Mitigating Plasticity Loss in Continual Reinforcement Learning by Reducing Churn

Hongyao Tang [1 2] **Johan Obando-Ceron** [1 2] **Pablo Samuel Castro** [1 2] **Aaron Courville** [1 2] **Glen Berseth** [1 2]

Abstract

Plasticity, or the ability of an agent to adapt to new tasks, environments, or distributions, is crucial for continual learning. In this paper, we study the loss of plasticity in deep continual RL from the lens of churn: network output variability for out-of-batch data induced by mini-batch training. We demonstrate that (1) the loss of plasticity is accompanied by the exacerbation of churn due to the gradual rank decrease of the Neural Tangent Kernel (NTK) matrix; (2) reducing churn helps prevent rank collapse and adjusts the step size of regular RL gradients adaptively. Moreover, we introduce Continual Churn Approximated Reduction (C-CHAIN) and demonstrate it improves learning performance and outperforms baselines in a diverse range of continual learning environments on OpenAI Gym Control, ProcGen, DeepMind Control Suite, and MinAtar benchmarks.

1. Introduction

Reinforcement learning (RL), when coupled with non-linear function approximators, suffers from optimization challenges due to the non-stationarity of the data and the learning objectives (Sutton & Barto, 1988; van Hasselt et al., 2018). This difficulty is amplified when there is a sequence of changing tasks, as in continual RL (Abel et al., 2023).

One of the causes for this is what is known as *loss of plasticity* (Berariu et al., 2021; Lyle et al., 2022; Dohare et al., 2024), whereby an agent gradually loses its ability to adapt to new data or objective function. It is hypothesized that this is due to the agent's network tendency to overfit early experience, hampering its ability to learn on later experience (Nikishin et al., 2022a). Addressing this pathology is challenging, and there have been a number of recently proposed remedies, such as resetting dormant neurons (Sokar et al., 2023), regularization (Kumar et al., 2023b), and variants of backpropagation (Dohare et al., 2024). It has also been argued that rank decrease of the empirical Neural Tangent Kernel (NTK) (Achiam et al., 2019) is a consistent indicator of plasticity loss (Lyle et al., 2024; Lewandowski et al., 2023). However, the true causal factors and underlying mechanism remain largely unknown.

In this work, we study the loss of plasticity from the angle of *churn* (Schaul et al., 2022; Tang & Berseth, 2024): network output variability for out-of-batch data induced by mini-batch training. While it has been shown that churn is endemic to most deep RL networks, affecting both generalization (Bengio et al., 2020) and interference (Liu et al., 2023), we demonstrate that plasticity loss is highly correlated with increased churn. We use the NTK matrix as a formal tool to establish the connection between churn and plasticity loss, based on which we analyze the learning dynamics of continual learning across a sequence of tasks. We demonstrate that under the continual changes in the data distribution and objective function, the agent gradually loses the rank information of its NTK matrix, leading to highly correlated gradients and eventually the exacerbation of churn. Consequently, this destroys the stability and convergence of the learning dynamics, which is related to the pathological learning behavior of plasticity loss.

Driven by the connection between plasticity and churn, we extend the idea of churn reduction originally proposed for single-MDP RL (Tang & Berseth, 2024) to continual RL. We introduce Continual Churn Approximated Reduction (C-CHAIN). C-CHAIN continually minimizes the churn for the data out of the training batch alongside the regular training of continual RL. Moreover, we formally demonstrate that C-CHAIN has a two-fold efficacy in mitigating plasticity loss with a gradient decorrelation effect by suppressing the off-diagonal entries of the NTK matrix, and a step-size adjustment effect by taking the gradient projection of the data out of the training batch.

To evaluate our approach under extreme non-stationarity, we conduct the experiments in various continual RL environments built on OpenAI Gym Control (Brockman et al.,

[1]Mila - Québec AI Institute [2]Université de Montréal. Correspondence to: Hongyao Tang <tang.hongyao@mila.quebec>, Glen Berseth <glen.berseth@mila.quebec>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

2016), ProcGen (Cobbe et al., 2020), DeepMind Control Suite (Tassa et al., 2018), and MinAtar (Young & Tian, 2019) benchmarks, with a total of 24 continual RL environments. The results show that reducing churn effectively improves the agent's performance in continual RL, and outperforms related methods in most environments.

Our main contributions can be summarized as:

- We demonstrate the connection between plasticity loss and increased churn, and show the pathological learning dynamics this connection induces.
- We unbox the efficacy of reducing churn in continual RL by identifying a gradient decorrelation effect and a step-size adjustment effect.
- We propose C-CHAIN[1] and demonstrate it effectively mitigates the loss of plasticity and outperforms prior methods in a range of continual RL settings.

2. Related Work

Nonstationarity in Reinforcement Learning Nonstationarity naturally occurs in various contexts, with continual RL being among the most prominent examples. Nonstationarity can manifest as gradual changes. This may happen when an RL agent continuously improves its policy over time (Zhang et al., 2024) or when the data generation process slowly evolves (Ellis et al., 2024). These different forms of non-stationarity can coexist. For example, in continual reinforcement learning (Xie et al., 2021), both the reward function and the transition dynamics may change over time, requiring learning systems to adapt dynamically.

Due to the use of a non-linear function approximator under non-stationarity, the model weights and optimization may not lead to an optimal policy. There are two main challenges in optimizing under distribution shift, catastrophic forgetting (also called *backward transfer*) (Rolnick et al., 2019; Chaudhry et al., 2019; Nath et al., 2023) or the loss of plasticity (called *forward transfer*) (Berariu et al., 2021; Lyle et al., 2022; Dohare et al., 2024).

Continual Learning and the Loss of Plasticity Continual learning focuses on training models to learn sequential tasks. It is broadly divided into continual supervised learning, which aims to retain knowledge across tasks in static datasets (Wang et al., 2024), and continual RL, where agents adapt to dynamic environments while retaining prior experience (Khetarpal et al., 2022; Kumar et al., 2023a; Dohare et al., 2024; Elsayed & Mahmood, 2024). The loss of plasticity refers to a phenomenon in neural network training where the model gradually loses its ability to adapt or significantly update its parameters (Dohare et al., 2024). This typically occurs when the network settles into sharp local minima or becomes overly specialized to the initial stages of training, resulting in reduced learning capacity, slower convergence, or poor generalization to new data (Nikishin et al., 2022b; 2024).

The underlying mechanisms responsible for the loss of plasticity during training remain an area of active research (Ma et al., 2023a; Lyle et al., 2024). Researchers have proposed various strategies to mitigate the loss of plasticity; approaches include ensuring active unit engagement (Sokar et al., 2023), mitigating gradient starvation (Gogianu et al., 2021; Dohare et al., 2023), employing smaller batch sizes (Ceron et al., 2023), minimizing deviations from initial parameter values (Lewandowski et al., 2023; Kumar et al., 2023a), resetting optimizer (Asadi et al., 2023; Ellis et al., 2024), and regularizing weight orthogonality (Chung et al., 2024). Periodically resetting network parameters has been effective (Ash & Adams, 2020; Frati et al., 2024), especially effective in data-efficient reinforcement learning (Schwarzer et al., 2023; Nauman et al., 2024). Ceron et al. (2024a;b); Liu et al. (2025) showed that dynamic sparse training leads to improved performance and enhanced network plasticity.

3. Preliminaries

3.1. Reinforcement Learning

Consider a Markov Decision Process (MDP) defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \rho_0, T \rangle$, with the state set $\mathcal{S}$, the action set $\mathcal{A}$, the transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the discounted factor $\gamma \in [0, 1)$, the initial state distribution ρ_0 and the horizon T. The agent interacts with the MDP by performing actions with its policy $a_t \sim \pi(s_t)$ that defines the mapping from states to action distributions. The objective of an RL agent is to optimize its policy to maximize the expected discounted cumulative reward $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_t]$, where $s_0 \sim \rho_0(s_0)$, $s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t)$ and $r_t = \mathcal{R}(s_t, a_t)$. The state-action value function q^π defines the expected cumulative discounted reward for all $s, a \in \mathcal{S} \times \mathcal{A}$ and the policy π, i.e., $q^\pi(s, a) = \mathbb{E}_\pi\big[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s, a_0 = a\big]$.

In deep RL, policy and value functions are approximated with deep neural networks, conventionally denoted by Q_θ and π_ϕ with network parameters θ and ϕ. The parameterized policy π_ϕ can be updated by taking the gradient of the objective, i.e., $\phi' \leftarrow \phi + \alpha \nabla_\phi J(\pi_\phi)$ with a step size α (Silver et al., 2014; Mnih et al., 2016; Schulman et al., 2017; Haarnoja et al., 2018).

3.2. Continual Learning and Plasticity

Beyond the standard MDP setting which describes a stationary decision-making task, we consider a continual learning scenario where the agent learns to solve tasks

[1] https://github.com/bluecontra/C-CHAIN

$\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$ that arrive in a sequence. In this work, each task is an MDP with a different reward or dynamics function and has a budget of N interactions for the agent. Aside from this, we do not use any assumption on the task sequence, e.g., the similarity between tasks. The non-stationarity manifests in the continual switch of learning tasks. We focus on plasticity in this work, and the objective of a continual RL agent in this context is to maximize the *average performance* over the course of continual learning on the task sequence.

Formally, for any point i within the total budget, we have the policy π_i where $i \in \{1, ..., k\}$, and the performance $J(\pi_i)$ corresponding to it. Note that $J(\pi_i)$ is the performance evaluated on the task $\mathcal{T}_{\lceil i \rceil}$, i.e., the current task at time step i. For convenience, we define the average performance regarding a task $\mathcal{T}_j$ as $J(\mathcal{T}_j) = \frac{1}{N}\sum_{i=(j-1)*N+1}^{j*N} J(\pi_i)$. Thus, the average performance of the agent throughout the learning on the sequence $\mathbb{T}$ is:

$$J(\mathbb{T}) = \frac{1}{k}\sum_{j=1}^{k} J(\mathcal{T}_j) = \frac{1}{kN}\sum_{i=1}^{kN} J(\pi_i). \tag{1}$$

Similar to the concept of Area Under Curve (AUC), the average performance metric aggregates the learning performance in terms of efficiency, stability and convergence. We use average performance to evaluate and compare different methods, and in practice, we take intervals within $\{1, ..., k\}$ for the estimation of $J(\mathbb{T})$. In the context of continual learning, the loss of plasticity means that the agent gradually exhibits worse learning performance on later tasks. Empirically, it can often be identified when we observe $\bar{J}(\mathcal{T}_i) - J(\mathcal{T}_i) \geq \bar{J}(\mathcal{T}_j) - J(\mathcal{T}_j)$ for $1 \leq i \leq j \leq k$, where $\bar{J}(\mathcal{T}_i)$ denotes the performance of an agent of good plasticity on the task $\mathcal{T}_i$, e.g., a newly initialized network for the current task.

4. Mitigating the Loss of Plasticity by Reducing Churn

In this section, we formally study the loss of plasticity in continual RL from the lens of churn. First, we establish a connection between plasticity loss and churn by taking the NTK matrix as a framework (Section 4.1), based on which we analyze the continual learning dynamics (Section 4.2). Further, we introduce Continual Churn Approximated Reduction (C-CHAIN), and formally dissect the efficacy of churn reduction on plasticity (Section 4.3).

4.1. NTK as the Bridge between Plasticity and Churn

The loss of plasticity is a phenomenon that stems from the pathological learning behavior of a deep network, while churn is an innate feature of the network where the network output for datapoints not included in the current training batch is implicitly and potentially uncontrollably changed.

To study the relationship between the two concepts, we use the empirical NTK (Achiam et al., 2019) matrix as a formal tool, as its rank has been shown to be a good indicator of plasticity loss (Lyle et al., 2024). For a parameterized network function $f_\theta(x) \in \mathbb{R}$, we use $g(x)$ to denote the gradient of f_θ with respect to its parameters θ at the data x, i.e., $g(x) = \nabla_\theta f_\theta(x) \in \mathbb{R}^d$ where $|\theta| = d$ is the dimensionality. The Neural Tangent Kernel matrix N_θ is the matrix of gradient dot products between all data points (Achiam et al., 2019; Lyle et al., 2024):

$$\begin{aligned} N_\theta(i,j) &= \nabla_\theta f_\theta(x_i)^\top \nabla_\theta f_\theta(x_j) \ \text{ for } x_i, x_j \\ N_\theta &= G_\theta^\top G_\theta \end{aligned} \tag{2}$$

where $G_\theta = [g(x_1), g(x_2), \ldots, g(x_i), \ldots]$ denotes the matrix form of the gradients for all datapoints. This is also referred to as the outer-product approximation of the Hessian matrix (Lewandowski et al., 2023). In practice, the empirical NTK matrix is often used by sampling a batch of datapoints for the convenience of computation and visualization. Lyle et al. (2024) show that networks tend to exhibit a low rank of the empirical NTK matrix N_θ when they lose plasticity. This is also observed when the full gradient is approximated by the penultimate layer network representation (Kumar et al., 2022; Ma et al., 2023b).

Now consider the parameter changes from f_θ to $f_{\theta'}$ and let $\Delta_\theta = \theta' - \theta$. For data $\bar{x} \in B_{\text{ref}}$ (the *reference* data) not included in training batch B_{train} used for the parameter update Δ_θ, the churn of f_θ at $\bar{x}$ with respect to Δ_θ is (Tang & Berseth, 2024):

$$\begin{aligned} C_f(\bar{x}, \theta, \Delta_\theta) &= f_{\theta'}(\bar{x}) - f_\theta(\bar{x}) \\ &= \nabla_\theta f_\theta(\bar{x})^\top \Delta_\theta + O(\|\Delta_\theta\|^2). \end{aligned} \tag{3}$$

The parameter update Δ_θ made by mini-batch training can have different forms, which depends on the context. Letting $L(\theta)$ be a loss function of f_θ, we can express the parameter update in a general form via the chain rule: $\Delta_\theta = -\eta \mathbb{E}_x[\nabla_\theta f_\theta(x) \nabla_{f_\theta} L(\theta, x)]$, where $x \in B_{\text{train}}$ is the data sampled for the mini-batch training. We use vanilla stochastic gradient descent and use η for the learning rate here for legibility. By plugging the general form back, we have the NTK expression of the churn:

$$\begin{aligned} C_f(\bar{x}, \theta, \Delta_\theta) &\approx -\eta \nabla_\theta f_\theta(\bar{x})^\top \mathbb{E}_x[\nabla_\theta f_\theta(x) \nabla_{f_\theta} L(\theta, x)] \\ &= -\eta \mathbb{E}_x[N_\theta(\bar{x}, x) \nabla_{f_\theta} L(\theta, x)]. \end{aligned} \tag{4}$$

Further, we can obtain the churn in vector form, which is equivalent to the approximate change of the function value for all datapoints:

$$\begin{aligned} C_f(\theta, \Delta_\theta) &\approx -\eta G_\theta^\top G_\theta S G_L \\ &= -\eta N_\theta S G_L, \end{aligned} \tag{5}$$

where G_L is the gradient matrix of $\nabla_{f_\theta} L(\theta)$ and S is a diagonal matrix of the same size as N_θ with $\{0, 1\}$-binary values

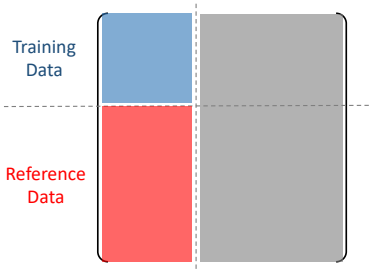


Figure 1. **An illustration of the matrix** $N_\theta S$, where the datapoints are arranged with a separation between the training data and the remaining ones (i.e., the reference data).

on its diagonal that corresponds to the sampling results of B_{train}. The vector form in Equation 5 indicates that the NTK matrix N_θ plays an important role in determining the churn, independent of the loss function (or objective function) and the sampling strategy (or data distribution). Intuitively, N_θ determines how the explicit mini-batch training shapes the entire function landscape by generalization or interference between different datapoints. Figure 1 shows an illustration of the matrix $N_\theta S$. For the symmetric NTK matrix N_θ, multiplying the sampling matrix S_i zero-masks out the columns corresponding to the reference data, denoted by the ■ area. The remaining entries consist of the training updates to the sampled training data (■) and the changes caused by churn to the reference data (■).

As the empirical NTK matrix plays important roles in both plasticity loss and churn, it serves as a natural bridge between the two. We further our analysis below.

4.2. Exacerbation of Churn Induced by NTK Collapse

Starting from a random parameter initialization, the network of a learning agent digests data from the sequential tasks $\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$ and updates it parameters to solve each task at hand the best. Notice that each task is an MDP and the data on the task is collected online by the agent within a budget of interaction steps.

To formally characterize the learning process, let $\mathcal{E}_i(\theta) = f_i^* - f_\theta$ be the error vector of the network f_θ regarding the optimal objective function f_i^* corresponding to task $\mathcal{T}_i$ with $i \in \{1, \dots, k\}$. We consider the commonly-used loss function $L_i(\theta, x) = \frac{1}{2}[f_i^*(x) - f_\theta(x)]^2$. The dynamics of the error $\mathcal{E}_i$ for iterative updates $\theta_t \to \theta_{t+1}$ within the same task $\mathcal{T}_i$ can be derived as:

$$\begin{aligned}\mathcal{E}_i(\theta_{t+1}) - \mathcal{E}_i(\theta_t) &= \big(f_i^* - f_{\theta_{t+1}}\big) - \big(f_i^* - f_{\theta_t}\big) \\ &= f_{\theta_t} - f_{\theta_{t+1}} \\ &= -C_f(\theta_t, \theta_{t+1} - \theta_t).\end{aligned} \tag{6}$$

By plugging the vector form of churn in Equation 5 and $G_L = -\big(f_i^* - f_{\theta_t}\big) = -\mathcal{E}_i(\theta_t)$, we further have:

$$\begin{aligned}\mathcal{E}_i(\theta_{t+1}) &\approx \mathcal{E}_i(\theta_t) - \eta N_{\theta_t} S_i \mathcal{E}_i(\theta_t) \\ &= \big(I - \eta N_{\theta_t} S_i\big)\mathcal{E}_i(\theta_t).\end{aligned} \tag{7}$$

Equation 7 characterizes the iterative evolution of the error regarding the task $\mathcal{T}_i$, which is mainly determined by the factor $I - \eta N_{\theta_t} S_i$. N_{θ_t} is full-rank when it has positive diagonal values (i.e., non-zero gradients $f_\theta(x_i) \neq \mathbf{0}$) and has zero off-diagonal entries, i.e., $\nabla_\theta f_\theta(x_i)^\top \nabla_\theta f_\theta(x_j) = 0$ for $i \neq j$. In this case, it resembles the tabular function approximation where generalization and interference among different datapoints do not exist. The learning process is stable when a proper η is selected. However, in most practical cases, the off-diagonal entries of N_{θ_t} are non-zero and N_{θ_t} is not full-rank, making the learning dynamics intricate.

Next, we continue to analyze the learning dynamics by taking into consideration the task change in $\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$. Different from a stationary learning scenario, both the data distribution (related to S_i) and the objective function (related to $f_i^*, \mathcal{E}_i$) change throughout learning. In the early stages of learning, the parameters of the agent's network are close to the initialization, thus having no plasticity issue. Meanwhile, the empirical objective function estimated with finite online data (related to $S_i, \mathcal{E}_i$)tends to lie in low-dimensional space. With stochastic gradient descent, the network fits the empirical objective function with an implicit preference for simple functions (Arpit et al., 2017; Nakkiran et al., 2019; Damian et al., 2021) in a low-dimensional parameter subspace (Schneider et al., 2024; Tang et al., 2024). In this process, the agent prefers the parameters that have high correlations between different datapoints (Kumar et al., 2022) in order to generalize the updates, and thus loses the rank information in N_θ.

When the task changes from $\mathcal{T}_i$ to $\mathcal{T}_{i+1}$, the agent continues to fit the new objective function from $\mathcal{E}_{i+1}(\theta)$ with a distribution shift of sampling data S_{i+1}. However, the function landscape regarding the new data distribution was implicitly shaped via churn in prior training. Compared to the parameters in the early stage, the gradients between different datapoints correlate more and the rank of $N_\theta S_{i+1}$ becomes lower. This echoes the findings in prior works (Lyle et al., 2024; Kumar et al., 2021; Shah et al., 2020). As the task changes along the sequence $\mathbb{T}$, the rank decrease of the NTK matrix and the exacerbation of churn operate in a vicious cycle in the learning dynamics (Equation 7). Therefore, it leads to even less stable learning and worse approximation results, which is the pathological learning behavior identified as the loss of plasticity.

Naturally, a method that prevents the rank decrease of $N_\theta S_i$ and the exacerbation of churn should mitigate the loss of plasticity. Therefore, we dissect the efficacy of reducing churn on continual learning in the next subsection.

4.3. Effects of Continual Churn Reduction

We extend the study on churn reduction and explore its potential in mitigating plasticity loss in continual RL. We

Algorithm 1 Deep Continual RL with Continual Churn Approximated Reduction (C-CHAIN).

Task sequence $\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$, interaction budget per task N, continual RL algorithm $\mathbb{A}$
Initialize agent's parameter θ and a buffer D
for task $i = 1, 2, 3, ..., k$ **do**
  for step $i = 1, 2, 3, ..., N$ **do**
    Interact with the task by performing agent's policy
    Update the buffer D according to $\mathbb{A}$
    Perform regular parameter update $\Delta_\theta = \mathbb{A}(\theta, B_{\text{train}})$ with the training batch $B_{\text{train}} \sim D$
    Regularize agent's parameter to reduce churn for the reference batch $B_{\text{ref}} \sim D$ ($B_{\text{ref}} \cap B_{\text{train}} = \emptyset$)
  end for
end for

introduce Continual Churn Approximated Reduction (C-CHAIN). As depicted by the pseudo-code in Algorithm 1, the agent interacts with the tasks that arrive in a sequence. Alongside the regular training of the continual RL algorithm, C-CHAIN continually minimizes the churn for the data out of the current training batch (i.e., the reference batch B_{ref}).

Formally, the churn reduction loss function of C-CHAIN is defined with the reference data $\bar{x} \in B_{\text{ref}}$ as follows:

$$L_f^{\text{cr}}(\theta) = \frac{1}{2}\mathbb{E}_{\bar{x}\in B_{\text{ref}}}[C_f(\bar{x}, \theta, \Delta_\theta)^2] \tag{8}$$

Next, we look into the gradient of the churn reduction loss to shed light on how it influences the continual learning dynamics we discussed in the previous subsection:

$$\begin{aligned} \nabla_\theta L_f^{\text{cr}}(\theta, \bar{x}) &= C_f(\bar{x}, \theta, \Delta_\theta)\nabla_\theta C_f(\bar{x}, \theta, \Delta_\theta) \\ \nabla_\theta C_f(\bar{x}, \theta, \Delta_\theta) &\approx -\eta\nabla_\theta\big(\mathbb{E}_x[N_\theta(\bar{x}, x)\nabla_{f_\theta} L(\theta, x)]\big) \end{aligned} \tag{9}$$

For clarity we use $g = \nabla_\theta f_\theta(x), \bar{g} = \nabla_\theta f_\theta(\bar{x})$, thus the kernel $N_\theta(\bar{x}, x) = \bar{g}^\top g$, and $\nabla_f L_f = \nabla_{f_\theta} L(\theta, x)$. We then have the derivative in the expectation below:

$$\nabla_\theta\big(\bar{g}^\top g \nabla_f L_f\big) = \underbrace{\nabla_\theta\big(\bar{g}^\top g\big)\nabla_f L_f}_{①} + \underbrace{\bar{g}^\top g \nabla_\theta\big(\nabla_f L_f\big)}_{②} \tag{10}$$

The ① term is the partial derivative of the gradients of f_θ. With H_θ be the Hessian matrix of f_θ, we have:

$$① = \big(H_\theta(\bar{x})g + H_\theta(x)\bar{g}\big)\nabla_f L_f \tag{11}$$

Equation 11 indicates the first efficacy of reducing churn. It takes the gradient of the kernel $N_\theta(\bar{x}, x)$ regarding each pair of the reference data $\bar{x}$ and the training data x, with $\nabla_f L_f$ (as well as $C_f(\bar{x}, \theta, \Delta_\theta)$ playing the role of a weighting factor. As a result, the off-diagonal entries in the NTK matrix N_θ (corresponding to the ■ area of Figure 1) are suppressed to zero for minimization.

The ② term depends on the specific form of L_f regarding which the regular training (i.e., Δ_θ) is performed. Here we use the Temporal Difference (TD) learning of Q-network for L_f as a common example in deep RL for demonstration, where $L_Q(\theta) = \frac{1}{2}\mathbb{E}_{x,x'}[\big((r + \gamma Q^-(x')) - Q_\theta(x)\big)^2]$. Note that we use x for s, a and Q^- is the target Q-function detached from backpropagation. In this case, we have $\nabla_{Q_\theta} L_Q(\theta, x) = -Q_\theta(x) + (r + \gamma Q^-(x'))$ and thus:

$$② = -(\bar{g}^\top g)g = -\textbf{proj}_g \bar{g}\ \|g\|_2^2 \tag{12}$$

Equation 12 shows that the second efficacy of reducing churn can be interpreted as the projection of $\bar{g}$ on g with the square norm of g as a scaling factor. This term could either dampen or accelerate the regular gradient $g\nabla_f L_f$ which depends on the sign of the kernel between the reference data and the training data, and also the $\nabla_f L_f$.

The two terms work together when reducing churn during the continual learning process. The ① term decorrelates the gradients between the reference data and the training data by suppressing the off-diagonal entries in the NTK matrix N_θ. This combats the rank decrease and mitigates the loss of plasticity under the changes of the data distribution and the objective function as discussed in Section 4.2. Besides, the ② term regulates the regular training gradient based on additional gradient information of the reference data. More regularization occurs when the gradients correlates more.

To summarize, reducing churn improves the stability of the continual learning dynamics in Equation 7 and mitigates the loss of plasticity. In the next section, we conduct the empirical study for our formal findings discussed above.

5. Experiments

In our experiments, we first evaluate C-CHAIN in comparison with recent related methods, to find out whether it improves continual RL (Section 5.1). Then, we conduct the empirical analysis to examine whether reducing churn prevents the rank decrease and how the two effects contribute differently (Section 5.2). Finally, we extend the evaluation to more continual learning settings (Section 5.3).

5.1. Performance Evaluation

To evaluate the performance of continual RL, we adopt OpenAI Gym Control (Brockman et al., 2016) and ProcGen (Cobbe et al., 2020) and follow the setups in TRAC (Muppidi et al., 2024).

Setups For Gym Control, we use four environments: CartPole-v1, Acrobot-v1, LunarLander-v2 and MountainCar-v0. For each environment, a task sequence $\mathbb{T}$ is built by chaining k instances of the environment with a unique Gaussian observation noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ sampled once for each. The number k is 10 for all the environments, except that k is 5 for MountainCar-v0. For ProcGen, we use all sixteen environments in the suit, while only four

Table 1. **Performance comparison on continual Gym Control.** The reported scores are the average performance (Eq. 1) in terms of episode return with mean and standard error over six seeds. The top-2 values (excluding Oracle) are marked by **bold** and underline.

Env	Calibration Baselines		Related Methods			Ours
	Oracle	Vanilla	TRAC	Weight Clipping	L2 Init	C-CHAIN
C-Acrobot	-125.736 ± 8.303	-372.725 ± 3.268	-166.584 ± 8.922	**−118.821 ± 13.440**	-134.038 ± 14.292	-119.211 ± 6.929
C-CartPole	299.925 ± 5.986	61.766 ± 6.927	217.376 ± 26.307	154.306 ± 16.047	**266.114 ± 5.345**	160.396 ± 14.643
C-LunarLander	1.118 ± 2.002	-499.396 ± 19.571	-58.059 ± 17.474	-58.575 ± 2.896	**−6.666 ± 5.903**	-16.659 ± 8.369
C-MountainCar	-322.608 ± 9.384	-346.604 ± 4.635	-399.989 ± 0.010	-267.900 ± 6.883	-369.395 ± 27.914	**−245.746 ± 14.691**
Agg. Score	-147.302	-1156.958	-407.256	-290.99	-243.985	**−221.22**

Table 2. **Performance comparison on continual ProcGen.** The reported scores are the average performance (Eq. 1) in terms of episode return with mean and standard error over six seeds. The top-2 values (excluding Oracle) on each row are marked by **bold** and underline.

Env	Calibration Baselines		Related Methods						Ours
	Oracle	Vanilla	TRAC	Weight Clip	L2 Init	LayerNorm	ReDo	AdamRel	C-CHAIN
Starpilot	13.142 ± 0.262	8.028 ± 0.667	18.106 ± 0.716	8.200 ± 0.729	11.353 ± 0.207	14.085±0.652	9.824±0.221	11.863±0.910	**19.717 ± 0.699**
Fruitbot	1.613 ± 0.105	1.549 ± 0.139	0.855 ± 0.327	1.457 ± 0.090	1.657 ± 0.107	0.921±0.254	1.586±0.227	1.487±0.149	**1.689 ± 0.188**
Chaser	2.923 ± 0.016	2.714 ± 0.028	3.096 ± 0.021	2.812 ± 0.039	3.067 ± 0.023	2.960±0.034	2.894±0.032	2.720±0.053	**3.432 ± 0.033**
Dodgeball	5.017 ± 0.415	3.636 ± 0.488	6.757 ± 0.617	3.066 ± 0.600	4.859 ± 0.459	5.895±0.787	3.799±0.324	3.780±0.643	**7.690 ± 0.840**
Bigfish	4.206 ± 0.343	2.363 ± 0.632	5.783 ± 0.729	2.561 ± 0.334	3.752 ± 0.267	4.406±0.434	3.010±0.199	2.746±0.343	**7.866 ± 0.319**
Caveflyer	5.889 ± 0.285	5.605 ± 0.282	6.760 ± 0.400	5.157 ± 0.362	5.922 ± 0.307	6.103±0.228	5.595±0.330	5.232±0.491	**7.653 ± 0.587**
Climber	5.404 ± 0.659	4.557 ± 0.692	2.557 ± 0.500	5.003 ± 0.740	5.415 ± 0.662	5.349±0.658	5.372±0.634	5.207±0.677	**5.594 ± 0.579**
Ninja	2.169 ± 0.311	1.679 ± 0.705	1.278 ± 0.556	1.314 ± 0.601	**3.219 ± 0.364**	2.760±0.572	2.184±0.434	1.297±0.597	3.146 ± 0.436
Coinrun	4.953 ± 0.458	4.711 ± 0.676	3.943 ± 1.414	4.985 ± 0.757	4.857 ± 0.362	5.459±1.063	4.689±0.376	4.377±0.503	**6.829 ± 1.093**
Miner	5.147 ± 0.235	4.250 ± 0.314	4.288 ± 0.807	5.182 ± 0.634	6.153 ± 0.228	5.491±0.542	4.853±0.397	5.500±0.645	**9.924 ± 0.636**
Jumper	2.175 ± 0.330	1.095 ± 0.211	1.753 ± 0.456	1.536 ± 0.474	2.598 ± 0.299	2.666±0.471	1.269±0.274	2.225±0.509	**2.896 ± 0.371**
Heist	2.426 ± 0.295	2.404 ± 0.290	2.424 ± 0.294	2.506 ± 0.295	2.929 ± 0.343	2.402±0.292	2.691±0.368	2.668±0.372	**3.950 ± 0.681**
Leaper	0.350 ± 0.302	0.347 ± 0.302	0.334 ± 0.304	0.474 ± 0.305	0.557 ± 0.287	**0.809 ± 0.443**	0.467±0.303	0.511±0.314	0.668 ± 0.608
Maze	5.528 ± 0.525	5.324 ± 0.621	4.916 ± 0.774	5.314 ± 0.885	5.701 ± 0.999	5.084±0.550	4.747±0.616	5.299±0.867	**5.940 ± 0.424**
Plunder	7.113 ± 0.413	4.801 ± 0.430	9.719 ± 0.776	5.290 ± 0.389	6.695 ± 0.383	7.262±0.363	5.910±0.331	5.296±0.298	**10.340 ± 0.287**
Bossfight	2.735 ± 0.369	1.986 ± 0.667	4.722 ± 1.027	2.235 ± 0.630	3.791 ± 0.436	3.503±0.320	2.290±0.352	1.541±0.506	**4.920 ± 0.45**
Agg. Score	70.789	55.049	77.289	57.092	72.961	75.154	61.180	61.443	**101.792**

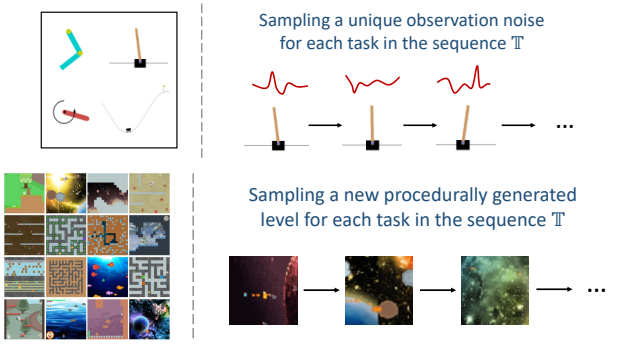


Figure 2. **The continual learning settings for Gym Control and ProcGen.** The task switches for continual CartPole *(above)* and Starpilot *(below)* are illustrated for demonstration.

(i.e., Starpilot, Fruitbot, Chaser, Dodgeball) were adopted in (Muppidi et al., 2024). For each environment, the task sequence consists of 5 instances procedurally generated by sampling a unique game level. The budget is 2M steps per task. Figure 2 illustrates the continual RL setups.

Baselines To calibrate the performance of continual RL, we make use of a standard Proximal Policy Optimization (PPO) (Schulman et al., 2017) agent as the *vanilla* baseline. The agent is initialized once before the continual learning starts. On the other side, we use a PPO agent that gets fully re-initialized every time the task is switched to be the *oracle* baseline. The oracle baseline learns each task from random initialization, thus free of the influence of task change. Additionally, we consider six related methods including TRAC (Muppidi et al., 2024), Weight Clipping (Elsayed et al., 2024b) and L2 Init (also called Regenerative Regularization) (Kumar et al., 2023b), Recycles Dormant Neurons (ReDo) (Sokar et al., 2023), Layer Normalization (Lyle et al., 2024), Adam with Relative Timesteps (AdamRel) (Ellis et al., 2024) as our baselines. Note that TRAC outperforms other related methods like Concatenated ReLU (Abbas et al., 2023), EWC (Schwarz et al., 2018), Modulating Masks (Nath et al., 2023) in the same settings adopted in our experiments. Therefore, we do not include them here. Besides, we found that L2 regularization performed very similarly (slightly worse) to L2 Init in our experiments, thus we use L2 Init for representation as they are very similar when the network parameterization is close to zero.

Implementation We use the official implementation of TRAC as the code base, based on which we implement all the baselines above as well as C-CHAIN (i.e., Algorithm 1 with the vanilla PPO agent as $\mathbb{A}$) to ensure that they only differ in the corresponding algorithm features. One thing to note is that C-CHAIN PPO, both B_{train} and B_{ref} are sampled from the online interaction data collected by the policy in the current iteration. Therefore, CHAIN PPO does not use

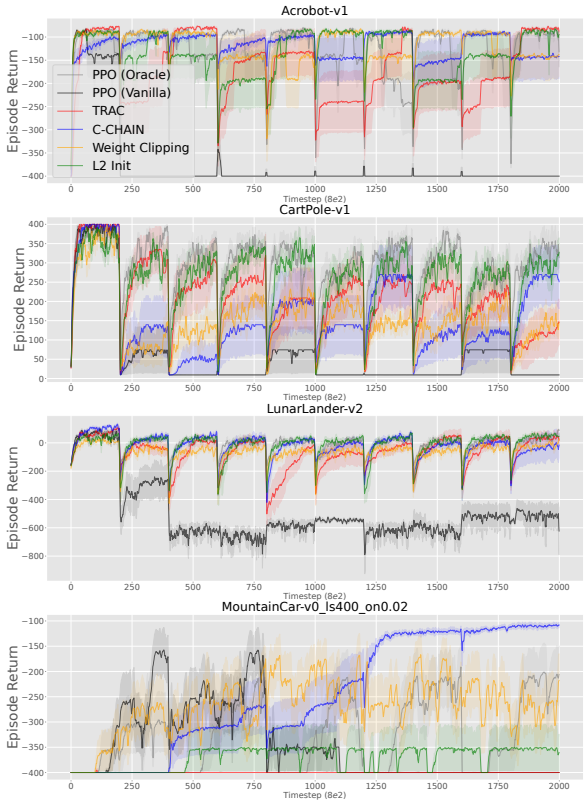


Figure 3. **Learning curves of different methods in four continual Gym Control environments.** The curves and the shades are means and standard errors over six seeds.

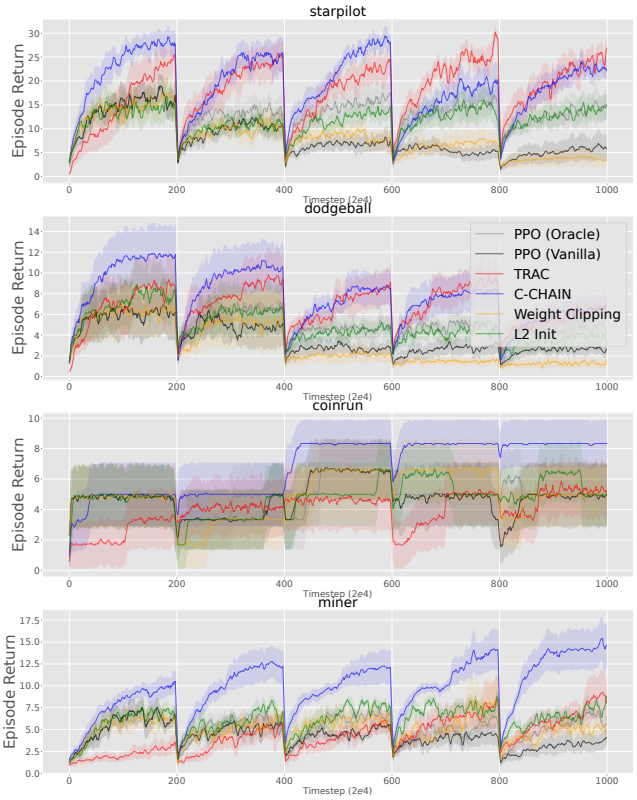


Figure 4. **Learning curves of different methods in four (of sixteen) continual ProcGen environments.** The curves and the shades are means and standard errors over six seeds.

a cross-task buffer and does not need to be aware of task switches. For the hyperparameters of the PPO baseline agent, we use the default values and keep them consistent across all the methods. For the hyperparameters specific to each method, we search around the recommendation values in the original papers and report the best. More experiment details are provided in Appendix A.

Quantitative Comparison Table 1,2 summarizes the quantitative evaluation for different methods on continual Gym Control and ProcGen, respectively. Figure 3 and Figure 4 show the learning curves. For clarity, we plot the learning curves for three related baseline methods for ProcGen. As task switch happens frequently, the learning curves of all methods show a phasic pattern. For the calibration baselines, i.e., Vanilla and Oracle, Vanilla quickly degrades and collapses (especially in Gym Control) after learning the first task in the sequence, while Oracle learns each task as it re-initializes when task switch happens. Overall, C-CHAIN and the other baseline methods effectively prevent the collapse of Vanilla and improve the learning performance greatly. One may note that TRAC, L2 Init and C-CHAIN outperform Oracle in ProcGen. This is because, aside from task switch, non-stationarity also exists in the learning process of single-MDP RL. Oracle still suffers from plasticity loss due to the change of sampling and policy learning within each task, while C-CHAIN also mitigates it effectively. This also indicates that naive network resetting is not adequate to address plasticity loss in continual RL.

Among the methods in comparison, TRAC is a competitive baseline in most environments except for the failure in environments like MountainCar and Coinrun. Weight Clipping works well in Gym Control but performs almost on par with Vanilla in ProcGen. Similarly, L2 Init performs the best in LunarLander and CartPole while it falls below C-CHAIN by a clear margin. A possible explanation is that both Weight Clipping and L2 Init mitigate plasticity loss by constraining the feasible parameter space near the initialization. Therefore, for problems where the agent needs a complex solution that is not easy to be found near the initialization, Weight Clipping and L2 Init can limit the learning in this sense. Our method C-CHAIN achieves the best aggregation scores in both continual Gym Control and ProcGen, demonstrating its superiority in improving continual RL.

Moreover, for continual MountainCar that needs more exploration and is a bit more difficult than the other three Gym control tasks, we notice that TRAC almost totally fail on it. The low early-stage performance of C-CHAIN might also be due to the exploration feature of MountainCar, as reducing churn could also slow down the generalization of exploration behavior learned by the policy. Therefore, C-

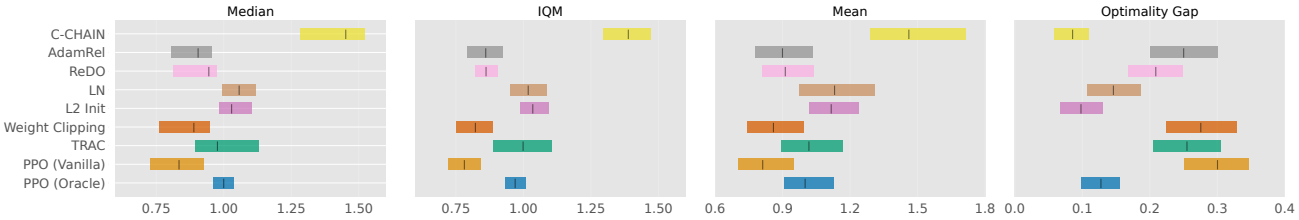


Figure 5. **Performance comparison with Reliable metrics (Agarwal et al., 2021).** For each method in the comparison, the results are aggregated over sixteen continual ProcGen environments with six random seeds for each.

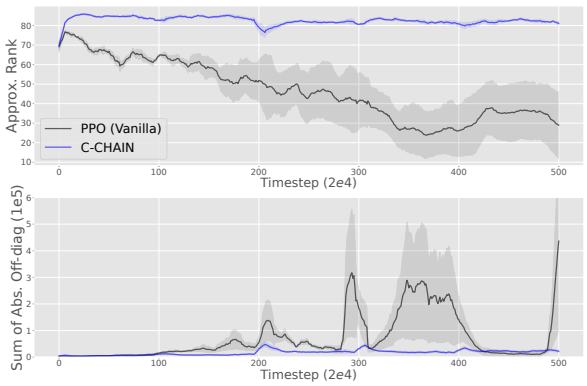


Figure 6. **Analysis on NTK matrix** in terms of approximate rank *(above)* and the sum of absolute off-diagonal values *(below)* in continual ProcGen Starpilot.

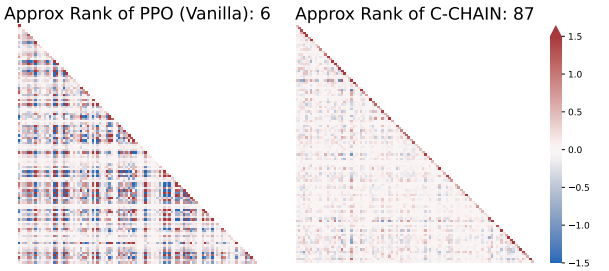


Figure 7. **Visualization of the empirical NTK matrix (100 by 100)** for Vanilla *(left)* and C-CHAIN *(right)* at 8M timesteps (10M in total) in continual ProcGen Starpilot.

CHAIN learns slowly but improves steadily, while vanilla PPO learns quicker and collapses. Continual ProcGen environments like Leaper, Jumper have sparse or even episodic rewards. To gain more understanding for these environments, we looked into the average scores of the Max, Mean, Min curves and observed that C-CHAIN improves the average Max scores over PPO Vanilla and the average Mean scores (either by improving Max or by reducing failure numbers); while it does not fully avoid the (near-)zero Min scores due to the limited exploration ability of PPO base.

Reliable Metrics To obtain a conclusive evaluation, we provide the aggregation evaluation by using Reliable metrics (Agarwal et al., 2021). The evaluation uses Mean, Median, Inter-Quantile Mean (IQM) and Optimality Gap, with 95% confidence intervals. It aggregates over 9 methods, 6 seeds for each of 16 ProcGen tasks as reported in Table 2, i.e., 864 runs in total.

The results are shown in Figure 5. We can observe that C-CHAIN performs the best regarding all four metrics and outperforms the second-best with no overlap of Confidence Intervals for Median, IQM, and a minor one for Mean.

5.2. Empirical Analysis

To better understanding how C-CHAIN improves continual RL and examine whether the empirical efficacy of C-CHAIN matches our formal analysis. We conduct empirical analysis on (1) the rank evolvement of the empirical NTK matrix (corresponding to Section 4.2) and (2) the ablation of the two effects of reducing churn (i.e., ①, ② in Section 4.3).

On the Evolvement of the Empirical NTK Matrix We empirically compute and record the NTK matrix of the agent's policy network at every iteration for the analysis. The details are provided in Appendix B. Figure 6 shows the approximate rank (Kumar et al., 2021) of the empirical NTK matrix and the summation of absolute off-diagonal values. For the vanilla PPO agent, the curve of approximate rank exhibits large instability, overall decreasing to a low rank throughout learning. This corresponds to the sharp increase of the scale of off-diagonal values. As expected, C-CHAIN suppresses the off-diagonal values of the NTK matrix and maintains a stable and high rank. This explains the performance of C-CHAIN in Figure 4 and Table 1,2, and empirically confirms our formal analysis in Section 4.

Moreover, as shown by the visualization in Figure 7, we can observe that the vanilla PPO agent exhibits a highly correlated NTK with off-diagonal cells of large absolute values (i.e., blue or red). The existence of high correlation leads to low rank, i.e., the sign of plasticity loss, which also echoes the empirical findings in (Lyle et al., 2024). In contrast, C-CHAIN shows a less correlated NTK and thus maintains the plasticity of the agent.

On the Ablation of the Two Effects of C-CHAIN We empirically dissect the gradient (denoted by $\bar{g}^{\text{cr}}$) of C-CHAIN's churn reduction loss function (Equation 8) into a projective component (i.e., $\mathbf{proj}_{g^{\text{PPO}}}\bar{g}^{\text{cr}}$ and g^{PPO} for the gradient of the PPO base agent) and an orthogonal component (i.e., $g^{\text{cr}} - \mathbf{proj}_{g^{\text{PPO}}}\bar{g}^{\text{cr}}$). We use the projective component to

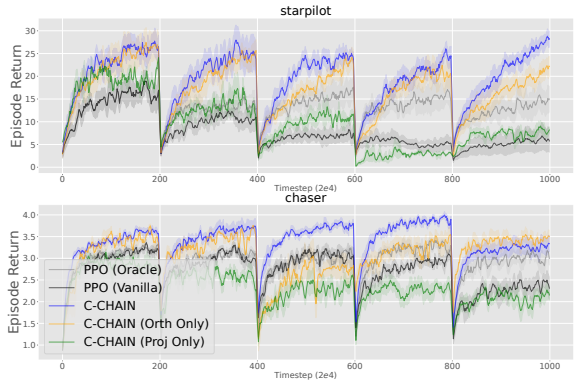


Figure 8. **Ablation study for the projective and orthogonal gradient components** of C-CHAIN in ProcGen Starpilot and Chaser.

approximate the ② term and use the orthogonal component for the ① term. By applying either component only, we have two variants of C-CHAIN, denoted by *Proj Only* and *Orth Only*. We then evaluate the contribution of the two components by comparing with C-CHAIN and Vanilla.

As in Figure 8, we can observe that both the orthogonal and the projective components are beneficial to the vanilla baseline. The two components contribute collectively and add up to the effectiveness of C-CHAIN. Moreover, the orthogonal component contributes more than the projective component. This indicates that suppressing the off-diagonal entries of the NTK (i.e., the ① term) is critical to C-CHAIN. This naturally delineates the difference between the efficacy of C-CHAIN and the related methods that alters RL gradient with projective information in different ways (Chaudhry et al., 2019; Yu et al., 2020).

5.3. C-CHAIN in More Continual Learning Settings

Continual DeepMind Control Suite (DMC) In addition to the discrete-action settings above, we evaluate C-CHAIN in continual continuous control. We build three continual RL environments based on DMC (Tassa et al., 2018): *Continual Walker* (Stand-Walk-Run), *Continual Quadruped* (Walk-Run-Walk), *Continual Dog* (Stand-Walk-Run-Trot), each of which is a continual RL scenario that chains the corresponding individual tasks in a sequence.

Table 3. **Performance comparison in continual DMC.** The reported scores are means and standard errors over twelve seeds.

Env	Oracle	Vanilla	C-CHAIN
C-Walker	395.971 ± 8.116	305.199 ± 18.519	$\mathbf{472.828 \pm 17.865}$
C-Quadruped	234.529 ± 19.193	250.153 ± 26.385	$\mathbf{314.510 \pm 44.321}$
C-Dog	137.080 ± 3.133	129.744 ± 5.914	$\mathbf{174.098 \pm 9.169}$

The results are shown in Table 3. We can observe that C-CHAIN PPO significantly outperforms PPO in all three settings, showing the effectiveness for continual control. C-CHAIN also outperforms PPO Oracle because it does not reset for each task and thus can *transfer* the learned skill from previous tasks to future ones (e.g., Dog-Stand to Dog-Walk). Details are provided in Appendix A.2.

Continual MinAtar To evaluate the generality across different base agents, we implement C-CHAIN DoubleDQN (van Hasselt et al., 2016) and evaluate it in a continual MinAtar (Young & Tian, 2019) setting built by us. Specifically, we chain three tasks: SpaceInvaders, Asterix, Seaquest. The results in Table 4 show that C-CHAIN also improves the continual learning performance upon DoubleDQN in a sequence of totally *different* tasks. Details are provided in Appendix A.3.

Table 4. **Performance comparison in continual MinAtar.** The reported scores are means and standard errors over twelve seeds.

Env	Vanilla	C-CHAIN
C-MinAtar	22.044 ± 0.733	$\mathbf{29.513 \pm 0.682}$

Continual Supervised Learning (SL) on MNIST We extend our empirical evaluation of C-CHAIN to continual supervised learning setting. We follow the settings in L2 Init (Kumar et al., 2023b) and adopt RandomLabel-MNIST and Permuted-MNIST as our testbed. The results and implementation details can be found in Table 8 and Appendix A.4. The results show that C-CHAIN improves the vanilla agent but does not perform on par with L2 Init and Weight Clipping. The efficacy of C-CHAIN is relatively limited in the two continual SL settings, in contrast to its superiority in the continual RL experiments. We hypothesize this is because RL suffers more from churn accumulation due to the chain effect (Tang & Berseth, 2024), which acts as a prominent cause of plasticity loss in continual RL, thus C-CHAIN can address it effectively. Besides, the two MNIST settings are simple, where a good solution can be found near the initialization as preferred by L2 Init and Weight Clipping. A more thorough study on this point is expected in the future with more continual supervised learning settings.

6. Conclusion

In this paper, we study plasticity loss from the lens of churn. We established a formal connection between the two with NTK and demonstrated the interplay between the rank decrease of NTK and the exacerbation of churn in learning dynamics. We show that our method C-CHAIN effectively improves continual RL results across a wide range of tasks.

Limitation and Future work Beyond theoretical advancements, applying churn reduction to real-world continual learning settings, as robotics or large language models (LLMs), could address plasticity loss and catastrophic forgetting in adaptive learning settings (Wu et al., 2024; Zhai et al., 2024). Investigating its role in reinforcement learning from human feedback (RLHF) could improve stability in policy updates, ensuring LLMs retain learned behaviors while adapting to evolving human preferences. Besides, Equation 11 reveals an interesting direction to realizing churn reduction in a more direct and effective way by estimating Hessian (Elsayed et al., 2024a).

Acknowledgements

We want to acknowledge funding support from NSERC, FQRNT, Google, CIFAR AI and compute support from Digital Research Alliance of Canada, Mila IDT, and NVidia.

We thank the anonymous reviewers for their valuable help in improving our manuscript. We would also like to thank the Python community (Van Rossum & Drake Jr, 1995; Oliphant, 2007) for developing tools that enabled this work, including NumPy (Harris et al., 2020), Matplotlib (Hunter, 2007), Jupyter (Kluyver et al., 2016) and Pandas (McKinney, 2013).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbas, Z., Zhao, R., Modayil, J., White, A., and Machado, M. C. Loss of plasticity in continual deep reinforcement learning. In *Conference on Lifelong Learning Agents*, volume 232, pp. 620–636, 2023.

Abel, D., Barreto, A., Roy, B. V., Precup, D., van Hasselt, H. P., and Singh, S. A definition of continual reinforcement learning. In *NeurIPS*, 2023.

Achiam, J., Knight, E., and Abbeel, P. Towards characterizing divergence in deep q-learning. *arXiv preprint*, arXiv:1903.08894, 2019.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *NeurIPS*, 2021.

Arpit, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A. C., Bengio, Y., and Lacoste-Julien, S. A closer look at memorization in deep networks. In *ICML*, volume 70, pp. 233–242, 2017.

Asadi, K., Fakoor, R., and Sabach, S. Resetting the optimizer in deep RL: an empirical study. In *NeurIPS*, 2023.

Ash, J. and Adams, R. P. On warm-starting neural network training. *NeurIPS*, 33:3884–3894, 2020.

Bengio, E., Pineau, J., and Precup, D. Interference and generalization in temporal difference learning. In *ICML*, 2020.

Berariu, T., Czarnecki, W., De, S., Bornschein, J., Smith, S. L., Pascanu, R., and Clopath, C. A study on the plasticity of neural networks. *arXiv preprint*, arXiv:2106.00042, 2021.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint*, arXiv:1606.01540, 2016.

Ceron, J. S. O., Bellemare, M. G., and Castro, P. S. Small batch deep reinforcement learning. In *NeurIPS*, 2023.

Ceron, J. S. O., Courville, A., and Castro, P. S. In value-based deep reinforcement learning, a pruned network is a good network. In *ICML*. PMLR, 2024a.

Ceron, J. S. O., Sokar, G., Willi, T., Lyle, C., Farebrother, J., Foerster, J. N., Dziugaite, G. K., Precup, D., and Castro, P. S. Mixtures of experts unlock parameter scaling for deep rl. In *International Conference on Machine Learning*, pp. 38520–38540. PMLR, 2024b.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with A-GEM. In *ICLR*, 2019.

Chung, W., Cherif, L., Precup, D., and Meger, D. Parseval regularization for continual reinforcement learning. In *NeurIPS*, 2024.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, volume 119, pp. 2048–2056, 2020.

Damian, A., Ma, T., and Lee, J. D. Label noise SGD provably prefers flat global minimizers. In *NeurIPS*, pp. 27449–27461, 2021.

Dohare, S., Hernandez-Garcia, J. F., Rahman, P., Sutton, R. S., and Mahmood, A. R. Maintaining plasticity in deep continual learning. *arXiv preprint*, arXiv:2306.13812, 2023.

Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., and Sutton, R. S. Loss of plasticity in deep continual learning. *Nature*, 632(8026):768–774, 2024.

Ellis, B., Jackson, M. T., Lupu, A., Goldie, A. D., Fellows, M., Whiteson, S., and Foerster, J. Adam on local time: Addressing nonstationarity in rl with relative adam timesteps. *arXiv preprint*, arXiv:2412.17113, 2024.

Elsayed, M. and Mahmood, A. R. Addressing loss of plasticity and catastrophic forgetting in continual learning. *arXiv preprint, arXiv:2404.00781*, 2024.

Elsayed, M., Farrahi, H., Dangel, F., and Mahmood, A. R. Revisiting scalable hessian diagonal approximations for applications in reinforcement learning. In *ICML*, 2024a.

Elsayed, M., Lan, Q., Lyle, C., and Mahmood, A. R. Weight clipping for deep continual and reinforcement learning. *RLJ*, 5:2198–2217, 2024b.

Frati, L., Traft, N., Clune, J., and Cheney, N. Reset it and forget it: Relearning last-layer weights improves continual and transfer learning. In *ECAI 2024*, pp. 2998–3005. 2024.

Gogianu, F., Berariu, T., Rosca, M. C., Clopath, C., Busoniu, L., and Pascanu, R. Spectral normalisation for deep reinforcement learning: an optimisation perspective. In *ICML*, pp. 3734–3744, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *IOS Press*, pp. 87–90. 2016. doi: 10.3233/978-1-61499-649-1-87.

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *ICLR*, 2021.

Kumar, A., Agarwal, R., Ma, T., Courville, A., Tucker, G., and Levine, S. DR3: value-based deep reinforcement learning requires explicit regularization. In *ICLR*, 2022.

Kumar, S., Marklund, H., Rao, A., Zhu, Y., Jeon, H. J., Liu, Y., and Van Roy, B. Continual learning as computationally constrained reinforcement learning. *arXiv preprint, arXiv:2307.04345*, 2023a.

Kumar, S., Marklund, H., and Roy, B. V. Maintaining plasticity in continual learning via regenerative regularization. *arXiv preprint*, arXiv:2308.11958, 2023b.

Lewandowski, A., Tanaka, H., Schuurmans, D., and Machado, M. C. Directions of curvature as an explanation for loss of plasticity. volume arXiv:2312.00246, 2023.

Liu, J., Ceron, J. S. O., Courville, A., and Pan, L. Neuroplastic expansion in deep reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=20qZK2T7fa.

Liu, V., Wang, H., Tao, R. Y., Javed, K., White, A., and White, M. Measuring and mitigating interference in reinforcement learning. In *ICML*, 2023.

Lyle, C., Rowland, M., and Dabney, W. Understanding and preventing capacity loss in reinforcement learning. In *ICLR*, 2022.

Lyle, C., Zheng, Z., Khetarpal, K., Hasselt, H. V., Pascanu, R., Martens, J., and Dabney, W. Disentangling the causes of plasticity loss in neural networks. *arXiv preprint*, arXiv:2402.18762, 2024.

Ma, G., Li, L., Zhang, S., Liu, Z., Wang, Z., Chen, Y., Shen, L., Wang, X., and Tao, D. Revisiting plasticity in visual reinforcement learning: Data, modules and training stages. *arXiv preprint, arXiv:2310.07418*, 2023a.

Ma, Y., Tang, H., Li, D., and Meng, Z. Reining generalization in offline reinforcement learning via representation distinction. In *NeurIPS*, 2023b.

McKinney, W. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 1 edition, February 2013. ISBN 9789351100065. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1449319793.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48, pp. 1928–1937, 2016.

Muppidi, A., Zhang, Z., and Yang, H. Fast trac: A parameter-free optimizer for lifelong reinforcement learning. In *NeurIPS*, 2024.

Nakkiran, P., Kaplun, G., Kalimeris, D., Yang, T., Edelman, B. L., Zhang, F., and Barak, B. SGD on neural networks learns functions of increasing complexity. In *NeurIPS*, pp. 3491–3501, 2019.

Nath, S., Peridis, C., Ben-Iwhiwhu, E., Liu, X., Dora, S., Liu, C., Kolouri, S., and Soltoggio, A. Sharing lifelong reinforcement learning knowledge via modulating masks. In *Conference on Lifelong Learning Agents*, volume 232, pp. 936–960, 2023.

Nauman, M., Bortkiewicz, M., Miłoś, P., Trzcinski, T., Ostaszewski, M., and Cygan, M. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. In *ICML*, 2024.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P., and Courville, A. C. The primacy bias in deep reinforcement learning. In *ICML*, Proceedings of Machine Learning Research, 2022a.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. The primacy bias in deep reinforcement learning. In *ICML*, pp. 16828–16847, 2022b.

Nikishin, E., Oh, J., Ostrovski, G., Lyle, C., Pascanu, R., Dabney, W., and Barreto, A. Deep reinforcement learning with plasticity injection. *NeurIPS*, 2024.

Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. doi: 10.1109/MCSE.2007.58.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience replay for continual learning. In *NeurIPS*, pp. 348–358, 2019.

Schaul, T., Barreto, A., Quan, J., and Ostrovski, G. The phenomenon of policy churn. *arXiv preprint*, arXiv:2206.00730, 2022.

Schneider, J., Schumacher, P., Guist, S., Chen, L., Häufle, D. F. B., Schölkopf, B., and Büchler, D. Identifying policy gradient subspaces. *arXiv preprint, arXiv:2401.06604*, 2024.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint*, arXiv:1707.06347, 2017.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *ICML*, volume 80, pp. 4535–4544, 2018.

Schwarzer, M., Ceron, J. S. O., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. Bigger, better, faster: Human-level atari with human-level efficiency. In *ICML*, pp. 30365–30380, 2023.

Shah, H., Tamuly, K., Raghunathan, A., Jain, P., and Netrapalli, P. The pitfalls of simplicity bias in neural networks. In *NeurIPS*, 2020.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. A. Deterministic policy gradient algorithms. In *ICML*, 2014.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. The dormant neuron phenomenon in deep reinforcement learning. In *ICML*, pp. 32145–32168, 2023.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16: 285–286, 1988.

Tang, H. and Berseth, G. Improving deep reinforcement learning by reducing the chain effect of value and policy churn. In *NeurIPS*, 2024.

Tang, H., Zhang, M., Chen, C., and Hao, J. The ladder in chaos: Improving policy learning by harnessing the parameter evolving path in a low-dimensional space. In *NeurIPS*, 2024.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P., and Riedmiller, M. A. Deepmind control suite. *arXiv preprint*, arXiv:1801.00690, 2018.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. Deep reinforcement learning and the deadly triad. *arXiv preprint*, arXiv:1812.02648, 2018.

Van Rossum, G. and Drake Jr, F. L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Wang, L., Zhang, X., Su, H., and Zhu, J. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Wu, T., Luo, L., Li, Y.-F., Pan, S., Vu, T.-T., and Haffari, G. Continual learning for large language models: A survey. *arXiv preprint, arXiv:2402.01364*, 2024.

Xie, A., Harrison, J., and Finn, C. Deep reinforcement learning amidst lifelong non-stationarity. In *ICML*, 2021.

Yang, Y., Zhang, G., Xu, Z., and Katabi, D. Harnessing structures for value-based planning and reinforcement learning. In *ICLR*, 2020.

Young, K. and Tian, T. Minatar: An atari-inspired testbed for more efficient reinforcement learning experiments. *arXiv preprint*, arXiv:1903.03176, 2019.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. In *NeurIPS*, 2020.

Zhai, Y., Tong, S., Li, X., Cai, M., Qu, Q., Lee, Y. J., and Ma, Y. Investigating the catastrophic forgetting in multimodal large language model fine-tuning. In *Conference on Parsimony and Learning*, pp. 202–227. PMLR, 2024.

Zhang, W., Li, Y., Yang, B., and Lu, Z. Tackling non-stationarity in reinforcement learning via causal-origin representation. In *ICML*, volume 235, pp. 59264–59288, 2024.

A. Experimental Details

A.1. Continual RL

Environment Setups We follow the setups in (Muppidi et al., 2024). For Gym Control, we use four environments: CartPole-v1, Acrobot-v1, LunarLander-v2 and MountainCar-v0. For each environment, a task sequence $\mathbb{T}$ is built by chaining k instances of the environment with a unique Gaussian observation noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ sampled once for each. The number k is 10 and the interaction budget N is 0.16M for all the environments, except that k is 5 and N is 0.32M for MountainCar-v0. For CartPole-v1, Acrobot-v1 and LunarLander-v2, we use $\sigma = 2.0$ as provided in (Muppidi et al., 2024). For MountainCar-v0, it was not originally included in (Muppidi et al., 2024) and we found $\sigma = 2.0$ is too large for this environment to be learnable. Therefore, we use $\sigma = 0.02$ for MountainCar-v0.

For ProcGen, we use all sixteen environments in the suite, while only four (i.e., Starpilot, Fruitbot, Chaser, Dodgeball) were adopted in (Muppidi et al., 2024). For each environment, the task sequence consists of 5 instances procedurally generated by sampling a unique level of the environment. The budget is 2M steps per task.

Algorithm Implementation We use the official implementation[2] of TRAC (Muppidi et al., 2024) as our code base. We implement all other methods to ensure that they only differ in the corresponding algorithm features. For the hyperparameters of the PPO baseline agent, we use the default values recommended in the code base and keep them consistent across all the methods. For the hyperparameters specific to each method, we search around the recommendation values in the original papers and report the best.

For C-CHAIN, we follow the practice in (Tang & Berseth, 2024) and we only reduce churn for the policy network of the PPO agent. The coefficient λ_π of the policy churn reduction regularization is determined by an auto-adjustment mechanism, which keeps a consistent relative scale (denoted by β) between the churn reduction regularization terms and the original DRL objectives. Specifically, by maintaining the running means of the absolute PPO loss $|\bar{L}_\pi|$ and the policy churn reduction term $|\bar{L}_{\text{PC}}|$, the churn reduction regularization coefficient λ_π is computed dynamically as $\lambda_\pi = \beta \frac{|\bar{L}_\pi|}{|\bar{L}_{\text{PC}}|}$. For the size of B_{ref}, we did the experiments with different batch sizes on continual Gym control tasks. Our findings were that using a 2x, 4x or 8x batch size for B_{ref} (compared to the regular training batch size) sometimes improved the learning performance, but not consistently. To some degree, increasing the batch size of B_{ref} acts similarly to increasing the regularization coefficient, as both of them reduce more churn. Thus, we did not search for the best batch size for B_{ref} and set it equal to the training batch size to alleviate the hyperparameter choice burden.

The hyperparameters are provided in Table 5. The codebase will be released upon publication of this work.

Table 5. **Hyperparameters of different methods used in continual Gym Control and ProcGen environments.** We use '-' to denote the 'not applicable' situation.

Agent	Hyperparam	Gym Control	ProcGen
Vanilla PPO (Schulman et al., 2017)	Learning Rate	$1e^{-3}$	$1e^{-3}$
	Optimizer	Adam	Adam
	Discount Factor (γ)	0.99	0.99
	GAE Parameter (λ)	0.95	0.95
	Training Interval	800 steps	1000 steps
	Mini-batch Size	32	125
	Update Epoch	5	3
	Clipping Range Parameter (ϵ)	0.2	0.2
	Entropy Loss Coef	0	0.01
TRAC (Muppidi et al., 2024)	(Hyperparam Free)	-	-
Weight Clipping (Elsayed et al., 2024b)	Clipping Range	Best in $\{0.1, 0.5, 1, 10\}$	Best in $\{0.1, 0.5, 1, 10\}$
L2 Init (Kumar et al., 2023b)	Regularization Coeff	Best in $\{0.1, 1, 10, 100, 1000\}$	Best in $\{0.1, 1, 10, 100, 1000\}$
C-CHAIN (Ours)	Target Relative Loss β for Auto λ_π	Best in $\{1000, 1e4, 1e5\}$	Best in $\{1000, 1e4, 1e5\}$

Compute Resource For the continual Gym Control and ProcGen experiments, we allocate a single V100 GPU, 16 CPUs and 32GB memory for 4 to 6 jobs, typically running for around 4 hours for Gym Control and 20 hours for ProcGen to

[2] https://github.com/ComputationalRobotics/TRAC

complete.

A.2. Continual DeepMind Control Suite

We use the public implementations of PPO and DMC task setups in `CleanRL`[3] as our codebase. The actor and critic networks are two-layer MLPs with 256 units for each layer. We made no change to the network structure, recommended hyperparameter choices, etc. The hyperparameters are in Table 6.

Upon the code base, we build three continual RL settings in with DMC tasks:

- *Continual Walker*: run Walker-stand, Walker-walk, Walker-run, sequentially.
- *Continual Quadruped*: run Quadruped-walk, Quadruped-run, Quadruped-walk, sequentially (repeat because only two Quadruped tasks are available in DMC).
- *Continual Dog*: run Dog-stand, Dog-walk, Dog-run, Dog-trot, sequentially.

We run 1M steps for each task.

Table 6. Hyperparameters of PPO and C-CHAIN used in continual DMC environments. The values of conventional hyperparameters are taken from the recommended values in `CleanRL`.

PPO Hyperparameters	
Learning Rate	$3e^{-4}$
Training Interval	2048 steps
Discount Factor (γ)	0.99
GAE Parameter (λ)	0.95
Num. of Minibatches	32
Update Epoch	10
Clipping Range Parameter (ϵ)	0.2
Target Relative Loss β for Auto λ_π	0.5 for Continual Walker 0.05 for Continual Quadruped and Continual Dog

Similarly, we compare PPO Oracle, PPO Vanilla, and C-CHAIN in the three settings. The results are summarized in Table 3 and Figure 9 shows the learning curves. We can observe that C-CHAIN PPO significantly outperforms PPO in all three settings, showing the effectiveness for continual control. C-CHAIN also outperforms PPO Oracle because it does not reset for each task and thus can *transfer* the learned skill from previous tasks to future ones (e.g., Dog-Stand to Dog-Walk).

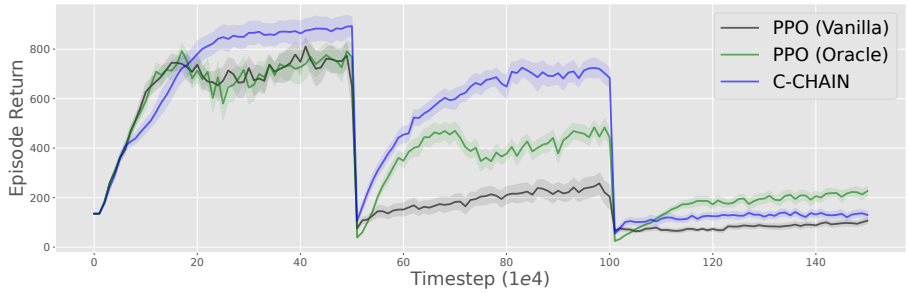
A.3. Continual MinAtar

We use the official code of MinAtar paper[4] as our codebase. For DoubleDQN, we modify the DQN implementation provided in the official MinAtar code with no change to the network structure, recommended hyperparameter choices, etc. We use DoubleDQN as the base agent, and apply C-CHAIN to the training of the value network, with no modification to the hyperparameters of DoubleDQN. The hyperparameters are summarized in Table 7.

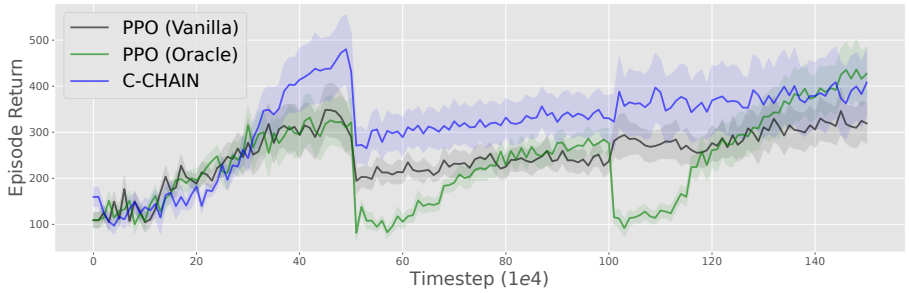
To build the continual MinAtar setting, we chain three tasks: SpaceInvaders, Asterix, Seaquest, by padding the observation space to be $[10, 10, 10]$. We run 1.5M steps for each task, i.e., 4.5M in total.

We compare DoubleDQN (i.e., Vanilla) and C-CHAIN DoubleDQN with 12 random seeds. The results are shown in Table 4 and Figure 10 shows the learning curves. The results show that C-CHAIN also improves the continual learning performance upon DoubleDQN in a sequence of totally different tasks. This indicates that C-CHAIN is not only beneficial to PPO in continual RL but has the potential to be a general and easy-to-use remedy to different off-the-shelf RL algorithms in combating plasticity loss in continual RL scenarios.

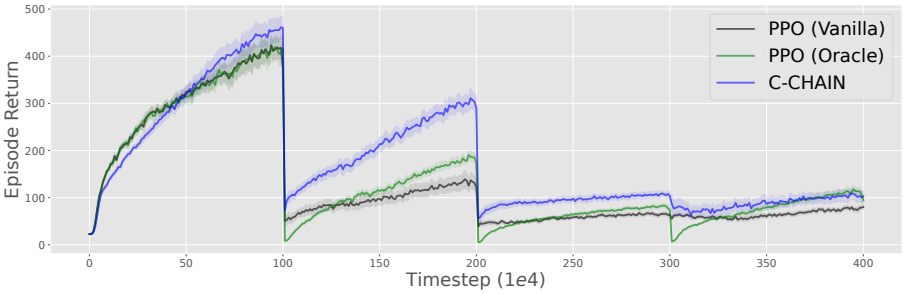
[3] https://github.com/vwxyzjn/cleanrl
[4] https://github.com/kenjyoung/MinAtar



(a) Continual Walker



(b) Continual Quadruped



(c) Continual Dog

Figure 9. **Learning curves for PPO and C-CHAIN PPO in continual DMC.** The curves and the shades are means and standard errors over twelve seeds.

Table 7. Hyperparameters of DoubleDQN and C-CHAIN used in continual MinAtar environments. The values of conventional hyperparameters are taken from the recommended values in (Young & Tian, 2019).

DoubleDQN Hyperparameters	
Learning Rate	$3e^{-4}$
Training Interval	1 step
Discount Factor (γ)	0.99
Hard Replacement Interval	1000 steps
Replay Buffer Size	0.5M
Batch Size	32
Initial ϵ	1.0
End ϵ	0.1
ϵ Decay Steps	0.5M
Initial Random Steps	10k
Target Relative Loss β for Auto λ_Q	0.01

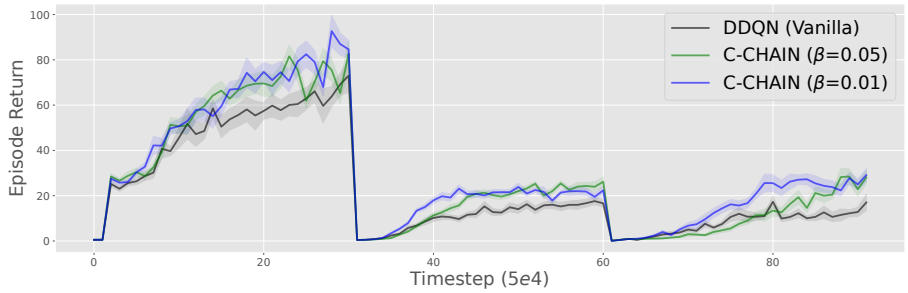


Figure 10. **Learning curves for DoubleDQN and C-CHAIN in continual MinAtar.** The curves and the shades are means and standard errors over twelve seeds.

A.4. Continual Supervised Learning

We follow the settings in L2 Init (Kumar et al., 2023b) and adopt RandomLabel-MNIST and Permuted-MNIST as our testbed. For RandomLabel-MNIST, each task in the task sequence is a classification learning task with 1200 images randomly sampled from the MNIST dataset and random labels assigned to each individual image. The task sequence consists of 50 tasks and we use train the agent for a total of 400 epochs per task. For Permuted-MNIST, each task is a classification learning task with 10000 images sampled from MNIST. Different from RandomLabel-MNIST, a Permuted MNIST task is characterized by applying a fixed randomly sampled permutation to the input pixels of all the images. The task sequence consists of 500 tasks and the agent is trained for one epoch per task.

We use the official implementation of L2 Init[5] as the code base and implement C-CHAIN and Weight Clipping with simple modifications. We make no modifications to common hyperparameters and use the recommended values in the code base.

The results are summarized in Table 8. We can observe that C-CHAIN improves the vanilla agent but does not perform on par with L2 Init and Weight Clipping. This shows that the efficacy of C-CHAIN is relatively limited in the two continual supervised learning environments, especially in contrast to its superiority in the continual RL experiments. We hypothesize this is because RL suffers more from churn accumulation and thus more severe plasticity loss due to the chain effect of churn (Tang & Berseth, 2024). Besides, Permuted-MNIST and RandomLabel-MNIST are simple, where the agent can find a good solution near the initialization. We expect to evaluate C-CHAIN in more complex continual supervised learning environments in the future.

B. Empirical NTK Analysis Details

During the learning process, we collect the empirical NTK matrices in each training iteration of the PPO agent. Specifically, for the interaction experience collected within each iteration, we divide it into m mini-batches randomly ($m = 100$ for our NTK analysis in ProcGen). Then, for each mini-batch, we compute the gradient of the policy network's parameters θ regarding the PPO policy optimization objective, denoted by g_i with $i \in \{1, 2, \ldots, m\}$. The empirical NTK matrix N_θ

[5] https://github.com/skumar9876/L2_Init

Table 8. **Performance comparison in continual supervised learning environments.** The reported scores are the average performance (Equation 1) in terms of accuracy with mean and standard error over three seeds.

Algorithm	RandomLabel-MNIST	Permuted-MNIST
Vanilla	0.1501 ± 0.0030	0.6430 ± 0.0029
C-CHAIN	0.2482 ± 0.0141	0.6797 ± 0.0042
L2 Init	0.8607 ± 0.0021	0.8039 ± 0.0027
Weight Clipping	0.4304 ± 0.0052	0.8281 ± 0.0001

is then computed by following the definition in Equation 2: $N_\theta(i,j) = g_i^\top g_j$ for $i, j \in \{1, 2, \ldots, m\}$. With the empirical NTK matrix, statistics for the off-diagonal values (i.e., Figure 6, *below* and Figure 11, *middle*) and the diagonal values (i.e., Figure 11, *below*) can be computed conveniently.

For the computation of approximate rank, we follow the prior convention in (Yang et al., 2020; Kumar et al., 2021). For a matrix N, let $\{\sigma_i(N)\}$ be the singular values of it arranged in decreasing order, i.e., $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$. the approximate rank regarding a threshold δ is defined as $\text{srank}_\delta(N) = \min\{k : \sum_{i=1}^k \sigma_i(N) \geq (1-\delta)\sum_{i=1}^d \sigma_i(N)\}$. In this paper, we use $\delta = 0.01$. In other words, this means the approximate rank is the first k singular values that hold more than 99% information of all singular values. Accordingly, we use this computation for the results of approximate rank in Figure 6 *(above)* and Figure 11 *(above)*.

Apart from our analysis on empirical NTK shown in Figure 6, we provide additional NTK analysis by including TRAC in comparison and also showing the summation of the diagonal values of empirical NTK matrix. The results are shown in Figure 11. C-CHAIN not only suppresses the off-diagonal values to decorrelate gradients, but also suppresses the diagonal values to prevent the increased scale of gradient, which is also deemed to be a pathology of neural network. Moreover, we found TRAC lies between C-CHAIN and Vanilla. This indicates that the NTK statistics used in our work also explain the efficacy of TRAC.

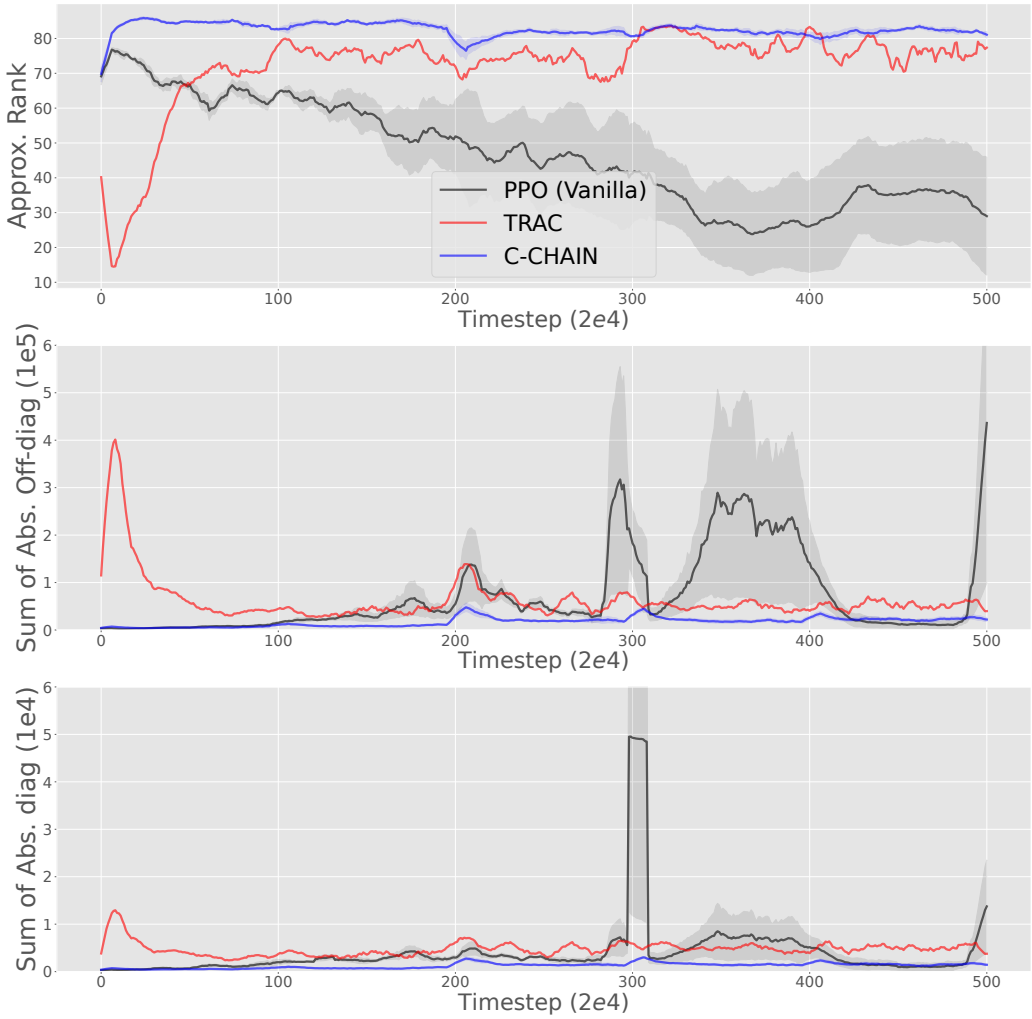


Figure 11. **Analysis on NTK matrix** in terms of approximate rank *(above)*, the sum of absolute off-diagonal values *(middle)*, and the sum of absolute diagonal values*(below)* in continual ProcGen Starpilot.

C. Complete Learning Curves

The learning curves for the missing twelve contiunal ProcGen environments are provided in Figure 12, Figure 13 and Figure 14.

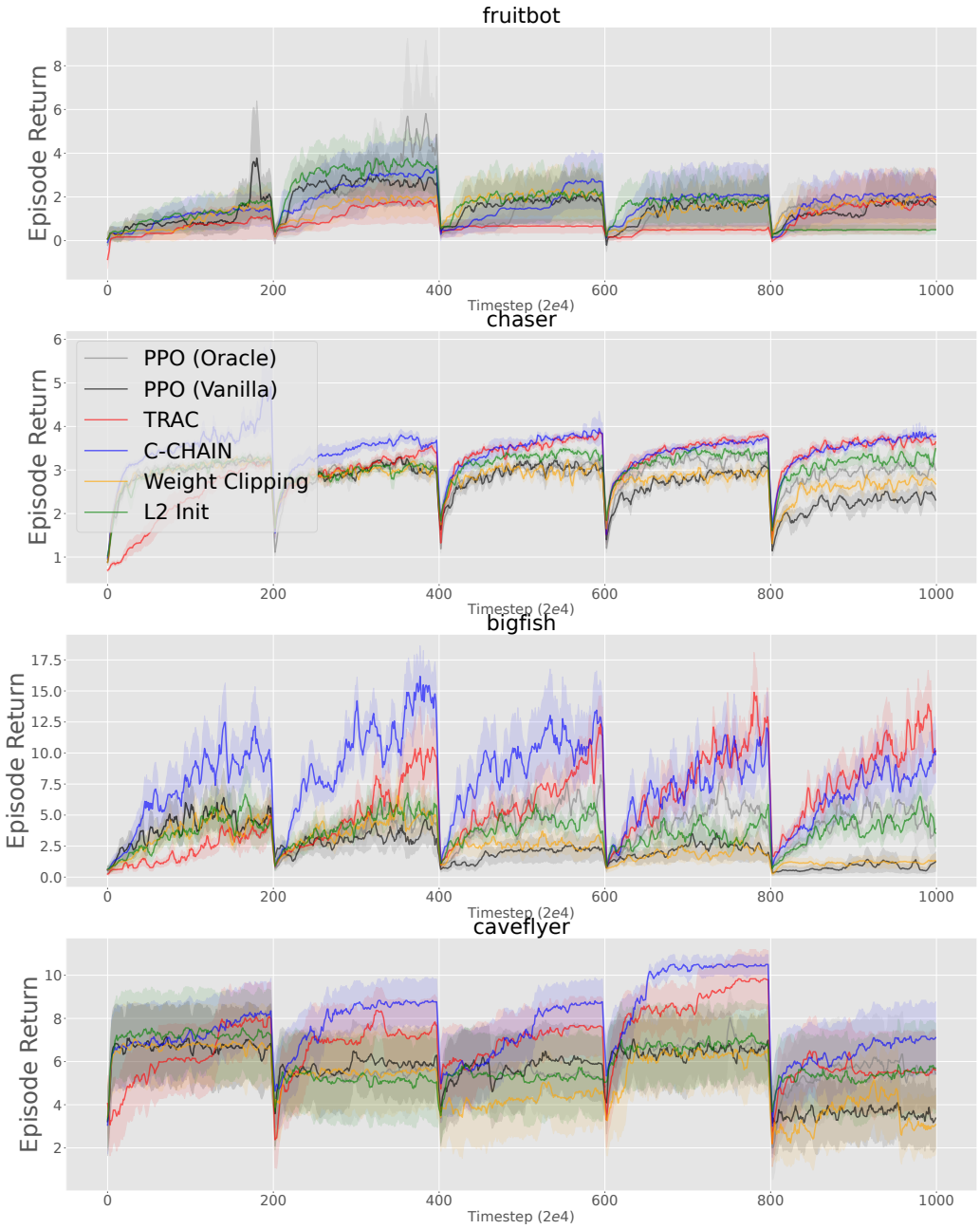


Figure 12. **Learning curves of different methods in four (of sixteen) continual ProcGen environments**: fruitbot, chaser, bigfish, caveflyer. The curves and the shades are means and standard errors over six seeds.

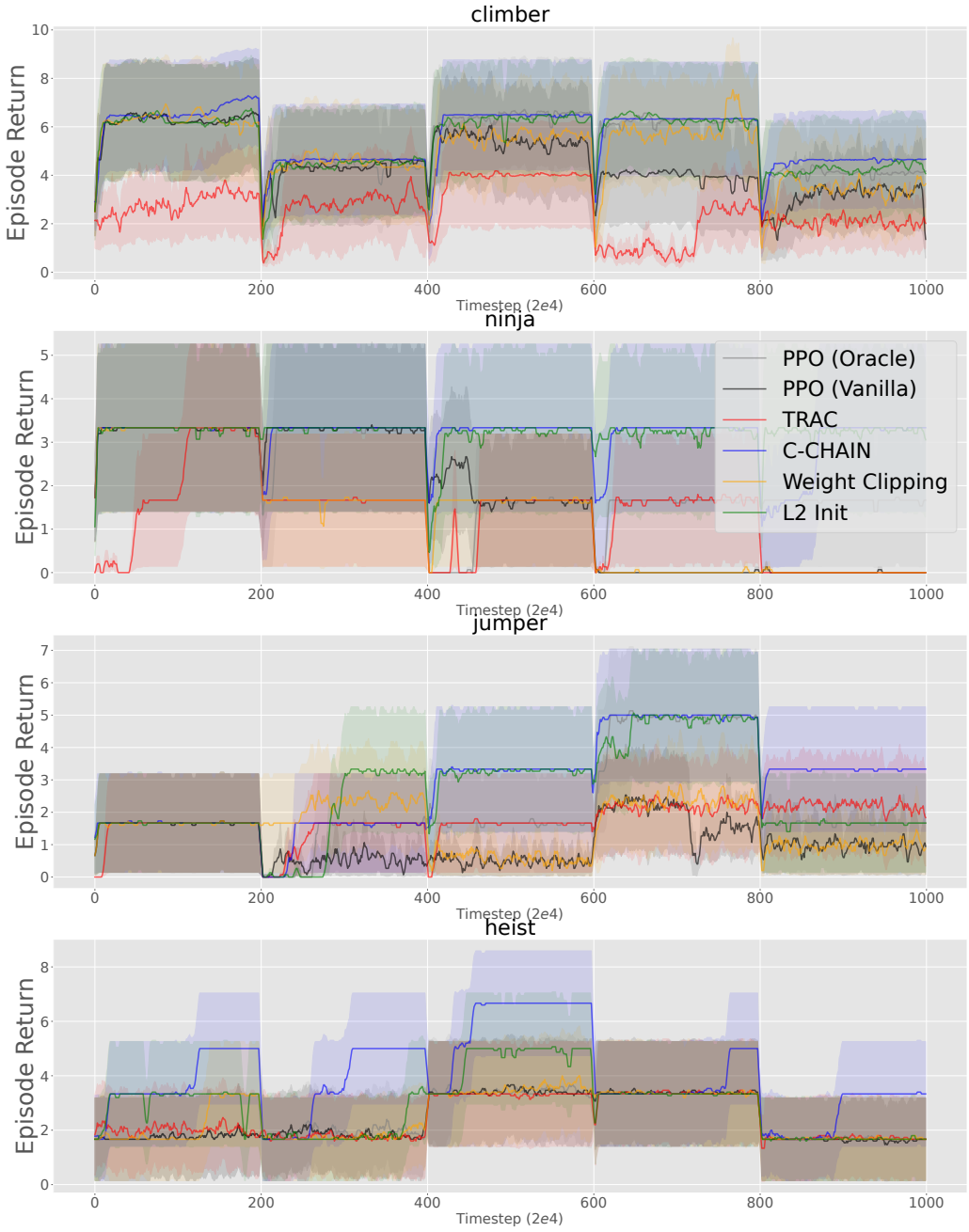


Figure 13. **Learning curves of different methods in four (of sixteen) continual ProcGen environments**: climber, ninja, jumper, heist. The curves and the shades are means and standard errors over six seeds.

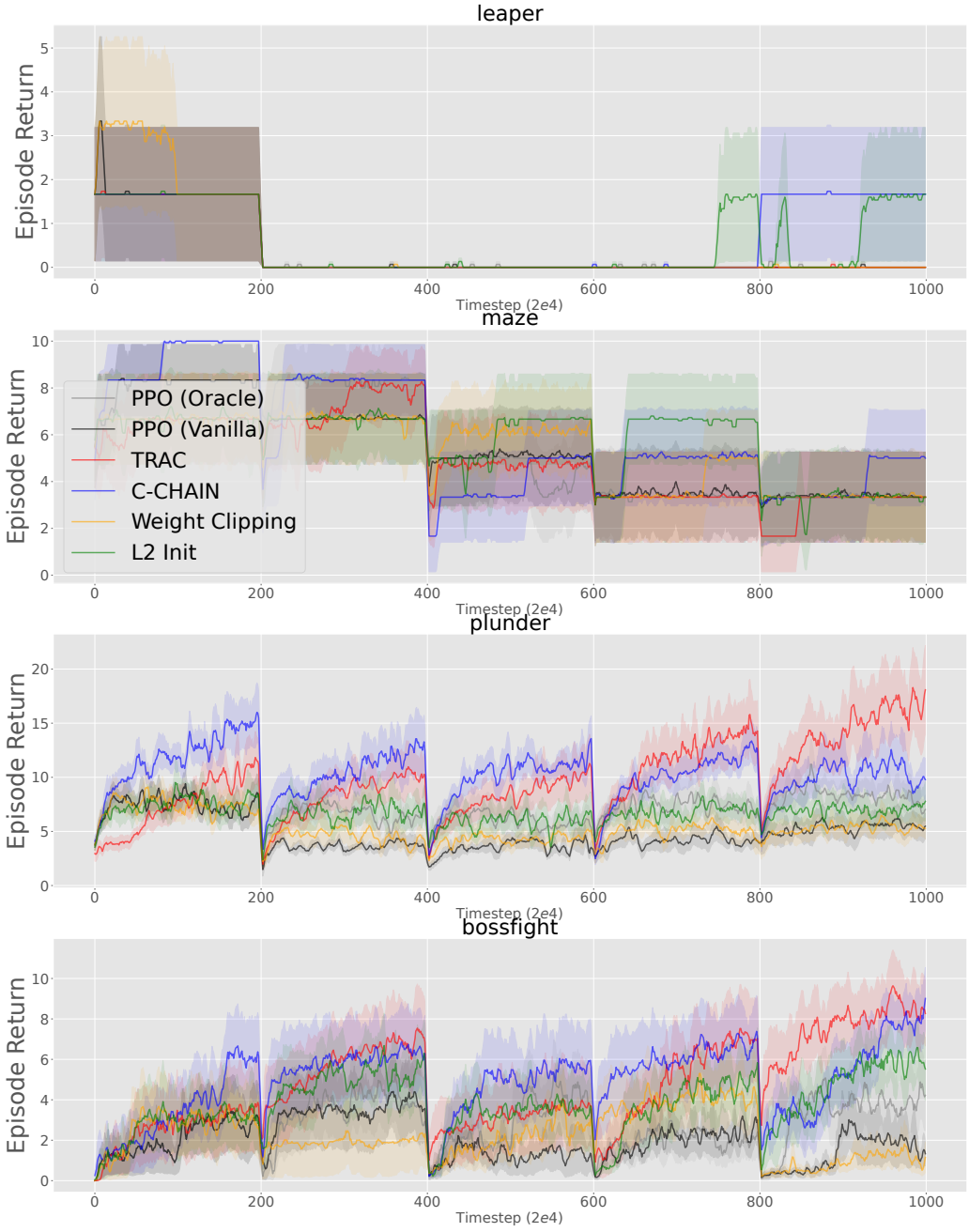


Figure 14. **Learning curves of different methods in four (of sixteen) continual ProcGen environments**: leaper, maze, plunder, bossfight. The curves and the shades are means and standard errors over six seeds.