Regularize, Expand and Compress: NonExpansive Continual Learning

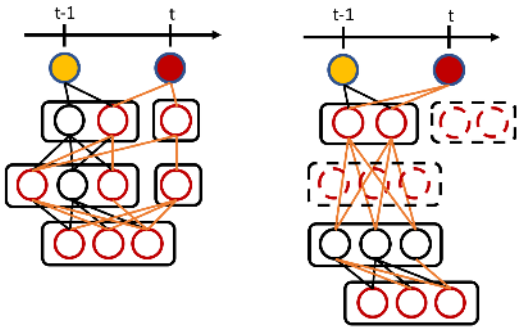
Jie Zhang[1], Junting Zhang[2], Shalini Ghosh[3], Dawei Li[3], Jingwen Zhu[3], Heming Zhang[2], Yalin Wang[1*]
[1] Arizona State University, Tempe, AZ, USA; [2] University of Southern California,
Los Angeles, CA, USA; [3] Samsung Research America, Mountain View, CA, USA

Abstract

Continual learning (CL), the problem of lifelong learning where tasks arrive in sequence, has attracted increasing attention in the computer vision community lately. The goal of CL is to learn new tasks while maintaining the performance on the previously learned tasks. There are two major obstacles for CL of deep neural networks: catastrophic forgetting and limited model capacity. Inspired by the recent breakthroughs in automatically learning good neural network architectures, we develop a nonexpansive AutoML framework for CL termed Regularize, Expand and Compress (REC) to solve the above issues. REC is a unified framework with three highlights: 1) a novel regularized weight consolidation (RWC) algorithm to avoid forgetting, where accessing the data seen in the previously learned tasks is not required; 2) an automatic neural architecture search (AutoML) engine to expand the network to increase model capability; 3) smart compression of the expanded model after a new task is learned to improve the model efficiency. The experimental results on four different image recognition datasets demonstrate the superior performance of the proposed REC *over other CL algorithms.*

1. Introduction

In many real-world applications, batches of data arrive periodically (e.g., daily, weekly, or monthly) with the data distribution changing over time. This presents a challenge for continual learning (CL), and is an important topic of study in machine learning. The primary goal of continual learning is to learn consecutive tasks without forgetting the knowledge learned from earlier tasks, and leverage the previous knowledge to obtain better performance or faster convergence on the new tasks. One naive way is to fine-tune the model for every new task; however, such retraining typically degenerates the model performance on both new tasks and the old ones. If the new tasks are greatly different from the old ones, we might not be able to obtain the optimal model for the new tasks. Meanwhile, the retraining may adversely affect the old tasks, causing them to drift from their optimal solution. This is known as "catastrophic forgetting", a phenomenon where training a model with new tasks interferes the previously learned old knowledge, leading to a performance degradation or even overwriting of the old knowledge by the new one.



(a) Partial retraining w/ expansion (b) Partial expansion w/ compression

Figure 1. (a) The previous state-of-the-art CL method, DEN [30], selectively retrains the old network, and dynamically expands the model capacity. (b) The proposed REC method expands the network through network transformation based AutoML, and then subsequently compresses the model back to its original size.

To overcome above catastrophic forgetting problem, many approaches have been proposed [15, 19, 23, 32, 33]. Kirkpatrick *et al*. [15] propose using a regularization term to prevent the new weights from deviating too much from the previously learned weights, based on their significance to old tasks. Their method uses a fixed neural network architecture, which would not scale up when network capacity gets saturated with more and more new tasks to learn. Dynamically expanding the network [30] (DEN) is one way to overcome the problem caused by static architecture — it expands the network capacity whenever it detects that the loss for the new task would not reach a pre-defined threshold. However, DEN involves many hyperparameters and the final performance is highly sensitive to these parameters; it relies on hand-crafted heuristics to explore the tuning space. This search space can be considerably large, and human experts usually find a sub-optimal solution in a time-

*We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

consuming parameters tuning process. To this end, we aim to automatically expand the network for CL, with better performance and less parameter redundancy than human-designed architectures. To better facilitate automatic knowledge transfer without human expert tuning and model design with optimized model complexity, we unprecedentedly propose a regularized nonexpansive CL framework while taking learning efficiency into consideration.

AutoML refers to automatically learn a suitable machine learning (ML) model for a given task — Neural Architecture Search (NAS) [34] is a subfield of AutoML for deep learning, which searches for optimal hyperparameters of designing a network architecture using reinforcement learning (RL). The RL framework has a main controller that observes the generated children networks' performance on the validation set as the reward signal — it then assigns a high probability to the architecture candidate that have high validation accuracy to update the model. If we use this approach directly in the continual learning setting, it would forget old tasks' knowledge, and it would be a wasteful process since each new task network architecture has to be searched from scratch by the controller, ignoring the correlations between previously learned tasks and the new task. We hereby propose a regularized weight consolidation (RWC) approach to obtain an effective classifier by exploiting inherent correlations between old tasks and new task. Furthermore, to narrow down the architecture search space and save time, network transformation [4] is utilized to accelerate meta-learning of the new network.

However, if we keep expanding the network for more and more new tasks, the model size will grow drastically to violate piratical efficiency requirements (e.g., low memory footprint, low power usage). Many network-expansion-based continual learning algorithms [24, 30] increase the model capability but also decrease the learning efficiency in terms of memory cost and power usage. Therefore, we conduct model compression after completing the learning of each new task — we compress the expanded model to the initial model size (before network expansion), with negligible performance loss on both old and new tasks. Fig 1 illustrates the main difference of our approach with network-expansion-based continual learning algorithms.

In this work, we focus on 1) overcoming catastrophic forgetting for CL and 2) improving the network capacity without decreasing learning efficiency. We propose a new sparse group *regularized* weight consolidation (RWC), to address the first problem. Compared to previous works, *e.g*. EWC [15], RWC can identify and retrain discriminative subset of parameters by incorporating inherent correlations among multiple learned new tasks and extract more meaningful features from old tasks, while EWC only considers the previous tasks' Fisher Information. The experimental results show RWC achieves higher average per-task accuracy compared to EWC, especially later tasks. To address the second problem, we aim to automatically *expand* the network for CL with high performance and optimized model complexity without human expert tuning. We therefore consider the newly expanded layer as a new task-specific layer, where l_1 regularization is adopted to promote sparsity for the new weight so that each neuron only connects with few neurons in the following layer. This will efficiently learn a discriminative representation for the new task while reducing the computation overheads. We then *compress* the expanded model to the same model size as the initial model, with negligible performance loss on both old and new tasks. This is different from previous network-expansion-based CL algorithms, e.g., DEN [30] and PGN [24], which reduce the model efficiency after learning new tasks. As far as we know, this is the first regularization-based nonexpansive AutoML algorithm for CL.

The key contributions of this work can be summarized as follows:

1. We propose to Regularize, Expand and Compress (REC) for CL, which automatically expands the network capacity for continuous learning a new task with fewer parameters than human-designed architectures. The final model is a non-expensive model but the performance is significantly enhanced by network expanding procedure.

2. To overcome the catastrophic forgetting of the previously learned tasks, we propose Regularized Weight Consolidation (RWC) — it identifies and retrains the discriminative subset of weights by exploiting inherent correlations among the tasks and trains the newly added layer as a task-specific layer for the new task.

3. Furthermore, REC applies an economical and efficient network transformation on arrival of the new task, which is advantageous over traditional AutoML frameworks, which discards the trained network and searching the architecture from scratch.

2. Related Work

2.1. Overcoming Catastrophic Forgetting

Recently, a lot of lifelong learning methods were proposed to address the catastrophic forgetting problem. The first group of methods uses regularized learning. Elastic Weight Consolidation (EWC) [15] shows that task-specific synaptic consolidation may overcome catastrophic forgetting in neural networks and observes the important weights for the previous tasks and selectively adjusts the plasticity of the weights. Inspired by EWC, Schwarz *et al*. [26] propose online EWC, which enlarges the EWC scalability by limiting the regularization term computational cost when the number of tasks increases. Synaptic Intelligence [31] computes an online importance measure along an entire learn-

Table 1. Comparisons of the lifelong learning approaches for overcoming catastrophic forgetting. EWC: Elastic Weight Consolidation [15]; DEN: Dynamically expandable network [30]; LwF: Learning without forgetting [19]; GEM: Gradient of Episodic Memory [21]; PGN: Progressive neural network [24] and our algorithm REC.

	EWC	DEN	LwF	GEM	PGN	REC
No memory growth	✓		✓	✓		✓
No exemplar	✓	✓	✓		✓	✓
Expanding network capacity when necessary		✓			✓	✓
AutoML ability						✓

ing trajectory, which is similar to EWC. Rotate-EWC [20] (REWC) is a modified version of EWC — it approximately diagonalizes the Fisher information matrix of the network parameters that compute the factorized rotation of the parameter space used in conjunction with EWC.

The second group of the strategies is associated with learning task-specific parameters. Learning without forgetting (LwF) [19] leverages distillation regularization on the new tasks — the soft labels of previously learned tasks are enforced to be similar to the network with the current task by using knowledge distillation [11]. Less-forgetful learning [14] is proposed to regularize the L_2 distance between the final hidden activations and the old tasks' parameters for preserving the old task feature mappings.

The third group of methods expands the network capacity. Progressive neural network (PGN) [24] is proposed to block any changes to the pre-trained network models on previously learned tasks and expands the network architecture by allocating sub-networks with the fixed capacity to be trained with the new information. PathNet [7] uses agents embedding into a neural network to find which parts of the network can be reused for learning new tasks and freezes task-relevant paths for avoiding catastrophic forgetting. Dynamically expanding network (DEN) [30] increases the number of trainable parameters to continually learn new tasks and dynamically selects neurons to retrain or expand neuron capacity by using group sparse regularization.

The other family of the methods uses episodic memory, where the previously learned task samples are stored to effectively recall the experience in the past. Gradient of Episodic Memory (GEM) [21] performs positive forward transfer, minimizes negative backward transfer to previously learned tasks and learns the subset of correlations to a set of tasks without using task descriptors. Incremental Classifier and Representation Learning (iCaRL) [23] combines classification loss on new tasks and distillation loss on previously learned tasks with a K-nearest neighbor classifier and selects the exemplars for each task by letting the embeddings of the selected samples closer to the center point of each class. Table 1 shows the multiple merits of REC, comparing with previous researches in this area.

2.2. AutoML and Knowledge Distillation

There are many works on AutoML to improve the performance of deep neural networks [34, 22, 3]. Neural Architecture Search (NAS) [34] searches the transferable network blocks via reinforcement learning and outperforms many manually designed network architecture. ENAS [22] uses a controller to discover network architectures by searching an optimal subgraph within a large computational graph and shares parameters among child models to enable efficient NAS. EAS [3] efficiently explores network architecture via network transformation [4] which is a functionality preserving method to expand the architecture with a fixed number of units or filters.

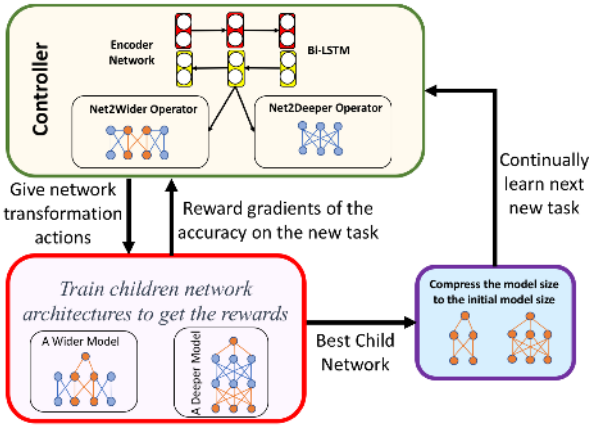


Figure 2. Illustration of our CL framework. REC first searches the best child network by RWC with Net2Deeper and Net2Wider operators in the controller for a new coming task, then compresses the expanded network to the same size as the initial model and continually learns next new task.

Besides, Knowledge distillation (KD) [11] is also very related to our work. KD is widely used to compress a network with a different architecture that approximates the original network where knowledge is transferred from a large teacher network to a small student network. The student network is trained with KD loss –a modified cross-entropy loss– that ensures the teacher network and student network are similar. In our work, we adopt the KD to compress the expanded network after learning each new task.

3. Method

Fig. 2 is an overview of our NonExpensive AutoML framework REC for CL with three components.

3.1. Problem Definition and Overview

We define the continual learning problem as follows — there will be an unknown number of tasks with un-

known distributions, arriving in sequence. Our goal is to learn a deep model in such a continual learning scenario without catastrophic forgetting. For the evaluation protocol, we report the classification accuracy on each of previous $T-1$ tasks and the current task T after training on the T-th task. Given a sequence of T tasks, task at time point $t = 1, 2, \cdots, T$ with N_t images comes with dataset $\mathbf{D}_t = \{\mathbf{x}_i^t, y_i^t\}_{i=1}^{N_t}$. Specifically, for task t, $y_i^t \in \{1, ..., K\}$ is the label for the i-th sample $\mathbf{x}_i^t \in \mathbb{R}^{d_t}$ in task t. We denote the training data matrix by $\mathbf{X}^t$ for $\mathbf{D}_t$, i.e., $\mathbf{X}^t = (\mathbf{x}_1^t, \cdots, \mathbf{x}_{N_t}^t)$. When the dataset of task t comes, all the previous training datasets $\mathbf{D}_1, \cdots, \mathbf{D}_{t-1}$ are not available any more, but the deep model parameter $\theta^{t-1} = \{\theta_l^{t-1}\}_{l=1}^{L}$ can be accessed. The continual learning problem at time point t when given data $\mathbf{D}_t$ can be defined as solving the following problem:

$$\min_{\theta_t} \mathcal{F}(\theta_t | \theta_{t-1}, \mathbf{D}_t), t = 1, \cdots, T \tag{1}$$

where $\mathcal{F}$ is the loss function of solving θ^t, θ^t is the parameter for task t.

Kirkpatrick et al. [15] proposed EWC that consists of a quadratic penalty on the difference between the parameter θ^t and θ^{t-1} to slow down the catastrophic forgetting for previously learned tasks. The posterior distribution $p(\theta^t|\mathbf{D}_t)$ is used to describe the problem by the Bayes' rule.

$$\log p(\theta^t|\mathbf{D}_t) = \log p(\mathbf{D}_t|\theta^t) + \log p(\theta^t|\mathbf{D}_{t-1}) - \log p(\mathbf{D}_t), \tag{2}$$

where the posterior probability $\log p(\theta^t|\mathbf{D}_{t-1})$ embeds all the information from task $t-1$. However, the problem (2) is intractable so that EWC approximates it as a Gaussian distribution with mean of parameter $\bar{\theta}^{t-1}$ and a diagonal I of the Fisher Information matrix $\mathbb{F}$. The matrix $\mathbb{F}$ is computed by $\mathbb{F}_i = I(\theta^t)_{ii} = E_x[(\frac{\partial}{\partial \theta_i^t} \log p(\mathbf{D}_t|\theta^t))^2 | \theta^t]$. Therefore, the problem of EWC on task t can be written as follows:

$$\min_{\theta^t} \quad \mathcal{F}_t(\theta^t) + \frac{\lambda}{2} \sum_i \mathbb{F}_i(\theta_i^t - \bar{\theta}_i^{t-1})^2, \tag{3}$$

where $\mathcal{F}_t$ is the loss function for task t, λ denotes how important the task $t-1$ is compared to the task t and i labels each weight of the parameter θ.

3.2. Regularized Weight Consolidation

The main problem of EWC is that EWC only enforces task t close to task $t-1$, but ignores the inherent correlations between task $t-1$ and task t and such relationship might potentially help overcome catastrophic forgetting on the task $t-1$. Learning multiple related tasks jointly can improve performance relative to learning each task separately, when the tasks are related — this idea is incorporated into Multi-Task Learning (MTL) [6]. It has been commonly

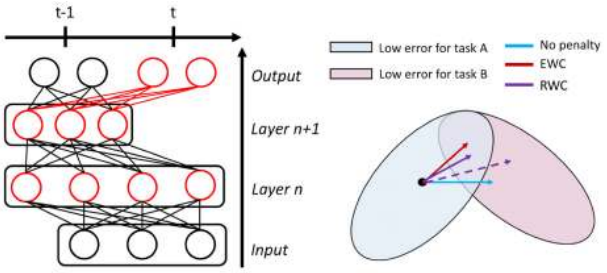


Figure 3. RWC retrains the entire network learned on previous tasks while regularizing it to prevent forgetting from the original model. RWC (purple solid line) learns better parameter representations to overcome catastrophic forgetting by studying MTL with the sparsity-inducing norm (purple dash line) and EWC (red line).

used to obtain better generalization performance than learning each task individually. We regularized Eq. 3 via MTL and propose a new objective function Eq. 4 to overcome catastrophic forgetting from multiple tasks simultaneously:

$$\min_{\theta^t} \mathcal{F}_t(\theta^t) + \frac{\lambda}{2} \sum_i \mathbb{F}_i(\theta_i^t - \bar{\theta}_i^{t-1})^2 + \lambda_2 |||[\theta^t; \theta^{t-1}]|||_{2,1}, \tag{4}$$

where λ_2 is the non-negative regularization parameter and $|||[\theta^t; \theta^{t-1}]|||_{2,1} = ||||\theta^t||_2, ||\theta^{t-1}||_2||_1$ is the $l_{2,1}$-norm regularization to learn the related representations and capture the common subset of relevant parameters from each layer for task $t-1$ and task t.

Specifically, we further consider some important parameters which have better representation power to a subset of tasks. The sparsity-inducing norm [8] has been studied in this paper to select such discriminative parameter subset by incorporating inherent correlations among multiple tasks. To this end, the l_1 sparse norm is imposed to learn the new task-specific parameters while learning task relatedness among multiple tasks. Therefore, the objective function for task t becomes:

$$\begin{aligned} \min_{\theta^t} \quad & \mathcal{F}_t(\theta^t) + \frac{\lambda}{2} \sum_i \mathbb{F}_i(\theta_i^t - \bar{\theta}_i^{t-1})^2 \\ & + \lambda_2 |||[\theta^t; \theta^{t-1}]|||_{2,1} + \lambda_3 ||\theta^t||_1, \end{aligned} \tag{5}$$

where λ_3 is the non-negative regularization parameter. We call our algorithm Regularized Weight Consolidation (RWC) and Fig. 3 shows the geometric illustration of RWC.

3.3. NonExpansive Continual Learning

RWC is a regularization-based CL, it might be needed to expand the network if the task is very different from the existing ones or the network capacity is not sufficient when more and more newly coming tasks. Due to human experts usually find a sub-optimal solution, this encourages us to propose AutoML based network expanding method for CL to find a global optimal solution. We name it Regularize, Expand, Compress (REC) and summarize the steps in Al-

Algorithm 1: REC

Input : Dataset $\mathbf{D}_1, \cdots, \mathbf{D}_T, \lambda, \lambda_1, \lambda_2$
Output: θ_c^T

```
begin
    for t = 1 → T do
        if t = 1 then
            Train an initial network with weights θ^1
             by using Eq. 1.
        else
            Search a best child network θ^t by Alg. 2
             with Eq. 8.
            Compress θ^t to the same model size as θ^1
             by Eq. 10 and use θ_c^t for next task.
```

gorithm 1 and the details of expanding network are outlined in Algorithm 2.

We consider net2wider and net2deeper operators [4] to expand the network capacity. The net2wider network transformation function is as follows:

$$\pi_{wider}(j) = \begin{cases} j & j \leq O_l, \\ random\ sample\ from\ \{1, ..., O_l\} & j > O_l, \end{cases} \tag{6}$$

where O_l represents the outputs of the original layer l. And the net2deeper network transformation function is

$$\gamma(\pi_{deeper}(j)) = \gamma(j) \quad \forall j. \tag{7}$$

where the constraint γ holds for the rectified linear activation. We learn a meta-controller to generate network transformation actions (Eq. 6 and Eq. 7) when given the initial network architecture. Specifically, we use an encoder network [3], which is implemented with an input embedding layer and a bidirectional recurrent neural network [25], to learn a low-dimensional representation of the initial network and be embedded into different operators to generate different network transformation actions. Besides, we use a shared sigmoid classifier to make the Net2Wider decision according to the hidden state of the layer learned by the bidirectional encoder network [3] and the wider network can be further combined with a Net2Deeper operator.

We integrate RWC (Eq. 5) into the AutoML framework as the loss function for CL settings. After we learning the network θ^{t-1} on the data $\mathbf{D}^{t-1}$, we will automatically search the best child network θ^t for task t among all the generated children networks $\theta_1^t, ..., \theta_m^t$ (m is the number of children networks). The network expansion will be finished by Net2wider and Net2Deeper operators when it is necessary to expand the network. If the controller decides to expand the network, the newly added layer will not have the previous tasks' Fisher Information. We consider the newly added layer as a new task-specific layer, l_1 regularization is adopted to promote sparsity in the new weight so that each neuron only connected with few neurons in the layer below. This will efficiently learn the best representation for the new task while reducing the computation overheads. The modified RWC in the network expansion scenario as follows:

$$\begin{aligned} \min_{\theta^t} \quad & \mathcal{F}_t(\theta^t) + \frac{\lambda}{2} \sum_{\substack{i \neq deeper \\ i \neq wider}} \mathbb{F}_i(\theta_i^t - \bar{\theta}_i^{t-1})^2 \\ & + \lambda_2 ||[\theta^t; \theta^{t-1}]||_{2,1} + \lambda_3 ||\theta^t_{\substack{i=deeper \\ i=wider}}||_1, \end{aligned} \tag{8}$$

where the subscript $deeper$ and $wider$ refer to the newly added layer in task t.

After the controller generates the child network, the child network will achieve an accuracy A_{val} on the validation set of task t and this will be used as the reward signal R^t to update the controller. We maximize the expected reward to find the optimal child network. The empirical approximation of our AutoML REINFORCE rule [28] as follows:

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{s=1}^{S} \nabla_C \log P(a_s | a_1, \cdots, a_{s-1}; C) R_i^t, \tag{9}$$

where m is the number of children networks that the controller C samples and a_s and g_s represents the action and state of predicting s-th hyperparameter to design a child network architecture, respectively. In Alg. 2, T is the transition function. Since R^t is non-differentiable, we use policy gradient to update the controller. We use a non-linear transformation $tan(A_{val} \times \pi/2)$ on validation set of task t as done in [3] and use the transformed value as the reward. We also use an exponential moving average of previous rewards with a decay of 0.95 to reduce the variance. To balance the old task and new task knowledge, we set maximum expanding layers are 2 and 3 on net2wider and net2deeper operators, respectively.

If the network keeps expanding as more and more tasks will be given, the model will suffer the inefficient problem and have extra memory cost. Thus, the model compression technique is needed to reduce the memory cost and receive a nonexpansive model. Here, we use soft-label (the logits) as knowledge distillation (KD) [11] instead of the hard labels to train the student model. To be noticed, the θ^t has learned the knowledge of new task t and old tasks $1, \cdots, t-1$. The compressed model θ_c^t will have the similar performance as θ^t and it is not really necessary learning the parameter of θ^{t-1} again. We follow Ba et al. [2] that the student model is trained to minimize the mean of the l_2 loss on the training data $\{\mathbf{x}_i^t, z_i^t\}_{i=1}^{N^t}$, where z_i^t is the logits of the child model θ^t i-th training sample. We compress the θ^t to the same size model as θ^1 as long as we expand the network, the KD loss is listed below:

$$\min_{\theta_c^t} \mathcal{F}_{kd}(f(\mathbf{x}^t; \theta_c^t), \mathbf{z}^t) = \frac{1}{N^t} \sum_i ||f(\mathbf{x}_i^t; \theta_c^t) - z_i^t||_2^2, \tag{10}$$

Algorithm 2: Automatically Network Transformation

Input : Dataset $\mathbf{D}_t$, θ^{t-1}
Output: The best expended model θ^t

begin
  for $i = 1 \rightarrow m$ **do**
    for $s = 1 \rightarrow S$ **do**
      $a_s \leftarrow \pi_{deeper}(g_{s-1}; \theta^{t-1}_{deeper})$ or $\pi_{wider}(g_{s-1}; \theta^{t-1}_{wider})$
      $g_s \leftarrow T(g_{s-1}, a_s)$
      $\theta^t \leftarrow \theta^t_{newLayer}$
    $R_i \leftarrow tanh(A^t_i(g_S) \times \pi/2)$
    $\theta^t_i \leftarrow \bigtriangledown_{\theta^t_{i-1}} J(\theta^t_{i-1})$

where θ^t_c is the weights of the student network and $f(\mathbf{x}^t_i; \theta^t_c)$ is the prediction of task t i-th training sample.

The final student network θ^t_c is trained to convergence with hard and soft labels by the following loss function:

$$\min_{\theta^t_c} \mathcal{F}(f(\mathbf{x}^t; \theta^t_c), \mathbf{y}^t) + \mathcal{F}_{kd}(f(\mathbf{x}^t; \theta^t_c), \mathbf{z}^t), \qquad (11)$$

where $\mathcal{F}$ is the loss function (cross-entropy) for training with ground truth $\mathbf{y}^t$ of task t.

4. Experiments

4.1. Experimental Settings

Datasets. We evaluate our algorithm on most commonly used datasets for CL. We list them as follows:

– MNIST-permutation: MNIST [18] is used as the most common datasets among all lifelong learning works, which consists of ten handwritten digits classes with 60,000/10,000 training and testing examples. One way to create the datasets for multiple tasks is randomly permuting the pixels by a fixed permutation [15] so that the input distribution for each task is unrelated.

– MNIST-Variation: MNIST-variation [18] dataset rotates the MNIST dataset by a fixed angle between 0 to 180 degrees for each different task. We use $180/T$ as the fixed angle to create T tasks.

– CIFAR-100: CIFAR-100 [16] dataset contains 60,000 32×32 color images in 100 object classes. Each class has 500/100 images for training and testing. We consider each task with a set of classes, it contains $100/T$ classes when there are T tasks. Different from MNIST-permutation dataset, the input distributions are similar for all tasks but the output distributions for each task are different.

– CUB-200: CUB-200 [29] is a fine-grained image classification benchmark, we use CUB-200-2011 version in this work. It contains 11,788 images of 200 types of birds with 5,994/5,794 for training and testing. Each image has detailed annotations and a bounding box. We crop the bounding boxes from the original images and resize them to 224×224. We use the same way to create multiple tasks as CIFAR-100 dataset.

For the first three datasets, we choose $T = 10$ tasks. Since the fine-grained CUB-200 dataset is more challenging than others, we set $T = 4$ tasks to show better comparisons on lifelong learning. For all datasets, we use 0.1 ratio to split validation set and the model observes the tasks in sequence. We generate multiple tasks for each dataset first and all comparison methods then use the same task order and the same categories within the task for fair comparisons.

Base network settings. For two MNIST datasets, we use a two-layer fully-connected neural network of 100-100 units with ReLU activations as our initial network. For CIFAR-100 dataset, we use a modified version of AlexNet [17] which has five convolutional layers (64-128-256-256-128 depth with 5×5 filter size), and three fully-connected layers (384-192-100 neurons at each layer) and the standard data augmentation is used in this dataset. For CUB-200 dataset, we use a pre-trained VGG-16 [27] model from ImageNet [5] and fine-tune it on the CUB-200 data for better initialization. We follow the setting of Liu et al. [20], which adds a global pooling layer after the final convolutional layer of the VGG-16. The fully-connected layers are changed to 512-512 and the size of the output layer is the number of classes in each task. All models and algorithms are implemented using Tensorflow [1] library.

Comparison methods. We compare our algorithm with six other methods: 1) SN: A single network trained across all tasks. 2) Net2Net [4]: Network expanding by Net2Net [4] on new task. 3) EWC [15]: A deep network trained with elastic weight consolidation. 4) Net2Net-EWC: Network expanding by Net2Net [4] with elastic weight consolidation [15] when learning new task. 5) DEN [30]: Dynamically expandable network. 6) REWC [20]: Rotate Elastic Weight Consolidation. 7) RWC: A deep network trained with regularized weight consolidation. 8)REC: Regularize, Expand and Compress.

Hyperparameter settings. All hyper-parameters in RWC are optimized using a grid-search and the best results for each model are reported. For two MNIST datasets, the SGD optimizer is used with a learning rate of 0.001 and we set batch size of 256 with 8 epochs, $\lambda_1 = 2$, $\lambda_2 = 0.0001$ and $\lambda_3 = 0.001$ in all experiments. For CIFAR-100 dataset, we use SGD optimizer with momentum parameter of 0.9, learning rate of 0.01, batch size of 128 with 20 epochs, $\lambda_1 = 10$, $\lambda_2 = 0.015$ and $\lambda_3 = 0.0001$. For CUB dataset, the Adam optimizer is used with a learning rate of 0.001, batch size of 32 and 50 epochs, $\lambda_1 = 100$, $\lambda_2 = 0.001$ and $\lambda_3 = 0.005$. For network transformation based AutoML experimental settings, we followed the training details of Cai *et al*. [3].

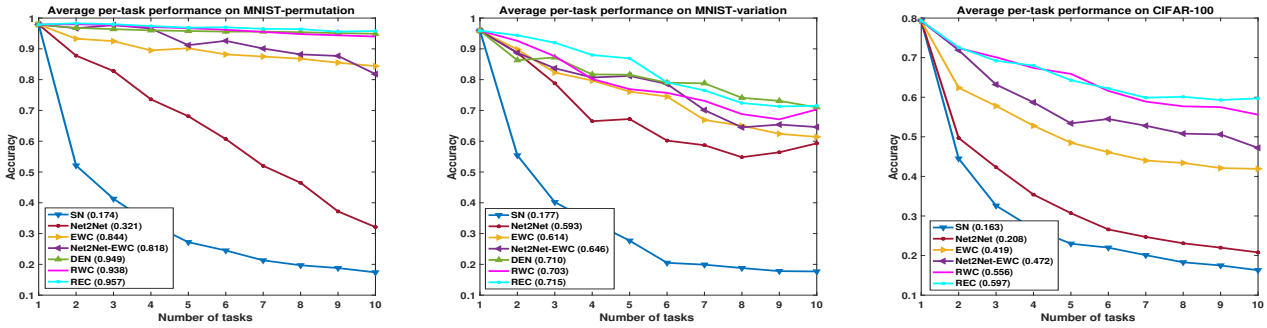


Figure 4. The experimental results of continual training on MNIST-permutation, MNIST-variation and CIFAR-100 datasets. We report the average per-task performance (Accuracy) of the models over $T = 10$ task. The numbers in the legend represent average per-task performance after the model has finished learning task t.

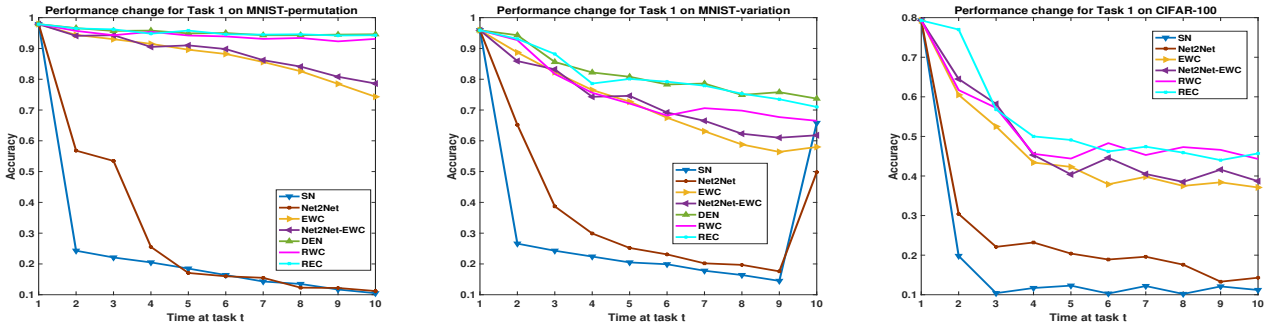


Figure 5. Forgetting experiment for task 1 on MNIST-permutation, MNIST-variation and CIFAR-100 datasets. We report the accuracy of different models on task $t = 1$ at each training stage to see how the model performance changes over time for all datasets.

Table 2. Comparisons of the model size and the average task accuracy after training 10 tasks on MNIST-permutation. $\#W(1)$: total parameters of task 1. $\#W(10)$: total parameters of task 10. ACC (10): average per-task accuracy after training task 10.

Methods	#W (1)	#W (10)	ACC (10)
SN	0.01M	0.01M	17.4%
Net2Net	0.01M	0.02M	32.1%
EWC	0.01M	0.01M	84.4%
Net2Net-EWC	0.01M	0.02M	81.8%
DEN	0.01M	0.14M	94.9%
RWC	0.01M	0.01M	93.8%
REC	**0.01M**	**0.01M**	**95.7%**

4.2. Experimental Results

We evaluate our methods from both model accuracy and model complexity, where we measure the model size at the end of the training process.

Comparisons of the model performance. We report the average per-task accuracy of MNIST-permutation, MNIST-variation and CIFAR-100 datasets when $T = 10$ in Fig. 4 and average the results over five runs. Overall, REC outperforms all comparison methods and overcomes catastrophic forgetting especially on the later tasks (after task 5). We can observe that the regularization based network (EWC, RWC) has worse performance than expandable networks (DEN, REC), which shows that selectively expand networks help improve the performance by a large margin. Specifically, REC performs better than DEN on two MNIST datasets and RWC performs similarly with DEN on MNIST-permutation dataset while using fewer parameters. We also observe that directly apply Net2Net [4] on lifelong learning does not perform well since it forgets the old tasks' knowledge as finetuning (SN), but adding EWC as the loss function can help enhance the old tasks' performance on Net2Net. REC has better performance than Net2Net-EWC, because we consider the new task-specific parameters and the discriminative common subset between the old tasks and the new one.

We also evaluate the catastrophic forgetting over time on the earliest task, Fig. 5 shows the test accuracy of the first task throughout the whole lifelong learning process on MNIST-permutation, MNIST-variation and CIFAR-100 datasets. It shows that our methods (RWC and REC) overcome forgetting on old tasks compared with all other methods on MNIST-permutation and CIFAR-100 datasets. It is worth noting that DEN performs slightly better than our method on task 1 after learning later tasks on MNIST-variation dataset due to they selectively expands network for the new task, it will give a bias towards to the earliest task. Our REC is a nonexpensive network and our overall average per-task performance is better than DEN, which shows that our method has better performance on later learned tasks and achieve a more balanced performance when learn-

Table 3. Comparisons of the model size and the average task accuracy after training 10 tasks on CIFAR-100 dataset. $\#W(1)$: total parameters of task 1. $\#W(10)$: total parameters of task 10. ACC (10): average per-task accuracy after training task 10.

Methods	#W (1)	#W (10)	ACC (10)
SN	4M	4M	16.3%
Net2Net	4M	6.3M	20.8%
EWC	4M	4M	41.9%
Net2Net-EWC	4M	7.4M	47.2%
RWC	4M	4M	55.6%
REC	**4M**	**4M**	**59.7%**

ing sequential tasks in the temporal dimension comparing with DEN. Besides, we have an interesting founding on MNIST-variation dataset, the SN and Net2Net has irregular performance on task 1 after learning task 10, it is due to the task 10 is the upside-down flipped image of task 1 and such flip gives benefit on some digits such as '1','0','8'. And SN and Net2Net forget too much task 1' knowledge after learning task 9, they only can keep the most recently learned task knowledge when they learn task 10 comparing with EWC, RWC and REC and this causes the irregular performance.

Comparisons of the model complexity. Table 2 and Table 3 report the comparisons of the model size and the average per-task performance after training $T = 10$ tasks of different approaches on MNIST-permutation and CIFAR-100 datasets, respectively. Overall, REC performs similarly or better than all other approaches with smaller model size. We observe that DEN performs better than RWC and worse than REC on MNIST-permutation dataset, but it has 1.4X network expansion comparing with ours. For CIFAR-100 dataset, We compute our AUROC after learning $T = 10$ tasks, REC can achieve 0.887 comparing with DEN (0.923), however, our model size is 50% of DEN's model. Besides, we notice that DEN involves 7 hyperparameters and very sensitive to them, we slightly change one of them from 10^{-3} to 10^{-2}, the result becomes 0.8907 on MNIST-permutation dataset. Our method only has three hyperparameters and it needs much less expert tuning comparing with DEN. Training times is a limitation of the current version of REC, since REC is a reinforcement learning based algorithm, a varies number of trails are needed and this results in more training time than other methods. We will improve the training efficiency of our work in the future. Besides, we did not consider complexity network structures (e.g. ResNet [10], DenseNet [13]), we will extend the current work to more network architectures in the future.

Results on CUB-200 dataset. Fig. 6 shows the comparison results when $T = 4$ on CUB-200 dataset with EWC [15] and REWC [20]. It shows that RWC has comparable results with REWC, RWC has better performance on task 3 and task 4 while has worse performance on task 2. We test REC with only new task validation set (REC-new), which has similar results as RWC on later tasks. This might be caused by using only new task validation set is not sufficient to compute the rewards on a more subtle dataset. We hypothesis the exemplars from old tasks will help improve the nonexpansive AutoML system's performance. Thus, we use the validation sets of all learned tasks to compute the rewards and report the results (REC-all) in Fig. 6. The results show that exemplars from old tasks help improve the performance of AutoML based algorithm and we will investigate the relationship between the number of exemplars and the performance of REC in our future work.

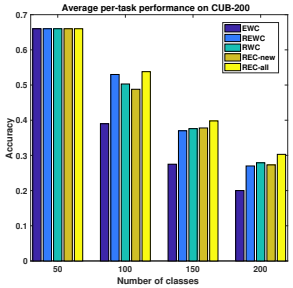


Figure 6. Comparison results with EWC and REWC on CUB-200 dataset when $T = 4$.

Table 4. Comparison results of average per-task accuracy after training task 10 on MNIST-permutation dataset.

Method	EWC	EWC+l_1	EWC+$l_{2,1}$	RWC
ACC(10)	84.4%	87.7%	88.5%	94.0%

Ablation study on each component in RWC. We study how the different components used in RWC affect the final performance of lifelong learning. We report the average per-task accuracy after training task 10 on MNIST-permutation of different strategies *EWC*, *EWC with l_1-norm only*, *EWC with l_2-norm only* and *RWC* in Table 4. It shows that $l_{2,1}$-norm has a stronger effect of the performance than l_1-norm while our method RWC outperforms the single regularization strategies, which demonstrates the meaningful and useful of our method by studying common weights subset with discriminative new task parameters.

5. Conclusions and Limitations

In this work, we develop a regularized continual learning framework via nonexpansive AutoML (REC). REC is achieved at two stages: continually network expansion and model compression. To overcome catastrophic forgetting, we propose RWC. We achieve better accuracy and smaller model size than other CL methods on four datasets.

Model compression is an optional stage for the current work with a trade-off between the compressed model and the original model. REC is our initial work for overcoming catastrophic forgetting and we will speed-up the hyperparameter optimization [12] in our future work. The AutoML training time is another limitation with REC, however it can be further improved by optimality tightening [9] or parallelization [34] or similar approaches for reducing the training time. We plan to reduce the training complexity in our future work.

References

[1] M. Abadi et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 6

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014. 5

[3] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. AAAI, 2018. 3, 5, 6

[4] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015. 2, 3, 5, 6, 7

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 6

[6] T. Evgeniou and M. Pontil. Regularized multi–task learning. In *Proc. SIGKDD*, pages 109–117. ACM, 2004. 4

[7] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 3

[8] P. Gong, J. Ye, and C.-s. Zhang. Multi-stage multi-task feature learning. In *Advances in neural information processing systems*, pages 1988–1996, 2012. 4

[9] F. S. He, Y. Liu, A. G. Schwing, and J. Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*, 2016. 8

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 8

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3, 5

[12] T. Hinz, N. Navarro-Guerrero, et al. Speeding up the hyperparameter optimization of deep convolutional neural networks. *International Journal of Computational Intelligence and Applications*, page 1850008, 2018. 8

[13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017. 8

[14] H. Jung, J. Ju, M. Jung, and J. Kim. Less-forgetful learning for domain expansion in deep neural networks. *arXiv preprint arXiv:1711.05959*, 2017. 3

[15] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017. 1, 2, 3, 4, 6, 8

[16] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 6

[18] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 6

[19] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 1, 3

[20] X. Liu, M. Masana, L. Herranz, J. Van de Weijer, A. M. Lopez, and A. D. Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. *arXiv preprint arXiv:1802.02950*, 2018. 3, 6, 8

[21] D. Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017. 3

[22] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 3

[23] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017. 1, 3

[24] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 2, 3

[25] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. 5

[26] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018. 2

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[28] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 5

[29] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 6

[30] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 1, 2, 3, 6

[31] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017. 2

[32] J. Zhang and Y. Wang. International conference on medical image computing and computer-assisted intervention. In *MICCAI*, pages 850–859. Springer, 2019. 1

[33] J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck, H. Zhang, and C.-C. J. Kuo. Class-incremental learning via deep model consolidation. In *Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2020. 1

[34] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2, 3, 8