actions from the buffer.

The quality of the new subspace and the previous one can thus be evaluated by comparing $W(\alpha^{\text{new}})$ and $W(\alpha^{\text{old}})$. If $W(\alpha^{\text{new}}) > (1 + \epsilon) \cdot W(\alpha^{\text{old}})$, the subspace is extended to the new subspace (*i.e.* the one after the grow phase): $\Theta_{j+1} = \tilde{\Theta}_{j+1}$. Otherwise, the subspace is pruned back to the old subspace (*i.e.* the one before the grow phase): $\Theta_{j+1} = \Theta_j$. Note that, if the subspace is extended, the previously learned policies have to be mapped in the new subspace such that $\alpha_i^{j+1} \in \mathbb{R}_+^{j+1}$ *i.e.* $\forall\ i \leq j,\ \alpha_i^{j+1} := (\alpha_i^j, 0)$ and $\alpha_{j+1}^{j+1} := \alpha^{\text{new}}$. If the subspace is not extended, then old values can be kept *i.e.* $\forall\ i \leq j,\ \alpha_i^{j+1} := \alpha_i^j$ and $\alpha_{j+1}^{j+1} := \alpha^{\text{old}}$.

The policy $\pi_{\theta_{j+1}}$ and value function W_ϕ are updated using SAC [15]. See Appendix A for more details about the algorithm and our implementation of CSP.

3.5 Scalability of CSP

By having access to an infinite number of policies, the subspace is highly expressive so it can capture a wide range of diverse behaviors. This enables positive transfer to many new tasks without the need for additional parameters. As a consequence, the number of parameters scales *sublinearly* with the number of tasks. The final size of the subspace depends on the sequence of tasks, with longer and more diverse sequences requiring more anchors. The speed of growth is controlled by the threshold ϵ which trades-off performance gains for memory efficiency (*i.e.* the higher the ϵ the more performance losses are tolerated to reduce memory costs). See Appendix A for more intuition about CSP.

4 Experiments

4.1 Environments

We evaluate CSP on a number of CRL scenarios focused on both locomotion using Brax [12] as well as robotic manipulation using Continual World [49]. For Continual World, we ran experiments on all 8 triplet sequences proposed in the paper. These tasks were specifically designed to challenge both transfer and forgetting (see Appendix C for more details).

For Brax, we perform an in-depth study by designing a number of continuous control scenarios to separately evaluate capabilities specific to CRL agents such as forgetting, transfer, or scalability. For each capability, we create two scenarios: one based on HalfCheetah and one based on Ant. First, we generate a large number of environments by changing the dynamics of the standard environment. We aimed to ground our environment variations in realistic situations such as increased or decreased gravity, friction, or limb lengths. Then, we design task sequences based on the transfer matrices for these environments, following [49]. Unless mentioned otherwise, each CRL scenario has 8 tasks in the sequence. See Appendix C for more details about our designed scenarios.

Forgetting and **Transfer** are evaluated by selecting sequences of tasks with significant negative and positive backward transfer according to the matrix. **Robustness** probes whether the method can deal with adversarial perturbations in the environment such as the actions being inverted for each task, which poses a significant challenge for both forgetting and transfer. **Compositionality** tests the agent's ability of combining two previously seen skills to solve a new task. **Scalability** is tested by looking at how performance and memory change with the number of tasks. **Learning Efficiency** is tested by using smaller budgets (see Section 2).

4.2 Baselines

We compare CSP with a number of state-of-the-art CRL methods such as **PNN** [38], **EWC** [22], as well as with **FT-1** which finetunes a single model on the entire sequence of tasks, **FT-L2** which is like FT-1 with an additional L_2 regularization applied during finetuning. We also compare CSP with **SAC-N** which trains one model for each task from scratch. While SAC-N avoids forgetting, it cannot transfer knowledge across tasks. Finally, we compare CSP with a method called **FT-N** which combines the best of both SAC-N and FT-1. Like SAC-N, it stores one model per task after training on it and like FT-1, it finetunes the previous model to promote transfer. However, FT-N and SAC-N scale poorly (*i.e.* linearly with the number of tasks) with both memory and compute, which makes them unfeasible for real-world applications. Note that our method is not directly comparable with CLEAR [36] since we assume no access to data from prior tasks. Storing data from all prior tasks as CLEAR does, is impractical for long task sequences due to prohibitive memory costs. All methods use SAC [15] as a base algorithm and have been equally well-tuned on our tasks. Full experimental details can be found in Appendix B.

4.3 Ablations

We also perform a number of ablations to understand how the different components of CSP influence performance. First, we want to understand how good the critic is at finding the best policy in the subspace. To do this, we implement **CSP-MID** which uses the subspace's midpoint a proxy for the best policy in the subspace for task $j \ \forall \ j \in 1...N$ rather than using the critic to select the best policy. The midpoint is defined as $\theta_{mid} \coloneqq \frac{1}{m} \sum_{i=1}^{m} \theta_i$ where $m = |\Theta_j|$. The midpoint is a natural choice as it uniformly combines all prior anchors so it contains knowledge of all prior tasks.

At the other end of the spectrum, we have **CSP-ORACLE** which selects the best policy by sampling a large number (*i.e.* thousands) of policies in the subspace and computing Monte-Carlo estimates of their returns. These estimates are expected to be more accurate than the critic's, so CSP-ORACLE can be considered an upper bound to CSP. However, CSP-ORACLE is less efficient than CSP since it requires singificantly more interactions with the environment (*i.e.* millions).

We also want to understand how much performance we lose, if any, by not adding one anchor for each task. To do this, we run **CSP-LINEAR** which always extends the subspace. In addition, we vary the threshold ϵ used to decide whether to extend the subspace based on how much performance is gained by doing so. This analysis can shed more light on the trade-off between performance gain and memory cost as the threshold varies.

The paper includes a total of 4116 experiments, over 25 different CRL scenarios with 36 different RL environments. Each experiment takes on average 15 hours on 1 GPU (NVIDIA V100). The means and standard deviations are computed over 3 runs.

4.4 Metrics

Agents are evaluated across a range of commonly used CRL metrics [49, 32]. For brevity, following the notation in Section 2, we will use $\pi_i^j \coloneqq \Pi(i, [t_1, \ldots, t_j])$ to denote the policy selected by Π for task i obtained after training on the sequence of tasks $t_1, ..., t_j$ (in this order with j included). Note that Π can also be defined for a single task *i.e.* $\pi_i^{t_i} \coloneqq \Pi(i, [t_i])$ is the policy for task i after training the system only on task t_i (for the corresponding budget b_i).

Average Performance is the average performance of the model across all tasks, after training on the entire sequence of tasks. Let $P_i(\pi)$ be the performance of the system on task i using policy π defined as $P_i(\pi) \coloneqq \mathbb{E}_{\pi, \mathcal{T}_i} \left[\sum r_i \left(s, a \right) \right]$ where $r_i(s, a)$ is the reward received on task i when taking action a in state s. Then, the final average performance across all tasks can be computed as $P \coloneqq \frac{1}{N} \sum_{i=1}^{N} P_i(\pi_i^N)$.

Table 1: Aggregated results across 4 of our CRL scenarios based on HalfCheetah, each consisting of a sequence of 8 tasks. These scenarios were designed to test forgetting, transfer, compositionality, and robustness. CSP outperforms all baselines, while scaling sublinearly with the number of tasks. CSP's performance is also not too far from that of CSP-ORACLE which uses millions of interactions to find the best policy whereas CSP uses none.

Method	Performance	Transfer	Forgetting	Growing Factor
FT-1	0.75 ± 0.16	0.20 ± 0.14	-0.45 ± 0.07	**1.0**
FT-L2	0.81 ± 0.09	0.09 ± 0.14	-0.28 ± 0.10	2.0
EWC	0.98 ± 0.14	0.14 ± 0.11	-0.28 ± 0.10	3.0
PNN	1.06 ± 0.17	0.06 ± 0.18	0.00 ± 0.00	47.3
SAC-N	1.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	8.0
FT-N	1.22 ± 0.10	0.22 ± 0.10	0.00 ± 0.00	8.0
CSP-MID	$0.64 \pm 0.17.$	-0.22 ± 0.17	-0.15 ± 0.09	4.0 ± 0.7
CSP-ORACLE	1.53 ± 0.09	0.48 ± 0.1	0.05 ± 0.07	4.0 ± 0.7
CSP (ours)	**1.32 ± 0.07**	**0.31 ± 0.07**	**0.00 ± 0.03**	4.0 ± 0.7

Forward Transfer measures how much a CRL system is able to transfer knowledge from task to task. At task i, it compares the performance of the of the system trained on all previous tasks $t_1, ..., t_i$ to the same model trained solely on task t_i. This measure is defined as $\mathrm{FT}_i \coloneqq \frac{P_i(\pi_i^i) - P_i(\pi_i^{t_i})}{1 - P_i(\pi_i^{t_i})}$, and thus the forward transfer for all tasks is $\mathrm{FT} \coloneqq \frac{1}{N} \sum_{i=1}^{N} \mathrm{FT}_i$.

Forgetting evaluates how much a system has forgotten about task i after training on the full sequence of tasks. It thus compares the performance of policy π_i^i with the performance of policy π_i^N and is defined as $\mathrm{F}_i \coloneqq P_i(\pi_i^N) - P_i(\pi_i^i)$. Similarly to the average transfer, we report the average forgetting across all tasks $\mathrm{F} \coloneqq \frac{1}{N} \sum_{i=1}^{N} \mathrm{F}_i$.

Growing Factor measures how much the size of the model grows during training. Formally, it is defined as the ratio between the number of parameters of the model at task t_N and its size at task t_1. It is defined as $GF \coloneqq \frac{|\psi_N|}{|\psi_1|}$ where ψ_i is the model's set of parameters after training on task i.

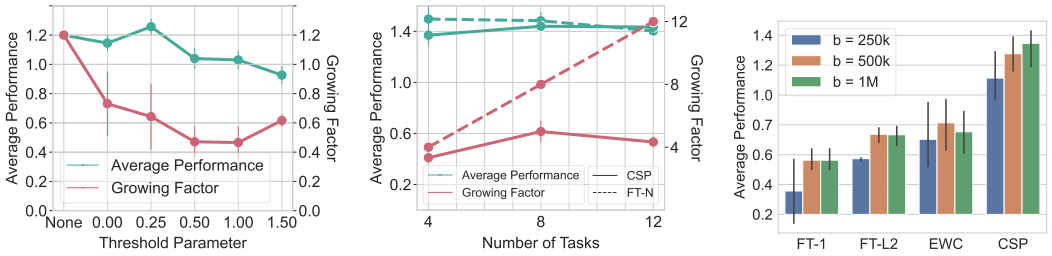
5 Results

5.1 Performance on Brax

Table 1 shows the aggregated results across 4 of our CRL scenarios based on HalfCheetah, namely the ones that probe forgetting, transfer, robustness, and compositionality. The other scenarios are studied separately to avoid skewing the results, given that they use different regimes (*e.g.* budget, number of tasks). The results are normalized using the performance of the SAC-N agents. See Appendix D for results on each scenario, including those based on Ant.

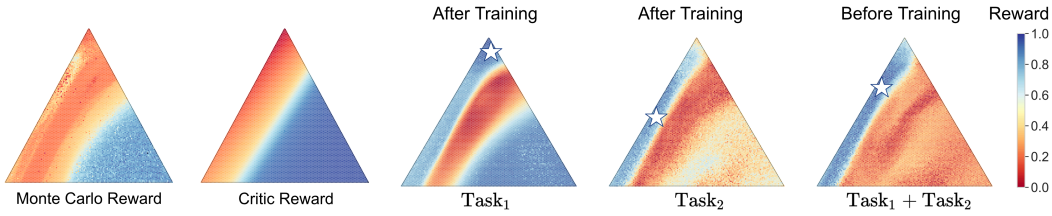
CSP outperforms all the baselines along final average performance, forward transfer, and forgetting, while the memory costs are kept relatively low. Naive methods like FT-1 have low memory costs and good transfer, but suffer from catastrophic forgetting. In contrast, methods that aim to reduce forgetting such as FT-L2 or EWC do so at the expense of transfer. The only competitive methods in terms of performance are the ones where the number of parameters increases at least linearly with the number of tasks, such as PNN, SAC-N, and FT-N. These methods have no forgetting because they store the models trained on each task. SAC-N has no transfer since it trains each model from scratch, while FT-N promotes transfer as it finetunes the previous model. However, these methods are unfeasible for real-world scenarios with long task sequences due to their poor scalability. To summarize, **CSP outperforms by a wide margin methods with good scalability but poor stability or plasticity**. At the same time, **CSP is significantly more memory efficient than methods with good stability or plasticity but poor scalability**. In conclusion, CSP strikes a better balance between stability, plasticity, and scalability than prior CRL approaches.

Varying the Threshold. Figure 2a shows how performance and memory vary with the threshold ϵ used to decide whether to extend the subspace. As expected, as ϵ increases, performance decreases, but so do memory costs. Note that *performance is still above* 80% *even as the memory costs are cut*



(a) Varying the Threshold (b) Scalability (c) Learning Efficiency

Figure 2: (a) shows how performance and memory vary with the threshold, (b) shows how these scale with the number of tasks, and (c) shows performance for different budgets.



(a) Smoothness and Critic Accuracy (b) Diversity and Compositionality

Figure 3: (a) shows the value of each policy in the subspace estimated using both Monte-Carlo simulations (left) and our critic's predictions (right), demonstrating both that the subspace is smooth and that our critic learns accurate estimates of the reward and can thus find the best policy in the subspace; (b) shows the subspace for three different tasks, the third being a combination of the first two, demonstrating that the subspace already contains a policy with high reward on the compositional task before being trained on it. The star represents the best policy in the subspace for the corresponding task. The subspace contains policies covering the whole spectrum of rewards, suggesting that it captures diverse behaviors.

by more than 60%, relative to CSP-LINEAR which trains an anchor (*i.e.* new set of parameters) for each task. Thus, CSP can drastically reduce memory costs without significantly hurting performance. Practitioners can set the threshold to trade-off performance gains for memory efficiency (*i.e.* the higher the ϵ the more performance losses are tolerated in order to reduce memory costs).

Scalability. Figure 2b shows how performance and memory scale with the number of tasks in the sequence (on the compositionality scenario), for both CSP and FT-N which is our strongest baseline. CSP maintains both strong performance and low memory cost even as the number of tasks increases. In contrast, FT-N's growing factor scales linearly, which makes it impractical for long task sequences.

Learning Efficiency. Figure 2c shows the average performance for three different budgets (*i.e.* number of interactions allowed for each task) on the robustness scenario, comparing CSP with the more scalable baselines FT-1, FT-L2, and EWC. These results demonstrate that CSP can learn efficiently even with a reduced budget, outperforming the baselines. By keeping track of an infinite number of policies, CSP enables positive transfer to a wide range of tasks without requiring many interactions to learn new skills.

5.2 Properties of the Subspace

Evaluating the Subspace. Table 1 shows that as expected, using the middle of the subspace as a proxy for the best policy is suboptimal (*i.e.* CSP-MID is worse than CSP). Importantly, CSP's performance is not too far from that of CSP-ORACLE, indicating that *the learned critic can be effectively used to find the best policy in the subspace for a given task*, despite the fact that it uses *no additional interactions* with the environment to do so. In contrast, CSP-ORACLE requires *millions of samples* to select the best policy in the subspace using Monte-Carlo rollouts.

Smoothness and Critic Accuracy. Figure 3a shows a snapshot of a trained subspace, along with the expected reward of all policies in the subspace for a given task. The expected reward is computed

Table 2: Results on 4 robotic manipulation scenarios from Continual World.

Method	T1	T2	T3	T4
CSP (ours)	0.76 ± 0.20	0.79 ± 0.03	$\mathbf{0.82 \pm 0.08}$	$\mathbf{0.58 \pm 0.09}$
FT-N	$\mathbf{0.77 \pm 0.08}$	$\mathbf{0.86 \pm 0.10}$	0.78 ± 0.15	0.49 ± 0.14

using both Monte-Carlo rollouts, as well as our critic's predictions using Equation 2. As illustrated, the subspace is smooth, and the critic's estimates are highly accurate.

Diversity and Compositionality. Figure 3b illustrates that the subspace contains behaviors composed of previously learned skills. This allows CSP to efficiently find good policies for many different combinations of prior tasks without the need of training additional parameters. The figure also shows that for a given task, the policies in the subspace cover the whole spectrum of rewards, thus emphasizing the diversity of behaviors expressed by the subspace. See Appendix E for more quantitative and qualitative analyses of the subspace.

5.3 Performance on Continual World

Table 2 shows results for 4 challenging robotic manipulation scenarios from Continual World, each of them containing 3 tasks. These scenarios have been designed such that the second task reduces transfer from the first to the third task. CSP achieves comparable performance with FT-N, which is state-of-the-art on these scenarios [49]. However, in contrast with CSP, FT-N cannot be used for long task sequences due to prohibitive memory and computational costs. See Appendix D for additional results on Continual World.

6 Related Work

Continual Reinforcement Learning. CRL methods aim to avoid catastrophic forgetting, as well as enable transfer to new tasks, while remaining scalable to a large number of tasks. In the past few years, multiple approaches have been proposed to address one or more of these challenges [29, 21, 32, 49].

One class of methods focuses on preventing catastrophic *forgetting*. Some algorithms achieve this by storing the parameters of models trained on prior tasks [38, 9, 48]. However, these methods scale poorly with the number of tasks (*i.e.* at least linearly) in both compute and memory, which makes them unfeasible for more realistic scenarios. Other methods maintain a buffer of experience from prior tasks to alleviate forgetting [24, 18, 34, 36, 7]. However, this is also not scalable as the memory cost increases significantly with the number and complexity of the tasks. In addition, many real-world applications in domains like healthcare or insurance prevent data storage due to privacy or ethical concerns. Another class of methods focuses on improving *transfer* to new tasks. Naive approaches like finetuning that train a single model on each new task provide good scalability and plasticity, but suffer from catastrophic forgetting of previously learned tasks. To overcome this effect, methods like elastic weight consolidation (EWC) [22] alleviate catastrophic forgetting by constraining how much the network's weights change, but this can in turn reduce plasticity. Another class of methods employs knowledge distillation to improve transfer in CRL [17, 37, 23, 40, 5, 20, 46]. However, since these methods train a single network, they struggle to capture a large number of diverse behaviors.

There is also a large body of work which leverages the shared structure of the tasks [50, 30, 25, 27, 1, 43], meta-learning [19, 44, 4, 8, 33, 52, 39], or generative models [35, 42, 41, 2] to improve CRL agents, but these are less related to CSP and not very competitive.

Mode Connectivity and Neural Network Subspaces. Based on prior studies of mode connectivity [13, 11, 28], [47, 6] proposed the neural network subspaces to connect the solutions of different models. More similar to our work, [10] leverages subspace properties to mitigate forgetting on a sequence of supervised learning tasks, but doesn't focus on other aspects of the continual learning problem. Our work was also inspired by [14] which learns a subspace of policies for a single task for fast adaptation to new environment. In contrast to [14], we consider the continual RL setting and adaptively grow the subspace as the agent encounters new tasks. Our work is first to demonstrate the effectiveness of mode connectivity for the continual RL setting.

7 Discussion

In this paper, we propose CSP, a new CRL method which adaptively builds a subspace of policies. Learning about a large number of diverse behaviors allows CSP to strike a good balance between stability, plasticity, and scalability. We demonstrate our method's superior capabilities relative to state-of-the-art CRL approaches on a wide range of continuous control scenarios. Our paper is first to use a subspace of models for CRL, and thus opens up many interesting directions for future work. For example, one can assume a fixed size for the subspace and update the anchors whenever the agent encounters a new task. Another promising direction is to leverage the structure of the subspace to meta-learn or search for good policies on a given task. Finally, active learning techniques could be employed to more efficiently evaluate the subspace.

References

[1] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael L. Littman. Toward good abstractions for lifelong learning. 2017.

[2] Craig Atkinson, B. McCane, Lech Szymanski, and Anthony V. Robins. Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. *ArXiv*, abs/1812.02464, 2021.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Shawn L. E. Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O. Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. *ArXiv*, abs/2002.09571, 2020.

[5] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

[6] Gregory W. Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 769–779. PMLR, 2021.

[7] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[8] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexandre Lacoste, David Vazquez, and Laurent Charlin. Online fast adaptation and knowledge accumulation: a new approach to continual learning. *NeurIPS*, 2020.

[9] Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno A. Olshausen. Superposition of many models into one. *ArXiv*, abs/1902.05522, 2019.

[10] Thang Doan, Seyed Iman Mirzadeh, Joelle Pineau, and Mehrdad Farajtabar. Efficient continual learning ensembles in neural network subspaces. *arXiv preprint arXiv:2202.09826*, 2022.

[11] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML 2018: Thirty-fifth International Conference on Machine Learning*, pages 1308–1317, 2018.

[12] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[13] T. Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018.

[14] Jean-Baptiste Gaya, Laure Soulier, and Ludovic Denoyer. Learning a subspace of policies for online adaptation in reinforcement learning. *ArXiv*, abs/2110.05169, 2021.

[15] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[16] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018.

[17] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[18] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI*, 2018.

[19] Khurram Javed and Martha White. Meta-learning representations for continual learning. In *NeurIPS*, 2019.

[20] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Policy consolidation for continual reinforcement learning. *ArXiv*, abs/1902.00255, 2019.

[21] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *ArXiv*, abs/2012.13490, 2020.

[22] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526, 2017.

[23] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:2935–2947, 2018.

[24] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, 2017.

[25] Kevin Lu, Aditya Grover, P. Abbeel, and Igor Mordatch. Reset-free lifelong learning with skill-space planning. *ArXiv*, abs/2012.03548, 2021.

[26] Song Duong Jean-Baptiste Gaya Pierre-Alexandre Kamienny Daniel H. Thompson Ludovic Denoyer, Alfredo de la Fuente. Salina: Sequential learning of agents. `https://gitHub.com/facebookresearch/salina`, 2021.

[27] Daniel Jaymin Mankowitz, Augustin Zídek, André Barreto, Dan Horgan, Matteo Hessel, John Quan, Junhyuk Oh, H. V. Hasselt, David Silver, and Tom Schaul. Unicorn: Continual learning with a universal, off-policy agent. *ArXiv*, abs/1802.08294, 2018.

[28] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *ArXiv*, abs/2010.04495, 2021.

[29] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks : the official journal of the International Neural Network Society*, 113:54–71, 2019.

[30] Ramakanth Pasunuru and Mohit Bansal. Continual and multi-task architecture search. *ArXiv*, abs/1906.05226, 2019.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[32] Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Kumar Gupta. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. *ArXiv*, abs/2110.10067, 2021.

[33] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *ArXiv*, abs/1810.11910, 2019.

[34] Matthew Riemer, Tim Klinger, Djallel Bouneffouf, and Michele M. Franceschini. Scalable recollections for continual lifelong learning. In *AAAI*, 2019.

[35] Anthony V. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.*, 7:123–146, 1995.

[36] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. In *NeurIPS*, 2019.

[37] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *CoRR*, abs/1511.06295, 2016.

[38] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

[39] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in Psychology*, 4, 2013.

[40] Jonathan Schwarz, Wojciech M. Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *ArXiv*, abs/1805.06370, 2018.

[41] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, 2017.

[42] Daniel L. Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, 2013.

[43] Shagun Sodhani, Franziska Meier, Joelle Pineau, and Amy Zhang. Block contextual mdps for continual learning. *arXiv preprint arXiv:2110.06972*, 2021.

[44] Giacomo Spigler. Meta-learnt priors slow down catastrophic forgetting in neural networks. *ArXiv*, abs/1909.04170, 2019.

[45] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[46] Kalifou René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, Natalia Díaz Rodríguez, and David Filliat. Discorl: Continual reinforcement learning via policy distillation. *ArXiv*, abs/1907.05855, 2019.

[47] Mitchell Wortsman, Maxwell Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. *ArXiv*, abs/2102.10472, 2021.

[48] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *ArXiv*, abs/2006.14769, 2020.

[49] Maciej Wołczyk, Michal Zajkac, Razvan Pascanu, Lukasz Kuci'nski, and Piotr Milo's. Continual world: A robotic benchmark for continual reinforcement learning. *ArXiv*, abs/2105.10919, 2021.

[50] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *NeurIPS*, 2018.

[51] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

[52] Wei Zhou, Yiying Li, Yongxin Yang, Huaimin Wang, and Timothy M. Hospedales. Online meta-critic learning for off-policy actor-critic methods. *ArXiv*, abs/2003.05334, 2020.

A Algorithm Details

A.1 Scalability of CSP

By having access to an infinite number of policies, the subspace is highly expressive so it can capture a wide range of diverse behaviors. This enables positive transfer to many new tasks without the need for training additional parameters. As a consequence, the number of parameters scales *sublinearly* with the number of tasks. The speed of growth is controlled by the threshold ϵ which defines how much performance we are willing to give up for decreasing the number of parameters (by the size of one policy network). Practitioners can set the threshold to trade-off performance gains for memory efficiency (*i.e.* the higher the ϵ the more performance losses are tolerated to reduce memory costs). In practice, we noticed that setting $\epsilon = 0.1$ allows good performance and a limited growth of parameters.

As the agent learns to solve more and more tasks, we expect the subspace to grow more slowly (or stop growing entirely) since it already contains many useful behaviors which *transfer* to new tasks. On the other hand, if the agent encounters a task that is significantly different than previous ones and all other behaviors in the subspace, the subspace still has the *flexibility* to grow and incorporate entirely new skills. The number of anchors in the final subspace is adaptive and depends on the sequence of tasks. The longer and more diverse the task sequence, the more anchors are needed. This property is important for real-world applications with open-ended interactions where it's unlikely to know a priori how much capacity is required to express all useful skills.

A.2 Using the critic as a discriminator

The subspace critic W_ϕ plays a central role in our method. Compared to the vanilla SAC critic, we only add the convex combination α as an input (concatenated with the states and actions). In this way, it is optimized not only to evaluate the future averaged return on (s, a) pairs of a single policy, but an infinity of policies, characterized by the convex combination α.

At the end of each task, the subspace critic has the difficult task to estimate by how much the new anchor policy θ_{j+1} improves the performance of the current subspace Θ_j. To do so, one has to find the best combination of policies in the last subspace Θ_j, calling it α^{old} and the one in the current subspace Θ_{j+1}, calling it α^{new}. In practice, we found that sampling 1024 random (s, a) pairs from the replay buffer at the end of the task allows to have an accurate estimation of the best policies.

A.3 Sampling combinations in the subspace

Yet, it is important to allow the critic to estimate α^{old}. During training, we noticed that sampling with a simple flat Dirichlet distribution (i.e. a uniform distribution over the simplex induced by the current subspace) is not enough to make the critic able to accurately estimate the performance of the last subspace (indeed, the chances of sampling a policy in the last subspace are almost surely 0.). This is why we decided to sample in both the current and the last subspace. The distribution we use is then a mixture of two Dirichlet (equal chances of sampling in the last subspace and in the current subspace). We did not perform an ablation to see if balancing the mixture would increase performances.

We also tried to sample with a peaked distribution (concentration of the Dirichlet equal to the inverse of the number of the anchors) to see if it increased performances. In some cases the new subspace is able to find good policies faster with this distribution. It can be a good trade off between always choosing the last anchor and sampling uniformly.

B Experimental Details

All the experiments were implemented with SaLinA [26], a flexible and simple python library for learning sequential agents. We used Soft Actor Critic [15] as the routine algorithm for each method. It is a popular and efficient off-policy algorithm for continuous domains. We use a version where the entropy parameter is learned, as proposed in [16]. For each task and each methods, the twin critic networks are reset as well as the replay buffer (which has a maximum size of 1M interactions, corresponding to the maximum budget of each task).

B.1 Architecture Details

Brax scenarios:
For the twin critics and policy, We use a network with 4 linear layers, each consisting of 256 neurons. We use leaky ReLU activations (with $\alpha = 0.2$) after every layer.

Continual World triplets:
We use the same architectures as for Brax scenarios, but following [49], we add Layer Normalization [3] after the first layer, followed by a tanh activation. For the subspace method, we did not add this layer.

Specific architecture of CSP:
Our Pytorch implementation of CSP uses the `nn.ModuleList` object to store anchor networks. The additional computational cost compared to a single network is negligible during both training and inference as it is mentioned in [47].

B.2 Experimental Protocol

Brax scenarios:
Considering FT-N as the upper bound with which we want to compare, we decided to apply the following protocol for each short scenario. First, we run a gridsearch on SAC hyper-parameters (see Table 3) on FT-N and select the best set in terms of final average performance (see Section 4 for the details about this metric). Then, we freeze these hyper-parameters and performed a specific gridsearch for CSP and each baseline (see Table 4). Each hyper-parameter set is evaluated over 3 seeds. We believe this ensures fair comparisons, and even gives a slight advantage to FT-N compared to our method. Note that we set the number of parallel environments to 128 and the number of update per step to 0.5. They can be seen as tasks constraints. These values are reasonable, and FineTuning them would have increased the number of experiments by a lot.

Continual World triplets:
We bypassed the preliminary SAC gridsearch and use the set of hyper-parameters proposed in [49]. We performed a specific gridsearch for CSP and each baseline (see Table 4).

B.3 Baselines

EWC:
This regularization based method aims to protect parameters that are important for the previous tasks. After each task, it uses the Fisher information matrix to approximate the importance of each weight. We followed [49] to compute the Fisher matrix and use an analytical derivation of it.

FT-L2:
This baseline is proposed by [49]. It can be seen as a simplified version of EWC where the regularization coefficients for each parameters are equal to 1.

PNN:
This method proposed by [38] creates a new network at the beginning of each task, as well as lateral networks that will take as inputs - for each hidden layer - the output of the networks trained on former tasks. In this method, the number of parameters, training and inference times are growing exponentially with respect to the number of tasks. We used a simple linear layer for each lateral connection.

B.4 Ablations

Ablations on the threshold parameter (see Figure 2) of CSP have been performed on the long version of the Distraction scenario of HalfCheetah. Concerning the scalability ablation, we used the Compositional scenario of HalfCheetah and duplicated it in 3 versions (4 tasks, 8 tasks, 12 tasks). Concerning the learning efficiency, we used the Distraction Scenario (short version) and ran it with a budget of 250k,500k and 1M interactions. Results were averaged over 3 seeds.

Table 3: Hyper-parameters search for SAC over Brax scenarios. Red text indicates that the hyper-parameter is seen as a constraint of the environment, as explained in B.2.

Hyper-parameter	**Values tested**
lr policy	$\{0.0003, 0.001\}$
lr critic	$\{0.0003, 0.001\}$
reward scaling	$\{1., 10.\}$
target output std	$\{0.05, 0.1\}$
policy update delay	$\{2, 4\}$
target update delay	$\{2, 4\}$
lr entropy	0.0003
update target network coeff	0.005
batch size	256
n parallel environments	128
gradient update per step	0.5
discount factor	0.99
replay buffer size	$1M$
warming steps (random uniform policy)	$12,800$

Table 4: Specific hyper-parameter search for regularization based baselines and our model.

Hyper-parameter	**Value**
FT-L2, EWC	
regularization coefficient	$\{10^{-2}, 10^{0}, 10^{2}, 10^{4}, 10^{6}\}$
CSP	
threshold	$\{0.1, 0.25\}$
combination rollout length	$\{20, 100\}$
distribution type	{flat,peaked}

C Environment Details

C.1 Designing the Tasks

We used the flexibility of Brax physics engine [12], and two of its continuous control environments **Halfcheetah** (obs dim: 23, action dim: 6) and **Ant** (obs dim: 27, action dim: 8) to derive multiple tasks from them. To do so, we tweaked multiple environment parameters (Table 5)) and tried to learn a policy on these new tasks. From that pool of trials, we kept tasks (see Table 6) that were both challenging and diversified.

Table 5: Environment parameters tweaked to create interesting tasks for our scenarios. The Range column indicates the range of the multiplying factor applied to the environment parameter in question.

Environment Parameter	Range	Description
mass	$[0.5, 1.5]$	mass of a particular part of the agent's body (torso, legs, feet,...)
radius	$[0.5, 1.5]$	radius of a particular part of the agent's body (torso, legs, feet,...)
gravity	$[0.15, 1.5]$	gravity of the environment
friction	$[0.4, 1.5]$	friction of the environment
actions	$\{1, -1\}$	invert action values if set to -1. Used for distraction tasks.
observations (mask)	$[0.1, 0.8]$	proportion of masked observations to simulate defective sensors
actions (mask)	$[0.25, 0.75]$	proportion of masked actions to simulate defective modules

Table 6: List of the 21 tasks used to create our scenarios and the parameter changes associated.

	Task Name	Parameter changes
HalfCheetah	normal	`{}`
	carrystuff	`{torso_mass: 4, torso_radius: 4}`
	carrystuff_hugegravity	`{torso_mass: 4, torso_radius: 4, gravity: 1.5}`
	defectivesensor	`{masked_obs: 0.5}`
	hugefeet	`{feet_mass: 1.5, feet_radius: 1.5}`
	hugefeet_rainfall	`{feet_mass: 1.5, feet_radius: 1.5, friction: 0.4}`
	inverted_actions	`{action_coefficient: -1.}`
	moon	`{gravity: 0.15}`
	tinyfeet	`{feet_mass: 0.5, feet_radius: 0.5}`
	tinyfeet_moon	`{feet_mass: 0.5, feet_radius: 0.5, gravity: 0.15}`
	rainfall	`{friction: 0.4}`
Ant	normal	`{}`
	disabled3feet_1	`{action_mask: 0.75 (only 1st leg available)}`
	disabled3feet_2	`{action_mask: 0.75 (only 2nd leg available)}`
	disableddiagonalfeet_1	`{action_mask: 0.5 (1st diagonal)}`
	disableddiagonalfeet_2	`{action_mask: 0.5 (2nd diagonal)}`
	disabledforefeet	`{action_mask: 0.5 (forefeet)}`
	disabledhindfeet	`{action_mask: 0.5 (hindfeet)}`
	inverted_actions	`{action_coefficient: -1.}`
	moon	`{gravity: 0.7}`
	rainfall	`{friction: 0.4}`

C.2 Designing the Scenarios

Inspired by [49], we studied the relationship between these changes with a simple protocol: we learn a new task with a policy that has been pre-trained on a former task. We drew forgetting and transfer tables for each pair of tasks (see Figure 4). With this information, we designed 2 types of scenarios representing a particular challenge in continual learning: the **Forgetting Scenarios** are designed such that a single policy tends to forget the former task when learning a new one. The **Transfer Scenarios** are designed such that a single policy has more difficulties to learn a new task after having learned the former one, rather than learning it from scratch. In addition we designed **Distraction Scenarios**, that alternate between a normal task and a very different distraction task that disturbs the whole learning process of a single policy (we simply inverted the actions). While this challenge looks particularly simple from a human perspective (a simple -1 vector applied on the output is fine to find an optimal policy in a continual setting), we figured out that the Fine-tuning policies struggle to recover good performances (the final average reward actually decreases). Finally, we created **Compositional Scenarios**, that present two first tasks that will be useful to learn the last one, but a very different distraction task is put at the third place to disturb this forward transfer. In order to give a complete overview of the performances of our model and baselines, we designed a **short version** (4 tasks) and a **long version** (8 tasks, i.e. the short version repeated twice) of all the scenarios. Here is the detailed sequence for each scenario (short version):

1. **Forgetting Scenarios:**

- HalfCheetah:
  `hugefeet → moon → carrystuff → rainfall`

- Ant:
  `normal → hugefeet → rainfall → moon`

2. **Transfer Scenarios:**

- HalfCheetah:
  `carrystuff_hugegravity → moon → defectivesensor → hugefeet_rainfall`

- Ant:
  `disableddiagonalfeet_1 → disableddiagonalfeet_2 → disabledforefeet → disabledhindfeet`

3. **Distraction Scenarios:**

- HalfCheetah:
  `normal → inverted_actions → normal → inverted_actions`

- Ant:
  `normal → inverted_actions → normal → inverted_actions`

4. **Compositional Scenarios:**

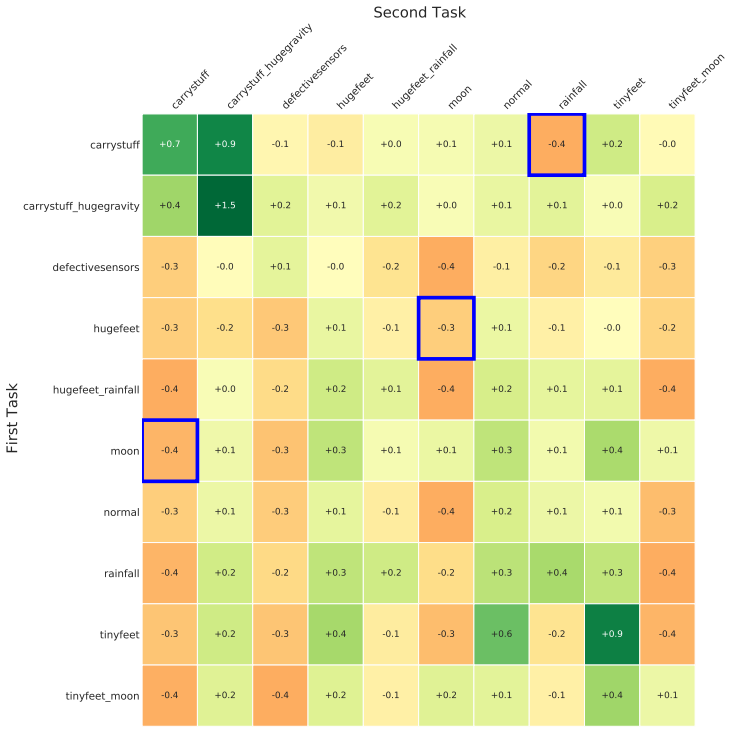
- HalfCheetah:
  `tinyfeet → moon → carrystuff_hugegravity → tinyfeet_moon`

- Ant:
  `disabled3feet_1 → disabled3feet_2 → disabledforefeet → disabledhindfeet`

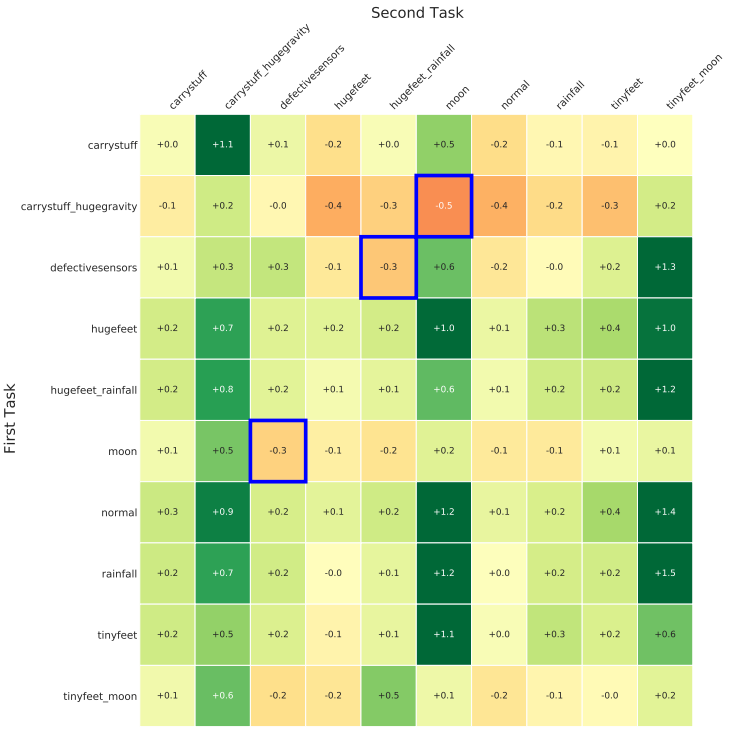
C.3 Continual World Tasks

We also tested our model and our baselines on the 8 triplets (3-tasks scenarios) proposed by [49]. The tasks originally come from Metaworld [51], and have a budget of 1M interactions each. The scenarios were specially designed such that there is a positive forward transfer from task 1 to task 3, but task 2 is used as a distraction that interferes with these learning dynamics. These tasks are designed with the 2.0 version of the open-source MuJoCo physics engine [45]. Here is the detailed sequence of the triplets:

1. `push-v1 → window-close-v1 → hammer-v1`
2. `hammer-v1 → window-close-v1 → faucet-close-v1`
3. `window-close-v1 → handle-press-side-v1 → peg-unplug-side-v1`
4. `faucet-close-v1 → shelf-place-v1 → peg-unplug-side-v1`
5. `faucet-close-v1 → shelf-place-v1 → push-back-v1`
6. `stick-pull-v1 → peg-unplug-side-v1 → stick-pull-v1`
7. `stick-pull-v1 → push-back-v1 → push-wall-v1`
8. `push-wall-v1 → shelf-place-v1 → push-back-v1`



(a) Forgetting table



(b) Transfer table

Figure 4: Forgetting (a) and Transfer (b) tables of our HalfCheetah tasks. The pairs selected to create our scenario are highlighted in blue. Results are averaged over 3 seeds using a classical RL algorithm.

D Detailed Results

D.1 HalfCheetah

For each of the 4 scenarios, we ran two versions: A short one (4 tasks) and a long one (8 tasks, the short one repeated twice). The normalized results are respectively presented in Table 9 and Table 10. For a complete transparency, we also provide in Table 7 the raw rewards that were used to normalize the results. See Section B for more details about the computation of the normalized results. The best hyper-parameter sets for each scenario are provided in Table 8

Table 7: Raw cumulative rewards obtained on each task of the 4 scenarios. These are obtained with a single policy learned from scratch (SAC-N) and the set of hyperparameters selected by the gridsearch of the FT-N method (see B). This explains why similar tasks have different averaged returns across scenarios. Results are averaged over 3 seeds.

	Task	**Cumulative rewards**
Forgetting Scenario	hugefoot	2209
	moon	2982
	carrystuff	6309
	rainfall	1001
Transfer Scenario	carrystuff_hugegravity	7233
	moon	3599
	defectivemodule	5909
	hugefoot_rainfall	2942
Distraction Scenario	normal	4932
	inverted_actions	5833
	normal	4932
	inverted_actions	5833
Compositional Scenario	tinyfoot	6311
	moon	3932
	carrystuff_hugegravity	6319
	tinyfoot_moon	1355

Table 8: Hyper-parameters selected for each HalfCheetah scenario. They have been selected on short versions. For the long versions, we ran the same set of selected hyperparameters.

Hyper-parameter	**Scenario**			
	Forgetting	Transfer	Distraction	Compositional
lr policy	0.001	0.0003	0.001	0.0003
lr critic	0.0003	0.0003	0.001	0.0003
reward scaling	1.	1.	1.	10.
target output std	0.1	0.05	0.1	0.1
policy update delay	2	2	4	4
target update delay	2	2	2	4
FT-L2				
regularization coefficient	10^4	10^0	10^2	10^2
EWC				
regularization coefficient	10^{-2}	10^0	10^{-2}	10^0
CSP				
threshold	0.1	0.1	0.1	0.1
combination rollout length	100	20	20	100
distribution type	peaked	peaked	flat	flat

D.2 Ant

We only ran the short version of the 4 scenarios. The long version is currently running and will be displayed in a future version of this paper. The normalized results are respectively presented in Table 13. For a complete transparency, we also provide in Table 11 the raw rewards that were used to normalize the results. See Section B for more details about the computation of the normalized results. The best hyper-parameter sets for each scenario are provided in Table 12

Table 9: Detailed results of the short version (4 tasks) of our 4 Halfcheetah scenarios. Results presents mean and standard deviation of our 4 metrics, are split by scenario and method, and are averaged over 3 seeds.

	Method	Performance	Transfer	Forgetting	Growing Factor
Forgetting scenario	FT-1	0.99 ± 0.15	0.57 ± 0.4	-0.58 ± 0.27	**1.0**
	FT-L2	0.88 ± 0.26	-0.13 ± 0.27	0.0 ± 0.0	2.0
	EWC	1.05 ± 0.09	0.08 ± 0.15	-0.03 ± 0.06	3.0
	PNN	1.05 ± 0.45	0.05 ± 0.45	0.0 ± 0.0	12.4
	SAC-N	1.0 ± 0.35	0.0 ± 0.35	0.0 ± 0.0	4.0
	FT-N	**1.89 ± 0.2**	**0.89 ± 0.21**	0.0 ± 0.0	4.0
	CSP-MID	0.38 ± 0.4	0.26 ± 0.37	-0.88 ± 0.03	3.0 ± 0.0
	CSP-ORACLE	2.03 ± 0.01	1.0 ± 0.02	0.03 ± 0.03	3.0 ± 0.0
	CSP (ours)	1.76 ± 0.08	0.76 ± 0.08	**0.0 ± 0.0**	3.0 ± 0.0
Transfer scenario	FT-1	0.45 ± 0.03	-0.07 ± 0.08	-0.47 ± 0.05	**1.0**
	FT-L2	0.35 ± 0.09	-0.49 ± 0.18	-0.16 ± 0.09	2.0
	EWC	0.21 ± 0.01	-0.79 ± 0.0	0.0 ± 0.0	3.0
	PNN	0.84 ± 0.13	-0.17 ± 0.13	0.0 ± 0.0	12.4
	SAC-N	1.0 ± 0.02	0.0 ± 0.02	0.0 ± 0.0	4.0
	FT-N	0.94 ± 0.05	-0.06 ± 0.05	0.0 ± 0.0	4.0
	CSP-MID	0.52 ± 0.08	-0.19 ± 0.07	-0.29 ± 0.11	3.33 ± 0.47
	CSP-ORACLE	1.14 ± 0.07	0.14 ± 0.04	0.0 ± 0.03	3.33 ± 0.47
	CSP (ours)	**0.98 ± 0.06**	**-0.03 ± 0.06**	**0.0 ± 0.0**	3.33 ± 0.47
Distraction scenario	FT-1	0.36 ± 0.06	-0.18 ± 0.06	-0.45 ± 0.12	**1.0**
	FT-L2	0.53 ± 0.05	-0.47 ± 0.05	0.0 ± 0.0	2.0
	EWC	0.78 ± 0.41	-0.2 ± 0.39	-0.02 ± 0.08	3.0
	PNN	1.09 ± 0.03	0.1 ± 0.03	0.0 ± 0.0	12.4
	SAC-N	1.0 ± 0.18	0.0 ± 0.18	0.0 ± 0.0	4.0
	FT-N	1.11 ± 0.03	0.11 ± 0.03	0.0 ± 0.0	4.0
	CSP-MID	0.43 ± 0.12	-0.07 ± 0.1	-0.5 ± 0.18	3.0 ± 0.82
	CSP-ORACLE	1.29 ± 0.08	0.28 ± 0.07	0.01 ± 0.0	3.0 ± 0.82
	CSP (ours)	**1.13 ± 0.09**	**0.13 ± 0.09**	**0.0 ± 0.0**	3.0 ± 0.82
Compositional scenario	FT-1	1.19 ± 0.09	0.33 ± 0.04	-0.15 ± 0.05	**1.0**
	FT-L2	1.2 ± 0.13	0.28 ± 0.08	-0.08 ± 0.05	2.0
	EWC	1.0 ± 0.17	0.07 ± 0.16	-0.07 ± 0.03	3.0
	PNN	1.0 ± 0.28	0.0 ± 0.29	0.0 ± 0.0	12.4
	SAC-N	1.0 ± 0.2	0.0 ± 0.2	0.0 ± 0.0	4.0
	FT-N	**1.5 ± 0.18**	**0.5 ± 0.17**	0.0 ± 0.0	4.0
	CSP-MID	.72 ± 0.12	-0.04 ± 0.22	-0.24 ± 0.11	3.33 ± 0.47
	CSP-ORACLE	1.58 ± 0.05	0.59 ± 0.05	-0.01 ± 0.0	3.33 ± 0.47
	CSP (ours)	1.39 ± 0.1	0.39 ± 0.1	**0.0 ± 0.0**	3.33 ± 0.47
Aggregate	FT-1	0.75 ± 0.08	0.16 ± 0.15	-0.41 ± 0.12	**1.0**
	FT-L2	0.74 ± 0.13	-0.2 ± 0.15	-0.06 ± 0.04	2.0
	EWC	0.76 ± 0.17	-0.21 ± 0.18	-0.03 ± 0.04	3.0
	PNN	1 ± 0.22	-0.01 ± 0.23	0 ± 0	12.4
	SAC-N	1 ± 0.19	0 ± 0.19	0 ± 0	4.0
	FT-N	**1.36 ± 0.12**	**0.36 ± 0.12**	0 ± 0	4.0
	CSP-MID	0.51 ± 0.18	-0.01 ± 0.19	-0.48 ± 0.11	3.2 ± 0.4
	CSP-ORACLE	1.51 ± 0.05	0.5 ± 0.05	0.01 ± 0.02	3.2 ± 0.4
	CSP (ours)	1.32 ± 0.08	0.31 ± 0.08	**0 ± 0.0**	3.2 ± 0.4

D.3 Continual World

We provide the final average performance of the methods over the 8 triplets proposed in [49]. See Table 14.

Table 10: Detailed results of the long version (8 tasks) of our 4 Halfcheetah scenarios. Results presents mean and standard deviation of our 4 metrics, are split by scenario and method, and are averaged over 3 seeds.

	Method	Performance	Transfer	Forgetting	Growing Factor
Forgetting scenario	FT-1	0.95 ± 0.37	0.79 ± 0.34	-0.84 ± 0.08	**1.0**
	FT-L2	1.07 ± 0.12	0.53 ± 0.25	-0.46 ± 0.15	2.0
	EWC	1.37 ± 0.33	0.77 ± 0.21	-0.4 ± 0.12	3.0
	PNN	1.35 ± 0.45	0.35 ± 0.45	0.0 ± 0.0	47.3
	FT-N	1.68 ± 0.15	0.68 ± 0.15	0.0 ± 0.0	8.0
	CSP-MID	0.55 ± 0.4	-0.1 ± 0.45	-0.36 ± 0.15	4.33 ± 1.7
	CSP-ORACLE	1.98 ± 0.04	0.87 ± 0.13	0.11 ± 0.14	4.33 ± 1.7
	CSP (ours)	**1.69 ± 0.12**	**0.69 ± 0.12**	**0. ± 0.0**	4.33 ± 1.7
Transfer scenario	FT-1	0.42 ± 0.07	-0.16 ± 0.07	-0.42 ± 0.07	**1.0**
	FT-L2	0.37 ± 0.07	-0.22 ± 0.04	-0.42 ± 0.09	2.0
	EWC	0.52 ± 0.08	-0.36 ± 0.05	-0.11 ± 0.05	3.0
	PNN	0.87 ± 0.08	-0.13 ± 0.08	0.0 ± 0.0	47.3
	FT-N	0.89 ± 0.12	-0.11 ± 0.12	0.0 ± 0.0	8.0
	CSP-MID	0.62 ± 0.17	-0.29 ± 0.13	-0.09 ± 0.11	4.67 ± 0.47
	CSP-ORACLE	1.2 ± 0.06	0.15 ± 0.09	0.05 ± 0.03	4.67 ± 0.47
	CSP (ours)	**0.97 ± 0.08**	**-0.04 ± 0.08**	**0.0 ± 0.0**	4.67 ± 0.47
Distraction scenario	FT-1	0.36 ± 0.18	-0.24 ± 0.13	-0.39 ± 0.09	**1.0**
	FT-L2	0.61 ± 0.08	-0.38 ± 0.06	-0.02 ± 0.02	2.0
	EWC	0.66 ± 0.05	-0.3 ± 0.03	-0.04 ± 0.04	3.0
	PNN	0.97 ± 0.09	-0.03 ± 0.09	0.0 ± 0.0	47.3
	FT-N	0.84 ± 0.03	-0.16 ± 0.03	0.0 ± 0.0	8.0
	CSP-MID	0.54 ± 0.03	-0.31 ± 0.04	-0.14 ± 0.04	2.0 ± 0.0
	CSP-ORACLE	1.27 ± 0.03	0.26 ± 0.04	0.01 ± 0.0	2.0 ± 0.0
	CSP (ours)	**1.18 ± 0.03**	**0.18 ± 0.03**	**0.0 ± 0.0**	2.0 ± 0.0
Compositional scenario	FT-1	1.28 ± 0.03	0.42 ± 0.02	-0.13 ± 0.04	1.0
	FT-L2	1.19 ± 0.09	0.41 ± 0.21	-0.22 ± 0.13	2.0
	EWC	1.38 ± 0.11	0.43 ± 0.15	-0.05 ± 0.13	3.0
	PNN	1.04 ± 0.07	0.04 ± 0.08	0.0 ± 0.0	47.3
	FT-N	**1.48 ± 0.08**	**0.48 ± 0.08**	0.0 ± 0.0	8.0
	CSP-MID	0.84 ± 0.09	-0.16 ± 0.06	0.0 ± 0.04	5.0 ± 0.82
	CSP-ORACLE	1.68 ± 0.22	0.64 ± 0.12	0.04 ± 0.1	5.0 ± 0.82
	CSP (ours)	1.42 ± 0.03	0.42 ± 0.03	**0.0 ± 0.0**	5.0 ± 0.82
Aggregate	FT-1	0.75 ± 0.16	0.2 ± 0.14	-0.45 ± 0.07	1.0
	FT-L2	0.81 ± 0.09	0.09 ± 0.14	-0.28 ± 0.1	2.0
	EWC	1.22 ± 0.1	0.22 ± 0.1	0 ± 0	8.0
	PNN	0.98 ± 0.14	0.14 ± 0.11	-0.15 ± 0.09	3.0
	FT-N	1.06 ± 0.17	0.06 ± 0.18	0 ± 0	47.3
	CSP-MID	0.64 ± 0.17	-0.22 ± 0.17	-0.15 ± 0.09	4 ± 0.7
	CSP-ORACLE	1.53 ± 0.09	0.48 ± 0.1	0.05 ± 0.07	4 ± 0.7
	CSP (ours)	**1.32 ± 0.07**	**0.31 ± 0.07**	**0 ± 0.0**	4 ± 0.7

E Analysis of the Subspace

The he best way to visualize the reward and critic value landscapes of the subspaces is when there are 3 anchors (see Figure 3). To do saw, we draw 8192 evenly spaced points in the 3-dimensional simplex of $\mathbb{R}^3$, and average the return over 10 rollouts for the reward landscape, and 1024 pairs of (s, a) for the critic landscape. We used the short version of the Compositional Scenario of HalfCheetah to display the results.

Table 11: Raw cumulative rewards obtained on each task of the 4 scenarios. These are obtained with a single policy learned from scratch (SAC-N) and the set of hyperparameters selected by the gridsearch of the FT-N method (see B). This explains why similar tasks have different averaged returns across scenarios. Results are averaged over 3 seeds.

	Task	**Cumulative rewards**
Forgetting Scenario	normal	3752
	hugefeet	2841
	rainfall	1596
	moon	1401
Transfer Scenario	disableddiagonalfeet_1	3021
	disableddiagonalfeet_2	4119
	disabledforefeet	1014
	disabledhindfeet	1021
Distraction Scenario	normal	3542
	inverted_actions	4199
	normal	3542
	inverted_actions	4199
Compositional Scenario	disabled3feet_1	770
	disabled3feet_2	641
	disabledforefeet	201
	disabledhindfeet	288

Table 12: Hyper-parameters selected for each Ant scenario. They have been selected on short versions. For the long versions, we ran the same set of selected hyperparameters.

Hyper-parameter	**Scenario**			
	Forgetting	Transfer	Distraction	Compositional
lr policy	0.001	0.001	0.001	0.0003
lr critic	0.001	0.001	0.001	0.0003
reward scaling	10.	1.	1.	10.
target output std	0.05	0.05	0.1	0.1
policy update delay	2	2	2	4
target update delay	4	2	4	4
FT-L2				
regularization coefficient	10^4	10^0	10^0	10^2
EWC				
regularization coefficient	10^{-2}	10^4	10^2	10^{-2}
CSP				
threshold	0.1	0.1	0.1	0.1
combination rollout length	100	100	100	20
distribution type	peaked	peaked	peaked	peaked

F Limitations of CSP

CSP prevents forgetting of prior tasks, promotes transfer to new tasks, and scales sublinearly with the number of tasks. Despite all these advantages, our method still has a number of limitations. While the subspace grows only sublinearly with the number of tasks, this number is highly dependent on the task sequence. In the worst case scenario, it increases linearly with the number of tasks. On the other hand, the more similar the tasks are, the lower the size of the subspace needed to learn good policies for all tasks. Thus, one important direction for future work is to learn a subspace of policies with a fixed number of anchors. Instead of training an additional anchor for each new task, one could optimize a policy in the current subspace on the new task while ensuring that the best policies for prior tasks don't change too much. This could be formulated as a constrained optimization problem where all the anchors defining the subspace are updated for each new task, but some regions of the

Table 13: Detailed results of the short version (4 tasks) of our 4 Ant scenarios. Results presents mean and standard deviation of our 4 metrics, are split by scenario and method, and are averaged over 3 seeds.

	Method	Performance	Transfer	Forgetting	Growing Factor
Forgetting scenario	FT-1	1.36 ± 0.29	0.42 ± 0.26	-0.06 ± 0.04	**1.0**
	FT-L2	1.73 ± 0.34	0.61 ± 0.3	0.12 ± 0.14	2.0
	EWC	1.83 ± 0.16	0.81 ± 0.09	0.02 ± 0.07	3.0
	PNN	0.77 ± 0.09	-0.23 ± 0.09	0.0 ± 0.0	12.3
	SAC-N	1.0 ± 0.32	0.0 ± 0.32	0.0 ± 0.0	1.0
	FT-N	**1.97 ± 0.11**	**0.97 ± 0.11**	0.0 ± 0.0	4.0
	CSP-MID	1.09 ± 0.11	0.21 ± 0.16	-0.11 ± 0.27	3.38 ± 0.48
	CSP-ORACLE	2.01 ± 0.07	0.96 ± 0.04	0.05 ± 0.03	3.38 ± 0.48
	CSP (ours)	1.78 ± 0.06	0.78 ± 0.06	**0.0 ± 0.0**	3.38 ± 0.48
Transfer scenario	FT-1	-0.12 ± 0.29	-0.65 ± 0.7	-0.47 ± 0.41	**1.0**
	FT-L2	0.85 ± 0.23	-0.06 ± 0.29	-0.09 ± 0.07	2.0
	EWC	0.53 ± 0.02	-0.47 ± 0.02	0.0 ± 0.0	3.0
	PNN	0.98 ± 0.02	-0.02 ± 0.01	0.0 ± 0.0	12.3
	SAC-N	1.0 ± 0.51	0.0 ± 0.51	0.0 ± 0.0	1.0
	FT-N	0.83 ± 0.47	-0.17 ± 0.47	0.0 ± 0.0	4.0
	CSP-MID	0.2 ± 0.04	-0.23 ± 0.11	-0.57 ± 0.08	3.67 ± 0.47
	CSP-ORACLE	1.51 ± 0.25	0.49 ± 0.25	0.02 ± 0.03	3.67 ± 0.47
	CSP (ours)	**1.41 ± 0.27**	**0.41 ± 0.27**	**0.0± 0.0**	3.67 ± 0.47
Distraction scenario	FT-1	0.26 ± 0.09	-0.19 ± 0.16	-0.56 ± 0.12	**1.0**
	FT-L2	0.46 ± 0.08	-0.46 ± 0.03	-0.08 ± 0.08	2.0
	EWC	0.42 ± 0.05	-0.5 ± 0.03	-0.07 ± 0.08	3.0
	PNN	0.72 ± 0.16	-0.29 ± 0.16	0.0 ± 0.0	12.3
	SAC-N	1.00 ± 0.05	0.0 ± 0.05	0.0 ± 0.0	4.0
	FT-N	0.89 ± 0.11	-0.11 ± 0.11	0.0 ± 0.0	4.0
	CSP-MID	0.21 ± 0.16	-0.46 ± 0.07	-0.32 ± 0.09	3.0 ± 0.0
	CSP-ORACLE	1.04 ± 0.04	-0.01 ± 0.03	0.05 ± 0.01	3.0 ± 0.0
	CSP (ours)	**0.94 ± 0.04**	**-0.06 ± 0.04**	**0.0± 0.0**	3.0 ± 0.0
Compositional scenario	FT-1	-0.84 ± 0.11	-1.06 ± 0.25	-0.78 ± 0.31	**1.0**
	FT-L2	0.12 ± 0.31	-0.8 ± 0.17	-0.08 ± 0.13	2.0
	EWC	0.26 ± 0.51	0.06 ± 0.73	-0.81 ± 1.0	3.0
	PNN	0.11 ± 0.35	-0.89 ± 0.35	0.0 ± 0.0	12.3
	SAC-N	1.00 ± 0.32	0.0 ± 0.32	0.0 ± 0.0	4.0
	FT-N	2.52 ± 0.03	1.52 ± 0.03	0.0 ± 0.0	4.0
	CSP-MID	2.29 ± 0.21	1.28 ± 0.26	0.0 ± 0.21	3.33 ± 0.47
	CSP-ORACLE	3.36 ± 0.71	2.3 ± 0.65	0.06 ± 0.06	3.33 ± 0.47
	CSP (ours)	**3.06 ± 0.77**	**2.06 ± 0.77**	**0.0 ± 0.0**	3.33 ± 0.47
Aggregate	FT-1	0.17 ± 0.2	-0.37 ± 0.34	-0.47 ± 0.22	**1.0**
	FT-L2	0.79 ± 0.24	-0.18 ± 0.2	-0.03 ± 0.11	2.0
	EWC	0.76 ± 0.19	-0.03 ± 0.22	-0.22 ± 0.29	3.0
	PNN	0.65 ± 0.16	-0.36 ± 0.15	0.0 ± 0.0	12.3
	SAC-N	1.0 ± 0.30	0.00 ± 0.30	0.0 ± 0.0	2.5
	FT-N	1.55 ± 0.18	0.55 ± 0.18	0.0 ± 0.0	4.0
	CSP-MID	0.95 ± 0.13	0.2 ± 0.15	-0.25 ± 0.16	3.3 ± 0.4
	CSP-ORACLE	1.98 ± 0.27	0.94 ± 0.24	0.05 ± 0.03	3.3 ± 0.4
	CSP (ours)	**1.8 ± 0.29**	**0.8 ± 0.29**	**0.0 ± 0.0**	3.3 ± 0.4

subspace are regularized to not change very much. This would result in the subspace having different regions which are good for different tasks.

While the memory costs increase sublinearly with the number of tasks, the computational costs increase linearly with the number of tasks, in the current implementation of CSP. This is because we train an additional anchor for each new task, which can be removed if it doesn't significantly improve

Table 14: Detailed results of 8 triplets scenarios from Continual World. Results presents mean and standard deviation of the final average performance, are split by scenario and method, and are averaged over 3 seeds.

Method	T1	T2	T3	T4	T5	T6	T7	T8	**Agregate**
FT-1	0.24 ± 0.13	0.25 ± 0.07	0.39 ± 0.16	0.34 ± 0.05	0.30 ± 0.01	0.32 ± 0.25	0.17 ± 0.07	0.34 ± 0.05	0.29 ± 0.1
EWC	0.45 ± 0.12	0.27 ± 0.09	0.38 ± 0.09	0.31 ± 0.12	0.32 ± 0.07	0.33 ± 0.18	0.20 ± 0.10	0.32 ± 0.08	0.32 ± 0.11
PNN	**0.84 ± 0.08**	0.72 ± 0.17	0.90 ± 0.05	0.43 ± 0.08	0.33 ± 0.23	0.46 ± 0.21	0.44 ± 0.12	0.36 ± 0.2	0.56 ± 0.14
SAC-N	0.69 ± 0.17	0.71 ± 0.13	0.79 ± 0.19	0.47 ± 0.14	**0.60 ± 0.13**	0.55 ± 0.11	0.54 ± 0.15	0.45 ± 0.12	0.6 ± 0.14
FT-N	0.77 ± 0.08	**0.86 ± 0.1**	0.78 ± 0.15	0.49 ± 0.14	0.52 ± 0.13	**0.61 ± 0.06**	**0.61 ± 0.13**	0.52 ± 0.06	0.65 ± 0.11
CSP (ours)	0.76 ± 0.2	0.79 ± 0.03	**0.82 ± 0.08**	**0.58 ± 0.09**	0.58 ± 0.06	0.54 ± 0.06	0.58 ± 0.04	**0.53 ± 0.08**	**0.65 ± 0.08**

performance. However, the computational costs can also be reduced if you have access to maximum reward on a task. This is typically the case for sparse reward tasks where if the agent succeeds, it receives a reward of 1 and 0 otherwise. In this case, there is no need to train one anchor per task. Instead, one can simply find the best policy in the current subspace and compare its performance with the maximum reward to decide whether to train an additional anchor or not. Hence, in this version of CSP (which is a minor modification of the current implementation) both memory and compute scale sublinearly with the number of tasks. However, this assumption doesn't always hold, so here we decided to implement the more general version of CSP.

In this work, we don't specifically leverage the structure of the subspace in order to find good policies. Hence, one promising research direction is to further improve transfer efficiency by leveraging the structure of the subspace to find good policies for new tasks. For example, this could be done by finding the convex combination of the anchors which maximizes return on a given task. Regularizing the geometry of the subspace to impose certain inductive biases could also be a fruitful direction for future work.

G Broader Impact Statement

This work proposes a new method for the continual reinforcement learning setting, which is a very broad framework. Many real-world applications can be framed as continual RL problems. For example, household robots need to constantly learn new skills and adapt to their changing environments. Similarly, online recommendation systems need to adapt to each user and their changing preferences. Like other continual RL algorithms, our method aims to learn a policy which maximizes reward on a sequence of tasks (each of them with a corresponding reward function). The tasks or reward functions are typically specified by the designer. Depending on the goals of the designer, executing the resulting policy could result in positive or negative consequences.

H Compute Resources

Each algorithm was trained using two Intel(R) Xeon(R) CPU cores (E5-2698 v4 @ 2.20GHz) and one NVIDIA GP100 GPU. We used PyTorch [31] for all our experiments. Each run took between approximately 8 and 30 hours to complete. The total runtime depended on three factors: the computation time of the algorithm, the length of the scenario, and the behavior of the policy.