Overcoming Catastrophic Forgetting via Direction-Constrained Optimization

Yunfei Teng[1], Anna Choromanska[1⋆], Murray Campbell[2], Songtao Lu[2], Parikshit Ram[2], and Lior Horesh[2]

[1] New York University, USA
[2] IBM Research, USA

Abstract. This paper studies a new design of the optimization algorithm for training deep learning models with a *fixed* architecture of the classification network in a continual learning framework. The training data is non-stationary and the non-stationarity is imposed by a sequence of distinct tasks. We first analyze a deep model trained on only one learning task in isolation and identify a region in network parameter space, where the model performance is close to the recovered optimum. We provide empirical evidence that this region resembles a cone that expands along the convergence direction. We study the principal directions of the trajectory of the optimizer after convergence and show that traveling along a few top principal directions can quickly bring the parameters outside the cone but this is not the case for the remaining directions. We argue that catastrophic forgetting in a continual learning setting can be alleviated when the parameters are constrained to stay within the intersection of the plausible cones of individual tasks that were so far encountered during training. Based on this observation we present our direction-constrained optimization (DCO) method, where for each task we introduce a linear autoencoder to approximate its corresponding top forbidden principal directions. They are then incorporated into the loss function in the form of a regularization term for the purpose of learning the coming tasks without forgetting. Furthermore, in order to control the memory growth as the number of tasks increases, we propose a memory-efficient version of our algorithm called compressed DCO (DCO-COMP) that allocates a memory of fixed size for storing all autoencoders. We empirically demonstrate that our algorithm performs favorably compared to other state-of-art regularization-based continual learning methods. The codes are publicly available at `https://github.com/yunfei-teng/DCO`.

Keywords: Continual / Lifelong Learning · Deep Learning · Optimization.

1 Introduction

A key characteristic feature of intelligence is the ability to continually learn over time by accommodating new knowledge and transferring knowledge between

⋆ Senior lead.

correlated tasks while retaining previously learned experiences. This ability is often referred to as *continual or lifelong learning*. In a continual learning setting one needs to deal with a continual acquisition of incrementally available information from non-stationary data distributions (online learning) and avoid *catastrophic forgetting* [28], i.e., a phenomenon that occurs when training a model on currently observed task leads to a rapid deterioration of the model's performance on previously learned tasks. In the commonly considered scenario of continual learning the tasks come sequentially and the model is not allowed to inspect again the samples from the tasks seen in the past [29]. Within this setting, there exist two types of approaches that are complementary and equally important in the context of solving the continual learning problem: i) methods that assume fixed architecture of deep model and focus on designing the training strategy that allows the model to learn many tasks and ii) methods that rely on existing training strategies (mostly SGD [5] and its variants, which themselves suffer catastrophic forgetting [12]) and focus on expanding the architecture of the network to accommodate new tasks. In this paper we focus on the first framework.

Training a network in a continual learning setting, when the tasks arrive sequentially, requires solving many optimization problems, one per task. A space of solutions (i.e., network parameters) that correspond to good performance of the network on all encountered tasks determine a common manifold of plausible solutions for all these optimization problems. In this paper we seek to understand the geometric properties of this manifold. In particular we analyze how this manifold is changed by each new coming task and propose an optimization algorithm that efficiently searches through it to recover solutions that well-represent all previously-encountered tasks. The contributions of our work could be summarized as follows:

- We empirically analyse the deep learning loss landscape and show that there is a cone in the network parameter space where the model performance is close to the recovered optimum.
- We propose a new regularization-based continual learning algorithm that explicitly encourages the model parameters to stay inside the plausible cone by identifying a few top forbidden principal directions for each task.
- We propose an autoencoder architecture that significantly reduces the memory complexity to save the top forbidden principal directions for a given task.
- We design a compression method to control the memory growth and avoid introducing a new autoencoder per task, thereby requiring only a constant size memory overhead irrespective of the number of tasks.

The paper is organized as follows: Section 2 reviews recent progress in the research area of continual learning, Section 3 provides empirical analysis of the geometric properties of the deep learning loss landscape and builds their relation to the continual learning problem, Section 4 introduces our algorithm that we call DCO since it is based on the idea of direction-constrained optimization, Section 5 contains empirical evaluations, and finally Section 6 concludes the paper. Additional results are contained in the Supplement.

2 Related Work

Continual learning and the catastrophic forgetting problem has been addressed in a variety of papers. A convenient literature survey dedicated to this research theme was recently published [29]. The existing approaches can be divided into three categories [29,10] listed below.

Regularization-based methods modify the objective function by adding a penalty term that controls the change of model parameters when a new task is observed. In particular these methods ensure that when the model is being trained on a new task, the parameters stay close to the ones learned on the tasks seen so far. EWC [18] approximates the posterior of the model parameters after each task with a Gaussian distribution and uses tasks' Fisher information matrices to measure the overlap of tasks. The idea is extended in [33] where the authors introduce Kronecker factored Laplace approximation. SI [42] introduces the notion of synaptic importance, enabling the assessment of the importance of network parameters when learning sequences of classification tasks, and penalizes performing changes to the parameters with high importance when training on a new task in order to avoid overwriting old memories. Relying on the importance of the parameters of a neural network when learning a new task is also a characteristic feature of another continual learning technique called MAS [1]. The RWALK method [6] is a combination of an efficient variant of EWC and a modified SI technique that computes a parameter importance score based on the sensitivity of the loss over the movement on the Riemannian manifold. Additionally, RWALK stores a small subset of representative samples from the previous tasks and uses them while training the current task, which is essentially a form of a replay strategy described later in this section. The recently proposed OGD algorithm [9] and its variant GPM [36] rely on constraining the parameters of the network to move within the orthogonal space to the gradients of previous tasks. [9] is memory-consuming and not scalable as it requires saving the gradient directions of the neural network predictions on previous tasks. Finally, the recursive method of [24] modifies the gradient direction of each step to minimize the expected forgetting by introducing an additional projection matrix which requires a per-step update with linear memory complexity in the number of model parameters. All methods discussed so far constitute a family of techniques that keep the architecture of the network fixed. The algorithm we propose in this paper also belongs to this family.

Another regularization method called LwF [23] optimizes the network both for high accuracy on the next task and for preservation of responses on the network outputs corresponding to the past tasks. This is done using only examples for the next task. The encoder-based lifelong learning technique [30] uses per-task under-complete autonecoders to constraint the features from changing when the new task arrives, which has the effect of preserving the information on which the previous tasks are mainly relying. Both these methods fundamentally differ from the aforementioned techniques and the approach we propose in this paper in that they require a separate network output for each task. Finally, P&C [37] builds upon EWC and takes advantage of the knowledge distillation mechanism

to preserve and compress the knowledge obtained from the previous tasks. Such a mechanism could as well be incorporated on the top of SI, MAS, or our technique.

The next two families of continual learning methods are not directly related to the setting considered in this paper and are therefore reviewed only briefly.

Dynamic architecture methods either expand the model architecture [2,35,41,22] to allocate additional resources to accommodate new tasks (they are typically memory expensive) or exploit the network structure by parameter pruning or masking [26,27]. Some techniques [16] interleaves the periods of network expansion with network compression, network pruning, and/or masking phases to better control the growth of the model.

Replay methods are designed to train the model on a mixture of samples from a new task and samples from the previously seen tasks. The purpose of replaying old examples is to counter-act the forgetting process. Many replay methods rely on the design of sampling strategies [17,3]. Other techniques, such as GEM [25], A-GEM [7] and MER [32], use replay specifically to encourage positive transfer between the tasks (increasing the performance on preceding tasks when learning a new task). ORTHOG-SUBSPACE [8] reduces the interference between tasks by learning the tasks in different subspaces. Replay methods typically require large memory. Deep generative replay technique [38,34] addresses this problem and employs a generative model to learn a mixed data distribution of samples from both current and past tasks. Samples generated this way are used to support the training of a classifier. Finally, note that the setting considered in our paper does not rely on the replay mechanism.

In addition to the above discussed research directions, very recently authors started to look at task agnostic and multi-task continual learning where no information about task boundaries or task identity is given to the learner [31,14,43,15,40]. These approaches lie beyond the scope of this work.

Remark: Regularization-based methods and replay methods are usually implemented with a fixed architecture of the classification network, but they require additional memory to save regularization terms or data samples. Conversely, dynamic architecture methods do not explicitly keep extra information in the memory, but they rely on the expansion and modification of the network architecture itself. Our approach falls into the family of regularization-based methods since we do not allow the architecture of the classification network to dynamically change and we also do not allow replay.

3 Loss landscape properties

The experimental observations provided in this section extend and complement the behavior characterization of SGD [11] connecting its dynamics with random landscape theory that stems from physical systems. The results that will be presented here were obtained on MNIST and CIFAR-10 data sets (CIFAR-10 results are deferred to the Supplement). The details of the experimental setup of this section can be found in the Supplement (Section 9). Consider learning only one task. We analyze the top principal components of the trajectory of SGD

after convergence, i.e., after the optimizer reached a saturation level[3]. Let x^* denotes the value of the parameters in the beginning of the saturation phase. The convergence trajectory will be represented as a sequence of optimizer steps, where each step is represented by the change of model parameters that the optimizer induced (gradient). We consider n steps after model convergence and compute the gradient of the loss function at these steps that we refer to as $\nabla L(x_1;\zeta_1), \nabla L(x_2;\zeta_2), \ldots, \nabla L(x_n;\zeta_n)$ (x_i denotes the model parameters at the i^{th} step and ζ_i denotes the data mini-batch for which the gradient was computed at that step). We use them to form a matrix $G \in \mathbb{R}^{d\times n}$ (i-th column of the matrix is $\nabla L(x_i;\zeta_i)$) and obtain the eigenvectors $\{v_i\}$ of GG^T.[4] We furthermore define the averaged gradient direction $\bar{g} = \frac{1}{n}\sum_{i=1}^{n} \nabla L(x_i;\zeta_i)$. We first study the landscape of the deep learning loss function along directions v_i and $\bar{g}$, i.e., we analyze the function

$$f(\alpha, \beta, v_i) = L(x^* - \alpha\bar{g} + \beta v_i; \zeta), \tag{1}$$

where α and β are the step sizes along $-\bar{g}$ and v_i respectively and ζ denotes the entire training data set. We will show how this function is connected to our algorithm later in section 4.4.

Remark: Below, the eigenvector with the lower-index corresponds to a larger eigenvalue.

Observation 1: Behavior of the loss for $\alpha = 0$ and changing β For each eigenvector v_i, we first fix α to 0 and change β in order to study the behavior of $f(0, \beta, v_i)$. Fig. 1a captures the result. It can be observed that as the model parameters move away from the optimal point x^* the loss gradually increases. At the same time, the rate of this increase depends on the eigendirection that is followed and grows faster while moving along eigenvectors with the lower-index. Thus we have empirically shown that *the loss changes more slowly along the eigenvectors with the high index, i.e., the landscape is flatter along these directions.*

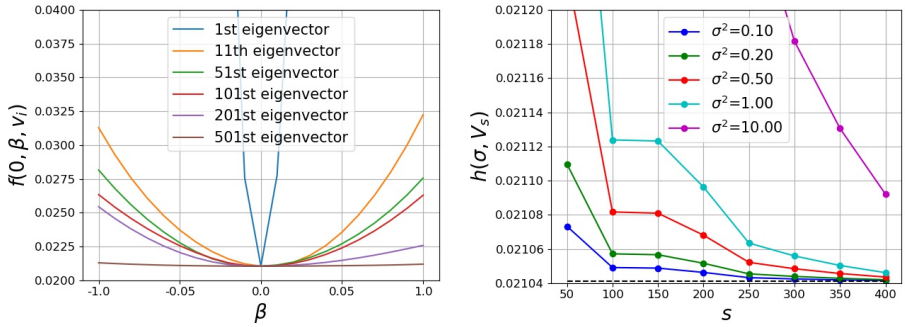
Observation 2: Behavior of the loss in the subspaces spanned by groups of eigenvectors Here we generalize Observation 1 to the subspaces spanned by a set of eigenvectors. For the purpose of this observation only we consider the following metric instead of the one given in Equation 1:

$$h(\sigma, V_s) = \mathbb{E}_{\delta\sim\mathcal{N}(0,\frac{\sigma^2}{d}I)} L(x^* + V_s V_s^T \delta; \zeta), \tag{2}$$

where δ is the random perturbation, σ is the standard deviation, and $V_s = [v_{s-49}, v_{s-48}, \cdots, v_s]$ is the matrix of eigenvectors of 50 consecutive indexes. To be more concrete, we locally (in the ball of radius σ around x^*) sample the space spanned by the eigenvectors in V_s. The expectation is computed over 3000

[3] The optimization process is typically terminated when the loss starts saturating but we argue that running the optimizer further gives benefits in the continual learning setting.

[4] The explanation of the difference between GG^T and the Fisher information matrix underlying the EWC method is deferred to the Supplement (Section 8).



(a) Fixing $\alpha = 0$ and varying β. (b) Varying both σ and s.

Fig. 1: **left (a)**: The behavior of the loss function for $\alpha = 0$ and varying β when moving along different eigenvectors on MNIST (the complementary plot obtained on CIFAR-10 can be found in the Supplement, Fig. 9); **right (b)**: The behavior of the loss function when varying σ and s on MNIST (the complementary plot obtained on CIFAR-10 can be found in the Supplement, Fig. 10).

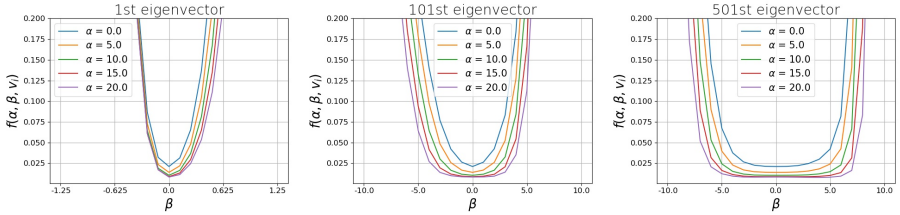


Fig. 2: The behavior of the loss function when both α and β are changing for eigenvectors with different index on MNIST (the complementary plot obtained on CIFAR-10 can be found in the Supplement, Fig. 11).

random draws of δ. In Fig. 1b we examine the behavior of $h(\sigma, V_s)$ for various values of σ and s. The plot confirms what was shown in Observation 1 that the loss landscape becomes flatter in the subspace spanned by the eigenvectors with high index.

Observation 3: Behavior of the loss for changing α and β We generalize Observation 1 and examine what happens with $f(\alpha, \beta, v_i)$ when both α and β change. Fig. 2 captures the result. We can see that as α increases, or in other words *as we go further along the averaged gradient direction, the loss landscape becomes flatter. This property holds for an eigenvector with an arbitrary index.* Thus for larger values of α we can go further along eigenvector directions without significantly changing the loss. This can be seen as a *cone* that expands along $-\bar{g}$. Furthermore, the findings of Observation 1 are also confirmed in Fig. 2. For the eigenvectors with higher index the loss changes less rapidly (the cone

is wider along these directions). These properties underpin the design of new continual learning algorithm proposed in this work. When adding the second task, the algorithm constrains the optimizer to stay within the cone of the first task. Intuitively this can be done by first pushing the optimizer further into the cone along $-\bar{g}$ and then constraining the optimizer from moving along eigenvectors with low indexes in order to prevent forgetting the first task. This procedure can be generalized to an arbitrary number of tasks as will be shown in the next section.

4 Algorithm

In Section 3 we analyzed the loss landscape for a single task and discovered the existence of the cone in the model's parameter space where the model sustains good performance. We then discussed the consequence of this observation in the continual learning setting. In this section we propose a *tractable* continual learning algorithm that for each task finds its cone and uses it to constrain the optimization problem of learning the following tasks. We refer to the model that is trained in the continual learning setting as $\mathcal{M}$. The proposed algorithm relies on identifying the top directions along which the loss function for a given task increases rapidly and then constraining the optimization from moving along these directions (we will refer to these directions as "prohibited") when learning subsequent tasks. Note that each new task adds prohibited directions. In order to efficiently identify and constrain the prohibited directions we use *reduced linear autoencoders* whose design was tailored for the purpose of the proposed algorithm. We train separate autoencoders for each learned task. The j^{th} autoencoder admits on its input gradients of the loss function that are obtained when training the model $\mathcal{M}$ on the j^{th} task. The intuitive idea behind this approach is that autoencoder with small feature vector will capture the top directions of the gradients it is trained on. We refer to our method as *direction-constrained optimization* (DCO) method. Furthermore, we will show that we can relax the need to allocate new memory for each task and propose a memory-efficient version of our algorithm called *compressed* DCO (DCO-COMP) which requires a memory of fixed size for storing all autoencoders.

4.1 Loss function

We next explain the loss function that is used to train the model $\mathcal{M}$ in a continual learning setting. From Section 3, we recognize that the matrix G formed by the gradients obtained from the current task can be used to describe the properties of the loss landscape. Furthermore, we can incorporate it as a regularization term into the loss function to prevent the increment of loss for the current task when training for a future task. Therefore, the loss function that is used to train the model on the i^{th} task takes the form:

$$L_i(x;\xi) = L_{ce}(x;\xi) + \lambda \sum_{j=1}^{i-1} \left\| (G^j)^T (x - x_j^*) \right\|_2^2, \tag{3}$$

where ξ is a training example, L_{ce} is a cross-entropy loss, λ is a hyperparameter controlling the strength of the regularization, G^j is the regularization matrix whose columns are the sampled gradients, and x_j^* are the parameters of model $\mathcal{M}$ obtained at the end of training the model on the j^{th} task. However, directly saving matrix G^j will make the algorithm become *intractable*. Thus, we instead introduce an autoencoder ENC^j to approximate Eq. 3 by:

$$L_i(x;\xi) = L_{ce}(x;\xi) + \lambda \sum_{j=1}^{i-1} \left\| ENC^j(x - x_j^*) \right\|_2^2, \tag{4}$$

where $ENC^j(\cdot)$ is the operation of the encoder of the autoencoder trained on task j. The linear autoencoders with appropriate regularization are able to recover the principal components of gradients [4]. The top principal components correspond to the directions where the loss changes the quickest and we consider these directions as the prohibited directions. This is well-aligned with our observations from Section 3.

4.2 Reduced linear autoencoders

In our algorithm, the role of autoencoder is to identify the top k directions of the optimizer's trajectory after convergence, where this trajectory is defined by gradient steps, obtained during training the model $\mathcal{M}$. A traditional linear autoencoder, consisting of two linear layers, would require $2 \times d \times k$ number of parameters, where d denotes the number of parameters of the model $\mathcal{M}$. Commonly used deep learning models however contain millions of parameters [20,39,13], which makes a traditional autoencoder not tractable for this application. In order to reduce the memory footprint of the autoencoder we propose an architecture that is inspired by the singular value decomposition. The proposed autoencoder admits a matrix on its input and is formulated as

$$AE(M) = U diag(U^\top M V) V^\top, \tag{5}$$

where $diag(U^\top MV)$ is a matrix formed by zeroing out the non-diagonal elements of $U^\top MV$, M is an autoencoder input matrix of size $m \times n$, and U and V are autoencoder parameters of size $m \times k$ and $n \times k$ respectively. Thus, the total number of parameters of the proposed autoencoder is $k(n+m)$, which is significantly lower than in case of traditional autoencoder (knm), especially when n and m are large. We call this architecture a *reduced linear autoencoder*.

We use a separate encoder ENC_l and decoder DEC_l for each layer l of the model $\mathcal{M}$. We couple them between layers using a common "feature vector" which is created by summing outputs of all encoders. This way the feature vector will contain the information from all layers. The proposed autoencoder is then

Algorithm 1 DCO/DCO-COMP Algorithm

Require: η and η_a: learning rates of the model and autoencoders respectively. $\gamma_1, \gamma_2 \in (0, 1]$: pulling strengths that controls the searching scope of the model parameters. N: number of additional epochs used to train the model after saturation. C: number of points to average. θ: step size for pushing the optimizer inside the cone ($\theta \leq 1$ corresponds to parameter interpolation; $\theta > 1$ corresponds to parameter extrapolation). m: the size of the batch of gradients fed into autoencoders. τ: the period of updates of the model parameters in step 3. n: number of tasks. $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$: training data from task $1, 2, \ldots, n$. $|\mathcal{T}_i|$: number of iterations (mini-batches) required to process all data samples from task i.

Procedure:

for $i = 1$ **to** n **do**

step 1: train model until convergence

$x_0 \leftarrow x$

repeat

$\xi \leftarrow$ randomly sample from $\mathcal{T}_i$

$x \leftarrow x - \eta \nabla_x L_i(x; \xi) - \gamma_1 (x - x_0)$

until convergence

step 2: push the model parameters into the cone

$x_1 \leftarrow 0, x_2 \leftarrow 0$

for $j = 1$ **to** $N \times |\mathcal{T}_i|$ **do**

$\xi \leftarrow$ randomly sample from $\mathcal{T}_i$

$x \leftarrow x - \eta \nabla_x L_{ce}(x; \xi)$

if $j \leq C$ **then** $x_1 \leftarrow x_1 + x$

else if $j > N \times |\mathcal{T}_i| - C$ **then** $x_2 \leftarrow x_2 + x$

end for

$x_i^* \leftarrow x_1 - \theta \frac{(x_1 - x_2)}{\|x_1 - x_2\|}$ {push into the cone}

step 3: train autoencoder until convergence

repeat

$g \leftarrow 0, G \leftarrow \{\}$

for $j = 1$ **to** m **do**

$\xi \leftarrow$ randomly sample from $\mathcal{T}_i$

$g \leftarrow g + \nabla_x L_{ce}(x; \xi)$; $G \leftarrow G \cup \nabla_x L_{ce}(x; \xi)$

if τ **divides** j **then** $x \leftarrow x - \eta g$; $g \leftarrow 0$

end for

$G \leftarrow \frac{G}{\sqrt{\|G\|_2^2 / m}}$ {Normalize batch of gradients}

$W \leftarrow W - \frac{\eta_a}{m} \nabla_W L_{mse}(W; G)$; $x \leftarrow x - \gamma_2 (x - x_i^*)$

until convergence

step 4: store autoencoder parameters

$W^i \leftarrow W$

if use DCO-COMP **then**

compress and update $\{W^1, \cdots, W^i\}$ by Equation 15 and Equation 16.

end if

end for

Output: x_n^*

formulated as

$$\begin{aligned} AE(G) &= \{DEC_1(ENC(G)), \\ &\quad \ldots, DEC_L(ENC(G))\}, \end{aligned} \tag{6}$$

where

$$ENC(G) = \sum_l ENC_l(G_l), \tag{7}$$

$$ENC_l(G_l) = diag(U_l^\top G_l V_l), \tag{8}$$

$$DEC_l(ENC(G)) = U_l ENC(G) V_l^\top, \tag{9}$$

$G = \{G_1, G_2, \ldots, G_L\}$ is a set of matrices such that each matrix contains gradients of the model for a given layer, and L is number of layers in the model. Finally, in order to enable processing the gradients of the convolutional layers we reshape them from their original size $o \times i \times w \times h$ to $o \times iwh$, where o is number of output channels, i is number of input channels, and w and h are width and height of the kernel of the convolutional layer. We train the autoencoder with standard mean square error loss

$$L_{mse}(W; G) = \|AE(G) - G\|_2^2, \tag{10}$$

where $W = \{U_1, V_1, \ldots, U_L, V_L\}$ is set of autoencoder's parameters. In the next section we propose a memory efficient variant DCO-COMP and show a compression scheme which allows us to avoid scaling the memory size as the number of tasks increases and results in a solution with a fixed memory size.

Remark: Using one autoencoder per task in a continual learning context has been explored in the literature before. For example, [30] uses an autoencoder to capture the features that are crucial for its corresponding task. Authors show experiments for only two tasks. Another method [2] embeds autoencoders into the classification network to identify the tasks and make predictions. How do we differ from these approaches? First, we utilize autoencoders to encode optimizer directions. Second, as opposed to [2] the autoencoders are not used within the classification network, thus they are not utilized at testing, but only at training. Third, as opposed to [30] we demonstrate experiments on multiple tasks. Finally, note that autoencoders have not been used before to support parameter-wise regularization-based continual learning frameworks.

4.3 Compression of autoencoders

To avoid scaling the memory size as the number of tasks increases, we compress the autoencoders recursively so that only a *constant* memory of size $k \times (m + n)$ is required to store all autoencoders during training. More specifically, after training on the i^{th} task, all autoencoders are compressed together such that each autoencoder keeps $\frac{1}{2^i} \times k \times (m + n)$ parameters separately. On top of that, by introducing shared parameters across autoencoders, whose size is fixed to $\frac{1}{2} \times k \times (m + n)$, we ensure that the information that would be lost due to

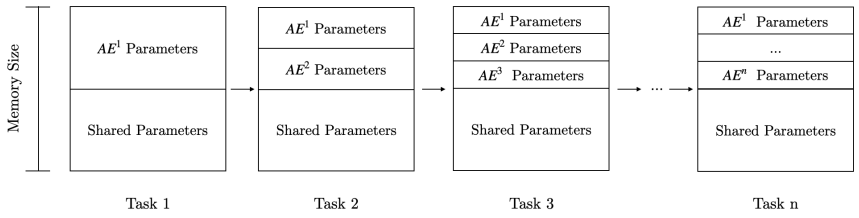


Fig. 3: The memory size of the autoencoders remains unchanged across the tasks.

compression is instead partially absorbed by the shared parameters. The memory allocation of autoencoders on each task is illustrated in Fig 3.

Denote the shared parameters and the j^{th} compressed autoencoder's parameters as $\bar{W} = \{\bar{U}_1, \bar{V}_1, \ldots, \bar{U}_L, \bar{V}_L\}$ and $\tilde{W}^j = \{\tilde{U}_1^j, \tilde{V}_1^j, \ldots, \tilde{U}_L^j, \tilde{V}_L^j\}$, respectively. Similar as before, we use a separate encoder $\widehat{ENC}_l$ and decoder $\widehat{DEC}_l$ for each layer l. The formulation of the j^{th} compressed autoencoder is given as

$$\begin{aligned} \widehat{AE}^j(W^j) &= \{\widehat{DEC}_1^j(\widehat{ENC}^j(W^j)), \\ &\quad \ldots, \widehat{DEC}_L^j(\widehat{ENC}^j(W^j))\}, \end{aligned} \tag{11}$$

where

$$\widehat{ENC}^j(W_l^j) = \sum_l \widehat{ENC}_l^j(W_l^j), \tag{12}$$

$$\widehat{ENC}_l^j(W_l^j) = diag\left(\bar{U}_l^\top U_l^j (V_l^j)^\top \bar{V}_l + (\tilde{U}_l^j)^\top U_l^j (V_l^j)^\top \tilde{V}_l^j\right), \tag{13}$$

$$\widehat{DEC}_l^j(\widehat{ENC}^j(W^j)) = \tilde{U}_l^j \widehat{ENC}^j(W^j)(\tilde{V}_l^j)^\top \tag{14}$$

where $W^j = \{U_1^j, V_1^j, \ldots, U_L^j, V_L^j\}$ is set of j^{th} autoencoder's uncompressed parameters. We train the compressed autoencoders with standard mean square error loss:

$$L_{mse}(\tilde{W}^1, \cdots, \tilde{W}^i, \bar{W}; W^1, \cdots, W^i) = \sum_{j=1}^{i} \left\| \widehat{AE}^j(W^j) - W^j \right\|_2^2 \tag{15}$$

Then we assign the j^{th} autoencoder with a new set of parameters:

$$W^j = \{(\bar{U}_1, \tilde{U}_1^j), (\bar{V}_1, \tilde{V}_1^j), \cdots, (\bar{U}_L, \tilde{U}_L^j), (\bar{V}_L, \tilde{V}_L^j)\} \tag{16}$$

Remark: We are not able to re-access the model gradients from previous tasks anymore, but the learned prohibited directions for each task could be recovered from the corresponding autoencoder parameters. Thus we make compression directly on the autoencoder parameters.

4.4 Resulting algorithm

The proposed algorithm comprises of four steps. In the first step we train the model $\mathcal{M}$ using the loss function proposed in Equation 4 until convergence. This loss function penalizes moving along "prohibited" directions recovered for the previous tasks. In the second step, we continue to train the model for additional N epochs and make either interpolation or extrapolation (depending on the step size) between the averages of the first and the last C points on the optimizer's trajectory. This step is equivalent to pushing the model parameters deeper into the cone and aligns well with Section 3 (see conclusions resulting from Fig. 2). In the third step we train the autoencoder to recover "prohibited" directions for the current task. This again aligns well with Section 3 (see conclusions resulting from Fig. 1a and 1b). Finally, we store the autoencoder parameters with or without compression. The algorithm's pseudo code is captured in Algorithm 1.

5 Experiments

In this section we compare the performance of DCO and DCO-COMP with state-of-the-art regularization-based continual learning methods such as EWC [18], SI [42], RWALK [6] and GPM [36], as well as the vanilla SGD [5]. We use open source codes[5][6] for the experiments.

5.1 Data sets and architectures

In our experiments we consider commonly used continual learning data sets: (1) **Permuted MNIST**. For each task we used a different permutation of the pixels of images from the original MNIST data set [21]. We generated 5 data sets this way corresponding to 5 tasks. (2) **Split MNIST**. We divide original MNIST data set into 5 disjoint subsets corresponding to labels $\{\{0,1\},\{2,3\},\{4,5\},\{6,7\},\{8,9\}\}$. (3) **Split CIFAR-100**. We divide original CIFAR-100 data set [19] into 10 disjoint subsets corresponding to labels $\{\{0-9\},\cdots,\{90-99\}\}$. Additionally, we consider a cross-domain learning scenario, where tasks come from different domains. For a cross-domain learning experiment (**MNIST/Fashion MNIST**) we combine MNIST and Fashion MNIST data sets together.

For Permuted MNIST, Split MNIST, and MNIST/Fashion MNIST we use the original image of size $1\times 28\times 28$. We then normalize each image by mean (0.1307) and standard deviation (0.3081). For Split CIFAR-100, we use the original image of size $3\times 32\times 32$. We then normalize each image by mean $(0.5071, 0.4867, 0.4408)$ and standard deviation $(0.2675, 0.2565, 0.2761)$. Also, in the experiments with Split MNIST and Split CIFAR-100 we use a multi-head setup [42,6] and we provide task descriptors [25,7] to the model at both training and testing.

Finally, for Permuted MNIST and MNIST/Fashion MNIST experiments we use a Multi-Layer Perceptron (MLP) with two hidden layers, each having 256

[5] https://github.com/facebookresearch/agem
[6] https://github.com/sahagobinda/GPM

units with ReLU activation functions (we refer to this architecture as MLP-256). For Split MNIST, we use a MLP with two hidden layers each having 100 units with ReLU activation functions (we refer to this architecture as MLP-100). For Split CIFAR-100, we use the same convolutional neural network as in [6] (we refer to this architecture as ConvNet). In all architectures we turn off the biases.

5.2 Training details

To train the model, we use SGD optimizer [5] with momentum of 0.9. The batch sizes are set to 128, 128, 128 and 64 respectively for MNIST/Fashion MNIST, Permuted MNIST, Split MNIST and Split CIFAR-100.

For MNIST/Fashion MNIST, we use a constant learning rate of 1×10^{-3}. For Permuted MNIST and Split MNIST, we use a constant learning rate of 1×10^{-3} and add weight decay penalty of 0.001. For Split CIFAR-100, we use a learning rate of 1×10^{-2} for the first task and then drop it by a factor of 0.1 for the remaining tasks.

For DCO on Split CIFAR-100, we also clip the l_2 norm of the gradients induced by regularization terms with a threshold of 1 to avoid exploding gradient problem.

5.3 Hyperparameters

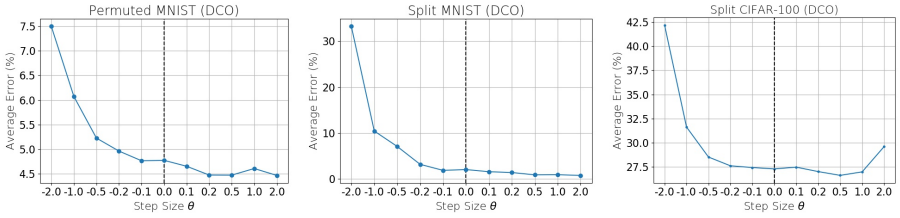


Fig. 4: Average error versus the step size θ (**left:** Permuted MNIST **middle:** Split MNIST **right:** Split CIFAR-100).

The values of the hyperparameters explored in the experiments are reported in the Supplement (Section 10.1). Here we illustrate how the final average error of DCO varies as the step size θ increases on Permuted MNIST, Split MNIST and Split CIFAR-100 (see Fig. 4). The figure further confirms our findings in Section 3 that when moving deeper inside the cone, the performance improves. After some point the performance however eventually starts dropping, as can been seen on the right plot. This is most likely because of falling outside the cone, due to either inaccurate estimation of the direction pointing towards the center of the cone or the fact that the cone is bounded.

5.4 Metric and Results

We denote $e_{i,j}$ as *test* classification error of the model on j^{th} task after getting trained on i tasks and evaluate the performance of each method based on the following metrics:

1. **Average error**, which represents the average performance on all tasks learned so far. The average error E_i^A on the i^{th} task ($j \leq i$) is defined as $E_i^A = \frac{1}{i}\sum_{j=1}^{i} e_{i,j}$.
2. **Forward interference (FWI) error**, which shows how preserving the knowledge of the previous tasks impairs the model's learning ability for a new task. The final FWI error is defined as $E_n^{FWI} = \frac{1}{n}\sum_{j=1}^{n} e_{j,j}$.
3. **Backward transfer (BWT) error** [25], which directly reflects how much the model has forgotten the previously learned tasks at the end of training. The final BWT error is defined as $E_n^{BWT} = \frac{1}{n}\sum_{j=1}^{n} e_{n,j} - e_{j,j}$.

Table 1: Average Error E_n^A (%) for Permuted MNIST, Split MNIST, and Split CIFAR-100.

Method	Permuted MNIST	Split MNIST	Split CIFAR-100
SGD	17.41	9.68	33.13
EWC	7.29	2.7	30.23
SI	6.38	2.29	29.91
RWALK	6.7	5.67	29.08
GPM	5.84	4.97	32.93
DCO	**4.68**	**1.46**	**26.61**
DCO-COMP	**4.84**	**1.34**	**28.22**

In Table 1 we demonstrate that DCO performs favorably compared to the baselines in terms of the final average error. In Fig. 5 we show how the average error behaves when adding new tasks. The figure reveals that DCO consistently outperforms other methods. In most cases DCO-COMP performs similarly to DCO and across all experiments, just as DCO, it is superior to other techniques.

In Fig. 6 we report both FWI error and BWT error for each method. In most cases DCO(-COMP) obtains the lowest FWI and BWT error among the regularization-based methods. Since the average error is the sum of FWI error and BWT error, we can conclude that DCO(-COMP) shows strong forward-learning ability while being the most efficient in alleviating the effect of catastrophic forgetting among all considered techniques.

In Fig. 7 we report the memory-performance trade-off of DCO-COMP on permuted MNIST. As the number of prohibited directions k increases, the FWI error slightly increases but the BWT error drops dramatically. Consequently, the DCO-COMP with largest k shows the lowest final average error.

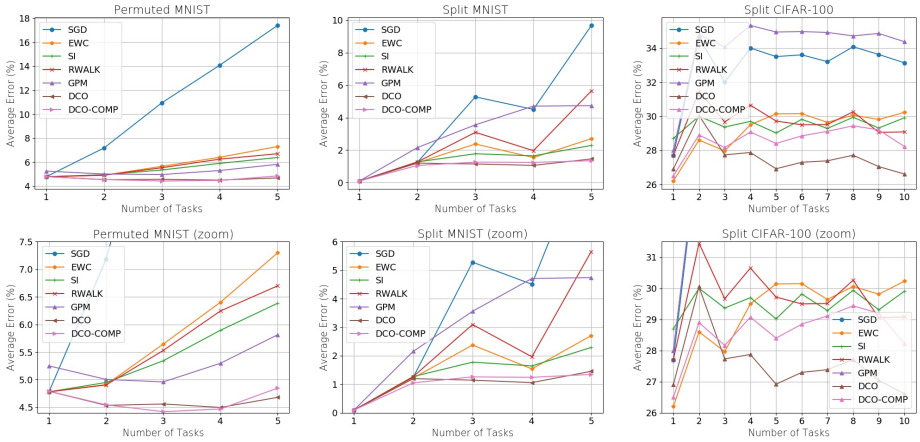


Fig. 5: Average error versus the number of tasks (original plots are on the top and zoomed are on the bottom; **left:** Permuted MNIST, **middle:** Split MNIST, **right:** Split CIFAR-100).

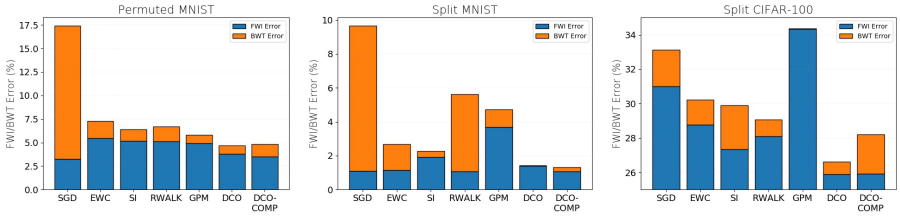
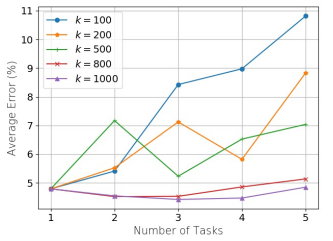
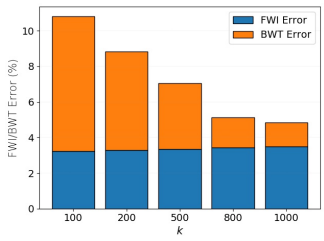


Fig. 6: FWI error and BWT error (**left:** Permuted MNIST, **middle:** Split MNIST, **right:** Split CIFAR-100).





k	Average Error (%)
100	10.81
200	8.83
500	7.04
800	5.14
1000	4.85

Fig. 7: DCO-COMP on permuted MNIST (**left**: Average error versus the number of tasks; **middle**: FWI and BWT errors; **right**: Final average error versus number of prohibited directions k).

Finally, in Fig. 8 we report the results of cross-domain experiment on MNIST/Fashion MNIST. DCO(-COMP) performs favorably to GPM and outperforms all other continual learning methods.

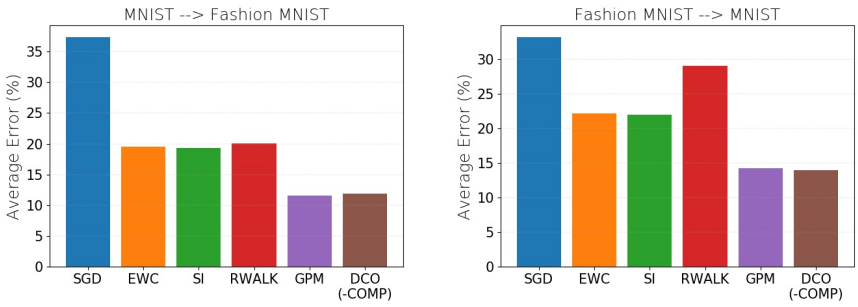


Fig. 8: Average error (**left:** MNIST → Fashion MNIST; **right:** Fashion MNIST → MNIST).

6 Conclusion

This paper elucidates the interplay between the local geometry of a deep learning optimization landscape and the quality of a network's performance in a continual learning setting. We derive a new continual learning algorithm counter-acting the process of catastrophic forgetting that explores the plausible manifold of parameters on which all tasks achieve good performance based on the knowledge of its geometric properties. Experiments demonstrate that this online algorithm achieves improvement in performance compared to more common approaches, which makes it a plausible method for solving a continual learning problem. Due to explicitly characterizing the manifold shared between the tasks, our work potentially provides a tool for better understanding how quickly the learning capacity of the network with a fixed architecture is consumed by adding new tasks and identifying the moment when the network lacks capacity to accommodate new coming task and thus has to be expanded. This direction will be explored in the future work.

References

1. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: ECCV (2018)
2. Aljundi, R., Chakravarty, P., Tuytelaars, T.: Expert gate: Lifelong learning with a network of experts. In: CVPR (2017)
3. Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. In: NeurIPS (2019)
4. Bao, X., Lucas, J., Sachdeva, S., Grosse, R.B.: Regularized linear autoencoders recover the principal components, eventually. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 6971–6981 (2020)
5. Bottou, L.: Online algorithms and stochastic approximations. In: Online Learning and Neural Networks. Cambridge University Press (1998)

6. Chaudhry, A., Dokania, P.K., Ajanthan, T., Torr, P.H.S.: Riemannian walk for incremental learning: Understanding forgetting and intransigence. In: ECCV (2018)
7. Chaudhry, A., Ranzato, M.A., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with a-gem. In: ICLR (2019), https://openreview.net/forum?id=Hkf2_sC5FX
8. Chaudhry, A., Khan, N., Dokania, P.K., Torr, P.H.: Continual learning in low-rank orthogonal subspaces. arXiv preprint arXiv:2010.11635 (2020)
9. Farajtabar, M., Azizan, N., Mott, A., Li, A.: Orthogonal gradient descent for continual learning. CoRR **abs/1910.07104** (2019)
10. Farquhar, S., Gal, Y.: Towards robust evaluations of continual learning. CoRR **abs/1805.09733** (2018)
11. Feng, Y., Tu, Y.: How neural networks find generalizable solutions: Self-tuned annealing in deep learning. CoRR **abs/2001.01678** (2020)
12. Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. In: ICLR (2014)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
14. He, X., Sygnowski, J., Galashov, A., Rusu, A.A., Teh, Y.W., Pascanu, R.: Task agnostic continual learning via meta learning. CoRR **abs/1906.05201** (2019)
15. Hou, S., Pan, X., Loy, C.C., Wang, Z., Lin, D.: Learning a unified classifier incrementally via rebalancing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
16. Hung, C.Y., Tu, C.H., Wu, C.E., Chen, C.H., Chan, Y.M., Chen, C.S.: Compacting, picking and growing for unforgetting continual learning. In: NeurIPS (2019)
17. Isele, D., Cosgun, A.: Selective experience replay for lifelong learning. In: AAAI (2018)
18. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. PNAS **114**(13), 3521–3526 (2017)
19. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/kriz/cifar.html (2009)
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
21. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
22. Li, X., Zhou, Y., Wu, T., Socher, R., Xiong, C.: Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In: ICML (2019), http://proceedings.mlr.press/v97/li19m.html
23. Li, Z., Hoiem, D.: Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence **40**(12), 2935–2947 (2017)
24. Liu, H., Liu, H.: Continual learning with recursive gradient optimization. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=7YDLgf9_zgm
25. Lopez-Paz, D., Ranzato, M.A.: Gradient episodic memory for continual learning. In: NeurIPS (2017)
26. Mallya, A., Davis, D., Lazebnik, S.: Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In: ECCV (2018)
27. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: CVPR (2018)

28. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, vol. 24, pp. 109–165. Elsevier (1989)
29. Parisi, G., Kemker, R., Part, J., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. Neural Networks (2018). https://doi.org/10.1016/j.neunet.2019.01.012
30. Rannen, A., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. In: ICCV (2017)
31. Rao, D., Visin, F., Rusu, A., Pascanu, R., Teh, Y.W., Hadsell, R.: Continual unsupervised representation learning. In: NeurIPS (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: ICLR (2019), https://openreview.net/forum?id=B1gTShAct7
33. Ritter, H., Botev, A., Barber, D.: Online structured laplace approximations for overcoming catastrophic forgetting. arXiv preprint arXiv:1805.07810 (2018)
34. Rostami, M., Kolouri, S., Pilly, P.K.: Complementary learning for overcoming catastrophic forgetting using experience replay. In: IJCAI (2019). https://doi.org/10.24963/ijcai.2019/463, https://doi.org/10.24963/ijcai.2019/463
35. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. CoRR **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. Schwarz, J., Luketina, J., Czarnecki, W.M., Grabska-Barwinska, A., Teh, Y.W., Pascanu, R., Hadsell, R.: Progress & compress: A scalable framework for continual learning. In: ICML (2018), http://proceedings.mlr.press/v80/schwarz18a.html
38. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: NeurIPS (2017)
39. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
40. Tao, X., Hong, X., Chang, X., Dong, S., Wei, X., Gong, Y.: Few-shot class-incremental learning. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
41. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: ICLR (2018), https://openreview.net/forum?id=Sk7KsfW0-
42. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: ICML (2017)
43. Zeno, C., Golan, I., Hoffer, E., Soudry, D.: Task agnostic continual learning using online variational bayes. CoRR **abs/1803.10123** (2018)