Towards Interpretable Deep Local Learning with Successive Gradient Reconciliation

Yibo Yang [1] Xiaojie Li [2 3] Motasem Alfarra [1] Hasan Hammoud [1] Adel Bibi [4] Philip Torr [4] Bernard Ghanem [1]

Abstract

Relieving the reliance of neural network training on a global back-propagation (BP) has emerged as a notable research topic due to the biological implausibility and huge memory consumption caused by BP. Among the existing solutions, local learning optimizes gradient-isolated modules of a neural network with local errors and has been proved to be effective even on large-scale datasets. However, the reconciliation among local errors has never been investigated. In this paper, we first theoretically study non-greedy layer-wise training and show that the convergence cannot be assured when the local gradient in a module *w.r.t.* its input is not reconciled with the local gradient in the previous module *w.r.t.* its output. Inspired by the theoretical result, we further propose a local training strategy that successively regularizes the gradient reconciliation between neighboring modules without breaking gradient isolation or introducing any learnable parameters. Our method can be integrated into both local-BP and BP-free settings. In experiments, we achieve significant performance improvements compared to previous methods. Particularly, our method for CNN and Transformer architectures on ImageNet is able to attain a competitive performance with global BP, saving more than 40% memory consumption.

1. Introduction

Back-propagation (BP) has been a crucial ingredient for the success of deep learning (LeCun et al., 2015). Although BP is easy to implement and seems indispensable for training deep neural networks, it has been a concern that BP is distinct from how the brain learns and updates, known as biological implausibility (Lillicrap et al., 2020; Bengio et al., 2015). First, BP encounters the weight transport problem (Grossberg, 1987), which means the update of each layer during backward propagation relies on the symmetric weight used for forward propagation (Lillicrap et al., 2016; Liao et al., 2016). Second, compared to the brain that updates neurons instantly using local signals, training neural networks with BP suffers from update locking because the gradient descent in a layer can be only performed after the forward and backward propagations of its subsequent layers (Jaderberg et al., 2017; Frenkel et al., 2021; Dellaferrera & Kreiman, 2022; Halvagal & Zenke, 2023). Moreover, due to update locking, all the activations and gradients of a whole model need to be stored, which is the dominant cause of the huge memory consumption for training modern neural networks whose capacity is continually expanding.

In order to tackle these issues, different approaches are proposed to relieve the reliance on a global BP (Nøkland, 2016; Clark et al., 2021; Silver et al., 2022; Ren et al., 2023; Journé et al., 2023; Belilovsky et al., 2019; Wang et al., 2021; Fournier et al., 2023), among which training with local errors is promising due to its less impaired performance. It divides a neural network into several gradient-isolated modules and trains each locally. The initial explorations adopt greedy layer-wise training for a good initialization before finetuning with BP (Hinton et al., 2006; Bengio et al., 2006). Later studies show that greedy layer-wise training can achieve competitive performance on the large scale ImageNet using auxiliary classifiers (Belilovsky et al., 2019). Since learning sequentially does not enable to optimize the deep representation simultaneously, recent studies favor the non-greedy fashion where each layer is updated locally with a mini-batch data and then passes the output into the next layer (Belilovsky et al., 2020; Siddiqui et al., 2023; Wang et al., 2021). However, a performance drop is still inevitable when increasing the number of local modules, and current studies have not rigorously pinpointed the inherent limitation of local training compared to global BP.

In this paper, we investigate the defect of *non-greedy* layer-wise training from a view of the reconciliation among local errors and propose a remedy for it. Consider a local layer $x_k = f(x_{k-1}, \theta_k)$, where x_{k-1} is the input of layer k and also the output of layer $k-1$, and θ_k is the learnable parameters of this layer. Local training optimizes θ_k independently

[1]King Abdullah University of Science and Technology [2]Harbin Institute of Technology (Shenzhen) [3]Peng Cheng Laboratory [4]University of Oxford. Correspondence to: Yibo Yang <yibo.yang93@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

with a classifier head $l_k = h(x_k, w_k)$ that is parameterized by w_k to produce the local error signal l_k using a loss function, *e.g.* cross entropy loss. Note that the local layer f is a function with two variables x_{k-1} and θ_k. If x_k is the last-layer representation and we use global BP to train the network, the gradient *w.r.t.* the first variable, $\frac{\partial l_k}{\partial x_{k-1}}$, can be back-propagated to prior layers whose update can lead to a new x_{k-1} that reduces l_k. In layer-wise training, the optimization regarding x_{k-1} is achieved by updating θ_{k-1} with the previous local error l_{k-1}. However, there is no assurance that the update of θ_{k-1} in the previous layer can cause a change of Δx_{k-1} that most satisfies the demand of the current layer to reduce l_k. A previous study (Jaderberg et al., 2017) deals with the correctness of local gradients by learning local gradient generators. But it relies on the global true gradient so breaks gradient isolation. We point out that the discordant local update, which global BP is naturally immune to, is the fundamental defect in the context of non-greedy layer-wise training.

To this end, we first theoretically analyze the convergence of a two-layer model. The two layers are updated with their respective local errors in a non-greedy manner. Our result indicates that when the gradient of the second layer *w.r.t.* its input, *i.e.,* $\frac{\partial l_2}{\partial x_1}$, has a large distance to the gradient of the first layer *w.r.t.* its output, *i.e.,* $\frac{\partial l_1}{\partial x_1}$, the convergence of the second layer cannot be guaranteed. Inspired by the result, we propose a simple yet interpretable and effective method, named successive gradient reconciliation. When training the $(k-1)$-th layer locally, we store the gradient *w.r.t.* the output $\frac{\partial l_{k-1}}{\partial x_{k-1}}$, and make the output require gradient being the input of the next layer. At the update of the k-th layer, we calculate the gradient *w.r.t.* the input $\frac{\partial l_k}{\partial x_{k-1}}$, and add a regularizer on the local error to minimize the distance between the two gradients. By doing so, the optimization of each layer is towards a direction such that the parameters θ_k not only decrease the local error l_k, but also enable the change of input Δx_{k-1} coming from the previous layer to decrease l_k. The process is performed layer by layer to successively reconcile local objectives and transmit them into the last layer for a better final representation without breaking gradient isolation.

Our method effectively improves the performance of local training in both local-BP and BP-free cases. In local-BP training, we successfully enable memory-efficient training by removing a global BP. In BP-free training, we adopt a fixed local classifier head borrowing the ideas from (Frenkel et al., 2021; Yang et al., 2022a), and achieve significantly better performance than existing BP-free methods. The contributions of this study can be listed as follows:

- We derive a theoretical convergence bound in the context of non-greedy layer-wise training, and show that the reconciliation among local errors is crucial to ensure the convergence of the last-layer error.
- We propose successive gradient reconciliation, which successively reconciles the local updates of every two neighboring layers without breaking gradient isolation or introducing any learnable parameters. We integrate our method in both local-BP and BP-free settings.
- We conduct extensive experiments on CIFAR-10, CIFAR-100, and ImageNet to verify the effectiveness of our method, and show that our method surpasses previous methods and is able to attain a competitive performance with global BP saving over 40% memory consumption for CNN and Transformer architectures.

2. Related Work

Back-propagation has been important for deep network training due to its effective credit assignment into hierarchical representations (Rumelhart et al., 1986). Despite the success, the weight transport and the update locking problems constrain BP from being biologically plausible and memory efficient (Lillicrap et al., 2020; Crick, 1989; Grossberg, 1987), which has attracted wide research interests from both neuroscience and machine learning communities to propose alternatives to a global BP (Bengio et al., 2015; Hinton, 2022; Halvagal & Zenke, 2023; Song et al., 2024).

Gradient estimation. Introducing auxiliary optimization variables for each layer can spare the need to back propagation a global error (Taylor et al., 2016; Li et al., 2020). Gradient estimation can be also achieved by forward automatic differentiation (Baydin et al., 2022; Silver et al., 2022; Ren et al., 2023) and forward propagation again with a disturbed input (Hinton, 2022; Dellaferrera & Kreiman, 2022). However, these methods have not been scalable on large datasets and are not competitive with BP.

Credit assignment. In order to relax the weight transport problem, different credit assignment strategies are proposed with only sign symmetry (Liao et al., 2016; Xiao et al., 2019) or random feedback connection, known as feedback alignment (FA) (Lillicrap et al., 2016). Later studies improve over FA by adjusting the random connection (Akrout et al., 2019) and directly propagating the global error into each layer (Nøkland, 2016; Clark et al., 2021). However, most of these methods cannot achieve satisfactory performance on large datasets (Bartunov et al., 2018) and local updates still cannot be performed without waiting. DRTP (Frenkel et al., 2021) refrains from update locking by assigning local errors directly from labels instead of the global error, but further introduces performance deterioration.

Local learning. Another promising solution, which our method can be categorized as, is to look for a local update rule such that layers are optimized independently to satisfy both weight-transport free and update unlocking. Hebbian plastic rule based on synaptic plasticity has been leveraged for local update in an unsupervised or self-supervised manner (Illing et al., 2021; Journé et al., 2023; Halvagal & Zenke, 2023). Greedy layer-wise training is initially proved to be effective in producing a good initialization (Yang et al., 2022b) for the subsequent finetuning with BP (Hinton et al., 2006; Bengio et al., 2006). Later studies adopt auxiliary heads to train layers locally for supervised learning (Mostafa et al., 2018; Nøkland & Eidnes, 2019; Belilovsky et al., 2019; 2020; You et al., 2020) and self-supervised learning (Löwe et al., 2019; Xiong et al., 2020; Siddiqui et al., 2023). Although there is still back-propagation within the auxiliary head composed of multiple layers, it helps to attain a comparable performance with global BP on ImageNet (Belilovsky et al., 2019). The recent well-performing methods favor non-greedy layer-wise training (Wang et al., 2021; Siddiqui et al., 2023). However, the fundamental defect of layer-wise training over global BP has not been unveiled to be clear. In (Jaderberg et al., 2017), a gradient generator is proposed to correct local errors with the global BP signal, but it breaks the gradient isolation among local modules. In (Wang et al., 2021), a reconstruction loss is added to each local error, but the decoder increases the length of local BP and incurs more local parameters and memory cost. Different from these studies, our method reconciles the local errors successively without breaking gradient isolation or introducing learnable parameters, and can be applied in BP-free training. A previous work studies the convergence of layer-wise training (Shin, 2022), but the analysis is conducted in a linear model with global BP instead of local learning. Memory reduction can be also achieved by gradient checkpoint or reversible architecture (Chen et al., 2016; Gomez et al., 2017), but these strategies rely on the tradeoff between computation and memory. As a comparison, local learning can detach each layer so naturally brings memory efficiency.

3. Method

In Sec. 3.1, we analyze local learning and derive a convergence bound that shows its defect. Based on the result, we propose a method to remedy in Sec. 3.2, and show how to apply our method in local-BP training and BP-free training in Sec. 3.3. Finally, in Sec. 3.4, we use a simplified two-layer model to empirically showcase how our method facilitates the loss reduction of the output layer.

3.1. Theoretical Result

We consider a two-layer model composed of $\boldsymbol{x}_1 = f(\boldsymbol{x}_0, \boldsymbol{\theta}_1)$ and $\boldsymbol{x}_2 = f(\boldsymbol{x}_1, \boldsymbol{\theta}_2)$, where $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are the parameters of the two layers, respectively, $\boldsymbol{x}_0$ is the input data, $\boldsymbol{x}_1$ is the output of the first layer and also the input of the second layer, $\boldsymbol{x}_2$ is the output of the second layer, and f can be a composite function including linear and non-linear transformations. In layer-wise training, we have two local heads that produce local errors $\mathcal{L}_1(\boldsymbol{x}_1)$ and $\mathcal{L}_2(\boldsymbol{x}_2)$ to update the first layer and the second layer, respectively. Different from a global BP, the gradient of $\mathcal{L}_2$ does not back propagate into the first layer. Accordingly, the training in the i-th iteration can be formulated as:

$$\boldsymbol{\theta}_1^{(i+1)} \leftarrow \boldsymbol{\theta}_1^{(i)} - \eta_1^{(i)} \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1(\boldsymbol{\theta}_1^{(i)}), \tag{1}$$

$$\boldsymbol{\theta}_2^{(i+1)} \leftarrow \boldsymbol{\theta}_2^{(i)} - \eta_2^{(i)} \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1^{(i+1)}, \boldsymbol{\theta}_2^{(i)}), \tag{2}$$

where η_1, η_2 are the learning rates of the two layers. We analyze the convergence of the above local learning based on the PL condition (Karimi et al., 2016) and gradient Lipschitz. The assumptions and results are stated as following.

Assumption 3.1 (PL Condition)**.** Let $\mathcal{L}_2^*$ be the optimal function value of the second layer loss $\mathcal{L}_2$. There exists a μ such that $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, we have

$$\begin{aligned} \|\nabla \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\|^2 &= \|\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\|^2 + \|\nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\|^2 \\ &\geq \mu \left(\mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) - \mathcal{L}_2^*\right), \end{aligned}$$

where $\nabla \mathcal{L}_2 = [\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2; \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2]$.

Assumption 3.2 (Layer-wise Lipschitz and global Lipschitz)**.** There exists L_1, L_2, and $L > 0$ such that for all $\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,a}, \boldsymbol{\theta}_{2,b}, \boldsymbol{\theta}_2$, we have

$$\begin{aligned} \|\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_2)\| &\leq L_1 \|\boldsymbol{\theta}_{1,a} - \boldsymbol{\theta}_{1,b}\|, \\ \|\nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,a}) - \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,b})\| &\leq L_2 \|\boldsymbol{\theta}_{2,a} - \boldsymbol{\theta}_{2,b}\|, \end{aligned}$$

and

$$\|\nabla \mathcal{L}_2(\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_{2,a}) - \nabla \mathcal{L}_2(\boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_{2,b})\| \leq L \left\| \begin{bmatrix} \boldsymbol{\theta}_{1,a} - \boldsymbol{\theta}_{1,b} \\ \boldsymbol{\theta}_{2,a} - \boldsymbol{\theta}_{2,b} \end{bmatrix} \right\|.$$

Theorem 3.3. *Based on Assumptions 3.1 and 3.2, if the learning rates are set as $\eta_1^{(i)} = \eta_1$ and $\eta_2^{(i)} = \eta_2$, where*

$$\begin{cases} 0 < \eta_1 \leq \min\left(\frac{\sqrt{L_1^2 + 8L^2} - L_1}{4L^2}, \frac{2}{\mu}, \frac{1}{2L_2}\right) \\ \max\left(0, \frac{1 - \sqrt{1 - 2L_2\eta_1}}{L_2}\right) < \eta_2 \leq \frac{1 + \sqrt{1 - 2L_2\eta_1}}{L_2} \end{cases},$$

we have the following convergence

$$\mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* \leq (1 - \alpha\mu)\left(\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^*\right) + \alpha \left\|\boldsymbol{\epsilon}^{(i)}\right\|^2, \tag{3}$$

and recursively applying Eq. (3) we have

$$\begin{aligned} \mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* &\leq (1 - \alpha\mu)^{i+1}\left(\mathcal{L}_2^{(0,0)} - \mathcal{L}_2^*\right) \\ &\quad + \alpha \sum_{k=0}^{i} (1 - \alpha\mu)^k \left\|\boldsymbol{\epsilon}^{(i-k)}\right\|^2, \end{aligned} \tag{4}$$

where $\mathcal{L}_2^{(i,i)}$ denotes $\mathcal{L}_2(\boldsymbol{\theta}_1^{(i)}, \boldsymbol{\theta}_2^{(i)})$, i.e., *the second layer loss value with parameters $\boldsymbol{\theta}_1^{(i)}$ and $\boldsymbol{\theta}_2^{(i)}$ in the i-th iteration, $\alpha = \frac{\eta_1}{2}$, and $\boldsymbol{\epsilon}^{(i)} \triangleq \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2^{(i,i)} - \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1^{(i)}$.*

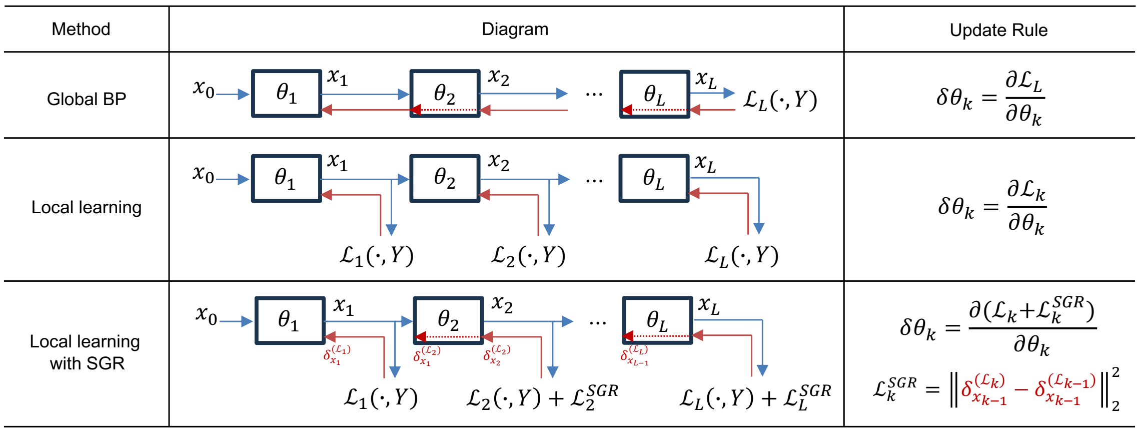

Method
Diagram
Update Rule
Global BP
$\delta\theta_k = \frac{\partial \mathcal{L}_L}{\partial \theta_k}$
Local learning
$\delta\theta_k = \frac{\partial \mathcal{L}_k}{\partial \theta_k}$
Local learning with SGR
$\delta\theta_k = \frac{\partial(\mathcal{L}_k + \mathcal{L}_k^{SGR})}{\partial \theta_k}$
$\mathcal{L}_k^{SGR} = \left\| \delta_{x_{k-1}}^{(\mathcal{L}_k)} - \delta_{x_{k-1}}^{(\mathcal{L}_{k-1})} \right\|_2^2$
$\mathcal{L}_1(\cdot, Y)$
$\mathcal{L}_2(\cdot, Y) + \mathcal{L}_2^{SGR}$
$\mathcal{L}_L(\cdot, Y) + \mathcal{L}_L^{SGR}$

Figure 1. An illustration to compare our method with non-greedy local learning and global BP. The blue arrows indicate forward propagation, while the red solid arrows and red dashed arrows denote the backward gradient *w.r.t.* the output feature and the input feature of each block, respectively. In global BP, gradients are passed into prior blocks to update the parameters, but the updates in local learning may be deviated by local errors. Our method successively reconciles local updates in a forward mode without breaking gradient isolation.

We care more about $\mathcal{L}_2$ because it is the loss of the last layer that is used to produce the representation for inference. Note that in local learning, the update of $\boldsymbol{\theta}_1$ in Eq. (1) is only dependent on $\mathcal{L}_1$ due to gradient isolation. The result in Eq. (3) indicates that when the local gradient $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1^{(i)}$ has a large distance with $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2^{(i,i)}$, *i.e.,* $\left\|\boldsymbol{\epsilon}^{(i)}\right\|$ is large, the optimization of the i-th iteration does not ensure a loss reduction of $\mathcal{L}_2$. Moreover, $\boldsymbol{\epsilon}$ has an accumulative effect during training, as shown in Eq. (4). As a result, only when $\|\boldsymbol{\epsilon}\|$ in every iteration keeps small, can we ensure that $\mathcal{L}_2$ is converged towards its optimality.

3.2. Successive Gradient Reconciliation

Even though learning by local update rules has been widely adopted in prior studies (Ren et al., 2023; Siddiqui et al., 2023; Frenkel et al., 2021; Belilovsky et al., 2020), our theoretical result in Theorem 3.3 implies that a defect of local learning, which may impede its convergence, lies in the reconciliation between local gradient and the global one. As a comparison, when training with global BP, the local gradient in Eq. (1), $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1^{(i)}$, is replaced by the global gradient, $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2^{(i,i)}$, via back propagation, which means $\|\boldsymbol{\epsilon}\| = 0$. Therefore, global BP is inherently spared from discordant updates. In (Jaderberg et al., 2017), a generator is learned to produce a synthetic local gradient close to the global one. But it relies on the global BP to provide the true gradient, so breaks gradient isolation.

We propose successive gradient reconciliation (SGR) for local learning. It is able to reconcile local updates without relying on the true global gradient or breaking gradient isolation. For any two neighboring layers, we have

$$\|\nabla_{\boldsymbol{\theta}_k} \mathcal{L}_{k+1} - \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_k\| \leq \left\|\boldsymbol{J}_f(\boldsymbol{\theta}_k)^T\right\| \left\| \delta_{\boldsymbol{x}_k}^{(\mathcal{L}_{k+1})} - \delta_{\boldsymbol{x}_k}^{(\mathcal{L}_k)} \right\|,$$

where $\boldsymbol{J}_f(\boldsymbol{\theta}_k)$ is the Jacobian matrix of the output feature $\boldsymbol{x}_k$ *w.r.t.* the parameters $\boldsymbol{\theta}_k$ in this layer, and $\delta_{\boldsymbol{x}_k}^{(\mathcal{L}_{k+1})}$ is the abbreviation for $\frac{\partial \mathcal{L}_{k+1}}{\partial \boldsymbol{x}_k}$, *i.e.,* the gradient *w.r.t.* $\boldsymbol{x}_k$ from the next layer's loss $\mathcal{L}_{k+1}$. Instead of directly minimizing the left side of the above equation, which needs to perform back propagation through two layers, we minimize the gradient distance from two neighboring local errors. After the update of the $(k-1)$-th layer, we store the gradient $\delta_{\boldsymbol{x}_{k-1}}^{(\mathcal{L}_{k-1})}$, and make the output feature $\boldsymbol{x}_{k-1}$ *detached but require gradient* being the input of the k-th layer. At the update of k-th layer $(k \geq 2)$, we optimize with the following objective:

$$\min_{\boldsymbol{\theta}_k} \quad \mathcal{L}_k + \lambda \cdot \mathcal{L}_k^{SGR}, \tag{5}$$

$$\mathcal{L}_k^{SGR} = \left\| \delta_{\boldsymbol{x}_{k-1}}^{(\mathcal{L}_k)} - \delta_{\boldsymbol{x}_{k-1}}^{(\mathcal{L}_{k-1})} \right\|_2^2, \tag{6}$$

where $\mathcal{L}_k$ is the local error for the k-th layer, *e.g.* the cross entropy loss, λ is a hyper-parameter, and $\mathcal{L}_k^{SGR}$ is an MSE between the two gradient terms. As shown in Figure 1, $\mathcal{L}_k^{SGR}$ is added on each layer (except the first layer) to successively reconcile neighboring local updates. The implementation is outlined in Algorithm 1 in the Appendix.

For each layer, it is a function with two dependent variables, the input $\boldsymbol{x}_{k-1}$ and the parameter $\boldsymbol{\theta}_k$. In local learning, the gradient $\delta_{\boldsymbol{x}_{k-1}}^{(\mathcal{L}_k)}$ cannot be passed into prior layers, and its change is decided by prior local errors, which is the cause of the discordant local updates. Our method can be interpreted as optimizing $\boldsymbol{\theta}_k$ such that it not only reduces the local error $\mathcal{L}_k$, but also enables the change of input caused by the previous layer, *i.e.,* $\delta_{\boldsymbol{x}_{k-1}}^{(\mathcal{L}_{k-1})}$, to reduce $\mathcal{L}_k$. By doing so, the local objectives are successively delivered into the last layer for a better output representation without breaking gradient isolation. Although the reconciliation concerned by our Theorem 3.3 is between the local gradient and the global

Table 1. Attribute comparison between our method (BP-free version) and the prior studies that also focus on the evils of BP.

Method	transport free	update unlocking	BP-free
Globap BP	✗	✗	✗
FA (Lillicrap et al., 2016)	✓	✗	✗
DFA (Nøkland, 2016)	✓	partially	✓
PEPITA (Dellaferrera & Kreiman, 2022)	✓	partially	✓
DRTP (Frenkel et al., 2021)	✓	✓	✓
Ours (BP-free)	✓	✓	✓

true gradient, while our method only deals with neighboring local gradients, the following proposition indicates that both $\|\boldsymbol{\epsilon}\| \to 0$ for all local layers and $\mathcal{L}_k^{SGR} \to 0, \forall k \geq 2$, lead to an equivalence to global BP.

Proposition 3.4. *For a model composed of L local layers parameterized by $\boldsymbol{\theta}_k$ with their local errors $\mathcal{L}_k$, $k = 1, ..., L$, when $\mathcal{L}_k^{SGR} = 0$ for all layers $2 \leq k \leq L$ for a batch of data, we have*

$$\nabla_{\boldsymbol{\theta}_k} \mathcal{L}_L = \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_k, \quad \forall 1 \leq k \leq L-1,$$

which implies that all local updates are equivalent to learning with the global true gradient back propagated from the last-layer error $\mathcal{L}_L$.

3.3. Local-BP Training and BP-free Training

We apply our method in both local-BP and BP-free setups. In local-BP training, a model is divided into multiple layers and each layer can be a stack of multiple blocks. Besides, the local classifier head can also have more than one layer, which is proved to be effective in improving the performance of layer-wise training (Belilovsky et al., 2019). Therefore, there is still back propagation within the classifier head and the backbone. We adopt the auxiliary classifier head design following (Belilovsky et al., 2019; Wang et al., 2021), and add our $\mathcal{L}_k^{(SGR)}$ on all the layers $k \geq 2$. The local error $\mathcal{L}_k$ can be both the loss functions for supervised learning and self-supervised learning.

In BP-free training, the gradient to each layer's output, $\delta \boldsymbol{x}_k$, is based on an analytical update rule, and each layer is only a basic block composed of a linear transformation and a non-linear activation. Therefore, there is no back propagation through multiple layers. A comparison between several BP-free methods for supervised learning is shown in Table 1. FA (Lillicrap et al., 2016) uses a random matrix to replace the transport of weight matrix but still relies on BP. DFA (Nøkland, 2016) directly projects the last-layer gradient into each layer via random matrices, so spares the need of BP. But local updates cannot be performed until a full forward propagation is finished, which means that update unlocking is only partially solved. PEPITA (Dellaferrera & Kreiman, 2022) improves over forward-forward learning (Hinton, 2022) and needs a second forward propagation. As a result, update unlocking is also partially solved.

DRTP directly acquires $\delta \boldsymbol{x}_k$ by projecting labels with a random matrix. Our method in the BP-free setup also adopts a fixed matrix but with a particular structure named equiangular tight frame (ETF), which has the maximal equiangular separation (Papyan et al., 2020), and can be formulated as,

$$\boldsymbol{m}_{k_1}^T \boldsymbol{m}_{k_2} = \begin{cases} 1, & k_1 = k_2 \\ -\frac{1}{K-1}, & k_1 \neq k_2 \end{cases} \tag{7}$$

where $\boldsymbol{m}_{k_1}$ is the classifier vector for class k_1 and K is the number of total classes. It has been observed that the intermediate layers in a neural network gradually increase the separability among different classes (He & Su, 2023), and using an ETF structure as the last-layer classifier head does not harm representation learning (Zhu et al., 2021; Yang et al., 2022a; 2023; Zhong et al., 2023; Du et al., 2023). Inspired by these studies, we initialize and fix an ETF classifier for each local layer to calculate the cross entropy error. It enables the gradient $\delta \boldsymbol{x}_k$ to better induce separability, and our $\mathcal{L}_k^{SGR}$ helps to carry the local separability into the last layer. In this way, our method keeps the benefits of DRTP but achieves significantly better performance than the methods in Table 1, as will be shown in experiments.

3.4. Justification

Although our method adds a regularization on the classification loss $\mathcal{L}_k$, which may degrade the original optimality of $\mathcal{L}_k$ if there is only one layer, the following proposition based on a linear two-layer model shows that training with our $\mathcal{L}_2^{SGR}$ loss on the second layer helps to induce a larger reduction of the classification loss $\mathcal{L}_2$ in inference.

Proposition 3.5. *Consider a two-layer linear model composed of $\boldsymbol{x}_1 = \boldsymbol{\theta}_1 \boldsymbol{x}_0$ and $\boldsymbol{x}_2 = \boldsymbol{\theta}_2 \boldsymbol{x}_1$, where $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are the learnable linear matrices, and $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$ are the input and output of the model, respectively. We use fixed ETF structures $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ for the local classifier heads and the cross entropy (CE) loss for local errors $\mathcal{L}_1(\boldsymbol{M}_1 \boldsymbol{x}_1, y)$ and $\mathcal{L}_2(\boldsymbol{M}_2 \boldsymbol{x}_2, y)$. Denote $\mathcal{L}_2'$ as the CE loss value in inference after performing one step of gradient descent of $\mathcal{L}_1$ and $\mathcal{L}_2$ by Eq. (1) and (2), and denote $\hat{\mathcal{L}}_2'$ as the one with our $\mathcal{L}_2^{SGR}$ on the second layer. Assume that the prediction logit for the ground truth label in the second layer is larger than the one in the first layer, we have $\hat{\mathcal{L}}_2' \leq \mathcal{L}_2'$, when $\mathcal{L}_2^{SGR}$ is small.*

The result empirically supports that although training with our regularization $\mathcal{L}_2^{SGR}$ will shift the update direction of $\boldsymbol{\theta}_2$ from the steepest descent of $\mathcal{L}_2$, it helps to look for a position of $\boldsymbol{\theta}_2$ such that the change of $\boldsymbol{x}_1$ caused by the previous local error also enables to minimize $\mathcal{L}_2$, which can lead to a larger loss reduction than only optimizing $\boldsymbol{\theta}_2$.

Table 2. The results (top-1 accuracy) of local training with ResNet-18 on ImageNet. The "Greedy" result is our implementation following (Belilovsky et al., 2019). All models are divided into 4 local modules according to feature spatial resolution in ResNet-18.

Greedy	non-greedy	w/ SGR				
(Belilovsky et al., 2019)		$\lambda = 1000$	$\lambda = 2000$	$\lambda = 3000$	$\lambda = 5000$	$\lambda = 7000$
63.65	69.44	70.32	70.53	70.62	70.73	70.69

Table 3. The results of local training with ResNet-32 and PlainNet on CIFAR-10. Each residual or PlainNet block is a local module. The local classifier is composed of a convolution layer and a linear layer. "reforward" denotes forward again for each local update to use the updated output as the input of the next layer.

Method	ResNet	PlainNet
layer-wise	84.49±0.36	80.66±0.42
w/ reforward	84.38±0.52	80.74±0.44
w/ SGR (ours)	**85.65±0.38**	**81.67±0.29**

Table 4. BP-free training with PlainNet on CIFAR-10 and CIFAR-100. Each linear-nonlinear transformation in PlainNet is a local module. The local classifier is a fixed matrix as Eq. (7). The 2nd row results are derived from (Dellaferrera & Kreiman, 2022).

Method	BP-free	CIFAR-10	CIFAR100
FA	✗	57.51±0.57	27.15±0.53
DRTP	✓	50.53±0.81	20.14±0.68
PEPITA	✓	56.33±1.35	27.56±0.60
layer-wise	✓	69.17±0.91	48.30±0.64
w/ reforward	✓	69.05±0.63	47.12±0.55
w/ SGR (ours)	✓	**72.40±0.75**	**49.41±0.44**

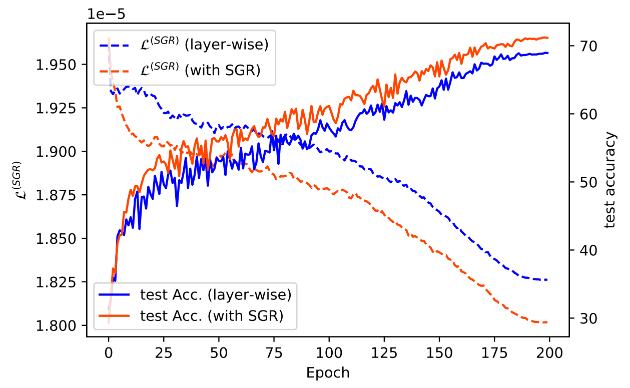


Figure 2. The training curves of average $\mathcal{L}^{(SGR)}$ of all neighboring layers and accuracy on test set.

4. Experiments

In experiments, we test the effectiveness of our proposed successive gradient reconciliation (SGR), and demonstrate that it is able to significantly save memory consumption of CNN and Transformer architectures, and surpass previous studies on CIFAR-10, CIFAR-100, and ImageNet datasets. The implementation details are provided in Appendix E.

4.1. Ablation Studies

We ablate the hyper-parameter λ in Eq. (5) and integrate our SGR on non-greedy layer-wise training to show its ability to improve performance in both local-BP and BP-free cases.

We add our $\mathcal{L}^{(SGR)}$ as defined in Eq. (6) with different λ in layer-wise training of ResNet-18 (He et al., 2016) on ImageNet. As shown in Table 2, most choices of λ achieve more than 1% performance improvement over non-greedy layer-wise training. These results with different λ do not have a large deviation, which indicates that our method has stable performance and is not sensitive to the choice of λ. As λ increases from 1000 to 5000, the accuracy slightly gets better, showing that a stronger reconciliation of local updates within a proper range can lead to a better performance.

In local-BP training, we use ResNet-32 and PlainNet that removes the identity connection of ResNet. Each local module may contain multiple layers and the local classifier is composed of a convolution layer and a linear layer, so there is still BP within each local update. As shown in Table 3, when armed with our method, layer-wise training attains more than 1% accuracy improvement with both ResNet and PlainNet. The reconciliation of local updates can be also naively achieved by a second forward propagation after the local update of each layer such that the input of the next layer is based on the updated output. As shown in Tables 3 and 4, this practice does not bring obvious performance gain because it performs too many steps of optimization for each batch of data and thus overfits in each iteration.

In BP-free training, we use PlainNet where each block is only a linear-nonlinear transformation and the local classifier is a fixed ETF structure as defined in Eq. (7). Therefore, the gradient *w.r.t.* output feature of each block can be analytically derived and there is no BP within each local module. As shown in Table 4, our method improves by more than 3% on CIFAR-10 and more than 1% on CIFAR-100. As shown in Figure 2, when our method is adopted, the average loss value of $\mathcal{L}^{(SGR)}$ is significantly lower than the baseline without our method, and accordingly, the test accuracy is

Table 5. Results on ImageNet with different architectures. Experiments are conducted on 8 NVIDIA A100 GPUs. "Train time" is the whole training time of each experiment. "Memory" is the memory consumption per GPU for training tested with a batchsize of 256. Experiments of the same row are conducted with the same training setting for fair comparison. K is the number of local modules.

Network	Method	K	top-1 Acc. (%)	top-5 Acc. (%)	Train time (hour)	Memory (GB)
ResNet-50	Global BP	-	76.55	93.06	7.0	21.49
	InfoPro (Wang et al., 2021)	2	76.23	92.92	10.8	21.59 (↑0.4%)
	SGR (ours)	2	76.35	93.03	17.1	17.91 (↓16.6%)
	InfoPro (Wang et al., 2021)	3	75.33	92.57	21.2	18.73 (↓12.8%)
	SGR (ours)	3	75.42	92.57	15.1	**15.34 (↓28.6%)**
ResNet-50 (self-supervised)	Global BP	-	71.50	-	73.0	45.09
	(Siddiqui et al., 2023)	4	70.48	-	140.9	70.53 (↑56.4%)
	SGR (ours)	2	70.30	89.22	98.6	**34.86 (↓22.7%)**
ResNet-101	Global BP	-	77.97	94.06	11.0	31.70
	InfoPro (Wang et al., 2021)	2	77.61	93.78	18.6	26.14 (↓17.6%)
	SGR (ours)	2	77.69	93.84	28.5	22.99 (↓27.5%)
	InfoPro (Wang et al., 2021)	3	77.02	93.47	24.3	21.87 (↓31.0%)
	SGR (ours)	3	77.02	93.24	22.9	**18.18 (↓42.6%)**
VIT-small	Global BP	-	79.40	94.11	52.2	20.70
	InfoPro (Wang et al., 2021)	3	78.15	94.00	61.8	14.96 (↓27.7%)
	SGR (ours)	3	78.65	94.03	62.5	**11.73 (↓43.3%)**
Swin-tiny	Global BP	-	81.18	95.61	43.8	26.78
	InfoPro (Wang et al., 2021)	2	79.38	94.14	53.2	23.34 (↓12.9%)
	SGR (ours)	3	80.05	94.95	56.3	**15.83 (↓40.9%)**
Swin-small	Global BP	-	83.02	96.29	75.9	42.60
	InfoPro (Wang et al., 2021)	2	81.19	95.00	86.6	33.08 (↓22.4%)
	SGR (ours)	3	81.64	95.45	90.8	**22.83 (↓46.4%)**

better throughout training. It reveals that gradient reconciliation among local updates, which our method targets, is correlated with the performance of local learning.

4.2. Results on ImageNet

Our method is able to help CNN and Transformer (Dosovitskiy et al., 2021; Liu et al., 2021) architectures save a significant proportion of memory consumption on ImageNet, while achieving competitive performance with global BP. As shown in Table 5, our method on ResNet-50 has better performances than (Wang et al., 2021) and a similar performance to (Siddiqui et al., 2023) with larger memory conservation than both methods in supervised and self-supervised learning, respectively. When $K = 3$, our method trains faster with lower memory consumption than InfoPro on both ResNet-50 and ResNet-101. On the three Transformer-based architectures, our method saves more than 40% memory without inducing a large increment of train time or unbearable performance loss (more than 1.5%) compared with global BP, and achieves better performances than InfoPro with close train time overhead.

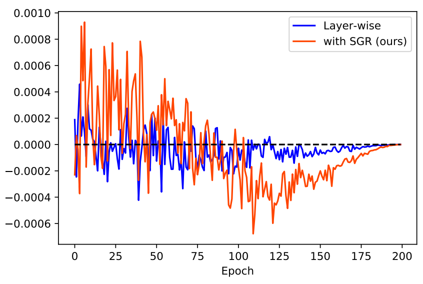
4.3. Results on CIFAR

We verify the performance of our method using ResNet-32 on CIFAR-10 and CIFAR-100 and compare with greedy layer-wise training (Belilovsky et al., 2019), DGL (Belilovsky et al., 2020), and InfoPro (Wang et al., 2021). Following (Belilovsky et al., 2019), we adopt a local classifier composed of a convolution layer and a linear layer for classification. As shown in Table 6, our SGR surpasses all compared methods in most cases. Especially when K=2, we achieve a competitive performance with global BP (our 92.91% v.s. BP 92.82% on CIFAR-10, and our 74.63% v.s. BP 74.78% on CIFAR-100).

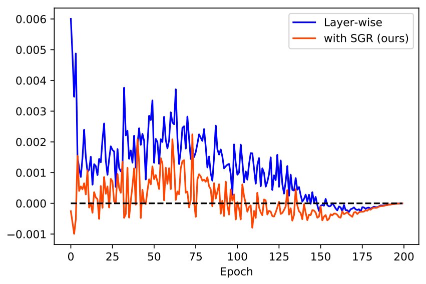
Since local learning spares the effort of a global BP, it will not suffer from the notorious gradient vanishing and explosion problems that are easy to emerge in training deep neural networks such as the VGG architecture (Simonyan & Zisserman, 2015). As shown in Table 7, we compare the performances of our method and training with global BP in VGG-Net of different depth. As the depth goes larger, the performance of global BP decreases sharply until a failed

Table 6. Local learning results with ResNet-32 on CIFAR-10 and CIFAR-100 and comparison with prior studies. K is the number of local modules divided from ResNet-32. K=16 refers to the case where each residual block including the stem layer correspond to one local module. K=3 divides the model according to feature spatial resolution. $K = 2$ leaves the last 5 blocks as the second local module.

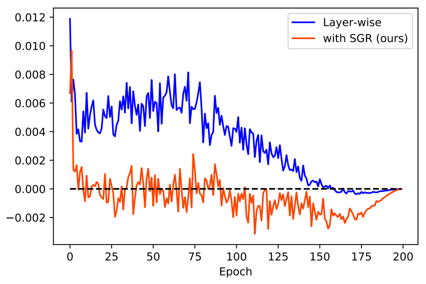
Method	CIFAR-10 (BP: 92.82±0.22)			CIFAR-100 (BP: 74.78±0.31)		
	K=16	K=3	K=2	K=16	K=3	K=2
Greedy (Belilovsky et al., 2019)	76.96±0.54	85.12±0.35	90.06±0.43	48.77±0.41	63.55±0.32	70,68±0.38
DGL (Belilovsky et al., 2020)	84.01±0.40	87.61±0.51	91.05±0.27	63.31±0.28	70,59±0.39	72.34±0.31
InfoPro (Wang et al., 2021)	**85.69±0.47**	90.43±0.36	91.74±0.29	66.27±0.44	71.44±0.24	73.59±0.25
SGR (ours)	85.65±0.38	**91.34±0.45**	**92.91±0.36**	**66.61±0.31**	**72.15±0.27**	**74.63±0.20**



(a) Layer-2



(b) Layer-3



(c) Layer-4

Figure 3. We measure the change of classification loss in each layer caused by the input update from prior layers using a 4-layer PlainNet. The black dashed line denotes the zero baseline such that the area below it means that the updates of prior layers can produce a new output feature enabling to reduce the loss value of the current layer as its input.

Table 7. Results of VGG-Net with different depth on CIFAR-10.

Method	d=12	d=17	d=27	d=37	d=47	d=57
Global BP	90.76	90.95	90.46	78.47	37.68	fail
SGR (ours)	84.81	86.38	86.95	86.09	85.30	85.51

convergence. In contrast, our SGR is able to keep a stable performance even in a very deep network.

4.4. Analysis

In traditional local learning, local updates are not reconciled, so the updates of prior layers do not necessarily change the output feature towards a direction that minimizes the current local loss. In this subsection, we investigate whether our SGR helps to remedy this defect. We measure the classification loss change in one layer caused by the update of its input feature, *i.e.,* $\Delta\mathcal{L}_k = \mathcal{L}_k(\boldsymbol{x}_{k-1}^{(t+1)}) - \mathcal{L}_k(\boldsymbol{x}_{k-1}^{(t)})$, where $\boldsymbol{x}_{k-1}^{(t)}$ is the input feature of this local layer in the t-th iteration and $\boldsymbol{x}_{k-1}^{(t+1)}$ is the new one after the update of all its previous layers $1, .., k-1$.

As shown in Figure 3, for a 4-layer PlainNet, $\Delta\mathcal{L}_2$ in the 2nd layer with and without our method are both surrounding zero. In deeper layers, the scale of $\Delta\mathcal{L}$ without our method (the blue one) is growing larger, which indicates that the discordant local updates have an accumulative effect on deep layers. Besides, the curves without our method in the 3-rd and 4-th layers are above zero most of the time, implying that the prior local updates contribute little to the deep layers' learning. As a comparison, our method does not grow the scale of $\Delta\mathcal{L}$ apparently in deep layers. Particularly in the 4-th layer, the curve with our method is below zero in most epochs, which means the first 3 layers help to produce a better feature as the input of the last layer, and is in line with our better performance observed in Figure 2.

5. Conclusion

In this paper, we point out a fundamental defect of local learning that global BP is naturally immune to. Our theoretical result indicates that the convergence of local learning cannot be assured when the local updates are not reconciled. Based on the result, we propose a method, named successive gradient reconciliation, to successively reconcile local updates without breaking gradient isolation or introducing any learnable parameters. Our method can be applied to both local-BP and BP-free local learning. Experimental results demonstrate that our method surpasses previous methods and is able to achieve a comparable performance with global BP saving more than 40% memory consumption for CNN and Transformer architectures. Future studies may explore applying our method to large model finetuning.

Impact Statement

Our method reduces memory consumption in training neural network, making machine learning models more friendly in resource-limited environments. Furthermore, by aligning more closely with biological learning processes, our method may contribute to the intersection of AI and neuroscience, potentially leading to developments of more biologically plausible AI systems. The main goal of this paper is to advance the field of machine learning. There is no ethic impact that we feel must be specifically highlighted here.

Acknowledgements

This work was supported by KAUST-Oxford CRG Grant Number DFR07910, the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research (OSR) under Award No. OSR-CRG2021-4648, SDAIA-KAUST Center of Excellence in Data Science, Artificial Intelligence (SDAIA-KAUST AI), and a UKRI grant Turing AI Fellowship (EP/W002981/1).

References

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., and Tweed, D. B. Deep learning without weight transport. *NeurIPS*, 32, 2019.

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *NeurIPS*, 31, 2018.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

Beck, A. and Tetruashvili, L. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet. In *ICML*, pp. 583–593. PMLR, 2019.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of cnns. In *ICML*, pp. 736–745. PMLR, 2020.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. *NeurIPS*, 19, 2006.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Clark, D., Abbott, L., and Chung, S. Credit assignment through broadcasting a global error vector. *NeurIPS*, 34: 10053–10066, 2021.

Crick, F. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

Dellaferrera, G. and Kreiman, G. Error-driven input modulation: solving the credit assignment problem without a backward pass. In *ICML*, pp. 4937–4955. PMLR, 2022.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *arXiv preprint arXiv:2010.11929*, 2021.

Du, Y., Yang, Y., Tao, D., and Hsieh, M.-H. Problem-dependent power of quantum neural networks on multiclass classification. *Physical Review Letters*, 131(14): 140601, 2023.

Fournier, L., Patel, A., Eickenberg, M., Oyallon, E., and Belilovsky, E. Preventing dimensional collapse in contrastive local learning with subsampling. In *ICML 2023 Workshop on Localized Learning (LLW)*, 2023.

Frenkel, C., Lefebvre, M., and Bol, D. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in neuroscience*, 15:629892, 2021.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. *NeurIPS*, 30, 2017.

Grossberg, S. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1): 23–63, 1987.

Halvagal, M. S. and Zenke, F. The combination of hebbian and predictive plasticity learns invariant object representations in deep sensory networks. *Nature Neuroscience*, pp. 1–10, 2023.

He, H. and Su, W. J. A law of data separation in deep learning. *Proceedings of the National Academy of Sciences*, 120(36):e2221704120, 2023.

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

Hinton, G. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18 (7):1527–1554, 2006.

Illing, B., Ventura, J., Bellec, G., and Gerstner, W. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. *NeurIPS*, 34:30365–30379, 2021.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. In *ICML*, pp. 1627–1635. PMLR, 2017.

Journé, A., Rodriguez, H. G., Guo, Q., and Moraitis, T. Hebbian deep learning without feedback. In *ICLR*, 2023.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*, pp. 795–811. Springer, 2016.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.

Li, J., Xiao, M., Fang, C., Dai, Y., Xu, C., and Lin, Z. Training neural networks by lifted proximal operator machines. *TPAMI*, 44(6):3334–3348, 2020.

Liao, Q., Leibo, J., and Poggio, T. How important is weight symmetry in backpropagation? In *AAAI*, volume 30, 2016.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pp. 10012–10022, 2021.

Löwe, S., O'Connor, P., and Veeling, B. Putting an end to end-to-end: Gradient-isolated learning of representations. *NeurIPS*, 32, 2019.

Mostafa, H., Ramesh, V., and Cauwenberghs, G. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

Nøkland, A. Direct feedback alignment provides learning in deep neural networks. *NeurIPS*, 29, 2016.

Nøkland, A. and Eidnes, L. H. Training neural networks with local error signals. In *ICML*, pp. 4839–4850. PMLR, 2019.

Papyan, V., Han, X., and Donoho, D. L. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. Scaling forward gradient with local losses. In *ICLR*, 2023.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Shin, Y. Effects of depth, width, and initialization: A convergence analysis of layer-wise training for deep linear neural networks. *Analysis and Applications*, 20(01):73–119, 2022.

Siddiqui, S. A., Krueger, D., LeCun, Y., and Deny, S. Blockwise self-supervised learning at scale. *arXiv preprint arXiv:2302.01647*, 2023.

Silver, D., Goyal, A., Danihelka, I., Hessel, M., and van Hasselt, H. Learning by directional gradient descent. In *ICLR*, 2022.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Song, Y., Millidge, B., Salvatori, T., Lukasiewicz, T., Xu, Z., and Bogacz, R. Inferring neural activity before plasticity as a foundation for learning beyond backpropagation. *Nature Neuroscience*, pp. 1–11, 2024.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. Training neural networks without gradients: A scalable admm approach. In *ICML*, pp. 2722–2731. PMLR, 2016.

Wang, Y., Ni, Z., Song, S., Yang, L., and Huang, G. Revisiting locally supervised learning: an alternative to end-to-end training. In *ICLR*, 2021.

Xiao, W., Chen, H., Liao, Q., and Poggio, T. Biologically-plausible learning algorithms can scale to large datasets. In *ICLR*, 2019.

Xiong, Y., Ren, M., and Urtasun, R. Loco: Local contrastive representation learning. *NeurIPS*, 33:11142–11153, 2020.

Yang, Y., Chen, S., Li, X., Xie, L., Lin, Z., and Tao, D. Inducing neural collapse in imbalanced learning: Do we really need a learnable classifier at the end of deep neural network? *NeurIPS*, 35:37991–38002, 2022a.

Yang, Y., Wang, H., Yuan, H., and Lin, Z. Towards theoretically inspired neural initialization optimization. In *NeurIPS*, volume 35, pp. 18983–18995, 2022b.

Yang, Y., Yuan, H., Li, X., Lin, Z., Torr, P., and Tao, D. Neural collapse inspired feature-classifier alignment for few-shot class-incremental learning. In *ICLR*, 2023.

You, Y., Chen, T., Wang, Z., and Shen, Y. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *CVPR*, pp. 2127–2135, 2020.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. Barlow twins: Self-supervised learning via redundancy reduction. In *ICML*, pp. 12310–12320. PMLR, 2021.

Zhong, Z., Cui, J., Yang, Y., Wu, X., Qi, X., Zhang, X., and Jia, J. Understanding imbalanced semantic segmentation through neural collapse. In *CVPR*, pp. 19550–19560, 2023.

Zhu, Z., Ding, T., Zhou, J., Li, X., You, C., Sulam, J., and Qu, Q. A geometric analysis of neural collapse with unconstrained features. *NeurIPS*, 34:29820–29834, 2021.

A. Proof of Theorem 3.3

We restate the assumptions and results, and then provide the proof.

Assumption A.1 (PL Condition)**.** Let $\mathcal{L}_2^*$ be the optimal function value of the second layer loss $\mathcal{L}_2$. There exists a μ such that $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, we have:

$$\begin{aligned}\left\|\nabla \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\right\|^2 &= \left\|\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\right\|^2 + \left\|\nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\right\|^2 \\ &\geq \mu \left(\mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) - \mathcal{L}_2^*\right),\end{aligned}$$

where $\nabla \mathcal{L}_2 = [\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2; \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2]$.

Assumption A.2 (Layer-wise Lipschitz and global Lipschitz)**.** There exists L_1, L_2, and $L > 0$ such that for all $\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,a}, \boldsymbol{\theta}_{2,b}, \boldsymbol{\theta}_2$, we have:

$$\begin{aligned}\left\|\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_2)\right\| &\leq L_1 \left\|\boldsymbol{\theta}_{1,a} - \boldsymbol{\theta}_{1,b}\right\|, \\ \left\|\nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,a}) - \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{2,b})\right\| &\leq L_2 \left\|\boldsymbol{\theta}_{2,a} - \boldsymbol{\theta}_{2,b}\right\|,\end{aligned}$$

and

$$\left\|\nabla \mathcal{L}_2(\boldsymbol{\theta}_{1,a}, \boldsymbol{\theta}_{2,a}) - \nabla \mathcal{L}_2(\boldsymbol{\theta}_{1,b}, \boldsymbol{\theta}_{2,b})\right\| \leq L \left\| \begin{bmatrix} \boldsymbol{\theta}_{1,a} - \boldsymbol{\theta}_{1,b} \\ \boldsymbol{\theta}_{2,a} - \boldsymbol{\theta}_{2,b} \end{bmatrix} \right\|.$$

Theorem A.3. *Based on Assumptions 3.1 and 3.2, if the learning rates are set as* $\eta_1^{(i)} = \eta_1$ *and* $\eta_2^{(i)} = \eta_2$*, where*

$$\begin{cases} 0 < \eta_1 \leq \min\left(\frac{\sqrt{L_1^2 + 8L^2} - L_1}{4L^2}, \frac{2}{\mu}, \frac{1}{2L_2}\right) \\ \max\left(0, \frac{1 - \sqrt{1 - 2L_2\eta_1}}{L_2}\right) < \eta_2 \leq \frac{1 + \sqrt{1 - 2L_2\eta_1}}{L_2} \end{cases},$$

we have the following convergence

$$\mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* \leq (1 - \alpha\mu)\left(\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^*\right) + \alpha \left\|\boldsymbol{\epsilon}^{(i)}\right\|^2, \tag{8}$$

and recursively applying Eq. (8) we have

$$\begin{aligned}\mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* \leq{} & (1 - \alpha\mu)^{i+1}\left(\mathcal{L}_2^{(0,0)} - \mathcal{L}_2^*\right) \\ & + \alpha \sum_{k=0}^{i} (1 - \alpha\mu)^k \left\|\boldsymbol{\epsilon}^{(i-k)}\right\|^2,\end{aligned} \tag{9}$$

where $\mathcal{L}_2^{(i,i)}$ *denotes* $\mathcal{L}_2(\boldsymbol{\theta}_1^{(i)}, \boldsymbol{\theta}_2^{(i)})$, i.e., *the second layer loss value with parameters* $\boldsymbol{\theta}_1^{(i)}$ *and* $\boldsymbol{\theta}_2^{(i)}$ *in the* i*-th iteration,* $\alpha = \frac{\eta_1}{2}$*, and* $\boldsymbol{\epsilon}^{(i)} \triangleq \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2^{(i,i)} - \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1^{(i)}$.

Lemma A.4. *With assumption A.1, we have*

$$\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^{(i+1,i+1)} \geq \eta_2^{(i)}\left(1 - \frac{L_2}{2}\eta_2^{(i)}\right)\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \eta_1^{(i)} \nabla_1^T \mathcal{L}_1^{(i)} \boldsymbol{\epsilon}^{(i)} + \eta_1^{(i)}\left(1 - \frac{L_1}{2}\eta_1^{(i)}\right)\left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2, \tag{10}$$

where $\boldsymbol{\epsilon}^{(i)} = \nabla_1 \mathcal{L}_2^{(i,i)} - \nabla_1 \mathcal{L}_1^{(i)}$, $\nabla_1 \mathcal{L}_2^{(i,i)}$ *is the abbreviation of* $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2(\boldsymbol{\theta}_1^{(i)}, \boldsymbol{\theta}_2^{(i)})$*, and* $\nabla_1 \mathcal{L}_1^{(i)}$ *refers to* $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1(\boldsymbol{\theta}_1^{(i)})$.

Lemma A.5. *With assumption A.1, we have*

$$\left\|\nabla \mathcal{L}_2^{(i,i)}\right\|^2 - \left\|\boldsymbol{\epsilon}^{(i)}\right\|^2 \leq \left(2\left(L\eta_1^{(i)}\right)^2 + 1\right)\left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + 2\nabla_1^T \mathcal{L}_1^{(i)} \boldsymbol{\epsilon}^{(i)}, \tag{11}$$

where $\boldsymbol{\epsilon}^{(i)} = \nabla_1 \mathcal{L}_2^{(i,i)} - \nabla_1 \mathcal{L}_1^{(i)}$*, and* $\nabla \mathcal{L}_2^{(i,i)} = [\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_2^{(i,i)}; \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2^{(i,i)}]$.

Proof of Lemma A.4. The layer wise training based on Eq. (1) and (2) is restated as follows:

$$\boldsymbol{\theta}_1^{(i+1)} \leftarrow \boldsymbol{\theta}_1^{(i)} - \eta_1^{(i)} \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1(\boldsymbol{\theta}_1^{(i)}),$$
$$\boldsymbol{\theta}_2^{(i+1)} \leftarrow \boldsymbol{\theta}_2^{(i)} - \eta_2^{(i)} \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\boldsymbol{\theta}_1^{(i+1)}, \boldsymbol{\theta}_2^{(i)}).$$

Based on descent lemma (Beck & Tetruashvili, 2013) and Assumption A.2, we have

$$\begin{aligned} \mathcal{L}_2^{(i+1,i+1)} &\leq \mathcal{L}_2^{(i+1,i)} + \nabla_2^T \mathcal{L}_2^{(i+1,i)} \left(\boldsymbol{\theta}_2^{(i+1)} - \boldsymbol{\theta}_2^{(i)}\right) + \frac{L_2}{2} \left\|\boldsymbol{\theta}_2^{(i+1)} - \boldsymbol{\theta}_2^{(i)}\right\|^2 \\ &= \mathcal{L}_2^{(i+1,i)} - \eta_2^{(i)} \left(1 - \frac{L_2}{2}\eta_2^{(i)}\right) \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2, \end{aligned} \tag{12}$$

and

$$\begin{aligned} \mathcal{L}_2^{(i+1,i)} &\leq \mathcal{L}_2^{(i,i)} + \nabla_1^T \mathcal{L}_2^{(i,i)} \left(\boldsymbol{\theta}_1^{(i+1)} - \boldsymbol{\theta}_1^{(i)}\right) + \frac{L_1}{2} \left\|\boldsymbol{\theta}_1^{(i+1)} - \boldsymbol{\theta}_1^{(i)}\right\|^2 \\ &= \mathcal{L}_2^{(i,i)} - \eta_1^{(i)} \nabla_1^T \mathcal{L}_2^{(i,i)} \nabla_1 \mathcal{L}_1^{(i)} + \frac{L_1}{2} (\eta_1^{(i)})^2 \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2. \end{aligned} \tag{13}$$

Sum up (12) and (13) to obtain

$$\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^{(i+1,i+1)} \geq \eta_2^{(i)} \left(1 - \frac{L_2}{2}\eta_2^{(i)}\right) \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \eta_1^{(i)} \nabla_1^T \mathcal{L}_2^{(i,i)} \nabla_1 \mathcal{L}_1^{(i)} - \frac{L_1}{2} (\eta_1^{(i)})^2 \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2.$$

Then let $\nabla_1 \mathcal{L}_2^{(i,i)} = \boldsymbol{\epsilon}^i + \nabla_1 \mathcal{L}_1^{(i)}$:

$$\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^{(i+1,i+1)} \geq \eta_2^{(i)} \left(1 - \frac{L_2}{2}\eta_2^{(i)}\right) \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \eta_1^{(i)} \nabla_1^T \mathcal{L}_1^{(i)} \boldsymbol{\epsilon}^i + \eta_1^{(i)} \left(1 - \frac{L_1}{2}(\eta_1^{(i)})\right) \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2,$$

which concludes the proof of Lemma A.4. □

Proof of Lemma A.5.

$$\begin{aligned} \left\|\nabla \mathcal{L}_2^{(i,i)}\right\|^2 &= \left\|\nabla_2 \mathcal{L}_2^{(i,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &\leq \left(\left\|\nabla_2 \mathcal{L}_2^{(i,i)} - \nabla_2 \mathcal{L}_2^{(i+1,i)}\right\| + \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|\right)^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &= \left\|\nabla_2 \mathcal{L}_2^{(i,i)} - \nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i,i)} - \nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &\leq 2\left(\left\|\nabla_2 \mathcal{L}_2^{(i,i)} - \nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2\right) + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &\leq 2\left(\left\|\nabla_2 \mathcal{L}_2^{(i,i)} - \nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)} - \nabla_1 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2\right) + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &= 2\left(\left\|\nabla \mathcal{L}_2^{(i,i)} - \nabla \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2\right) + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &\leq 2L^2\left\|\boldsymbol{\theta}_1^{(i)} - \boldsymbol{\theta}_1^{(i+1)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2 \\ &\leq 2\left(L\eta_1^{(i)}\right)^2 \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)}\right\|^2. \end{aligned}$$

Since $\nabla_1 \mathcal{L}_2^{(i,i)} = \boldsymbol{\epsilon}^{(i)} + \nabla_1 \mathcal{L}_1^{(i)}$, we have

$$\begin{aligned} &\left\|\nabla \mathcal{L}_2^{(i,i)}\right\|^2 \\ \leq& 2\left(L\eta_1^{(i)}\right)^2 \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)} - \nabla_1 \mathcal{L}_1^{(i)} + \nabla_1 \mathcal{L}_1^{(i)}\right\|^2 \\ =& 2\left(L\eta_1^{(i)}\right)^2 \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_2^{(i,i)} - \nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left(\nabla_1 \mathcal{L}_2^{i,i} - \nabla_1 \mathcal{L}_1^{(i)}\right)^T \nabla_1 \mathcal{L}_1^{(i)} \\ =& \left(2\left(L\eta_1^{(i)}\right)^2 + 1\right) \left\|\nabla_1 \mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2 \mathcal{L}_2^{(i+1,i)}\right\|^2 + \left\|\boldsymbol{\epsilon}^{(i)}\right\|^2 + 2\nabla_1^T \mathcal{L}_1^{(i)} \boldsymbol{\epsilon}^{(i)}. \end{aligned}$$

Therefore, we have,

$$\left\|\nabla\mathcal{L}_2^{(i,i)}\right\|^2 - \left\|\boldsymbol{\epsilon}^{(i)}\right\|^2 \leq \left(2\left(L\eta_1^{(i)}\right)^2 + 1\right)\left\|\nabla_1\mathcal{L}_1^{(i)}\right\|^2 + 2\left\|\nabla_2\mathcal{L}_2^{(i+1,i)}\right\|^2 + 2\nabla_1^T\mathcal{L}_1^{(i)}\boldsymbol{\epsilon}^{(i)}, \tag{14}$$

which concludes the proof of Lemma A.5. □

Proof of Theorem A.3. We let $\eta_1^{(i)} = \eta_1$ and $\eta_2^{(i)} = \eta_2$. Recall Lemma (A.4) and (A.5), we could calculate a factor α such that

$$\begin{aligned}
\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^{(i+1,i+1)} &\geq \begin{bmatrix} \eta_2\left(1-\frac{L_2}{2}\eta_2\right) \\ \eta_1\left(1-\frac{L_1}{2}(\eta_1)\right) \\ \eta_1 \end{bmatrix}^T \begin{bmatrix} \left\|\nabla_2\mathcal{L}_2^{(i+1,i)}\right\|^2 \\ \left\|\nabla_1\mathcal{L}_1^{(i)}\right\|^2 \\ \nabla_1^T\mathcal{L}_1^{(i)}\boldsymbol{\epsilon}^i \end{bmatrix} \\
&\overset{\textbf{(A)}}{\geq} \alpha \begin{bmatrix} 2 \\ 2\left(L\eta_1\right)^2 + 1 \\ 2 \end{bmatrix}^T \begin{bmatrix} \left\|\nabla_2\mathcal{L}_2^{(i+1,i)}\right\|^2 \\ \left\|\nabla_1\mathcal{L}_1^{(i)}\right\|^2 \\ \nabla_1^T\mathcal{L}_1^{(i)}\boldsymbol{\epsilon}^i \end{bmatrix} \\
&\overset{\textbf{(B)}}{\geq} \alpha\left(\left\|\nabla\mathcal{L}_2^{(i,i)}\right\|^2 - \|\boldsymbol{\epsilon}^{(i)}\|^2\right),
\end{aligned} \tag{15}$$

where **(B)** holds because of the conclusion of Lemma A.5, and to ensure the correctness of **(A)**, we solve the following inequalities

$$\begin{cases} \eta_2\left(1-\dfrac{L_2}{2}\eta_2\right) \geq 2\alpha \\ \eta_1\left(1-\dfrac{L_1}{2}(\eta_1)\right) \geq 2\alpha\left(L\eta_1\right)^2 + \alpha \\ \eta_1 = 2\alpha. \end{cases} \tag{16}$$

The third of Eq.(16) implies $\alpha = \frac{\eta_1}{2}$. Plug it into the second to obtain $-\frac{\sqrt{L_1^2+8L^2}+L_1}{4L^2} < \eta_1 \leq \frac{\sqrt{L_1^2+8L^2}-L_1}{4L^2}$. Plug it into the first one to obtain $\frac{1-\sqrt{1-2L_2\eta_1}}{L_2} \leq \eta_2 \leq \frac{1+\sqrt{1-2L_2\eta_1}}{L_2}$. Because the learning rate cannot be negative, its value range is truncated by zero. And in the interval of η_2, to avoid the square $\sqrt{1-2L_2\eta_1}$ from being negative, we let $\eta_1 \leq \frac{1}{2L_2}$. All above conditions imply

$$\begin{cases} 0 < \eta_1 \leq \min\left(\frac{\sqrt{L_1^2+8L^2}-L_1}{4L^2}, \frac{1}{2L_2}\right) \\ \max\left(0, \frac{1-\sqrt{1-2L_2\eta_1}}{L_2}\right) < \eta_2 \leq \frac{1+\sqrt{1-2L_2\eta_1}}{L_2} \\ \alpha = \frac{\eta_1}{2} \end{cases}, \tag{17}$$

Then, by Assumption A.1 (PL condition) and Eq. (15), we have

$$\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^{(i+1,i+1)} \geq \alpha\left(\left\|\nabla\mathcal{L}_2^{(i,i)}\right\|^2 - \|\boldsymbol{\epsilon}^{(i)}\|^2\right) \quad \geq \alpha\mu\left(\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^\star\right) - \alpha\|\boldsymbol{\epsilon}^{(i)}\|^2.$$

Re-arranging both sides, we have

$$\begin{aligned}
\mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* &\leq \alpha\|\boldsymbol{\epsilon}^{(i)}\|^2 - \alpha\mu\left(\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^*\right) + \mathcal{L}_2^{(i,i)} - \mathcal{L}_2^* \\
&= \alpha\|\boldsymbol{\epsilon}^{(i)}\|^2 + (1-\alpha\mu)\left(\mathcal{L}_2^{(i,i)} - \mathcal{L}_2^*\right).
\end{aligned} \tag{18}$$

Recusively performing Eq. (18), we have

$$\mathcal{L}_2^{(i+1,i+1)} - \mathcal{L}_2^* \leq (1-\alpha\mu)^{i+1}\left(\mathcal{L}_2^{(0,0)} - \mathcal{L}_2^*\right) + \alpha\sum_{k=0}^{i}(1-\alpha\mu)^k\|\boldsymbol{\epsilon}^{(i-k)}\|^2 \quad .$$

To ensure the convergence, $1-\alpha\mu$ should be in $(0,1)$, which implies $0 < \alpha < \frac{1}{\mu}$ and accordingly $\eta_1 < \frac{2}{\mu}$. Combining the learning rate conditions Eq. (17), we conclude the proof. □

B. Proof of Proposition 3.4

Proposition B.1. *For a model composed of L local layers parameterized by $\boldsymbol{\theta}_k$ with their local errors $\mathcal{L}_k$, $k = 1, ..., L$, when $\mathcal{L}_k^{SGR} = 0$ for all layers $2 \leq k \leq L$ for a batch of data, we have*

$$\nabla_{\boldsymbol{\theta}_k} \mathcal{L}_L = \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_k, \quad \forall 1 \leq k \leq L-1,$$

which implies that all local updates are equivalent to learning with the global true gradient back propagated from the last-layer error $\mathcal{L}_L$.

Proof of Proposition B.1. Since $\mathcal{L}_L^{SGR} = 0$, we have

$$\frac{\partial \mathcal{L}_L}{\partial \boldsymbol{x}_{L-1}} = \frac{\partial \mathcal{L}_{L-1}}{\partial \boldsymbol{x}_{L-1}}. \tag{19}$$

Multiplying both sides of Eq. (19) by $\boldsymbol{J}_f(\boldsymbol{\theta}_{L-1}) = \frac{\partial \boldsymbol{x}_{L-1}}{\partial \boldsymbol{\theta}_{L-1}}$, we have

$$\nabla_{\boldsymbol{\theta}_{L-1}} \mathcal{L}_L = \nabla_{\boldsymbol{\theta}_{L-1}} \mathcal{L}_{L-1}.$$

Multiplying both sides of Eq. (19) by $\boldsymbol{J}_f(\boldsymbol{x}_{L-2}) = \frac{\partial \boldsymbol{x}_{L-1}}{\partial \boldsymbol{x}_{L-2}}$, and considering $\mathcal{L}_{L-1}^{SGR} = 0$, we have

$$\frac{\partial \mathcal{L}_L}{\partial \boldsymbol{x}_{L-2}} = \frac{\partial \mathcal{L}_{L-1}}{\partial \boldsymbol{x}_{L-2}} = \frac{\partial \mathcal{L}_{L-2}}{\partial \boldsymbol{x}_{L-2}}. \tag{20}$$

Similarly multiplying Eq. (20) the Jacobian matrix of feature $\boldsymbol{x}_{L-2}$ *w.r.t.* $\boldsymbol{\theta}_{L-2}$, we have $\nabla_{\boldsymbol{\theta}_{L-2}} \mathcal{L}_L = \nabla_{\boldsymbol{\theta}_{L-2}} \mathcal{L}_{L-2}$. Recursively performing this process until the first layer, we have

$$\nabla_{\boldsymbol{\theta}_k} \mathcal{L}_L = \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_k, \quad 1 \leq k \leq L-1. \tag{21}$$

□

C. Proof of Proposition 3.5

Proposition C.1. *Consider a two-layer linear model composed of $\boldsymbol{x}_1 = \boldsymbol{\theta}_1 \boldsymbol{x}_0$ and $\boldsymbol{x}_2 = \boldsymbol{\theta}_2 \boldsymbol{x}_1$, where $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are the learnable linear matrices, and $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$ are the input and output of the model, respectively. We use fixed ETF structures $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ for the local classifier heads and the cross entropy (CE) loss for local errors $\mathcal{L}_1(\boldsymbol{M}_1 \boldsymbol{x}_1, y)$ and $\mathcal{L}_2(\boldsymbol{M}_2 \boldsymbol{x}_2, y)$. Denote $\mathcal{L}_2'$ as the CE loss value in inference after performing one step of gradient descent of $\mathcal{L}_1$ and $\mathcal{L}_2$ by Eq. (1) and (2), and denote $\hat{\mathcal{L}}_2'$ as the one with our $\mathcal{L}_2^{SGR}$ on the second layer. Assume that the prediction logit for the ground truth label in the second layer is larger than the one in the first layer, we have $\hat{\mathcal{L}}_2' \leq \mathcal{L}_2'$, when $\mathcal{L}_2^{SGR}$ is small.*

Lemma C.2. *For a fixed ETF classifier $\boldsymbol{M}$ defined in Eq. (7), consider two features $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ belonging to the same class y, such that $\hat{\boldsymbol{x}} = \boldsymbol{x} - \eta\, \delta\boldsymbol{x}$, where $\eta > 0$, $\delta\boldsymbol{x} = -(1-p_y)\boldsymbol{m}_y + \sum_{k \neq y} p_k \boldsymbol{m}_k$, $\boldsymbol{m}_k$ is the classifier vector of $\boldsymbol{M}$ for class k, and $p_y + \sum_{k \neq y} p_k = 1$, $p_y, p_k > 0$. When using cross entropy loss $\mathcal{L}(\boldsymbol{M}\boldsymbol{x}, y) = -\log \frac{\exp(\boldsymbol{x}^T \boldsymbol{m}_y)}{\sum \exp(\boldsymbol{x}^T \boldsymbol{m}_k)}$, we have*

$$\mathcal{L}(\boldsymbol{M}\hat{\boldsymbol{x}}, y) < \mathcal{L}(\boldsymbol{M}\boldsymbol{x}, y). \tag{22}$$

Proof of Lemma C.2. Based on Eq. (7), we have

$$\begin{aligned}\hat{\boldsymbol{x}}^T \boldsymbol{m}_y &= \boldsymbol{x}^T \boldsymbol{m}_y + \eta(1-p_y) + \frac{\eta}{K-1} \sum_{k \neq y} p_k \\ &= \boldsymbol{x}^T \boldsymbol{m}_y + \frac{K}{K-1}(1-p_y)\eta,\end{aligned}$$

where $K > 0$ is the number of total classes. For $k \neq y$, we have

$$\hat{\boldsymbol{x}}^T \boldsymbol{m}_k = \boldsymbol{x}^T \boldsymbol{m}_k - \eta(1-p_y)\frac{1}{K-1} - \eta p_k + \eta \frac{1}{K-1} \sum_{k' \neq y, k' \neq k} p_k \tag{23}$$

$$= \boldsymbol{x}^T \boldsymbol{m}_k - \frac{K}{K-1} p_k \eta. \tag{24}$$

Since $0 < p_i < 1, \forall 1 \leq i \leq K$, we have

$$\begin{aligned}\mathcal{L}(\boldsymbol{M}\hat{\boldsymbol{x}}, y) &= -\log \frac{\exp(\hat{\boldsymbol{x}}^T \boldsymbol{m}_y)}{\sum \exp(\hat{\boldsymbol{x}}^T \boldsymbol{m}_k)} \\ &= -\log \frac{\exp\left(\boldsymbol{x}^T \boldsymbol{m}_y + \frac{K}{K-1}(1-p_y)\eta\right)}{\exp\left(\boldsymbol{x}^T \boldsymbol{m}_y + \frac{K}{K-1}(1-p_y)\eta\right) + \sum_{k\neq y} \exp(\boldsymbol{x}^T \boldsymbol{m}_k - \frac{K}{K-1} p_k \eta)} \\ &< -\log \frac{\exp\left(\boldsymbol{x}^T \boldsymbol{m}_y\right)}{\exp\left(\boldsymbol{x}^T \boldsymbol{m}_y\right) + \sum_{k\neq y} \exp(\boldsymbol{x}^T \boldsymbol{m}_k)} = \mathcal{L}(\boldsymbol{M}\boldsymbol{x}, y),\end{aligned}$$

which concludes the proof of Lemma C.2. □

Proof of Proposition C.1. In the first linear layer $\boldsymbol{x}_1 = \boldsymbol{\theta}_1 \boldsymbol{x}_0$, we have the gradient of $\mathcal{L}_1(\boldsymbol{M}_1 \boldsymbol{x}_1, y)$ *w.r.t.* $\boldsymbol{\theta}_1$ as

$$\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1 = \begin{bmatrix} \boldsymbol{x}_0^T \\ \vdots \\ \boldsymbol{x}_0^T \end{bmatrix} \otimes \delta \boldsymbol{x}_1, \tag{25}$$

where $\otimes$ performs multiplication for each row vector $\boldsymbol{x}_0^T$ and each element of $\delta\boldsymbol{x}_1$ and $\delta\boldsymbol{x}_1$ is the gradient from $\mathcal{L}_1$ *w.r.t.* $\boldsymbol{x}_1$:

$$\delta \boldsymbol{x}_1 = -(1 - p_y^{(1)}) \boldsymbol{m}_y^{(1)} + \sum_{k\neq y} p_k^{(1)} \boldsymbol{m}_k^{(1)}, \tag{26}$$

where $\boldsymbol{m}_k^{(1)}$ is the classifier vector of $\boldsymbol{M}_1$ for class k, and $p_k^{(1)}$ is the probability predicted by the first layer for class k. Then we have $\boldsymbol{x}_1'$ after the update of $\boldsymbol{\theta}_1$ as:

$$\boldsymbol{x}_1' = \boldsymbol{\theta}_1' \boldsymbol{x}_0 = (\boldsymbol{\theta}_1 - \eta \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1) \boldsymbol{x}_0 = \boldsymbol{x}_1 - \eta \left\| \boldsymbol{x}_0 \right\|^2 \delta \boldsymbol{x}_1, \tag{27}$$

where η is the learning rate.

Similarly, in the second layer $\boldsymbol{x}_2 = \boldsymbol{\theta}_2 \boldsymbol{x}_1$, when our method is not adopted, we have

$$\boldsymbol{\theta}_2' = \boldsymbol{\theta}_2 - \eta \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_1^T \end{bmatrix} \otimes \delta \boldsymbol{x}_2, \tag{28}$$

where $\delta\boldsymbol{x}_2$ is the gradient from $\mathcal{L}_2$ *w.r.t.* $\boldsymbol{x}_2$. In inference, we have the new output as

$$\boldsymbol{x}_2' = \boldsymbol{\theta}_2' \boldsymbol{x}_1' = \boldsymbol{x}_2 - \eta \left\| \boldsymbol{x}_0 \right\|^2 \boldsymbol{\theta}_2 \delta \boldsymbol{x}_1 - \eta \left\| \boldsymbol{x}_1 \right\|^2 \delta \boldsymbol{x}_2 + \eta^2 \left\| \boldsymbol{x}_0 \right\|^2 \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_1^T \end{bmatrix} \otimes \delta \boldsymbol{x}_2 \cdot \delta \boldsymbol{x}_1. \tag{29}$$

When our SGR is adopted, we have

$$\mathcal{L}_2^{(SGR)} = \frac{1}{2} \left\| \frac{\partial \mathcal{L}_1}{\partial \boldsymbol{x}_1} - \frac{\partial \mathcal{L}_2}{\partial \boldsymbol{x}_1} \right\|^2 = \frac{1}{2} \left\| \delta \boldsymbol{x}_1 - \boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2 \right\|^2.$$

In our case, we denote the updated second layer parameter as $\hat{\boldsymbol{\theta}}_2'$, which can be formulated as

$$\hat{\boldsymbol{\theta}}_2' = \boldsymbol{\theta}_2 - \eta \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_1^T \end{bmatrix} \otimes \delta \boldsymbol{x}_2 - \eta \delta \boldsymbol{x}_2 \left(\delta \boldsymbol{x}_1 - \boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2 \right)^T. \tag{30}$$

Accordingly, we denote the new output in inference in our case as $\hat{\boldsymbol{x}}_2'$, which can be formulated as

$$\hat{\boldsymbol{x}}_2' = \hat{\boldsymbol{\theta}}_2' \boldsymbol{x}_1' = \boldsymbol{x}_2' - \eta \delta \boldsymbol{x}_2 \left(\delta \boldsymbol{x}_1 - \boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2 \right)^T \left(\boldsymbol{x}_1 - \eta \left\| \boldsymbol{x}_0 \right\|^2 \delta \boldsymbol{x}_1 \right) \tag{31}$$

$$= \boldsymbol{x}_2' - \eta \delta \boldsymbol{x}_2 \left(\delta \boldsymbol{x}_1^T \boldsymbol{x}_1 - \eta \left\| \boldsymbol{x}_0 \right\|^2 \left\| \delta \boldsymbol{x}_1 \right\|^2 - (\boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2)^T \boldsymbol{x}_1 + \eta \left\| \boldsymbol{x}_0 \right\|^2 (\boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2)^T \delta \boldsymbol{x}_1 \right). \tag{32}$$

We assume that $\mathcal{L}_2^{(SGR)}$ is small, which indicates that $\left\| \delta \boldsymbol{x}_1 \right\|^2 \approx (\boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2)^T \delta \boldsymbol{x}_1$, and thus we have

$$\hat{\boldsymbol{x}}_2' \approx \boldsymbol{x}_2' - \eta \delta \boldsymbol{x}_2 \left(\delta \boldsymbol{x}_1^T \boldsymbol{x}_1 - (\boldsymbol{\theta}_2^T \delta \boldsymbol{x}_2)^T \boldsymbol{x}_1 \right) = \boldsymbol{x}_2' - \eta \delta \boldsymbol{x}_2 \left(\delta \boldsymbol{x}_1^T \boldsymbol{x}_1 - \delta \boldsymbol{x}_2^T \boldsymbol{x}_2 \right). \tag{33}$$

Because we assume that the prediction logit for the label class in the second layer is larger than the one in the first layer, based on the definition of ETF classifier, we have $\delta \boldsymbol{x}_2^T \boldsymbol{x}_2 \leq \delta \boldsymbol{x}_1^T \boldsymbol{x}_1$. Consequently, the output feature in inference using our method is in a form of $\hat{\boldsymbol{x}}_2' = \boldsymbol{x}_2' - \beta \delta \boldsymbol{x}_2$ where $\beta \geq 0$. Considering the conclusion of Lemma C.2, we have

$$\hat{\mathcal{L}}_2'(\hat{\boldsymbol{x}}_2') \leq \mathcal{L}_2'(\boldsymbol{x}_2'), \tag{34}$$

which concludes the proof of Proposition C.1. □

D. Analysis of Computation Cost of SGR

Eq. (6) introduced by our method requires calculating the gradient of δx, which is the gradient of the local error *w.r.t.* the input x of this block. However, compared with InfoPro, our method is not obviously slower, and in some cases even faster (ResNet 50 and 101 when K=3) while being more efficient in memory consumption. That is because InfoPro extra introduces heavy reconstruction heads composed of multiple layers (e.g. 4 convolution layers on ImageNet) other than the local classification heads, while our method only uses light local classifiers.

Although our method concerns the gradient calculation of Eq. (6), which includes a second-order derivative, here we provide its analytical computation cost in a block to show that the introduced cost is bearable.

Consider a local block $y = \sigma(Wx)$, where $x \in \mathcal{R}^n$ is the input of this block, $y \in \mathcal{R}^m$ is the output of this block, $W \in \mathcal{R}^{m \times n}$ is the parameter, and σ is the ReLU nonlinear activation. Its local error is given by $\mathcal{L}(y, \mathcal{Y})$, where $\mathcal{L}$ is the loss function and $\mathcal{Y}$ is the ground truth. Our SGR loss term is in the form of $\mathcal{L}^{SGR} = \frac{1}{2} \| \frac{\partial \mathcal{L}}{\partial x} - g \|_2^2$, where g is the gradient from the previous local error towards x and thus is a constant vector here. We have $\frac{\partial \mathcal{L}}{\partial x} = W^T \left[\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx) \right]$, where $\otimes$ is the element-wise multiplication between two vectors, σ' is the function of the first-order derivative of σ. Because the second-order derivative of ReLU, i.e. σ'' would be zero, we have

$$\frac{\partial}{\partial W} \left(\frac{\partial \mathcal{L}}{\partial x} \right)_i = \left[\mathbf{0}_m, \ldots, \frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx), \ldots, \mathbf{0}_m \right] \in \mathcal{R}^{m \times n},$$

where i denotes the i-th element of $\frac{\partial \mathcal{L}}{\partial x}$, and $\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx)$ lies in the i-th column of the gradient matrix above with all the other columns as $\mathbf{0}_m$.

Then we have the gradient of our SGR loss *w.r.t.* W as:

$$\frac{\partial \mathcal{L}^{SGR}}{\partial W} = \left[\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx) \right] \left(\frac{\partial \mathcal{L}}{\partial x} - g \right)^T = \left[\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx) \right] \left(W^T \left[\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx) \right] - g \right)^T. \tag{35}$$

From the equation above we can see that although our method concerns second-order derivatives, the gradient calculation of our $\mathcal{L}^{SGR}$ is simply matrix multiplications with $\frac{\partial \mathcal{L}}{\partial y} \in \mathcal{R}^m$, $\sigma'(Wx) \in \mathcal{R}^m$, and $W \in \mathcal{R}^{m \times n}$. The FLOPS of the equation above is listed in Table 8.

Therefore, the theoretical computational cost is $O(3mn + m)$, which is only in a similar scale of linear layer computation. That is why training with our method will not be severely slowed down compared with global BP training. Additionally, we provide the following two strategies that can further accelerate training by leveraging the advantages brought by our method.

(1) Due to the high memory efficiency of our method, we can use a larger batchsize to speedup training from improved GPU parallelism utilization. (2) Because our method detaches all the other blocks when training each block locally, we support asynchronous training for all the blocks. Suppose we have 3 local blocks, f_1, f_2, f_3, for a neural network, the forward propagation and local training of $f_1(x^{(t+2)})$, $f_2(x^{(t+1)})$, $f_3(x^{(t)})$ can be performed asynchronously, where $x^{(t)}$ denotes the batch of train data in iteration t. This strategy can also speedup training.

Table 8. Computation analysis in Eq. (35)

term	FLOPs
$A = \left[\frac{\partial \mathcal{L}}{\partial y} \otimes \sigma'(Wx)\right]$	$O(m)$
$B = \left(W^T A - g\right)^T$	$O((2m-1)n+n)$
$A \cdot B$	$O(mn)$
total	$O(3mn+m)$

E. Implementation Details

Training details

We train all models following the common practices. To train ResNet models, we use the SGD optimizer with a learning rate of 0.1, a momentum of 0.9, and weight decay of 0.0001. We train these networks for 100 epochs with a batch size of 1024. The initial learning rate is set to 0.1 and decreases by a factor of 0.1 at epochs 30, 60, and 90. Data preprocessing includes random resizing, flipping, and cropping. For ViT-S/16 training on ImageNet, we use a batch size of 4096. We use the AdamW optimizer with a learning rate of 0.0016, and we apply a cosine learning rate annealing schedule after a linear warm-up for the first 20 epochs. The training process lasts for 300 epochs, and we apply data augmentations like random resized cropping, horizontal flipping, RandAugment, and Random Erasing. Training Swin Transformers on ImageNet uses a batch size of 1024. We use the AdamW optimizer with a learning rate of 0.001, along with betas (0.9, 0.999), epsilon 1e-08, and a weight decay of 0.05. Learning rate scheduling includes a linear warm-up for the first 20 epochs, followed by a cosine annealing schedule with a minimum learning rate of 1e-05. For Swin Transformer models, including "tiny" and "small" versions, we have a dropout rate of 0.2, an input image size of 224×224, and a patch size of 16×16. Training runs for 300 epochs, and we incorporate data augmentations like random resized cropping, horizontal flipping, RandAugment, and Random Erasing. The coefficient of our SGR loss λ is set as 10k in these supervised learning experiments on ImageNet.

In our self-supervised experiments, we follow the Barlow Twins training procedure (Zbontar et al., 2021). We use a ResNet-50 model and train it for 300 epochs. We optimize it using LARS with a learning rate of 1.6, a momentum of 0.9, and a weight decay of 1e-06. The batch size is 2048, and the learning rate schedule includes a linear warm-up for the first 10 epochs, followed by cosine annealing until the 300th epoch, maintaining a minimum learning rate of 0.0016. We use a three-layer MLP projector with 8192 hidden units and 8192 output units. To evaluate the transferability of feature representations on the ImageNet dataset through linear classification, we employ the SGD optimizer with a learning rate of 0.3, a momentum of 0.9, and a weight decay of 1e-06. We train the model using a cosine annealing learning rate schedule over 100 epochs, with a batch size of 256. Data augmentation techniques include random resizing and flipping. The model architecture is based on a ResNet-50 backbone with specific stages frozen, along with a linear classification head.

On CIFAR, we train all models for 200 epochs with an initial learning of 0.1 and a cosine annealing learning rate scheduler. We use a batchsize of 128 and adopt the SGD optimizer with a momentum of 0.9 and a weight decay of 5e-4. Standard data pre-processing and augmentations are adopted. The coefficient of our SGR loss λ is set as 1 as default.

Local classifier setups

For experiments on ImageNet, we divide a model into 2 or 3 local modules such that the training memory consumption reaches its minimum. We adopt the same local classifier following (Wang et al., 2021) for fair comparison. For experiments on CIFAR, when $K = 16$, each residual block and the initial stem layer is a local module. When $K = 3$, the model is divided according to the feature spatial resolution. When $K = 2$, we keep the last stage (feature size of 8×8) as one local module, and all the other blocks as another local layer. We adopt the same local classifier following (Belilovsky et al., 2019) ($k = 2$ in their paper) for fair comparison. There are two layers in the classifier including one convolution layer and one linear layer for classification.

F. Others

We provide more discussions and results, some of which are suggested by reviewers in the review process.

The limitations of our work include: 1) The method helps local training to better approach to global BP training by mitigating the discordant local updates successively. But there is still a gap with global BP training because the regularization

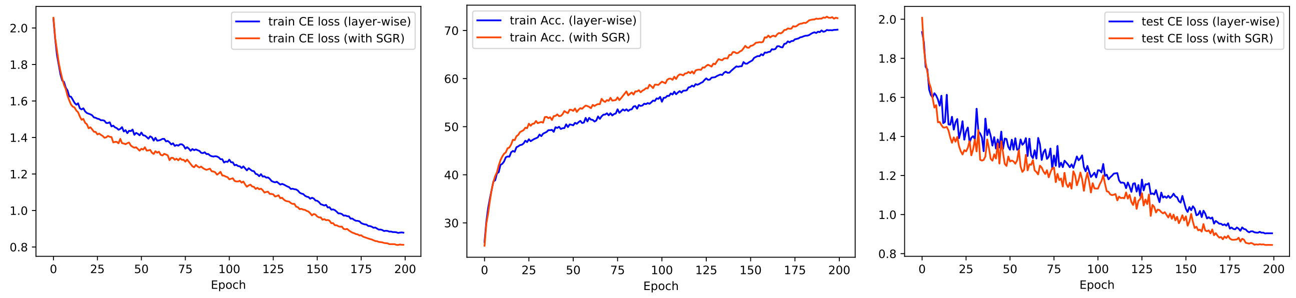


Figure 4. Train loss (left), train accuracy (middle), and test loss (right) curves with and without our method. The corresponding SGR loss value and test accuracy curves are shown in Figure 2.

term could not be optimized to 0 in real implementation. 2) The analysis and method in this study mainly deal with local training from scratch. How to apply our method into finetuning a pretrained model locally, especially for large models, deserves future exploration. 3) The train time of our method is on par with InfoPro, but is still longer than global BP training due to the introduced regularization term.

Further insights into how this regularization affects the convergence behavior and final performance. Our method adds a regularization onto the classification loss. It seems that the regularization will impede the original optimality of the classification loss and deteriorate the final performance. This is true for one-layer network. After all the local training of one layer is just the global BP training of this network. However, our method is applied in the local training of multiple-layer networks. For the last layer that outputs the final representation, its classification loss is dependent on two variables, the parameters of the last layer, and the input of the last layer. In global BP training, the chain rule can pass the gradient of the last-layer loss *w.r.t.* the input of the last layer into prior layers to change their parameters and produce a better input feature that decreases the last-layer loss. But in local training, gradients of all layers are isolated. We have no way to ensure that the input of the last layer, *i.e.* the output of its previous layer optimized by the previous local error, most satisfies the demand of minimizing the last-layer loss. In this case, from a perspective of optimization, our method does not solely optimize the last-layer parameters with its classification loss, but moves the parameters to a direction, such that the change of its input feature caused by the previous local error also enables to minimize the last-layer classification loss. That is to say training with our regularization $\mathcal{L}^{SGR}$ simultaneously optimizes the parameter variable and the input variable, to induce a larger reduction of the classification loss, and a better model performance finally. The input variable is optimized in an implicit manner, by mitigating the discordant local updates from the top to the end of a deep network successively. As shown in Proposition 3.4, when $\mathcal{L}^{SGR} = 0$ for all layers, each local update will be equivalent to the true gradient directly from the last-layer classification loss, in which situation, the input change of each local layer most satisfies the demand to minimize the last-layer classification loss.

Discussion of the comparison to other memory-saving techniques. Other memory-saving techniques, including gradient checkpointing and reversible architectures, are based on the trade-off between memory cost and computation cost. These methods still rely on global BP training, which means the activations of all layers are required when performing backpropagation of the last-layer error. In gradient checkpointing, during the forward propagation, the prior activations are removed once it is propagated into the next layer to save memory cost. In the backward propagation, however, the activations are recovered by performing forward propagation once again for each layer. Similarly, reversible architectures recover input activations of each layer in the backward propagation by re-performing these layers with their output activations using a dual path. Therefore, both gradient checkpointing and reversible architecture require to re-calculate each layer, which heavily increases their computational burden. In contrast, our method belongs to local training. It naturally enjoys memory efficiency because the forward and backward propagations of each layer are performed locally. After the local update of one layer, the activations of this layer can be detached in this iteration without performing the operations inside this layer again. Therefore, we only need to consume the memory for one layer's update while performing the forward and backward propagations of each layer only once.

More results. A PyTorch-like pseudocode for the implementation of our SGR is shown in Algorithm 1. The training curves, including train loss, train accuracy, and test loss, are shown in Figure 4. The corresponding SGR loss value and test accuracy curves are shown in Figure 2.

Algorithm 1 A PyTorch-like pseudocode for local training with SGR

```
Given: a model divided into a module list [$\mathcal{M}_k$] parameterized by $\boldsymbol{\theta}_k$ with local heads $\mathcal{L}_k(\cdot, Y)$, $1 \le k \le L$; train data
$\mathcal{S} \in \{(\boldsymbol{x}_0^t, Y^t)\}_{t \le T}$ of samples or mini-batches; hyper-parameter lambda;
optimizer_list = [torch.optim.SGD($\boldsymbol{\theta}_k$) for k in range(1,L+1)]
SGR_criterion = nn.MSELoss()
for $(\boldsymbol{x}_0^t, Y^t) \in \mathcal{S}$ do
  input = $\boldsymbol{x}_0^t$
  for $k = 1$ to $L$ do
    optimizer = optimizer_list[k]
    input.requires_grad_()
    output = $\mathcal{M}_k$(input)
    loss_cls = $\mathcal{L}_k$(output, $Y$)
    if $k >= 2$ then
      delta_now = torch.autograd.grad(
                loss_cls, input, retain_graph=True, create_graph=True)[0]
      delta_pre_norm = torch.flatten(delta_pre, 1)
      delta_pre_norm = delta_pre_norm / torch.sqrt(
                torch.sum(delta_pre_norm ** 2, dim=1, keepdims=True))
      delta_now_norm = torch.flatten(delta_now, 1)
      delta_now_norm = delta_now_norm / torch.sqrt(
                torch.sum(delta_now_norm ** 2, dim=1, keepdims=True))
      loss_SGR = SGR_criterion(delta_now_norm, delta_pre_norm)
      loss = loss_cls + lambda * loss_SGR
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    else
      delta_pre = torch.autograd.grad(
                loss_cls, output, retrain_graph=True)[0].detach()
      optimizer.zero_grad()
      loss_cls.backward()
      optimizer.step()
    end if
    input = output.detach()
  end for
end for
```