Class Incremental Learning on Imbalanced Data with Example Influence

Lin Chen, Chenxing Hong, Hu Ding, Liyuan Wang, Youliang Chu

23020221154072, 36920221153082^{AI}, 23020221154081, 22020221154118, 23020221154078
Xiamen University, Xiamen 361005, China

Abstract

Modern machine learning suffers from catastrophic forgetting when learning new classes incrementally. The performance dramatically degrades due to the missing data of old classes. Incremental learning methods have been proposed to retain the knowledge acquired from the old classes, by using knowledge distilling and keeping a few exemplars from the old classes. However, these methods struggle in real world data.We believe this is because of the combination of two factors: (a) the data imbalance between the old and new classes, and (b) the increasing number of visually similar classes. Distinguishing between an increasing number of visually similar classes is particularly challenging, when the training data is unbalanced. Inspired by Influence Function (IF), we first study example influence via adding perturbation to example weight and computing the influence derivation.Moreover,We found that the last fully connected layer has a strong bias towards the new classes, and this bias can be corrected by a linear model,finally, we used meta-learning and the influence function (IF) to select significant samples for incremental learning. Empirical results show that our algorithm significantly outperforms state-of-the-art methods on class- incremental benchmark CL datasets.

Introduction

Natural learning systems are inherently incremental where new knowledge is continuously learned over time while existing knowledge is maintained (Rebuffi et al. 2017; Li and Hoiem 2017). Many computer vision applications in the real world require incremental learning capabilities. For example, a face recognition system should be able to add new persons without forgetting the faces already learned. However, most deep learning approaches suffer from *catastrophic forgetting* (McCloskey and Cohen 1989) — a significant performance degradation, when the past data are not available.

The missing data for old classes introduce two challenges — (a) maintaining the classification performance on old classes, and (b) balancing between old classes and new classes. Distillation (Li and Hoiem 2017; Rebuffi et al. 2017; Castro et al. 2018) has been used to effectively address the former challenge. Recent studies (Rebuffi et al. 2017; Castro et al. 2018) also show that selecting a few exemplars from the old classes can alleviate the imbalance problem. These methods perform well on small datasets. However, they suffer from a significant performance degradation when the number of classes becomes large (e.g. thousands of classes).

Why is it more challenging to handle a large number of classes for incremental learning? We believe this is due to the coupling of two factors. First, the training data are unbalanced. Secondly, as the number of classes increases, it is more likely to have visually similar classes (e.g. multiple dog classes in ImageNet) across different incremental steps. Under the incremental constraint with data imbalance, the increasing number of visually similar classes is particularly challenging since the small margin around the boundary between classes is too sensitive to the data imbalance. The boundary is pushed to favor classes with more samples.

The sequential paradigm in CL means CL does not access past training data. Comparing to traditional machine learning, the training data in CL is thus more precious. It is valuable to explore the influence difference among training examples. Following the accredited influence chain "Data-Model-Performance", exploring this difference is equivalent to tracing from performance back to example difference. With appropriate control, this may improve the learning pattern towards better performance.

In this work, we present a method to address the data imbalance problem in large scale incremental learning. Firstly, we found a strong bias towards the new classes in the classifier layer (i.e. the last fully connected layer) of the convolution neural network (CNN).Based upon this finding, we propose a simple and effective method, called BiC (bias correction), to correct the bias. We add a bias correction layer after the last fully connected (FC) layer (shown in Fig. 1), which is a simple linear model with two parameters. The bias correction layer is learned at the second stage, after learning the convolution layers and FC layer at the first stage. The data, including exemplars from the old classes and samples from the new classes, are split into a training set for the first stage and a validation set for the second stage. The validation set is helpful to approximate the real distribution of both old and new classes in the feature space, allowing us to estimate the bias in FC layer. We found that the bias can be effectively corrected with a small validation set.On the other hand,the example influence can be directly used to control the magnitude of training loss for each example. Also, we utilize a meta-learning approach to help incremental learning and select those samples that are important to store by means of an influence function (IF).

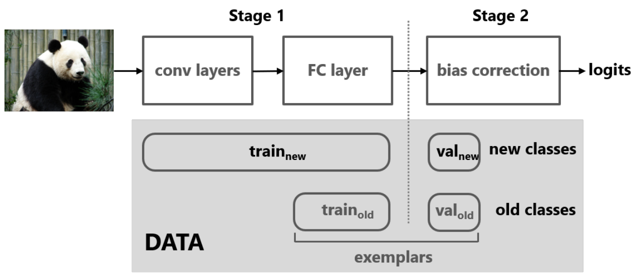


Figure 1: Overview of our method. The exemplars from the old classes and the samples of the new classes are split into training and validation sets. The training set is used to train the convolution layers and FC layer (in stage 1). The validation set is used for bias correction (in stage 2).

Our method achieves remarkably good performance, especially on large scale datasets. The experimental results show that our method outperforms state-of-the-art algorithms.

Related Work

Incremental learning has been a long standing problem in machine learning (Cauwenberghs and Poggio 2000; Mensink et al. 2013; Kuzborskij, Orabona, and Caputo 2013). Before the deep learning took off, people had been developing incremental learning techniques by leveraging linear classifiers, ensemble of weak classifiers, nearest neighbor classifiers, etc. Recently, thanks to the exciting progress in deep learning, there has been a lot of research on incremental learning with deep neural network models. The work can be roughly divided into three categories depending on whether they require real data or synthetic data or nothing from the old classes.

Without using old data: Methods in the first category do not require any old data. (Jung et al. 2016) presented a method for domain transfer learning. They try to maintain the performance on old tasks by freezing the final layer and discouraging the change of shared weights in feature extraction layers. (Kirkpatrick et al. 2017) proposed a technique to remember old tasks by constraining the important weights when optimizing a new task. One limitation of this approach is that the old and new tasks may conflict on these important weights. (Li and Hoiem 2017) presented a method that applies knowledge distillation (Hinton et al. 2015) to maintain the performance on old tasks. (Li and Hoiem 2017) separated the old and new tasks in multi-task learning, which is different from learning classifier incrementally. (Shmelkov, Schmid, and Alahari 2017) applied knowledge distillation for learning object detectors incrementally. (Rannen et al. 2017) utilized autoencoder to retain the knowledge from old tasks. (Sun et al. 2018; Sun, Cong, and Xu 2018) updated knowledge dictionary for new tasks and kept dictionary coefficients for old tasks.

Using synthetic data: Both (Shin et al. 2017) and (Venkatesan et al. 2017) employed GAN (Goodfellow et al. 2020) to replay synthetic data for old tasks. (Shin et al. 2017) applied cross entropy loss on synthesis data with the old solver's response as the target. (Venkatesan et al. 2017) utilized a root mean-squared error for learning the response of old tasks on synthetic data. (Shin et al. 2017; Venkatesan et al. 2017) highly depends on the capability of generative models and struggles with complex objects and scenes.

Using exemplars from old data: Methods in the third category require part of the old data. (Rebuffi et al. 2017) proposed a method to select a small number of exemplars from each old class. (Castro et al. 2018) keeps classifiers for all incremental steps and used them as distillation. It introduces balanced fine-tuning and temporary distillation to alleviate the imbalance between the old and new classes. (Lopez-Paz and Ranzato 2017) proposed a continuous learning framework where the training samples for different tasks are used one by one during training. It constrains the cross entropy loss on softmax outputs of old tasks when the new task comes. (Xiao et al. 2014) proposed a training method that grows a network hierarchically as new training data are added. Similarly, (Rusu et al. 2016) increases the number of layers in the network to handle new coming data.

Example Influence: In recent years, as the impressive Interpretable Machine Learning (IML) develops, people realize the importance of exploring the nature of data-driven machine learning. Examples are different, even they belong to the same distribution. Because of such difference, the example contributes differently to the learning pattern. In other words, the influence acquired in advance from different training examples can significantly improve the CL training. In contrast to complicated model design, a model-agnostic algorithm estimates the training example influence via computing the derivation from a test loss to a training data weight. One typical example method is the Influence Function , which leverages a pure second-order derivation (Hessian) with the chain rule. In this paper, to avoid the expensive computation of Hessian inverse, we design a meta learning based method, which can be used to control the training.

Methodology

In this section, we describe our model in detail. We first introduce a Bias Correction Layer(Bic) to deal with the imbanced problem in the validation set and new data. Then, we present our Example Influence on Stability and Plasticity in incremental learning,Finally,we discuss the Meta Learning on Stability and Plasticity and come up with an indicator to select the important samples are good for the incremental learning.

Rehearsal-based CL

Given T different tasks w.r.t. datasets $\{D_1, \cdots, D_T\}$, Continual Learning (CL) seeks to learn them in sequence. For the t-th dataset (task), $D_t = \{(x_t^{(n)}, y_t^{(n)})\}_{n=1}^{N_t}$ is split into a training set D_t^{trn} and a test set D_t^{tst}, where N_t is the number of examples. At any time, CL aims at learning a multi-task/multi-class predictor to predict tasks/classes that have been learned (say task-incremental and class-incremental CL). To suppress the catastrophic forgetting, the rehearsal-based CL builds a small size memory buffer M_t sampled from D_t^{trn} for each task (, $|M_t| \ll |D_t^{\text{trn}}|$). At training phase,

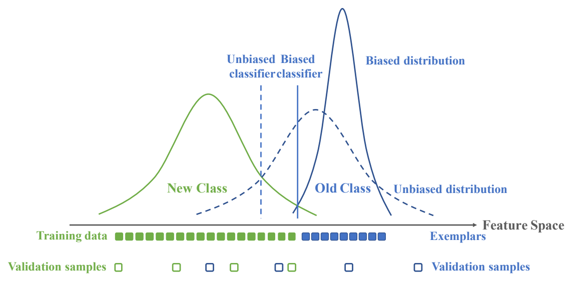


Figure 2: Diagram of bias correction. Since the number of exemplars from old classes is small, they have narrow distributions on the feature space. This causes the learned classifier to prefer new classes. Validation samples, not involved in training feature representation, may better reflect the unbiased distribution of both old and new classes in the feature space. Thus, we can use the validation samples to correct the bias. (Best viewed in color)

the data in the whole memory $M = \cup_{k<t} M_k$ will be retrained together with the current tasks. Accordingly, a mini-batch training step of task t in rehearsal-based CL is denoted as

$$\min_{\theta_t} \quad \ell(B_{\text{old}} \cup B_{\text{new}}, \theta_t), \tag{1}$$

where ℓ is the empirical loss. θ_t is the trainable parameters at task t and is updated from scratch.

Validation Set

We estimate the bias by using a small validation set. The basic idea is to exclude the validation set from training the feature representation, allowing them to reflect the unbiased distribution of both old and new classes on the feature space (shown in Fig. 2). Therefore, we split the exemplars from the old classes and the samples from the new classes into a training set and a validation set. The training set is used to learn the convolution and fully connected layers (see Fig. 1), while the validation set is used for the bias correction.

Fig. 1 illustrates the generation of the validation set. The stored exemplars from the old classes are split into a training subset (referred to $train_{old}$) and a validation subset (referred to val_{old}). The samples for the new classes are also split into a training subset (referred to $train_{new}$) and a validation subset (referred to val_{new}). $train_{old}$ and $train_{new}$ are used to learn the convolution and FC layers (see Fig. 1). val_{old} and val_{new} are used to estimate the parameters in the bias correction layer. Note that val_{old} and val_{new} are balanced.

Bias Correction Layer

The bias correction layer should be simple with a small number of parameters, since val_{old} and val_{new} have small size. Thus, we use a linear model (with two parameters) to correct the bias. This is achieved by adding a bias correction layer in the network (shown in Fig. 1). We keep the output logits for the old classes $(1, \dots, n)$ and apply a linear model to correct the bias on the output logits for the new classes $(n+1, \dots, n+m)$ as follows:

$$q_k = \begin{cases} o_k & 1 \le k \le n \\ \alpha o_k + \beta & n+1 \le k \le n+m \end{cases}, \tag{2}$$

where α and β are the bias parameters on the new classes and o_k is the output logits for the k-th class. Note that the bias parameters (α, β) are shared by all new classes, allowing us to estimate them with a small validation set. When optimizing the bias parameters, the convolution and fully connected layers are frozen. The classification loss (softmax with cross entropy) is used to optimize the bias parameters as follows:

$$L_b = -\sum_{k=1}^{n+m} \delta_{y=k} \log[softmax(q_k)]. \tag{3}$$

We found that this simple linear model is effective to correct the bias introduced in the fully connected layer.

Example Influence on Stability and Plasticity

Stability and Plasticity

Suppose the parameter of a model is initialized to θ_0. At the training on the t-th task, given test sets of an old task $D_k^{\text{tst}} (k < t)$ and the current task D_t^{tst}, the Stability S_t^k and Plasticity P_t can be evaluated by:

$$\begin{aligned} S_t^k &= p(D_k^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}}) - p(D_k^{\text{tst}} | \theta_k), \\ P_t &= p(D_t^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}}) - p(D_t^{\text{tst}} | \theta_{t-1}), \end{aligned}$$

where $p(D_1 | \theta, D_2)$ represents the performance (accuracy in classification) of D_1 conditioned to the model θ training on D_2. $p(D|\theta)$ denotes the performance of D tested on the model θ.

The S of a task is evaluated by the performance difference on the test set after training on any later tasks, which is also known as Forgetting. The P of a task is defined as the ability to integrate new knowledge, which is regarded as the test performance of this task. As many existing CL methods demonstrate, the SP inevitably interferes mutually.

Example Influence on SP

At the training on the t-th task, with a sampled example $x^{\text{trn}} \in D_t^{\text{trn}}$, the example influence from x^{trn} to Stability S_t^k and Plasticity P_t for $k < t$ can be evaluated by the gap from deleting it then retraining the model:

$$\begin{aligned} &I_S(D_k^{\text{tst}}, x^{\text{trn}}) \\ &= p(D_k^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}}) - p(D_k^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}} / x^{\text{trn}}), \\ &I_P(D_t^{\text{tst}}, x^{\text{trn}}) \\ &= p(D_t^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}}) - p(D_t^{\text{tst}} | \theta_{t-1}, D_t^{\text{trn}} / x^{\text{trn}}), \end{aligned}$$

where $D_t^{\text{trn}} / x^{\text{trn}}$ denotes the dataset D_t^{trn} without the training example x^{trn}.

However, deleting every example to compute full influences is impractical due to the highly computational cost. Instead, the performance change can be indicated by the loss change, which leads to a derivable way to approximate the influence:

$$\begin{aligned} I_S(D_k^{\text{tst}}, x^{\text{trn}}) &\overset{\text{def}}{=} \frac{\partial \ell(D_k^{\text{tst}})}{\partial \epsilon}, \\ I_P(D_t^{\text{tst}}, x^{\text{trn}}) &\overset{\text{def}}{=} \frac{\partial \ell(D_t^{\text{tst}})}{\partial \epsilon}, \end{aligned} \tag{4}$$

where ϵ is the weight perturbation to the training example and $\overset{\text{def}}{=}$ means define. This influence can be computed by the Influence Function that will be introduced in the next section.

Meta Learning on Stability and Plasticity

Influence Function for SP

A mini-batch, B, from the training data is sampled, and the normal model update is

$$\hat{\theta} = \arg\min_{\theta} \ell\left(B, \theta\right). \tag{5}$$

In Influence Function (IF), a small weight perturbation ϵ is added to the training example $x^{\text{trn}} \in B$

$$\hat{\theta}_{\epsilon,x} = \arg\min_{\theta} \ell\left(B, \theta\right) + \epsilon\ell(x^{\text{trn}}, \theta), \quad x^{\text{trn}} \in B. \tag{6}$$

We can easily promote this to the mini-batch

$$\hat{\theta}_{E,B} = \arg\min_{\theta} \ell\left(B, \theta\right) + E^{\top} L(B, \theta), \tag{7}$$

where L denotes the loss vector for a mini-batch and $E \in R^{|B|\times 1}$ denotes the perturbation on each example in it. It is easy to know that the example influence $I(D^{\text{tst}}, B)$ is reflected in the derivative $\nabla_E \ell(D^{\text{tst}}, \hat{\theta}_{E,x})\big|_{E=0}$. By the chain rule, the example influence in IF can be computed by

$$\begin{aligned} I(D^{\text{tst}}, B) &\overset{\text{def}}{=} \nabla_E \ell(D^{\text{tst}}, \hat{\theta}_{E,x})\big|_{E=0} \\ &= -\nabla_{\theta}\ell(D^{\text{tst}}, \hat{\theta}) H^{-1} \nabla_{\theta}^{\top} L(B, \hat{\theta}), \end{aligned} \tag{8}$$

where$H = \nabla_{\theta}^{2}\ell(B, \hat{\theta})$ is a Hessian. Unfortunately, the inverse of Hessian requires the complexity $O(|B|q^2 + q^3)$ and huge storage for neural networks (maybe out-of-memory), which is challenging for efficient training.

In Eq. (8), we have $I(D^{\text{tst}}, B) = [I(D^{\text{tst}}, x^{\text{trn}}) | x^{\text{trn}} \in B]$ and find the loss will get larger if $I(D^{\text{tst}}, x^{\text{trn}}) > 0$, which means the negative influence on the test set D^{tst}. Similarly, $I(D^{\text{tst}}, x^{\text{trn}}) < 0$ means the positive influence on the test set D^{tst}. *Fortunately, the second-order derivation in IF is not necessary under the popular meta learning paradigm, instead we can easily get the derivation like IF through a one-step pseudo update.* In the following, we will introduce a simple yet effective meta-based method, named MetaSP, to simulate IF at each step with a two-level optimization to avoid computing Hessian inverse.

Simulating IF for SP

Based on the meta learning paradigm, we transform the example influence computation into solving a meta gradient descent problem, named **MetaSP**. For each training step in a rehearsal-based CL, we have two mini-batches data B_{old} and B_{new} in respect to old and new tasks. Our goal is to obtain the influence on S and P from every example in $B_{\text{old}} \cup B_{\text{new}}$. Note that both S-aware and P-aware influence are applied to every example regardless of old or new tasks. That is, the contribution of an example is not deterministic. Data of old tasks may also affect the new task in positive, and vice-versa. In rehearsal-based CL, we turn to computing the derivations $\nabla_E \ell(V_{\text{old}}, \hat{\theta})|_{E=0}$ for example influence. To compute the derivation, as shown in Fig.3(a), our MetaSP has two key steps:

Algorithm 1: Computation of Example Influence (**MetaSP**)

```
Input: $B_{\text{old}}, B_{\text{new}}, V_{\text{old}}, V_{\text{new}}$ ;        // Training
      batches, Validation batches
Output: $I^*$ ;   // Pareto example influence
      on SP
1 $\hat{\theta}_{E,B} = \arg\min_{\theta}\ \ell(B_{\text{old}} \cup B_{\text{new}}, \theta) + E^{\top} L(B_{\text{old}} \cup B_{\text{new}}, \theta) + Bic(V_{\text{old}} \cup V_{\text{new}}, \theta)$ ;      // Pseudo
   update
2 $I(V_{\text{old}}, B) = \nabla_E \ell(V_{\text{old}}, \hat{\theta}_{E,B})$ ;      // Gradient
   from old val loss
3 $I(V_{\text{new}}, B) = \nabla_E \ell(V_{\text{new}}, \hat{\theta}_{E,B})$;      // Gradient
   from new val loss
4 $\gamma^* \leftarrow$ Eq. (13);              // Optimal fusion
   hyper-parameter
5 $I^* = \gamma^* \cdot I(V_{\text{old}}, B) + (1 - \gamma^*) \cdot I(V_{\text{new}}, B)$;
   // Influence fusion
```

(1) Pseudo update. This step is to simulate Eq. (7) in IF via a pseudo update

$$\begin{aligned} \hat{\theta}_{E,B} = \arg\min_{\theta} \quad & \ell(B_{\text{old}} \cup B_{\text{new}}, \theta) \\ & + E^{\top} L(B_{\text{old}} \cup B_{\text{new}}, \theta), \end{aligned} \tag{9}$$

where L denotes the loss vector for a mini-batch combining both old and new tasks.

(2) Compute example influence. This step computes example influence on S and P for all training examples as simulating Eq. (8). Based on the pseudo updated model in Eq. (9), we compute S- and P-aware example influence via two validation sets V_{old} and V_{new}. Noteworthily, because the test set D^{tst} is unavailable at training phase, we use two dynamic validation sets V_{old} and V_{new} to act as the alternative in the CL training process. One is sampled from the memory buffer (V_{old}) representing the old tasks, and the other is from the seen training data representing the new task (V_{new}). With E initialized to 0, the two kinds of example influence are computed as

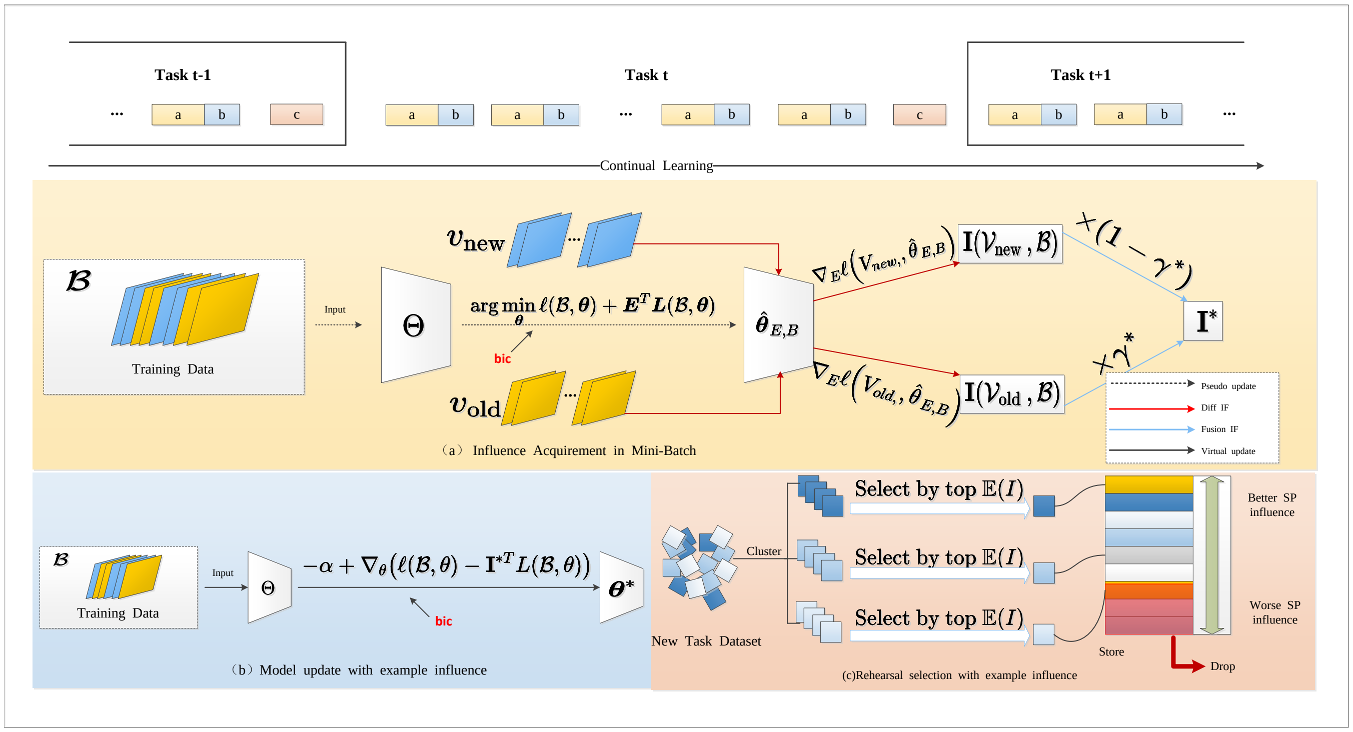
$$\begin{aligned} I(V_{\text{old}}, B) &= \nabla_E \ell(V_{\text{old}}, \hat{\theta}_{E,B}), \\ I(V_{\text{new}}, B) &= \nabla_E \ell(V_{\text{new}}, \hat{\theta}_{E,B}). \end{aligned} \tag{10}$$

Generally, each elements in two influence vectors $I(V_{\text{old}}, B)$ and $I(V_{\text{new}}, B)$ represents the example influence on S and P. Similar to IF, elements with positive value mean negative influence while elements with negative value mean positive influence.

Using Influence for Continual Learning

Before Using: Influence for SP Pareto Optimality

As shown in Eq. (10), the example influence is equal to the derivation from validation loss of old and new tasks to the perturbations E. However, the two kinds of influence



Task t-1
Task t
Task t+1
Continual Learning
$\mathcal{B}$
Training Data
Input
Θ
$\arg\min_{\theta} \ell(\mathcal{B},\theta) + E^T L(\mathcal{B},\theta)$
bic
$\mathcal{V}_{\text{new}}$
$\mathcal{V}_{\text{old}}$
$\hat{\theta}_{E,B}$
$\nabla_E \ell(\mathcal{V}_{new}, \hat{\theta}_{E,B})$
$\nabla_E \ell(\mathcal{V}_{old}, \hat{\theta}_{E,B})$
$\mathbf{I}(\mathcal{V}_{\text{new}}, \mathcal{B})$
$\mathbf{I}(\mathcal{V}_{\text{old}}, \mathcal{B})$
$\times(1-\gamma^*)$
$\times\gamma^*$
$\mathbf{I}^*$
Pseudo update
Diff IF
Fusion IF
Virtual update
（a） Influence Acquirement in Mini-Batch
$-\alpha + \nabla_\theta(\ell(\mathcal{B},\theta) - \mathbf{I}^{*T} L(\mathcal{B},\theta))$
θ^*
（b） Model update with example influence
Cluster
New Task Dataset
Select by top $\mathbb{E}(I)$
Better SP influence
Worse SP influence
Store
Drop
(c)Rehearsal selection with example influence

Figure 3: Evaluating and making use of example influence in mini-batch Continual Learning. (a) At each iteration in CL training, MetaSP updates in pseudo and use two validation sets representing old tasks and new task to obtain the example influence on S and P. The two kinds of influence are fused towards a Pareto optimal. (b) The computed influence can be directly used to update CL model and (c) select examples for rehearsal storing and dropping.

are independent and interfere with each other. That is, using only one of them may fail the other performance. We prefer to find a solution that makes a trade-off between the influence on both S and P. Thus, we integrate the two influence $I(V_{\text{old}}, B)$ and $I(V_{\text{new}}, B)$ into a DOO problem with two gradients from different objectives.

$$\min_{E} \quad \left\{ \ell(V_{\text{old}}, \hat{\theta}_{E,B}), \ell(V_{\text{new}}, \hat{\theta}_{E,B}) \right\}. \tag{11}$$

The goal of Problem (11) is to obtain a fused way that satisfies the SP Pareto optimality.

SP Pareto Optimality

Pareto Dominate Let E_a, E_b be two solutions for Problem (11), E_a is said to dominate E_b ($E_a \prec E_b$) if and only if $\ell(V, \hat{\theta}_{E_a,B}) \leq \ell(V, \hat{\theta}_{E_b,B}), \forall V \in \{V_{\text{old}}, V_{\text{new}}\}$, and $\ell(V, \hat{\theta}_{E_a,B}) < \ell(V, \hat{\theta}_{E_b,B}), \exists V \in \{V_{\text{old}}, V_{\text{new}}\}$.
SP Pareto Optimal E is called SP Pareto optimal if no other solution can have better values in $\ell(V_{\text{old}}, \hat{\theta}_{E,B})$ and $\ell(V_{\text{new}}, \hat{\theta}_{E,B})$.

Inspired by the Multiple-Gradient Descent Algorithm (MGDA), we transform Problem (11) to a min-norm problem. Specifically, according to the KKT conditions, we have

$$\begin{aligned} \gamma^* = \arg\min_{\gamma} \big\| \gamma \nabla_E \ell(V_{\text{old}}, \hat{\theta}_{E,B}) + \\ (1-\gamma) \nabla_E \ell(V_{\text{new}}, \hat{\theta}_{E,B}) \big\|_2^2, \\ s.t., 0 \leq \gamma \leq 1. \end{aligned} \tag{12}$$

Referring to the study from Sener, the optimal γ^* is easily computed as

$$\gamma^* = \min \left(\max \left(\frac{(\nabla_E \ell(V_{\text{new}}, \hat{\theta}_{E,B})}{\|\nabla_E \ell(V_{\text{new}}, \hat{\theta}_{E,B})\|_2^2}, 0 \right), 1 \right). \tag{13}$$

Thus, the SP Pareto influence of the training batch can be computed by

$$I^* = \gamma^* \cdot I(V_{\text{old}}, B) + (1-\gamma^*) \cdot I(V_{\text{new}}, B). \tag{14}$$

This process can be seen in Fig.3(a). Different from the S-aware and P-aware influence, the integrated influence consider the Pareto optimum to both S and P, , reducing the negative influence on S or P and keeping the positive influence on both S and P. Then we will introduce how to leverage example influence in CL training, our algorithm can be seen in Alg. 1.

Model Update Using Example Influence

With the computed example influence in each mini-batch, we can easily control the model update of this mini-batch to

Algorithm 2: Using Example Influence in Rehearsal-based Continual Learning.

```
Input: Initialized $\theta_0$, Learning rate $\alpha$, Training set
       $\{D_1^{\text{trn}}, \cdots, D_T^{\text{trn}}\}$, Memory $M$
Output: $\theta_T$ ;                          // Final model
1  for task $t = 1 : T$ do
2      2 $\theta_t$= TrainNewTask($\theta_{t-1}$, $D_t^{\text{trn}}$, $M$) (Alg. 3)
3      $C_1, C_2, \cdots, C_{\frac{|M|}{t}} \leftarrow$ K-Means($D_t^{\text{trn}}$);
4      Rank $C_i$ with $E(I^*(x)), x \in C_i$;
5      Rank $M$ with $E(I^*(x)), x \in M$;
6      for $i = 1 : \frac{|M|}{t}$ do
7          Pop the bottom of $M$;
8          Push the top of $C_i$ to $M$;
9      end
10 end
```

adjust the training towards an ensemble positive direction. Given parameter θ from the previous iteration the step size α, the model can be updated in traditional SGD as $\theta^* = \theta - \alpha \cdot \nabla_\theta \left(\ell(B, \theta)\right)$, where $B = B_{\text{old}} \cup B_{\text{new}}$. By regularizing the update with the example influence I^* , we have

$$\theta^* = \theta - \alpha \cdot \nabla_\theta \left(\ell(B, \theta) + (-I^*)^\top L(B, \theta)\right). \qquad (15)$$

MetaSP offers regularized updates at every step for rehearsal-based CL, which leads the CL training to better SP but with only the complexity of $O(|B|q + vq)$ (v denotes the validation size) compared with that of IF, $O(|B|q^2 + q^3)$.

We show this application in Fig.3(b). By updating like the above equation, we can make use of the influence of each example to a large extent. In this way, some useless examples are restrained and some positive examples are emphasized, which may improve the acquisition of new knowledge and the maintenance of old knowledge simultaneously.

Algorithm 3: Training New Task

```
Input: Initialized $\theta_t$, Training set $D_t^{\text{trn}}$, Memory $M$,
       Learning rate $\alpha$
Output: Trained $\theta_t$
1  for $i = 1$ :ITER_NUM do
2      $B_{\text{new}} \sim D_t^{\text{trn}}$;
3      if $t = 1$ then
4          $\theta_t = \theta_t - \alpha \cdot \nabla_\theta \ell(B_{\text{new}}, \theta_t)$;
5      else
6          $B_{\text{old}} \sim M$, $V_{\text{old}} \sim M$, $V_{\text{new}} \sim D_t^{\text{trn}}$;
7          $I^* \leftarrow$ METASP($B_{\text{old}}$, $B_{\text{new}}$, $V_{\text{old}}$, $V_{\text{new}}$);
8          $\theta_t = \theta_t - \alpha \cdot \nabla_\theta(\ell(B_{\text{old}} \cup B_{\text{new}}, \theta_t)$
9                $+(-I^*)^\top L(B_{\text{old}} \cup B_{\text{new}}, \theta_t))$;
10     end
11 end
```

Rehearsal Selection Using Example Influence

Rehearsal in fixed budget needs to consider *storing* and *dropping* to keep the memory M having the core set of all old tasks. In tradition, storing and dropping are both based on randomly example selection, which ignores the influence difference on SP from each example. Given influence $I^*(x)$ representing contributions from example x to SP, we further design to use it to improve the rehearsal strategy under fixed memory budget. The above example influence on S and P is computed in mini-batch level, we can promote it to the whole dataset according to the law of large numbers, and the influence value for the example x is the value of expectation over batches, , $E(I^*(x))$.

The fixed-size memory is divided averagely by the seen task number. After task t finishes its training, we conduct our influence-aware rehearsal selection strategy as shown in Fig.3(c). For storing, we first cluster all training data into $\frac{|M|}{t}$ groups using K-means to diversify the store data. Each group is ranked by its SP influence value, and the most positive influence on both SP will be selected to store. For dropping, we rank again on the memory buffer via their influence value, and drop the most negative $\frac{|M|}{t}$ example. In this way, M always stores diverse examples with positive SP influence.

Experiments

In this section we first introduce the experimental setup and then we evaluate our method .

Experimental Setup

We perform experiments on two datasets. cifar-100 and ImageNet-Subset with 100 classes. we use a public implementation of existing CIL methods from the framework FaCIL [27] and implement our BicIFCIL methods in the same framework for a fair comparison. We follow the approach of LUCIR and PODNET by starting with a large first task with half of the classes in each dataset and equally distributing the remaining classes in subsequent tasks. We use ResNet-32 for CIFAR-100 and ResNet-18 for ImageNet-Subset. we use an initial learning rate of 0.1 and divide it by 10 after 80 and 120 calendar hours (160 calendar hours in total) for CIFAR-100. for ImageNet-Subset, the learning rate starts at 0.1 started and divided by 10 after 30 and 60 calendar hours (90 calendar hours in total). The batch size for all experiments was 128. for the second phase of training, the learning rate was set to 0.1 and we trained for 30 calendar hours. For the datasets used in the experiments, we first performed an imbalance cut with an imbalance factor of 0.01.We arranged these data in a shuffled and ordered way in incremental learning.

We used the average precision of all categories and the average incremental precision of all tasks as evaluation metrics. We first evaluated different methods in the imbalanced-CIL scenario with an imbalance rate rho = 0.01 and 20 examples per category.

Results for our method

Our method for Imbalanced-CIL. In Table 2we integrate our proposed two-stage strategy into three existing methods: EEIL, LUCIR (with CNN classifier), and PODNET. In general, the two-stage strategy helps on all three methods in both 5- and 10-task settings. The improvement is especially noticeable in the Shuffled imbalanced-CIL scenario. Specifically, for EEIL, our method only improves by a small

	Methods	CIFAR-100		ImageNet-Subset	
		5 tasks	*10 tasks*	*5 tasks*	*10 tasks*
Ordered	EEIL	38.46	37.50	50.68	50.63
	+ (Ours)	38.97+0.51	37.58+0.08	51.36+0.68	50.74+0.11
	LUCIR	42.69	42.15	52.91	52.80
	+ (Ours)	**45.88**+3.19	**45.73**+3.58	54.22+1.31	55.41+2.61
	PODNET	44.07	43.96	58.78	58.94
	+ (Ours)	44.38+0.31	44.35+0.39	**58.82**+0.04	**59.09**+0.15
Shuffled	EEIL	31.91	32.44	42.87	43.72
	+ (Ours)	34.19+2.28	33.70+1.26	49.31+6.44	48.26+4.54
	LUCIR	35.09	34.59	45.80	46.52
	+ (Ours)	**39.40**+4.31	**39.00**+4.41	**52.08**+6.28	51.91+5.39
	PODNET	34.64	34.84	49.69	51.05
	+ (Ours)	36.37+1.73	37.03+2.19	51.55+1.86	**52.60**+1.55
Conventional	EEIL	57.41	54.22	53.84	47.30
	+ (Ours)	59.10+1.69	56.91+2.69	57.45+3.61	53.40+6.10
	LUCIR	61.15	58.74	67.21	65.04
	+ (Ours)	63.48+2.33	60.57+1.83	68.82+1.61	67.44+2.40
	PODNET	63.15	61.16	70.13	65.66
	+ (Ours)	**64.58**+1.43	**62.63**+1.47	**71.08**+0.95	**68.47**+2.81

Table 1: Comparison of average incremental accuracy on CIFAR-100 and ImageNet-Subset in the imbalanced-CIL and conventional CIL scenarios.

margin on Ordered imbalanced-CIL scenario but boosts significantly on Shuffled imbalanced-CIL. It outperforms EEIL by 2.28 and 1.26 on CIFAR-100 when T = 5 and T = 10, respectively for Shuffled imbalanced-CIL. The improvement is even larger on ImageNet-Subset with 6.44 and 4.54 improvement in absolute accuracy. For LUCIR, we see a consistent boost by adding our method, improving from 1.31 to 6.28 for CIFAR-100 and ImageNet-Subset, respectively. PODNET is the best baseline in most scenarios where we observe a smaller gain with our proposed method compared to LUCIR. Overall, PODNET and LUCIR with our method can achieve very competitive results, which improves the consistency for both Ordered imbalanced-CIL and Shuffled imbalanced-CIL.

Our method for conventional CIL. Surprisingly, as seen in Table 1, when we combine ours with existing methods the performance is improved not only in imbalanced-CIL scenarios but also for conventional CIL. We believe this is due to the imbalance caused by limited memory for storing exemplars from previous tasks.

Results on real-world imbalanced dataset We experiment with 100 classes chosen from the iNaturalist dataset. We randomly chose 100 classes from the pantae super category and tested LUCIR and LUCIR+ with the data seperated into 5 tasks with a base task of 50 classes. Results show that LUCIR can achieve an accuracy of 32.34%, and LUCIR+ with two stage training about 1.46% higher. iNaturalist is a real-world dataset with imbalanced distribution, and thus the value of ρ is undefined. We estimate it to be about 0.01. It shows how our method perform in real-world dataset under imbalanced distribution.

Conclusion

In this paper we proposed two novel scenarios for class incremental learning over imbalanced distributions (Imbalanced-CIL). Ordered imbalanced considers the case where subsequent tasks contain consistently fewer samples than previous ones. Shuffled Imbalanced-CIL, on the other hand, refers to the case in which the degree of imbalance for each task is different and randomly distributed. Our experiments demonstrate that the existing state-of-the-art in CIL is significantly less robust when applied to long-tailed class distribution. To address the problem of imbalanced-CIL, we propose a meta-learning method with a learnable Bias Correction Layer scaling layer that compensates for class imbalance. Our approach significantly outperforms the state-of-the-art on CIFAR-100 and ImageNet100 with long-tailed class imbalance. Our approach is complimentary to existing methods for CIL and can be easily and profitably integrated into them. We believe that our work can serve as a test bed for future development of imbalanced class incremental learning.

References

Castro, F. M.; Marín-Jiménez, M. J.; Guil, N.; Schmid, C.; and Alahari, K. 2018. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, 233–248.

Cauwenberghs, G.; and Poggio, T. 2000. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Hinton, G.; Vinyals, O.; Dean, J.; et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Jung, H.; Ju, J.; Jung, M.; and Kim, J. 2016. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13): 3521–3526.

Kuzborskij, I.; Orabona, F.; and Caputo, B. 2013. From n to n+ 1: Multiclass transfer incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3358–3365.

Li, Z.; and Hoiem, D. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12): 2935–2947.

Lopez-Paz, D.; and Ranzato, M. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

McCloskey, M.; and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, 109–165. Elsevier.

Mensink, T.; Verbeek, J.; Perronnin, F.; and Csurka, G. 2013. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE transactions on pattern analysis and machine intelligence*, 35(11): 2624–2637.

Rannen, A.; Aljundi, R.; Blaschko, M. B.; and Tuytelaars, T. 2017. Encoder based lifelong learning. In *Proceedings of the IEEE International Conference on Computer Vision*, 1320–1328.

Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2001–2010.

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*.

Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30.

Shmelkov, K.; Schmid, C.; and Alahari, K. 2017. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, 3400–3409.

Sun, G.; Cong, Y.; and Xu, X. 2018. Active lifelong learning with" watchdog". In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Sun, G.; Yang, C.; Liu, J.; Liu, L.; Xu, X.; and Yu, H. 2018. Lifelong metric learning. *IEEE transactions on cybernetics*, 49(8): 3168–3179.

Venkatesan, R.; Venkateswara, H.; Panchanathan, S.; and Li, B. 2017. A strategy for an uncompromising incremental learner. *arXiv preprint arXiv:1705.00744*.

Xiao, T.; Zhang, J.; Yang, K.; Peng, Y.; and Zhang, Z. 2014. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, 177–186.