

Adaptive Orthogonal Projection for Batch and Online Continual Learning

Yiduo Guo¹, Wenpeng Hu², Dongyan Zhao^{1,*}, Bing Liu^{3,*}

¹ Wangxuan Institute of Computer Technology, Peking University

² School of Mathematical Sciences, Peking University

³ Department of Computer Science, University of Illinois at Chicago

yiduo@stu.pku.edu.cn, {wenpeng.hu, zhaody}@pku.edu.cn, liub@uic.edu

Abstract

Catastrophic forgetting is a key obstacle to continual learning. One of the state-of-the-art approaches is *orthogonal projection*. The idea of this approach is to learn each task by updating the network parameters or weights only in the direction orthogonal to the subspace spanned by all previous task inputs. This ensures no interference with tasks that have been learned. The system OWM that uses the idea performs very well against other state-of-the-art systems. In this paper, we first discuss an issue that we discovered in the mathematical derivation of this approach and then propose a novel method, called AOP (*Adaptive Orthogonal Projection*), to resolve it, which results in significant accuracy gains in empirical evaluations in both the batch and online continual learning settings without saving any previous training data as in replay-based methods.

Introduction

Many techniques have been proposed to solve the problem of *catastrophic forgetting* (CF) in *continual learning* (CL), which aims to incrementally learn a sequence of tasks (Chen and Liu 2018). Each task i consists of k_i (≥ 1) classes to be learned. Once a task is learned, its training data is often no longer accessible. CF means that in learning a new task, the parameters learned for previous tasks need to be modified, which may cause significant accuracy drop for the previous tasks (McCloskey and Cohen 1989). A literature survey will be given in the next section.

This paper focuses on a particular setting of CL, *class incremental learning* (Class-IL). In Class-IL, the system incrementally learns more and more classes from a sequence of tasks. At the test time, the learned model can classify a test case to any class without the task-id provided. Another main paradigm of CL is *task incremental learning* (Task-IL), where a model is constructed for each task in training. In testing, the task id is supplied for each test case so that the model for the task can be applied to classify the test case.

This paper further focuses on the recent orthogonal projection approach to Class-IL in the OWM system (Zeng et al. 2019). OWM works as follows: In learning each task, the network parameters are updated only in the direction orthogonal to the subspace spanned by inputs of all previous tasks.

This ensures no interference with the parameters learned for previous tasks and thus will cause no CF for previous tasks.

OWM performs very well compared to other state-of-the-art approaches. It does especially well in the scenario where each task consists of a single class. In most CL papers, each task consists of multiple classes to be learned. However, learning one class (which is equivalent to one class per task) incrementally perhaps occurs most frequently in applications because in an application whenever a new object/class appears, one would want to learn it immediately to make the system up to date rather than waiting for many objects (or classes) to appear and then learn them together. For example, in the chatbot context, when a new skill is identified and data prepared, the company naturally would want the chatbot to learn the new skill immediately so that the new service can be provided to the users without waiting and accumulating a few of skills and learn them together. For us humans, whenever we encounter a new object, we learn to recognize it right away and we never wait to see a number of new objects to appear and then learn to recognize them together. Learning tasks with one class per task is also the most difficult CL scenario as it has the maximum number of tasks. It is well known that when the number of tasks increases, CF becomes more severe, which results in lower classification accuracy. We should also note that incrementally learning one class at a time (or one class per task) is the most general case of CL because incrementally learning n classes as a task together can be reduced to learning n classes one by one incrementally. Needless to say that both OWM and our method can learn with any number of classes per task.

This paper identifies an issue in the mathematical derivation of the OWM method, which deals with CF by storing an orthogonal projector computed based on the training inputs of old tasks to ensure that the weight updates in learning each new task occur only in the direction orthogonal to the subspace representing the inputs of all old tasks. In order to deal with the matrix invertibility problem, it introduces a small constant α in computing the projector. However, α is irrelevant to the inputs of the old tasks, which gives an inaccurate estimation of the old task input space and causes weak performance. OWM treats α as a hyperparameter, which we will show it is inappropriate. We then propose an *Adaptive Orthogonal Projection* (AOP) method to resolve this problem of OWM, which computes an α value for each training

*Corresponding authors.

batch based on some holistic consideration of all old task data and the current batch. With this issue fixed, the CL accuracy results improve significantly. Experimental evaluation shows that AOP not only outperforms the original OWM but also other existing state-of-the-art baselines markedly in both batch and online CL settings. To the best of our knowledge, it is the first gradient orthogonal method for online CL.

Note that both OWM and AOP do not save any training examples or build data generators. AOP is thus more general and applicable to settings where the training data is no longer accessible after learning. The inaccessibility of the old data could be due to unrecorded legacy data, proprietary data, and data privacy, e.g., in federated learning (Zhang et al. 2020).

Related Work

As OWM has been discussed, this section focuses on other CL methods for dealing with CF. One popular approach is using regularization to ensure the learned knowledge from old tasks is minimally affected in learning a new task. EWC is a representative of the approach (Kirkpatrick et al. 2017). Many other papers use related approaches (Zenke, Poole, and Ganguli 2017; Fernando et al. 2017; Aljundi et al. 2017; Ritter, Botev, and Barber 2018; Xu and Zhu 2018; Kemker and Kanan 2018; Parisi et al. 2018; Ahn et al. 2019; Hu et al. 2021; Dhar et al. 2019; Adel, Zhao, and Turner 2020). *Knowledge distillation*, which is a kind of regularization, has also been commonly used (Li and Hoiem 2017; Wu et al. 2019; Castro et al. 2018; Belouadah and Popescu 2019; Liu et al. 2020; Lee et al. 2019; Tao et al. 2020).

Another popular approach is the *replay* approach, which memorizes a small number of training examples of every task and uses them in training the new task to keep the parameters of the previous tasks minimally changed. Many systems employ this approach, e.g., iCaRL (Rebuffi et al. 2017), GEM (Lopez-Paz and Ranzato 2017), A-GEM (Chaudhry et al. 2019a), RPSnet (Rajasegaran et al. 2019) and others (Rusu et al. 2016; Wu et al. 2019; Rolnick et al. 2019; de Masson d’Autume et al. 2019; Hou et al. 2019). Instead of saving training examples, *pseudo replay* approaches learn a data generator to generate pseudo samples of previous tasks to be used in training the new task to ensure the previously learned knowledge is up to date. Related work includes (Shin et al. 2017; Wu et al. 2018; Kamra, Gupta, and Liu 2017; Seff et al. 2017; Hu et al. 2019; Lesort et al. 2018; Ostapenko et al. 2019; Hayes et al. 2019; von Oswald et al. 2020).

Apart from the above popular methods, there are also others. For example, Progressive Networks deals with CF by building independent models and making connections among them (Rusu et al. 2016). HAT (Serrà et al. 2018) and CAT (Ke, Liu, and Huang 2020) learn hard attentions for each task to block each previous task’s parameters from being updated. BNS uses reinforcement learning (Qin et al. 2021). OGD saves previous tasks’ gradients (Farajtabar et al. 2020).

Several works have performed knowledge transfer across tasks (Ke, Liu, and Huang 2020; Ke et al. 2021; Schwarz et al. 2018; Fernando et al. 2017; Rusu et al. 2016). Early techniques under *lifelong learning* mainly perform knowledge transfer but do not tackle CF (Chen and Liu 2014; Ruvolo and Eaton 2013; Benavides-Prado, Koh, and Riddle 2020).

The proposed AOP differs from all the above approaches as it is based on orthogonal projection and stores no previous data. Chaudhry et al. (2020) proposed an orthogonal method by learning different tasks in different orthogonal subspaces. Saha, Garg, and Roy (2021) proposed an orthogonal method by taking new task’s gradient steps in the orthogonal direction to the gradient subspaces deemed important for the past tasks. But these are Task-IL methods. AOP is for Class-IL.

In computer vision, several researchers used the term *incremental learning* to mean Class-IL as they learn like any other Class-IL method with multiple classes per task (Wu et al. 2019; Castro et al. 2018; Liu et al. 2020; Lee et al. 2019; Belouadah and Popescu 2019, 2020). Some traditional methods also exist for learning one class at a time. They typically save some randomly selected exemplars or the mean of each class. In testing, a distance function over the *nearest exemplar/class mean* is used for classification (Rebuffi et al. 2017; Lee et al. 2018; Javed and Shafait 2018; Bendale and Boulton 2015). AOP does not use any of these approaches.

Recently, online continual learning (online CL) has been studied by many researchers. However, online CL methods mainly use the replay strategy. For example, the **MIR** method (Aljundi et al. 2019a) is a replay-based method for online CL. Its main idea is to enable the model to focus on the replay buffer samples that have larger losses. **GSS** (Aljundi et al. 2019b) is another replay method. It uses the gradient information to diversify the data stored in the replay buffer. **ASER** (Shim et al. 2020) is also a replay-based method. It has a new replay buffer update/retrieve strategy that is inspired by the Shapley value theory. Again, AOP is different as it is based on orthogonal projection and it does not save any previous data.

Problem with OWM

OWM deals with CF in learning each new task by using an *orthogonal projector* to make the parameter updates occur only in the direction orthogonal to the space representing the inputs of all previous tasks. OWM treats the input training data \mathbf{A} of all previous tasks as the previous input space. It computes the orthogonal projector $\mathbf{P} = \mathbf{I} - \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$, where \mathbf{I} is the identity matrix and $\mathbf{P} \in \mathbb{R}^{d \times d}$ with d being the dimension of the input sample or example. \mathbf{P} is used in learning the new task to project the parameter updates in the direction orthogonal to the space of all training inputs of the previous tasks, i.e., $\Delta \mathbf{W} = \kappa \mathbf{P} \Delta \mathbf{W}^{BP}$, where κ is the learning rate and $\Delta \mathbf{W}^{BP}$ is the parameter adjustment calculated by backpropagation. This ensures that the new task learning will not interfere with the previous tasks and thus will not cause CF. Since the CL setting does not allow memorizing all the previous data \mathbf{A} , in OWM \mathbf{P} is incrementally computed (see below).

Inaccurate Estimation of Input Space in OWM

An accurate orthogonal projector is the key for OWM to overcome CF. However, to avoid the matrix invertibility problem, OWM adds a small constant α to the original equation $\mathbf{P} = \mathbf{I} - \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$. So the projector becomes $\mathbf{P}' = \mathbf{I} - \mathbf{A}(\alpha \mathbf{I} + \mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$. OWM treats α as a hy-

perparameter. We argue that this is problematic for accurate construction of the projector \mathbf{P} .

We note that $\mathbf{A}^T \mathbf{A}$ is a positive semi-definite matrix, which means that there exists an orthogonal matrix $\mathbf{U} \in R^{d \times d}$ such that:

$$\mathbf{U} \mathbf{A}^T \mathbf{A} \mathbf{U}^T = \begin{pmatrix} \delta_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \delta_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \delta_3 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ & & & \dots & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix} \quad (1)$$

Due to the lack of eigenvalues, the last few rows of the above matrix may be all zeros, which causes the invertibility problem. If a small constant α is added to solve this problem, Eqn. 1 becomes,

$$\mathbf{U}(\mathbf{A}^T \mathbf{A} + \alpha \mathbf{I}) \mathbf{U}^T = \begin{pmatrix} \delta_1 + \alpha & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \delta_2 + \alpha & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \delta_3 + \alpha & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ & & & \dots & & & \\ 0 & 0 & 0 & \dots & \alpha & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & \alpha & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & \alpha \end{pmatrix} \quad (2)$$

This means that we get an approximation of the input space \mathbf{X} of the past tasks with an extra space \mathbf{S} that is constructed by the constant α with an orthogonal basis. What we need is a value that is relevant to or contains the information about the previous tasks' input space, which can give better performances. Adding a fixed value that is irrelevant to the input space or represents something only local (e.g., a special eigenvalue) cannot solve this problem but leads to a worse performance. And OWM uses the RLS (Recursive least squares) algorithm to incrementally compute or update the projector \mathbf{P}' during training as it cannot memorize the input data \mathbf{A} of all tasks in the CL setting.

Let \mathbf{W}_l be the l th layer's weights/parameters of the model, where $l \in \{0, 1, 2, \dots, L\}$ and L is the total number of layers of the model. For each batch i ($= 1, 2, 3, \dots, n_j$) in the j th task, OWM updates $\mathbf{P}_l \in R^{d \times d}$ for weight \mathbf{W}_l , which is noted as $\mathbf{P}_l(i, j)$ and is calculated iteratively,

$$\begin{aligned} \mathbf{P}_l(i, j) &= \mathbf{P}_l(i-1, j) - \mathbf{Q}_l(i, j) \mathbf{x}_{l-1}(i, j)^T \mathbf{P}_l(i-1, j) \\ \mathbf{Q}_l(i, j) &= \mathbf{P}_l(i-1, j) \mathbf{x}_{l-1}(i, j) / [\alpha + \mathbf{x}_{l-1}(i, j)^T \mathbf{P}_l(i-1, j) \mathbf{x}_{l-1}(i, j)] \\ \mathbf{P}_l(0, 0) &= \mathbf{I} \\ \mathbf{P}_l(0, j) &= \mathbf{P}_l(n_{j-1}, j-1) \end{aligned} \quad (3)$$

where $\mathbf{x}_{l-1}(i, j)$ is the output of the $(l-1)$ th layer in response to the mean of inputs in the i th batch of the j th task, n_{j-1} is the number of batches in the $(j-1)$ th task.

As OWM uses an iterative method to calculate the orthogonal projector, it differs from the traditional matrix computation, which adds α only once to avoid the inversion problem. In OWM, the extra space \mathbf{S} induced by α is added to the history of input spaces in each iteration, which leads to a large deviation from the correct direction.

In summary, OWM uses a constant α unrelated to the input space to solve the matrix-invertibility problem, but that results in a rather inaccurate estimation of the previous task input space and consequently a poorer projector \mathbf{P}' , which causes poor performances.

Proposed Solution

According to Woodbury matrix identity, we have:

$$\begin{aligned} \mathbf{P}' &= \mathbf{I} - \mathbf{A}(\alpha \mathbf{I} + \mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \\ &= \mathbf{I} - \mathbf{A}(\mathbf{I} + \alpha^{-1} \mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \alpha^{-1} \\ &= \alpha \left(\sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T + \alpha \mathbf{I} \right)^{-1} \end{aligned} \quad (4)$$

where $\mathbf{x}_i \in R^{d \times 1}$ is the i th input vector of $\mathbf{A} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \in R^{n \times d}$ where d is the dimension of the input vector, n is the number of all previous input vectors. \mathbf{P}' is equivalent to the inversion of the approximate correlation matrix $\Phi'(n) = \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T + \alpha \mathbf{I}$. The original correlation matrix of the inputs $\mathbf{A} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \in R^{n \times d}$ is $\Phi(n) = \mathbf{A} \mathbf{A}^T = \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T$.

We propose a more holistic approximation of $\Phi(n)$ from the statistics perspective to reduce the poor influence of α , which is also inspired by the technique to approximate the correlation matrix $\Phi(n)$ used in tracking the performance of RLS algorithm in non-stationary environments (Haykin 2008). According to Eleftheriou's proof in 1986 (Eleftheriou and Falconer 1986), we can give an approximation equation $\Phi(n) = \sum_{i=1}^n \lambda E(\mathbf{x}_i \mathbf{x}_i^T) + \tilde{\Phi}(n)$, where $\tilde{\Phi}(n)$ is a Hermitian perturbation matrix whose individual entries are represented by zero-mean random variables that are statistically independent of the input vector \mathbf{x}_i and λ is a weighting factor. When n is large and λ is close to unity, we may view $\Phi(n)$ as a quasi-deterministic matrix in the sense that for large n we have:

$$E[\|\tilde{\Phi}(n)\|^2] \ll E[\|\Phi(n)\|^2] \quad (5)$$

where $\|\cdot\|$ denotes matrix norm. Under this condition and note that we assume λ is 1, we may go one step further by ignoring the perturbation matrix $\tilde{\Phi}(n)$, and thereby approximate the correlation matrix $\Phi(n)$ as

$$\Phi(n) \approx \sum_{i=1}^n E(\mathbf{x}_i \mathbf{x}_i^T) \quad \text{for large } n \text{ and } \lambda = 1 \quad (6)$$

We can calculate $E(\mathbf{x}_i \mathbf{x}_i^T)$ based on the expectation rule,

$$E(\mathbf{x}_i \mathbf{x}_i^T) = E(\mathbf{x}_i)(E(\mathbf{x}_i))^T + Cov(\mathbf{x}_i) \quad (7)$$

where $Cov(\mathbf{x}_i)$ is the variance of \mathbf{x}_i . Let us assume that variable $\mathbf{X}^j \in R^{d \times 1}$ represents the training data of task j and d is the input dimension. OWM uses the mean of all inputs in one batch as the estimation of the expectation of the variable \mathbf{X}^j , and then uses this estimation as the input vector to update the projector. In the end, it gets the projector $\mathbf{P}' = (\sum_{j=1}^N \sum_{i=1}^{n_j} \mathbf{x}_i^j (\mathbf{x}_i^j)^T + \alpha \mathbf{I})^{-1}$, where N is the number of tasks learned so far, n_j is the number of batches in task j , \mathbf{x}_i^j is the mean of all inputs in the i th batch of task j .

We propose to update the value of α for each batch in training for a task j according to some holistic information of the task. The goal is to replace the coarse estimation of the input space of the j th task, which is $\sum_{i=1}^{n_j} \mathbf{x}_i^j (\mathbf{x}_i^j)^T + \frac{\alpha}{N} \mathbf{I}$ in OWM, to a more accurate matrix. We can replace the input signal correlation matrix $\mathbf{x}_i^j (\mathbf{x}_i^j)^T$ by its expectation, and use the expectation rule to calculate its value. But that would also have the matrix-invertibility problem. We propose to replace the value of scalar α by the average of the whole expectation estimation of the current batch, i.e.,

$$\begin{aligned} \alpha_i^j &= \text{average}(E(\mathbf{x}_i^j (\mathbf{x}_i^j)^T)) \\ &= \frac{\|E(\mathbf{x}_i^j (\mathbf{x}_i^j)^T)\|_{1,1}}{d^2} \end{aligned} \quad (8)$$

where the operation $\text{average}(Y)$ is to get the average value of all entries in matrix Y and d is the dimension of the matrix $E(\mathbf{x}_i^j (\mathbf{x}_i^j)^T)$.

The final proposed method for computing the orthogonal projector is by calculating \mathbf{P} and α successively, which gives us $\mathbf{P}^* = (\sum_{j=1}^N \sum_{i=1}^{n_j} (\mathbf{x}_i^j (\mathbf{x}_i^j)^T + \frac{\alpha_i^j}{n} \mathbf{I}))^{-1}$ from Eqn. 10 below. When the number of trained batches is small, $\mathbf{x}_i^j (\mathbf{x}_i^j)^T + \frac{\alpha_i^j}{n} \mathbf{I} \approx \mathbf{x}_i^j (\mathbf{x}_i^j)^T + \alpha_i^j \mathbf{I}$ is a more accurate estimation of the input space than the method in the original OWM because we estimate the representation of input space from the correlation matrix $\mathbf{x}_i^j (\mathbf{x}_i^j)^T$ and the statistical properties of the input vectors. For the proposed correlation matrix $\Phi^*(n) = \sum_{j=1}^N \sum_{i=1}^{n_j} (\mathbf{x}_i^j (\mathbf{x}_i^j)^T + \frac{\alpha_i^j}{n} \mathbf{I})$ and the original correlation matrix $\Phi(n) = \sum_{j=1}^N \sum_{i=1}^{n_j} \mathbf{x}_i^j (\mathbf{x}_i^j)^T$, we have the following proposition:

Proposition 1: For any matrix norm $\|\cdot\|$, we have:

$$\|\Phi^*(n) - \Phi(n)\| \leq \|T(\Phi(n))\| \quad (9)$$

where $T(\Phi(n))$ is a diagonal matrix whose single diagonal element value is the mean of the sum of singular values for all input $\mathbf{x}_i^j (\mathbf{x}_i^j)^T$. The proof is given in Appendix 1.

The new method can be integrated with the RLS algorithm and the rule of expectation to solve the matrix-invertibility problem and to achieve better performances empirically. That is, we change the update rule of $\mathbf{P}_l \in R^{d \times d}$ for weights \mathbf{W}_l , which are the weights of layer l . When we get the inputs of the i th batch in task j , we update $\mathbf{P}_l(i, j)$ as follows (please note the notation change due to the use of layer number l):

$$\begin{aligned} \mathbf{P}_l(i, j) &= \mathbf{P}_l(i-1, j) - \mathbf{Q}_l(i, j) \mathbf{x}_{l-1}(i, j)^T \mathbf{P}_l(i-1, j) \\ \mathbf{Q}_l(i, j) &= \mathbf{P}_l(i-1, j) \mathbf{x}_{l-1}(i, j) / [\alpha_i^j + \mathbf{x}_{l-1}(i, j)^T \mathbf{P}_l(i-1, j) \mathbf{x}_{l-1}(i, j)] \\ \alpha_i^j &= \text{average}(E(\mathbf{x}_{l-1}(i, j) (\mathbf{x}_{l-1}(i, j))^T)) \\ &= \text{average}(E(\mathbf{x}_{l-1}(i, j)) E(\mathbf{x}_{l-1}(i, j))^T + \\ &\quad \text{Cov}(\mathbf{x}_{l-1})(i, j)) \\ &\approx \text{average}(\mathbf{x}_{l-1}(i, j) (\mathbf{x}_{l-1}(i, j))^T + \text{Cov}(\mathbf{x}_{l-1}(i, j))) \\ \mathbf{P}_l(0, 0) &= \mathbf{I} \\ \mathbf{P}_l(0, j) &= \mathbf{P}_l(n_{j-1}, j-1) \end{aligned} \quad (10)$$

where $\mathbf{x}_{l-1}(i, j)$ is the output of the $(l-1)$ th layer in response to the average of inputs in the i th batch of task j , $\text{average}(\cdot)$

gets the average value of all entries in a matrix, and n_{j-1} is the number of batches in task $j-1$.

After the inputs of the i th batch in task j go through the network, we calculate the projector and then update the weight matrix of each layer by

$$\begin{aligned} \mathbf{W}_l(i, j) &= \mathbf{W}_l(i-1, j) - \kappa(i, j) \Delta \mathbf{W}_l^{BP}(i, j) \quad \text{if } j=1 \\ \mathbf{W}_l(i, j) &= \mathbf{W}_l(i-1, j) - \kappa(i, j) \mathbf{P}_l(n_{j-1}, j-1) \Delta \mathbf{W}_l^{BP}(i, j) \\ &\quad \text{if } j=2, 3, \dots \\ \mathbf{P}_l(0, 0) &= \mathbf{I}_l \end{aligned} \quad (11)$$

where $\mathbf{W}_l(i-1, j)$ is the l th layer's weights, $\Delta \mathbf{W}_l^{BP}(i, j)$ is the weight modifications for weight $\mathbf{W}_l(i-1, j)$ calculated by the standard backpropagation (BP) method, $\kappa(i, j)$ is the learning rate, and \mathbf{I}_l is an identity matrix whose dimension is the same as the dimension of the weight's row.

Now let us examine the effect of multiplying the projection matrix on the backward pass of the BP algorithm to see why the new method works. After training task j , according to the gradient descent algorithm, the l th fully-connected (FC) layer's weight matrix \mathbf{W}_l is changed to $\mathbf{W}_l(n_j, j) = \mathbf{W}_l(n_{j-1}, j-1) - \kappa \mathbf{P}_l(n_{j-1}, j-1) \Delta \mathbf{W}_l^{BP}(j)$, where κ is the learning rate and $\Delta \mathbf{W}_l^{BP}(j)$ represents the gradient computed by BP during training. For a test sample \mathbf{x}' , which is from the j' th ($j' \leq j$) task's test set or distribution, the output of the $(l-1)$ th FC layer is \mathbf{x}'_{l-1} . According to the gradient update rule in AOP, the output of the l th FC layer \mathbf{x}'_l can be decomposed to:

$$\begin{aligned} \mathbf{x}'_l &= (\mathbf{W}_l(n_j, j) - \kappa \mathbf{P}_l(n_{j-1}, j-1) \Delta \mathbf{W}_l^{BP}(j)) \mathbf{x}'_{l-1} \\ &= (\mathbf{W}_l(n_{j'}, j') - \sum_{i=j'+1}^j \kappa \mathbf{P}_l(n_{i-1}, i-1) \Delta \mathbf{W}_l^{BP}(i)) \mathbf{x}'_{l-1} \\ &\approx \mathbf{W}_l(n_{j'}, j') \mathbf{x}'_{l-1} \end{aligned} \quad (12)$$

where n_{i-1} is the number of batches in task $i-1$, $\Delta \mathbf{W}_l^{BP}(i)$ is the gradient computed by BP during the training of the i th task. We obtain the approximation equation in Eqn. 12 because any projector calculated after the training of j' th task is approximately orthogonal to the input from the j' th task's distribution. Here we assume that $\mathbf{W}_l(n_{j'}, j')$ is the optimum weight for classifying j' th task that we obtain when we train the j' task. So what we actually approximate is the optimal model performance for classifying \mathbf{x}'_l which is achieved when the training process of the j' th task is finished, and a new task has not arrived yet. In this way, after training a task, the model can maintain the task's performance because the inner product of the new task's gradient update with samples from previous tasks' distribution is approximately zero.

In summary, according to Eqn. 10, each training batch has a different α value (thus *adaptive*) computed with respect to the current batch's mean and variance. That not only solves the invertible matrix problem but also provides a more accurate estimation of the whole space, which in turn computes a more accurate projector. According to Eqns. 11 and 12, we can use the orthogonal projector in optimization to avoid CF. OWM uses a fixed α value for the whole training process. Experimental results show that the proposed AOP achieves markedly better results than OWM and other baselines.

Table 1: Datasets details

Dataset	Classes	Training		Test
		Total	Per class	
MNIST	10	60,000	5,421-6,742	10,000
EMNIST-47	47	112,800	2,400	18,800
CIFAR10	10	50,000	5,000	10,000
CIFAR100	100	50,000	500	10,000

Empirical Evaluation

This section first evaluates the proposed AOP and compares it with OWM and several other baselines in the **batch CL setting**. In this setting, when each task arrives, its full data is available and training can be done in multiple epochs. It then evaluates AOP in the **online CL setting**, which assumes that the data from each task comes in a data stream and we need to update the model once a small batch of data from the stream is available. Effectively, the data for each task is only trained for one epoch in online CL.¹

Experiment Datasets

Four image classification datasets are used in our experiments: **MNIST** (LeCun, Cortes, and Burges 1998), **EMNIST-47** (Cohen et al. 2017), **CIFAR10** and **CIFAR100** (Krizhevsky and Hinton 2009). Details of the datasets are given in Table 1. For the setting of multiple classes per task, we form tasks with k ($k > 1$) classes in each task. If the number of remaining classes is less than k , we use them to form a task. For the online CL setting, we divide MNIST dataset into 5 disjoint tasks. Each task has two unique classes. We do the same for the CIFAR10 dataset to create 5 different tasks. For the CIFAR100 dataset, we divide it into 10 tasks. Each task has 10 different classes.

Baseline Systems

For batch CL, we use the following classic and latest *class incremental learning* (Class-IL) baselines. (1) **EWC** (Kirkpatrick et al. 2017) is a classic baseline. (2) **LwF** (Li and Hoiem 2017) uses knowledge distillation to overcome forgetting. (3) **IMM** (Lee et al. 2017) combines the sequentially trained independent models for different tasks to perform all the tasks in the sequence. (4) **PGMA** (Hu et al. 2019) adapts the model to fit different data by parameter generation. This is also a pseudo-replay method. (5) **RPSnet** (Rajasegaran et al. 2019) chooses optimal paths for each new task. This is a strong replay method that saves some training examples from previous tasks in a replay buffer. We use the same buffer size as in its original paper (which is 4000 slots). (6) **OWM** (Zeng et al. 2019) is the original orthogonal projection system that we try to improve. (7) **HNET** (von Oswald et al. 2020) is a Class-IL method without memorizing old data but only the embeddings of old tasks.

For online CL, we consider the following baselines. (8) **ER** (Chaudhry et al. 2019b) is a replay method and uses random sampling to sample data for replaying. (9) **MIR** (Aljundi et al. 2019a) is also a replay method and samples buffer data

¹The code of AOP : <https://github.com/gdydpku/Official-pytorch-implementation-of-AOP-AAAI-2022->.

depending on the loss of pseudo update. (10) **GSS** (Aljundi et al. 2019b) is again a replay method and works by diversifying the replay buffer data depending on the gradient. (11) **ASER** (Shim et al. 2020) is a replay method as well and utilizes the Shapley value theory to update and sample buffer data. (12) **AGEM** (Chaudhry et al. 2019a) is a gradient-based replay method that constrains the parameter updates by calculating the average gradient of the replay buffer data. (13) **GDumb** (Prabhu, Torr, and Dokania 2020) is a replay method using a greedy sampler to update the memory.

Note that most Class-IL methods can work with one or more classes per task. For one class per task, we create the classification heads for all classes upfront, one per task. To learn a new class/task, the system simply makes sure its training data go to the corresponding head of the class and not to any other head. LwF cannot learn with one class per task as it incrementally adds a new head for a new task with its own cross-entropy loss, which does not work for one class. We changed it to one cross-entropy for all classes like other baselines assuming the system knows the total number of classes to learn. For all baselines, we use the open source code² released by their authors except EWC as the original code was not released. We use a popular third party code.³

Evaluation Setting: Following the existing CL evaluation, for each dataset, after all tasks are learned, we test using the test sets of all tasks and report the average accuracy over 5 random runs. The tasks and their sequence are created following the order of the classes in the original data.

Training Details

Architecture: Since AOP aims to improve OWM (Zeng et al. 2019), it uses the same architecture as OWM. For MNIST, EMNIST and all experiments that use pre-trained features, a two-layer MLP and a single output unit are employed. Since LwF grows the network with the increase of tasks, given a sequence of N tasks, assuming the size of the hidden layer for the non-growth methods is m , we set the hidden size of LwF for each task as m/N . After learning all tasks, the parameter size of all methods will be of the same magnitude. We fix m/N to 100. For CIFAR10 and CIFAR100 from scratch. The same model as OWM (Zeng et al. 2019) is used, i.e., a CNN model with 3 convolutional layers.

For online CL experiments, we use the same MLP architecture mentioned above for AOP and online baselines on MNIST. For CIFAR10 and CIFAR100, we use the same MLP architecture with pre-trained experiments. We also test online CL without a pre-trained feature extractor and provide the results in Appendix 2, which show that our method still outperforms baselines without the pre-trained model.

Training: To provide a fair comparison among CL methods,

² The code of LwF: <https://github.com/ngailapdi/LwF>.
The code of IMM: https://github.com/btjhjeon/IMM_tensorflow.
The code of PGMA: https://github.com/morning-dews/PGMA_tensorflow.
The code for RPSnet: <https://github.com/brjathu/RPSnet>.
The code for OWM: <https://github.com/beijixiong3510/OWM>.
The code for iCaRL: <https://github.com/srebuffi/iCaRL>.
The code for HNET: <https://github.com/chrhenning/hypercl>.
The code for ER/MIR: https://github.com/optima/Maximally_Interfered_Retrieval.
The code for ASER/AGEM: <https://github.com/RaptorMai/online-continual-learning>.
The code for GDumb: <https://github.com/drimpossible/GDumb>.

³ <https://github.com/moskomule/ewc.pytorch>

Table 2: Batch CL accuracy for one class per task of AOP and all baselines (average over 5 runs). Column “w/o PreT” denotes with/without using the shared pre-trained feature extractor for AOP and all baselines. HNET is not included as it does not work with one class per task.

Dataset	w/o PreT	EWC	LwF	IMM	PGMA	RPSnet	OWM	AOP
MNIST (10 tasks)	no	9.9	19.9	29.2	71.4	40.3	94.5 \pm 0.1	95.1 \pm 0.1
EMNIST-47 (47 tasks)	no	2.1	4.6	18.7	10.1	10.1	77.5 \pm 0.2	79.4 \pm 0.1
CIFAR10 (10 tasks)	no	10.0	10.0	10.2	20.1	16.3	44.3 \pm 0.1	58.5 \pm 0.2
CIFAR100 (100 tasks)	no	1.0	2.1	1.2	1.8	2.9	7.7 \pm 0.2	15.9 \pm 0.1
CIFAR10 (10 tasks)	yes	10.2	19.4	51.2	56.2	55.5	83.0 \pm 0.2	86.0 \pm 0.1
CIFAR100 (100 tasks)	yes	2.9	6.3	12.6	12.4	4.1	63.2 \pm 0.1	64.0 \pm 0.1

Table 3: Batch CL accuracy for 2 classes per task in batch CL using a pre-trained feature extractor (average over 5 runs).

Dataset	w/o PreT	EWC	LwF	IMM	PGMA	RPSnet	OWM	AOP
MNIST (5 tasks)	no	18.8	52.5	67.5	81.7	96.2	91.6 \pm 0.1	94.4 \pm 0.2
EMNIST-47 (24 tasks)	no	4.2	17.2	20.5	21.8	32.9	71.7 \pm 0.1	74.8 \pm 0.1
CIFAR10 (5 tasks)	no	10.2	12.2	32.4	40.5	32.9	52.8 \pm 0.1	58.3 \pm 0.2
CIFAR10 (5 tasks)	yes	31.8	57.6	77.3	74.3	85.4	83.4 \pm 0.2	85.9 \pm 0.3
CIFAR100 (50 tasks)	yes	3.7	23.3	26.5	17.5	25.3	57.7 \pm 0.1	61.6 \pm 0.1

Table 4: Online CL accuracy results (average over 5 runs) on MNIST, CIFAR10 and CIFAR100 datasets with different replay or memory buffer sizes B .

Method	MNIST			CIFAR10			CIFAR100		
(Number of tasks)	(5 tasks)			5 tasks			(10 tasks)		
w/o PreT	no			yes			yes		
B	$B=0.1k$	$B=0.5k$	$B=1k$	$B=0.2k$	$B=0.5k$	$B=1k$	$B=1k$	$B=2k$	$B=5k$
AGEM	56.9	57.7	61.6	28.1	24.1	24.6	10.7	10.8	10.6
GSS	70.4	80.7	87.5	47.5	63.9	74.2	40.5	47.3	51.6
ER	78.7	85.3	90.3	63.1	74.5	81.2	40.9	48.0	52.9
MIR	79.0	88.3	91.3	68.0	74.7	81.0	40.7	49.2	55.5
ASER	61.6	71.0	82.1	53.9	68.9	77.4	41.5	42.8	50.8
GDumb	81.2	91.0	94.5	77.8	81.2	83.5	47.6	52.9	57.8
OWM	78.8 \pm 0.1	78.8 \pm 0.1	78.8 \pm 0.1	82.1 \pm 0.2	82.1 \pm 0.2	82.1 \pm 0.2	48.5 \pm 0.1	48.5 \pm 0.1	48.5 \pm 0.1
AOP	95.0 \pm 0.1	95.0 \pm 0.1	95.0 \pm 0.1	85.8 \pm 0.1	85.8 \pm 0.1	85.8 \pm 0.1	58.0 \pm 0.2	58.0 \pm 0.2	58.0 \pm 0.2

Table 5: Accuracy for 5 classes per task in batch CL using pre-trained features for AOP and 3 top baselines using EMNIST-47 and CIFAR100 as they have a large number of classes

Method	EMNIST-47	CIFAR100
(Number of tasks)	(10 tasks)	(20 tasks)
w/o PreT	no	yes
PGMA	17.8	29.6
RPSnet	74.8	51.4
OWM	68.0 \pm 0.1	61.5 \pm 0.1
AOP	76.9 \pm 0.1	66.1 \pm 0.2

we train all the networks using the SGD optimizer (momentum=0.9) for both batch CL and online CL. For MNIST based experiments, we set 20 epochs per task. We use 100 epochs for EMNIST, CIFAR10 and CIFAR100. For online CL, we use one epoch to simulate the stream data setting.

Hyper-parameters: We set the batch-size as 64 for all methods in batch CL and set the batch-size as 10 for all methods in online CL. For replay-based online CL baselines, we set their buffer batch size (data sampled from the replay buffer) as 10 as well following existing work (Aljundi et al. 2019a).

The proposed AOP method does not need to save any data. For batch CL, we tune the learning rate by using 10% of randomly selected training examples of each dataset as the validation set. After that, we use the tuned learning rate to train the system over the whole training set. Grid search is used in tuning. The tuning range for the learning rate is from 0 to 1 with step 0.01. The learning rates for different data are as follows: 0.5 for MNIST, 0.2 for EMNIST-47, CIFAR10 (without pre-training) and CIFAR100 (without pre-training). For experiments that use pre-trained features in batch CL, we set the learning rate to 0.35 for the one class per task setting, and 0.04 for the multi-class per task setting. For all online CL experiments, the learning rates of AOP for different datasets are as follows: 0.03 for MNIST, 0.02 for CIFAR100 and CIFAR10. Other hyper-parameters are given in Appendix 3.

Results for One Class Per Task in Batch CL

We show the experiment results with or without the pre-trained feature extractor in learning with one per task here. The reason for showing the results with pre-trained features is that they produce significantly better results. Pre-trained feature extractors or language models have been instrumental for the advances in natural language processing (NLP) (De-

vlin et al. 2019). This paradigm is also being embraced in computer vision (Studer et al. 2019; Misra and Maaten 2020).

Without using a pre-trained feature extractor: The first block of results in Table 2 gives the accuracy results of the four datasets without pre-training.

We observe that learning a large number of classes one by one is very challenging for most methods, i.e., EWC, LwF, IMM, PGMA and RPSnet. Results in this setting are not reported in their papers, but about all class incremental learning methods can naturally learn with one class per task. OWM is the strongest baseline, but AOP outperforms it on all datasets. AOP gets a 14.2% improvement on CIFAR10, and 8.18% improvement on CIFAR100.

Using pre-trained feature extractors: We now apply pre-trained feature extractors to AOP and all baselines. We pre-train a WRN-28 model to extract features with size 640 (Zagoruyko and Komodakis 2016) using ImageNet after removing classes from ImageNet that are similar to those classes in CIFAR10 or CIFAR100. After removal, we are left with 750 ImageNet classes. Note that no pre-trained feature extractors for MNIST and EMNIST-47 are needed as a simple model already produces very good results.

The second block in Table 2 gives the accuracy results for learning with one class per task for CIFAR10 or CIFAR100. We observe an improvement in accuracy for both datasets across the board. The improvements for OWM and AOP are huge, up to 50% for AOP and even more for OWM. However, AOP still performs better than OWM. Note again that when pre-trained feature extractor is used, it is used in AOP and also in all baselines.

Results for Two/Five Classes Per Task in Batch CL

The results for two classes per task are given in Table 3 except HNET. HNET achieves 95.02 on MNIST which we could reproduce and is already not better than our AOP (95.12) but better than most baselines. However, its accuracy drops quickly with more tasks. It gets only 25.42 on EMNIST-47, much worse than AOP (80.97) with 24 tasks. Its CIFAR10 (51.02) and CIFAR100 (3.22, 50 tasks) results are also very poor. We did a lot of tuning on the authors’ code, but could not get better results. From Table 3, we can see that on the datasets with a smaller number of tasks, i.e., MNIST, CIFAR10 and EMNIST, RPSnet is the strongest baseline, but for datasets with a large number of tasks, RPSnet does poorly although it is a replay method. OWM is still the strongest baseline overall. AOP outperforms all baselines except RPSnet for MNIST.

Five classes per task: We use EMNIST-47 and CIFAR100 to test 5 classes per task as they have a large number of classes to form many tasks. We see from Table 5 that AOP outperforms all 3 top-performing baselines markedly.

Comparing with the Nearest-Mean Approach

To be more complete, we also compare with a well-known traditional method of *class incremental learning* based on nearest-mean, iCaRL (Rebuffi et al. 2017). iCaRL finds the nearest prototype over the mean of

the saved exemplars per class for classification. With pre-trained features, it achieves 82.56/69.23/40.35/43.29 and 92.70/74.87/68.28/45.75 in accuracy for one class per task learning and two classes per task learning on MNIST/EMNIST-47/CIFAR10/CIFAR100 respectively, but the results of AOP are 95.12/79.42/85.95/64.02 and 94.38/74.81/85.53/61.55 respectively. AOP does not save any information about old classes and outperforms iCaRL considerably.

Results for Online CL

Table 4 gives the accuracy results of online CL experiments. From Table 4, we can observe that our method AOP strongly outperforms the online CL baselines in all experiments. When the replay or memory buffer size is small, the baselines except OWM (which does not save old data) have very poor results. But our method AOP still maintain a high accuracy as it does not need to store any replay buffer data. It thus has a great advantage when privacy is a concern. The forgetting rate results are given in Appendix 4, which shows that AOP’s forgetting rate is only slightly higher than OWM, but OWM’s accuracy is substantially lower than AOP (see Table 4).

Training efficiency: About training time, which is important for online CL, from Table 6, we can observe that AOP together with OWM are the fastest methods among all methods. Memory-based methods first need to store some training examples from the previous and the current task. Then they train the samples from the memory buffer to overcome forgetting. So their training time becomes longer. AOP does not increase the training time from that of OWM. But its accuracy is markedly better than that of OWM. AOP is the first gradient orthogonal method that achieves the state-of-the-art performance in online CL.

Table 6: Online CL training time of AOP and baselines on CIFAR10. All systems use the pre-trained feature extractor.

Metric	AOP	OWM	AGEM	ER	MIR	ASER	GDumb	GSS
training time/s	24	24	35	40	60	95	100	840

Conclusion

In this work, we discovered a critical weakness in the derivation of the recent orthogonal projection approach used in OWM for dealing with catastrophic forgetting in continual learning and proposed a technique to resolve it. With this issue fixed, the classification accuracy improves. Extensive experiments showed that the proposed method AOP outperforms not only the original OWM, but also other state-of-the-art baselines markedly including even those using replay methods in both batch CL and online CL settings. AOP does not memorize any previous data or build data generators for generating pseudo training data. Orthogonal projection is a highly promising method for overcome catastrophic forgetting in continual learning. Our current technique only deals with one issue. We believe there is still a great deal of room for improvement. Our future work will focus on further improving this method.

Acknowledgments

The work was not supported by any grant.

References

- Adel, T.; Zhao, H.; and Turner, R. E. 2020. Continual Learning with Adaptive Weights (CLAW). *ICLR*.
- Ahn, H.; Cha, S.; Lee, D.; and Moon, T. 2019. Uncertainty-based Continual Learning with Adaptive Regularization. In *NeurIPS*.
- Aljundi, R.; Babiloni, F.; Elhoseiny, M.; Rohrbach, M.; and Tuytelaars, T. 2017. Memory Aware Synapses: Learning what (not) to forget. *arXiv preprint arXiv:1711.09601*.
- Aljundi, R.; Caccia, L.; Belilovsky, E.; Caccia, M.; Lin, M.; Charlin, L.; and Tuytelaars, T. 2019a. Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*.
- Aljundi, R.; Lin, M.; Goujaud, B.; and Bengio, Y. 2019b. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*.
- Belouadah, E.; and Popescu, A. 2019. IL2M: Class Incremental Learning With Dual Memory. In *ICCV*.
- Belouadah, E.; and Popescu, A. 2020. ScaLL: Classifier Weights Scaling for Class Incremental Learning. In *The IEEE Winter Conference on Applications of Computer Vision*.
- Benavides-Prado, D.; Koh, Y. S.; and Riddle, P. 2020. Towards Knowledgeable Supervised Lifelong Learning Systems. *Journal of Artificial Intelligence Research*, 68.
- Bendale, A.; and Boulton, T. 2015. Towards open world recognition. In *CVPR*, 1893–1902.
- Castro, F. M.; Marín-Jiménez, M. J.; Guil, N.; Schmid, C.; and Alahari, K. 2018. End-to-end incremental learning. In *ECCV*, 233–248.
- Chaudhry, A.; Khan, N.; Dokania, P. K.; and Torr, P. H. 2020. Continual learning in low-rank orthogonal subspaces. *arXiv preprint arXiv:2010.11635*.
- Chaudhry, A.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2019a. Efficient Lifelong Learning with A-GEM. In *ICLR*.
- Chaudhry, A.; Rohrbach, M.; Elhoseiny, M.; Ajanthan, T.; Dokania, P. K.; Torr, P. H.; and Ranzato, M. 2019b. Continual learning with tiny episodic memories.
- Chen, Z.; and Liu, B. 2014. Topic modeling using topics from many domains, lifelong learning and big data. In *ICML*.
- Chen, Z.; and Liu, B. 2018. *Lifelong Machine Learning*. Morgan & Claypool Publishers.
- Cohen, G.; Afshar, S.; Tapson, J.; and van Schaik, A. 2017. EMNIST: an extension of MNIST to handwritten letters. <http://arxiv.org/abs/1702.05373>.
- de Masson d’Autume, C.; Ruder, S.; Kong, L.; and Yogatama, D. 2019. Episodic Memory in Lifelong Language Learning. In *NeurIPS*.
- Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.
- Dhar, P.; Singh, R. V.; Peng, K.; Wu, Z.; and Chellappa, R. 2019. Learning without Memorizing. In *CVPR*.
- Eleftheriou, E.; and Falconer, D. 1986. Tracking properties and steady-state performance of RLS adaptive filter algorithms. *IEEE transactions on acoustics, speech, and signal processing*, 34(5): 1097–1110.
- Farajtabar, M.; Azizan, N.; Mott, A.; and Li, A. 2020. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, 3762–3773. PMLR.
- Fernando, C.; Banarse, D.; Blundell, C.; Zwols, Y.; Ha, D.; Rusu, A. A.; Pritzel, A.; and Wierstra, D. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.
- Hayes, T. L.; Kafle, K.; Shrestha, R.; Acharya, M.; and Kanan, C. 2019. REMIND Your Neural Network to Prevent Catastrophic Forgetting. *arXiv preprint arXiv:1910.02509*.
- Haykin, S. S. 2008. *Adaptive filter theory*. Pearson Education India.
- Hou, S.; Pan, X.; Loy, C. C.; Wang, Z.; and Lin, D. 2019. Learning a unified classifier incrementally via rebalancing. In *CVPR*, 831–839.
- Hu, W.; Lin, Z.; Liu, B.; Tao, C.; Tao, Z.; Ma, J.; Zhao, D.; and Yan, R. 2019. Overcoming Catastrophic Forgetting for Continual Learning via Model Adaptation. In *ICLR*.
- Hu, W.; Qin, Q.; Wang, M.; Ma, J.; and Liu, B. 2021. Continual Learning by Using Information of Each Class Holistically. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 7797–7805.
- Javed, K.; and Shafait, F. 2018. Revisiting distillation and incremental classifier learning. In *ACCV*, 3–17. Springer.
- Kamra, N.; Gupta, U.; and Liu, Y. 2017. Deep Generative Dual Memory Network for Continual Learning. *arXiv preprint arXiv:1710.10368*.
- Ke, Z.; Liu, B.; and Huang, X. 2020. Continual Learning of a Mixed Sequence of Similar and Dissimilar Tasks. In *NeurIPS*.
- Ke, Z.; Liu, B.; Ma, N.; Xu, H.; and Shu, L. 2021. Achieving Forgetting Prevention and Knowledge Transfer in Continual Learning. *Advances in Neural Information Processing Systems (NeurIPS-2021)*, 34.
- Kemker, R.; and Kanan, C. 2018. FearNet: Brain-Inspired Model for Incremental Learning. In *ICLR*.
- Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; and Others. 2017. Overcoming catastrophic forgetting in neural networks. volume 114, 3521–3526. National Acad Sciences.
- Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. *Technical Report TR-2009, University of Toronto, Toronto*.
- LeCun, Y.; Cortes, C.; and Burges, C. J. 1998. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.
- Lee, K.; Lee, K.; Lee, H.; and Shin, J. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NIPS*, 7167–7177.

- Lee, K.; Lee, K.; Shin, J.; and Lee, H. 2019. Overcoming Catastrophic Forgetting with Unlabeled Data in the Wild. In *ICCV*.
- Lee, S.-W.; Kim, J.-H.; Jun, J.; Ha, J.-W.; and Zhang, B.-T. 2017. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, 4655–4665.
- Lesort, T.; Caselles-Dupré, H.; Garcia-Ortiz, M.; Stoian, A.; and Filliat, D. 2018. Generative Models from the perspective of Continual Learning. <https://arxiv.org/abs/1812.09111>.
- Li, Z.; and Hoiem, D. 2017. Learning Without Forgetting. *PAMI*, 40(12): 2935–2947.
- Liu, Y.; Liu, A.-A.; Su, Y.; Schiele, B.; and Sun, Q. 2020. Mnemonics Training: Multi-Class Incremental Learning without Forgetting. *arXiv preprint arXiv:2002.10211*.
- Lopez-Paz, D.; and Ranzato, M. 2017. Gradient Episodic Memory for Continual Learning. In *NIPS*, 6470–6479.
- McCloskey, M.; and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning & motiv.*, volume 24.
- Misra, I.; and Maaten, L. v. d. 2020. Self-Supervised Learning of Pretext-Invariant Representations. In *CVPR*.
- Ostapenko, O.; Puskas, M.; Klein, T.; Jahnichen, P.; and Nabi, M. 2019. Learning to remember: A synaptic plasticity driven framework for continual learning. In *CVPR*, 11321–11329.
- Parisi, G. I.; Tani, J.; Weber, C.; and Wermter, S. 2018. Life-long Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization. *arXiv preprint*.
- Prabhu, A.; Torr, P. H.; and Dokania, P. K. 2020. Gdumb: A simple approach that questions our progress in continual learning. In *EECV*, 524–540.
- Qin, Q.; Hu, W.; Peng, H.; Zhao, D.; and Liu, B. 2021. BNS: Building Network Structures Dynamically for Continual Learning. *Advances in Neural Information Processing Systems (NeurIPS-2021)*, 34.
- Rajasegaran, J.; Hayat, M.; Khan, S.; Shahbaz, F.; and Shao, K. L. 2019. Random Path Selection for Incremental Learning. In *NeurIPS*.
- Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. icarl: Incremental classifier and representation learning. In *CVPR*, 2001–2010.
- Ritter, H.; Botev, A.; and Barber, D. 2018. Online structured laplace approximations for overcoming catastrophic forgetting. In *NIPS*, 3738–3748.
- Rolnick, D.; Ahuja, A.; Schwarz, J.; Lillicrap, T. P.; and Wayne, G. 2019. Experience Replay for Continual Learning. In *NeurIPS*.
- Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.
- Ruvolo, P.; and Eaton, E. 2013. ELLA: An efficient lifelong learning algorithm. In *ICML*.
- Saha, G.; Garg, I.; and Roy, K. 2021. Gradient Projection Memory for Continual Learning. *arXiv preprint arXiv:2103.09762*.
- Schwarz, J.; Luketina, J.; Czarnecki, W. M.; Grabska-Barwinska, A.; Teh, Y. W.; Pascanu, R.; and Hadsell, R. 2018. Progress & Compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*.
- Seff, A.; Beatson, A.; Suo, D.; and Liu, H. 2017. Continual learning in generative adversarial nets. *arXiv preprint arXiv:1705.08395*.
- Serrà, J.; Surís, D.; Miron, M.; and Karatzoglou, A. 2018. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*.
- Shim, D.; Mai, Z.; Jeong, J.; Sanner, S.; Kim, H.; and Jang, J. 2020. Online Class-Incremental Continual Learning with Adversarial Shapley Value. *arXiv preprint arXiv:2009.00093*.
- Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay. In *NIPS*, 2994–3003.
- Studer, L.; Alberti, M.; Pondenkandath, V.; Goktepe, P.; Kolonko, T.; Fischer, A.; Liwicki, M.; and Ingold, R. 2019. A Comprehensive Study of ImageNet Pre-Training for Historical Document Image Analysis. *arXiv:1905.09113*.
- Tao, X.; Hong, X.; Chang, X.; and Gong, Y. 2020. Bi-Objective Continual Learning: Learning ‘New’ While Consolidating ‘Known’. In *AAAI*, 5989–5996.
- von Oswald, J.; Henning, C.; Sacramento, J.; and Grewe, B. F. 2020. Continual learning with hypernetworks. *ICLR*.
- Wu, C.; Herranz, L.; Liu, X.; van de Weijer, J.; Raducanu, B.; et al. 2018. Memory replay GANs: Learning to generate new categories without forgetting. In *NIPS*, 5962–5972.
- Wu, Y.; Chen, Y.; Wang, L.; Ye, Y.; Liu, Z.; Guo, Y.; and Fu, Y. 2019. Large Scale Incremental Learning. In *CVPR*.
- Xu, J.; and Zhu, Z. 2018. Reinforced continual learning. In *NIPS*, 899–908.
- Zagoruyko, S.; and Komodakis, N. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146*.
- Zeng, G.; Chen, Y.; Cui, B.; and Yu, S. 2019. Continuous Learning of Context-dependent Processing in Neural Networks. *Nature Machine Intelligence*.
- Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. In *ICML*, 3987–3995.
- Zhang, J.; Zhang, J.; Ghosh, S.; Li, D.; Tasci, S.; Heck, L.; Zhang, H.; and Kuo, C.-C. J. 2020. Class-incremental Learning via Deep Model Consolidation. In *CVPR*.