LEARNING TO LEARN WITHOUT FORGETTING USING ATTENTION

Anna Vettoruzzo
Center for Applied Intelligent Systems Research (CAISR)
Halmstad University
Sweden
anna.vettoruzzo@hh.se

Joaquin Vanschoren
Automated Machine Learning Group
Eindhoven University of Technology
Netherlands
j.vanschoren@tue.nl

Mohamed-Rafik Bouguelia
Center for Applied Intelligent Systems Research (CAISR)
Halmstad University
Sweden
mohamed-rafik.bouguelia@hh.se

Thorsteinn Rögnvaldsson
Center for Applied Intelligent Systems Research (CAISR)
Halmstad University
Sweden
thorsteinn.rognvaldsson@hh.se

ABSTRACT

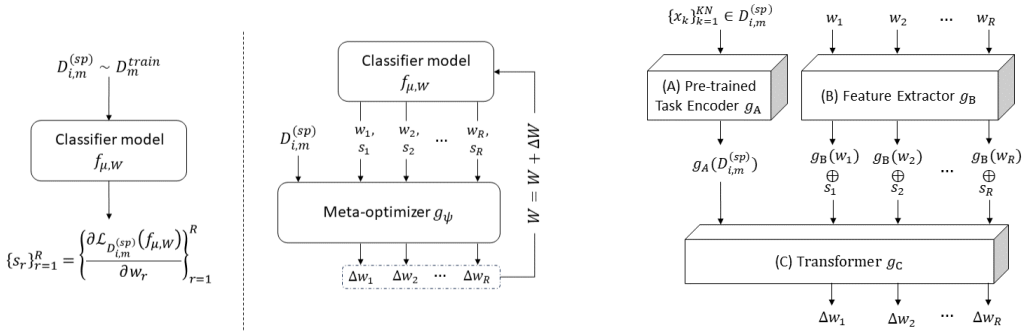
Continual learning (CL) refers to the ability to continually learn over time by accommodating new knowledge while retaining previously learned experience. While this concept is inherent in human learning, current machine learning methods are highly prone to overwrite previously learned patterns and thus forget past experience. Instead, model parameters should be updated selectively and carefully, avoiding unnecessary forgetting while optimally leveraging previously learned patterns to accelerate future learning. Since hand-crafting effective update mechanisms is difficult, we propose meta-learning a transformer-based optimizer to enhance CL. This meta-learned optimizer uses attention to learn the complex relationships between model parameters across a stream of tasks, and is designed to generate effective weight updates for the current task while preventing catastrophic forgetting on previously encountered tasks. Evaluations on benchmark datasets like SplitMNIST, RotatedMNIST, and SplitCIFAR-100 affirm the efficacy of the proposed approach in terms of both forward and backward transfer, even on small sets of labeled data, highlighting the advantages of integrating a meta-learned optimizer within the continual learning framework.

Keywords: Continual learning, Meta-learning, Few-shot learning, Catastrophic forgetting

1 INTRODUCTION

Continual learning (CL), or lifelong learning, refers to the ability of a machine learning system to continuously acquire knowledge from a stream of learning tasks while retaining previously learned experience (Cossu et al., 2021). CL involves a non-stationary stream of tasks where the data distribution evolves over time (Son et al., 2023). For instance, consider an autonomous vehicle driving across different countries under varying weather conditions and road surfaces. The vehicle must constantly adapt to changes in the environment by leveraging past knowledge to accelerate the learning process without forgetting how to handle past environments. Conventional neural networks will update all model parameters (weights) when adapting to new tasks, overwriting previously stored information, and therefore tend to gradually forget previously learned knowledge, a phenomenon known as catastrophic forgetting. While one could store samples from all previously encountered tasks and use them to repeatedly retrain the model, this is very inefficient. Instead, model parameters should be updated selectively and carefully, protecting certain parameters to avoid forgetting while allowing others to adapt quickly to accelerate future learning.

Current CL techniques do this via regularization based on hand-crafted heuristics, e.g., by protecting weights that had a significant impact in previous tasks (Kirkpatrick et al., 2017), by keeping an episodic memory with examples from previous tasks (which has scalability and privacy issues), or by gradually extending the architecture with new parameters (which limits scalability). Meta-learning methods (Schmidhuber, 1987; Santoro et al., 2016; Finn et al., 2017; Vettoruzzo et al., 2024) can be used to adapt to new tasks very efficiently, e.g., by meta-learning a gradient descent optimizer, but prior works typically ignores the sequentiality and non-stationarity of CL problems. Finally, sparsification techniques allow learning a subnetwork for every single task, thereby reducing interference across diverse tasks while



(a) Proposed approach (b) Meta-optimizer

Figure 1: (a) Overall framework of the proposed approach. The support set of the current data batch $D_{i,m}^{(sp)}$ is fed into the classifier model $f_{\mu,W}$ to derive importance scores $\{s_r\}_{r=1}^R$. These scores, along with the weights of the classifier model that we want to optimize $W = \{w_r\}_{r=1}^R$, and $D_{i,m}^{(sp)}$, serve as inputs to the meta-optimizer g_ψ for predicting weight updates tailored to task $\mathcal{T}_m$ from which the data are sampled. (b) Architecture of the meta-optimizer. (A) The pre-trained task encoder learns a vector representation that characterizes the current task. (B) The feature extractor maps weight values into the feature space. (C) The transformer encoder predicts the weight updates.

promoting information sharing among related tasks (Kang et al., 2022; Sokar et al., 2021). However, determining the optimal sparsity level can be challenging and the model's capacity may quickly be surpassed on longer streams of tasks.

This paper presents a novel approach, integrating meta-learning into the CL framework. It introduces a transformer-based meta-optimizer network that learns how to adapt each parameter of a separate classifier network using attention to learn the complex relationships between the parameters across a lifelong stream of tasks. The meta-optimizer is trained to predict task-specific weight updates for fast adaptation to new tasks while minimizing interference with previous tasks to avoid forgetting and scales to long streams of tasks. Additionally, the proposed approach ensures robust performance on future tasks while leveraging only a small set of labeled data, thanks to the generalization capability of meta-learning. This appears to be particularly useful in real-world applications where new tasks are regularly encountered, and the model needs to generalize to them. The idea of this paper draws inspiration from the presence of task-specific pathways within the classifier's weights (Dean, 2021). The meta-optimizer learns which sets of weights are important for particular tasks, and avoids overwriting pre-existing knowledge already stored in these weights. Additionally, by identifying overlapping pathways for related tasks, it also has the potential to optimize shared weights and increase performance on previously seen tasks (backward transfer).

To achieve this, we need to provide information about the current task and weights to the transformer-based optimizer. As illustrated in Figure 1, we first compute a task-specific *importance score* for each weight based on the gradient of the loss function when feeding labeled data from the current task to the classifier model before training on that task. By zeroing out the importance scores below a certain threshold, the meta-optimizer is forced to optimize the subset of weights considered important for the current task, thereby preventing catastrophic forgetting. The meta-optimizer takes as input a small set of labeled data from the current task, the current weights of the classifier model, and the importance scores, and outputs the weight updates in sequence, taking into account the specific task and the values of all previous weights. Internally, a pre-trained task embedding is used to allow the transformer to take into account the similarities between the current and previous tasks.

We demonstrate the ability of our meta-optimizer to predict effective weight updates on various datasets within the CL framework, even with small sets of labeled data. This proves particularly useful when the inference process needs to be performed effectively after observing only a limited amount of labeled data, a common scenario in real-world applications. Moreover, we find that predicting only the weight updates of the last layers of the classifier is sufficient to achieve good results in terms of backward transfer, supporting the idea that the last layers encode more task-specific information compared to earlier layers (Ramasesh et al., 2020). Importantly, this approach also eliminates the need to select optimizer-related hyperparameters for the classifier network, such as learning rate, weight decay, and optimizer type. Instead, the hyperparameters of the meta-optimizer proposed in this paper are selected only once at the beginning of the training, and the model itself automatically learns which optimization strategy is best suited for each task. This is especially important in the CL setting, where the environment is constantly evolving, and hyperparameters that

work well for a given task may not be effective for future ones. To our knowledge, this work represents the first approach proposing the use of a transformer model as a meta-optimizer applied in the context of CL. This establishes a foundation for future research to explore the potential of transformers in optimizing neural networks for CL.

In the remainder of this paper, we'll discuss the relationships with prior work in Sect. 2, and preliminaries in Sect. 3, before detailing our method in Sect. 4. We will present our experiments in Sect. 5 and conclusions in Sect. 6.

2 Related work

The CL problem has a number of variants, depending on whether the task identity is available at test time, i.e., task-incremental learning (TIL), or whether it has to be inferred, in which case one may have to predict new classes, i.e., class-incremental learning (CIL), or the same classes across domains, i.e., domain-incremental learning (DIL) (Van de Ven & Tolias, 2019; Wang et al., 2023). We can classify CL methods by the strategy used to alleviate catastrophic forgetting. Memory-based methods, such as those proposed in Buzzega et al. (2020); Rebuffi et al. (2017); Lopez-Paz & Ranzato (2017); Chaudhry et al. (2018), maintain a buffer of past data for knowledge rehearsal during inference. Regularization-based methods, as explored in Kirkpatrick et al. (2017); Serra et al. (2018); Aljundi et al. (2019); Zenke et al. (2017), impose regularization on network parameters with diverse heuristics during training. Meanwhile, dynamic architectural methods, as presented in Sokar et al. (2021); Hemati et al. (2023); Vladymyrov et al. (2023); Kang et al. (2022); Aljundi et al. (2017); Serra et al. (2018); Golkar et al. (2019); Rusu et al. (2016), allocate distinct sets of parameters for each task. Related to the last category, one way to completely avoid interference among diverse tasks is to grow new neural network branches for each new task while freezing the parameters of previous tasks (Kang et al., 2022) or dedicating a model copy to each task (Aljundi et al., 2017). However, these methods lead to high memory requirements as the number of tasks increases. Alternative approaches maintain a constant architecture size, allocating only a portion of the network to each task, where neurons are selected based on the concept of network sparsity (Sokar et al., 2021), self-attention based masks (Serra et al., 2018) or neuron-specific metrics (Golkar et al., 2019). A related idea is to apply hypernetworks (Ha et al., 2017) combined with regularization and generative replay techniques to generate task-specific network parameters and mitigate forgetting (Von Oswald et al., 2019). Since generating all weights is computationally expensive for large architectures, partial hypernetworks suggest generating weights only for the final layers of the classifier while keeping the initial layers unchanged (Hemati et al., 2023; Vladymyrov et al., 2023).

In meta-learning (Vettoruzzo et al., 2024), the goal is to train a model on a series of tasks so that a new task can be quickly learned with only a small set of labeled data, for instance by finding an initialization of the model parameters that allows fine-tuning on a small set of labeled data from that task (Finn et al., 2017), or by meta-learning the optimization rule that is at the basis of the neural network training (Chen et al., 2022). Instead of manually choosing an optimization strategy (such as Tseng (1998); Riedmiller & Braun (1993); Duchi et al. (2011); Kingma & Ba (2017)), Andrychowicz et al. (2016); Ravi & Larochelle (2016); Chen et al. (2017) propose an LSTM-based meta-learner that is directly trained to optimize a classifier model. Similarly, Wichrowska et al. (2017); Metz et al. (2022) and Jain et al. (2024) explore more intricate model architectures to achieve the same objective through advanced and complex structures. In particular, Jain et al. (2024) introduces a novel class of learnable optimizers based on spatio-temporal attention transformers, aiming to train large and intricate architectures like transformers and match state-of-the-art hand-designed optimizers. Differently from Jain et al. (2024), the primary focus of this paper is to introduce a method that learns to selectively update the parameters of a classifier network for each task in a sequential training stream, adjusting the optimization strategy accordingly while retaining the previous knowledge stored in the classifier's weights. Indeed, none of the previous approaches work well in this CL scenario, as they are primarily designed for cases where tasks are presented simultaneously. This results in a tendency to gradually forget the previous knowledge when new tasks are added to the stream and old tasks cannot be revisited. To address this challenge, La-MAML (Gupta et al., 2020) proposes a solution that mitigates catastrophic forgetting by incorporating a replay buffer and optimizing a meta-objective. MER (Riemer et al., 2018), inspired by GEM (Lopez-Paz & Ranzato, 2017), integrates an experience replay module to maximize the transfer of information and minimize interference between tasks. In contrast, OML (Javed & White, 2019) meta-learns an optimal representation offline and subsequently freezes it for continual learning on new tasks. Although these methods demonstrate notable performance on CL datasets, they require a large memory to store previous examples in the replay buffer, and they cannot generalize to training streams where the data distribution evolves significantly. Furthermore, they are not designed to deal with tasks where only a small set of labeled data is available at test time.

Our contribution involves designing an approach that replaces hand-crafted heuristics with meta-learning to adapt effectively and efficiently (in terms of labeled data) to streaming tasks without forgetting past knowledge. The proposed method directly optimizes the weights of a neural network classifier, even on small data samples, prevents interference

with existing knowledge, doesn't require a memory buffer, and scales to long streams of tasks. It ensures a balance between the ability to preserve past knowledge and the ability to learn new tasks - an aspect commonly referred to as the stability-plasticity trade-off (Grossberg, 2012).

3 PRELIMINARIES AND NOTATION

3.1 META-LEARNING

Meta-learning has recently emerged as a powerful paradigm to enable a learner to effectively learn unseen tasks with only a few samples by leveraging prior knowledge learned from related tasks (Vettoruzzo et al., 2024). In the meta-training phase a set of training tasks $\{\mathcal{T}_i\}_{i=1}^{T}$ is sampled from a task distribution $p(\mathcal{T})$. Each task corresponds to data generating distributions $\mathcal{T}_i \triangleq \{p_i(x), p_i(y|x)\}$, and the data sampled from each task is divided into a *support set*, $D_i^{(sp)}$, containing K training examples per class, and a *query set*, $D_i^{(qr)}$. During the meta-test phase, a completely new task $\mathcal{T}_{new}$ with few labeled data $D_{new}^{(sp)} \triangleq \{x_k, y_k\}_{k=1}^{K}$ is observed and the model is fine-tuned on $D_{new}^{(sp)}$ to achieve good generalization performance on new unlabeled test examples from $\mathcal{T}_{new}$. Optimization-based methods typically frame meta-learning as a bi-level optimization problem. At the inner lever, a neural network f_θ produces task-specific parameters θ_i' using $D_i^{(sp)}$, while at the outer level, the initial set of meta-parameters θ is updated by optimizing the performance of $f_{\theta_i'}$ on the query set of the same task. The result is a model initialization θ that can effectively be adapted to new tasks using only a few (K) training examples and a few gradient updates.

3.2 CONTINUAL LEARNING

Continual learning (CL) is characterized by learning from evolving data distributions. In practice, training samples from different distributions arrive sequentially, requiring a CL model to learn new tasks incrementally. Moreover, the model must retain the previous knowledge and perform well on the test set of all tasks seen so far, with limited or no access to samples from previous tasks. Therefore, in the context of CL, data is presented as a non-stationary *stream* of tasks, defined as $\overrightarrow{\mathcal{T}} = \{\mathcal{T}_m\}_{m=1}^{M}$. At each stage m, the entire dataset of task $\mathcal{T}_m$ is available for updating the model. The data sampled from each task is split into a training set D_m^{train} and a test set D_m^{test}. A label t_m identifies the current task and may or may not be available, depending on the categorization presented in Section 2. By combining meta-learning with continual learning (a scenario defined as meta-continual learning (MCL) in Son et al. (2023)), different batches of data can be sampled from D_m^{train} during training. Each batch is then divided into a limited support set $D_{i,m}^{(sp)}$, used to adapt the model to the current task, and a query set $D_{i,m}^{(qr)}$, to evaluate the performance of the adapted model, as described in Section 3.1. As the distribution evolves over time, different tasks are characterized by different data distributions, i.e., $p_m(x, y) \neq p_{m+1}(x, y)$, which implies that either the input distribution, the output distribution, or both may differ between different experiences. At inference time, when any task $\mathcal{T}_n$ (which can potentially be one of the earliest or the future ones) is sampled from the stream, only a small set of labeled data sampled from D_n^{test} is available. The model can adapt to it and then make accurate predictions on new unlabeled test examples in D_n^{test}. Consequently, the main challenge in CL is to mitigate catastrophic forgetting on previously encountered tasks while continually acquiring new knowledge from new tasks (Son et al., 2023).

4 PROPOSED APPROACH

In this section, we provide a detailed description of the proposed approach. The main objective is to meta-learn an optimizer that dynamically updates the weights of a classifier network, as illustrated in Figure 1a. The goal is to customize the optimization process for each task by learning to optimize a specific set of weights, thereby minimizing interference with previously acquired knowledge while promoting knowledge transfer across related tasks. Additionally, we enhance the generalization capability of our proposed approach by leveraging meta-learning algorithms to enable efficient and effective adaptation to each task without necessitating an identifier for the current task (defined as t_m in Section 3.2) and without the requirement of a replay buffer to store examples from previous tasks. In summary, our adjoint network leverages task-specific pathways within the classifier's weights, which allows for a more nuanced and adaptable approach to weight updates. This ensures that new knowledge can be incorporated without indiscriminately overwriting existing knowledge, directly addressing the challenge of catastrophic forgetting. Thanks to meta-learning, the model also guarantees to quickly adapt to new tasks by transferring relevant knowledge to it.

As depicted in Figure 1a, the meta-optimizer g_ψ, embodied as a transformer model, aims to predict weight updates for the classifier $f_{\mu,W}$, which is parameterized by two sets of parameters. Here, μ represents the meta-learned parameters

that aim to enhance generalizability across different tasks, while $W = \{\mathrm{w}_r\}_{r=1}^{R}$ consists of all weights optimized using the output of the meta-optimizer. For the sake of simplicity, we will refer to θ as the set of parameters that are meta-learned, comprising both ψ and μ, and to W as the classifier weights that are directly optimized by the meta-optimizer instead of being *trained* with a fixed optimizer. For a given task $\mathcal{T}_m$, θ is adapted to θ' based on batches of data from that task so that the produced W achieves good predictions for the task at hand. Therefore, we can formulate this as an optimization problem where the objective is as follows:

$$\min_{\theta} \sum_{D_{i,m}} \mathcal{L}(\{\mu \cup \underbrace{g_\psi(D_{i,m}^{(sp)})}_{W}\}, D_{i,m}^{(qr)}) \tag{1}$$

where $D_{i,m}, i = 1, ..., I$ are the data batches sampled from the dataset D_m associated with task $\mathcal{T}_m$. Ideally, all weights of the classifier model can be part of W except for those in the last layer (i.e., classifier head), which are always meta-learned to ensure adaptation to the N classes of the current task. Indeed, the classifier head in the proposed approach maintains a consistent structure (i.e., N output units) throughout all the tasks, neither changing (as in TIL) nor expanding to accommodate additional classes with each new task (as in CIL). Therefore, the classifier head is similar to the one used in DIL approaches, with the main difference being that it can be adapted to classify different classes thanks to meta-learning. Indeed, the primary goal of the classifier model is to distill informative features that characterize the current task in the deeper layers and enable accurate classification within the N classes after the adaptation of the classifier head to the current task. For this reason, our approach stands between TIL and CIL methods. Unlike TIL, it does not require a label identifier for the current task, and contrary to CIL, it does not aim to expand the classifier to include all classes encountered so far. Rather, our approach aligns more closely with the meta-learning framework detailed in Section 3.1. Each task in the training stream consists of data batches, each featuring a small support set and a query set. The support set is used to adapt the model to the specific task, enabling classifications within the N classes present in the current task, and the query set is used to validate the adapted model and update the meta-learned parameters. We present the architecture of the meta-optimizer in Section 4.1, and we provide the complete algorithm in Section 4.2.

4.1 META-OPTIMIZER

As illustrated in Figure 1b, the meta-optimizer comprises three key components: (A) a pre-trained task encoder g_A, (B) a feature extractor g_B, and (C) a transformer encoder model g_C. For each data batch $D_{i,m}$ sampled from $\mathcal{T}_m$, the pre-trained task encoder g_A takes as input the data from the support set of the current batch $D_{i,m}^{(sp)}$ without the corresponding labels. This data is denoted as $\{x_k\}_{k=1}^{KN} \in D_{i,m}^{(sp)}$, where K examples are available for each of the N classes. After extracting meaningful features from the data, an averaging layer computes the mean of the transformed data along the KN examples, producing a vector representation $g_A(D_{i,m}^{(sp)})$ that characterizes the current task. This representation is invariant to permutations of the examples and is expected to capture task-specific information. The feature extractor g_B is primarily responsible for mapping weight values into a size suitable for the transformer model. For convolutional weights $\{\mathrm{w}_r\}_{r=1}^{R}$, with dimension $n_{input} \times n_{output} \times k \times k$, two encoding approaches are employed: (a) *sequential allocation*, where weights are flattened into a one-dimensional vector of size $(n_{input} \cdot n_{output} \cdot k \cdot k, 1)$, and (b) *spatial allocation* which generates $n_{input} \cdot n_{output}$ vectors with size k^2. The importance scores are modified in a similar manner: for *sequential allocation*, they are flattened into a vector of size $n_{input} \cdot n_{output} \cdot k \cdot k$, while for *spatial allocation*, an average value for each kernel is computed by averaging the k^2 scores and squeezing the vector into a size $(n_{input} \cdot n_{output}, k^2)$.

The output from the feature extractor is concatenated with the importance score vector before being fed into the transformer model. This operation is denoted as $g_B(\mathrm{w}_r) \oplus s_r$ for $r = 1, \cdots, R$ in Figure 1b. It is worth noting that both the output of the pre-trained task encoder and the output of the feature extractor, once concatenated with the importance score, have the same dimensionality. Finally, the transformer encoder g_C learns weight updates specific to the current task, i.e., $\{\Delta\mathrm{w}_r\}_{r=1}^{R}$. The transformer mainly consists of a learnable positional encoding, one or more encoder layers, and a linear layer on top that generates the corresponding update for the input weight vector.

In essence, the meta-optimizer operates as a cohesive unit, extracting task-specific information, effectively mapping weight values, and generating task-specific weight updates through the transformer encoder. This entire model is meta-trained to enhance its generalization capability across different tasks, ensuring effective learning with only a limited number of examples.

4.2 ALGORITHM

As highlighted before, θ refers to the set of parameters that are meta-learned and W to the classifier weights that are directly optimized by the meta-optimizer. To further emphasize this distinction, we use the notation θ' to indicate the parameters optimized by a fixed optimizer and $\tilde{W}$ to indicate the weights updated by the meta-optimizer. It is important to notice that θ comprises both the parameters of the meta-optimizer, denoted as ψ, and some parameters of the classifier model, i.e., μ. Indeed, as previously mentioned, the last layer of the classifier model is always meta-learned since it must be adapted to the classes of the current task. The complete algorithm of the proposed approach is presented in Algorithms 1 and 2.

At each stage m, a task $\mathcal{T}_m$ is sampled from the training stream $\vec{\mathcal{T}}$ and the whole model is trained to learn such task effectively, without forgetting the previously encountered ones. As mentioned in Section 3.2, the data from each task is split into D_m^{train} and D_m^{test}. In line 6, batches of data $D_{i,m}, i = 1, ..., I$ are sampled from D_m^{train} and each batch is split into two disjoint sets, $D_{i,m}^{(sp)}$ and $D_{i,m}^{(qr)}$. The support set is used in line 9 to *Adapt&Optimize* the classifier model for the current task. This function is presented in detail in Algorithm 2. After a first initialization step (lines 1 and 2 in Algortihm 2), Q iterations are performed to adapt the meta-learned parameters θ' to the current task. For each iteration, an importance score is computed in line 5 of Algorithm 2 for each weight $\{\mathrm{w}_r\}_{r=1}^R$ in W. This calculation is performed by inputing $D_{i,m}^{(sp)}$ in the classifier model $f_{\mu',W}$ and computing the gradient of the loss function $\nabla_W \mathcal{L}_{D_{i,m}^{(sp)}}(f_{\mu',W})$ before the weights W are optimized for the current data batch. These gradients are referred to as importance scores $\{s_r\}_{r=1}^R$. They provide information about the importance of optimizing each weight for making predictions on $D_{i,m}$, and more generally on $\mathcal{T}_m$, and encourage the meta-optimizer to learn an update strategy similar to gradient descent. Scores below a certain threshold are zeroed out since they do not serve in solving the task at hand. This guarantees that we don't overwrite knowledge from previous tasks, thus avoiding forgetting.

Algorithm 1 Meta-training from a stream of tasks

Require Stream of tasks $\vec{\mathcal{T}}$, inner steps Q, learning rates α, β

1: Randomly initialize $\theta = \{\mu, \psi\}$ and W
2: **for** $m = 1, \cdots, M$ **do**
3: Sample $\mathcal{T}_m$ from $\vec{\mathcal{T}}$
4: Sample data D_m^{train} from $\mathcal{T}_m$
5: **while** not done **do**
6: Get data batches $D_{i,m}, i = 1, ..., I$ from D_m^{train}
7: **for all** $D_{i,m}$ **do**
8: Split data $D_{i,m} = \{D_{i,m}^{(sp)}, D_{i,m}^{(qr)}\}$
9: $\theta'_i, \tilde{W}_i \leftarrow$ *Adapt&Optimize*$(Q, \alpha, D_{i,m}^{(sp)}, \theta, W)$
10: $\{\mu'_i, \psi'_i\} \leftarrow \theta'_i$
11: **end for**
12: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{D_{i,m}} \mathcal{L}_{D_{i,m}^{(qr)}}(f_{\mu'_i, \tilde{W}_i})$
13: Set $W \leftarrow \tilde{W}_I$
14: **end while**
15: **end for**
16: **return** θ, W

Algorithm 2 *Adapt&Optimize*

Require Adaptation steps Q, learning rate α, support set $D_{i,m}^{(sp)}$, meta-learned parameters $\theta = \{\mu, \psi\}$, optimized parameters $\tilde{W}$

1: Initialize $\mu' \leftarrow \mu, \psi' \leftarrow \psi$
2: Set $\theta' = \{\mu', \psi'\}$
3: **for** $q = 1, \cdots, Q$ **do**
4: Initialize $W \leftarrow \tilde{W}$
5: Compute importance scores $\{s_r\}_{r=1}^R \leftarrow \nabla_W \mathcal{L}_{D_{i,m}^{(sp)}}(f_{\mu',W})$
6: $\Delta W \leftarrow g_{\psi'}(D_{i,m}^{(sp)}, W, \{s_r\}_{r=1}^R)$
7: $\tilde{W} \leftarrow W + \Delta W$
8: $\theta' \leftarrow \theta' - \alpha \nabla_{\theta'} \mathcal{L}_{D_{i,m}^{(sp)}}(f_{\mu',\tilde{W}})$
9: **end for**
10: **return** $\theta', \tilde{W}$

In line 6 of Algorithm 2, the resulting scores $\{s_r\}_{r=1}^R$, together with W and $D_{i,m}^{(sp)}$ are input in the meta-optimizer $g_{\psi'}$ which predicts the weight updates ΔW, as described in Section 4.1. The updates are then summed to the current weight values in line 7, resulting in the optimized weights $\tilde{W}$. These weights are then used to adapt θ' to the specific task. This adaptation is performed using the optimized weights $\tilde{W}$ and the batch of data sampled from the current task, i.e., $D_{i,m}^{(sp)}$. At the end of this process, the adapted parameters $\theta' = \{\mu', \psi'\}$ and the optimized weights $\tilde{W}$ are returned and used in line 12 of Algorithm 1 to update the original θ. To do so, the post-adaption loss is computed with the query set $D_{i,m}^{(qr)}$ and the classifier model $f_{\mu'_i, \tilde{W}_i}$, where μ'_i are the parameters adapted to the support set of the current batch $D_{i,m}^{(sp)}$ and $\tilde{W}_i$ are optimized with the meta-optimizer. Here, any optimizer of choice, such as Adam, can be used to update the parameters θ. The result of meta-training (line 16) is a set of parameters $\theta = \{\mu, \psi\}$ that generalize well for

all tasks. Meanwhile, the parameters W are optimized in a way that enables them to solve the current task effectively without overwriting the knowledge already encoded from earlier tasks.

During inference, the model is given a small set of labeled data from a task $\mathcal{T}_n \in \vec{\mathcal{T}}$ which may correspond to one of the previous or future (new) tasks. More specifically, a small set of labeled data $D_{test,n}^{(sp)}$ is sampled from the test set D_n^{test} of a task $\mathcal{T}_n$. The parameters θ and W are then adapted and optimized as outlined in Algorithm 2, possibly with a different number of adaptation steps Q than the one used for training. The model, now parameterized by θ'_n and $\tilde{W}_n$, can be used to make accurate predictions on new unlabeled test examples from D_n^{test}.

5 Experiments

5.1 Datasets and benchmark methods

The effectiveness of the proposed approach is evaluated across common CL datasets, including SplitMNIST, RotatedMNIST, and SplitCIFAR-100. The selection of these datasets is primarily guided by their extensive usage in CL literature and the intention to assess the effectiveness of the proposed approach in DIL scenarios by presenting results on the RotatedMNIST dataset. SplitMNIST is a variation of the original MNIST dataset, comprising five streaming tasks, each featuring two consecutive MNIST digits. RotatedMNIST is generated by rotating MNIST images by a random degree rotation between 0 and 180, consisting of a total of 10 tasks. This dataset is usually used in domain-incremental learning, where the structure of the problem is always the same (classifying digits from 0 to 9), but the input distribution changes due to domain shifts. Similarly to SplitMNIST, SplitCIFAR-100 is constructed by selecting five classes per task from the CIFAR-100 dataset, and creating a stream with five distinct tasks.

Results are compared with several established CL methods. Given that the proposed approach lies between TIL and CIL, as discussed in Section 4, a diverse set of baselines is used for comparison. For TIL, this includes regularization-based strategies like EWC (Kirkpatrick et al., 2017) and SI (Zenke et al., 2017), approaches that integrate meta-learning with CL such as La-MAML (Gupta et al., 2020) and Sparse-MAML (Von Oswald et al., 2021), as well as dynamic architectural methods like WSN (Kang et al., 2022). Furthermore, two CIL methods, iCaRL (Rebuffi et al., 2017) and DER++ (Buzzega et al., 2020), are also included in the comparison.

For all methods, three metrics are used for performance evaluation. The average accuracy ("Avg accuracy") represents the mean classification accuracy across all tasks. Backward transfer (BWT) (Lopez-Paz & Ranzato, 2017) measures the ability to retain past knowledge from previous tasks. Negative BWT occurs when learning a task results in decreased performance on some preceding tasks, indicating catastrophic forgetting. Forward transfer (FWT), on the other hand, measures the ability to transfer knowledge to learn a new task that has never been encountered before.

5.2 Training setting

To ensure fair and reliable comparisons, all baselines use similar hyperparameters and model architectures. The default setting is used for the value of the specific parameters for the baseline methods. All models are trained for 10^5 steps, with a batch size of 32 and $K = 5$, where K denotes the number of training examples per class in the support set of each data batch. The importance scores are normalized within the range $[0, 1]$ and only the top 60% are retained, while the rest are zeroed out, as discussed in Section 4.2. Since negligible differences (less than 0.1%) were observed when comparing *sequential allocation* with *spatial allocation* of the weights, the latter is chosen for all experiments to more efficiently utilize GPU memory. Two distinct classifier architectures are used for SplitMNIST/RotatedMNIST and SplitCIFAR-100. The first classifier includes a convolutional layer with 32 filters and a classifier head, while the second model comprises three convolutional layers with 100 filters and concludes with a classifier head. The meta-optimizer, as described in Section 4.1, consists of three parts. The task encoder (part A) is pre-trained on the SVHN dataset for SplitMNIST and RotatedMNIST, and on ImageNet for SplitCIFAR-100. The feature extractor (part B) comprises one hidden layer with 16 neurons, while the transformer architecture (part C) is a transformer encoder with two layers, GeLU activation, and four heads. A hyperbolic tangent operation in the range [-3, 3] is applied on top of the transformer encoder to ensure the transformer's output is not too large. More details about the training setting can be found in Appendix A.1. To account for statistical variations, each algorithm is run three times in full. All experiments are executed on a single Nvidia A100-SXM4 GPU with 40GB of RAM, using Python and the PyTorch library. The source code is available at https://github.com/annaVettoruzzo/L2L_with_attention.

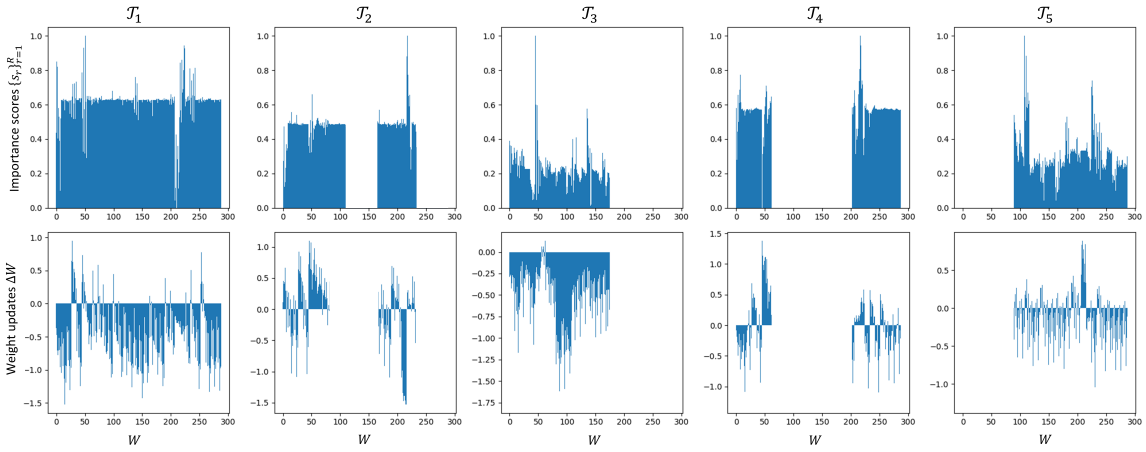


Figure 2: Visualization of the importance scores $\{s_r\}_{r=1}^R$ and the weight updates learned by the meta-optimizer g_ψ after adaptation to each task $\mathcal{T}_i \in \vec{\mathcal{T}}$ for SplitMNIST dataset.

5.3 Results

First, we verify whether the meta-optimizer learns to optimize weights selectively, updating only the weights considered important for a given task (i.e., resulting in a high importance score), hence resulting in low forgetting. Figure 2 shows the importance scores given to the meta-optimizer (top), and the corresponding weight updates learned by the meta-optimizer for each task in the training stream (bottom) for SplitMNIST, showing a strong correlation. More experiments on the influence of the importance scores in the proposed approach can be found in Appendix A.2.

The full performance results are presented in Table 1a for SplitMNIST/RotatedMNIST and in Table 1b for SplitCIFAR-100. To highlight the effectiveness of the proposed approach in balancing the different metrics, Figure 3 shows a Pareto plot illustrating the average BWT versus FWT across all datasets. On average, our proposed approach significantly outperforms most other CL techniques in both forward and backward transfer. It is competitive with Sparse-MAML on BWT but significantly outperforms it in terms of FWT. Indeed, the primary goal of this paper is to introduce an approach that not only prevents forgetting previous knowledge, as indicated by the BWT metric, but also ensures robust performance on future tasks while leveraging only a small set of labeled data, which is particularly noticeable in the FWT metric, where the proposed approach outperforms all other baselines by a significant margin. In terms of "Avg accuracy" and BWT, TIL methods (such as EWC, SI, La-MAML, Sparse-MAML, and WSN) show superior results on SplitMNIST and SplitCIFAR-100. However, these methods require additional information during training and evaluation, as discussed in Section 2. Specifically, TIL methods use a task identifier t_m to implement a multi-head approach, enhancing prediction outcomes, especially in simpler datasets like SplitMNIST. Nevertheless, when applied to more complex datasets, such as RotatedMNIST and SplitCIFAR-100, TIL methods show a large negative BWT, indicating their inability to retain past knowledge. An exception is WSN, which has a BWT close to 0 on SplitMNIST and SplitCIFAR-100. Indeed, as stated in the original paper (Kang et al., 2022), WSN is almost immune to catastrophic forgetting as each task is associated with a distinct subnetwork model that does not interfere with others. However, this poses a challenge as the number of tasks increases and the network exhausts its capacity to allocate new weights, as observed in RotatedMNIST. The low BWT, in this case, is also influenced by the use of a single-head classifier, given that the structure of the problem, i.e., classifying digits from 0 to 9, remains the same for all tasks. CIL methods like iCaRL and DER++ show lower performance on SplitMNIST and SplitCIFAR-100. These methods infer the task identity and expand the classifier head as new classes are encountered. This approach is more challenging compared to TIL and the method proposed in this paper. Yet, when considering the RotatedMNIST dataset, where CIL methods behave similarly to the proposed approach since the classes are all the same across tasks, our method still shows better backward and forward transfer. Specifically, while iCaRL cannot be extended to this scenario (thus not included in Table 1a), the proposed approach shows a better backward and forward transfer compared to DER++, highlighting the efficacy of the meta-optimizer to extract context from the few input data provided for each task and learn an optimization strategy tailored to that task.

Table 1: Performance comparison on (a) SplitMNIST and RotatedMNIST and (b) SplitCIFAR-100. The baseline methods are organized into TIL methods, CIL methods, and the proposed approach. Results report the mean and std across three different runs of all the algorithms for Average accuracy, BWT and FWT.

Method	**SplitMNIST**			**RotatedMNIST**		
	Avg accuracy	BWT	FWT	Avg accuracy	BWT	FWT
EWC	$\mathbf{99.76 \pm 0.04}$	-0.03 ± 0.01	68.40 ± 4.48	53.32 ± 1.95	-38.86 ± 0.49	54.94 ± 0.50
SI	99.82 ± 0.00	$\mathbf{-0.01 \pm 0.01}$	70.62 ± 4.01	45.46 ± 1.92	-48.97 ± 0.43	55.42 ± 0.36
La-MAML	99.35 ± 0.18	-0.24 ± 0.13	68.67 ± 4.45	66.45 ± 1.82	-23.78 ± 1.03	52.95 ± 0.52
Sparse-MAML	93.54 ± 3.00	-5.49 ± 2.93	82.56 ± 2.53	12.41 ± 1.61	-15.81 ± 2.30	16.03 ± 0.15
WSN	99.74 ± 0.05	-0.09 ± 0.03	70.22 ± 3.56	45.28 ± 2.52	-48.46 ± 1.14	56.04 ± 0.55
iCaRL	84.85 ± 0.20	-8.63 ± 0.11	36.85 ± 0.04	–	–	–
DER++	84.61 ± 2.61	-11.03 ± 2.58	68.14 ± 1.45	$\mathbf{73.67 \pm 0.75}$	-18.91 ± 0.70	55.65 ± 0.90
Proposed	92.78 ± 0.66	-6.74 ± 1.52	$\mathbf{90.40 \pm 0.26}$	62.09 ± 1.78	$\mathbf{-1.70 \pm 1.08}$	$\mathbf{66.07 \pm 0.61}$

(a) Results on SplitMNIST and RotatedMNIST

Method	**SplitCIFAR-100**		
	Avg accuracy	BWT	FWT
EWC	52.49 ± 1.62	-29.66 ± 1.97	43.29 ± 0.12
SI	55.84 ± 0.83	-25.32 ± 1.26	42.98 ± 0.50
La-MAML	52.64 ± 1.60	-22.59 ± 0.84	41.10 ± 0.37
Sparse-MAML	37.68 ± 3.57	-0.82 ± 1.30	34.39 ± 1.61
WSN	$\mathbf{82.88 \pm 0.13}$	$\mathbf{-0.22 \pm 0.10}$	44.47 ± 0.95
iCaRL	20.59 ± 1.58	-12.97 ± 0.92	10.67 ± 0.63
DER++	41.32 ± 0.41	-36.11 ± 0.61	39.62 ± 1.36
Proposed	49.96 ± 2.33	-14.95 ± 3.16	$\mathbf{49.95 \pm 0.31}$

(b) Results on SplitCIFAR-100

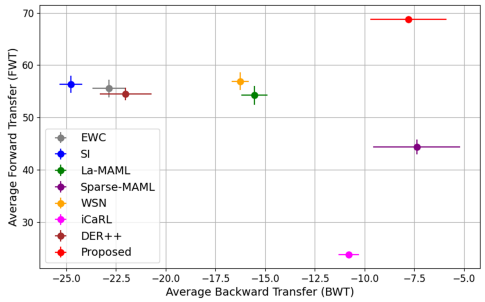
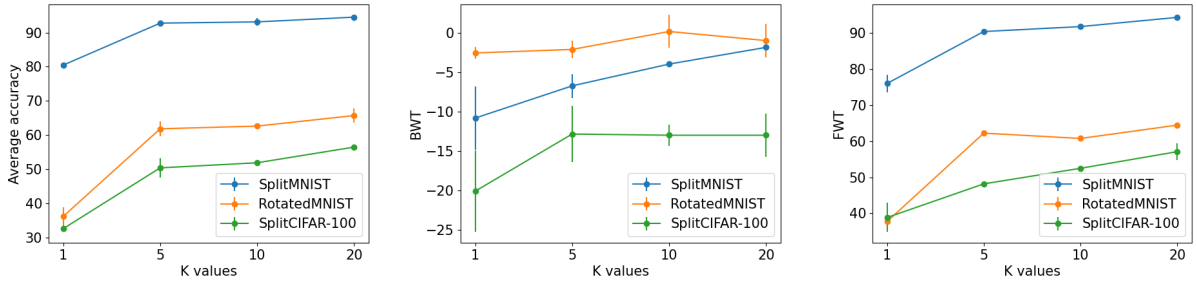


Figure 3: Pareto plot of average BWT vs. FWT



(a) Average accuracy (b) BWT (c) FWT

Figure 4: Visualization of the three metrics (average accuracy, BWT, FWT) varying the number of labeled examples in the support set of each task from $K = 1$ to $K = 20$, for all three datasets.

Even better results can be achieved by increasing the number of labeled examples in the support sets, allowing the model to better capture task-specific information and thus better adapt the meta-optimizer to predict weight updates specifically tailored for the task at hand. This is shown in Figure 4, where the three performance metrics are computed for the three datasets when training the proposed approach with $K = 1, 5, 10$, and 20 examples.

To assess the meta-optimizer's capability in effectively learning an optimization strategy tailored for each task, we perform an ablation study comparing MAML with the proposed approach, as shown in Table 2. The results indicate that the proposed approach outperforms MAML in both SplitMNIST and SplitCIFAR-100. Only on the RotatedMNIST dataset, both MAML and the proposed approach yield comparable results. To better understand this, we can examine the test accuracy for different tasks during training. While MAML achieves better test accuracy on the last-seen task, it suffers from catastrophic forgetting (i.e., high BWT). In contrast, the test accuracy obtained with the proposed approach is lower for the last-seen task, but the BWT is significantly improved. This lower accuracy can be attributed to the limited network capacity, which constrains the classifier's ability to diversify weight updates across different tasks.

Table 2: Comparison between the proposed approach and MAML (Finn et al., 2017) The primary distinction between these two methods is the use of the meta-optimizer in the proposed approach to learn a task-specific update rule, as opposed to directly optimizing all the weights of the classifier model using gradient descent, as in MAML. Results report the mean and std across three different runs of all the algorithms for "Avg accuracy", BWT and FWT.

		MAML	Proposed
SplitMNIST	Avg accuracy	86.11 ± 1.30	$\mathbf{92.78 \pm 0.66}$
	BWT	-14.28 ± 1.36	$\mathbf{-6.74 \pm 1.52}$
	FWT	88.72 ± 0.54	$\mathbf{90.40 \pm 0.26}$
RotatedMNIST	Avg accuracy	$\mathbf{65.65 \pm 1.36}$	62.09 ± 1.78
	BWT	-19.46 ± 0.98	$\mathbf{-1.70 \pm 1.08}$
	FWT	$\mathbf{67.00 \pm 0.37}$	66.07 ± 0.61
SplitCIFAR-100	Avg accuracy	46.15 ± 0.96	$\mathbf{49.96 \pm 2.33}$
	BWT	-36.22 ± 2.82	$\mathbf{-14.95 \pm 3.16}$
	FWT	49.72 ± 0.45	$\mathbf{49.95 \pm 0.31}$

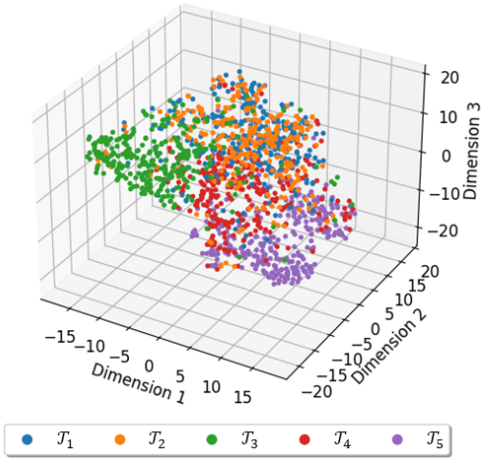


Figure 5: Visualizaton of the optimized weights at test time for all SplitMNIST tasks $\{\mathcal{T}_m\}_{m=1}^{M}$.

However, while we could not experiment with larger network architectures due to memory constraints, that would likely address this limitation.

Finally, Figure 5 illustrates the ability of the model to learn diverse weight updates on small training samples using the same model architecture. This plot visualizes the optimized weights W for the different tasks in SplitMNIST by applying tSNE (Van der Maaten & Hinton, 2008) to reduce the data dimensionality. As expected, the weights are optimized differently for each task, showcasing the ability of the meta-optimizer to effectively recognize task differences without requiring a task identifier. Moreover, certain weight values remain constant across different tasks, allowing the classifier model to reuse previously acquired knowledge in solving similar tasks.

6 Conclusion

This paper introduces a novel approach to address the challenges of CL by integrating meta-learning techniques. The primary contribution is the introduction of a meta-learned transformer-based optimizer, designed to learn an update strategy that enhances the capability of machine learning models to continuously learn while retaining knowledge over time. Notably, the proposed approach does not require a memory buffer to store information from past tasks. It can scale to long streams of tasks since it doesn't add new parameters to the model. It could also scale to larger images by using larger classifiers and focusing on updating the last layer. Indeed, the complexity of the meta-model primarily depends on the number of weights to optimize rather than the input data. Furthermore, using a meta-learned optimizer eliminates the need to manually select hyperparameters for the classifier network, which is critical in dynamic CL environments. Through comprehensive evaluations on benchmark datasets, the results demonstrate the effectiveness of the proposed approach. Leveraging only a limited set of labeled data, the method achieves strong performance in terms of both backward and forward transfer. The meta-optimizer effectively predicts task-specific weight updates, showcasing its potential in real-world applications where adapting to new tasks with limited labeled data is crucial.

Finally, the pioneering use of a transformer model as a meta-optimizer in the context of CL opens up avenues for future research to further explore its capabilities. In particular, it would be interesting to meta-learn larger transformer models on longer streams, which should improve both BWT and FWT as it can meta-learn over more tasks, or to combine it with episodic memory buffers to improve BWT. Further works could extend the N-way classification approach detailed in this paper to accommodate a dynamic classifier head capable of predicting all encountered classes, as in CIL. Finally, given that the optimizer works with batches of data as input, the proposed approach could also be extended to online CL scenarios.

Acknowledgments

This work was supported by the EU's Horizon Europe research and innovation program under grant agreement No. 952215 (TAILOR).

REFERENCES

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3366–3375, 2017.

Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Zhangyang Wang, Howard Heaton, Jialin Liu, and Wotao Yin. Learning to optimize: A primer and a benchmark. *The Journal of Machine Learning Research*, 23(1):8562–8620, 2022.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pp. 748–756. PMLR, 2017.

Andrea Cossu, Antonio Carta, Vincenzo Lomonaco, and Davide Bacciu. Continual learning for recurrent neural networks: an empirical evaluation. *Neural Networks*, 143:607–627, 2021.

Jeff Dean. Introducing pathways: A next-generation ai architecture, 2021. URL https://blog.google/technology/ai/introducing-pathways-next-generation-ai-architecture/.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Siavash Golkar, Micheal Kagan, and Kyunghyun Cho. Continual learning via neural pruning. In *Real Neurons & Hidden Units: Future directions at the intersection of neuroscience and artificial intelligence@ NeurIPS 2019*, 2019.

Stephen T Grossberg. *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control*, volume 70. Springer Science & Business Media, 2012.

Gunshi Gupta, Karmesh Yadav, and Liam Paull. Look-ahead meta learning for continual learning. *Advances in Neural Information Processing Systems*, 33:11588–11598, 2020.

David Ha, Andrew Dai, and Quoc V. Leham. Hypernetworks. In *International Conference on Learning Representations*, 2017.

Hamed Hemati, Vincenzo Lomonaco, Davide Bacciu, and Damian Borth. Partial hypernetworks for continual learning. In *2nd Conference on Lifelong Learning Agents (CoLLAs)*, 2023.

Deepali Jain, Krzysztof M Choromanski, Kumar Avinava Dubey, Sumeet Singh, Vikas Sindhwani, Tingnan Zhang, and Jie Tan. Mnemosyne: Learning to train transformers with transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

Khurram Javed and Martha White. Meta-learning representations for continual learning. *Advances in neural information processing systems*, 32, 2019.

Xisen Jin, Arka Sadhu, Junyi Du, and Xiang Ren. Gradient-based editing of memory examples for online task-free continual learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29193–29205. Curran Associates, Inc., 2021.

Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D. Yoo. Forget-free continual learning with winning subnetworks. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 10734–10750, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2017.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *International Conference on Learning Representations*, 2020.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2016.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE international conference on neural networks*, pp. 586–591. IEEE, 1993.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Diploma thesis, Technische Universität München, 1987.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pp. 4548–4557. PMLR, 2018.

Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021.

Jaehyeon Son, Soochan Lee, and Gunhee Kim. When meta-learning meets online and continual learning: A survey. *arXiv preprint arXiv:2311.05241*, 2023.

Paul Tseng. An incremental gradient (-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, 8(2):506–531, 1998.

Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Anna Vettoruzzo, Mohamed-Rafik Bouguelia, Joaquin Vanschoren, Thorsteinn Rognvaldsson, and KC Santosh. Advances and challenges in meta-learning: A technical review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Max Vladymyrov, Andrey Zhmoginov, and Mark Sandler. Continual few-shot learning using hypertransformers. *arXiv preprint arXiv:2301.04584*, 2023.

Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2019.

Johannes Von Oswald, Dominic Zhao, Seijin Kobayashi, Simon Schug, Massimo Caccia, Nicolas Zucchet, and João Sacramento. Learning where to learn: Gradient sparsity in meta and continual learning. *Advances in Neural Information Processing Systems*, 34:5250–5263, 2021.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pp. 3751–3760. PMLR, 2017.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. PMLR, 2017.

A Appendix

A.1 Experimental details

For training the models, we use the same dataset splits described in Jin et al. (2021). During training, we select $K = 5$ samples per class for the support set and we utilize a maximum of 100 examples per class for the query set. For each task, the model is trained for 10^5 steps. We set the number of adaptation steps, Q in Algorithm 2, to three for both SplitMNIST and RotatedMNIST, and to five for SplitCIFAR-100. However, during test time, we perform 50 adaptation steps to achieve the reported results. The SGD optimizer is used in the *Adapt&Optimize* algorithm (Algorithm 2) with a learning rate of 10^{-3}, while the Adam optimizer is used for updating the parameters θ, in line 12 of Algorithm 1, with a learning rate of 10^{-4} and a weight decay value of 10^{-5}. To update the transformer's parameter, $\psi \in \theta$, we utilize a learning rate schedule comprising a warmup phase of 3000 steps followed by linear decay, as recommended by Vaswani et al. (2017). We also employ an L2 regularizer to avoid exploding gradients.

Due to the large variety of baseline methods selected for performance comparison, we try to be as consistent as possible with the methodology proposed in the original papers, utilizing the default values for the parameters specific to each method. For the test setting, we adopt the same approach used in the original papers (Kirkpatrick et al., 2017; Zenke et al., 2017; Gupta et al., 2020; Von Oswald et al., 2021; Kang et al., 2022; Rebuffi et al., 2017; Buzzega et al., 2020), meaning that only in La-MAML (Gupta et al., 2020) and Sparse-MAML (Von Oswald et al., 2021) the model is fine-tuned on a subset of the test task, before making predictions on the remaining test data. Results do not show a considerable improvement for these methods either considering the BWT and the FWT. While fine-tuning for enough time on the new task could improve the model performance, mostly considering the FWT metric, we hypothesize that fine-tuning only for a few steps (e.g., 50 in our experiments) does not influence the final results.

Two distinct classifier architectures are used in the experiments, with similar architectures as the ones proposed in La-MAML (Gupta et al., 2020), Sparse-MAML (Van der Maaten & Hinton, 2008), and WSN (Kang et al., 2022), which are the closest to our approach. For the SplitMNIST and RotatedMNIST datasets, we utilize a simple classifier, including a convolutional layer with 32 filters of size 3, followed by batch normalization, LeakyReLU, max pooling with size 2, and a classifier head. A 3-layer CNN is used for SplitCIFAR-100 comprising three convolutional layers with 100 filters and a kernel size of 3, also followed by batch normalization, LeakyReLU, and max pooling, and concludes with a classifier head. The architecture of the meta-optimizer comprises a task encoder (part A), a feature extractor (part B), and a transformer encoder (part C), as described in Section 4.1. The task encoder includes a pre-trained feature extractor, followed by an averaging layer to compute the mean of the extracted features, yielding a single vector representing the support set of the task. For SplitMNIST and RotatedMNIST experiments, the task encoder is pre-trained on the SVHN dataset, and it comprises two hidden layers with 100 neurons. Conversely, for SplitCIFAR-100 experiments, the convolutional part of VGG-11, pre-trained on ImageNet, is employed. The second component of the meta-optimizer (i.e., part B) is a shallow network consisting of a single hidden layer with 16 neurons. Finally, the transformer encoder consists of two stacked transformer encoder layers, each incorporating a multi-head self-attention block with four attention heads and a feed-forward network with a size of 64. We add positional embeddings to the input features using a learnable 1D encoding, similar to Dosovitskiy et al. (2020). The transformer layers are trained using the GELU activation function and dropout regularization with a probability of 0.2. Additionally, we apply a hyperbolic tangent operation within the range $[-3, 3]$ on top of the transformer encoder to ensure the output does not become too large.

A.2 Ablation studies on the meta-optimizer

Due to the complexity of the meta-optimizer, we conduct several ablation studies to verify the significance of each component in achieving the final performance. These ablation studies are performed by removing one component at a time while keeping the other parts consistent, as outlined below:

- The pre-trained task encoder g_A in Figure 1b is replaced by a simpler model comprising of a flattening operation, an averaging layer over the KN examples in the input support set, and a simple linear layer that transforms the average representation into a size suitable for the transformer model.
- The feature extractor g_B in Figure 1b is modified with a simple linear layer that transforms the input weights into a size suitable for the transformer encoder.
- The transformer encoder is replaced by an LSTM network with two LSTM layers with the same hidden size as the transformer encoder, followed by a linear layer and a hyperbolic tangent operation in the range $[-3, 3]$, as used with the transformer encoder.
- The importance scores are ablated by simply removing them from the proposed approach.

Table 3: Ablation studies on the different components of the meta-optimizer for SplitMNIST and SplitCIFAR-100. The first column shows the ablated component and results show the average accuracy ("Avg Acc"), BWT and FWT metric as described in Section 5.1 .

Ablated component	**SplitMNIST**			**SplitCIFAR-100**		
	Avg Acc	BWT	FWT	Avg Acc	BWT	FWT
Task encoder	93.18	−8.55	88.24	47.71	−16.31	48.37
Feature extractor	92.56	−10.56	82.18	47.01	−17.24	43.76
Transformer encoder	70.80	−27.22	90.52	25.45	−2.22	25.59
Importance scores	84.26	−16.15	80.40	43.17	−18.90	41.78

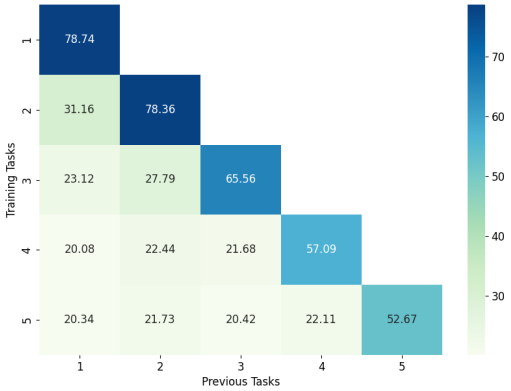


Figure 6: Accuracy matrix at test time for the transfer learning baseline with SplitCIFAR-100. Each cell of the matrix a_{ij} represents the accuracy of task $\mathcal{T}_j$ when the last task seen by the model is $\mathcal{T}_i$.

Table 4: Results for the transfer learning baseline with SplitCIFAR-100 dataset using a VGG11 model pre-trained on ImageNet and a classifier model on top of it.

Transfer learning	
Avg accuracy	27.45
BWT	−50.74
FWT	41.97

The results in Table 3 indicate that while the task encoder and the feature extractor contribute marginally to the final performance, the transformer encoder and the importance scores are necessary for the proposed approach. This finding aligns with the expectations, as the transformer encoder, unaffected by long dependency issues owing to the non-sequential input processing, plays a crucial role in learning to optimize for CL (Chen et al., 2022). Additionally, the self-attention mechanism and the positional embeddings provide even more information about the relationship between the input weights, which is fundamental in the proposed approach. The other essential component is the importance scores. Containing information related to the gradient of the loss function they help the transformer encoder to learn to optimize only the classifier weights that are more important for the current task. This mitigates catastrophic forgetting of the model. Notably, removing the importance scores results in a significant decrease in the BWT metric compared to the proposed approach, underscoring their pivotal contribution. It is also interesting to notice that removing the importance scores causes a big drop in the FWT metric, suggesting that learning to optimize all the model weights ends up in a model too tailored to the current task that cannot easily be generalized to new tasks.

A.3 Transfer learning baseline

A simple transfer learning baseline is implemented to verify that solely pre-training the task encoder g_A in the meta-optimizer (see Figure 1b) cannot directly address the problem outlined in this paper. Specifically, we concatenate the task encoder pre-trained on ImageNet with the classifier model used with SplitCIFAR-100, and we fine-tune only the classifier on the support set of the tasks. To ensure fairness with other baseline methods proposed in this paper, we fine-tune the classifier model for 10^5 steps using the Adam optimizer with a learning rate set to 10^{-4} and a weight decay value of 10^{-5} for the current task observed by the model. Subsequently, we evaluate it on previous and future tasks by fine-tuning the model with only 50 adaptation steps. Results in Figure 6 and Table 4 demonstrate that while the model achieves high performance on the last seen task (in the diagonal of the matrix in Figure 6), it fails to generalize to previous or future tasks, as highlighted by the low BWT and FWT metrics. This outcome is expected since mere application of transfer learning is inadequate to effectively retain previous knowledge and generalize to future tasks.

A.4 LIMITATIONS AND SOLUTIONS

While our approach of meta-learning a meta-optimizer using attention demonstrates promising results in addressing CL challenges, it is important to acknowledge some inherent limitations. Firstly, the tight coupling between the meta-optimizer and the classifier imposes constraints on the applicability of the trained meta-optimizer to novel scenarios requiring different classifier architectures. The meta-optimizer is explicitly designed to work well for the classifier it was trained for, necessitating retraining or fine-tuning for different classifier architectures. We have demonstrated in Appendix A.2 that the feature extractor is not a primary contributor to the model performance. Therefore simply replacing its architecture with a more appropriate one for the number of weights $W = \{w_r\}_{r=1}^{R}$ that need to be learned to optimize would be sufficient to extend the proposed approach to more complex classifier architectures. Another potential issue that might arise with a deeper classifier is the scalability of the proposed approach, as the size of the input weights to the meta-optimizer increases substantially, leading to computational overheads. One potential mitigation strategy involves focusing on optimizing only the last layers, as they encode more task-specific information compared to earlier layers (Ramasesh et al., 2020). Furthermore, the computational demands associated with second-order derivatives in meta-learning, coupled with training times and data requirements for transformers, pose practical challenges for training such a model. Lastly, the fixed size of the output classifier head limits adaptability in scenarios with varying numbers of classes, such as in CIL. While this is not the scope of the proposed approach, future work aims to extend it to accommodate a dynamic classifier head capable of predicting all encountered classes.

A.5 DEFAULT NOTATION

$\mathcal{T}_m$	A task
$\vec{\mathcal{T}} = \{\mathcal{T}_m\}_{m=1}^{M}$	A stream of tasks
$p(\mathcal{T})$	A task distribution
$p_m(x, y)$	A data distribution
D_m^{train}	Training data sampled from $\mathcal{T}_m$
D_m^{test}	Test data sampled from $\mathcal{T}_m$
$D_{i,m}$	A batch of data
$D_{i,m}^{(sp)}$	A support set of data
$D_{i,m}^{(qr)}$	A query set of data
$\{x_k\}_{k=1}^{KN} \in D_{i,m}^{(sp)}$	Data in the support set
K	K-shot classification
N	N-way classification
$f_{\mu,W}$	Classifier network
g_ψ	Meta-optimizer network
ψ	Parameters of the meta-optimizer
μ	Meta-learned parameters of the classifier network
$W = \{w_r\}_{r=1}^{R}$	Set of classifier's parameters to meta-optimize
R	Number of classifier's parameters to meta-optimize
$\theta = \{\mu, W\}$	Parameters of the classifier network
$\{s_r\}_{r=1}^{R}$	Importance scores
$\Delta W = \{\Delta w_r\}_{r=1}^{R}$	Weight updates for the meta-optimized weights
θ'	Updated parameters with a fixed optimizer
$\tilde{W}$	Updated parameters with the meta-optimizer
α, β	Learning rates
Q	Adaptation steps