Amphibian: A Meta-Learning Framework for Rehearsal-Free, Fast Online Continual Learning

Gobinda Saha *gsaha@purdue.edu*
Department of Electrical and Computer Engineering
Purdue University

Kaushik Roy *kaushik@purdue.edu*
Department of Electrical and Computer Engineering
Purdue University

Reviewed on OpenReview: *https://openreview.net/forum?id=n4AaKOBWbB*

Abstract

Online continual learning is challenging as it requires fast adaptation over a stream of data in a non-stationary environment without forgetting the knowledge acquired in the past. To address this challenge, in this paper, we introduce Amphibian - a gradient-based meta-learner that learns to scale the direction of gradient descent to achieve the desired balance between fast learning and continual learning. For this purpose, using only the current batch of data, Amphibian minimizes a meta-objective that encourages alignments of gradients among given data samples along selected basis directions in the gradient space. From this objective, it learns a diagonal scale matrix in each layer that accumulates the history of such gradient alignments. Using these scale matrices Amphibian updates the model online only in the directions having positive cumulative gradient alignments among the data observed so far. With evaluation on standard continual image classification benchmarks, we show that such meta-learned scaled gradient descent in Amphibian achieves better accuracy in online continual learning than relevant baselines while enabling fast learning with less data and few-shot knowledge transfer to new tasks. We also introduce Amphibian-β a unified and principled framework for analyzing and understanding the fast learning and continual learning dynamics. Additionally, with loss landscape visualizations, we show such gradient updates incur minimum loss to the old task enabling fast continual learning in Amphibian.

1 Introduction

Autonomous intelligent systems are envisioned to operate in non-stationary environments where distribution of online data streams changes over time. In such environments, AI models (usually artificial neural networks, ANNs) need to acquire knowledge quickly while maintaining the stability of past experiences. This is a challenging scenario as the learning method needs to strike the right balance between *learning without forgetting* and *fast learning* objectives. However, standard gradient-based training methods for ANNs overwrite the past knowledge with the information from the new batch of data - leading to 'catastrophic forgetting' (28; 46). Such forgetting prevents effective knowledge transfer from the past thus also hampering fast learning ability.

To address these challenges, a popular line of work in continual learning (CL) (33; 17) uses memory rehearsal (34; 10; 26; 38) - where a subset of past data is stored in a memory buffer and used with the current batch of data to jointly train the model. Such rehearsal-based strategy guides the optimization process such that losses of the past data do not increase, preventing catastrophic forgetting. However, effectiveness of these methods depends on large memory storage which also arises data privacy concerns. In contrast, rehearsal-free methods (20; 50; 36; 39) in continual learning use explicitly designed regularization objectives and/or constrained gradient update rules to prevent forgetting. Though these methods are effective in offline (multi-epoch) CL setups, compared to rehearsal-based methods they underperform in online continual learning (OCL) (27). This is primarily due to the added objective or constraints that focus on forgetting mitigation rather than encouraging fast learning. From fast learning viewpoint, meta-learning (14) or 'learning to learn' (42) is an exciting proposition since it optimizes a meta-objective that encourages representation learning in ANNs suitable for fast adaptation. Such meta-objective is adapted in (18; 4; 8) for pre-training models offline, then deployed for continual learning tasks. In contrast, (32; 16) adapted the meta-objective for fully online continual learning. However, they use memory rehearsal to mitigate forgetting.

In this paper, we propose a meta-learner - **Amphibian** - that learns fast with minimum forgetting during online continual learning. Without any memory rehearsal, Amphibian achieves better balance between *learning without forgetting* and *fast learning* with three key components. **First**, to obtain meta-gradients, it optimizes a meta-objective that on top of minimizing loss on the given batch of samples, encourages their gradient alignments along the selected basis directions representing the gradient space. **Second**, from the same meta-objective, it *learns* diagonal scaling matrices that contain learning/scaling rates along each gradient basis. In our formulation, each scale value accumulates the history of gradient alignments (along the corresponding basis) among the observed samples over the entire learning sequence. **Finally**, it scales the meta-gradients with scaling matrices to update the model along the directions with cumulative positive gradient alignments among the observed data. Thus, the combination of meta-objective optimization and meta-learned gradient scaling enables Amphibian to learn fast and continually. We evaluate Amphibian in both task- and class-incremental online continual learning setups (16; 40) on long and diverse sequences of image classification tasks (including ImageNet-100) using different network architectures (including ResNet) and achieve better performance in both continual and fast learning metrics compared to the twelve most relevant baselines.

Additionally, we propose a regularized version of Amphibian - **Amphibian-β** that offers a unified and principled framework for analyzing and understanding the dynamics of fast learning and continual learning in popular CL algorithms. Within this framework, by varying a single hyperparameter, β, we have shown to achieve the functionality of meta-learning-based (this work), gradient-constraint-based (36; 37), and regularization-based (20; 50) CL algorithms. We show explicit weight regularization or gradient constraints used in the regularization and gradinet-constrained-based methods trade off fast learning ability in favor of mitigating forgetting, hence they underperform in the OCL setup. Our findings reveal it is hard to design explicit regularization and gradient constraints for achieving the perfect balance between fast learning and continual learning. Hence, Amphibian-like meta-learners that automatically learn these constraints from the data would be a promising candidate for this OCL setup.

We summarize the **contributions** of this paper as follows:

- We introduce Amphibian which minimizes a novel meta-objective and uses meta-learned gradient scaling to enable fast online continual learning without rehearsal.

- We provide a detailed derivation of this objective and provide justification for its effectiveness in fast continual learning.

- With evaluation on long sequences of tasks, we show that Amphibian not only learns continually with higher accuracy but also demonstrates the ability of a truly fast learner by learning fast with less data and enabling few-shot knowledge transfer to the new tasks compared to the relevant baselines.

- We analyze a regularized version of Amphibian - Amphibian-β and provide insight that regularized objectives or constraints used by the representative rehearsal-free methods to minimize forgetting restrict the fast learning ability of the model in OCL.

- With visualization of loss landscapes of sequential tasks, we show that scaled model update in Amphibian along gradient directions with positive cumulative gradient alignments induces minimum to no increase in loss of the past tasks which enables continual learning.

2 Related Works

Online Continual Learning (OCL). We consider a supervised learning setup where T tasks $[\tau_1, \tau_2, ..\tau_T]$ are learnt by observing their training data $[\mathcal{D}^1, \mathcal{D}^2, ..\mathcal{D}^T]$ sequentially. At any time-step j, the learning model receives a batch of data, $\mathcal{B}_i^j$ or simply $\mathcal{B}_i = \{(x_n^j, y_n^j)\}_{n=0}^{N_i}$ as the set of N_i input-label pairs randomly drawn from the current data stream, $\mathcal{D}^i$. In **online** continual learning (27), the model needs to learn from a single pass over these data streams with the objective of minimizing the empirical risk on the data from all the t tasks seen so far. The objective (16; 43) is given by:

$$\sum_{i=1}^{t} \mathbb{E}_{\mathcal{B}_i}[\ell_i(\theta; \mathcal{B}_i)] = \mathbb{E}_{\mathcal{B}_{1:t}} \sum_{i=1}^{t} [\ell_i(\theta; \mathcal{B}_i)]. \tag{1}$$

Here $\ell_i(.;.)$ is the loss function to be minimized for task τ_i by updating the model parameters θ.

Rehearsal-based Methods. The above risk minimization requires all the data, $\mathcal{B}_{1:t-1}$ from past tasks which may not be accessible to OCL agents at the same time. Rehearsal or experience replay (ER) (34; 10) methods offer a solution by storing a limited amount of past data in episodic memory, $\mathcal{D}^{\mathcal{M}}$. Such techniques then sample a memory batch, $\mathcal{B}_{1:t-1}^{\mathcal{M}} \sim \mathcal{D}^{\mathcal{M}}$ (that approximates $\mathcal{B}_{1:t-1}$) for jointly minimizing the objective in Equation 1 with the current batch, $\mathcal{B}_t$. Later works built on this idea where they differ in the way memory is selected and replayed. For instance: GSS (2) selects memory based on gradients, MIR (1) selects memory that incurs maximum change in loss, ASER (40) performs memory selection and retrieval using Shapley value scores. To improve replay, RAR (51) uses repeated augmented rehearsal, CLS-ER (3) proposes dual memory learning and DER (7) uses logit distillation loss. Gradient Episodic Memory (GEM) (26) and Averaged-GEM (A-GEM) (9) use memory data to compute gradient constraints for new task so that loss on past tasks does not increase. We refer to (6; 41) for a recent literature review on online continual learning.

Rehearsal-free Methods. One line of work in this category expands the network (35; 48) for continual learning but they are not evaluated for OCL. Other methods under this category minimize the loss $(\mathbb{E}_{\mathcal{B}_t}[l_t(\theta; \mathcal{B}_t)])$ on the current batch with additional regularization terms and/or constraints on gradient updates. For example, Elastic Weight Consolidation (EWC) (20) and Synaptic Intelligence (SI) (50) add quadratic regularization terms that penalize changes in the important parameters for the past task. While EWC computes parametric importance from the Fisher diagonal matrix after training, SI finds them online during training from the loss sensitivity of the parameters. Gradient projection methods (49; 13; 36; 37) constrain the current gradient, $\nabla_\theta l_t$ to be orthogonal to the past gradient directions. For instance, Gradient Projection Memory (GPM) (36) optimizes a new task in the orthogonal directions to the gradient spaces important for the past tasks, whereas Scaled Gradient Projection (SGP) (37) relaxes the constraint in GPM to allow gradient updates along the old gradient spaces. Natural Continual Learning (NCL) (19) learns continually by combining gradient projection with regularization. Our proposed Amphibian is also a rehearsal-free method. However, unlike these methods it does not need any explicit regularization or constraints; rather it *meta-learns from data to scale the gradients* for fast continual learning. In the next section, we introduce a meta-learning method - MAML (14) and then discuss relevant meta-learning-based continual learning works.

3 Preliminaries

Model-Agnostic Meta-Learning (MAML). MAML (14) is a widely used gradient-based meta-learner that utilizes bi-level (inner-loop and outer-loop) optimization to obtain model parameters, θ_0 that is amenable to fast adaptation to new tasks. MAML trains the model on a set of T tasks simultaneously where each task, τ_i has a dataset, $\mathcal{D}^{\tau_i} = \{\mathcal{D}_{in}^{\tau_i}, \mathcal{D}_{out}^{\tau_i}\}$ partitioned for inner and outer loop optimization. One step of **inner-loop optimization** on task τ_i is defined as:

$$U(\theta_0^{\tau_i}; \mathcal{D}_{in}^{\tau_i}) = \theta_1^{\tau_i} = \theta_0^{\tau_i} - \alpha \nabla_{\theta_0^{\tau_i}} \ell_{in}(\theta_0^{\tau_i}; \mathcal{D}_{in}^{\tau_i}), \tag{2}$$

where $U(.;.)$ is a stochastic gradient descent (SGD) operator, α is learning rate and ℓ_{in} is inner-loop loss function. U can be composed for k such updates as $U_k(\theta_0^{\tau_i}; \mathcal{D}_{in}^{\tau_i}) = U.. \circ U \circ U(\theta_0^{\tau_i}; \mathcal{D}_{in}^{\tau_i}) = \theta_k^{\tau_i}$. In the **outer-loop optimization**, loss for each task is computed at corresponding, $\theta_k^{\tau_i}$ on $\mathcal{D}_{out}^{\tau_i}$ and expected loss over all the tasks (meta-loss) is minimized (Equation 3) to obtain the parameters, θ_0.

$$\min_{\theta_0} \mathbb{E}_{\tau_{1:t}}[\ell_{out}(\theta_k^{\tau_i}; \mathcal{D}_{out}^{\tau_i})] \tag{3}$$

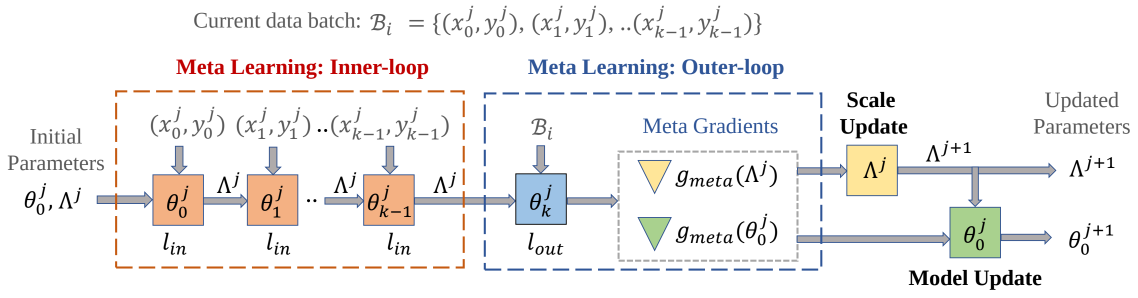
Current data batch: $\mathcal{B}_i = \{(x_0^j, y_0^j), (x_1^j, y_1^j), ..(x_{k-1}^j, y_{k-1}^j)\}$
Meta Learning: Inner-loop
Meta Learning: Outer-loop
Initial Parameters
θ_0^j, Λ^j
(x_0^j, y_0^j) (x_1^j, y_1^j) ..(x_{k-1}^j, y_{k-1}^j)
θ_0^j θ_1^j θ_{k-1}^j Λ^j
l_{in}
$\mathcal{B}_i$
θ_k^j
l_{out}
Meta Gradients
$g_{meta}(\Lambda^j)$
$g_{meta}(\theta_0^j)$
Scale Update
Λ^{j+1}
Updated Parameters
θ_0^{j+1}
Model Update

Figure 1: Illustration of continual learning in Amphibian on a batch of data. Given the model, θ_0^j and scale matrix, Λ^j, **first** we perform k inner-loop gradient update on θ_0^j to obtain θ_k^j with the samples of the current batch, β_i. In **second** step, we compute the meta-loss on θ_k^j with the entire batch, β_i to obtain meta gradients. In **third** step we update Λ^j and **finally** the model, θ_0^j with these gradients.

Meta-learning and Continual Learning. The above meta-loss minimization trains a model for fast adaptation, however it does not explicitly encourage continual learning. Thus, (18) proposed online-aware meta-learning where at first, a model is pre-trained offline on a set of tasks to learn a better representation for CL, then keeping that representation frozen, the rest of the network is fine-tuned on CL tasks. The authors in (4; 23; 8) used such meta-learning-based offline pre-training strategy, while allowing varying degrees of adaptation to the model during CL tasks. In contrast, Meta Experience Replay (MER) (32) combines meta-objective of Reptile (30) with memory rehearsal for OCL, whereas La-MAML (16) minimizes the MAML objective (Equation 3) in online setup where losses on the past tasks are computed on the memory batch, $\mathcal{B}_{1:t-1}^{\mathcal{M}} \sim \mathcal{D}^{\mathcal{M}}$. Unlike these methods, we train a model from scratch with a meta-objective (Section 4) without rehearsal for fast online continual learning.

Representation of the Gradient Space in Neural Network: In (36) authors show that SGD (gradient) updates lie in subspace spanned by the input data points. As inside any layer of neural network, the dimension of inputs are smaller than the dimension of the gradients, the gradients thus lie in a low dimensional subspace. Therefore the gradients (or gradient space) in each layer of the ANN can be represented by low dimensional basis vectors. For details see Appendix Section 6. Let, $\theta_0 \in \mathbb{R}^{C_o \times C_i \times k \times k}$ represent filters in a convolutional (Conv) layer, where C_i (C_o) is the number of input (output) channels of that layer and k is the kernel size of the filters. Following (36), θ_0 (hence gradient, $\nabla_{\theta_0}\ell$) can be reshaped into a $(C_i \times k \times k) \times C_o$ dimensional matrix. Thus gradients in a Conv layer can be described by $(C_i \times k \times k)$ dimensional space (instead of $C_o \times C_i \times k \times k$). Similarly, if $\theta_0 \in \mathbb{R}^{m \times n}$ represents a weight matrix in a fully-connected (FC) layer where m (n) is the dimension of outgoing (incoming) hidden units, the gradient space will be n dimensional (instead of $m \times n$) in this layer.

4 Continual Learning with Amphibian

Here, we describe the steps (illustrated in Figure 1) for online continual learning in Amphibian.

Learning Overview: At any time j over the learning sequence, Amphibian receives a batch of data, $\mathcal{B}_i \sim \mathcal{D}^{\tau_i}$ with N_i input-label data pairs from current task, τ_i. We aim to update the current model, θ_0^j using the update rule:

$$\theta_0^{j+1} = \theta_0^j - \Lambda^{j+1} g_{meta}(\theta_0^j). \tag{4}$$

Here g_{meta} is the gradient obtained from the meta (inner-outer loop) learning process using only the current data, $\mathcal{B}_i$. In our method, we use the gradient space formulation as described in Section 3 and consider the standard bases (e_m) of appropriate dimensions as the bases of gradient space to represent the gradients inside the neural network. We introduce a **scaling matrix**, Λ in the update rule. This is a diagonal matrix; for the convolution layer and FC layer (defined above) it is of size $(C_i \times k \times k) \times (C_i \times k \times k)$ and $n \times n$ respectively. Each diagonal element, λ_m, of this matrix is initialized with λ_m^o and then meta-learned simultaneously with g_{meta}. Over the continual learning trajectory, Λ accumulates the history of gradient alignment among observed data samples. Then it scales the meta-gradient (Equation 4) accordingly to update the model along the direction of positive gradient alignments. Thus Λ essentially *learns the learning* *rate* of the bases of the gradient space during continual learning. The number of learnable parameters in this matrix is $(C_i \times k \times k)$ for convolution layers and n for FC layers. Detailed calculations of the number of learnable gradient scales in our different experiments are given in Appendix Section 5.3. It is to be noted that the gradient (and parameters) in Equation 4 are appropriately reshaped before applying the scaled transformation. To learn shared information among tasks during meta-learning, authors in (24) introduced layer-wise transformation matrices (T) which (linearly) transform the network activations in forward pass and transform (both scale and rotate) the gradients in backward pass. In contrast, we meta-learn shared information for continual learning across tasks by Λ matrices that are strictly diagonal matrices and only scale the gradient magnitudes in the backward pass.

Meta-Learning Step-1: At first, on the given batch, $\mathcal{B}_i$ we perform k **inner-loop** updates on θ_0^j to obtain θ_k^j as:

$$U_k(\theta_0^j; \mathcal{B}_i) = \theta_k^j = \theta_0^j - \sum_{k'=0}^{k-1} \Lambda^j \nabla_{\theta_{k'}^j} \ell_{in}(\theta_{k'}^j; \mathcal{B}_i[k']). \tag{5}$$

These inner-loop steps differ in two ways from MAML inner-update step (Equation 2). First, for each inner update, we use one sample (if $k = N_i$) or a subset of samples (if $k < N_i$) from $\mathcal{B}_i$ without replacement, whereas MAML uses entire batch ($\mathcal{D}_{in}^i = \mathcal{B}_i$). Second, in our method, each inner gradient is scaled by the Λ^j matrix, whereas MAML uses a constant scalar learning rate, α. Though meta-learnable per parametric learning rate (25; 16) vector, $\boldsymbol{\alpha}$ and block diagonal preconditioners (31) have been used in such updates, we learn diagonal Λ which differs in dimensions and interpretation.

Meta-Learning Step-2: In this online learning setup, unlike MAML, Amphibian does not have access to the data from all the tasks seen so far. Moreover, as Amphibian is a rehearsal-free learner, we can not store past examples in memory and use them to approximately minimize the outer-loop MAML objective (Equation 3) as in (16). Instead, in the **outer loop** of meta-learning, we compute the meta-loss, ℓ_{out} on current data ($B_i = \mathcal{D}_{out}$) at θ_k^j and minimize the following objective :

$$\min_{\theta_0^j, \Lambda^j} \mathbb{E}_{\mathcal{B}_i}[\ell_{out}(\theta_k^j; \mathcal{B}_i)] = \min_{\theta_0^j, \Lambda^j} \mathbb{E}_{\mathcal{B}_i}[\ell_{out}(U_k(\theta_0^j, \Lambda^j; \mathcal{B}_i); \mathcal{B}_i)]. \tag{6}$$

Minimizing this objective with respect to θ_0^j is approximately equivalent to (see Appendix Section 1 for full derivation):

$$\min_{\theta_0^j} \mathbb{E}_{\mathcal{B}_i}[\ell_{out}(\theta_k^j; \mathcal{B}_i)] = \min_{\theta_0^j} \left(\ell_{out}(\theta_0^j) - \sum_{m=1}^{M} \lambda_m^j \frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} \cdot e_m e_m^T \frac{\partial \ell_{in}(\theta_0^j)}{\partial \theta_0^j} \right), \tag{7}$$

where M is the dimension of gradient space. Here, ℓ_{out} is computed on entire batch $\mathcal{B}_i$ while ℓ_{in} is computed on a sample (or subset) from $\mathcal{B}_i$. First term on the right-hand side of the objective in Equation 7 minimizes the loss on current batch of data, $\mathcal{B}_i$. The second term encourages positive alignment (inner product) of gradients computed on samples of $\mathcal{B}_i$ along selective gradient basis directions (e_m) depending on scale λ_m. For instance, for positive λ_m, inner product of data gradients along e_m is maximized whereas for zero or negative λ_m, such gradient alignments are not encouraged. Gradient of this objective is given by $g_{meta}(\theta_0^j)$ which is used for model update in Equation 4.

Scale Update Step: Next, gradient of the meta-objective (in Equation 6) with respect to each scale, λ_m^j in Λ^j can be simply expressed (using first-order approximation (14)) as follows:

$$g_{meta}(\lambda_m^j) = - \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot e_m e_m^T \sum_{k'=0}^{k-1} \frac{\partial \ell_{in}(\theta_{k'}^j)}{\partial \theta_{k'}^j} = - \ell'_{out}(\theta_k^j) \cdot e_m e_m^T \sum_{k'=0}^{k-1} \ell'_{in}(\theta_{k'}^j). \tag{8}$$

Derivation is given in Appendix Section 2. Equation 8 denotes that if outer-loop gradient $g_{out} = \ell'_{out}(\theta_k^j)$ and accumulated inner-loop gradient $\bar{g}_{in} = \sum_{k'=0}^{k-1} \ell'_{in}(\theta_{k'}^j)$ has positive inner product (aligned) along e_m, then $g_{meta}(\lambda_m^j)$ will be negative, whereas if the inner product is zero (negative) then $g_{meta}(\lambda_m^j)$ will be zero

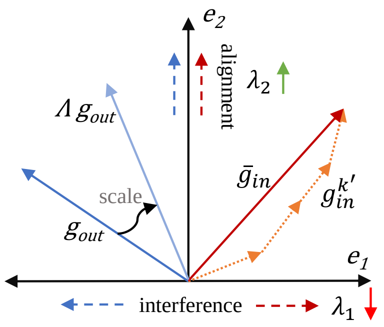

Figure 2: Conceptual illustration. Along e_2 (e_1) g_{out} and $\bar{g}_{in}$ has alignment (interference), hence λ_2 (λ_1) will increase (decrease). Considering, $g_{meta}(\theta_0^j) \approx g_{out}$, g_{meta} is scaled accordingly in model update to reflect alignment history.

(positive). The scale update rule: $\lambda_m^{j+1} = \lambda_m^j - \eta g_{meta}(\lambda_m^j)$ can be expressed as:

$$\lambda_m^{j+1} = \lambda_m^j + \eta \ell'_{out}(\theta_k^j) \cdot e_m e_m^T \sum_{k'=0}^{k-1} \ell'_{in}(\theta_{k'}^j) = \lambda_m^0 + \eta \sum_j \left(\ell'_{out}(\theta_k^j) \cdot e_m e_m^T \sum_{k'=0}^{k-1} \ell'_{in}(\theta_{k'}^j) \right), \tag{9}$$

where η is the learning rate for the scales. This update rule provides two valuable insights. **First**, the value of a scale will increase (decrease) if, along the corresponding basis, e_m direction outer- and inner-loop gradient trajectories have positive (negative) inner product or alignment (interference). Figure 2 provides a conceptual illustration to this point. Here e_1 and e_2 represent the gradient basis directions and λ_1 and λ_2 are associated scales. Since along e_2, g_{out} and $\bar{g}_{in}$ has positive inner product, λ_2 will increase. Whereas λ_1 will decrease, as along e_1 components of g_{out} and $\bar{g}_{in}$ has negative inner product. **Second**, over the entire continual learning sequence up to time j, λ_m^j accumulates the history of such gradient alignments or interferences. Now if we do not use accumulation and update the scales as $\lambda_m^{j+1} = \lambda_m^0 - \eta g_{meta}(\lambda_m^j)$ with current $g_{meta}(\lambda_m^j)$ and use these scales for model update (Equation 4), we would get fast learning on the current data but model will forget past data. In contrast, by using cumulatively updated (Equation 9) Λ in Equation 4, we ensure that the gradient step on each incoming data is accelerated along the direction of positive *cumulative* alignment, whereas blocked along the direction of zero or negative alignments thus minimizing catastrophic forgetting. Therefore, this online accumulation of gradient alignments in scale matrices over the learning sequence enables Amphibian to learn continually with minimum forgetting without any data rehearsal. An analysis of the evolution of Λ during training is given in Appendix Section 5.4 in support of this point.

Model Update Step: Finally, with $g_{meta}(\theta_0^j)$ and the updated scale Λ^{j+1} we perform model update on the current batch as in Equation 4. As the scales, λ_m can take both positive and negative values, to prevent gradient ascent (both in Equation 4 and inner loop Equation 5) we only use their positive part using $(\lambda_m)_+ = \mathbb{1}_{\lambda_m \geq 0}\lambda_m$ function, where $\mathbb{1}_{\geq 0} : \mathbb{R} \to \{0,1\}$. In the meta-learning steps, we ensure differentiability of this function using the straight-through estimator (5; 45). With the updated model θ_0^{j+1} and scale matrix Λ^{j+1}, we (continually) learn the next batch of data. The pseudocode of the algorithm is provided in Algorithm 1 with more details provided in Appendix Section 3.

5 Experimental Setup

Datasets and Models. We evaluate Amphibian and the baselines in online continual learning (OCL) (27) setups where models learn from the single pass over the data stream. We use 5 standard image classification benchmarks in continual learning: 20 tasks split **CIFAR-100** (16), 40 tasks split **TinyImagenet** (11), 25 tasks split **5-Datasets** (36), 20 tasks split **ImageNet-100** (47) and 10 tasks split **miniImageNet** (40). Similar to (16; 11), we use 5-layer network for CIFAR-100 and 5-Datasets, and 6-layer network for Tiny-Imagenet. For ImageNet-100 and miniImageNet experiments we use ResNet-18 (26) model. Details on the dataset statistics/splits, and network architectures are provided in the Appendix Section 4.1 and 4.2 respectively.

Baselines and Training. We compare Amphibian with rehearsal-free methods: EWC (20) and SI (50) which use parametric regularization; GPM (36) and SGP (37) which use gradient projection; and NCL (19) which uses both gradient projection and regularization. Although comparisons with rehearsal-free and

Algorithm 1 Amphibian Algorithm for Online Continual Learning

```
1: Inputs: $\theta$: neural network parameters (weights), $\ell_{in}$: inner-objective, $\ell_{out}$: outer (meta) objective, $\lambda^0$: initial values for all the scales, $\eta$: scale learning rate, $T$: number of tasks.
2: $j \leftarrow 0$, $\theta_0^0 \leftarrow \theta$    ▷ Initialize
3: $\Lambda^0 \leftarrow$ initialize $(\lambda^0)$    ▷ For weights/filters in each layer initialize diagonal matrix $\Lambda^0$ with $\lambda^0$
4: for $\tau_i \in 1, 2, ....., T$ do
5:     for batch, $\mathcal{B}_i \sim \mathcal{D}^{\tau_i}$ do    ▷ $\mathcal{D}^{\tau_i}$ is data stream of current task $\tau_i$
6:         $k \leftarrow$ size$(\mathcal{B}_i)$
7:         for $k' = 0$ to $k-1$ do
8:             $\theta^j_{k'+1} = \theta^j_{k'} - \Lambda^j \nabla_{\theta^j_{k'}} \ell_{in}(\theta^j_{k'}; \mathcal{B}_i[k'])$    ▷ Inner-loop updates
9:         end for
10:        $\lambda^{j+1}_m = \lambda^j_m - \eta \nabla_{\lambda^j_m} \ell_{out}(\theta^j_k; \mathcal{B}_i)$    ▷ Update scales in $\Lambda^j$
11:        $\theta^{j+1}_0 = \theta^j_0 - \Lambda^{j+1} \nabla_{\theta^j_0} \ell_{out}(\theta^j_k; \mathcal{B}_i)$    ▷ Update model parameters
12:        $j \leftarrow j + 1$
13:    end for
14: end for
```

rehearsal-based methods are not always fair (especially with large data memory), we compare with : ER (10), GEM (26), A-GEM (9), DER++ (7) and CLS-ER (3) having moderate data memory (100 to 400 samples). We also compare with MER (32) and La-MAML (16) which uses meta-learning and memory rehearsal for OCL. Following baselines, we do not use any offline pre-trained models, rather we train models from scratch. In Amphibian, scale learning rate (η) and initial scale value (λ^0_m) hyperparameters were set with grid search (as in (16)) with held out validation data from training sets. Similarly, all the hyperparameters of the baselines were tuned. Details of training setup, implementations and a list of all the hyperparameters considered in the baselines and our method are given in Appendix Section 4.

Evaluation Metrics. We measure OCL performance with two metrics: **ACC** - measures average test classification accuracy of all tasks and **BWT** (backward transfer) - measures influence of new learning on the past knowledge with negative BWT denotes forgetting. They are defined as:

$$\text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}; \quad \text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}. \tag{10}$$

Here, T is the total number of tasks and $R_{T,i}$ is the accuracy of the model on i^{th} task after learning the T^{th} task. Higher ACC in online setup signifies fast continual learning ability, however to gain better insight on the fast learning ability we introduce two additional metrics - task learning efficiency (**TLE**) and few-shot forward transfer (**FWT**$_{FS}$). TLE (15) is defined as the size of $\mathcal{D}_t$ required to achieve certain (γ%) classification accuracy on that task, t. This metric implies if less data (small TLE) is required to achieve a certain performance level, then the model can transfer knowledge faster, hence is a fast learner. For comparisons among the baselines, we set γ as 90% of the final Amphibian accuracies on each task, t. Additionally, in meta-learning (14; 30), fast learning capability is measured with *N-way K-shot* adaptation accuracy where a meta-learned model is trained on NK examples with few gradient steps and then tested. Thus a truly fast learner should also perform well under this setup. To evaluate this, after learning each OCL task, t we sample K examples from each of the N classes from the next task and fully adapt the model for n_s steps. Then record the test accuracy on this new task as FWT^t_{FS} - the few-shot forward transfer capacity of the model learned after task t. For all the tasks we measure and compare this capacity as **FWT**$_{FS}$ $= \frac{1}{T}\sum_{t=0}^{T-1} FWT^t_{FS}$.

6 Results and Analyses

6.1 Continual Learning and Fast Learning Performance Comparisons

Accuracy and Forgetting. First, we evaluate and compare the performance in task-incremental OCL setups where each task has a separate classifier and task identity is used during inference. In Table 1, we provide comparisons of ACC and BWT among various methods within this setup across 3 different

Table 1: Performance (mean ± std from 3 runs with random seeds) comparisons in online continual learning. (*) indicates results for CIFAR-100 and TinyImagenet are taken from (16).

		CIFAR-100		**TinyImagenet**		**5-Datasets**	
Rehearsal	**Methods**	ACC (%)	BWT (%)	ACC (%)	BWT (%)	ACC (%)	BWT (%)
Yes	ER*	47.8 ± 0.73	- 12.4 ± 0.83	39.3 ± 0.38	- 14.3 ± 0.89	83.4 ± 0.69	- 9.43 ± 1.24
	GEM*	48.2 ± 1.10	- 13.7 ± 0.70	40.5 ± 0.79	- 13.5 ± 0.65	86.9 ± 1.09	- 7.43 ± 0.61
	A-GEM*	46.9 ± 0.31	- 13.4 ± 1.44	38.9 ± 0.47	- 13.6 ± 1.73	82.4 ± 1.18	- 9.66 ± 0.82
	MER*	51.3 ± 1.05	- 13.4 ± 1.44	38.9 ± 0.47	- 13.6 ± 1.73	88.5 ± 0.28	- 7.00 ± 0.33
	DER++	53.4 ± 1.75	- 8.16 ± 1.12	44.1 ± 0.29	- 14.5 ± 0.49	86.7 ± 0.86	- 4.5 ± 1.09
	CLS-ER	60.5 ± 0.45	0.90 ± 0.50	51.8 ± 1.01	15.9 ± 0.71	89.1 ± 1.29	7.13 ± 0.09
	La-MAML*	61.1 ± 1.44	- 9.00 ± 0.20	52.5 ± 1.35	- 3.70 ± 1.22	89.0 ± 1.45	- 5.97 ± 2.00
No	EWC	50.4 ± 1.88	- 1.53 ± 0.98	43.6 ± 0.83	- 0.83 ± 1.17	80.1 ± 1.76	- 8.06 ± 1.22
	SI	51.1 ± 0.69	- 0.73 ± 0.24	42.3 ± 3.43	- 1.60 ± 0.14	80.9 ± 1.71	- 5.96 ± 2.45
	GPM	59.1 ± 1.26	- 0.00 ± 0.12	48.3 ± 2.69	- 0.30 ± 1.00	84.3 ± 2.57	- 2.06 ± 0.94
	SGP	61.3 ± 1.25	- 0.00 ± 0.13	52.7 ± 0.26	- 0.00 ± 0.44	86.1 ± 3.52	- 5.56 ± 3.20
	NCL	56.5 ± 1.08	- 0.00 ± 0.18	49.7 ± 0.91	- 0.83 ± 0.05	85.5 ± 0.58	- 0.00 ± 0.05
No	**Amphibian**	**65.0** ± **0.96**	- 1.30 ± 0.25	**54.8** ± **0.60**	- 0.72 ± 0.22	**89.3** ± **1.24**	- 4.87 ± 0.96

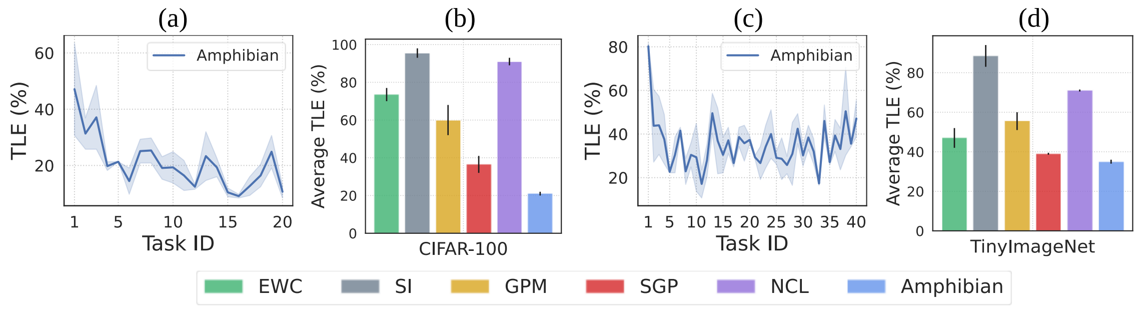

Figure 3: (a) Task learning efficiency (TLE) of Amphibian. (b) Average TLE (lower the better) of all the methods for CIFAR-100 tasks. (c) TLE of Amphibian and (d) Average TLE of all the methods for TinyImagenet.

datasets. Among the rehearsal-based methods, meta-learner La-MAML achieves better accuracy. Even without memory rehearsal, Amphibian outperforms La-MAML (by up to ∼8.5% in ACC) with less forgetting. Since the rehearsal-free methods are under-explored for OCL and Amphibian is a rehearsal-free method, in the following analyses and discussions we focus primarily on rehearsal-free methods. EWC and SI achieve similar performance, however, compared to Amphibian they severely underperform (by up to ∼14% in ACC). This shows that the parametric importance-based regularization used in these methods to ensure stability of knowledge (as indicated by small forgetting in Table 1) is not favorable for fast learning required in OCL. In contrast, gradient projection methods such as GPM and SGP achieve up to ∼10% ACC gain over EWC and SI. However, Amphibian outperforms these methods by up to ∼4% in ACC with marginally higher forgetting. This shows without any explicit constraints of orthogonal gradient projections (as in GPM and SGP), Amphibian properly *learned to scale* the gradient updates from the online data to perform better in OCL. NCL performance lies in between gradient projection and regularization-based methods and is outperformed by Amphibian. Moreover, in challenging 5-Dataset tasks where data arrives from dissimilar domains over time, Amphibian obtains the best performance over all the methods. **Next**, in Table 2, we compare Amphibian's performance with the baselines on 20 split ImageNet-100 tasks using the ResNet-18 model. Here, Amphibian archives ∼ 2.4% better accuracy, demonstrating its scalability to larger datasets and complex networks. We use a multi-glance OCL setup for CIFAR-100 and TinyImagenet experiments to ensure direct comparability with the results reported in (16). However, we used a single glance OCL setup for 5-Datasets and ImageNet-100 experiments to demonstrate Amphibian's effectiveness in this configuration as well.

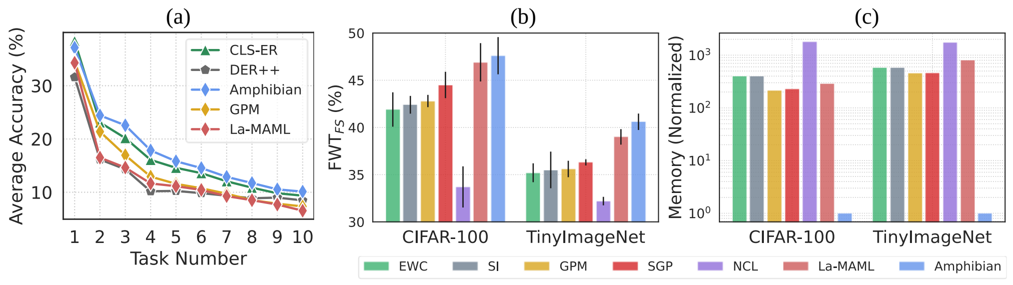

Figure 4: (a) Average accuracy for miniImageNet tasks in online class-incremental learning.(b) Few-shot forward transfer, FWT$_{FS}$ (higher the better). (c) Training time memory overhead comparisons.

Table 2: Performance comparisons for 20 IamgeNet-100 tasks on ResNet-18.

	ImageNet-100	
Methods	ACC (%)	BWT(%)
GPM	45.5 ± 1.33	- 0.00 ± 0.23
La-MAML	51.7 ± 1.15	- 5.60 ± 0.80
Amphibian	**54.1 ± 0.76**	**- 0.34 ± 0.20**

Finally, we evaluate our method in a class-incremental OCL setup. Following (40), we train a ResNet-18 model from scratch on 10 miniImageNet tasks and during inference, task identity is not used. In Figure 4(a), we compare the average accuracy of the model trained with different rehearsal-based and free methods. After 10 tasks, our method achieves better accuracy than the best rehearsal-based method - CLS-ER. However, we acknowledge that Amphibian's performance in class-incremental learning (CIL) is not as strong as in the task-incremental learning setup. Rehearsal-free CIL is very challenging, but we believe our proposed framework will serve as a solid foundation for future research in this area. All the following analyses are performed in task-incremental setups.

Task Learning Efficiency (TLE). As discussed in Section 5, a fast online continual learner should achieve high performance with less amount of data (smaller TLE per task). In Figure 3(a) we show TLE of Amphibian for each CIFAR-100 task. Averaging over all the tasks we obtain an average TLE of 21% for Amphibian. This means, on average, a new task in Amphibian can be learned to 90% of its final achievable accuracy by learning on 21% of the data from that task. In Figure 3(c) TLE of Amphibian for each TinyImageNet task is shown which gives an Average TLE of 35%. We compare the Average TLE of all the rehearsal-free methods in Figure 3(b) and (d). In both datasets, Amphibian outperforms all the other methods, indicating that Amphibian can obtain a high accuracy level on given tasks very quickly by observing fewer examples from the data stream, hence it is a fast learner.

Few-shot Forward Transfer (FWT$_{FS}$). In Figure 4(b) we compare FWT$_{FS}$ of different algorithms for CIFAR-100 and TinyImagenet datasets, where higher FWT$_{FS}$ means better few-shot (rapid) learner. For each FWT$^t_{FS}$, we use *5-way 5-shot* training data and adapt the network with $n_s = 10$ steps. Compared to the rehearsal-free methods, Amphibian achieves up to $\sim$ 5% better FWT$_{FS}$, which demonstrates the fast learning ability developed in the model from online continual learning in Amphibian. Here we show the performance of (meta-learned) La-MAML, which also achieves better performance than other rehearsal-free baselines but outperformed by Amphibian. This shows meta-learning plays a key role in building fast learning capability in Amphibian and La-MAML.

Memory overhead and Training Times. In Figure 4(c) we compare the memory overhead during training in each method for either storing the old model, important parameters and/or past data. The numbers are normalized by the memory overhead of Amphibian. For both datasets, all the baseline methods use orders of magnitude more memory than Amphibian. For instance, where Amphibian only requires up to $\sim$ 0.5% extra memory compared to network size for gradient scale (λ) storage (scale numbers given in Appendix Section 5.3), La-MAML requires 100% extra memory for per parametric learning rate storage and up to 179% extra memory for replay buffer. Among all the methods NCL has the highest memory overhead for model and projection matrix storage. Thus Amphibian most memory-efficient learner for OCL. Training time comparisons are provided in Appendix Section 5.1.

Table 3: Comparison of Amphibian with candidate rehearsal-free meta-learning methods for OCL.

	CIFAR-100			TinyImagenet		
Methods	ACC (%)	BWT(%)	FWT_{FS}(%)	ACC (%)	BWT(%)	FWT_{FS}(%)
Amphibian	65.0 ± 0.96	- 1.30 ± 0.25	47.6 ± 1.97	54.8 ± 0.60	- 0.72 ± 0.22	40.6 ± 0.87
La-MAML (No Rehearsal)	56.1 ± 0.87	- 12.8 ± 0.17	46.7 ± 1.89	49.2 ± 0.55	- 7.23 ± 0.16	39.5 ± 0.75
Online Meta-SGD	51.3 ± 2.54	- 15.7 ± 2.50	45.9 ± 1.44	45.1 ± 1.31	- 10.7 ± 0.57	37.7 ± 0.44
Online MAML	49.0 ± 1.81	- 18.0 ± 2.00	42.2 ± 1.48	40.4 ± 0.57	- 8.10 ± 0.74	31.8 ± 0.45

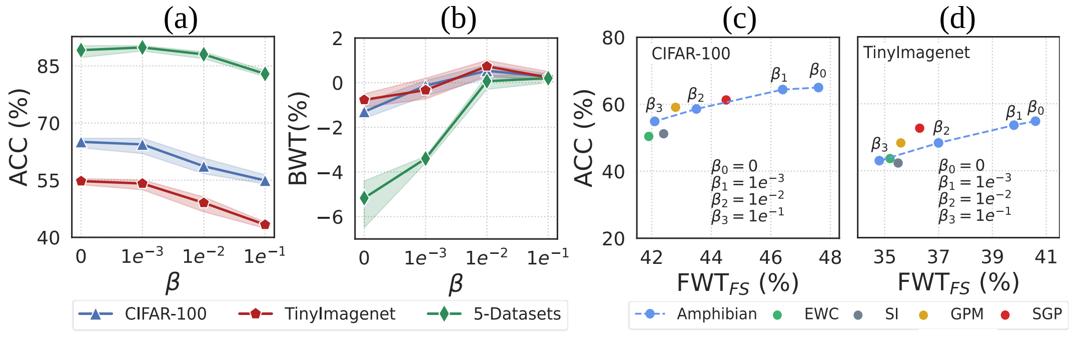

Figure 5: Variations of (a) ACC, (b) BWT, with β in Amphibian-β. ACC vs. FWT_{FS} comparisons for various Amphibian-β with rehearsal-free baselines for (c) CIFAR-100 and (d) TinyImagenet.

6.2 Amphibian vs. Online Rehearsal-free Meta-Learning

To our knowledge, there are no rehearsal-free meta-learners for OCL, so we adapt popular meta-learning approaches for OCL and compare them with Amphibian in Table 3. First method is **La-MAML(No Rehearsal)**, where we remove the memory buffer from La-MAML and compute the meta-loss with only current data (as Amphibian). The notable difference between this method and the Amphibian is that it learns *learning rates for each parameter* and uses that in inner- and outer-loop gradient updates, whereas Amphibian learns *a diagonal scale matrix at each layer* and scale the gradient directions accordingly. From Table 3, we find that La-MAML(No Rehearsal) vastly under-performs Amphibian, particularly it suffers from large forgetting (up to $\sim$ 11.5% more than Amphibian). This shows the novel meta-objective optimization and gradient scaling with the learned scale matrix (Λ) enable Amphibian to learn continually without rehearsal. In second method, we replace the meta-learned learning rate in outer-loop update from La-MAML(No Rehearsal) with a constant learning rate, whereas inner-loop still uses learnable learning rates. This converts the method to **online Meta-SGD** (25). Results show that ACC drops further, forgetting increases and fast learning capability reduces. Finally, we use constant inner- and outer-loop learning rates in online Meta-SGD, which converts the method to **online MAML** (14). With no learnable scale matrix or learning rate to encode the history of past data, this method performs the worst in OCL setup. In summary, these analyses clearly show the functional difference between our method and La-MAML and MAML, and highlight the importance of meta-learned gradient scaling in Amphibian for fast OCL performance.

6.3 Analysis of Amphibian: Amphibian-β

Here, we introduce a regularized version of Amphibian - **Amphibian-β** to understand the relationship between fast learning and forgetting. Amphibian-β uses the same online continual learning steps as Amphibian except it performs model update using the following rule:

$$\theta_0^{j+1} = \theta_0^j - \Lambda^{j+1} g_{meta}(\theta_0^j) - \beta(\theta_0^j - \theta_0^0) = \theta_0^0 - \sum_{j'=0}^{j} (1-\beta)^{j-j'} \Lambda^{j'+1} g_{meta}(\theta_0^{j'}). \tag{11}$$

So far, used j was used as the time index over the entire learning sequence (spreading across tasks). Here, we denote j as j^{th} time step at task τ_i. Thus, at the start of task τ_i, θ_0^0 would denote optimum parameter learned till task τ_{i-1}. Here, $\beta \in [0, 1]$. For $\beta = 0$ it reduces to Amphibian update whereas for $\beta = 1$ the model parameter stays at initial point θ_0^0. Thus increasing β from 0 we can regularize the Amphibian update to stay near the solution of the past tasks. This is a useful concept in continual learning as staying close to the old solution point provides a degree of protection against catastrophic forgetting. In Figure 5(a) and (b) we show how ACC and BWT vary when β is varied (from 0 to 0.1). For all datasets, increasing β reduces forgetting in Amphibian (Figure 5(b)), especially for $\beta > 1e^{-2}$ there is no forgetting. However, with increasing β, ACC degrades (Figure 5(a)). These results indicate that explicit regularization for forgetting reduction restricts the fast continual learning capacity of Amphibian. In Figure 5(c) and (d) we plot ACC vs FWT$_{FS}$ for Amphibian-β with rehearsal-free baselines for CIFAR-100 and TinyImagenet. These plots show as we increase β, both continual and fast learning performance of Amphibian becomes similar to the baselines. Such functional similarities provide valuable insight that explicit regularization or constraints used in these rehearsal-free methods primarily focus on forgetting mitigation at the expense of fast learning, hence they underperform in OCL setup. These analyses call for a rethinking of regularization or constraint design in rehearsal-free methods and provide motivation for exploring Amphibian-like learner that learns the required constraints for optimal balance between fast and continual learning.

6.4 Understating Continual Learning Dynamics in Amphibian

Continual learning works on the principle that learning a new task should not (or minimally) increase the loss of the old tasks. Amphibian does not explicitly minimize any such objectives, instead, it continually updates models in the direction of positive cumulative gradient alignments among the observed data. To understand how Amphibian enables continual learning we use loss landscape visualizations (43; 29). In Figure 6(a) we plot the loss contour of task 1 from CIFAR-100 in 2D plane. Here θ^t indicates the network model after learning task t. Here, the black line indicates the learning trajectory, with each cross point representing the projection of learned models on the plane along the trajectory. In Amphibian when we sequentially learn from task 1 to task 20, along the learning trajectory ($\theta^1 \rightarrow \theta^{10} \rightarrow \theta^{20}$ in Figure 6(a)) loss of task 1 only increase minimally from the initial point (θ^1). A similar trend can be seen for task 5 (in Figure 6(b), where along the learning trajectory ($\theta^5 \rightarrow \theta^{15} \rightarrow \theta^{20}$) loss increases in task 5 is minimal. Similar pattern is also found for TinyImagenet tasks (Appendix Section 5.2). Thus, in Amphibian, model updates along the directions with positive cumulative gradient alignments prevent a significant increase in the loss of past data enabling continual learning with minimum forgetting.

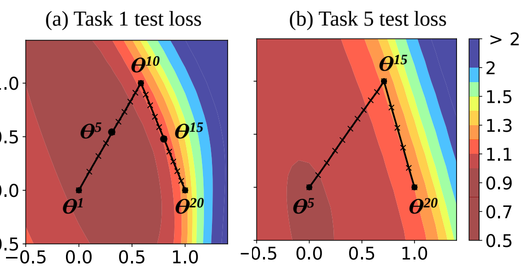

Figure 6: Dynamics of continual learning in Amphibian. Loss contour of (a) Task 1 and (b) Task 5 plotted on 2D planes defined by parameters $(\theta^1, \theta^{10}, \theta^{20})$ and $(\theta^5, \theta^{15}, \theta^{20})$ respectively.

7 Conclusions

In this paper, we introduce a rehearsal-free meta-learner - Amphibian that in a fully online manner learns to scale the gradient updates to enable fast online continual learning. To this end, Amphibian optimizes a novel meta-objective and learns scale matrices that accumulate the history of gradient alignments among the data samples observed over the learning trajectory. Using these scale matrices it updates the model in the direction of positive cumulative gradient alignments. On various continual image classification tasks, we show that such meta-learned scaled gradient update in Amphibian enables memory-efficient, data-efficient, and fast online continual learning. In conclusion, we believe Amphibian offers a unified framework for exploring meta-learning and continual learning, making it a valuable tool for dissecting inherent trade-offs and ultimately facilitating the development of improved algorithms that strike a desired balance between fast and continual learning.

Acknowledgments

This work was supported by the Center for the Co-Design of Cognitive Systems (COCOSYS), a DARPA sponsored JUMP center of Semiconductor Research Corporation (SRC), National Science Foundation and United States Department of Energy.

References

[1] Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., Page-Caccia, L.: Online continual learning with maximal interfered retrieval. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32 (2019)

[2] Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)

[3] Arani, E., Sarfraz, F., Zonooz, B.: Learning fast, learning slow: A general continual learning method based on complementary learning system. In: International Conference on Learning Representations (2022), `https://openreview.net/forum?id=uxxFrDwrE7Y`

[4] Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K.O., Clune, J., Cheney, N.: Learning to continually learn. arXiv preprint arXiv:2002.09571 (2020)

[5] Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)

[6] Bidaki, S.A., Mohammadkhah, A., Rezaee, K., Hassani, F., Eskandari, S., Salahi, M., Ghassemi, M.M.: Online continual learning: A systematic literature review of approaches, challenges, and benchmarks (2025), `https://arxiv.org/abs/2501.04897`

[7] Buzzega, P., Boschini, M., Porrello, A., Abati, D., Calderara, S.: Dark experience for general continual learning: a strong, simple baseline. In: Advances in Neural Information Processing Systems. vol. 33, pp. 15920–15930. Curran Associates, Inc. (2020)

[8] Caccia, M., Rodriguez, P., Ostapenko, O., Normandin, F., Lin, M., Page-Caccia, L., Laradji, I.H., Rish, I., Lacoste, A., Vázquez, D., et al.: Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. Advances in Neural Information Processing Systems **33**, 16532–16545 (2020)

[9] Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with A-GEM. In: International Conference on Learning Representations (2019)

[10] Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H.S., Ranzato, M.: Continual learning with tiny episodic memories. ArXiv **abs/1902.10486** (2019)

[11] Deng, D., Chen, G., Hao, J., Wang, Q., Heng, P.A.: Flattening sharpness for dynamic gradient projection memory benefits continual learning. Advances in Neural Information Processing Systems **34**, 18710–18721 (2021)

[12] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)

[13] Farajtabar, M., Azizan, N., Mott, A., Li, A.: Orthogonal gradient descent for continual learning. In: International Conference on Artificial Intelligence and Statistics. pp. 3762–3773. PMLR (2020)

[14] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International conference on machine learning. pp. 1126–1135. PMLR (2017)

[15] Finn, C., Rajeswaran, A., Kakade, S., Levine, S.: Online meta-learning. In: International Conference on Machine Learning. pp. 1920–1930. PMLR (2019)

[16] Gupta, G., Yadav, K., Paull, L.: Look-ahead meta learning for continual learning. Advances in Neural Information Processing Systems **33**, 11588–11598 (2020)

[17] Hadsell, R., Rao, D., Rusu, A.A., Pascanu, R.: Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences **24**(12), 1028–1040 (2020)

[18] Javed, K., White, M.: Meta-learning representations for continual learning. Advances in neural information processing systems **32** (2019)

[19] Kao, T.C., Jensen, K., van de Ven, G., Bernacchia, A., Hennequin, G.: Natural continual learning: success is a journey, not (just) a destination. Advances in Neural Information Processing Systems **34**, 28067–28079 (2021)

[20] Kirkpatrick, J., Pascanu, R., Rabinowitz, N.C., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences **114**, 3521 – 3526 (2017)

[21] Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)

[22] Le, Y., Yang, X.S.: Tiny imagenet visual recognition challenge (2015)

[23] Lee, E., Huang, C.H., Lee, C.Y.: Few-shot and continual learning with attentive independent mechanisms. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9455–9464 (2021)

[24] Lee, Y., Choi, S.: Gradient-based meta-learning with learned layerwise metric and subspace. In: International conference on machine learning. pp. 2927–2936. PMLR (2018)

[25] Li, Z., Zhou, F., Chen, F., Li, H.: Meta-sgd: Learning to learn quickly for few-shot learning. arXiv preprint arXiv:1707.09835 (2017)

[26] Lopez-Paz, D., Ranzato, M.A.: Gradient episodic memory for continual learning. In: Advances in Neural Information Processing Systems. vol. 30 (2017)

[27] Mai, Z., Li, R., Jeong, J., Quispe, D., Kim, H., Sanner, S.: Online continual learning in image classification: An empirical survey. Neurocomputing **469**, 28–51 (2022)

[28] Mccloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. The Psychology of Learning and Motivation **24**, 104–169 (1989)

[29] Mirzadeh, S.I., Farajtabar, M., Gorur, D., Pascanu, R., Ghasemzadeh, H.: Linear mode connectivity in multitask and continual learning. arXiv preprint arXiv:2010.04495 (2020)

[30] Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. arXiv preprint arXiv:1803.02999 (2018)

[31] Park, E., Oliva, J.B.: Meta-curvature. Advances in Neural Information Processing Systems **32** (2019)

[32] Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., Tesauro, G.: Learning to learn without forgetting by maximizing transfer and minimizing interference. In: International Conference on Learning Representations (2019)

[33] Ring, M.B.: Child: A first step towards continual learning. In: Learning to Learn (1998)

[34] Robins, A.V.: Catastrophic forgetting, rehearsal and pseudorehearsal. Connect. Sci. **7**, 123–146 (1995)

[35] Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. ArXiv **abs/1606.04671** (2016)

[36] Saha, G., Garg, I., Roy, K.: Gradient projection memory for continual learning. In: International Conference on Learning Representations (2021), `https://openreview.net/forum?id=3AOj0RCNC2`

[37] Saha, G., Roy, K.: Continual learning with scaled gradient projection. Proceedings of the AAAI Conference on Artificial Intelligence **37**(8), 9677–9685 (Jun 2023). https://doi.org/10.1609/aaai.v37i8.26157

[38] Saha, G., Roy, K.: Saliency guided experience packing for replay in continual learning. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). pp. 5273–5283 (January 2023)

[39] Serrà, J., Surís, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4548–4557. PMLR (10–15 Jul 2018)

[40] Shim, D., Mai, Z., Jeong, J., Sanner, S., Kim, H., Jang, J.: Online class-incremental continual learning with adversarial shapley value. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 9630–9638 (2021)

[41] Son, J., Lee, S., Kim, G.: When meta-learning meets online and continual learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **47**(1), 413–432 (2025). https://doi.org/10.1109/TPAMI.2024.3463709

[42] Thrun, S., Pratt, L.: Learning to learn. Springer Science & Business Media (2012)

[43] Verwimp, E., De Lange, M., Tuytelaars, T.: Rehearsal revealed: The limits and merits of revisiting samples in continual learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9385–9394 (2021)

[44] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. Advances in neural information processing systems **29** (2016)

[45] Von Oswald, J., Zhao, D., Kobayashi, S., Schug, S., Caccia, M., Zucchet, N., Sacramento, J.: Learning where to learn: Gradient sparsity in meta and continual learning. Advances in Neural Information Processing Systems **34**, 5250–5263 (2021)

[46] Wickramasinghe, B., Saha, G., Roy, K.: Continual learning: A review of techniques, challenges and future directions. IEEE Transactions on Artificial Intelligence pp. 1–21 (2023). https://doi.org/10.1109/TAI.2023.3339091

[47] Yan, S., Xie, J., He, X.: Der: Dynamically expandable representation for class incremental learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3014–3023 (2021)

[48] Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: 6th International Conference on Learning Representations (2018)

[49] Zeng, G., Chen, Y., Cui, B., Yu, S.: Continual learning of context-dependent processing in neural networks. Nature Machine Intelligence **1**(8), 364–372 (2019)

[50] Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3987–3995. PMLR (06–11 Aug 2017)

[51] Zhang, Y., Pfahringer, B., Frank, E., Bifet, A., Lim, N.J.S., Jia, Y.: A simple but strong baseline for online continual learning: Repeated augmented rehearsal. arXiv preprint arXiv:2209.13917 (2022)

Appendix

Derivation of Amphibian meta-objective and meta-gradients are provided in Section 1 and 2 respectively. A detailed explanation of the Amphibian algorithm is given in Section 3. Experimental details including dataset statistics, network architectures, a list of hyperparameters along with implementation details are provided in Section 4. Additional results and analyses are provided in Section 5.

1 Derivation of Amphibian Meta-Objective

In this section, we will show that when we optimize the following meta-objective in Amphibian:

$$\min_{\theta_0^j} \mathbb{E}_{\mathcal{B}_i}[\ell_{out}(\theta_k^j; \mathcal{B}_i)] = \min_{\theta_0^j} \mathbb{E}_{\mathcal{B}_i}[\ell_{out}(U_k(\theta_0^j, \Lambda^j; \mathcal{B}_i); \mathcal{B}_i)]. \tag{A.1}$$

where each of the k inner-update is taken using a sample (or subset of samples) from current batch, $\mathcal{B}_i$ from task τ_i and the meta-loss, ℓ_{out} is computed on the entire current batch data, $\mathcal{B}_i$, it is equivalent to minimizing the following objective:

$$\min_{\theta_0^j} \left(\ell_{out}(\theta_0^j) - \sum_{m=1}^{M} \lambda_m^j \frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} \cdot e_m e_m^T \frac{\partial \ell_{in}(\theta_0^j)}{\partial \theta_0^j} \right). \tag{A.2}$$

For that, let us define,

$$g_k = \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \text{ (gradient of meta-loss at } \theta_k^j) \tag{A.3}$$

$$\bar{g}_k = \frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} \text{ (gradient of meta-loss at } \theta_0^j) \tag{A.4}$$

$$g_{k'} = \frac{\partial \ell_{in}(\theta_{k'}^j)}{\partial \theta_{k'}^j} \text{ (gradient of inner-loss at } \theta_{k'}^j\text{, where } k' < k) \tag{A.5}$$

$$\bar{g}_{k'} = \frac{\partial \ell_{in}(\theta_0^j)}{\partial \theta_0^j} \text{ (gradient of inner-loss at } \theta_0^j\text{, where } k' < k) \tag{A.6}$$

$$\theta_{k'+1}^j = \theta_{k'}^j - \Lambda^j g_{k'} \text{ (sequence of parameter vectors)} \tag{A.7}$$

$$\bar{H}_k = \ell''_{out}(\theta_0^j) \text{ (Hessian of meta-loss at } \theta_0^j \text{)} \tag{A.8}$$

$$\bar{H}_{k'} = \ell''_{in}(\theta_0^j) \text{ (Hessian of inner-loss at } \theta_0^j \text{)} \tag{A.9}$$

First, let's write the gradient of meta-loss (outer-loop loss) at θ_k^j from Equation A.3 as (using Taylor's expansion (30)):

$$\begin{aligned}
g_k &= \ell'_{out}(\theta_k^j) = \ell'_{out}(\theta_0^j) + \ell''_{out}(\theta_0^j)(\theta_k^j - \theta_0^j) + O(||\theta_k^j - \theta_0^j||^2) \\
&\quad (\,' \text{ implies derivative w.r.t argument}) \\
&= \bar{g}_k + \bar{H}_k(\theta_k^j - \theta_0^j) + O(||\theta_k^j - \theta_0^j||^2) \quad \text{(using definition of } \bar{g}_k, \bar{H}_k) \\
&= \bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \Lambda^j g_{k'} + O(\Lambda^2) \quad \text{(using } \theta_k^j - \theta_0^j = -\sum_{k'=0}^{k-1} \Lambda^j g_{k'}\text{, from Equation 5)} \\
&= \bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \Lambda^j \bar{g}_{k'} + O(\Lambda^2) \quad \text{(using } g_{k'} = \bar{g}_{k'} + O(\Lambda) \text{)} \\
&= \bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \bar{g}_{k'}^{\Lambda} + O(\Lambda^2) \quad \text{(let scaled update, } \bar{g}_{k'}^{\Lambda} = \Lambda^j \bar{g}_{k'} \text{)}
\end{aligned} \tag{A.10}$$

Now, let's derive the meta-gradient (or MAML gradient (14)) for parameters θ_0^j, denoted as $g_{meta}(\theta_0^j)$:

$$
\begin{aligned}
g_{meta}(\theta_0^j) &= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_0^j} = \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j}\frac{\partial U(\theta_{k-1}^j)}{\partial \theta_0^j} \\
&= g_k \frac{\partial U(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \dots \frac{\partial U(\theta_0^j)}{\partial \theta_0^j} \\
&\text{(repeatedly applying chain rule and using, } \theta_k^j = U(\theta_{k-1}^j) \text{)} \\
&= \prod_{k'=0}^{k-1} \left(\frac{\partial}{\partial \theta_{k'}^j} (\theta_{k'}^j - \Lambda^j g_{k'}) \right) g_k \\
&= \prod_{k'=0}^{k-1} \left(\frac{\partial}{\partial \theta_{k'}^j} (\theta_{k'}^j - g_{k'}^\Lambda) \right) g_k \\
&= \prod_{k'=0}^{k-1} \left(I - H_{k'}^\Lambda \right) g_k \quad \text{(where, } H_{k'}^\Lambda \text{ is Hessian of scaled gradient, } g_{k'}^\Lambda \text{ at } \theta_{k'}^j\text{)}
\end{aligned}
\tag{A.11}
$$

Using Taylor's theorem and dropping higher order ($O(\Lambda^2)$) terms (30), we can write $H_{k'}^\Lambda \approx \bar{H}_{k'}^\Lambda$ and then using g_k from Equation A.10 in Equation A.11 we get:

$$
\begin{aligned}
g_{meta}(\theta_0^j) &= \left(\prod_{k'=0}^{k-1} (I - \bar{H}_{k'}^\Lambda) \right) \left(\bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \bar{g}_{k'}^\Lambda \right) + O(\Lambda^2) \\
&= \left(I - \sum_{k'=0}^{k-1} \bar{H}_{k'}^\Lambda) \right) \left(\bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \bar{g}_{k'}^\Lambda \right) + O(\Lambda^2) \\
&= \bar{g}_k - \sum_{k'=0}^{k-1} \bar{H}_{k'}^\Lambda \bar{g}_k - \bar{H}_k \sum_{k'=0}^{k-1} \bar{g}_{k'}^\Lambda + O(\Lambda^2)
\end{aligned}
\tag{A.12}
$$

Now, using $k = 1$ in Equation A.12 we can derive the equivalent objective in Equation A.2. For higher k, the form of objective becomes complicated but has a similar set of terms. Thus putting $k = 1$ in Equation A.12:

$$
\begin{aligned}
\frac{\partial \ell_{out}(\theta_k^j)}{\theta_0^j} &= g_{meta}(\theta_0^j) = \bar{g}_1 - \bar{H}_0^\Lambda \bar{g}_1 - \bar{H}_1 \bar{g}_0^\Lambda + O(\Lambda^2) \\
&= \bar{g}_1 - \frac{\partial}{\partial \theta_0^j}(\bar{g}_1 \cdot \bar{g}_0^\Lambda) \quad \left(\text{using } \frac{\partial}{\partial \theta_0^j}(\bar{g}_1 \cdot \bar{g}_0^\Lambda) = \bar{H}_0^\Lambda \bar{g}_1 + \bar{H}_1 \bar{g}_0^\Lambda\right) \\
&= \frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} - \frac{\partial}{\partial \theta_0^j} \left(\frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} \cdot \Lambda^j \frac{\partial \ell_{in}(\theta_0^j)}{\partial \theta_0^j} \right) \\
&\text{(expressing terms as derivatives)} \\
&= \frac{\partial}{\partial \theta_0^j} \left(\ell_{out}(\theta_0^j) - \sum_{m=1}^{M} \lambda_m^j \frac{\partial \ell_{out}(\theta_0^j)}{\partial \theta_0^j} \cdot e_m e_m^T \frac{\partial \ell_{in}(\theta_0^j)}{\partial \theta_0^j} \right),
\end{aligned}
\tag{A.13}
$$

which is precisely the gradient of the Amphibian meta-objective in Equation A.2.

2 Derivation of Meta-Gradients

In this section we first derive the **meta-gradients of scales** λ_m^j which is defined as:

$$\begin{aligned}
g_{meta}(\lambda_m^j) &= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \lambda_m^j} = \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \frac{\partial \theta_k^j}{\partial \lambda_m^j} \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \frac{\partial}{\partial \lambda_m^j} \Big(U(\theta_{k-1}^j) \Big) \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \frac{\partial}{\partial \lambda_m^j} \left(\theta_{k-1}^j - (\lambda_m^j)_+ e_m e_m^T \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \right) \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \left(\frac{\partial}{\partial \lambda_m^j} \theta_{k-1}^j - \frac{\partial}{\partial \lambda_m^j} \left((\lambda_m^j)_+ e_m e_m^T \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \right) \right) \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \left(\frac{\partial}{\partial \lambda_m^j} \theta_{k-1}^j - e_m e_m^T \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \frac{\partial (\lambda_m^j)_+}{\partial \lambda_m^j} \right) \\
&\quad \text{(taking } \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \text{ as constant w.r.t } \lambda_m^j \text{ to get the first-order MAML} \\
&\quad \text{approximation as in (30; 16))} \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \left(\frac{\partial}{\partial \lambda_m^j} \theta_{k-1}^j - e_m e_m^T \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \right) \\
&\quad \text{(} \frac{\partial (\lambda_m^j)_+}{\partial \lambda_m^j} \text{ is equal to identity using approximations from straight-through} \\
&\quad \text{estimation as in (5; 45))} \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \left(\frac{\partial}{\partial \lambda_m^j} U(\theta_{k-2}^j) - e_m e_m^T \frac{\partial \ell_{in}(\theta_{k-1}^j)}{\partial \theta_{k-1}^j} \right) \\
&= \frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot \left(\frac{\partial}{\partial \lambda_m^j} \theta_0^j - e_m e_m^T \sum_{k'=0}^{k-1} \frac{\partial \ell_{in}(\theta_{k'}^j)}{\partial \theta_{k'}^j} \right) \\
&\quad \text{(repeatedly expanding and differentiating the update function } U(.)\text{)} \\
g_{meta}(\lambda_m^j) &= -\frac{\partial \ell_{out}(\theta_k^j)}{\partial \theta_k^j} \cdot e_m e_m^T \sum_{k'=0}^{k-1} \frac{\partial \ell_{in}(\theta_{k'}^j)}{\partial \theta_{k'}^j} \\
&\quad \text{(assuming initial parameters, } \theta_0^j \text{ at time } j \text{ is constant w.r.t } \lambda_m^j\text{)}
\end{aligned} \tag{A.14}$$

This meta-gradient is used in Equation 9 (main paper) for scale updates.

Next to obtain the **meta-gradients of (weight) parameters**, θ_0^j lets recall Equation A.11:

$$g_{meta}(\theta_0^j) = \prod_{k'=0}^{k-1} \left(I - H_{k'}^{\Lambda} \right) g_k = \prod_{k'=0}^{k-1} \left(\frac{\partial}{\partial \theta_{k'}^j} (\theta_{k'}^j - \Lambda^j \frac{\partial \ell_{in}(\theta_{k'}^j)}{\partial \theta_{k'}^j}) \right) g_k \tag{A.15}$$

Setting all the first-order terms as constant in the right-hand side of this equation (to ignore the second-order derivatives), we get the first-order approximation of the meta-gradient as:

$$g_{meta}^{FO}(\theta_0^j) = g_k = \ell'_{out}(\theta_k^j), \tag{A.16}$$

which is used in model updates in Equation 4 (main paper). This approximation drastically reduces memory consumption while preserving the performance. This allows scaling of Amphibian to the larger networks with complex datasets (Table 2 (main paper)).

3 Amphibian Algorithm

The pseudocode of the Amphibian algorithm is provided in Algorithm 1 in the main paper. For the given online batch of data, $\mathcal{B}_i$, Amphibian performs k inner-loop updates, where in each update it uses a sample (or a subset of samples) from $\mathcal{B}_i$ (Line 8). On the final parameters obtained after inner-updates, θ_k^j, it evaluates meta-loss on the entire batch, $\mathcal{B}_i$ to obtain meta-gradients for scales, $\nabla_{\lambda_m^j} \ell_{out}(\theta_k^j; \mathcal{B}_i)$ and weights, $\nabla_{\theta_0^j} \ell_{out}(\theta_k^j; \mathcal{B}_i)$. With this meta-gradients, Ampibian first update the scales (Line 10) and then model weights (Line 11). In inner-updates (Line 8) and outer-update (Line 11), to avoid gradient ascent, only positive parts of the scales are used using: $(\lambda_m)_+ = \mathbb{1}_{\lambda_m \geq 0} \lambda_m$ function, where $\mathbb{1}_{\cdot \geq 0} : \mathbb{R} \rightarrow \{0, 1\}$. For bias parameters in each layer, we use $\boldsymbol{\lambda}$ vectors instead of diagonal Λ matrices in inner-updates (Line 8) and outer-update (line 11). Each element of $\boldsymbol{\lambda}$, learns the learning rate of the corresponding bias parameter which is updated using the similar update step in Line 10.

4 Experimental Details

4.1 Dataset Splits and Statistics

Split CIFAR-100 has 20 tasks each having 5 distinct classes from CIFAR-100 (21). Split TinyImagenet has 40 tasks where each task has 5 distinct classes from TinyImagenet-200 (22). Finally, we use a sequence of 5-Datasets including CIFAR-10, MNIST, SVHN, Fashion MNIST, and notMNIST where each dataset is split into five tasks (each having a 2 classes) to obtain a total of 25 tasks in the split 5-Datasets sequence. Dataset statistics used in these experiments are given in Table A.1 and A.2. Split miniImageNet (40) consists of splitting the miniImageNet dataset (44) into 10 disjoint tasks, where each task contains 10 classes. Here each image is of size $3 \times 84 \times 84$. ImageNet-100 is built by selecting 100 classes from the ImageNet-1k (12) dataset. Split ImageNet-100, which is used in our experiment, consists of splitting the ImageNet-100 into 20 disjoint tasks, where each task contains 5 classes. Here each image is of size $3 \times 224 \times 224$. For both split miniImageNet and ImageNet-100, 2% training data from each task is kept aside as validation sets.

Table A.1: Dataset Statistics. 10% training data from each task is kept aside as validation sets.

	Split CIFAR-100	**Split Tinyimagenet**	**Split 5-Datasets**
num. of tasks	20	40	25
input size	$3 \times 32 \times 32$	$3 \times 64 \times 64$	$3 \times 32 \times 32$
# Classes/task	5	5	2
# Training samples/tasks	2,250	2,250	See Table A.2
# Validation Samples/tasks	250	250	See Table A.2
# Test samples/tasks	500	250	See Table A.2

Table A.2: 5-Datasets statistics (36). For the datasets with monochromatic images, we replicate the image across all RGB channels so that size of each image becomes $3 \times 32 \times 32$. In split 5-Datasets, each dataset (in this table) is split into 5 tasks, each with 2 disjoint classes. 10% training data from each task is kept aside as validation sets.

	CIFAR-10	**MNIST**	**SVHN**	**Fashion MNIST**	**notMNIST**
# Classes	10	10	10	10	10
# Training samples	45,000	54,000	65,931	54,000	15,167
# Validation Samples	5,000	6,000	7,325	6,000	1,685
# Test samples	10,000	10,000	26,032	10,000	1,873

4.2 Network Architecture Details

For split CIFAR-100 experiments, similar to La-MAML (16), we used a 5-layer neural network with 3 convolutional layers each having 160 filters with 3×3 kernels, followed by two fully connected layers having

320 units each. For split 5-Datasets, we used a 5-layer neural network with 3 convolutional layers each having 200 filters with 3×3 kernels, followed by two fully connected layers having 400 units each. For split TinyImagenet experiments, similar to La-MAML (16), we used a 6-layer neural network with 4 convolutional layers each having 160 filters with 3×3 kernels, followed by two fully connected layers having 640 units each. For ImageNet-100 and miniImageNet experiments, we have used ResNet-18 model. This network consists of a front convolutional layer followed by 4 residual blocks each having four convolutional layers followed by a classifier layer. We used 40 filters in front convolutional layer and in the first residual block layers. For second, third and fourth residual blocks we used 80, 120 and 160 filters respectively. For ImageNet-100, in the front convolution layer, we used convolution with 7×7 kernel with stride 5. For miniImageNet, in the front convolution layer, we used convolution with 3×3 kernel with stride 2. For both of these cases, we used 2×2 average-pooling with stride 1 before the classifier layer. All the networks use ReLU in the hidden units and softmax with cross-entropy loss in the final layer.

4.3 List of Hyperparameters

A list of hyperparameters in our method and baseline approaches is given in Table A.3. As in (16), hyper-parameter for all the approaches are tuned by performing a grid-search using validation sets. For all the experiments, except split ImageNet-100, a batch size of 10 was used for training. In split ImageNet-100 experiments a batch size of 25 samples was used.

4.4 Baseline Implementations

For rehearsal-based methods - ER, GEM, A-GEM, MER and La-MAML, we used the implementation provided in La-MAML (16). CLS-ER and DER++ are implemented by adapting the codes by (3). EWC (20), SI (50) and NCL (19) are implemented adapting the codes from[1]. GPM (36) and SGP (37) are implemented using the respective official open-sourced code repositories.

4.5 Amphibian Implementation: Software, Hardware and Code

We implemented Amphibian in `python (version 3.7.6)` with `pytorch (version 1.5.1)` and `torchvision (version 0.6.1)` libraries.We ran the codes on a single NVIDIA TITAN Xp GPU (CUDA `version 12.1`) and reported the results in the paper.

[1]https://github.com/GMvandeVen/continual-learning (MIT License)

Table A.3: Hyperparameters grid considered for the baselines and Amphibian. The best values are given in parentheses. Here, 'lr' represents the learning rate. All the methods use SGD optimizer unless otherwise stated. The number of epochs in the OCL setup for all methods is 1. To maximally utilize the current batch of data in OCL, each method has a hyperparameter called glances (16; 51) which indicates the number of gradient updates or meta-updates made on each of these batches. In the table we represent Split CIFAR-100 as 'cifar', Split TinyImagenet as 'tinyimg' and Split 5-Datasets as '5data'.

Methods	**Hyperparameters**
ER	lr : 0.01 (5data), 0.03 (cifar), 0.1 (tinyimg); glances : 1 (5data), 10 (cifar, tinyimg) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
GEM	lr : 0.01 (5data), 0.03 (cifar, tinyimg); glances : 1 (5data), 2 (cifar, tinyimg) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
A-GEM	lr : 0.01 (tinyimg, 5data), 0.03 (cifar); glances : 1 (5data), 2 (cifar, tinyimg) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
MER	lr (α) : 0.05 (5data), 0.1 (cifar, tinyimg) lr (β) : 0.1 (cifar, tinyimg, 5data) lr (γ) : 1.0 (cifar, tinyimg, 5data) glances : 1 (5data), 10 (cifar, tinyimg) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
DER++	lr : 0.01 (5data), 0.03 (cifar), 0.1 (tinyimg) ; glances : 1 (5data), 2 (tinyimg), 10 (cifar) α: 0.1 (tinyimg), 0.2 (cifar, 5data) ; β: 0.5 (cifar, tinyimg), 1.0 (5data) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
CLS-ER	lr : 0.01, 0.03, 0.05 (cifar, tinyimg, 5data) ; glances : 1 (5data), 2 (tinyimg), 10 (cifar) r_s: r_p: 0.3 (5data), 0.5 (cifar), 0.9 (tinyimg) ; r_p: 0.5 (cifar), 0.8 (tinyimg), 1.0 (5data) λ: 0.1 (cifar, tinyimg, 5data), 0.15 memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
La-MAML	α_0 : 0.1 (cifar, tinyimg, 5data) lr (η) : 0.25 (5data), 0.3 (cifar, tinyimg) glances : 1 (5data), 2 (tinyimg), 10 (cifar) memory size (data samples) : 100 (5data), 200 (cifar), 400 (tinyimg)
EWC	lr : 0.1 (cifar, tinyimg, 5data) regularization coefficient, λ : 1e^2, 1e^3 (cifar), 1e^4, 2e^4 (5data), 1e^5 (tinyimg) glances : 1 (5data), 2 (tinyimg), 5 (cifar)
SI	optimizer : Adam (cifar, tinyimg, 5data) lr : 1e^{-3} (cifar, tinyimg, 5data) regularization coefficient, c : 1, 50 (tinyimag, 5data), 100 (cifar), 1000 glances : 1 (5data), 2 (tinyimg), 5 (cifar)
GPM	lr : 0.05, 0.1 (cifar, tinyimg, 5data) ϵ_{th} : 0.96 (tinyimg), 0.975 (5data), 0.98 (cifar) ϵ_{th} (increment/task) : 0.001 (cifar, tinyimg, 5data) n_s : 120 (cifar, tinyimg, 5data) glances : 1 (5data), 2 (tinyimg), 5 (cifar)
SGP	lr : 0.05, 0.1 (cifar, tinyimg, 5data) ϵ_{th} : 0.96 (tinyimg), 0.975 (5data), 0.98 (cifar) ϵ_{th} (increment/task) : 0.001 (cifar, tinyimg, 5data) scale coefficient (α) : 1 (5data), 5 (cifar), 10 (tinyimg) n_s : 120 (cifar, tinyimg, 5data) glances : 1 (5data), 2 (tinyimg), 5 (cifar)
NCL	lr : 0.05, 0.1 (cifar, 5data), 0.2 (tinyimg) p_w^{-2} : 2250 (cifar, tinyimg), 9000 (5data) glances : 1 (5data), 2 (tinyimg), 5 (cifar)
Amphibian	λ^0 : 0.1, 0.25 (tinyimg), 0.5 (cifar, 5data) lr (η) : 0.25, 0.5 (tinyimg), 1.0 (cifar, 5data) glances : 1 (5data), 2 (tinyimg), 5 (cifar)

5 Additional Results

5.1 Memory Overhead and Training Time

In Figure A.1(a) we show memory overhead comparisons during training for 5-Datasets tasks. In this case also, we observe that other baseline methods have orders of magnitude more memory overhead than Amphibian. Wall-clock training time comparisons among different methods for all three datasets are shown in Figure A.1(b). Training times for all the tasks in the continual learning sequence for different experiments are measured on a single NVIDIA TITAN Xp GPU. As Amphibian uses inner-and outer-loop meta-learning steps, it requires more wall clock time for each model update during training compared to the other rehearsal-free baselines. However, other rehearsal-based meta-learners such as La-MAML and MER take up to ∼2.7× and ∼70× more training time than Amphibian.

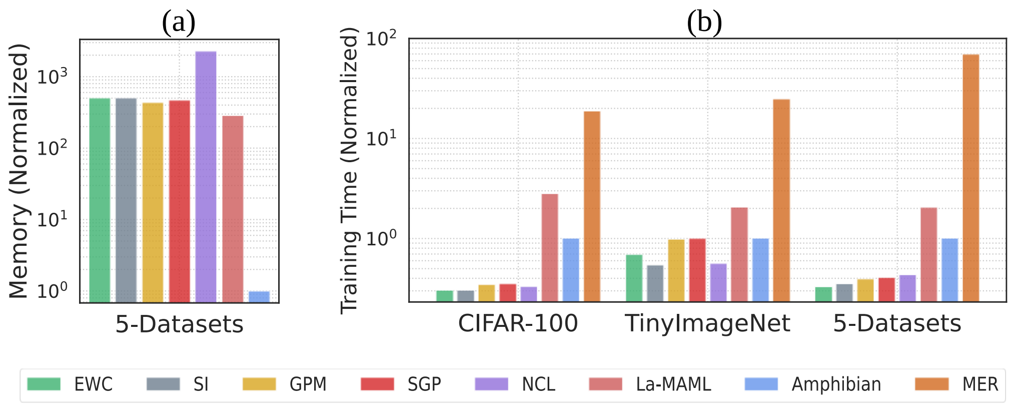

Figure A.1: (a) Comparison of memory overhead (normalized by the Amphibian memory overhead) during training for 5-Datasets experiments. (b) Wall-clock training time comparisons for sequential training of all the tasks in different datasets. Normalized with respect to the time taken by Amphibian.

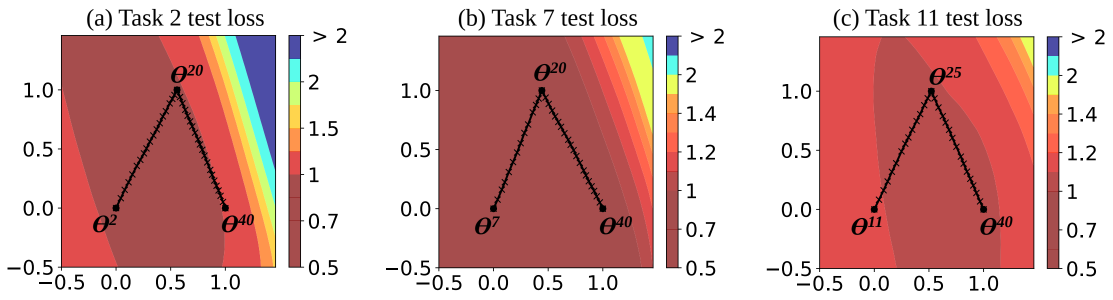

Figure A.2: Dynamics of continual learning in Amphibian. Loss contours of (a) Task 2, (b) Task 7, and (c) Task 11 from split TinyImagenet dataset are plotted on 2D planes defined by parameters $(\theta^2, \theta^{20}, \theta^{40})$, $(\theta^7, \theta^{20}, \theta^{40})$ and $(\theta^{11}, \theta^{25}, \theta^{40})$ respectively. Black lines indicate learning trajectories.

5.2 Continual Learning Dynamics in Amphibian: A Loss Landscape View

Loss Contour Plots. We used visualization tools developed in (29; 43) to plot the loss contours (in Figure 6 (main paper) and A.2) on 2D planes defined by model parameters (θ^t). Each of these hyperplanes in the parameter space is defined by three points θ^1, θ^2 and θ^3. Orthogonalizing $\theta^2 - \theta^1$ and $\theta^3 - \theta^1$ gives a two dimensional coordinate system with base vectors u and v. The value at point (x, y) is then calculated as the loss of a model with parameters $\theta^1 + u \cdot x + v \cdot y$. Please see the code/appendix in (29) for more details.

Continual Learning Dynamics in Amphibian. In Figure 6 (main paper) we showed that Amphibian updates incur minimum to no increase in losses of the past tasks for split CIFAR-100 tasks. Here, in Figure A.2, we show the loss contours for three split TinyImagenet tasks. In Figure A.2(a) we plot the loss contour of task 2 from TinyImagenet in a 2D plane. This figure shows when we sequentially learn from task 2 to task 40, along the entire learning trajectory ($\theta^2 \rightarrow \theta^{20} \rightarrow \theta^{40}$) loss of task 2 only increases minimally from the initial point (θ^2). A similar trend is also observed for other tasks. Here we show such trends for task 7 (Figure A.2(b)) and task 11 (Figure A.2(c)).

Table A.4: Number of meta-learnable gradient scales (in diagonal matrix, Λ) in each layer in Amphibian.

Network	**Layer**	**Size of Filters / Weights** ($C_o \times C_i \times k \times k$) / ($m \times n$)	**Number of Scales** (in Scale Matrix, Λ)
5-layer Network	Conv1	$160 \times 3 \times 3 \times 3$	27
(CIFAR-100)	Conv2	$160 \times 160 \times 3 \times 3$	1400
	Conv3	$160 \times 160 \times 3 \times 3$	1400
	FC1	320×2560	2560
	FC2	320×320	320
6-layer Network	Conv1	$160 \times 3 \times 3 \times 3$	27
(TinyImagenet)	Conv2	$160 \times 160 \times 3 \times 3$	1400
	Conv3	$160 \times 160 \times 3 \times 3$	1400
	Conv4	$160 \times 160 \times 3 \times 3$	1400
	FC1	640×2560	2560
	FC2	640×640	640
5-layer Network	Conv1	$200 \times 3 \times 3 \times 3$	27
(5-Datasets)	Conv2	$200 \times 200 \times 3 \times 3$	1800
	Conv3	$200 \times 200 \times 3 \times 3$	1800
	FC1	400×3200	3200
	FC2	400×400	400

5.3 Number of Gradient Scales in Amphibian

As we adopt low-dimensional gradient space representation (as discussed in Section 3 (main paper)) for gradients of weights/filters in the neural network, the number of learnable scales for gradient bases (in scale matrix, Λ) in Amphibian is very small compared to the size of the weights/filters. Table A.4 shows the number of meta-learnable scales for each layer. Such a small number of scales explains the extremely low memory overhead of Amphibian during training as shown in Figure 4(c) (main paper) and Figure A.1(a).

5.4 Analysis of Λ During Training

To demonstrate the inner workings of the proposed algorithm, in Figure A.3 we plot the layer-wise evolution of the Λ during training of 20 CIFAR-100 tasks. For each layer, we show a box plot of λ values contained the diagonal matrix Λ. In Figure A.3(a)-(e), the horizontal red dotted line indicates 0 value, whereas Task Id 0 indicates model initialization point where all the λ have a positive value of 0.5. In Figure A.3(f) this corresponds to 100% active gradient directions, meaning all the gradient directions are available for gradient descent. As we learn more and more tasks, across the layers, more λ becomes negative. In our algorithm, to ensure the stability of the old knowledge (hence less forgetting), we avoid taking gradient steps along the directions with negative λ. This trend is captured in the Figure A.3(f) where, as we learn more tasks, the total number of (active) gradient directions along which optimization steps can be taken is gradually reduced. This finding is in agreement with the previous findings in the EWC (20) (where a higher number of weight changes are heavily penalized as more tasks are learned continually) and GPM (36) (where the number of active gradient directions is reduced as more tasks are learned). The main difference is in those methods an explicit constraint or regularization is used to control stability-plasticity of the model for continual learning, whereas in Amphibian that constraint is learned implicitly from the data. Figure A.3(a)-(e) also reveals that at different layers different level of constraints are needed for online continual learning and it would be hard to hand design them explicitly. This is where learning from data offers the most benefits compared to the other methods in achieving optimal performance for fast online continual learning.

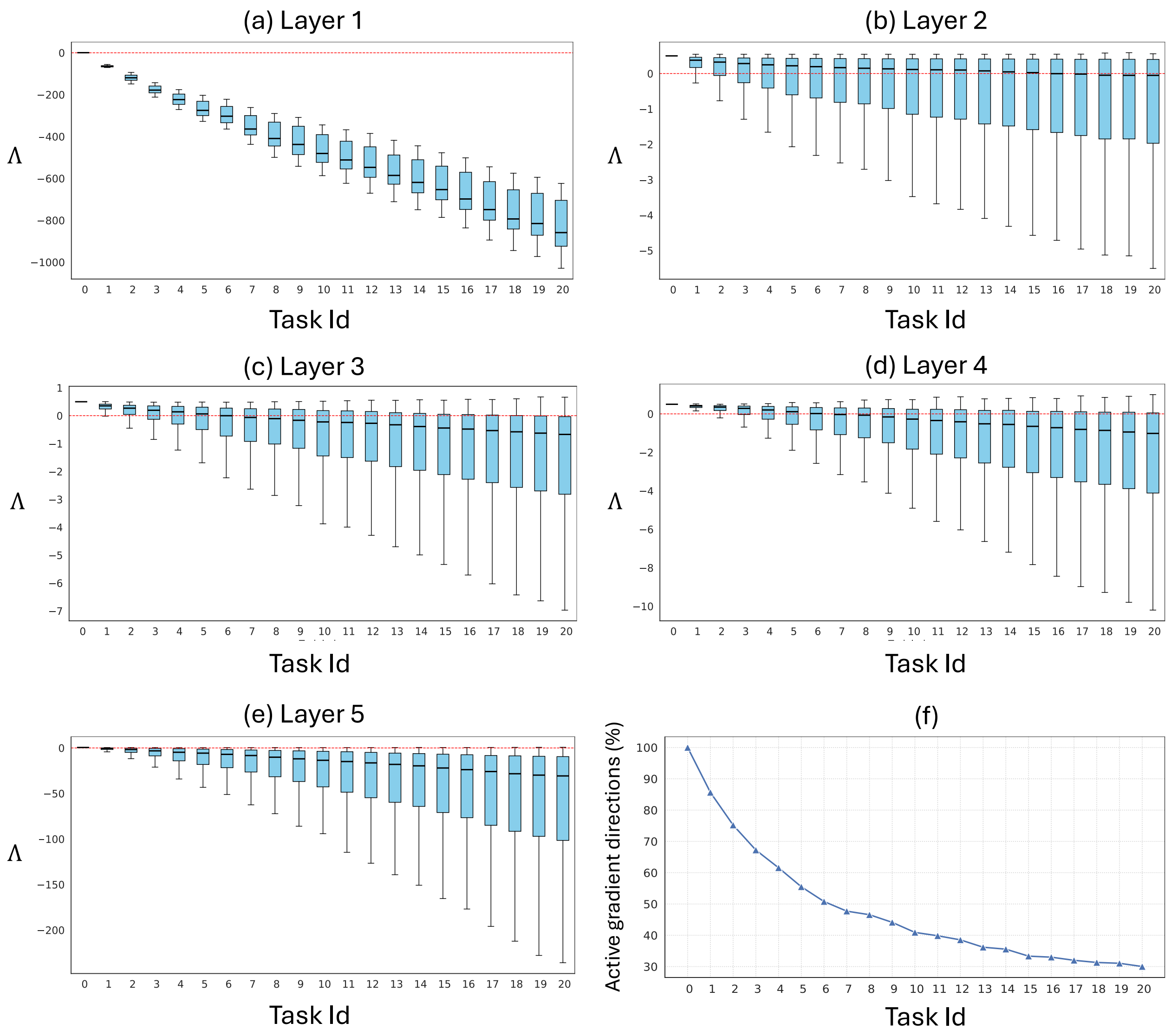

Figure A.3: (a)-(e) Layer-wise evolution of the Λ during training of 20 CIFAR-100 tasks. For each layer, we show a box plot of λ values contained in the diagonal matrix Λ. The horizontal red dotted line indicates 0 value and Task Id 0 indicates the model initialization point where all the λ are 0.5. As we learn more tasks more λ in each layer become negative. (f) Percentage (%) of active gradient directions after learning each CL task.

6 Representation of the Gradient Space in Neural Network

Here we elaborate on the point that SGD (gradient) updates lie subspace spanned by the input data points. Following (36), let's consider a single layer linear neural network in supervised learning setup where $\boldsymbol{x} \in \mathbb{R}^n$ is the input vector, $\boldsymbol{y} \in \mathbb{R}^m$ is the label vector in the dataset and $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ are the parameters (weights) of the network. The network is trained by minimizing the following mean-squared error loss function

$$L = \frac{1}{2}||\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y}||_2^2. \tag{A.17}$$

We can express gradient of this loss with respect to weights as

$$\nabla_{\boldsymbol{W}} L = (\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y})\boldsymbol{x}^T = \boldsymbol{\delta}\boldsymbol{x}^T, \tag{A.18}$$

where $\boldsymbol{\delta} \in \mathbb{R}^m$ is the error vector. Thus, **the gradient update will lie in the span of input** ($\boldsymbol{x}$), where elements in $\boldsymbol{\delta}$ scale the magnitude of $\boldsymbol{x}$ by different factors. Here, we have considered per-example loss (batch size of 1) for simplicity. In the batch/mini-batch setting, the batch loss is the summation of the losses due to individual examples, the total batch loss for k samples can be expressed as

$$L_{batch} = \sum_{i=1}^{k} L_i = \sum_{i=1}^{k} \frac{1}{2} ||\boldsymbol{W}\boldsymbol{x}_i - \boldsymbol{y}_i||_2^2. \tag{A.19}$$

The gradient of this loss with respect to weights can be expressed as

$$\nabla_{\boldsymbol{W}} L_{batch} = \boldsymbol{\delta_1} \boldsymbol{x}_1^T + \boldsymbol{\delta_2} \boldsymbol{x}_2^T + ... + \boldsymbol{\delta_k} \boldsymbol{x}_k^T. \tag{A.20}$$

Therefore, **the gradient update will remain in the subspace spanned by the k input examples**. In deep neural networks, the number of input examples is very large compared to the dimension of the inputs, $k >> n$. Assuming this, the largest subspace these k input examples can span will be n dimensional.

In this example, since weight $\boldsymbol{W}$ is $m \times n$ dimensional, naturally the corresponding gradients are $m \times n$ dimensional. However, according to Equation A.20, gradients will lie in the n dimensional subspace. So, in the paper, when we refer to: "the gradients lie in a low dimensional subspace", we imply gradients will lie in the n dimensional subspace within the original $m \times n$ dimensional space. With the n one hot vector e, we defined the canonical basis of that subspace. Following the definition in (36), we denoted that space as **gradient space**. We can draw the same conclusions by analyzing the convolutional layers of the neural networks, for that we refer to Section 4 in (36).