Online Hyperparameter Optimization for Class-Incremental Learning

Yaoyao Liu[1, 2], **Yingying Li**[3], **Bernt Schiele**[1], **Qianru Sun**[4]
[1]Max Planck Institute for Informatics, Saarland Informatics Campus
[2]Department of Computer Science, Johns Hopkins University
[3]Computing and Mathematical Sciences, California Institute of Technology
[4]School of Computing and Information Systems, Singapore Management University
yliu538@jhu.edu, yingli2@caltech.edu, schiele@mpi-inf.mpg.de, qianrusun@smu.edu.sg

Abstract

Class-incremental learning (CIL) aims to train a classification model while the number of classes increases phase-by-phase. An inherent challenge of CIL is the stability-plasticity tradeoff, i.e., CIL models should keep stable to retain old knowledge and keep plastic to absorb new knowledge. However, none of the existing CIL models can achieve the optimal tradeoff in different data-receiving settings—where typically the training-from-half (TFH) setting needs more stability, but the training-from-scratch (TFS) needs more plasticity. To this end, we design an online learning method that can adaptively optimize the tradeoff without knowing the setting as a priori. Specifically, we first introduce the key hyperparameters that influence the tradeoff, e.g., knowledge distillation (KD) loss weights, learning rates, and classifier types. Then, we formulate the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose a specific algorithm to solve it. We apply local estimated rewards and a classic bandit algorithm Exp3 to address the issues when applying online MDP methods to the CIL protocol. Our method consistently improves top-performing CIL methods in both TFH and TFS settings, e.g., boosting the average accuracy of TFH and TFS by 2.2 percentage points on ImageNet-Full, compared to the state-of-the-art. Code is provided at https://class-il.mpi-inf.mpg.de/online/

Introduction

Real-world problems are ever-changing, with new concepts and new data being continuously observed. Ideal AI systems should have the ability to learn new concepts from the new data, also known as *plasticity*, while maintaining the ability to recognize old concepts, also known as *stability*. However, there is a fundamental *tradeoff* between plasticity and stability: too much plasticity may result in the *catastrophic forgetting* of old concepts, while too much stability restricts the ability to adapt to new concepts (McCloskey et al. 1989; McRae et al. 1993; Ratcliff 1990). To encourage related research, Rebuffi et al. (2017) defined the class-incremental learning (CIL) protocol, where the training samples of different classes come to the model phase-by-phase and most of the past data are removed from the memory.

Recently, many methods have been proposed to balance the stability-plasticity tradeoff for *different data-receiving settings* of CIL. For example, the strong feature knowledge distillation (KD) loss function is usually adopted when a large amount of data is available beforehand (e.g., the *training-from-half* (TFH) setting) since it encourages stability (Hou et al. 2019; Liu et al. 2021a,b); while the weak logit KD loss function is popular when data and classes are received evenly in each phase (e.g., the *training-from-scratch* (TFS) setting) since it provides more plasticity (Rebuffi et al. 2017; Belouadah et al. 2019; Li et al. 2016).

Most CIL algorithms pre-fix the tradeoff balancing methods, usually according to which data-receiving setting will be used in the experiments. However, in real-world scenar-

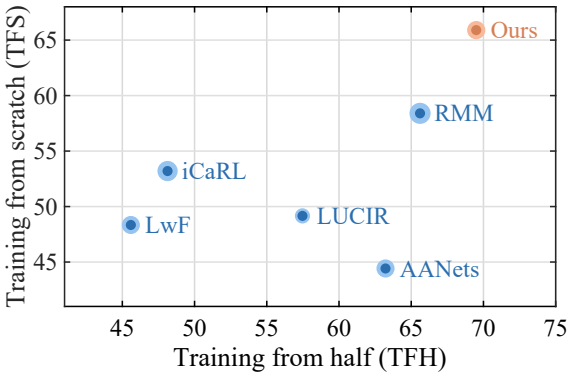

Figure 1: Average accuracy (%) on CIFAR-100 25-phase, using two data-receiving settings: 1) *training-from-half* (TFH): a large amount of data is available beforehand to pre-train the encoder; 2) *training-from-scratch* (TFS): classes come evenly in each phase. Dark blue and orange indicate the baselines and our method, respectively. Light-color circles are confidence intervals. Notice that methods with strong KD losses, e.g., LUCIR (Hou et al. 2019), AANets (Liu et al. 2021a), and RMM (Liu et al. 2021b), tend to provide worse performance in TFS than TFH, while methods with weak KD losses, e.g., iCaRL (Rebuffi et al. 2017) and LwF (Li et al. 2016), tend to provide worse performance in TFH than TFS. Our method uses an online learning algorithm to produce the key hyperparameters, e.g., the weights that control which KD losses are used. Thus, our method achieves the highest performance in both TFS and TFH.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ios, it is difficult to anticipate how data will be received in the future. Hence, a pre-fixed method is no longer proper for balancing the stability and plasticity of the actual data stream, thus generating worse performance. This can also be validated in Figure 1. Notice that the methods with weak KD, e.g., iCaRL (Rebuffi et al. 2017) and LwF (Li et al. 2016), provide worse performance in TFH than in TFS since weak KD provides too much plasticity for TFH, while the methods with strong KD, e.g., LUCIR (Hou et al. 2019), AANets (Liu et al. 2021a), and RMM (Liu et al. 2021b), perform worse in TFS than in TFH due to too much stability.

Therefore, a natural question is: how to design an *adaptive trade-off balancing method* to achieve good performance *without knowing* how data will be received *beforehand*? To tackle this, we propose an online-learning-inspired method to adaptively adjust key hyperparameters that affect the trade-off balancing performance in CIL. In our method, we introduce hyperparameters to control the choice of KD loss functions, learning rates, and classifier types, which are key algorithm choices that affect the tradeoff balancing performance.[1] In this way, deciding this choice is transformed into a hyperparameter optimization (HO) problem.

This HO problem cannot be directly solved because future data are not available. Thus, we borrow ideas from online learning, which is a widely adopted approach to adaptively tune the decisions without knowing the future data a priori while still achieving good performance in hindsight.

However, in CIL, our decisions affect not only the next phase but also all the future phases, which is different from the standard online learning setting (Anderson 2008). To capture the dependence across phases, we formulate the HO problem in CIL as an online MDP, which is a generalized version of online learning. Further, we propose a new algorithm based on (Even-Dar et al. 2009) to solve this online MDP problem. Our algorithm differs from the standard online MDP algorithm in (Even-Dar et al. 2009) in two aspects:

- In CIL, we cannot directly observe the reward (i.e., validation accuracy) because the validation data is not accessible during training. To address this issue, we estimate the reward by rebuilding local training and validation sets during policy learning in each phase, and computing the estimated reward on the local validation sets.
- In CIL, we only have access to the model generated by the selected hyperparameters instead of the models generated by other hyperparameters. In other words, we only have bandit feedback instead of full feedback as assumed in (Even-Dar et al. 2009). To address this, we revise the algorithm in (Even-Dar et al. 2009) by combining it with a classic bandit algorithm, Exp3 (Auer et al. 2002).

Empirically, we find our method performs well consistently. We conduct extensive CIL experiments by plugging our method into three top-performing methods (LUCIR, AANets, and RMM) and testing them on three benchmarks (i.e., CIFAR-100, ImageNet-Subset, and ImageNet-Full). Our results show the consistent improvements of the proposed method, e.g., boosting the average accuracy of TFH and TFS by 2.2 percentage points on ImageNet-Full, compared to the state-of-the-art (Liu et al. 2021b).

Lastly, it is worth mentioning that our method can also be applied to optimize other key hyperparameters in CIL, e.g., memory allocation (Liu et al. 2021b).

Summary of our contributions. Our contributions are three-fold: 1) an online MDP formulation that allows online updates of hyperparameters that affect the balance of the stability and plasticity in CIL; 2) an Exp3-based online MDP algorithm to generate adaptive hyperparameters using bandit and estimated feedback; 3) extensive comparisons and visualizations for our method in three CIL benchmarks, taking top-performing methods as baselines.

Related Work

Class-incremental learning (CIL). There are three main lines of work to address the stability-plasticity trade-off in CIL. *Distillation-based* methods introduce different knowledge distillation (KD) losses to consolidate previous knowledge when training the model on new data. The key idea is to enforce model prediction logits (Li et al. 2016; Rebuffi et al. 2017), feature maps (Douillard et al. 2020; Hou et al. 2019), or topologies in the feature space (Tao et al. 2020) to be close to those of the pre-phase model. *Memory-based* methods (Rebuffi et al. 2017; Shin et al. 2017; Liu et al. 2020; Prabhu et al. 2020) preserve a small number of old class data (called exemplars) and train the model on them together with new class data. *Network-architecture-based* methods (Rusu et al. 2016; Xu et al. 2018; Abati et al. 2020; Yan et al. 2021) design incremental network architectures by expanding the network capacity for new data or freezing partial network parameters to keep the knowledge of old classes. However, all the above methods are either too stable or too plastic to perform well in both TFS and TFH settings. Our method learns an online policy to generate hyperparameters that balance stability and plasticity. Therefore, our method performs well in both TFS and TFH settings.

Reinforcement learning (RL) aims to learn a policy in an environment, which is typically formulated as an MDP. Some CIL papers also deploy RL algorithms in their frameworks. Xu et al. (2018) used RL to expand its backbone network when a new task arrives adaptively. Liu et al. (2021b) used RL to learn a policy to adjust the memory allocation between old and new class data dynamically along with the learning phases. Our method focuses on learning a policy to produce key hyperparameters. Besides, the existing methods need to solve the complete MDP, which is time-consuming. Here, we formulate the CIL task as an online MDP. Thus, our method is more time-efficient.

Online learning observes a stream of samples and makes a prediction for each element in the stream. There are mainly two settings in online learning: full feedback and bandit feedback. *Full feedback* means that the full reward function is given at each stage. It can be solved by Best-Expert algorithms (Even-Dar et al. 2005). *Bandit feedback* means that

[1] The impact of KD losses has been discussed before. Learning rates naturally affect how fast the model learns new concepts. We also adjust the classifier type because empirical results (Rebuffi et al. 2017; Hou et al. 2019) show that the nearest class mean (NCM) and fully-connected (FC) classifiers perform better under more plasticity and stability, respectively.

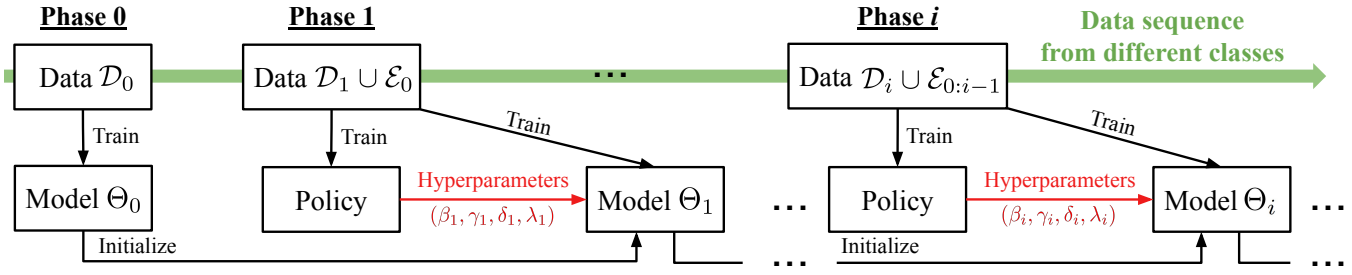

Figure 2: The computing flow of our method. We formulate the CIL task as an online MDP: each phase in CIL is a stage in the MDP, and the CIL models are the states. We train the policy to produce actions, which contain the hyperparameters we use in the CIL training. We illustrate the training process of each phase in Figure 3.

only the reward of the implemented decision is revealed. If the rewards are independently drawn from a fixed and unknown distribution, we may use e.g., Thompson sampling (Agrawal et al. 2012) and UCB (Auer et al. 2010) to solve it. If the rewards are generated in a non-stochastic version, we can solve it by e.g., Exp3 (Auer et al. 2002). *Online MDP* is an extension of online learning. Many studies (Even-Dar et al. 2009; Li et al. 2019a,b, 2021) aim to solve it by converting it to online learning. In our case, we formulate the CIL as an online MDP and convert it into a classic online learning problem. The rewards in our MDP are non-stochastic because the training and validation data change in each phase. Therefore, we design our algorithm based on Exp3 (Auer et al. 2002).

Hyperparameter optimization (HO). There are mainly two popular lines of HO methods: gradient-based and meta-learning-based. Gradient-based HO methods (Baydin et al. 2018) make it possible to tune the entire weight vectors associated with a neural network layer as hyperparameters. Meta-learning-based HO methods (Franceschi et al. 2018) use a bilevel program to optimize the hyperparameters. However, all these methods only consider time-invariant environments. Our online method learns hyperparameters that adapt to the time-varying environments in CIL.

Preliminaries

Class-incremental learning (CIL). The general CIL pipeline is as follows. There are multiple phases during which the number of classes gradually increases to the maximum (Douillard et al. 2020; Hou et al. 2019; Hu et al. 2021; Liu et al. 2020). In the 0-th phase, we observe data $\mathcal{D}_0$, and use it to learn an initial model Θ_0. After this phase, we can only store a small subset of $\mathcal{D}_0$ (i.e., exemplars denoted as $\mathcal{E}_0$) in memory used as replay samples in later phases. In the i-th phase ($i \geq 1$), we get new class data $\mathcal{D}_i$ and load exemplars $\mathcal{E}_{0:i-1}=\mathcal{E}_0 \cup \cdots \cup \mathcal{E}_{i-1}$ from the memory. Then, we initialize Θ_i with Θ_{i-1}, and train it using $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. We evaluate the model Θ_i on a test set $\mathcal{Q}_{0:i}$ for all classes observed so far. After that, we select exemplars $\mathcal{E}_{0:i}$ from $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ and save them in the memory.

Existing work mainly focus on two data-receiving settings: *training-from-half* (TFH) (Hou et al. 2019) and *training-from-scratch* (TFS) (Rebuffi et al. 2017) settings. In TFH, we are given half of all classes in the 0-th phase, and observe the remaining classes evenly in the subsequent N phases. In TFS, we are given the same number of classes in all N phases.

Methodology

As illustrated in Figures 2 and 3, we formulate CIL as an online MDP and learn a policy to produce the hyperparameters in each phase. In this section, we introduce the online MDP formulation, show optimizable hyperparameters, and provide an online learning algorithm to train the policy.

An Online MDP Formulation for CIL

Optimizing hyperparameters in CIL should be online inherently: training and validation data changes in each phase, so the hyperparameters should be adjusted accordingly. Thus, it is intuitive to formulate the CIL as an online MDP. In the following, we provide detailed formulations.

Stages. Each phase in the CIL task can be viewed as a stage in the online MDP.

States. The state should define the current situation of the intelligent agent. In CIL, we use the model Θ_i as the state of the i-th phase/stage. We use $\mathbb{S}$ to denote the state space.

Actions. We use a vector consisting of the hyperparameters in the i-th phase as the action $\mathbf{a}_i$. When we take the action $\mathbf{a}_i$, we deploy the corresponding hyperparameters. We denote the action space as $\mathbb{A}$. Please refer to the next subsection, *Optimizable Hyperparameters*, for more details.

Policy $\mathbf{p}=\{p(\mathbf{a}|\Theta_i)\}_{\mathbf{a}\in\mathbb{A}}$ is a probability distribution over the action space $\mathbb{A}$, given the current state Θ_i.

Environments. We define the training and validation data in each phase as the environment. In the i-th phase, the environment is $\mathcal{H}_i=(\mathcal{E}_{0:i-1} \cup \mathcal{D}_i, \mathcal{Q}_{0:i})$, where $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ is the training data and $\mathcal{Q}_{0:i}$ is the corresponding validation data. The environment is time-varying because we are given different training and validation data in each phase.

Rewards. CIL aims to train a model that is efficient in recognizing all classes seen so far. Therefore, we use the validation accuracy as the reward in each phase. Our objective is to maximize a cumulative reward, i.e., $R = \sum_{i=1}^{N} r_{\mathcal{H}_i}(\Theta_i, \mathbf{a}_i)$, where $r_{\mathcal{H}_i}(\Theta_i, \mathbf{a}_i)$ denotes the i-th phase reward, i.e., the validation accuracy of Θ_i. The reward function $r_{\mathcal{H}_i}$ changes with $\mathcal{H}_i$, so it is time-varying.

Optimizable Hyperparameters

In this part, we introduce the optimizable hyperparameters, and how to define the actions and action space based on the hyperparameters. We consider three kinds of hyperparameters that significantly affect stability and plasticity: 1) KD loss weights, 2) learning rates, and 3) classified types.

1) KD loss weights. We first introduce two KD losses (i.e., logit and feature KD losses) and then show how to use the KD loss weights to balance them.

Logit KD loss is proposed in (Hinton et al. 2015) and widely applied in CIL methods (Li et al. 2016; Rebuffi et al. 2017; Liu et al. 2021a). Its motivation is to make the current model Θ_i mimic the prediction logits of the old model Θ_{i-1}:

$$\mathcal{L}_{\text{logi}} = -\sum_{k=1}^{K} \eta_k(\mu(x;\Theta_{i-1})) \log \eta_k(\mu(x;\Theta_i)), \quad (1)$$

where $\mu(x;\Theta)$ is a function that maps the input mini-batch (x, y) to the prediction logits using the model Θ. $\eta_k(v) = v_k^{1/\tau} / \sum_j v_j^{1/\tau}$ is a re-scaling function for the k-th class prediction logit and τ is a scalar set to be greater than 1.

Feature KD loss (Hou et al. 2019; Douillard et al. 2020) aims to enforce a stronger constraint on the previous knowledge by minimizing the cosine similarity between the features from the current model Θ_i and the old model Θ_{i-1}. It can be computed as follows,

$$\mathcal{L}_{\text{feat}} = 1 - S_c(f(x;\Theta_i), f(x;\Theta_{i-1})), \quad (2)$$

where $f(x;\Theta)$ denotes a function that maps the input image x to the features using the model Θ. $S_c(v_1, v_2)$ denotes the cosine similarity between v_1 and v_2.

The overall loss in the i-th phase is a weighted sum of the classification and different KD losses:

$$\mathcal{L}_{\text{overall}} = \mathcal{L}_{\text{CE}} + \beta_i \mathcal{L}_{\text{logi}} + \gamma_i \mathcal{L}_{\text{feat}}, \quad (3)$$

where β_i and γ_i are the weights of the logit and feature KD losses, respectively. $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy classification loss. Existing methods (Li et al. 2016; Rebuffi et al. 2017; Hou et al. 2019; Liu et al. 2021a,b) can be viewed as using fixed heuristic KD loss weights, e.g., $\beta_i \equiv 1$ and $\gamma_i \equiv 0$ in iCaRL (Li et al. 2016; Rebuffi et al. 2017). Instead, our method optimizes β_i and γ_i online. Thus, we can balance the model's stability and plasticity by adjusting β_i and γ_i. We apply different weights for the logit and feature KD losses so that we can achieve fine-grained control over the intensity of knowledge distillation.

2) Learning rate is another important hyperparameter that affects the model's stability and plasticity. We empirically find that a lower learning rate makes the CIL model more stable, while a higher learning rate makes the CIL model more plastic. If we use λ_i to denote the learnable learning rate in the i-th phase, we update the CIL model as follows,

$$\Theta_i \leftarrow \Theta_i - \lambda_i \nabla_{\Theta_i} \mathcal{L}_{\text{overall}}. \quad (4)$$

Another hyperparameter, the number of training epochs, has similar properties to the learning rate. We choose to optimize the learning rate and fix the number of epochs because it empirically works better with our online learning algorithm.

3) Classifier type. Motivated by the empirical analysis, we consider two classifier types in our study: *nearest class mean* (NCM) (Rebuffi et al. 2017; Snell et al. 2017) and *fully-connected* (FC) (Hou et al. 2019; Liu et al. 2020) classifiers. For the NCM classifier, we first compute the mean feature for each class using the new data and old exemplars. Then we perform a nearest neighbor search using the Euclidean distance on the L_2 normalized mean features to get the final predictions. It is observed empirically that the NCM classifier tends to work better on the models with high plasticity, while the FC classifier performs better on the models with high stability (Rebuffi et al. 2017; Hou et al. 2019). Thus, we propose to use a hyperparameter, classifier type indicator δ_i, to control the final predictions during the evaluation:

$$\mu(x;\Theta_i) = \mu_{\text{ncm}}(x;\Theta_i)[\delta_i = 1] + \mu_{\text{fc}}(x;\Theta_i)[\delta_i = 0], \quad (5)$$

where $\delta_i \in \{0, 1\}$, μ_{ncm} and μ_{fc} are the predictions on the input image x using the NCM and FC classifiers, respectively.

Summary: actions and action space. In summary, we define the action as $\mathbf{a}_i = (\beta_i, \gamma_i, \lambda_i, \delta_i,)$, which consists of the following hyperparameters: KD loss weights β_i and γ_i, learning rate λ_i, and classifier type indicator δ_i. For the hyperparameters that may vary in a continuous range, we *discretize* them to define a *finite* action space.[2] In the next subsection, we show how to learn the policy in each phase.

Policy Learning

A common approach to solving an online MDP is to approximate it as an online learning problem and solve it using online learning algorithms (Even-Dar et al. 2005; Agrawal et al. 2012; Auer et al. 2002). We also take this approach, and our approximation follows (Even-Dar et al. 2009), which achieves the optimal regret. In their paper, Even-Dar et al. (2009) relax the Markovian assumption of the MDP by decoupling the cumulative reward function and letting it be time-dependent so that they can solve the online MDP by standard online learning algorithms. Such a decoupling requires the following assumptions. 1) Fast mixing: in CIL, the hyperparameters in an early phase do not have much impact on the test accuracy of the classes observed in the current phase. 2) The algorithm changes the hyperparameters slowly (this can be observed in ***Experiments*** & Figure 5). Thus, these assumptions fit our CIL problem.

However, we cannot directly apply the algorithms proposed in (Even-Dar et al. 2009) to our problem. It is because their paper assumes *full feedback*, i.e., we can observe the rewards of all actions in each phase. Therefore, its online learning problem could be solved by Best Expert algorithms (Even-Dar et al. 2005). In CIL, we cannot observe any reward (i.e., validation accuracy) because the validation data is not accessible during training. To address this issue, we rebuild the local training and validation sets in each phase. In this way, our problem has *bandit feedback*: we can compute the reward of the implemented action. Therefore, we can solve our online learning problem based on Exp3 (Auer et al. 2002), a famous bandit algorithm.

[2]Though discretization suffers the curse of dimensionality, our experiments show that with a coarse grid, we already have significant improvements over pre-fixed hyperparameters.

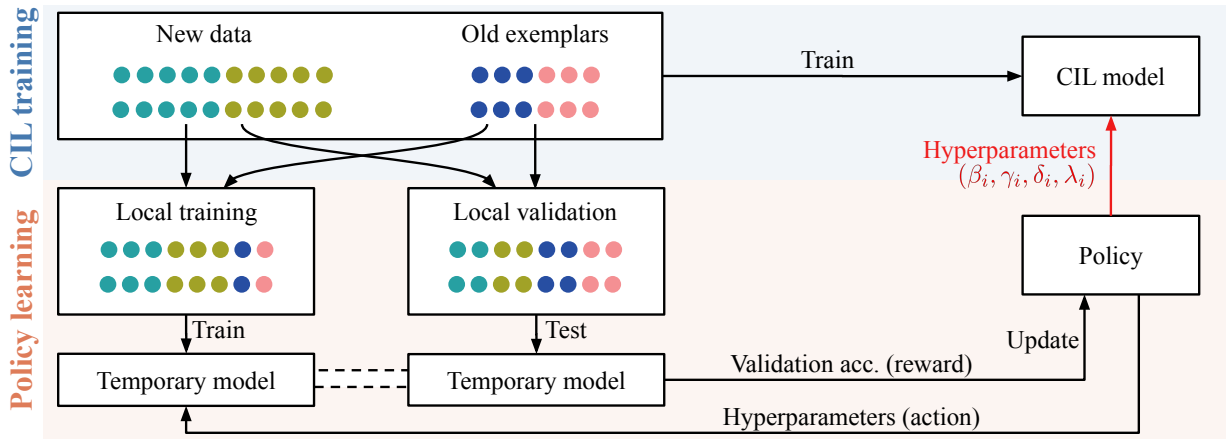

Figure 3: The training process of our online learning method in the i-th phase. It includes *policy learning* and *CIL training*. (a) Policy learning. 1) We construct a class-balanced subset from all training data as the local validation set and use the remaining data as the local training set. 2) We initialize the temporary model with Θ_{i-1}. 3) We sample an action using the current policy and deploy the hyperparameters on the temporary model according to the action. 4) We train it on the local training set for M_1 epochs, and evaluate it on the local test set. 5) We use the validation accuracy as the reward and update the policy. We update the policy for T iterations by repeating Steps 2-5. (b) CIL training. We sample an action using the learned policy and deploy the hyperparameters on the CIL model. Then, we train the CIL model on all training data for M_2 epochs.

In the following, we show how to rebuild the local training and validation sets, compute the decoupled cumulative reward, and learn the policy with Exp3.

Rebuilding local datasets. In the i-th phase, we need to access the validation set $\mathcal{Q}_{0:i}$ to compute the reward (i.e., the validation accuracy). However, we are not allowed to use $\mathcal{Q}_{0:i}$ during training because it violates the CIL benchmark protocol. Therefore, we replace $\mathcal{Q}_{0:i}$ with a class-balanced subset $\mathcal{B}_{0:i}$ sampled from the training data $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. $\mathcal{B}_{0:i}$ contains the same number of samples for both the old and new classes. In this way, we can rebuild the local training and validation sets, and obtain the local environment $h_i = ((\mathcal{E}_{0:i-1} \cup \mathcal{D}_i) \setminus \mathcal{B}_{0:i}, \mathcal{B}_{0:i})$.

Decoupled cumulative reward. We create the decoupled cumulative reward function $\hat{R}$ based on the original cumulative reward function $R = \sum_{j=1}^{N} r_{\mathcal{H}_j}(\Theta_j, \mathbf{a}_j)$. In the i-th phase, we compute $\hat{R}$ as follows,

$$\hat{R}(\mathbf{a}_i, h_i) = \underbrace{\sum_{j=1}^{i-1} r_{\mathcal{H}_j}(\Theta_j, \mathbf{a}_j)}_{\text{Part I}} + \underbrace{\sum_{j=i}^{i+n} r_{h_i}(\Theta_j, \mathbf{a}_i)}_{\text{Part II}}, \quad (6)$$

where Part I is the historical rewards from the 1-st phase to the (i-1)-th phase. It is a constant and doesn't influence policy optimization. Part II is the long-term reward of a time-invariant local MDP based on the local environment h_i. We use Part II as an estimation of the future rewards, following (Even-Dar et al. 2009). Because we don't know the total number of phases N during training, we assume there are n phases in the future. Furthermore, we fix the action $\mathbf{a}_i$ in Part II to simplify the training process. Thus, $\hat{R}$ can be reviewed as a function of $\mathbf{a}_i$ and h_i.

Training policy with Exp3. Exp3 (Auer et al. 2002) introduces an auxiliary variable $\mathbf{w} = \{w(\mathbf{a})\}_{\mathbf{a}\in\mathbb{A}}$. After updating $\mathbf{w}$, we can determine the policy $\mathbf{p}=\{p(\mathbf{a}|\Theta_i)\}_{\mathbf{a}\in\mathbb{A}}$ by $\mathbf{p} = \mathbf{w}/||\mathbf{w}||$. The updating rule of w is provided below.

In the 1-st phase, we initialize $\mathbf{w}$ as $\{1, \ldots, 1\}$. In each phase i ($i \geq 1$), we update $\mathbf{w}$ for T iterations. In the t-th iteration, we sample an action $\mathbf{a}_t \sim \mathbf{p}$, apply the action $\mathbf{a}_t$ to the CIL system, and compute the decoupled cumulative reward $\hat{R}(\mathbf{a}_t, h_i)$ using Eq. 6. After that, we update $w(\mathbf{a}_t)$ in $\mathbf{w}$ as,

$$w(\mathbf{a}_t) \leftarrow w(\mathbf{a}_t) \exp(\xi \hat{R}(\mathbf{a}_t, h_i)/p(\mathbf{a}_t|\Theta_i)), \quad (7)$$

where ξ can be regarded as the learning rate in Exp3.

Experiments

We evaluate the proposed method on three CIL benchmarks, incorporate our method into three top-performing baseline methods, and boost their performances consistently in all settings. Below we describe the datasets and implementation details, followed by the results and analyses.

Datasets and Implementation Details

Datasets. We employ CIFAR-100 (Krizhevsky et al. 2009), ImageNet-Subset (Rebuffi et al. 2017) (100 classes), and ImageNet-Full (Russakovsky et al. 2015) (1000 classes) as the benchmarks. We use the same data splits and class orders as the related work (Rebuffi et al. 2017; Liu et al. 2021a,b).

Network architectures. We use a modified 32-layer ResNet for CIFAR-100 and an 18-layer ResNet for ImageNet, following (Rebuffi et al. 2017; Hou et al. 2019; Liu et al. 2021a). We deploy the AANets (2021a) for the experiments based on AANets and RMM (2021b). Further, we use a cosine normalized classifier without bias terms as the FC classifier, following (Hou et al. 2019; Liu et al. 2020).

Configurations. We discretize the hyperparameter search space into 50 actions, i.e., card($\mathbb{A}$)=50. We update the policy

Methods	**CIFAR-100**, N=5			**CIFAR-100**, N=25			**ImgNet-Sub**, N=5			**ImgNet-Sub**, N=25		
	TFH	TFS	Avg.	TFH	TFS	Avg.	TFH	TFS	Avg.	TFH	TFS	Avg.
PODNet (2020)	64.7	63.6	64.2	60.3	45.3	52.8	64.3	58.9	61.6	68.3	39.1	53.7
DER (2021)	67.6	72.3	70.0	65.5	67.3	66.4	78.4	76.9	77.7	75.4	71.0	73.2
FOSTER (2022)	70.4	72.5	71.5	63.8	**70.7**	67.3	80.2	78.3	79.3	69.3	72.9	71.1
LUCIR (2019)	63.1	63.0	63.1	57.5	49.2	53.4	65.3	66.7	66.0	61.4	46.2	53.8
w/ ours	63.9	64.9	64.4	59.3	52.4	55.9	70.6	68.4	69.5	62.9	54.1	58.5
AANets (2021a)	65.3	63.1	64.2	63.2	44.4	53.8	77.0	68.9	73.0	72.2	60.7	66.5
w/ ours	67.0	65.1	66.1	64.1	50.3	57.2	77.3	70.6	74.0	72.9	64.8	68.9
RMM (2021b)	67.6	70.4	69.0	65.6	58.4	62.0	79.5	80.5	80.0	75.0	71.6	73.3
w/ ours	**70.8**	**72.7**	**71.8**	**69.5**	65.9	**67.7**	**81.0**	**82.2**	**81.6**	**76.1**	**73.2**	**74.7**

Table 1: Average accuracy (%) across all phases on CIFAR-100 and ImageNet-Subset (ImgNet-Sub). The first block shows some recent CIL methods. The second block shows three top-performing baselines (Hou et al. 2019; Liu et al. 2021a,b) *w/* and *w/o* our method plugged in. "TFH" and "TFS" denote the *training-from-half* and *training-from-scratch* settings, respectively. "Avg." shows the average of the "TFH" and "TFS" results. For "AANets" (Liu et al. 2021a), we use its version based on PODNet (Douillard et al. 2020). We rerun the baselines using their open-source code in a unified setting for a fair comparison.

Methods	**ImageNet-Full**, N=5		
	TFH	TFS	Avg.
LUCIR (2019)	64.5	62.7	62.0
w/ ours	65.8	66.1	66.0
RMM (2021b)	69.0	66.1	67.6
w/ ours	**70.7**	**68.9**	**69.8**

Table 2: Average accuracy (%) on ImageNet-Full.

for 25 iterations in each phase, i.e., T=25. For other configurations, we follow the corresponding baselines.

Results and Analyses

Tables 1 and 2 present the results of top-performing baselines *w/* and *w/o* our method and some recent related work. Table 3 summarizes the results in seven ablative settings. Figure 4 compares the activation maps (using Grad-CAM (Selvaraju et al. 2017)) produced by diffident methods in TFH and TFS. Figure 5 shows the values of hyperparameters produced by our method.

Comparison with the state-of-the-art. Tables 1 and 2 show that taking our method as a plug-in module for the state-of-the-art (Liu et al. 2021b) and other baselines (Hou et al. 2019; Liu et al. 2021a) consistently improves their performance. For example, RMM (2021b) *w/* ours gains 4.3 and 2.2 percentage points on CIFAR-100 and ImageNet-Full, respectively. Interestingly, we find that we can surpass the baselines more when the number of phases N is larger. E.g., on CIFAR-100, our method improves RMM by 5.7 percentage points when N=25, while this number is 2.8 percentage points when N=5. Our explanation is that the forgetting problem is more serious when the number of phases is larger. Thus, we need better hyperparameters to balance stability and plasticity.

No.	Optimizing			N=5		N=25	
	(β,γ)	δ	λ	TFH	TFS	TFH	TFS
1	Baseline			63.11	62.96	57.47	49.16
2	✓			63.20	63.60	58.27	50.91
3	✓	✓		63.23	64.08	58.20	51.94
4	✓	✓	✓	**63.88**	**64.92**	**59.27**	**52.44**
5	Cross-val fixed			63.33	64.02	57.50	51.64
6	Offline RL (2021b)			63.42	63.88	58.12	51.53
7	Bilevel HO (2018)			63.20	63.02	57.56	49.42

Table 3: Ablation results (average accuracy %) on CIFAR-100. (β,γ) are KD loss weights. λ and δ denote learning rates and classifier types, respectively. The baseline is LUCIR (2019). Row 4 shows our best result.

Ablation study. Table 3 concerns eight ablation settings, and shows the results in both TFH and TFS for different numbers of phases (N=5/25). The detailed analyses are as follows.

1) First block. Row 1 shows the baseline (Hou et al. 2019).

2) Second block: optimizable hyperparameters. In our study, we optimize three kinds of hyperparameters that affect the model's stability and plasticity: KD loss weights (β,γ), learning rate λ, and classifier type indicator δ. Comparing Row 2 to Row 1, we can observe that optimizing the KD loss weights boosts the TFS accuracy more significantly. It is because the baseline, LUCIR (Hou et al. 2019), applies a strong regularization term (i.e., feature KD loss) which harms the TFH performance. Our method changes the regularization term by adjusting the KD loss weights, so it achieves better performance. Comparing Row 3 to Row 2, we can see that optimizing the classifier type indicator δ performs further improvements, especially in TFS. It is because the baseline deploys an FC classifier by default while our method learns to switch between different classifiers in

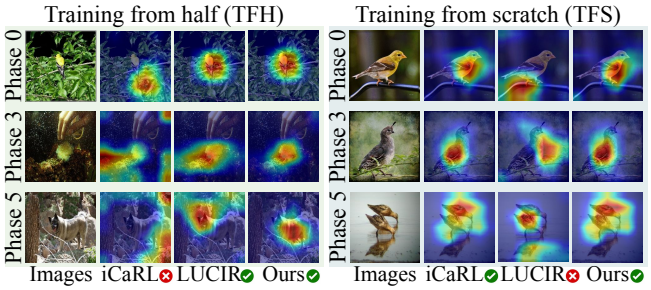

Figure 4: The activation maps using Grad-CAM (Selvaraju et al. 2017) for the last phase model on ImageNet-Subset 5-phase. Samples are selected from the classes coming in the 0-th, 3-rd, and 5-th phases. Green ticks mean successful activation of discriminative features on object regions, while red crosses mean unsuccessful.

different settings. Comparing Row 4 to Row 3, we can see the effectiveness of optimizing the learning rates in CIL. In summary, it is impressive that optimizing three kinds of hyperparameters together achieves the best results.

3) Third block: hyperparameter learning methods. For Row 5, we use cross-validation (i.e., all past, future, and validation data are accessible) to find a set of fixed hyperparameters and apply them to all phases. We can see that Row 5 results are consistently lower than ours in Row 4, although it can access more data. It shows that we need to update the hyperparameters online in different phases. For Row 6, we use the policy pre-trained in the 0-th phase of the target CIL task using the framework proposed in (Liu et al. 2021b) (compared to ours, it is offline). Comparing Row 6 with Row 4, we are happy to see that our online learning algorithm achieves better performance than the offline RL while we use much less training time (see the analysis in the supplementary). For Row 7, we use the bilevel hyperparameter optimization method (Franceschi et al. 2018). Comparing Row 7 with Row 4, we can observe that our method achieves more significant performance improvements. The reason is that (Franceschi et al. 2018) is designed for time-invariant environments, while our online algorithm can adapt to the time-varying environments in CIL.

Visualizing activation maps. Figure 4 demonstrates the activation maps visualized by Grad-CAM (Selvaraju et al. 2017) for the final model (obtained after five phases) on ImageNet-Subset 5-phase. The left and right sub-figures show the results for *training-from-half* (TFH) and *training-from-scratch* (TFS), respectively. We can observe: 1) LUCIR (Hou et al. 2019) makes predictions according to foreground (correct) and background (incorrect) regions in the TFH and TFS settings, respectively; 2) iCaRL (Rebuffi et al. 2017) behaves opposite to LUCIR in the two settings; 3) our method always makes predictions according to foreground (correct) regions in both TFH and TFS. The reasons are as follows. LUCIR applies a strong (feature) KD by default, so it performs better in TFH. iCaRL applies a weak (logit) KD by default, so it performs better in TFS. Our method can adjust the hyperparameters to change between different KD losses, so it performs well in both settings.

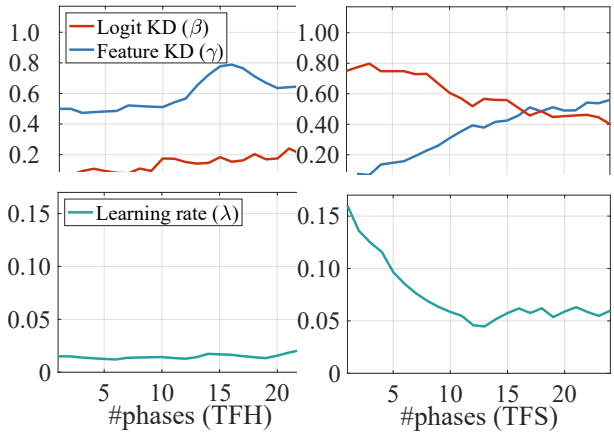

Figure 5: The hyperparameter values produced by our policy on CIFAR-100 25-phase for LUCIR (2019) *w/* ours. We smooth all curves with a rate of 0.8 for better visualization.

Hyperparameter values. Figure 5 shows the hyperparameter values produced by our policy on CIFAR-100 25-phase.

1) KD loss weights β and γ. From the figure, we have two observations. a) The policy learns to produce a larger initial value for γ and β in TFH and TFS, respectively. Our explanation is as follows. In TFH, we already have a pre-trained model, so we need a strong regularization term (i.e., feature KD loss) to make the model more stable and avoid forgetting. In TFS, we start from random initialization, so we need a weak regularization term (i.e., logit KD loss) to improve the model's plasticity. b) Both β and γ increase in TFH, while β decreases and γ increases in TFS. It is because we need stronger regularization to maintain the knowledge when more data is observed. The policy achieves that in different ways: it assigns higher weights for both KD losses in TFH, while it transfers from logit KD to feature KD in TFS.

2) Learning rates λ. In Figure 5, we can observe that the learning rate in TFS is much higher than in TFH. It can be explained that 1) we need a higher learning rate in TFS as the model is trained from scratch and needs to capture more new knowledge; 2) we need a lower learning rate in TFH because we need to avoid forgetting the pre-trained model.

3) Classifier type indicator δ. On CIFAR-100 25-phase, our policy learned to choose the NCM and FC classifiers in TFS and TFH, respectively.

Conclusions

In this study, we introduce a novel framework that allows us to optimize hyperparameters online to balance the stability and plasticity in CIL. To achieve this, we formulate the CIL task as an online MDP and learn a policy to produce the hyperparameters. Our approach is generic, and it can be easily applied to existing methods to achieve large-margin improvements in both TFS and TFH settings. It is worth mentioning that our method can also be applied to optimize other key hyperparameters in CIL.

Acknowledgments

This research was supported by A*STAR under its AME YIRG Grant (Project No. A20E6c0101).

References

Abati, D.; Tomczak, J.; Blankevoort, T.; Calderara, S.; Cucchiara, R.; and Bejnordi, B. E. 2020. Conditional Channel Gated Networks for Task-Aware Continual Learning. In *CVPR*, 3931–3940.

Agrawal, S.; and Goyal, N. 2012. Analysis of Thompson Sampling for the Multi-armed Bandit Problem. In *COLT*, volume 23, 39.1–39.26.

Anderson, T. 2008. *The theory and practice of online learning*. Athabasca University Press.

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1): 48–77.

Auer, P.; and Ortner, R. 2010. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2): 55–65.

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2018. Automatic differentiation in machine learning: a survey. *JMLR*, 18: 1–43.

Belouadah, E.; and Popescu, A. 2019. Il2m: Class incremental learning with dual memory. In *CVPR*, 583–592.

Douillard, A.; Cord, M.; Ollion, C.; Robert, T.; and Valle, E. 2020. PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning. In *ECCV*, 86–102.

Even-Dar, E.; Kakade, S. M.; and Mansour, Y. 2005. Experts in a Markov decision process. In *NIPS*, 401–408.

Even-Dar, E.; Kakade, S. M.; and Mansour, Y. 2009. Online Markov decision processes. *Mathematics of Operations Research*, 34(3): 726–736.

Franceschi, L.; Frasconi, P.; Salzo, S.; Grazzi, R.; and Pontil, M. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 1568–1577.

Hinton, G. E.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. *arXiv*, 1503.02531.

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, X.; Tang, K.; Miao, C.; Hua, X.-S.; and Zhang, H. 2021. Distilling Causal Effect of Data in Class-Incremental Learning. In *CVPR*, 3957–3966.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Li, Y.; Das, S.; and Li, N. 2021. Online optimal control with affine constraints. In *AAAI*, 8527–8537.

Li, Y.; and Li, N. 2019a. Online learning for markov decision processes in nonstationary environments: A dynamic regret analysis. In *2019 American Control Conference (ACC)*, 1232–1237. IEEE.

Li, Y.; Zhong, A.; Qu, G.; and Li, N. 2019b. Online markov decision processes with time-varying transition probabilities and rewards. In *ICML workshop on Real-world Sequential Decision Making*.

Li, Z.; and Hoiem, D. 2016. Learning Without Forgetting. In *ECCV*, 614–629.

Liu, Y.; Schiele, B.; and Sun, Q. 2021a. Adaptive Aggregation Networks for Class-Incremental Learning. In *CVPR*, 2544–2553.

Liu, Y.; Schiele, B.; and Sun, Q. 2021b. RMM: Reinforced Memory Management for Class-Incremental Learning. In *NeurIPS*, 3478–3490.

Liu, Y.; Su, Y.; Liu, A.; Schiele, B.; and Sun, Q. 2020. Mnemonics Training: Multi-Class Incremental Learning without Forgetting. In *CVPR*, 12245–12254.

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.

McRae, K.; and Hetherington, P. 1993. Catastrophic Interference is Eliminated in Pre-Trained Networks. In *CogSci*.

Prabhu, A.; Torr, P. H.; and Dokania, P. K. 2020. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. In *ECCV*, 524–540.

Ratcliff, R. 1990. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97: 285–308.

Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. iCaRL: Incremental classifier and representation learning. In *CVPR*, 5533–5542.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *IJCV*, 115(3): 211–252.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv*, 1606.04671.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *CVPR*, 618–626.

Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay. In *NIPS*, 2990–2999.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *NIPS*, 4077–4087.

Tao, X.; Chang, X.; Hong, X.; Wei, X.; and Gong, Y. 2020. Topology-Preserving Class-Incremental Learning. In *ECCV*, 254–270.

Wang, F.-Y.; Zhou, D.-W.; Ye, H.-J.; and Zhan, D.-C. 2022. FOSTER: Feature Boosting and Compression for Class-Incremental Learning. In *ECCV*.

Xu, J.; and Zhu, Z. 2018. Reinforced continual learning. In *NeurIPS*, 899–908.

Yan, S.; Xie, J.; and He, X. 2021. DER: Dynamically Expandable Representation for Class Incremental Learning. In *CVPR*, 3014–3023.