CE7454 Project 1: Class-Incremental Learning Challenge

Minghao LI
minghao002@e.ntu.edu.sg

Abstract

The challenge is to study Class-Incremental Learning (CIL), which allows a learning system to incorporate new concepts without forgetting old ones. In CIL, training data emerges sequentially in a stream format. At each timestamp, a new training dataset is received and the model needs to update with the new classes. The model is then tested on all seen classes to evaluate its discrimination. A good model balances depicting the characteristics of new classes while preserving the pattern of old classes. In this challenge, I employ several tricks to achieve data augmentation, regularization, and prototype normalization to enhance the extrapolation ability of the ADAMW-VPT. The data parallel technology empowers a large training batch size of up to 950 and the grid searching method to seek the optimal parameters for my CIL model. The results show that my approach outperforms the baseline by achieving 82.14% accuracy. Additionally, I fixed multiple bugs in the original repository and built a low-code framework for conducting massive CIL experiments. The corresponding code for further commitments is open-sourced at https://github.com/liminghao0914/RevisitingCIL.

1. Introduction

This section introduces the techniques and tricks employed in this report for optimization and regularization of the CIL model. Based on the preliminaries, I conduct massive experiments with various settings and parameters to seek the optimal hyperparameters and the impacts of the techniques and tricks.

1.1. Optimization

To further optimize the CIL model, I employ the data augmentation technology before the training process. Then, I leverage the convolution layer (ConvNet) of the multi-branch cosine incremental net (MBCIN) to extract the feature of the input. To mitigate the yielded differences between the fully connected (FC) layer in MBCIN and the feature extractor, I incorporate the L2-norm distance in the loss function.

Data Augmentation. To enhance the robustness of the CIL model, I add the different modes while retrieving and making the dataset. The augmentation methods are basic transformations and filters, including flipping (horizontal and vertical), rotation (per 90 degrees), and brightness adjustment. Further, I also utilize a unified operator to normalize each input image of the train set and test set.

Loss Function. I redesign the loss function by adding the L2-norm distance of the last FC layer and the ConvNet. The new loss function is shown below:

$$\mathcal{L} = \text{Loss}(\text{logits}, \text{labels}) + ||\mathbf{w} - \mathbf{x}||_2^{\beta}, \tag{1}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_n]$ represents the weight data of the FC layer and $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ represents the features extracted by ConvNet (e.g. CNN, ViT). For each vector, the n is decided by the incremental class. In this report, $n = 30$ is used as the assignment requirements. In the loss function $\mathcal{L}$, β is one of the hyper-parameters to tune the effect of the L2-norm distance.

Proxy-based Classifiers In [4], a cosine classifier is employed to assign a higher probability to the class with a more similar prototype. The cosine similarity is $f(x) = \left(\frac{W}{\|W\|_2}\right)^{\top} \left(\frac{[\phi^*(\mathbf{x}), \phi(\mathbf{x})]}{\|[\phi^*(\mathbf{x}), \phi(\mathbf{x})]\|_2}\right)$. To better partition the embedding vector space, it can use the proxy-based strategy [2], to increase the sparsity of the extracted feature by

$$\mathcal{P}(f(x)) = -\frac{\exp\left(-f(x)\right)}{\sum_{f(z)\in Z} \exp\left(-f(z)\right)}, \tag{2}$$

where all other proxies are Z and $f(z)$ is the consine similarity of each proxy. The visualized mechanism of the proxy for decrease the feature dimension is shown in Figure 1.

Based on the results of Equation 2, we also employ LogSoftmax [1] as the activation function for the cosine linear classifier. The final logits function would be

$$\log(\text{softmax}(\mathcal{P}(f_1(x)), \mathcal{P}(f_2(x)), \ldots, \mathcal{P}(f_n(x))) \tag{3}$$

1.2. Regularization

To tackle the overfitting issues of CIL model, I leverage four major policies, including weight decay of selective parameters, reduction of prototype value, parameter efficient fine-tuning, and employing the learning-rate schedulers.

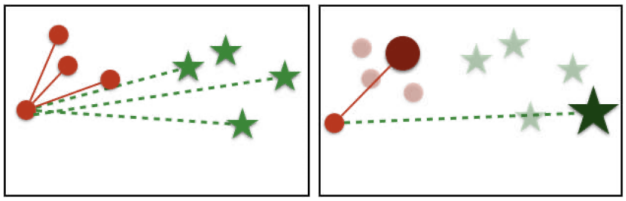


Figure 1. The power of proxies is demonstrated in the example. The left panel shows 48 possible combinations from the instances (small circles/stars). The right panel shows proxies (big circle/star) acting as a compact representation for each.

Parameters Weight Decay. The Weight Decay is done by adding a penalty term to the loss function, which results in smaller weight values. The weight decay value determines how much we penalize larger parameter values. A larger weight decay value means a greater penalty, which results in smaller weights. Conversely, a smaller weight decay value means less penalty, which could allow the weights to become larger. In practice, I add the L2 norm regularization to the loss function as Equation 1. The final loss to be calculated is

$$\hat{\mathcal{L}} = \mathcal{L} + \lambda \sum ||\omega||_2, \tag{4}$$

where ω represents all trainable parameters and λ is the weight decay parameter.

Prototype Random Weighting. To further enhance the generalization ability of CIL, (Pre-Trained Model) PTM [4] is applied in SimpleCIL with ConvNet-empowered embedding function $\phi(\cdot)$. The prototypes [3] are calculated as

$$\boldsymbol{p}_i = \frac{1}{K} \sum_{j=1}^{|\mathcal{D}^b|} \mathbb{I}\left(y_j = i\right) \mathbb{F}_j, \tag{5}$$

where $\mathbb{F}_j = \left[\phi^*\left(\mathbf{x}_j\right), \phi\left(\mathbf{x}_j\right)\right]$ In [4], The averaged embedding represents the most common pattern of the corresponding class, where the prototype is set as the classifier, i.e., $w_i = p_i$, to directly adjust the PTM for CIL by incorporating domain-specific features. To avoid overfitting brought by PTM, instead of using the exact value of p_i. I partially add random noise to each w_i in $\mathbf{w}$ as follows

$$w_i' = \alpha * \frac{1}{K} \sum_{j=1}^{\alpha} \mathbb{I}\left(y_j = i\right) \mathbb{F}_j + (1-\alpha) * \frac{1}{K} \sum_{k=1}^{\alpha} \mathbb{I}\left(y_k = i\right) \mathbb{F}_k, \tag{6}$$

where $\alpha \in [0, 1]$ is a hyper-parameter to assign the weight of the FC layer with partial noises added to randomly selected items in the prototype $\boldsymbol{p}_i$.

Learning-Rate Scheduler. A learning rate scheduler is a component in the training process of a machine learning model that adjusts the learning rate during the training process. The scheduler periodically adjusts the learning rate to decrease the probability of overfitting by a relatively small learning rate, which enables the model's updates to be smaller and reduces the model complexity. In the project, two schedulers are considered, including CosineAnnealingLR and StepLR. For implementation details, please refer to the `minghaocil.py` in the above repository.

Table 1. Hyper-parameter Settings

Hyper-parameter	Optimal	Grid
Epoch	35	[20, 25, 30, 35, 40]
Initial LR	0.2	[0.1,0.15,0.2,0.3]
Optimizer	SGD	[SGD, ADAM]
VPT Type	Deep	[Deep, Shallow]
Prompt Tkns	5	[3,5,8]
Loss Func	CE	[Focal Loss, Cross Entropy]
Alpha	0.3	[0, 0.3, 0.5, 0.8, 1]
Beta	0	[0, 0.01, 0.1, 0.5, 1]
Proxy Num	5	[1, 2, 3, 5, 6]

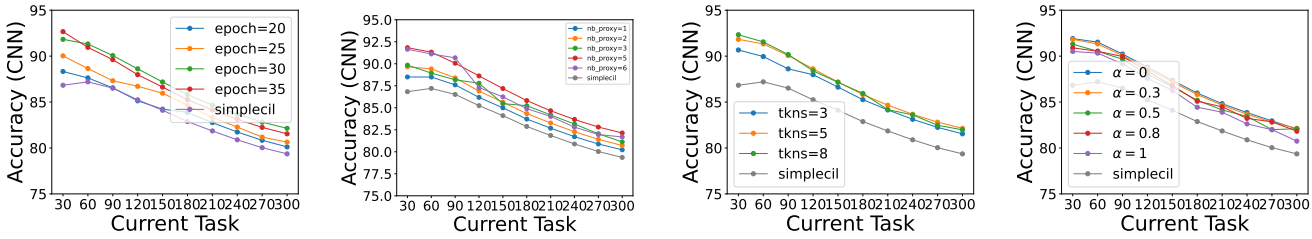
1.3. Traning Settings

For conducting experiments in a more efficient and effective way, this project employs the data parallel technique for the large batch size. Also, to find the optimal hyper-parameter set leading to the fastest convergence, this project uses a grid search policy under three experimental modes.

Data Parallel. This project uses the data parallel technique to boost training and fully utilize the GPU computational resource. In data parallelism, the model is replicated on every GPU. Each GPU receives a different subset of the input data and computes the forward and backward passes for that data. The gradients are then averaged across all GPUs during each update step. The experiments were synchronously conducted on 4 V100 GPU devices. Each device has up to 32G memory to store over 200 batches of the omnibenchmark training dataset.

Three Modes. For better debugging and reproducing experiences, I modified the original experiment workflow (i.e. Train-Evaluation) to three modes. They are 1) Train (only tuning the first task without evaluation), 2) Evaluation (load saved checkpoints and evaluation), and 3) Train-Evaluation. Please refer to `trainer.py` and `minghaocil.py` for details.

Grid Searching Policy. I employ a wide range of grids for each hyper-parameter and tricks to find an optimal parameter set and conduct the ablation experiments. To find the optimal hyper-parameter set, the project simply grid searches the parameters in a suitable region. This method is empowered by the three-fold mode. Please refer to `hp_search.sh` and `generate_config.py` for the experiment setting details.



(a) Epoch settings (b) The number of proxies (c) Prompt tokens (d) Random noise coefficient α

Figure 2. Impacts of the different settings towards the incremental performance.

2. Code Refractory

This section introduces the engineering efforts for the refractory of the original repository, which maintains several severe bugs on 1) data parallel, 2) VPT class, and 3) feature extraction. Besides, I add the load and save function of model checkpoints to reproduce the results better. It is now user-friendly for researchers to conduct experiments on multiple GPUs with checkpoint recorders. To expand the scalability of the CIL experiment, I also add several new arguments, including `"loss_fn"`, `"data_augmentation"`, `"alpha"`, and `"proxy_number"`, in the configuration template to explore their impacts. The project can be further extended as a low-code framework for ablation study for CIL model training.

3. Result

This section exhibits the best result of the proposed solution with the grid of multiple sets of parameters. The optimal setting is based on Adam-VPT using "`Deep`" VPT type. I use SGD as the optimizer with a 0.0005 weight decay coefficient. The loss function for the softmax-based classifier (see Equation 1) is cross-entropy (CE). Besides, the exponent β in the loss function Equation 1 is canceled as 0 for better training accuracy and saving computational resource. Based on the settled parameters, I investigate the incremental performance on the aspects of 1) epoch, 2) the number of proxies, 3) prompt tokens, and 4) the random noise coefficient α.

4. Experiment Analysis

In this section, I introduce the general intuition, mathematical exposition, and ablation study of the proposed optimization and regulation methodologies.

4.1. General Intuition

Since the key idea of CIL is to incorporate new concepts without forgetting old ones, the major direction of improving the performance of CIL model would be how to prevent overfitting and how to enhance the feature sparsity while maintaining training accuracy. In this challenge, only the first task can be trained, whose tuned model will be applied to other incremental data. Thus, we employ all the tricks and policies for de-overfitting to enhance the model extrapolation ability. In the following sections, we mainly analyze de-overfitting and generalization ability impacts.

4.2. Mathematical Exposition

Based on general intuition, in this section, I explain the mathematical mechanism of how the applied tricks work.

LogSoftmax. The softmax function can be unstable if the values of x are very large or small, since $\exp(x)$ can overflow or underflow for large or small values of x respectively. For instance, consider a case where all x_i are large and negative. Then, $\exp(x_i)$ will be very close to zero, and their sum in the denominator may become zero due to underflow, leading to division by zero.

In the log-softmax function, we subtract the maximum x_i from all x_i before exponentiating. This is known as the log-sum-exp trick:

$$\log(\text{softmax}(x)_i) = x_i - \max(x) - \log\left(\sum \exp(x_j - \max(x))\right)$$

Now, the largest $x_j - \max(x) = 0$, which avoids the possibility of overflow in $\exp(x_j - \max(x))$. Likewise, underflow is avoided because we're subtracting the maximum x_i before exponentiating, so we're exponentiating a nonpositive number. This makes log-softmax numerically stable.

Proxy Reduction. Obviously, the proxies are employed for the reduction of the dimension of the features that work well. It is important to prove whether the proxy approximation error is bounded and how the concrete loss will be changed due to the error.

It is assumed that the norms of proxies and data points are constant $|p_x| = N_p$ and $|x| = N_x$, we will denote $\alpha = \frac{1}{N_p N_x}$. The NCA loss (see Equation 2) is proxy bounded

as [2] noted

$$\hat{L}_{NCA}(x, y, Z) \leq \alpha L_{NCA}\left(x, p_y, p_Z\right) + (1-\alpha)\log(|Z|) + 2\sqrt{2\epsilon}, \quad (7)$$

where $\hat{L}_{NCA}$ is defined as L_{NCA} with normalized data points and $|Z|$ is the number of negative points used in the triplet.

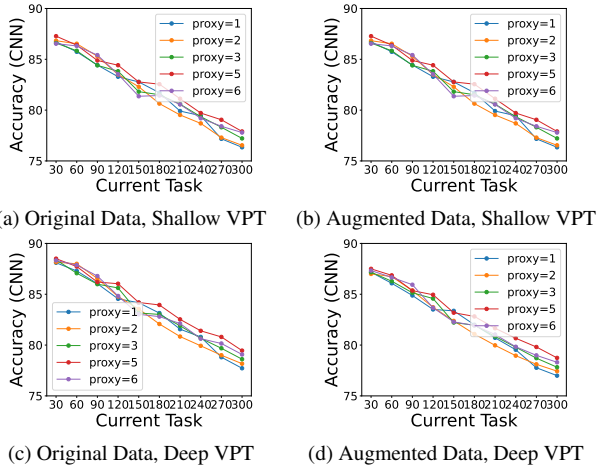
4.3. Ablation Study

Downscale proxy numbers. To evaluate the impact of the number of proxies, we reduced the proxy count in our experiments. The results showed that decreasing the number of proxies could negatively affect the accuracy of the classifier. However, it also highlighted the importance of choosing proxies that are representative of the data distribution.

To further explore the influences of the proxy-based strategy on the data augmentation for different VPT tyoes, I run a 2×2 experiment. From Figure 3, we can see that when the decreasing the proxy numbers, the accuracy is dropping rapidly in general. As illustrated in Figure 3b and Figure 3d, using random data augmentation strategy leads to the accuracy dropped, especially when not using any proxy. Compared to VPT-Shallow, proxy-based strategy has a more pronounced impact on VPT-Deep according to the comparison between Figure 3a and Figure 3c. It is known that VPT can be further divided into two types: VPT-Deep and VPT-Shallow, depending on where the prompts are inserted. VPT-Shallow only learns the prompts in the first transformer block, which indicates that the features extracted by fewer transformer blocks are more sparse that using deeper networks, i.e. VPT-Deep. Thus, the proxy-based strategy can handle the overfitting problem of VPT-Deep by downscaling the dimension of the extracted feature in the prototypes. The results perfectly fit the inner mechanism of the proxy shown in Figure 1.

Intensify Prototype Weighting Manipulation. I also examined the model's incremental performance under various α value in Equation 6 (see subsection 1.2). By artificially modifying more weights of prototype to the data, we observed how the model's performance fluctuates. In the ablation experiments, I conduct the impact of the coefficient α on the both original data and augmented data to see whether the prototype random weighting works on the random input data. We adopt the optimal hyper-parameter settings in Table 1.

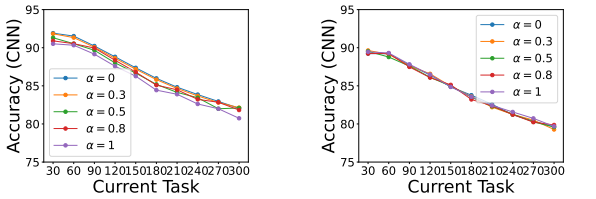
The results in Figure 4 reveal that while the a minor α can enhance the incremental performance on original data, *excessive* random weighting manipulation can significantly hinder its performance. However, utilizing this strategy on the augmented data may not maintain a significant enhancement. The possible reason is that the augmented data has already downsized the complex features and patterns. When applying heavy random weighting manipulation to original data, it might lead to introduce unnecessary noise, which can adversely affect the overall performance. The augmented data for regularization, on the other hand, has already overcome the overfitting problem. Thus, the coefficient α has subtle impacts on the incremental performance.



(a) Original Data, Shallow VPT (b) Augmented Data, Shallow VPT

(c) Original Data, Deep VPT (d) Augmented Data, Deep VPT

Figure 3. Different numbers of proxies for original/augmented data and shallow/deep VPT.



(a) Original Data (b) Augmented Data

Figure 4. Different coefficient α for original images and augmented images.

References

[1] Pierre Blanchard, Desmond J Higham, and Nicholas J Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330, 2021. 1

[2] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proceedings of the IEEE international conference on computer vision*, pages 360–368, 2017. 1, 4

[3] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017. 2

[4] Da-Wei Zhou, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Revisiting class-incremental learning with pre-trained models: Generalizability and adaptivity are all you need. *arXiv preprint arXiv:2303.07338*, 2023. 1, 2