Achieving a Better Stability-Plasticity Trade-off via Auxiliary Networks in Continual Learning (Appendix)

Sanghwan Kim Lorenzo Noci Antonio Orvieto Thomas Hofmann
ETH Zürich
Zürich, Switzerland
{sanghwan.kim, lorenzo.noci, antonio.orvieto, thomas.hofmann}@inf.ethz.ch

A. Motivation of Our Work: Using Auxiliary Network for Stability-Plasticity Balance

In the last few years, several papers [14, 15, 22, 24] have proposed to use an auxiliary network or an extra module which is solely trained on current dataset. They tried to combine this additional structure with a previous network or module which has been trained continually on old datasets.

Elastic Weight Consolidation (EWC) [12] is one of the initial works among weight regularization methods that regularize weights related to the previous tasks. After training each task, EWC copies and freezes the current network as an old network. Then, it estimates the importance of each parameter in the old network so that the weights with the high importance remain unchanged when the model is optimized on future task. Based on the EWC, [22] proposes *Active Forgetting with synaptic Expansion-Convergence* (AFEC) which further regularizes the weights relevant to the current task through a new set of parameters called expanded parameters. The expanded parameters are solely trained on the dataset of the new task initialized by the old network and are allowed to forget the previous tasks. As a result, AFEC can reduce potential negative transfer in continual learning by selectively merging the old parameters with the expanded parameters. The stability-plasticity balance in AFEC is adjusted via hyperparameters which scale the regularization terms for remembering the old tasks and learning the new tasks.

Learning without Forgetting (LwF) [13] prevents forgetting using knowledge distillation [2, 9]. They apply a distillation loss so that the model can learn soft targets generated by the old model instead of typical one-hot targets. The old model is copied and freezed before the training of the current task like EWC. *Deep Model Consolidation* (DMC) [24] is another distillation method built upon LwF. DMC proposes double distillation loss where the soft targets are generated using both the old model and a new model. The new model in DMC is basically the same concept as the expanded parameters in AFEC which is optimized on the current task. Then, the logits of new classes from the new model and the logits of old classes from the old model are concatenated to build the final soft targets. Then, the stability-plasticity balance is achieved by penalizing the model to generate the same output as the old model for old classes and the same output as the new model for new classes.

Adaptive Aggregation Networks (AANet) [15] explicitly expands ResNet [8] to have the two types of residual blocks at each residual level: the one for retaining old knowledge and the other for learning new knowledge. The outputs from the two residual blocks are linearly combined by aggregation weights and then proceeded to the next-level layer. They train AANets through bilevel optimization. In the first level, the parameters of the two residual blocks are trained. In the second level, the aggregation weights are adapted which decide the balance between stability and plasticity within the ResNet.

Recent work by [19] observes that multitask and continual solutions are connected by very simple curves with a low error in weight space, which is called *Linear Mode Connectivity*. They empirically prove that this connectivity is a linear path if the multitask learning and the continual learning share same initialization weights. Based on this observation, [14] proposes a simple linear connector that linearly add the weights of two networks following this linear path to emulate the multitask solution: the one model remembering the old tasks and the other model learning the new tasks. The stability-plasticity balance is preserved by combining the two networks.

The above methods [14, 15, 22, 24] all share the property that the auxiliary model or module is used to solve the stability-plasticity dilemma in continual learning. Consequently, these methods are able to learn the current task better than the original method while still retaining the knowledge of the previous tasks. However, the underlying mechanism of the interaction between the previous model and the auxiliary model is not widely studied. Therefore, in this work, we first formalize the framework of continual learning that adopts the auxiliary network called *Auxiliary Network Continual Learning* (ANCL).

Given this environment, we investigate the stability-plasticity trade-off from both a theoretical and empirical point of view and perform various analyses to better understand it.

B. The Detail Explanation of Methods in Table 1 of Main Paper

B.1. Weight Regularization Method

The continual learning aims to learn sequentially T tasks using a neural network f_θ, where $\theta \in \mathbb{R}^P$ denotes the learnable weights. In the standard continual learning framework, when presented with task t, the user has an access to previous network weights $\theta^*_{1:t-1} \in \mathbb{R}^P$, which are the result of continual learning from task 1 to $t-1$. It is well known that simply starting optimization from the weights $\theta^*_{1:t-1}$ to obtain the weights $\theta^*_{1:t}$ using the new data from the task t results in catastrophic forgetting [18] of the old tasks. A standard way to mitigate catastrophic forgetting is to include a regularization term which binds the dynamics of each network parameter θ_i ($i \in 1, \ldots, P$) to the corresponding old network parameter $\theta^*_{1:t-1,i}$ through a regularization term $R_{1:t-1,i} > 0$. The new optimization problem on task t then returns:

$$\theta^*_{1:t} = \underset{\theta=(\theta_1,\ldots,\theta_P)}{\arg\min} \left[L_{\text{reg}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2} \sum_i R_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2 \right] \tag{1}$$

where in classification problems $\mathcal{L}_{\text{t}}(\theta)$ is cross-entropy loss on the data of the task t and λ is the regularization strength which is usually selected by a grid search procedure. $R_{1:t-1,i}$ is the accumulated regularizer of each parameter θ_i until task $t-1$ and various regularization-based methods choose different ways to estimate this parameter.

For example, *Elastic Weight Consolidation* (EWC) [12] calculates R_i through the approximation of Fisher Information Matrix (FIM). The diagonal elements of FIM quantifies how curved the likelihood of each parameter is and can be approximated by a Hessian matrix near the optimum. In other words, the larger the change in the gradient, the more relevant the corresponding parameter is to the previous tasks. FIM is calculated after training each task, which means that R_i of EWC cannot fully reflect the learning trajectory of each network weight.

Compared to EWC, *Memory Aware Synapses* (MAS) [1] proposes accumulating the changes of each parameter throughout the update history. R_i is measured through the magnitude of the updates on each parameter according to the change in output. In other words, if the small change of specific parameter in a network causes the huge change of an output, that parameter should be memorized first.

B.2. Knowledge Distillation Method

Distillation-based approaches prevent forgetting through knowledge distillation [2, 9] which was originally designed to train a more compact student network from a larger teacher network. In this way, the main network can emulate the activation or logit of the previous network while learning a new task.

Learning without Forgetting (LwF) [13] proposes the following loss on task t to retain the old knowledge:

$$\mathcal{L}_{\text{LwF}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \sum_{c=1}^{C_{1:t}} -y^c(x_j; \theta^*_{1:t-1}) \log y^c(x_j; \theta). \tag{2}$$

Similarly to Eq. (1), θ is the weights of the main network that has been trained on a sequence of tasks so far and $\mathcal{L}_{\text{t}}(\theta)$ is cross-entropy loss on current classfication task t. $y(x_j; \theta^*_{1:t-1})$ and $y(x_j; \theta)$ are the temperature-scaled logits of the old network and the current network, respectively. The main network that continuously learns data from task 1 to $t-1$ is freezed and saved as the old network. $C_{1:t}$ denotes the total number of classes until task t and thus, $y^c(x_j)$ refers to the c^{th} output of the logit by an input x_j from the current dataset D_t. Note that the second term in Eq. (2) is also calculated on the current dataset as the previous data is not available. By emulating the soft targets generated by the previous model in addition to the ground truths in one-hot vector, the model can preserve the internal neural connection of the previous model while the weights are optimized for the new task.

The logit scaled by temperature τ on the c^{th} class position is calculated as follows:

$$y^c(x_j; \theta) = \frac{(\boldsymbol{o}^c(x_j; \theta))^{1/\tau}}{\sum_k (\boldsymbol{o}^k(x_j; \theta))^{1/\tau}}, \qquad y^c(x_j; \theta^*_{1:t-1}) = \frac{(\boldsymbol{o}^c(x_j; \theta^*_{1:t-1}))^{1/\tau}}{\sum_k (\boldsymbol{o}^k(x_j; \theta^*_{1:t-1}))^{1/\tau}} \tag{3}$$

where $\boldsymbol{o}(x_j; \theta)$ and $\boldsymbol{o}(x_j; \theta^*_{1:t-1})$ are the outputs of the current network and the old network before softmax is applied. The higher temperature generates more evenly distributed soft targets. For example, if the temperature goes to infinity ($i.e.$ $\tau \to \infty$), $y(x_j)$ becomes an uniform vector ($i.e.$ $y^c(x_j) = 1/C_{1:t}$ for all $c \in \{1, ..., C_{1:t}\}$).

Another distillation method, *less-forgetting learning* (LFL) [11], penalizes the differences of activations before last layer:

$$\mathcal{L}_{\text{LFL}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \| f(x_j; \theta) - f(x_j; \theta^*_{1:t-1}) \|_2^2. \tag{4}$$

$f(x_j; \theta)$ and $f(x_j; \theta^*_{1:t-1})$ are the centered and normalized activations of the main and old network respectively generated by input x_j from the current dataset D_t . The idea of LFL is the same as LwF except that the logits are replaced by the activations. The knowledge of the old tasks is retained by minimizing the gap between the activations from the previous model and the current model.

B.3. Memory Replay Method

Replay-based methods keep a part of the previous data (or exemplars) in a memory buffer. The memory buffer should contain the same number of exemplars for each class to build a balanced dataset and the few exemplars of each class should well represent the general features of their class. Then, a model is trained on the current dataset combined with the previous exemplars in the memory buffer to prevent the forgetting of the previous tasks. *Incremental Classifier and Representation Learning* (iCaRL) [21] first proposes the usage of the memory buffer built on LwF [13]. iCaRL calculates the mean of feature representations for each class and selects exemplars iteratively in a way that the mean of exemplars is closest to the class mean in feature representation space. This sampling strategy is called *herding* and is well known to outperform a random sampling strategy at the cost of more computations. Moreover, iCaRL applies a nearest mean of exemplars classification instead of using a classifier layer, which classifies an image to the closest class mean of exemplar feature representations. Let D_t be the current dataset of task t and $P_{1:t-1}$ be the exemplar sets in the memory buffer which contains data from task 1 to $t-1$. Then, the loss of iCaRL returns:

$$\mathcal{L}_{\text{iCaRL}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \sum_{c=1}^{C_{1:t}} -y^c(x_j; \theta^*_{1:t-1}) \log y^c(x_j; \theta) \tag{5}$$

where the cross-entropy loss (the first term) and the distillation loss (the second term) are both calculated on the combined dataset $D_t^+ = D_t \cup P_{1:t-1}$. The rest of the notations are equal to Eq. (2).

B.4. Bias Correction Method

In memory replay methods, a network is trained on the combined dataset, a highly unbalanced dataset with few exemplars from the previous classes and sufficient data from the new classes. As a result, the network is biased towards the data of new classes which hold a large majority in the combined dataset. This problem is called *task-recency bias* and recent articles show that alleviating the bias can significantly improve performance.

For instance, [23] proposes *Bias Correction* (BiC) to prevent task-recency bias. They divide the combined dataset into a train set and a validation set and the model learns these sets continually through two-stage training. During the first stage, the model is trained on the train set with the loss in Eq. (5). In the second stage, they suggest the usage of a linear transformation on the logits $\boldsymbol{o}_k$ of new classes to compensate for the task-recency bias:

$$q_k = \begin{cases} \boldsymbol{o}_k, & \text{if } 1 \le c \le C_{1:t-1} \\ \alpha \boldsymbol{o}_k + \beta, & \text{if } C_{1:t-1} < c \le C_{1:t}. \end{cases} \tag{6}$$

where the output of the old classes c ($1 \le c \le C_{1:t-1}$) remains the same and the output of the new classes c ($C_{1:t-1} < c \le C_{1:t}$) is linearly transformed by the learnable parameters α and β. These parameters are optimized by the validation set, while all other parameters in the network are frozen.

Another bias correction method proposed by [10] is called *Learning a Unified Classifier Incrementally via Rebalancing* (LUCIR) to tackle three problems that induce task-recency bias. The first problem they point out is that the feature norm of the new classes is larger than the feature norm of the old classes. In order to reduce the difference, they apply cosine normalization layer which are invariant to the magnitude of the feature instead of typical softmax layer. Then, the predicted probability $p_c(x)$ of class c by input x is calculated as follows:

$$p_c(x) = \frac{\exp\left(\theta_{L,c}^T f(x;\theta) + b_{L,c}\right)}{\sum_k \exp\left(\theta_{L,k}^T f(x;\theta) + b_{L,k}\right)} \longrightarrow p_c(x) = \frac{\exp\left(\eta \langle \bar{\theta}_{L,c}, \bar{f}(x;\theta) \rangle\right)}{\sum_k \exp\left(\eta \langle \bar{\theta}_{L,k}, \bar{f}(x;\theta) \rangle\right)} \tag{7}$$

where the left and right equations each represent the output probability of softmax layer and cosine normalization layer. f is the feature extractor and θ_L and b_L are the weights ($i.e.$ class embedding) and bias in the last layer L. $\bar{\theta}$ and $\bar{f}(x)$ denotes

l_2 normlaized vector ($\bar{v} = v/\|v\|_2$) and $\langle \bar{\theta}_{L,c}, \bar{f}(x) \rangle$ measures the cosine similarity between normalized weight and feature vector ($\langle \bar{v}_1, \bar{v}_2 \rangle = \bar{v}_1^T \bar{v}_2$). The learnable scalar η adjusts the peakness of softmax distribution since the range of $\langle \bar{\theta}_{L,c}, \bar{f}(x) \rangle$ is restricted to $[-1, 1]$.

The second problem is found in the distillation loss after applying cosine normalization in Eq. (7): the angle between the feature vector $f(x;\theta)$ and the class embedding $\theta_{L,c}$ can be optimized to become similar as the angle between the previous feature vector $f(x;\theta^*_{1:t-1})$ and the previous class embedding $\theta^*_{L,c}$ instead of being optimized to learn $f(x;\theta^*_{1:t-1})$ itself. Therefore, the authors suggest the usage of cosine embedding loss which directly regularizes the angle between the feature vector $f(x;\theta)$ and the old feature vector $f(x;\theta^*_{1:t-1})$:

$$\mathcal{L}_{\text{dis}}(x) = \sum_{c=1}^{C_{1:t}} \|\langle \bar{\theta}_{L,c}, \bar{f}(x;\theta) \rangle - \langle \bar{\theta}^*_{L,c}, \bar{f}(x;\theta^*_{1:t-1}) \rangle\|_2^2 \longrightarrow \mathcal{L}_{\text{dis}}(x) = 1 - \langle \bar{f}(x;\theta), \bar{f}(x;\theta^*_{1:t-1}) \rangle. \tag{8}$$

The equation on the left shows the previous distillation loss and the right one refers to the revised distillation loss.

The last problem addressed by [10] is inter-task confusion: the new class embeddings cluster together with the old class embeddings, which confuses the classification of the old and new classes. To prevent this, they employ *margin ranking loss*:

$$\mathcal{L}_{\text{mr}}(x) = \sum_{k=1}^{K} \max(m - \langle \bar{\theta}_L(x), \bar{f}(x;\theta) \rangle + \langle \bar{\theta}^k_L, \bar{f}(x;\theta) \rangle, 0) \tag{9}$$

where $\bar{\theta}_L(x)$ refers to the ground truth class embedding of x, θ^k_L is the embedding of top-K closest classes, and m is the margin threshold. This loss separate the current embeddings $\bar{\theta}_L(x)$ from the embeddings of K most similar class embedding $\bar{\theta}^k_L$. Combining all the solutions mentioned above, the loss of LUCIR finally returns:

$$\mathcal{L}_{\text{LUCIR}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \mathcal{L}_{\text{dis}}(x_j) + \lambda_{mr} \mathcal{L}_{\text{mr}}(x_j) \tag{10}$$

where cross-entropy loss and distillation loss are measured by the combined dataset D_t^+ and a margin ranking loss is calculated on the previous exemplars $P_{1:t-1}$. λ and λ_{mr} are hyperparameters found by grid search.

Built upon LUCIR, recent work [4] suggests *Pooled Outputs Distillation Network* (PODNet) which applies pooled out distillation loss and local similarity classifier. First, they define POD-width and POD-height losses as below:

$$\mathcal{L}_{\text{POD-width}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})) = \sum_{c=1}^{C} \sum_{h=1}^{H} \left\| \sum_{w=1}^{W} f_{l,c,w,h}(x;\theta) - \sum_{w=1}^{W} f_{l,c,w,h}(x;\theta^*_{1:t-1}) \right\|_2^2 \tag{11}$$

$$\mathcal{L}_{\text{POD-height}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})) = \sum_{c=1}^{C} \sum_{w=1}^{W} \left\| \sum_{h=1}^{H} f_{l,c,w,h}(x;\theta) - \sum_{h=1}^{H} f_{l,c,w,h}(x;\theta^*_{1:t-1}) \right\|_2^2 \tag{12}$$

where $f_l(x;\theta)$ and $f_l(x;\theta^*_{1:t-1})$ denotes the intermediate output of convolutional layer l by a sample x and both outputs are the representational matrix of size $C \times H \times W$. Thereafter, POD-width and POD-height losses are combined to build POD-spatial loss:

$$\mathcal{L}_{\text{POD-spatial}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})) = \mathcal{L}_{\text{POD-width}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})) + \mathcal{L}_{\text{POD-height}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})). \tag{13}$$

POD-flat loss is further defined as $\mathcal{L}_{\text{POD-flat}}(f_L(x;\theta), f_L(x;\theta^*_{1:t-1})) = \|f_L(x;\theta) - f_L(x;\theta^*_{1:t-1})\|_2^2$ with feature vector outputs of last convolutional layer L in the main network and the old network. At last, all POD losses are combined to build POD-final loss:

$$\mathcal{L}_{\text{POD-final}}(x) = \lambda_c \sum_{l=1}^{L-1} \mathcal{L}_{\text{POD-spatial}}(f_l(x;\theta), f_l(x;\theta^*_{1:t-1})) + \lambda_f \mathcal{L}_{\text{POD-flat}}(f_L(x;\theta), f_L(x;\theta^*_{1:t-1})) \tag{14}$$

where λ_c and λ_f adjusts the strength of two POD losses.

In addition, [4] indicates that the cosine normalization layer expressed in Eq. (7) optimizes a *global similarity*: the training objective increases the cosine similarity between feature vector and weights pushing all feature vectors toward a single proxy (or mode) [20]. To improve it, they design Local Similarity Classifier (LSC) considering the usage of multiple proxies, which

makes final embedding $f_L(x;\theta)$ robust to forgetting. In the setting of using K proxies, the similarity $s_{c,k}$ of kth proxy for each class c is first computed. Then, an averaged class similarity y_c becomes the output of the classification layer:

$$s_{c,k}(x) = \frac{\exp\langle\theta_{L,c,k}, f_L(x;\theta)\rangle}{\sum_i \exp\langle\theta_{L,c,i}, f_L(x;\theta)\rangle}, \qquad y_c(x) = \sum_{k=1}^{K} s_{c,k}(x)\langle\theta_{L,c,k}, f_L(x;\theta)\rangle. \tag{15}$$

[4] empirically found that NCA loss [7] converges faster than cross-entropy loss, thereby modifying it to implement the LSC loss with a small margin δ, a hinge $[\cdot]_+$, and a learnable parameter η:

$$\mathcal{L}_{\text{LSC}}(x) = \left[-\log \frac{\exp(\eta(y_c(x;\theta) - \delta))}{\sum_{i\neq c} \exp \eta y_i(x;\theta)}\right]_+. \tag{16}$$

Finally, we obtain the loss of PODNet:

$$\mathcal{L}_{\text{PODNET}} = \mathcal{L}_{\text{LSC}}(x_j) + \mathcal{L}_{\text{POD-final}}(x_j) \tag{17}$$

where the losses are calculated on the combined dataset D_t^+.

C. The Application of ANCL to Methods in Table 1 of Main Paper

In this section, we explain the loss function of ANCL applied to methods in Table 1. We first write down the loss of EWC and MAS on task t:

$$\mathcal{L}_{\text{EWC}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2}\sum_i F_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2, \tag{18}$$

$$\mathcal{L}_{\text{MAS}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2}\sum_i M_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2. \tag{19}$$

The notation of above losses are the same as Eq. (1) except that $F_{1:t-1}$ is the diagonal elements of Fisher Information Matrix (FIM) that has been accumulated until task $t-1$. If $F_{1:t-1}$ is replaced with the importance $M_{1:t-1}$ defined by MAS [1], it returns the loss of MAS in Eq. (19). Then, ANCL can be applied to EWC [12] and MAS [1] which generates *Auxiliary Network EWC* (A-EWC) and *Auxiliary Network MAS* (A-MAS) accordingly. The loss of A-EWC and A-MAS on task t is defined as follows:

$$\mathcal{L}_{\text{A-EWC}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2}\sum_i F_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2 + \frac{\lambda_a}{2}\sum_i F_{t,i}(\theta_i - \theta^*_{t,i})^2 \tag{20}$$

$$\mathcal{L}_{\text{A-MAS}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2}\sum_i M_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2 + \frac{\lambda_a}{2}\sum_i M_{t,i}(\theta_i - \theta^*_{t,i})^2 \tag{21}$$

where the importance F_t and M_t of the auxiliary parameters θ^*_t are calculated following the original methods (EWC or MAS) and λ and λ_a are fixed by grid search. The first two terms are equal to Eq. (1).

Similarly, ANCL can be extended to the distillation-based methods such as *Learning without Forgetting* (LwF) [13] in Eq. (2) or *less-forgetting learning* (LFL) [11] in Eq. (4). By applying ANCL to LwF and LFL, the new losses of *Auxiliary Network LwF* (A-LwF) and *Auxiliary Network LFL* (A-LFL) on task t are written as follows:

$$\mathcal{L}_{\text{A-LwF}} = \mathcal{L}_{\text{t}}(\theta) + \lambda\sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1,i}) \log y^c(x_j;\theta) + \lambda_a \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_t) \log y^c(x_j;\theta), \tag{22}$$

$$\mathcal{L}_{\text{A-LFL}} = \mathcal{L}_{\text{t}}(\theta) + \lambda\|f(x_j;\theta) - f(x_j;\theta^*_{1:t-1,i})\|_2^2 + \lambda_a\|f(x_j;\theta) - f(x_j;\theta^*_t)\|_2^2, \tag{23}$$

In Eq. (22), $y(x_j;\theta^*_t)$ represents the temperature-scaled logit of the auxiliary network and new regularization term is double summed over new class position c and data x_j of current task t. In Eq. (23), $f(x_j;\theta^*_t)$ is the normalized and centered activation of the auxiliary network. λ and λ_a are again found by grid search.

In the same way, we can apply ANCL to memory replay and bias correction approaches (iCaRL [21], BiC [23], LUCIR [10], PODNet [4]). These methods usually use the memory buffer of the previous data and thus losses are calculated on the combined dataset D_t^+(the current dataset D_t + the previous exemplars $P_{1:t-1}$).

First, we start from applying ANCL to the loss of iCaRL [21] in Eq. (5). Then, *Auxiliary Network iCaRL* (A-iCaRL), A-iCaRL returns the loss function as below:

$$\mathcal{L}_{\text{A-iCaRL}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1,i}) \log y^c(x_j;\theta) + \lambda_a \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_t) \log y^c(x_j;\theta) \tag{24}$$

where the first two terms are the same as iCaRL loss and the last term represents a new regularizer based on the auxiliary network. $y^c(x_j;\theta^*_t)$ denotes the temperature-scaled logit of the auxiliary network.

Similarly, we apply ANCL to BiC [23], so called *Auxiliary Network BiC* (A-BiC). A-BiC is also optimized by minimizing the loss function in Eq. (24). Note that BiC divides the combined dataset D_t^+ into a train set for main training stage and a validation set for bias correction stage. Two regularizers of A-BiC are measured on the train set like BiC.

Next, we adapt ANCL to the loss of LUCIR [10] in Eq. (10) to build the loss of *Auxiliary Network LUCIR* (A-LUCIR):

$$\mathcal{L}_{\text{A-LUCIR}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \mathcal{L}_{\text{dis}}(x_j) + \lambda_a \mathcal{L}^{aux}_{\text{dis}}(x_j) + \lambda_{mr} \mathcal{L}_{\text{mr}}(x_j) \tag{25}$$

where $\mathcal{L}^{aux}_{\text{dis}}(x) = 1 - \langle \bar{f}(x;\theta), \bar{f}(x;\theta^*_t) \rangle$ holds with the normalized feature vector $\bar{f}(x;\theta^*_t)$ of the auxiliary network.

Lastly, ANCL modifies PODNet loss (Eq. (17)) to define *Auxiliary Network PODNet* (A-PODNet) loss as follows:

$$\mathcal{L}_{\text{A-PODNET}} = \mathcal{L}_{\text{LSC}}(x_j) + \lambda \mathcal{L}_{\text{POD-final}}(x_j) + \lambda_a \mathcal{L}^{aux}_{\text{POD-final}}(x_j) \tag{26}$$

where $\mathcal{L}^{aux}_{\text{POD-final}}(x_j)$ stands for the POD loss with the feature representations of the auxiliary network as follows:

$$\mathcal{L}^{aux}_{\text{POD-final}}(x) = \lambda_c \sum_{l=1}^{L-1} \mathcal{L}_{\text{POD-spatial}}(f_l(x;\theta), f_l(x;\theta^*_t))) + \lambda_f \mathcal{L}_{\text{POD-flat}}(f_L(x;\theta)), f_L(x;\theta^*_t))) \tag{27}$$

Other than the above methods, ANCL can be adapted to most CL approaches with regularization terms based on the old parameters. However, it is hard to apply ANCL to the dynamic structure methods because these methods already contain an extra module or architecture to reflect plasticity.

D. Comparison with AFEC (Sec. 3.1 of Main Paper)

In this section, we compare our ANCL with AFEC [22] which is the most similar method as ours. Before that, we formally introduce *Active Forgetting with synaptic Expansion-Convergence* (AFEC) to clarify the difference easily. AFEC suggests to add an extra regularization term on EWC loss based on biologically inspired arguments. They argue that this term stimulates the active forgetting of previous knowledge that interferes with the new knowledge, whereas the existing regularization-based approaches highly concentrate on retaining the old weight. The loss of AFEC on task t can be expressed as follows:

$$\mathcal{L}_{\text{AFEC}} = \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{2} \sum_i F_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i})^2 + \frac{\lambda_a}{2} \sum_i F_{t,i}(\theta_i - \hat{\theta}_{t,i})^2. \tag{28}$$

The first two terms are the same as Eq. (18), while the last promotes active forgetting by regularizing the parameters $\theta \in \mathbb{R}^P$ towards the biologically inspired *expanded parameters* $\hat{\theta}_t \in \mathbb{R}^P$ (see precise definition in [22]) through FIM F_t on task t. The expanded parameters are solely trained on the current dataset allowing the forgetting of the old datasets and λ_a is a hyperparameter determined by grid search. As a result, the main network parameters θ efficiently learns from both the old parameters $\theta^*_{1:t-1}$ and the expanded parameters $\hat{\theta}_t$. Moreover, the last term of Eq. (28) can be applied to other regularization-based methods as a *plug-and-play* method. Thus, A-EWC is equivalent to AFEC[1] except that the importance F_t of the auxiliary network in ANCL is calculated only once before the training of the new task and then fixed afterward.[2]

The expanded parameters $\hat{\theta}_t$ in AFEC are computed as follows: at the beginning of each task, the expanded parameters are initialized by the old parameters, Then, at each epoch, it is trained on task t without regularization, returning the network parameters $\hat{\theta}_t$. For the current epoch of the original network, these weights are then taken into account to modify the regularized dynamics. This procedure is repeated every epoch, and hence for each epoch the dynamics of SGD is augmented with a freshly computed regularizer.

[1] However, A-MAS is not equal to AFEC as the new regularizer of ANCL is based on MAS importance not FIM.

[2] The code implementation of AFEC newly calculates F_t every epoch.

Methods	CL (original)	w/ AFEC [22]	w/ ANCL (ours)
EWC [12]	$58.13_{\pm 0.87}$	$60.60_{\pm 1.63}$	$60.86_{\pm 1.46}$
MAS [1]	$60.56_{\pm 0.82}$	$61.28_{\pm 0.81}$	$64.43_{\pm 1.17}$
LwF [13]	$78.87_{\pm 0.69}$	$78.77_{\pm 0.72}$	$79.42_{\pm 0.57}$
LFL [11]	$74.50_{\pm 0.57}$	$74.55_{\pm 0.62}$	$75.23_{\pm 0.67}$

Table 1. The averaged accuracy (%) on benchmark (1) CIFAR-100/10. Reported metrics are averaged over 3 runs (averaged accuracy $\pm$ standard error). AFEC [22] and ANCL (ours) are applied to CL approaches and compared.

Methods	CL (original)	w/ AFEC [22]	w/ ANCL (ours)
iCaRL [21]	$58.05_{\pm 0.94}$	$59.31_{\pm 0.97}$	$61.22_{\pm 0.88}$
BiC [23]	$56.74_{\pm 1.33}$	$57.08_{\pm 1.22}$	$58.32_{\pm 1.27}$
LUCIR [10]	$56.06_{\pm 0.45}$	$58.97_{\pm 0.92}$	$60.20_{\pm 0.78}$
PODNet [4]	$61.80_{\pm 0.77}$	$62.73_{\pm 0.68}$	$63.15_{\pm 0.62}$

Table 2. The averaged incremental accuracy (%) on benchmark (5) CIFAR-100/6. Reported metrics are averaged over 3 runs (averaged accuracy $\pm$ standard error). AFEC [22] and ANCL (ours) are applied to CL approaches and compared.

Finally, we evaluate ANCL and AFEC in Tab. 1 and Tab. 2. Tab. 1 demonstrates that ANCL outperforms AFEC in every method on task incremental scenario. Specifically, AFEC achieves similar improvement as ANCL for EWC while obtaining very marginal improvement for other methods. This results agree with our description above that EWC with AFEC and EWC with ANCL have the equivalent loss and the difference comes from how the new regularizer is calculated. Except EWC, ANCL can balance two regularizers better than AFEC which results in higher improvement in accuracy. In Tab. 2, we also compare AFEC and ANCL on class incremental scenario. Although AFEC gives some improvements, ANCL outperforms AFEC on all method. These results support that ANCL can effectively utilize the auxiliary network compared to AFEC which applies the fixed regularizer to all methods. In conclusion, the advantage of ANCL lies on its natural adaptation of the regularizer following the original methods such that the optimal solution well associate the old network with the auxiliary network.

E. The Mathematical Analysis of ANCL Gradients

In this section, we closely look into CL and ANCL by analyzing their gradients. For EWC [12] and MAS [1], we directly solve the optimal weights of CL and ANCL from their gradients and analyze them in respect of the stability-plasticity balance. For LwF [13], and LFL [11], we compare the gradient of CL and ANCL after some approximations. Then, we explain that other methods with memory buffer (iCaRL [21], BiC [23], LUCIR [10], and PODNet [4]) are originated from distillation-based method (LwF and LFL). Therefore, previous analysis on LwF and LFL can be extended and applied to their variations respectively.

First, we start with the EWC loss in Eq. (18). The gradient with respect to the i^{th} parameter θ_i of the model weights $\theta = (\theta_1, \dots, \theta_P) \in \mathbb{R}^P$ reads:

$$\nabla_{\theta_i} \mathcal{L}_{\text{EWC}} = \nabla_{\theta_i} \mathcal{L}_{\text{t}}(\theta) + \lambda F_{1:t-1,i}(\theta_i - \theta^*_{1:t-1,i}). \tag{29}$$

It consists of two terms: the first term updates θ_i toward the minima of the task-specific loss of the current task and the second term regularizes the model weight θ_i to be as close as possible to the old weight $\theta^*_{1:t-1,i}$ according to $\lambda F_{1:t-1,i}$. Then, the i^{th} model parameter is updated from the k^{th} to the $(k+1)^{th}$ iteration with learning rate η as follows:

$$\theta_i^{(k+1)} \leftarrow \theta_i^{(k)} - \eta \nabla_{\theta_i^{(k)}} \mathcal{L}_{\text{EWC}} \tag{30}$$

According to [16], the weight of the k^{th} iteration $\theta_i^{(k)}$ can be expressed by the initial weight $\theta^*_{1:t-1,i}$ via the recursive substitution of Eq. (30) during the k^{th} iteration:

$$\theta_i^{(k)} = \theta^*_{1:t-1,i} - \sum_{l=0}^{k-1} [(1 - \eta\lambda F_{1:t-1,i})^{(k-l-1)} \eta] g_i^{(l)}, \tag{31}$$

where $g_i^{(l)}$ denotes the gradient of task specific loss with respect to $\theta_i^{(l)}$ at the l^{th} iteration. If we take $k \to \infty$ on the both side of Eq. (31), we can obtain the optimal parameter $\theta^*_{1:t,i} = \lim_{k\to\infty} \theta_i^{(k)}$ on task t :

$$\theta^*_{1:t,i} = \underbrace{\theta^*_{1:t-1,i}}_{\text{Previous param.}} \underbrace{- \lim_{k\to\infty} \sum_{l=0}^{k-1} [(1 - \eta\lambda F_{1:t-1,i})^{(k-l-1)}\eta] g_i^{(l)}}_{\text{Task-specific updates}}. \tag{32}$$

Eq. (32) shows that the optimal weight $\theta^*_{1:t,i}$ after task t is obtained from the previous parameter $\theta^*_{1:t-1,i}$ and the sequence of task-specific updates $g_i^{(l)}$ which is adjusted by the effective learning rate $(1 - \eta\lambda F_{1:t-1,i})^{(k-l-1)}\eta$. Since the learning rate $\eta(> 0)$ and the importance of each parameter $F_{1:t-1,i}(> 0)$ are given from the beginning, the effective learning rate is fully depends on hyperparameter λ. For high λ, the weight updates are restricted as the value of $1 - \eta\lambda F_{1:t-1,i}$ becomes smaller which consequently reduces the effective learning rate. For low λ, the effective learning rate becomes approximately equal to η, which allows the free update of the weight on a new task. It is also empirically shown in [16] that when $(1-\eta\lambda F_{1:t-1,i}) > 1$ or $(1-\eta\lambda F_{1:t-1,i}) < 0$ holds, the training might become unstable because the effective learning rate will grow exponentially or change its sign every update.

Next, we analyze A-EWC loss in the same way. The gradient of Eq. (20) on i^{th} parameter θ_i shows:

$$\nabla_{\theta_i}\mathcal{L}_{\text{A-EWC}} = \nabla_{\theta_i}\mathcal{L}_{\text{EWC}} + \lambda_a F_{t,i}(\theta_i - \theta^*_{t,i}). \tag{33}$$

Similarly to EWC, θ_i of A-EWC is updated from the k^{th} to the $(k+1)^{th}$ iteration with learning rate η as follows:

$$\theta_i^{(k+1)} \leftarrow \theta_i^{(k)} - \eta\nabla_{\theta_i^{(k)}}\mathcal{L}_{\text{A-EWC}} \tag{34}$$

Then, we can extend the derivation of Eq. (31) following [16] to A-EWC through the recursive substitution of Eq. (34). Then, the updated weight $\theta_i^{(k)}$ at k^{th} iteration initialized with the previous optimal weight $\theta^*_{1:t-1,i}$ returns:

$$\theta_i^{(k)} = (1-\alpha-\beta)^k\theta^*_{1:t-1,i} + \sum_{l=0}^{k-1}(1-\alpha-\beta)^l(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \sum_{l=0}^{k-1}[(1-\alpha-\beta)^{(k-l-1)}\eta]g_i^{(l)} \tag{35}$$

where $\alpha = \eta\lambda F_{1:t-1,i}$ and $\beta = \eta\lambda_a F_{t,i}$ hold. The detailed derivation of Eq. (35) can be found in Appendix H.1.

Based on the experiment results of [16], one can similarly argue that $0 \le (1-\alpha-\beta) < 1$ is a sufficient condition for stable training with Eq. (35). Thus, we can safely assume that $\lambda > 0$ and $\lambda_a > 0$ are chosen appropriately where $(1-\alpha-\beta) < 0$ is not a case. Based on the assumption, we take $k \to \infty$ on Eq. (35). Then, the first term $(1-\alpha-\beta)^k\theta^*_{1:t-1,i}$ converges to zero as $\lim_{k\to\infty}(1-\alpha-\beta)^k = 0$ and the second term can be further calculated using the infinite sum of geometric sequence (*i.e.* $\sum_{n=0}^{\infty} ar^n = a/(1-r)$ for $|r| < 1$ and $a \neq 0$). Finally, the optimal parameter $\theta^*_{1:t,i} = \lim_{k\to\infty}\theta_i^{(k)}$ can be simply expressed as:

$$\theta^*_{1:t,i} = \underbrace{\frac{\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}}{\alpha+\beta}}_{\text{Interpolated param.}} \underbrace{- \lim_{k\to\infty} \sum_{l=0}^{k-1} [(1-\alpha-\beta)^{(k-l-1)}\eta] g_i^{(l)}}_{\text{Task-specific updates}}. \tag{36}$$

In Eq. (36), the optimal weight $\theta^*_{1:t,i}$ after task t is determined by the interpolation of the old parameter $\theta^*_{1:t-1,i}$ and the auxiliary parameter $\theta^*_{t,i}$ and the sequence of task-specific updates $g_i^{(l)}$ with the effective learning rate $(1-\alpha-\beta)^{(k-l-1)}\eta$. First, the interpolated parameter is determined by the relative ratio between α and β. If $\alpha > \beta$ is the case, the interpolation will lean toward the old parameter and if $\alpha < \beta$ holds, the interpolation will be located closer to the auxiliary parameter. Moreover, the effective learning rate now depends on both λ and λ_a. Thus, if λ and λ_a are both high, the effective learning rate will become smaller and consequently less task-specific updates will be made. As a result, the optimal weight will converge to the interpolation of two network parameters. Otherwise, the parameter will be simply optimized for the current task via task-specific updates. Compared to Eq. (32) where high λ interferes with learning a new task, the old parameter with high λ can still be updated toward the auxiliary parameter through the interpolation.

For MAS and A-MAS loss (Eq. (19) and Eq. (21)), a similar analysis as EWC and A-EWC can be applied if we simply replace the regularization factor F of EWC with the regularization factor M of MAS. In fact, the solutions of ANCL for

EWC and MAS are located on the interpolation between the old weights and the auxiliary weights and reaches the highest CKA scores with the multitask model on it, which can be seen in the trade-off analysis (Sec. 5) of the main paper.

Among distillation losses, we first take a derivative on LFL loss (Eq. (4)) with respect to the model parameters θ:

$$\nabla_\theta \mathcal{L}_{\text{LFL}} = \underbrace{\nabla_\theta \mathcal{L}_t(\theta)}_{\text{Task-specific grad.}} + \underbrace{2\lambda(f(x_j;\theta) - f(x_j;\theta^*_{1:t-1}))\nabla_\theta f(x_j;\theta)}_{\text{Distillation grad.}}. \tag{37}$$

The gradient of LFL loss can be divided into two part parts: the task-specific gradient that drives θ toward the minima of current task loss and the distillation gradient that transfers the knowledge of old network to the current network by minimizing the difference of activations. Compared to EWC and MAS, LFL has more flexibility to memorize the previous knowledge since it doesn't directly regularize its weight but activations. The model trained with too high λ will mainly focus on generating exactly the same activations as the old model while neglecting learning a new task.

Next, we take the same derivative on A-LFL loss in Eq. (23):

$$\nabla_\theta \mathcal{L}_{\text{A-LFL}} = \nabla_\theta \mathcal{L}_t(\theta) + 2\lambda(f(x_j;\theta) - f(x_j;\theta^*_{1:t-1}))\nabla_\theta f(x_j;\theta) + 2\lambda_a(f(x_j;\theta) - f(x_j;\theta^*_t))\nabla_\theta f(x_j;\theta) \tag{38}$$

which can be organized as:

$$\nabla_\theta \mathcal{L}_{\text{A-LFL}} = \underbrace{\nabla_\theta \mathcal{L}_t(\theta)}_{\text{Task-specific grad.}} + \underbrace{2(\lambda+\lambda_a)(f(x_j;\theta) - \frac{\lambda f(x_j;\theta^*_{1:t-1}) + \lambda_a f(x_j;\theta^*_t)}{\lambda+\lambda_a})\nabla_\theta f(x_j;\theta)}_{\text{Double distillation grad.}}. \tag{39}$$

Then, the double distillation gradient moves the old and new knowledge from the two networks (the old and auxiliary networks) into the current network by driving the activation of the main network to the interpolated activation between the auxiliary network and the old network. The interpolation is decided by the ratio of λ and λ_a which strike a balance between stability (old network) and plasticity (auxiliary network). Moreover, it is worth to note that Eq. (39) becomes a similar form as the graident of A-EWC if we assume one-hidden-layer network where $f(x_j;\theta^*_{1:t-1}) = W_{old}x_j$, $f(x_j;\theta^*_t) = W_{aux}x_j$, and $f(x_j;\theta) = Wx_j$ hold for $W_{old}, W_{aux}, W \in \mathbb{R}^{p\times d}$ and $x_j \in \mathbb{R}^{d\times 1}$:

$$\nabla_\theta \mathcal{L}_{\text{A-LFL}} = \nabla_\theta \mathcal{L}_t(\theta) + 2(\lambda+\lambda_a)(W - \frac{\lambda W_{old} + \lambda_a W_{aux}}{\lambda+\lambda_a})x_j x_j^T. \tag{40}$$

Now, the second term drifts the model weight W (except last layer) toward the interpolation of the old weight W_{old} and the auxiliary weight W_{aux} like the gradient of A-EWC.

Next, let's consider the loss of LwF (Eq. (2)). For simplicity, we assume that $y^c(x_j;\theta)$ and $y^c(x_j;\theta^*_{1:t-1})$ are softmax logits scaled by temperature τ such as:

$$y^c(x_j;\theta) = \frac{e^{\boldsymbol{o}^c(x_j;\theta)/\tau}}{\sum_k e^{\boldsymbol{o}^k(x_j;\theta)/\tau}}, \qquad y^c(x_j;\theta^*_{1:t-1}) = \frac{e^{\boldsymbol{o}^c(x_j;\theta^*_{1:t-1})/\tau}}{\sum_k e^{\boldsymbol{o}^k(x_j;\theta^*_{1:t-1})/\tau}}. \tag{41}$$

Then, referring to [9], the gradient of LwF with respect to θ can be calculated as follows (detailed proof in Appendix H.2):

$$\nabla_\theta \mathcal{L}_{\text{LwF}} = \nabla_\theta \mathcal{L}_t(\theta) + \frac{\lambda}{\tau}(y(x_j;\theta) - y(x_j;\theta^*_{1:t-1}))\nabla_\theta \boldsymbol{o}(x_j;\theta). \tag{42}$$

When τ is sufficiently large, the softmax in Eq. (41) can be approximated using $exp(\boldsymbol{o}(x_j;\theta)/\tau) \approx 1 + \boldsymbol{o}(x_j;\theta)/\tau$ which substitute $y(x_j;\theta)$ and $y(x_j;\theta^*_{1:t-1})$ in Eq. (42):

$$\nabla_\theta \mathcal{L}_{\text{LwF}} \approx \nabla_\theta \mathcal{L}_t(\theta) + \frac{\lambda}{\tau}(\frac{1+\boldsymbol{o}(x_j;\theta)/\tau}{C_{1:t} + \sum_k \boldsymbol{o}^k(x_j;\theta)/\tau} - \frac{1+\boldsymbol{o}(x_j;\theta^*_{1:t-1})/\tau}{C_{1:t} + \sum_k \boldsymbol{o}^k(x_j;\theta^*_{1:t-1})/\tau})\nabla_\theta \boldsymbol{o}(x_j;\theta) \tag{43}$$

We further assume the logits of the main network and the old network have zero-mean ($i.e.$ $\sum_k \boldsymbol{o}^k(x_j;\theta) = 0$ and $\sum_k \boldsymbol{o}^k(x_j;\theta^*_{1:t-1}) = 0$). Finally, we obtain the following approximation:

$$\nabla_\theta \mathcal{L}_{\text{LwF}} \approx \underbrace{\nabla_\theta \mathcal{L}_t(\theta)}_{\text{Task-specific grad.}} + \underbrace{\frac{\lambda}{C_{1:t}\tau^2}(\boldsymbol{o}(x_j;\theta) - \boldsymbol{o}(x_j;\theta^*_{1:t-1}))\nabla_\theta \boldsymbol{o}(x_j;\theta)}_{\text{Distillation grad.}}. \tag{44}$$

This demonstrates that the gradient of LwF is in the equivalent form as the gradient of LFL under sufficiently large temperature and zero-mean logits of both main and old network. Thereby, similar interpretation as LFL is applicable. One should just note that $\boldsymbol{o}(x_j;\theta)$ in LwF is a logit and $f(x_j;\theta)$ in LFL is an activation before last layer.

Similarly, we can approximate the gradient of A-LwF in Eq. (22):

$$\nabla_\theta \mathcal{L}_{\text{A-LwF}} \approx \nabla_\theta \mathcal{L}_{\text{t}}(\theta) + \frac{\lambda}{C_{1:t}\tau^2}(\boldsymbol{o}(x_j;\theta) - \boldsymbol{o}(x_j;\theta^*_{1:t-1}))\nabla_\theta \boldsymbol{o}(x_j;\theta) + \frac{\lambda_a}{C_{1:t}\tau^2}(\boldsymbol{o}(x_j;\theta) - \boldsymbol{o}(x_j;\theta^*_t))\nabla_\theta \boldsymbol{o}(x_j;\theta). \quad (45)$$

Then, we get the final form as follows:

$$\nabla_\theta \mathcal{L}_{\text{A-LwF}} \approx \underbrace{\nabla_\theta \mathcal{L}_{\text{t}}(\theta)}_{\text{Task-specific grad.}} + \underbrace{\frac{\lambda+\lambda_a}{C_{1:t}\tau^2}(\boldsymbol{o}(x_j;\theta) - \frac{\lambda \boldsymbol{o}(x_j;\theta^*_{1:t-1}) + \lambda_a \boldsymbol{o}(x_j;\theta^*_t)}{\lambda+\lambda_a})\nabla_\theta \boldsymbol{o}(x_j;\theta)}_{\text{Double distillation grad.}}. \quad (46)$$

Similar to A-LFL, the second part of gradient shift the logit of the main network toward the interpolation of the logits from the old network and the new network. The stability-plasticity dilemma is again solved by directly controlling λ and λ_a.

Unlike the analysis of EWC and MAS, we cannot claim that the optimal weight of A-LFL and A-LwF will be located on the interpolation between the old and auxiliary weights (actually, it is not true referring to Fig. 2 in the main paper). Instead, we can at least claim that the double distillation gradient added to the task-specific gradient in Eq. (39) and Eq. (46) tends to derive the activation of main network to the interpolated activation between the auxiliary and old network. In the trade-off analysis (Sec. 5) of the main paper, it is confirmed that the solutions of LwF and LFL tends to move toward the weight space between the old weights and the auxiliary weights and the highest CKA score with the multitask model is also achieved in the middle.

Other four methods (iCaRL [21], BiC [23], LUCIR [10], and PODNet [4]) for class incremental learning are basically originated from distillation-based methods (LwF and LFL). Thus, our previous analyses can be extended or reused according to their variations. For example, iCaRL and BiC are both based on the LwF loss (see Eq. (5)), except the facts that both methods retain a memory buffer and BiC has additional bias correction stage. Therefore, the analysis of LwF still holds for these two methods. LUCIR (Eq. (10)) also adopts the distillation loss with the margin ranking loss which is aligned with the extension of distillation-based methods. In PODNet, pooled outputs distillation (POD) loss (Eq. (17)) constrains the output of each intermediate convolutional layer, which is similar to LFL loss that restricts only the final output. Thus, extending the analysis of LFL to all outputs of intermediate layers will be sufficient to explain the gradient of PODNet.

In conclusion, CL losses bind the training dynamic of the current model with corresponding old one through hyperparameter λ. Thus, the high value of λ hinders learning a new data as it reduces the effective learning rate of the task-specific gradient or increases the size of the distillation gradient. However, ANCL loss function promotes the learning of a new data through the interpolated parameters or the interpolated activations in the double distillation gradient even when the current model is highly regularized toward the old model. This interpolated region is turned out to be a better trade-off between stability and plasticity by Sec. 5 in the main text.

F. Details on Experiments (Sec. 4 in the Main Paper)

F.1. Implementation Details

The model is trained from scratch and every experiment is conducted 3 times with different seeds to generate averaged metrics. The class order is fixed following iCaRL [21] to reduce variance in results such that each task always consists of the same set of classes. We build our code implementation based on Framework for Analysis of Class-Incremental Learning (FACIL) [17] which supports several benchmarks and implements existing CL methods. SGD optimizer with momentum 0.9 and batch size 128 is applied to all experiments. The initial learning rate of each task (for task incremental learning) or phase (for class incremental learning) is chosen among the set of learning rate [0.1, 0.05, 0.01, 0.005, 0.001] by grid search. The learning rate of the first task or initial phase is always chosen slightly bigger than the following tasks since the model is trained from scratch. Furthermore, the learning rate is decreased by the factor of 3 whenever there is no improvement in validation loss during 5 epochs. When the learning rate reaches a given minimum threshold (0.0001), training is stopped. The model can be trained for a maximum of 200 epochs per task. Lastly, we conduct all experiments using gpu "NVIDIA GeForce GTX 1080 Ti", and the code implementation of our work will be publicly available after final decision.

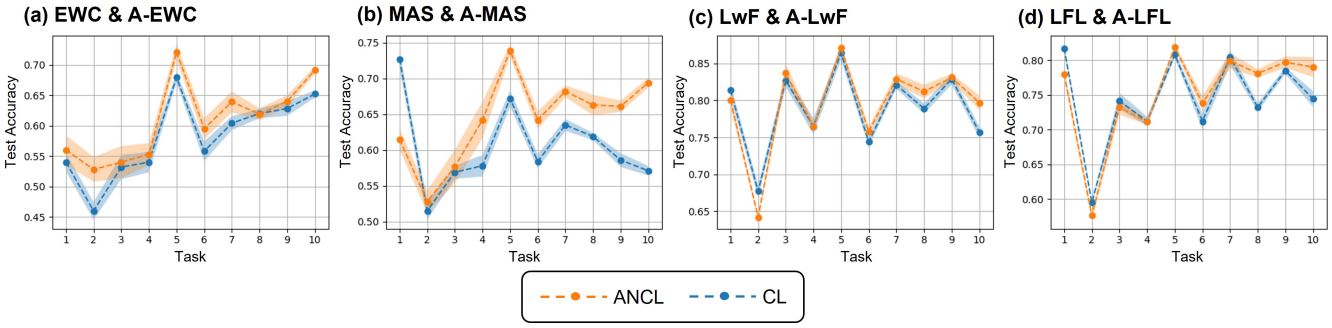

Figure 1. The final accuracy on all tasks of (1) CIFAR-100/10 with its standard error as an error band. Orange line represents ANCL methods ((a) A-EWC, (b) A-MAS, (c) A-LwF, and (d) A-LFL and blue line visualizes CL methods ((a) EWC, (b) MAS, (c) LwF, and (d) LFL).

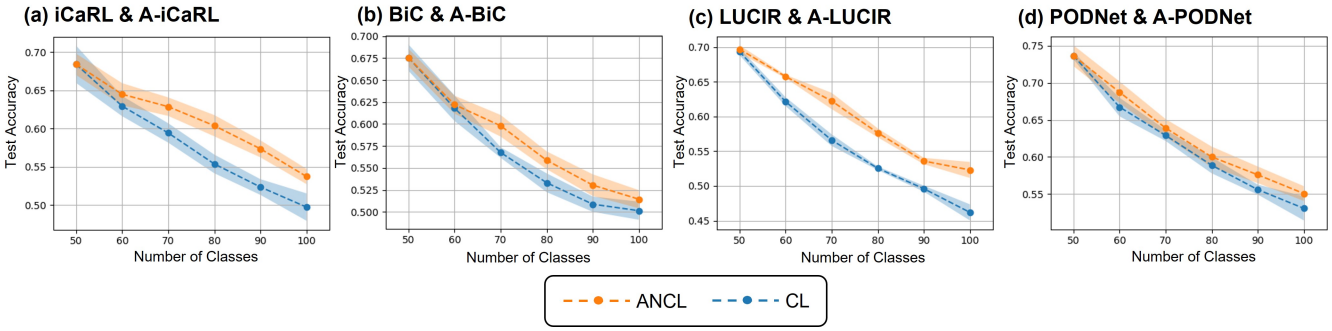

Figure 2. The accuracy at each phase on (5) CIFAR-100/6 with its standard error as an error band. Orange line represents ANCL methods ((a) A-iCaRL, (b) A-BiC, (c) A-LUCIR, and (d) A-PODNet) and blue line visualizes CL methods ((a) iCaRL, (b) BiC, (c) LUCIR, and (d) PODNet).

	CIFAR-100		Tiny ImageNet	
Methods	(5)	(6)	(7)	(8)
Fine-tuning	$40.63_{\pm 0.93}$	$38.09_{\pm 0.42}$	$18.80_{\pm 0.55}$	$16.34_{\pm 0.44}$
Joint	$66.10_{\pm 1.04}$	$64.40_{\pm 0.83}$	$44.01_{\pm 0.34}$	$43.54_{\pm 0.37}$
EEIL [3]	$41.61_{\pm 1.10}$	$39.48_{\pm 0.27}$	$20.99_{\pm 0.24}$	$19.26_{\pm 0.96}$
iCaRL [21]	$49.75_{\pm 0.93}$	$45.85_{\pm 0.89}$	$27.83_{\pm 0.43}$	$29.26_{\pm 0.86}$
w/ ANCL (ours)	$53.87_{\pm 0.59}$	$48.72_{\pm 0.33}$	$30.36_{\pm 0.97}$	$31.32_{\pm 0.41}$
BiC [23]	$50.13_{\pm 0.77}$	$47.97_{\pm 1.17}$	$32.87_{\pm 0.19}$	$29.45_{\pm 0.54}$
w/ ANCL (ours)	$51.35_{\pm 0.89}$	$50.44_{\pm 1.31}$	$34.26_{\pm 0.34}$	$32.40_{\pm 0.39}$
LUCIR [10]	$46.19_{\pm 0.52}$	$45.89_{\pm 0.68}$	$24.07_{\pm 0.46}$	$20.89_{\pm 0.78}$
w/ ANCL (ours)	$52.30_{\pm 0.48}$	$49.07_{\pm 0.43}$	$28.96_{\pm 0.58}$	$23.65_{\pm 0.55}$
PODNet [4]	$53.05_{\pm 0.89}$	$49.63_{\pm 0.83}$	$30.78_{\pm 0.27}$	$30.25_{\pm 0.52}$
w/ ANCL (ours)	$55.01_{\pm 0.32}$	$51.39_{\pm 0.62}$	$32.53_{\pm 0.50}$	$33.11_{\pm 0.61}$

Table 3. The final accuracy (%) on benchmark (5)-(8). Reported metrics are averaged over 3 runs (averaged accuracy $\pm$ standard error). ANCL methods are colored in gray.

F.2. Detail Results on Task Incremental Scenario

In addition to the averaged accuracy table in the main paper, we plot the final accuracy of the sequential task on benchmark (1) CIFAR-100/10 in Fig. 1. In every figure, the accuracy of most tasks of ANCL approaches are higher than that of CL approaches. Concretely, ANCL achieves better performance in the later task compare to CL at the cost of losing earlier task accuracy, which is well shown in Figure (b), (c), and (d). This is because ANCL methods is able to learn a new task

better than CL through better stability-plasticity balance. As a trade-off, ANCL losses a bit of ability to remember initial knowledge, but ANCL is still comparable to CL in earlier tasks.

F.3. Detail Results on Class Incremental Scenario

Similar to Fig. 1, we plot the accuracy at each phase on class incremental scenario in Fig. 2. ANCL and CL are trained on initial 50 classes and then incrementally learn 10 classes per phase on CIFAR-100. In all methods, ANCL outperforms CL in every phase. Especially, LUCIR shows the biggest improvement and PODNet shows the smallest improvement among 4 methods. Even though ANCL naturally mimicks the regularizer of the original method to adopt plasticity in their method, some methods are less compatible with ANCL.

The final accuracy of Tab. 3 in the main paper is shown in Tab. 3. In class incremental learning, the final accuracy means the accuracy on all classes (100 classes for CIFAR-100 and 200 classes for Tiny ImageNet) after the training of the last phase. Applying ANCL improves the final accuracy of CL by 1-4 %.

F.4. Hyperparameter Grid Search Result

In this section, we present the grid search results of λ and λ_a on the benchmark (1)-(8) evaluated in Tab. 2 and Tab. 3 in the main paper. For other hyperparameters required for specific methods, we follow the configuration of the original paper.

Methods	Hyperparameter	(1) CIFAR-100/10 (10 tasks)	(2) CIFAR-100/20 (20 tasks)
EWC [12]	λ	[5000, **10000**, 20000, 40000, 80000]	[10000, 20000, 40000, **80000**, 160000]
w/ ANCL (ours)	λ_a	[0.1, 1, **10**, 100, 1000, 10000]	[0.1, 1, 10, **100**, 1000, 10000]
MAS [1]	λ	[1, 5, 10, **50**, 100, 200]	[1, 5, 10, 50, **100**, 200]
w/ ANCL (ours)	λ_a	[0.1, 0.5, 1, **5**, 10, 50]	[0.1, 0.5, **1**, 5, 10, 50]
LwF [13]	λ	[0.1, 1, **10**, 100, 200, 400]	[0.1, 1, **10**, 100, 200, 400]
w/ ANCL (ours)	λ_a	[0.1, 0.5, **1**, 5, 10]	[0.1, 0.5, **1**, 5, 10]
LFL [11]	λ	[10, 100, 200, **400**, 800]	[100, 200, 400, **800**, 1600]
w/ ANCL (ours)	λ_a	[10, 50, **100**, 200, 400]	[5, **10**, 50, 100, 200]
Methods	Hyperparameter	(3) TinyImagenet-200/10 (10 tasks)	(4) TinyImagenet-200/20 (20 tasks)
EWC [12]	λ	[5000, **10000**, 20000, 40000, 80000]	[5000, **10000**, 20000, 40000, 80000]
w/ ANCL (ours)	λ_a	[10, 100, 1000, **2000**, 3000]	[10, 100, **1000**, 2000, 3000]
MAS [1]	λ	[1, 5, **10**, 50, 100, 200]	[1, 5, **10**, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.05, **0.1**, 0.5, 1, 5, 10]	[0.01, **0.05**, 0.1, 0.5, 1, 5, 10]
LwF [13]	λ	[0.1, 1, **10**, 100, 200, 400]	[0.1, 1, **10**, 100, 200, 400]
w/ ANCL (ours)	λ_a	[0.1, 0.5, 1, **5**, 10]	[0.1, 0.5, 1, **5**, 10]
LFL [11]	λ	[100, 200, 400, **800**, 1600]	[200, 400, 800, **1600**, 3200]
w/ ANCL (ours)	λ_a	[10, 50, 100, **200**, 300]	[10, **50**, 100,200, 300]

Table 4. Hyperparameter search for CL and ANCL on benchmarks (1)-(4) in Tab. 2 of the main paper. The set of parameters on which grid search is performed is shown, and selected hyperparameter is emphasized in bold.

G. Details on Stability-Plasticity Trade-off Analysis (Sec. 5 in Main Paper)

G.1. Training Regime for Analysis

In all analyses in the main text, the specific learning regime described in Fig. 3 is adopted stemming from the regime from [19]. On task t, every model starts training from the multitask weights θ_{t-1}^{multi} which is trained on combined dataset $D_{1:t-1}$ (*i.e.* train on $D_{1:t-1} = D_1 \cup \cdots \cup D_{t-1}$). If we fine-tune the model on data D_t without any regularization, it returns the auxiliary network θ_t^{aux} which will regularize ANCL later. ANCL and CL approach can be applied to obtain θ_t^{ANCL} and θ_t^{CL} each. Here, the multitask weights θ_{t-1}^{multi} on task $t-1$ works as the old network weights θ_{t-1}^{old} to regularize ANCL and CL. Finally, the initial weights are trained on data $D_{1:t}$ to train next multitask model θ_t^{multi} and it becomes next starting point for task $t+1$. We used fixed multitask weights as an initialization at the start of each task for fair comparison among all methods. Otherwise, every method will have different old and auxiliary network which end up confusing following analysis.

Methods	Hyperparameter	(5) CIFAR-100/6 (1+5 phases)	(6) CIFAR-100/11 (1+10 phases)
iCaRL [21]	λ	[0.01, 0.05, 0.1, **0.5**, 1, 5]	[0.01, 0.05, 0.1, 0.5, 1, **5**, 10]
w/ ANCL (ours)	λ_a	[0.01, **0.05**, 0.1, 0.5, 1, 5]	[0.01, 0.05, 0.1, **0.5**, 1, 5]
BiC [23]	λ	[1, 5, **10**, 50, 100, 200]	[1, 5, **10**, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.1, 0.5, 1, **5**, 10, 50]	[0.1, **0.5**, 1, 5, 10, 50]
LUCIR [10]	λ	[1, **5**, 10, 50, 100, 200]	[1, **5**, 10, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.1, 0.5, **1**, 5, 10]	[0.05 ,**0.1**, 0.5, 1, 5, 10]
PODNet [4]	λ	[1, 5, **10**, 50, 100, 200]	[1, 5, **10**, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.1, 0.5, **1**, 5, 10]	[0.1, 0.5, **1**, 5, 10]
Methods	Hyperparameter	(7) TinyImagenet-200/11 (1+10 phases)	(8) TinyImagenet-200/21 (1+20 phases)
iCaRL [21]	λ	[0.01, 0.05, **0.1**, 0.5, 1, 5]	[0.01, 0.05, 0.1, 0.5, **1**, 5]
w/ ANCL (ours)	λ_a	[0.01, **0.05**, 0.1, 0.5, 1, 5]	[0.01, 0.05, 0.1, **0.5**, 1, 5]
BiC [23]	λ	[1, 5, **10**, 50, 100, 200]	[1, 5, **10**, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.05, **0.1**, 0.5, 1, 5, 10, 50]	[0.005 , **0.01**, 0.05, 0.1, 0.5, 1, 5]
LUCIR [10]	λ	[1, 5, 10, **50**, 100, 200]	[1, **5**, 10, 50, 100, 200]
w/ ANCL (ours)	λ_a	[0.005, **0.01**, 0.05 ,0.1, 0.5, 1]	[0.1, 0.5, 1, **5**, 10]
PODNet [4]	λ	[1, 5, 10, 50, **100**, 200]	[10, 50, 100, 200, **400**, 800]
w/ ANCL (ours)	λ_a	[0.1, 0.5, **1**, 5, 10]	[0.1, 0.5, **1**, 5, 10]

Table 5. Hyperparameter search for CL and ANCL on benchmarks (5)-(8) in Tab. 3 of the main paper.. The set of parameters on which grid search is performed is shown, and selected hyperparameter is emphasized in bold.

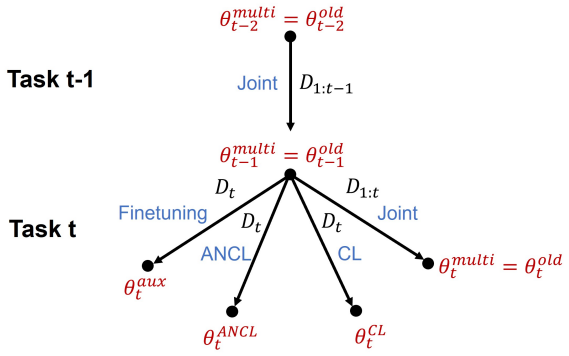

Figure 3. Training regime for analysis. D_t stands for the dataset of task t and $D_{1:t-1}$ means combined dataset from task 1 to task $t-1$.

G.2. Mode Connectivity Figures

Recent works [5, 6] find a simple curve between the two local optima of deep neural networks (DNN) such that train loss and test error remain low along the curve. The simple linear path with low error can be visualized on the loss surface of DNN in gradient-based optimization setting. This path called *Mode Connectivity* has been studied empirically and theoretically with some assumptions in different papers.

Mode connectivity in continual learning is first investigated by [19]. They empirically show that mode connectivity holds between continual learning and multitask solutions when every training starts from same initialization. We also follow [19] to visualize mean accuracy landscape of the main paper and mode connectivity figures in Fig. 4 (details in Appendix G.3).

In Fig. 4, θ_1^{old} is the old network weights trained on task 1 only and θ_2^{aux} is the auxiliary network weights that is trained on task 2 initialized by the old network. θ_2^{multi} is the multitask solution trained on the dataset of task 1 and 2. It is clear that the continual learning solutions (θ_1^{old} and θ_2^{aux}) are linearly connected to the multitask solution (θ_2^{multi}) by low loss and high accuracy path. Therefore, linear mode connectivity is valid in both loss and accuracy landscape. In the left and center column, the multitask weights are located in higher accuracy and lower loss contour compared to the continual learning weights. This is because the multitask model has a full access to previous datasets, consequently obtaining higher discriminating ability, while the continual models can learn only from the current dataset. The figures in right column visualize the mean loss and mean accuracy of task 1 and 2 and Fig. 4 (f) is used in mean accuracy landscape of the main paper. Higher accuracy and

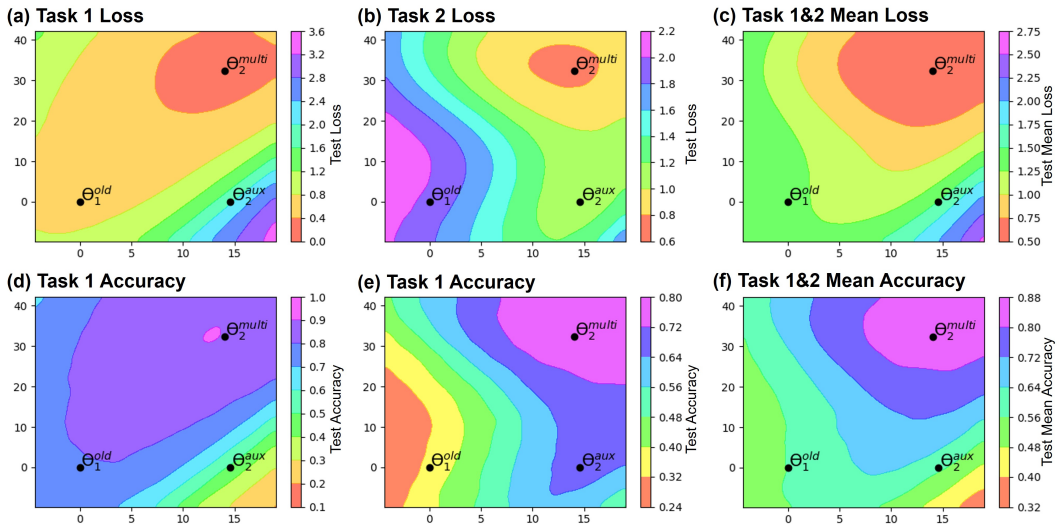

Figure 4. Test loss landscape (top low) and test accuracy landscape (bottom low) of task 1&2 of benchmark (1) CIFAR-100/10. θ_1^{old}, θ_2^{aux}, and θ_2^{multi} are used to plot the two-dimensional subspace following [19].

lower loss can be achieved in the middle of θ_1^{old} and θ_2^{aux}.

G.3. The Visualization of Mean Accuracy Landscape

In this Section, we explain how we visualize mean accuracy landscape in Sec. 5 of main text and Appendix G.2 following [19]. In order to build two basis vectors of the plane, we need three points w_1, w_2, and w_3 which corresponds to θ_1^{old}, θ_2^{aux}, and θ_2^{multi} respectively in mode connectivity figures. Each point refers to a high-dimensional weight vector which is obtained by flattening the weights of each layer in the neural network and then concatenating the flattened vectors including bias vector and batch normalization parameters. With three weight vectors at hand, we perform the following procedure:

1. Calculate two basis vectors: $\vec{u} = w_2 - w_1$, and $\vec{v} = w_3 - w_1$.

2. Orthogonalize the basis vectors by calculating

$$\vec{v} = \vec{v} - \|\vec{v}\| cos(\theta) \frac{\vec{u}}{\|\vec{u}\|} \tag{47}$$

$$= \vec{v} - \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|^2} \vec{u} \tag{48}$$

   where θ denotes the angle between $\vec{v}$ and $\vec{u}$.

3. Define a Cartesian coordinate system in (x, y) plane that maps each coordinate to a parameter space by calculating $p(x, y) = w_1 + x \cdot \vec{u} + y \cdot \vec{v}$.

4. For a defined grid on this coordinate system, we calculate the empirical loss and accuracy of each (x, y) coordinate by reconstructing the weights of neural network from the high-dimensional vector $p(x, y)$.

5. The flattened weights of CL w_{CL} can be expressed in the coordinate system (x_{CL}, y_{CL}) by projecting the weight vector to $\vec{u}$ and $\vec{v}$ each:

$$x_{CL} = \frac{(w_{CL} - w_1) \cdot \vec{u}}{\|\vec{u}\|^2}, \tag{49}$$

$$y_{CL} = \frac{(w_{CL} - w_1) \cdot \vec{v}}{\|\vec{v}\|^2} \tag{50}$$

   In the same way, (x_{ANCL}, y_{ANCL}) can be obtained from w_{ANCL}.

The original weights w_{CL} can be reconstructed from $p(x_{CL}, y_{CL})$ and a residual vector $\vec{R}$. The residual vector is aligned with the dimension orthogonal to both $\vec{u}$ and $\vec{v}$, thereby not being reflected in the coordinate system. Therefore, w_{CL} can be fully expressed as below:

$$w_{CL} = p(x_{CL}, y_{CL}) + \vec{R} = w_1 + x_{CL} \cdot \vec{u} + y_{CL} \cdot \vec{v} + \vec{R} \tag{51}$$

H. Mathematical Proofs

H.1. The Derivation of Eq. (35) in Appendix E

In this section, we derive Eq. (35) by extending the derivation in [16]. Let's start from rewriting the update rule of A-EWC from k^{th} iteration to $(k+1)^{th}$ iteration with a learning rate η:

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \eta(\nabla_{\theta_i^{(k)}} \mathcal{L}_t(\theta^{(k)}) + \lambda F_{1:t-1,i}(\theta_i^{(k)} - \theta^*_{1:t-1,i}), + \lambda_a F_{t,i}(\theta_i^{(k)} - \theta^*_{t,i})) \tag{52}$$

where $\theta_i^{(k+1)}$ and $\theta_i^{(k)}$ are the i^{th} model parameter on k^{th} and $(k+1)^{th}$ iteration each such that $\theta_i \in \theta = (\theta_1, \ldots, \theta_P)$ for the model weight $\theta \in \mathbb{R}^P$. $\nabla_{\theta_i^{(k)}} \mathcal{L}_t(\theta^{(k)})$ denotes the gradient of cross-entropy loss with respect to $\theta_i^{(k)}$ for a current task. $F_{1:t-1,i}$ and $F_{t,i}$ are the importance of EWC, the approximation of Fisher Information Matrix, for the old parameter $\theta^*_{1:t-1,i}$ and the auxiliary parameter $\theta^*_{t,i}$ respectively. Then, λ and λ_a adjust the strength of two regularizers.

After rearranging Eq. (52), the weight is updated as follows:

$$\theta_i^{(k+1)} = (1-\alpha-\beta)\theta_i^{(k)} + \alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i} - \eta g_i^{(k)} \tag{53}$$

where $\alpha = \eta\lambda F_{1:t-1,i}$, $\beta = \eta\lambda_a F_{t,i}$ and $g_i^{(k)} = \nabla_{\theta_i^{(k)}} \mathcal{L}_t(\theta^{(k)})$ are applied. From above update rule, we will prove that Eq. (35) holds true for all k^{th} iteration through mathematical induction. At 1st iteration, we have following equation by directly substituting $k = 0$ in Eq. (53):

$$\theta_i^{(1)} = (1-\alpha-\beta)\theta^*_{1:t-1,i} + (\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \eta g_i^{(0)} \tag{54}$$

where the model for t^{th} task is initialized by the final optimal weight of $(t-1)^{th}$ task such that $\theta_i^{(0)} = \theta^*_{1:t-1,i}$. Then, it automatically satisfies Eq. (35) as following:

$$\theta_i^{(1)} = (1-\alpha-\beta)\theta^*_{1:t-1,i} + \sum_{l=0}^{0}(1-\alpha-\beta)^l(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \sum_{l=0}^{0}[(1-\alpha-\beta)^{(1-l-1)}\eta]g_i^{(l)} \tag{55}$$

Assume Eq. (35) holds true at k^{th} iteration:

$$\theta_i^{(k)} = (1-\alpha-\beta)^k\theta^*_{1:t-1,i} + \sum_{l=0}^{k-1}(1-\alpha-\beta)^l(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \sum_{l=0}^{k-1}[(1-\alpha-\beta)^{(k-l-1)}\eta]g_i^{(l)}. \tag{56}$$

Then, at $(k+1)^{th}$ iteration, we have:

$$\theta_i^{(k+1)} = (1-\alpha-\beta)\theta_i^{(k)} + \alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i} - \eta g_i^{(k)} \tag{57}$$

$$= (1-\alpha-\beta)((1-\alpha-\beta)^k\theta^*_{1:t-1,i} + \sum_{l=0}^{k-1}(1-\alpha-\beta)^l(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \sum_{l=0}^{k-1}[(1-\alpha-\beta)^{(k-l-1)}\eta]g_i^{(l)}) + \alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i} - \eta g_i^{(k)} \tag{58}$$

$$= (1-\alpha-\beta)^{(k+1)}\theta^*_{1:t-1,i} + \sum_{l=0}^{k-1}(1-\alpha-\beta)^{(l+1)}(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) \\ - \sum_{l=0}^{k-1}[(1-\alpha-\beta)^{(k+1-l-1)}\eta]g_i^{(l)} + \alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i} - \eta g_i^{(k)} \tag{59}$$

$$= (1-\alpha-\beta)^{(k+1)}\theta^*_{1:t-1,i} + \sum_{l=0}^{k}(1-\alpha-\beta)^{l}(\alpha\theta^*_{1:t-1,i} + \beta\theta^*_{t,i}) - \sum_{l=0}^{k}[(1-\alpha-\beta)^{(k+1-l-1)}\eta]g_i^{(l)}. \tag{60}$$

The last equality satisfies Eq. (35) at $(k+1)^{th}$ iteration which ends our proof.

H.2. The Derivation of Eq. (42) in Appendix E

For the completeness of our paper, we expand the derivation of Eq. (42) according to [9]. We first rewrite the loss of LwF in Eq. (2) and define the second term as distillation loss $\mathcal{L}_{\text{D}}$:

$$\mathcal{L}_{\text{LwF}} = \mathcal{L}_{\text{t}}(\theta) + \lambda \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1}) \log y^c(x_j;\theta), \tag{61}$$

$$\mathcal{L}_{\text{D}} = \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1}) \log y^c(x_j;\theta). \tag{62}$$

Then, we take a derivative on the distillation loss $\mathcal{L}_{\text{D}}$ with respect to the logit $\boldsymbol{o}^h(x_j;\theta)$:

$$\nabla_{\boldsymbol{o}^h(x_j;\theta)}\mathcal{L}_{\text{D}} = \frac{\partial}{\partial \boldsymbol{o}^h(x_j;\theta)} \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1}) \log y^c(x_j;\theta) \tag{63}$$

$$= \sum_{c=1}^{C_{1:t}} -y^c(x_j;\theta^*_{1:t-1}) \frac{\partial \log y^c(x_j;\theta)}{\partial \boldsymbol{o}^h(x_j;\theta)} \tag{64}$$

$$= \sum_{c=1}^{C_{1:t}} -\frac{y^c(x_j;\theta^*_{1:t-1})}{y^c(x_j;\theta)} \frac{\partial y^c(x_j;\theta)}{\partial \boldsymbol{o}^h(x_j;\theta)} \tag{65}$$

$$= -\frac{y^h(x_j;\theta^*_{1:t-1})}{y^h(x_j;\theta)} \frac{\partial y^h(x_j;\theta)}{\partial \boldsymbol{o}^h(x_j;\theta)} + \sum_{c\neq h} -\frac{y^c(x_j;\theta^*_{1:t-1})}{y^c(x_j;\theta)} \frac{\partial y^c(x_j;\theta)}{\partial \boldsymbol{o}^h(x_j;\theta)} \tag{66}$$

$$= \frac{1}{\tau}(y^h(x_j;\theta) - y^h(x_j;\theta^*_{1:t-1})) \tag{67}$$

where in the last equality we applied the following:

$$\frac{\partial y^c(x_j;\theta)}{\partial \boldsymbol{o}^h(x_j;\theta)} = \partial(\frac{e^{\boldsymbol{o}^c(x_j;\theta)/\tau}}{\sum_k e^{\boldsymbol{o}^k(x_j;\theta)/\tau}})/\partial \boldsymbol{o}^h(x_j;\theta) = \begin{cases} \frac{1}{\tau}y^h(x_j;\theta)(1-y^h(x_j;\theta)), & \text{if } c=h \\ -\frac{1}{\tau}y^c(x_j;\theta)y^h(x_j;\theta), & \text{otherwise} \end{cases} \tag{68}$$

If we take derivative on the loss of LwF with respect to θ:

$$\nabla_\theta \mathcal{L}_{\text{LwF}} = \nabla_\theta \mathcal{L}_{\text{t}}(\theta) + \lambda \frac{\partial}{\partial \theta}\mathcal{L}_{\text{D}} \tag{69}$$

$$= \nabla_\theta \mathcal{L}_{\text{CE}}(\theta) + \lambda \frac{\partial \boldsymbol{o}(x_j;\theta)}{\partial \theta} \frac{\partial \mathcal{L}_{\text{D}}}{\partial \boldsymbol{o}(x_j;\theta)} \tag{70}$$

$$= \nabla_\theta \mathcal{L}_{\text{CE}}(\theta) + \frac{\lambda}{\tau}(y(x_j;\theta) - y(x_j;\theta^*_{1:t-1}))\nabla_\theta \boldsymbol{o}(x_j;\theta) \tag{71}$$

In the last equality, we applied Eq. (67) and it ends our derivation of Eq. (42).

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018. 2, 5, 7, 12

[2] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006. 1, 2

[3] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018. 11

[4] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, pages 86–102. Springer, 2020. 4, 5, 7, 10, 11, 13

[5] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR, 2018. 13

[6] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018. 13

[7] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17, 2004. 5

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[9] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1, 2, 9, 16

[10] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019. 3, 4, 5, 6, 7, 10, 11, 13

[11] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016. 3, 5, 7, 12

[12] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. 1, 2, 5, 7, 12

[13] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 1, 2, 3, 5, 7, 12

[14] Guoliang Lin, Hanlu Chu, and Hanjiang Lai. Towards better plasticity-stability trade-off in incremental learning: A simple linear connector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 89–98, 2022. 1

[15] Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2544–2553, 2021. 1

[16] Ekdeep Singh Lubana, Puja Trivedi, Danai Koutra, and Robert P Dick. How do quadratic regularizers prevent catastrophic forgetting: The role of interpolation. *arXiv preprint arXiv:2102.02805*, 2021. 7, 8, 15

[17] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification. *arXiv preprint arXiv:2010.15277*, 2020. 10

[18] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989. 2

[19] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*, 2020. 1, 12, 13, 14

[20] 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. 4

[21] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 3, 5, 6, 7, 10, 11, 13

[22] Liyuan Wang, Mingtian Zhang, Zhongfan Jia, Qian Li, Chenglong Bao, Kaisheng Ma, Jun Zhu, and Yi Zhong. Afec: Active forgetting of negative transfer in continual learning. *Advances in Neural Information Processing Systems*, 34:22379–22391, 2021. 1, 6, 7

[23] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019. 3, 5, 6, 7, 10, 11, 13

[24] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020. 1