Continual Learning by Using Information of Each Class Holistically

Wenpeng Hu[1,*], **Qi Qin**[2,*], **Mengyu Wang**[3], **Jinwen Ma**[1], **and Bing Liu**[3,†]
[1] Department of Information Science, School of Mathematical Sciences, Peking University
[2] Center for Data Science, AAIS, Peking University
[3] Wangxuan Institute of Computer Technology, Peking University
{wenpeng.hu, qinqi, wangmengyu, jwma, dcsliub}@pku.edu.cn

Abstract

Continual learning (CL) incrementally learns a sequence of tasks while solving the catastrophic forgetting (CF) problem. Existing methods mainly try to deal with CF directly. In this paper, we propose to avoid CF by considering the features of each class holistically rather than only the discriminative information for classifying the classes seen so far. This latter approach is prone to CF because the discriminative information for old classes may not be sufficiently discriminative for the new class to be learned. Consequently, in learning each new task, the network parameters for previous tasks have to be revised, which causes CF. With the holistic consideration, after adding new tasks, the system can still do well for previous tasks. The proposed technique is called *Per-class Continual Learning* (PCL). PCL has two key novelties. (1) It proposes a one-class learning based technique for CL, which considers features of each class holistically and represents a new approach to solving the CL problem. (2) It proposes a method to extract discriminative information after training to further improve the accuracy. Empirical evaluation shows that PCL markedly outperforms the state-of-the-art baselines for one or more classes per task. More tasks also result in more gains.

1 Introduction

Continual learning (CL) of a sequence of tasks in a neural network often suffers from *catastrophic forgetting* (CF) (McCloskey and Cohen 1989). CF means that in learning a new task, the network parameters learned for old tasks have to be modified, which can cause accuracy degrading for the old tasks. There are two main CL scenarios: *class continual learning* (CCL) and *task continual learning* (TCL), which are also called *class incremental learning* and *task incremental leaning* respectively. In both scenarios, each task consists of a number of classes. Once a task is learned, its training data (as least the bulk of it) is discarded or forgotten. In CCL, only a single classifier is built for all classes seen so far, which is used to classify each test case of any class without the task-id provided. In TCL, each task builds a separate classifier. In testing, each test case and its task-id are given so that the test case is classified only by the model of that task. *This paper works in the CCL scenario* and also assumes after learning a task, its training data is forgotten or no longer accessible in subsequent learning. The inaccessibility of the old task data could be due to many reasons, e.g., unrecorded legacy data, proprietary data, and privacy concerns such as in federated learning where the model can be shared from one party to another but the data must be kept private (Zhang et al. 2020).

There are three main approaches to dealing with CF (Chen and Liu 2018; Parisi et al. 2019). The first approach tries to avoid modifying those important parameters learned for old tasks in learning the new task (Kirkpatrick et al. 2017; Zenke, Poole, and Ganguli 2017; Li and Hoiem 2017). The second approach memorizes a small set of training data of each old task and use them in learning the new task (*replaying*) (Lopez-Paz and Ranzato 2017; Rusu et al. 2016; Rebuffi, Kolesnikov, and Lampert 2017). The third approach builds data generators to generate pseudo-examples for old tasks to be used used in learning the new task (Shin et al. 2017; Kamra, Gupta, and Liu 2017; Seff et al. 2017; Hu et al. 2019).

Before proposing our new approach, we first discuss a major cause for CF. Existing CL techniques mainly take the traditional approach of learning the discriminative information among classes *during training* to classify the classes. This approach is prone to CF because the discriminative information learned for old classes (from old tasks) may not be discriminative between the new classes and between old classes and the new classes. Thus, in learning the new classes, the network parameters for old classes have to be revised, which causes CF. The first approach for dealing with CF above tries to avoid revising those important parameters learned for old tasks (although it should), which leads to sub-optimal solutions. The second and third approaches both try to revise the old parameters using a small amount of saved old training examples or generated pseudo-examples.

Unlike the existing approaches, which deal with CF directly due to the above problem, this work proposes an entirely different approach, called *Per-class Continual Learning* (PCL), based on *one-class learning*, which aims to avoid CF altogether. **PCL makes two main contributions**. **First**, it proposes to use one-class learning for CCL. PCL learns one class at a time (*one class per task*) considering the features of the class holistically without overly biasing any specific features. This is crucial because the system does not know what

[*] Equal contribution

[†] Corresponding author. The work was done when Bing Liu was at Peking University on leave of absence from University of Illinois at Chicago, liub@uic.edu.

new classes may come in the future. Any commitment is premature and subject to change later, which causes CF. With the holistic consideration, each class can be identified naturally without resorting to any information from other classes for discrimination. This is achieved with the *one-class loss* in (Hu et al. 2020), which has a novel regularization called *H-reg* (*holistic regularization*). H-reg enables the learning algorithm to fully consider the features of each class, i.e., not to overly bias or favor any specific features. Sec. 3.4 extends PCL to make it learn with multiple classes per task.

It is important to note that ***learning one class at a time*** is probably the hardest case for CCL as it has the maximum number of tasks. As the number of tasks increases, the accuracy often drops quickly. It is perhaps also the most common case in practice because once a new class is encountered, we want to learn it immediately rather than wait for a few new classes to occur and learn them together. Thus, every CCL system should be able to learn one class per task well.

Second, it proposes a method to extract discriminative information *after training* for classification in testing by reducing the shared knowledge among classes as the shared knowledge blurs the decision boundary. This is useful because training each class separately is not ideal for classification, for which discriminative information is still more effective. This operation is enabled by parameter transfer from old classes to the new class in initialization, and it does not change the trained models and thus does not cause CF as the extraction is done after learning the new class.

PCL architecture consists of a pre-trained model or feature extractor (although it is not required) as the base and classification heads for the classes learned so far, one head per class. Due to the rich representation in the pre-trained model, in learning a new class, the base stays unchanged and only a new head is added and trained with the data of the class. Using a pre-trained feature extractor has been very popular in natural language processing (NLP). For example, in the past two years, the NLP field has been transformed by pre-trained models such as BERT (Devlin et al. 2019), and ALBERT (Lan et al. 2020). With the success in NLP, pre-trained features have also become very popular recently in computer vision (Studer et al. 2019; Misra and Maaten 2020; He et al. 2020). Note that PCL also outperforms baselines without pre-trained feature extractors (see Sec. 4).

Experimental evaluation using 4 image classification and 2 text classification datasets show that PCL outperforms state-of-the-art baselines with or without pre-trained feature extractor being used by PCL and the baselines with one or more classes per task.

2 Related Work

Due to the need to revise the old discriminative information when a new task/class is learned, existing approaches deal with CF directly. Our PCL avoids CF by exploiting the information in each class holistically using one-class learning. Although Progressive Networks also tries to avoid CF by building independent models and then making connections among them (Rusu et al. 2016). Its network size scales quadratically in the number of tasks. It cannot learn with one class per task. Some methods also transfer knowledge to new tasks (e.g., (Schwarz et al. 2018; Fernando et al. 2017; Rusu et al. 2016)) when the tasks similar, which is not the focus of this paper. Below, we review the other existing approaches (also see the surveys (Chen and Liu 2018; Parisi et al. 2019)).

Most techniques deal with CF by keeping the important parameters for old tasks minimally changed to maintain the old knowledge. For example, EWC (Kirkpatrick et al. 2017) quantifies the importance of weights (parameters) to old tasks, and selectively alters the learning rates of weights to protect the important weights. Many papers dealing with CF use a similar idea (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; Dhar et al. 2019; Adel, Zhao, and Turner 2020).

Zeng et al. (2019) proposed to find orthogonal projections of weight updates that do not disturb the weights of old tasks. Knowledge distillation was used by LwF (Li and Hoiem 2017) and others (Wu et al. 2019; Castro et al. 2018; Belouadah and Popescu 2019; Liu et al. 2020; Lee et al. 2019; Tao et al. 2020). PCL avoids CF and achieves better results.

Another body of work memorizes a small sample of the training data of each old task. This is called *experience replay*. Representative work includes GEM (Lopez-Paz and Ranzato 2017), A-GEM (Chaudhry et al. 2019), and many others (Rusu et al. 2016; Rebuffi, Kolesnikov, and Lampert 2017; Wu et al. 2019; Rolnick et al. 2019; de Masson d'Autume et al. 2019; Hou et al. 2019). Some methods also learn to generate pseudo old data and use them to jointly train with the new task data (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) (also known as rehearsal (von Oswald et al. 2020)). Our work does not save any old data or build any data generators.

There is a body of work called *class incremental learning* (CIL), which is basically another name for CCL. Like CCL, most methods under CIL also learn multiple classes per task. One typical approach is to memorize some exemplars of old tasks to perform knowledge distillation (Wu et al. 2019; Castro et al. 2018; Liu et al. 2020; Lee et al. 2019; Belouadah and Popescu 2019, 2020) to deal with CF. Another typical approach, which learns one class per task like PCL, memorizes some random or selected exemplars or the mean of each class to represent the class. In testing, a distance function over *nearest exemplar/class mean* is used for classification (Rebuffi, Kolesnikov, and Lampert 2017; Lee et al. 2018; Javed and Shafait 2018; Bendale and Boult 2015). PCL does not use any of the approaches or memorize any exemplars.

Some methods dynamically expand the network in learning each new task (Li and Hoiem 2017; Yoon et al. 2018; Li et al. 2019; Ostapenko et al. 2019), which PCL also does, but our addition is very small. Although many papers do not mention network expansion, since they protect old parameters, as the number of tasks increases the network has to be expanded.

This paper focuses on CCL. There are also many TCL techniques (Fernando et al. 2017), e.g., GEM (Lopez-Paz and Ranzato 2017), A-GEM (Chaudhry et al. 2019), HAT (Serrà et al. 2018), UCL (Ahn et al. 2019) and CAT (Ke, Liu, and Huang 2020). CAT is able to perform knowledge transfer. Some applications in sentiment analysis have been reported

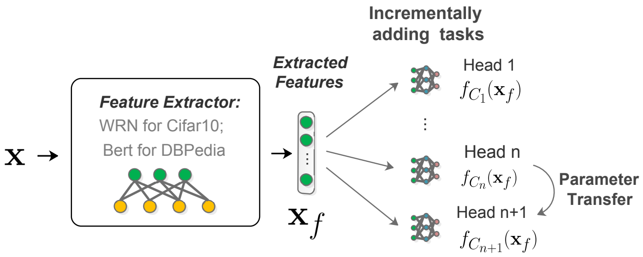

Figure 1: PCL architecture. Each class has a separate head.

in (Ke et al. 2020; Qin, Hu, and Liu 2020), which focus on knowledge transfer. There is also a less frequently used CL setting (Lv et al. 2019; van de Ven and Tolias 2019), which is like TCL but the task-id is not provided during testing.

Since PCL learns one class at a time, it is related to one-class learning (Schölkopf et al. 2001; Tax and Duin 2004; Hu et al. 2020; Perera, Nallapati, and Xiang 2019; Ruff et al. 2018). PCL mainly uses the one-class classification method in (Hu et al. 2020). However, one-class learning does not learn a sequence of tasks or deal with CF. For CL, significant changes need to be made to one-class learning.

Traditional lifelong learning methods are also related (Ruvolo and Eaton 2013; Chen and Liu 2014; Benavides-Prado, Koh, and Riddle 2020). However, they do not deal with CF.

3 Proposed PCL Model

The proposed PCL architecture is shown in Fig. 1, which consists of a pre-trained Feature Extractor shared by all tasks or classes, and Class Heads following it, one head for each class learned so far. In our experiments, the same pre-trained feature extractors are used for all baselines, and PCL also outperforms baselines without pre-trained feature extractors. Below, we first present learning with one class per task and then extends it to learning with multiple classes per task.

Formally, we denote the head for each class C_i as $f_{C_i}(\cdot)$. Each head is an independent network with a simple structure (a two-layer MLP with a single output unit following (Zeng et al. 2019; Hu et al. 2019) and a small number of parameters which ensures that adding new tasks will not lead to a huge model. In testing, given a test instance $\mathbf{x}$, we choose the head that gets the highest $f_{C_i}(\cdot)$ output value as the class of $\mathbf{x}$, i.e.,

$$y = \operatorname*{argmax}_{C_i}[f_{C_1}(\mathbf{x}_f), \ldots, f_{C_N}(\mathbf{x}_f)]. \tag{1}$$

where N is the total number of classes learned so far; $\mathbf{x}_f$ denotes the feature obtained by the pre-trained feature extractor: $\mathbf{x}_f = \mathcal{F}(\mathbf{x})$. $\mathbf{x}$ denotes the input data.

PCL has three key advantages: (1) it is easy to expand the network capacity. As the number of classes (or tasks) increase, we can simply add a separate head for each new class, which is very small in size as we will see later; (2) as PCL exploits features holistically based on one-class learning, it enables the system to avoid CF as no modification or change to the shared feature extractor or the old models (heads) is needed when learning a new class; (3) PCL outperforms baselines with or without pre-trained feature extractors. To make the architecture work effectively, we use a novel loss function called *one-class loss*, which has been used in (Hu et al. 2020) for one-class learning. We discuss it next.

3.1 One-class Loss

In learning each class C_i, the one-class loss is:

$$\mathcal{L} = \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}}[-\log(\mathrm{S}(f_{C_i}(\mathbf{x})))]}_{\text{NLL}} + \lambda \cdot \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} \|\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})\|_2^n}_{\text{H-reg}} \tag{2}$$

where $\mathbb{P}_{\mathbf{x}}^{C_i}$ denotes the data distribution of class C_i. The input data can be the extracted features $\mathbf{x}_f$ using pre-trained feature extractor or the original data $\mathbf{x}$. To simplify the notation, we use $\mathbf{x}$ to denote both. $\mathbf{x}$ is also normalized, $\mathbf{x} := \mathbf{x}/||\mathbf{x}||_2$, in our experiments. Exponent n and λ are hyper-parameters controlling the strength of the penalty and balancing the regularization respectively. $\mathrm{S}(\cdot)$ is the Sigmoid function, and $\mathrm{S}(f_{C_i}(\mathbf{x})) \in (0,1)$ can be seen as the probability of $\mathbf{x}$ belonging to C_i. We now explain the two terms in Eq. (2).

NLL (*Negative Log Likelihood* for one-class). Minimizing NLL means to train the model $f_{C_i}(\cdot)$ to output high values for the training data of the class, which, according to PCL's decision rule in Eq. (1), helps recognize instances from class C_i in testing. However, since we have only one class of data, only optimizing NLL leads to two major problems.

Problem-I (*incomparable outputs* for $f_{C_i}(\mathbf{x})$). The magnitudes of the parameter values cannot be controlled, which are sensitive to the input data. This can cause the final outputs $f_{C_i}(\mathbf{x})$ to be arbitrary and uncontrollable for different classes and make $f_{C_i}(\mathbf{x})$ values for different classes not comparable, but comparability is very important for classification decision making of PCL (see Eq. (1)). This leads to poor accuracy results as we will see in Sec. 4.5.

Problem-II (*feature bias*). Features (or dimensions) of the input data with high values are very likely to be emphasized by the classification head and their related parameters are likely to have very high values. But those features with high values may not be important features for recognizing the correct class of the input instance, which can lead to low accuracy. This problem is due to the fact that we don't have other classes to compare with in order to identify the important or discriminative features.

Holistic regularization (H-reg).[1] H-reg aims to solve the above two problems. For *Problem*-I, assume the head C_i is a two-layer MLP with a single output unit (which is the case in PCL) and $\sigma(\cdot)$ is the activation function. Then, we can show $f_{C_i}(\mathbf{x}) = \mathbf{w}_2 \cdot \sigma(\mathbf{w}_1\mathbf{x})$, where $\mathbf{w}_1$ and $\mathbf{w}_2$ are the parameters of the first and second layer respectively. Thus, we have:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} \|\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})\|_2^n = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} \|\mathbf{w}_2 \cdot \nabla_{\mathbf{w}_1\mathbf{x}}\sigma(\mathbf{w}_1\mathbf{x}) \cdot \mathbf{w}_1\|_2^n. \tag{3}$$

The exact expression depends on the activation function. For ReLU (used in PCL), the elements in $\nabla_{\mathbf{w}_1\mathbf{x}}\sigma(\mathbf{w}_1\mathbf{x})$ are either 1 ($\mathrm{ReLU}(\mathbf{w}_1\mathbf{x}) > 0$) or 0 ($\mathrm{ReLU}(\mathbf{w}_1\mathbf{x}) \leq 0$). Let us first consider $\nabla_{\mathbf{w}_1\mathbf{x}}\sigma(\mathbf{w}_1\mathbf{x}) \equiv 1$ for all elements. We have:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} \|\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})\|_2^n = \|\mathbf{w}_2 \cdot \mathbf{w}_1\|_2^n. \tag{4}$$

Clearly, H-reg can constrain the arbitrary growth of $\mathbf{w}_1$ and $\mathbf{w}_2$ parameter values and consequently the arbitrary growth

[1] H-reg has some resemblance to L2 regularization. We will see L2 is significant poorer in Sec. 4.

and magnitudes of $f_{C_i}(\cdot)$ for different classes because the arbitrary growth of the parameter values will lead to high penalties on H-reg and thus high losses, i.e., a trade-off between NLL and H-reg. Specifically, a high parameter value leads to a high $f_{C_i}(\cdot)$ and thus a low NLL, but a high value for H-reg. Therefore, the training goal of the one-class loss is to find a point where $f_{C_i}(\cdot)$ outputs a value as high as possible under the condition of having parameters with values as small as possible. Equivalently, it is to achieve $Sigmoid(f_{C_i}(\cdot))$ close to 1 while $f_{C_i}(\cdot)$ as small as possible. This is achievable as $Sigmoid(f_{C_i}(\cdot))$ flattens out after $f_{C_i}(\cdot)$ reaches a certain value. Since we do this in learning every class and also due to the input normalization $\mathbf{x} := \mathbf{x}/||\mathbf{x}||_2$, H-reg can bring $f_{C_i}(\cdot)$ values for different classes to a comparable level (a kind of calibration), which solves *Problem*-I above.

When $\nabla_{\mathbf{w}_1\mathbf{x}}\sigma(\mathbf{w}_1\mathbf{x}) \equiv 1$ for all elements is not true, the 0 valued elements in it simply block some neurons/units, which we can ignore because the blocked neurons have no contributions to the final $f_{C_i}(\cdot)$ output. Note that we suggest using piecewise linear function as the activation function, e.g., ReLU and Leaky-ReLU, as both Sigmoid and Tanh are too flat for high input values. Take Sigmoid as an example, $\nabla_{\mathbf{w}_1\mathbf{x}}\sigma(\mathbf{w}_1\mathbf{x}) = \sigma(\mathbf{w}_1\mathbf{x})(1-\sigma(\mathbf{w}_1\mathbf{x}))$. If $\mathbf{w}_1$ is already biased (with high values), the regularization tends to be blocked as we are likely to get 0 for the right-hand-side.

For *Problem*-II, as we know, the derivative $\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})$ shows the importance of each feature of $\mathbf{x}$. The features with large derivatives contribute more to the final output as small changes in them can lead to large changes in the $f_{C_i}(\mathbf{x})$ outputs and they also give large values for H-reg, which is undesirable for loss minimization. In this case, minimizing H-reg can ease the problem that the output is dominated by some specific features of the input $\mathbf{x}$. We can reach this conclusion using Eq. (4), the dimensions in $\mathbf{w}_2 \cdot \mathbf{w}_1$ corresponding to the contributions of the same feature dimensions of the input. In this case, the output will not be saturated by a few features of the input due to the H-reg expressed as the right-hand-side of Eq. (4). In addition to this, since the L2-norm in Eq. (4) gives more penalties to the features with high values and little penalty to the features with low values, the parameter values will be more balanced.

3.2 Parameter Transfer in Initialization

Although H-reg has the effect of not biasing any input features in learning each class, for classification, using discriminative features is still more effective. We discuss how to obtain such discriminative information for classification in testing in Sec. 3.3. To achieve that goal, in training each new class C_i, the network parameters θ_i of its model/head cannot be initialized randomly without control because that will result in the parameters of different networks not in the same parameter space and thus are not comparable.

We borrow a transfer learning technique, and make the assumption of Gaussian distribution for neural networks parameters (Lee et al. 2017) to ease this problem. Specifically, when learning a new class C_i, we use the mean of the model parameters from class 1 to $i-1$, i.e., $\mu^*_{1:i-1}$, to initialize θ_i, and L2-transfer in (Evgeniou and Pontil 2004; Kienzle and Chellapilla 2006) is also added to Eq. 2,

$$\mathcal{L} = \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}}[-\log(\mathrm{S}(f_{C_i}(\mathbf{x})))]}_{\text{NLL}} + \lambda \cdot \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} ||\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})||_2^n}_{\text{H-reg}} \\ + \eta \cdot \underbrace{||\theta_i - \mu^*_{1:i-1}||_2^2}_{\text{L2-transfer}} \qquad (5)$$

where η is a hyper-parameter.

3.3 Discriminative Information Extraction (DIE) for Classification in Testing

Since in training each class, we do not see the data of other classes, models for different classes may learn some similar knowledge as data of different classes may have some commonalities. Such commonalities blur the boundary between different classes and make the classification challenging. For classification in testing, we should reduce such commonalities to obtain more discriminative information among the classes learned so far for classification.

Inspired by (Lee et al. 2017, 2020), we propose a post-processing step to eliminate the *Shared Knowledge* to reduce their negative effects on classification in testing (no change to training). Specifically, we assume that the posterior distribution of the parameters is Gaussian $q_i \equiv q(\theta_i|\mu_i, \Sigma_i)$ for each class C_i. We minimize the following local KL-distance or the weighted sum of KL-divergence between each q_i and $q_{1:N}$ (N is the total number of classes learned so far):[2]

$$\mu^*_{1:N}, \Sigma^*_{1:N} = \mathop{\text{argmin}}_{\mu_{1:K}, \Sigma_{1:N}} \sum_{i=1}^{N} \alpha_i \cdot KL(q_i||q_{1:N}) \qquad (6)$$

where α_i is the mixing ratio with $\sum_{i=1}^{N} \alpha_i = 1$ (we use $1/N$ in our experiment). The optimal solution of the local KL-distance is $\mu^*_{1:N} = \sum_{i=1}^{N} \alpha_i \mu_i$. We take $\mu^*_{1:N}$ as the shared knowledge among all the classes and use it to adjust the original parameters of each model:

$$\theta^*_i = \theta_i - \gamma \cdot \mu^*_{1:N} \qquad (7)$$

where γ is a small value, we set it to 0.1 in our experiments.

In testing, we use θ^*_i as the parameters of the i^{th} model/class rather than the original θ_i in the network. However, θ_i is not physically changed to θ^*_i, but computed using Eq. (7) when needed in testing. Note that $\mu^*_{1:N}$ can be easily incrementally computed and maintained.

3.4 Learning with Multiple Classes Per Task

We now extend PCL to also learn with multiple classes per task, i.e., task $T_j = \{C_t, \cdots, C_{t+k_j}\}$ with k_j classes in the task, $t = 1 + \sum_{r=1}^{j-1} k_r$ is the beginning of task j. For each task T_j, we first learn each class $C_{t+h} \in T_j$ as above, $f_{C_{t+h}}(\mathbf{x})$, and then use supervised learning with the cross-entropy as the loss function to learn a model $S_j(\cdot) \in \mathbb{R}^{k_j}$ for all classes in T_j. In testing, for each test instance $\mathbf{x}$, we first find the class C^j with the highest probability from each task model $S_j(\mathbf{x})$. Assume M tasks have been learned so far, the class y for the test instance $\mathbf{x}$ is computed as follows,

$$y = \mathop{\text{argmax}}_{C^j}[f_{C^1}(\mathbf{x}), \ldots, f_{C^M}(\mathbf{x})] \qquad (8)$$

[2] Given a sequence of N classes, $q_{1:N}$ denotes an approximation of the true posterior distribution $p(\theta|C_1, \cdots, C_N)$ for the classes.

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
20news	20	11,314	377-600	7,532
DBPedia	14	560,000	40,000	70,000

Table 1: Datasets details

3.5 Number of Parameters

For the model of each class, a simple 2-layer MLP can already achieve good results. Each new class only expands the network by the small MLP (see Sec. 4.1), i.e., about 0.0641M parameters for each class for CIFAR10/CIFAR100 dataset. For the case of learning with multiple classes per task, the number of parameters is doubled, which is still very small.

4 Experiments

We now evaluate the proposed PCL technique (the code can be found here[3]) and compare it with both classic and the latest baselines with or without pre-trained feature extractors in learning with one or more classes per task. To make the comparison more complete, we also compare with the traditional nearest-mean approach that learns one class at a time incrementally.

Experimental Datasets: We use four benchmark image classification datasets and two text classification datasets in our experiments: **MNIST** (LeCun, Cortes, and Burges 1998), **EMNIST-47** (Cohen et al. 2017), **CIFAR10** and **CIFAR100** (Krizhevsky and Hinton 2009) for images; **20news** and **DBPedia** for text. 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.

Compared Baselines: We use the following classic and the latest state-of-the-art class continual learning (CCL) baselines. (1) **EWC** (Kirkpatrick et al. 2017) is a commonly used baseline in most CL papers. (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. (5) **RPSnet** (Rajasegaran et al. 2019) progressively chooses optimal paths for each new task while encouraging parameter sharing. This is the latest replay method that saves some training examples from previous tasks. (6) **OWM** (Zeng et al. 2019) is based on the idea of orthogonal data projection and has been shown to perform very well for a large number of tasks. (7) **HNET** (von Oswald et al. 2020) is a latest method that can work in the continual/incremental learning (CCL) setting without memorizing/generating old data but only the embeddings of old tasks. (8) **PCL-L2** is PCL with its H-reg replaced by the popular L2 regularization. See two more baselines in Sec. 4.4 when we compare with the traditional nearest-mean approach.

It is important to note that most CCL methods can work with one or more classes per task because when new classes (one or more) are added, training uses the same cross entropy loss considering all classes without using the data of old classes. For our baselines, LwF cannot 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 code released by their authors except EWC as the original code was not released, for which we use a popular third party code.[4]

Evaluation protocol: Following the existing CL evaluation method, for each dataset, after all tasks are learned, we test using the test sets of all tasks and report the average accuracy over 5 runs.

4.1 Training Details

For a fair comparison, our PCL uses the same classification model as the baselines. Specifically, following (Zeng et al. 2019; Hu et al. 2019), we use a MLP with two layers and a single output unit as the classification model after the shared feature extractor. As PCL and LwF grow the network with the increase of the arriving tasks, given a sequence of N tasks, assuming the size of the hidden layer for non-growth methods is m, we set the hidden size of our method and LwF for each task as m/N. The parameter size of all the methods will be of the same magnitude after learning all tasks. We fix m/N to 100.[5] For training, we use SGD with moment as the optimizer (learning_rate = 0.1). We run each experiment five times. For each run of PCL or a baseline, we execute 500 epochs and use the maximum accuracy as the final result of the run. We report the average result of the five runs. For text data, we use the TF-IDF vector of the top 2000 most frequent words to represent a document. We discuss the use of pre-training features later. Additionally, for 20news, we removed the headers, footers and quotes as those parts have explicit class label information.

Hyper-parameter Tuning: PCL has 3 parameters that need tuning: λ and n in H-reg (Sec. 3.1) and η for transfer (Sec. 3.2). We randomly select 10% of the examples from the training set of each dataset as the validation set to tune the hyper-parameters. After that, we use the tuned hyper-parameters to train the system over the whole training set. Grid search is used in tuning. The tuning range for λ is from 0 to 1 with step 0.1; n is from 1 to 20 with step 1; η is from 0 to 0.02 with step 0.0005. After tuning, we get the best hyper-parameters of $\lambda = 0.5$ and $n = 12$. For η, different data have different values, 0.001 for MNIST and EMNIST-47, 0.005 for CIFAR10 and DBPedia, 0.01 for CIFAR100 and 20news.

[3] https://github.com/morning-dews/PCL

[4] https://github.com/moskomule/ewc.pytorch

[5] The model size for each class in PCL is [InputSize-100]-[100-1], which is very small, e.g., 0.0641M for CIFAR10 and CIFAR100. For OWM, the model size is [InputSize-100*N]-[100*N-N]. The network sizes of PGMA and OWM are significantly larger than ours as they also have additional peripheral modules. Note that reducing the number m/N leads to worse results for both PCL and baselines.

Dataset	w/o PTF	EWC	LwF	IMM	PGMA	RPSnet	OWM	PCL-L2	PCL
MNIST (10 tasks)	no	9.91	19.96	29.16	71.36	40.29	94.46	83.85	**97.00**
EMNIST-47 (47 tasks)	no	2.13	4.59	18.69	10.13	10.08	77.45	51.38	**80.05**
CIFAR10 (10 tasks)	yes	10.21	19.39	51.22	56.22	55.54	83.03	77.95	**84.93**
CIFAR100 (100 tasks)	yes	2.93	6.25	12.58	12.37	4.13	63.26	54.83	**63.61**
20news (20 tasks)	no	4.98	5.61	5.00	11.37	8.32	52.02	49.01	**54.37**
DBPedia (14 tasks)	yes	7.14	7.14	7.14	66.40	50.58	95.37	68.12	**96.23**
CIFAR10 (10 tasks)	no	10.01	10.05	10.25	20.08	16.31	19.63	10.00	**31.58**
CIFAR100 (100 tasks)	no	1.03	2.13	1.21	1.86	1.96	3.67	1.87	**5.58**
DBPedia (14 tasks)	no	7.14	7.14	7.14	9.58	36.70	92.23	64.96	**93.51**

Table 2: Accuracy results for 1 class per task for PCL and all baselines except HNET as it does not work with one class per task. Note that column "w/o PTF" denotes with/without using the shared pre-trained feature extractor for PCL and baselines.

4.2 Results for One Class Per Task

Pre-trained feature extractors: Pre-trained feature extractors have been frequently used in computer vision (Studer et al. 2019; Misra and Maaten 2020) and natural language processing (NLP) (Devlin et al. 2019). We now apply pre-trained feature extractors to PCL and all baselines.

For CIFAR10 and CIFAR100, we pre-train a WRN model to extract features with size 640 (Zagoruyko and Komodakis 2016) using ImageNet after *manually removing classes from ImageNet that are similar to those classes in CIFAR10 or CIFAR100*. After removal, we are left with 771 ImageNet classes. No pre-trained feature extractors for MNIST and EMNIST-47 as a simple model already generates very good results. For the DBPedia text, we use the BERT (Devlin et al. 2019) feature extractor (the feature size is 768). BERT was not effective for 20news as it has too many symbols that have no embeddings in BERT and the dataset also has very long texts (maximum being over 15000). 'w/o PTF' in the table means with or without using a pre-trained feature extractor.

The first block of results in Table 2 are the accuracy values for learning with one class per task in the above setting. Each experiment is done 5 times and the average accuracy of the 5 runs is reported for each dataset and each model. *Note again that when pre-trained feature extractor is used, it is used in PCL and also in all baselines.* From the first block of the results in the table, we can make the following observations:

(1). 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 continual/incremental learning methods can naturally learn with one class per task.

(2). OWM is the strongest baseline, but PCL significantly outperforms it on all datasets with p-value < 0.01 on paired t-test. PCL gets a 2.54 points improvement on MNIST, 2.60 points on EMNIST-47, 1.89 points on CIFAR10, 2.35 points on 20news, and 1.28 points on DBPedia.

(3). PCL-L2 (PCL's H-reg is replaced with L2 regularization) does not do well for this setting (and nor for 2-class per task below). One reason is that there is still a very high $\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{x}}^{C_i}} \|\nabla_{\mathbf{x}} f_{C_i}(\mathbf{x})\|_2$, e.g., up to 5.11 on MNIST, but only 0.736 when optimizing H-reg, which shows that the output of $f_{C_i}(\cdot)$ is sensitive to the input $\mathbf{x}$. For example, $\Delta\mathbf{x} = 0.2$ will lead to an output change of up to 1.02 which is much higher than 0.147 for H-reg.

Without using a pre-trained feature extractor for CIFAR10, CIFAR100 or DBPedia: The second block of of results in Table 2 gives the accuracy values of the three datasets. We can observe that PCL again did much better, but all the results (baselines and PCL) are very low except for DBPedia of PCL. This shows that pre-trained feature extractors are very useful. Without them, the accuracy is too low to be of practical use. With pre-trained feature extractor, the performances of PCL and the baselines are all greatly improved. For PCL, the average accuracy improves from 31.58 to 84.93 on CIFAR10, and from 5.58 to 63.61 on CIFAR100. For baselines, we can also see clear improvements.

4.3 Results for More Classes Per Task

We now report the accuracy results for more than one class per task. Our main goal is to test every system's performance when the number of tasks is large, which is probably the most important criterion for evaluating CL methods. PCL uses the method in Sec. 3.4. As above, for MNIST, EMNIST-47 and 20news, we still use the original data. Pre-trained feature extractors are used by PCL and baselines only for CIFAR10, CIFAR100, and DBPedia as using pre-trained feature extractors produce better results for the systems.

Two classes per task. The results for two classes per task are given in Table 3 except HNET. HNET achieves 95.30 on MNIST which we could reproduce and is already poorer than our PCL (97.20) but better than most baselines. However, its accuracy drops quickly with more tasks. It gets only 25.42 on EMNIST-47, much worse than PCL (80.97) with 24 tasks. Its CIFAR10 (51.02) and CIFAR100 (3.22, 50 tasks) are also very poor, so are the two text data. 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 DBPedia, 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. PCL consistently outperforms all baselines. Note that learning multi-classes per task can always be replaced by learning 1 class at a time. Comparing the results of PCL in the 1 class setting (the first block in Table 2) and baselines in 2 classes setting (Table 3), we see that PCL's 1 class results are already better than those of the baselines' 2 classes results except RPSnet on CIFAR10, but RPSnet on a large number of tasks is quite weak.

Comparing the performance in this setting with the 1 class per task setting of PCL, we see that PCL works better in

Model	MNIST	EMNIST-47	CIFAR10	CIFAR100	20news	DBPedia
(Number of tasks)	(5 tasks)	(24 tasks)	(5 tasks)	(50 tasks)	(10 tasks)	(7 tasks)
EWC (Kirkpatrick et al. 2017)	18.78	4.17	31.82	3.67	6.88	14.11
LwF (Li and Hoiem 2017)	52.35	17.19	57.61	23.33	26.25	69.82
IMM (Lee et al. 2017)	67.25	20.51	77.29	26.52	24.60	73.79
PGMA (Hu et al. 2019)	81.70	21.85	74.31	17.47	13.10	83.06
RPSnet (Rajasegaran et al. 2019)	96.16	32.97	85.37	25.27	44.99	96.21
OWM (Zeng et al. 2019)	91.62	71.68	83.36	57.70	49.92	94.79
PCL-L2	85.35	53.02	79.35	55.21	50.52	68.23
PCL	**97.20**	**80.97**	**85.78**	**63.72**	**54.72**	**96.91**

Table 3: Accuracy for 2 classes per task using pre-trained features for PCL and baselines except HNET (see the reason in text)

Model	EMNIST-47	CIFAR100
(Number of tasks)	(10 tasks)	(20 tasks)
PGMA	17.80	29.63
RPSnet	74.82	51.44
OWM	58.00	49.16
PCL	**81.21**	**63.90**

Table 4: Accuracy for 5 classes per task using pre-trained features for PCL and 3 top baselines using EMNIST-47 and CIFAR100 as they have a large number of classes

Components	MNIST	EMNIST-47	CIFAR10	CIFAR100
NLL	11.35	2.73	9.10	1.22
NLL + *H-reg*	89.58	68.72	82.80	59.10
NLL + *H-reg* + $\mathbf{x}$-*N*	96.31	79.12	84.11	61.57
NLL + *H-reg* + $\mathbf{x}$-*N* + *DIE* (PCL)	**97.00**	**80.05**	**84.93**	**63.61**

Table 5: Accuracy results of ablation study of PCL using MNIST, EMNIST-47, CIFAR10 and CIFAR100. $\mathbf{x}$-N: normalization of $\mathbf{x}$, and DIE: discriminative information extraction (Sec. 3.2 and 3.3)

this 2 classes per task setting. All baselines have improved (including PCL-L2) too. This is because with 2 classes traditional supervised learning can be performed and the number of tasks is halved. PCL does not improve as much because its 1 class per task setting is already quite strong.

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 observe from Table 4 that PCL outperforms all 3 top-performing baselines.

4.4 Comparing with the Nearest-Mean Approach

To be more complete, we also compare with two traditional methods of *class incremental learning* based on nearest-mean: iCaRL (Rebuffi, Kolesnikov, and Lampert 2017) and the system in (Lee et al. 2018). Both save some examples or the means of old classes. PCL and the baselines above save no information of old classes. PCL outperforms them both.

iCaRL (Rebuffi, Kolesnikov, and Lampert 2017) 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 1 class per task learning and 2 classes per task learning on MNIST/EMNIST-47/CIFAR10/CIFAR100 respectively, but the results of PCL are 97.0/80.5/84.93/63.61 and 97.20/80.97/85.78/63.72 respectively. iCaRL did very poorly on the text data as it was not designed for text.

Lee et al. (2018) proposed a Mahalanobis distance-based score using the saved means and a shared covariance for classification. Its continual learning code was not released. The setup of this technique is very different. For comparison, we follow its setup. For CIFAR100, it takes the first 50 classes to do pre-training and then starts continual learning for the rest of the 50 classes one by one. After each new class is learned, it computes the AUC (Area under the ROC Curve) of the new class and draws a curve as the system learns the 51th class through the 100th class. Specifically, the AUCs of their system go from 0.79 (51th task) to about 0.40 (100th task), which are much lower than PCL's results that go from 0.80 (51th task) to 0.75 (100th task). For CIFAR10, their system used CIFAR100 for pre-training. After all 10 CIFAR10 classes are learned, it reported the average AUC score of 0.477, but PCL's average AUC score is 0.627.

4.5 Ablation Study

We use MNIST, EMNIST-47, CIFAR10 and CIFAR100 (pre-trained features are used for CIFAR10 and CIFAR100) with one class per task for ablation study. Table 5 shows the results of adding different components of PCL. We can see that all of them are useful. *H-reg* is highly effective. It helps improve the accuracy of PCL drastically, e.g., by 78.23 points for MNIST and 73.70 points for CIFAR10. *Data normalization* also made a big difference. Adding DIE (*Discriminative Information Extraction*) enabled by parameter transfer (Sec. 3.2 and 3.3) and improved the results further. More detailed analysis and insights about the effect of H-reg, normalization and DIE as well as error analysis are given in *Appendix A*.

5 Conclusion

This paper proposed a novel method, called PCL, for class continual learning (CCL). It has two key novelties: using an one-class learning for CCL, which can force the learning algorithm to fully or holistically consider the features of each class, i.e., not to bias/favor any specific features, and (2) discriminative information extraction after training. PCL learns with any number of classes per task, and it outperforms the latest baselines using both image and text datasets. In our future work, we plan to further improve the accuracy.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under grant 2018AAA0100205.

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* .

Belouadah, E.; and Popescu, A. 2019. IL2M: Class Incremental Learning With Dual Memory. In *ICCV*.

Belouadah, E.; and Popescu, A. 2020. ScaIL: 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 Boult, 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.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2019. Efficient Lifelong Learning with A-GEM. In *ICLR*.

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*.

Evgeniou, T.; and Pontil, M. 2004. Regularized multi–task learning. In *KDD*, 109–117.

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* .

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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.; Wang, M.; Qin, Q.; Ma, J.; and Liu, B. 2020. HRN: A Holistic Approach to One Class Learning. *Advances in Neural Information Processing Systems (NeurIPS)* 33.

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.; Wang, H.; and Shu, L. 2020. Continual Learning with Knowledge Transfer for Sentiment Classification. In *ECML-PKDD*.

Kemker, R.; and Kanan, C. 2018. FearNet: Brain-Inspired Model for Incremental Learning. In *ICLR*.

Kienzle, W.; and Chellapilla, K. 2006. Personalized handwriting recognition via biased regularization. In *ICML*, 457–464.

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.* .

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2020. ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations. In *ICLR*.

LeCun, Y.; Cortes, C.; and Burges, C. J. 1998. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/* .

Lee, J.; Joo, D.; Hong, H. G.; and Kim, J. 2020. Residual Continual Learning. *arXiv preprint arXiv:2002.06774* .

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, X.; Zhou, Y.; Wu, T.; Socher, R.; and Xiong, C. 2019. Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting. In *ICML*.

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.

Lv, G.; Wang, S.; Liu, B.; Chen, E.; and Zhang, K. 2019. Sentiment Classification by Leveraging the Shared Knowledge from a Sequence of Domains. In *DASFAA*, 795–811.

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.; Puscas, 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.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* .

Parisi, G. I.; Tani, J.; Weber, C.; and Wermter, S. 2018. Lifelong Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization. *arXiv preprint* .

Perera, P.; Nallapati, R.; and Xiang, B. 2019. OCGAN: One-class Novelty Detection Using GANs with Constrained Latent Representations. In *CVPR*.

Qin, Q.; Hu, W.; and Liu, B. 2020. Using the Past Knowledge to Improve Sentiment Classification. In *EMNLP-findings*.

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.; and Lampert, C. H. 2017. iCaRL: Incremental classifier and representation learning. In *CVPR*, 5533–5542.

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*.

Ruff, L.; Vandermeulen, R.; Goernitz, N.; Deecke, L.; Siddiqui, S. A.; Binder, A.; Müller, E.; and Kloft, M. 2018. Deep One-Class Classification. In *ICML*.

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*.

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* .

Schölkopf, B.; Platt, J. C.; Shawe-Taylor, J. C.; Smola, A. J.; and Williamson, R. C. 2001. Estimating the support of a high-dimensional distribution. In *Neural Computation*.

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*.

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.

Tax, D. M.; and Duin, R. P. 2004. Support vector data description. *Machine Learning* 54(1): 45–66.

van de Ven, G. M.; and Tolias, A. S. 2019. Three scenarios for continual learning. *https://arxiv.org/pdf/1904.07734.pdf* .

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.

Yoon, J.; Yang, E.; Lee, J.; and Hwang, S. J. 2018. Lifelong Learning with Dynamically Expandable Networks. In *ICLR*.

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*.

A Appendix

A.1 Ablation Study and Analysis

We use MNIST and CIFAR10 (pre-trained features are used for CIFAR10) as representatives for ablation study and show the deep changes of models after adding different components. Table 6 shows the results of adding different components of PCL. We can see that all of them are useful.

(1) *H-reg* is very effective. It helps improve the accuracy of PCL drastically, i.e., by 78.23 points for MNIST and 73.70 points for CIFAR10. As analyzed in Sec. 3.1, we now also experimentally show that H-reg can reduce the parameters' norm $\|\theta\|_2$. Moreover, after adding H-reg, the relative difference of $\mathbb{E}_{\text{neg}}$ and $\mathbb{E}_{\text{pos}}$ is much larger, which enables better classification. That is, there is less overlapping between the outputs of negative heads (the heads excluding the correct labels) and positive heads (the heads corresponded to the correct labels) compared with only using NLL. That is because H-reg enables the model to focus on the input features holistically. We can see that the change rate of the output of the correct head corresponding to the input $\mathbf{x}$, denoted by $\frac{\text{Std}}{\mathbb{E}_{\text{pos}}}$ in Table 6 is greatly reduced from 0.398 to 0.177, which shows that the output is more stable with the regularization.

(2) As discussed in Sec. 3.1, the normalization of $\mathbf{x}$ enables the models of different classes converge to a more stable and similar points. With $\mathbf{x}$-normalization, we see the change rate $\frac{\text{Std}}{\mathbb{E}_{\text{pos}}}$ is further reduced significantly, from 0.177 to 0.063, which gives more stable output and also make the outputs of different heads more similar. Note that normalization significantly decreases the output value for MNIST, $\mathbb{E}_{\text{pos}}$ from 4.23 to 0.507 and for CIFAR10 from 5.54 to 0.446, which are expected.

(3) After adding the *Discriminative Information Extraction* (DIE) mechanism, the margins between the outputs of $\mathbf{x}$-negative heads (see $\mathbb{E}_{\text{neg}}$) and $\mathbf{x}$-positive heads (see $\mathbb{E}_{\text{pos}}$) are enlarged, making the decision surface clearer for classification.

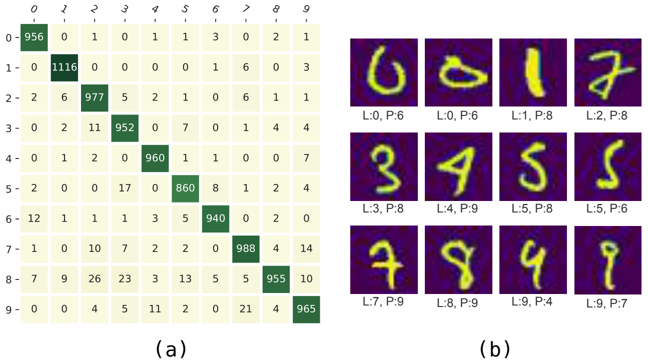

Figure 2: Error analysis on MNIST. The numbers in the heatmap (a) of position (i, j) denotes the number of instances of class j (ground-truth) being classified as class i (prediction). L in (b) indicates the ground-truth class label and P the predicted label.

A.2 Error Analysis

Fig. 2 (a) shows the classification distribution of different classes for MNIST. We can see that most instances are classified correctly. However, digit '8' is tricky as some instances of digits 2, 3, 5 can be classified as '8'. Some '1"s can also be recognized as '8' because they are very thick/wide and thus get high responses for '8'. Note that symmetry doesn't hold because '8' is not easily recognized as other digits due to the fact that '8' has specific features that other digits don't have. Apart from '8', we find that '0' is more easily classified as '6'; '9' is more easily classified as '4' or '7'. From the error instances shown in Fig. 2 (b), we can see their writings look similar.

Table 6: Ablation experiments of different components of PCL using MNIST and CIFAR10. H-reg: H-reg, $\mathbf{x}$-N: normalization of $\mathbf{x}$ and DIE: discriminative information extraction. With y as the ground-truth label for input $\mathbf{x}$, $\mathbb{E}_{\text{neg}} = \mathbb{E}[\sum_{C_i \neq y} \frac{f_{C_i}(\mathbf{x})}{N-1}]$ denotes the expectation of the output of heads excluding the correct head corresponding to the input $\mathbf{x}$ (we call these heads $\mathbf{x}$-negative heads). $\mathbb{E}_{\text{neg}} = \mathbb{E}[f_y(\mathbf{x})]$ denotes the expectation of the output of the correct head corresponding to the input $\mathbf{x}$ (we call these heads $\mathbf{x}$-positive heads) and Std denotes its standard-deviation. $\frac{\text{Std}}{\mathbb{E}_{\text{pos}}}$ is the change rate of the output of $\mathbf{x}$-positive heads. $\|\theta\|_2$ denotes the average of the norm of the network parameters. ACCs (accuracy) are given in percentage.

Components	***MNIST***					***CIFAR10***				
	ACC	$\mathbb{E}_{\text{neg}}$	$\mathbb{E}_{\text{pos}} \pm$ Std	$\frac{\text{Std}}{\mathbb{E}_{\text{pos}}}$	$\|\theta\|_2$	ACC	$\mathbb{E}_{\text{neg}}$	$\mathbb{E}_{\text{pos}} \pm$ Std	$\frac{\text{Std}}{\mathbb{E}_{\text{neg}}}$	$\|\theta\|_2$
NLL	11.35	31.9	32.2 ± 12.8	0.398	6.81	9.10	45.7	45.7 ± 9.6	0.210	6.43
NLL + H-reg	89.58	3.59	4.23 ± 0.75	0.177	3.80	82.80	4.56	5.54 ± 0.61	0.110	3.81
NLL + H-reg + $\mathbf{x}$*-N*	96.31	0.401	0.507 ± 0.032	0.063	4.90	84.11	0.348	0.446 ± 0.030	0.067	4.32
NLL + H-reg + $\mathbf{x}$*-N* + *DIE* (PCL)	97.00	0.383	0.495 ± 0.030	0.061	4.60	84.93	0.327	0.438 ± 0.029	0.066	4.35