Supplementary Material: Contextual Gradient Scaling for Few-Shot Learning

Method	miniImageNet	
	1-shot	5-shot
MAML [3]	$58.37 \pm 0.49\%$	$69.76 \pm 0.46\%$
BOIL [7]	-	$71.30 \pm 0.28\%$
L2F [2]	$59.71 \pm 0.49\%$	$77.04 \pm 0.42\%$
ALFA [1]	$59.74 \pm 0.49\%$	$\mathbf{77.96 \pm 0.41}\%$
CxGrad (Ours)	$\mathbf{60.19 \pm 0.45}\%$	$75.17 \pm 0.40\%$

Table 1: Test accuracy on 5-way miniImageNet classification.

Method	tieredImageNet	
	1-shot	5-shot
MAML [3]	$58.58 \pm 0.49\%$	$71.24 \pm 0.43\%$
L2F [2]	$64.04 \pm 0.48\%$	$81.13 \pm 0.39\%$
ALFA [1]	$64.62 \pm 0.49\%$	$82.48 \pm 0.38\%$
CxGrad (Ours)	$\mathbf{65.47 \pm 0.44}\%$	$\mathbf{82.52 \pm 0.35}\%$

Table 2: Test accuracy on 5-way tieredImageNet classification.

A. Experiment Settings

Our implementation and experiment settings are based on those of ALFA [1]. In all the experiments, The network architecture is 4 convolutional blocks [9] followed by a fully connected layer. Each block consists of a convolutional layer with 3×3 kernel, a batch normalization layer, a ReLU activation function, and a 2×2 max-pooling layer. The dimension of the context parameters and the dimension of the intermediate features in g_ϕ are both 100. The dimension of the outputs of g_ϕ is same as the number of convolution layers in the backbone. In Q_i, each class contains 15 samples. None of the data-augmentation methods are used in training. The batch size B is 4 for 1-shot and 2 for 5-shot. We optimize the model in the inner-loop for 5 steps and set the learning rates α, β, and η to be 0.01, 1.0, 0.001, respectively. Adam [6] is used as the meta-optimizer in outer-loop. Our model is trained for 50,000 iterations on miniImageNet and 125,000 iterations on tieredImageNet. An ensemble of models, whose ranks are top 5 in terms of meta-validation accuracy, is evaluated on 600 tasks from meta-test set. We run 3 independent runs with 3 different seeds and report the average results.

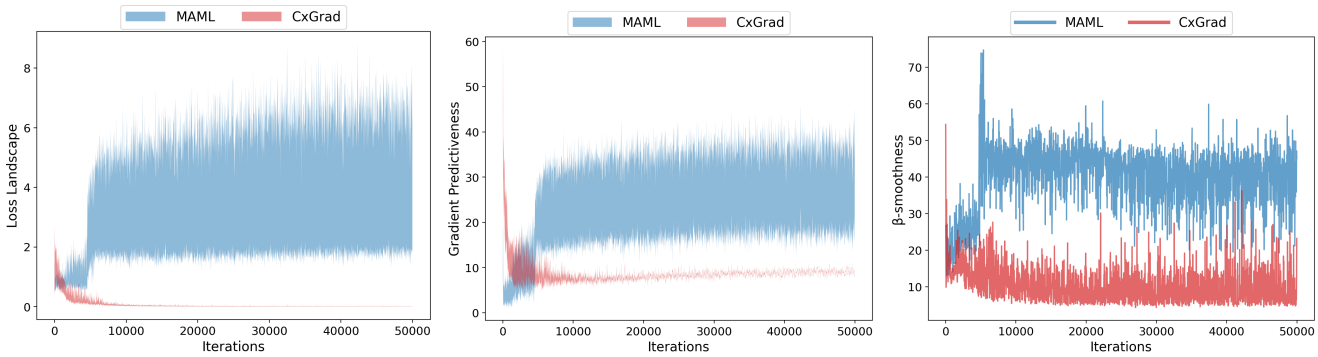
B. Experimental Results on a Bigger Backbone

In this section, we provide additional experimental results on a deeper backbone, especially ResNet-12 [4]. ResNet-12 consists of 4 residual blocks. Each residual block is composed of three convolutional blocks, each of which consists of a convolutional layer, a BN layer, and a ReLU activation function. A pointwise convolutional block is positioned at the skip connection for matching the number of channels between the residual inputs and the outputs. A 2×2 max-pooling layer is at the end of each residual block. The number of channels begins with 64 and gets doubled by each residual block. Finally, we aggregate the spatial dimension of the final representation by a global average pooling layer and pass it to the classifier. For ResNet-12, we apply the scaling process for all the residual blocks. More specifically, in each residual block, we only scale the weights for the three convolutional blocks, not for the pointwise convolutional block. Table 1 and Table 2 provide 5-way few-shot classification performance using ResNet-12 on miniImageNet and tieredImageNet, respectively.

C. Optimization Landscape

Ioffe [5] argued that batch normalization helps the training by reducing *internal covariate shift* (ICS). However, Santurkar [8] found that the true reason is that batch normalization smooths the optimization landscape in training. In other words, batch normalization improves the Lipschitzness of both the loss and the gradients of a model. To prove this argument, he measured the variation in loss, gradient predictiveness, and "effective" β smoothness in the vicinity of a certain point on the optimization landscape. For more details, please refer to Section 3 in [8]. Based on this study, Baik [2] analyzed the optimization landscape of MAML and his method L2F. Likewise, in this section, we also analyze how our method affects the optimization landscape in the inner-loop following [2, 8]. In Figure 1, we plot these three measurements and explain the meaning of each one.

In order to analyze the optimization landscape in the inner-loop, we have to observe the loss and the gradients in the vicinity of the model parameters adapted to $\mathcal{T}_i$. To



(a) Loss landscape (b) Gradient predictiveness (c) "effective" β smoothness

Figure 1: Optimization landscape during adaptation. Like other experiments, we plot these results from 5-way 5-shot mini-ImageNet few-shot classification. In order to analyze the landscape, we update the model parameter from a certain inner-loop step with various range of learning rates. We refer to these updated parameters as *points*. With these, we can analyze the local landscape from the certain step. (a) We measure the variation in loss calculated at the *points*. (b) We measure the variation in ℓ_2 distance between the gradients at the certain step and at each *point*. (c) "effective" β smoothness refers to the maximum ℓ_2 difference of (b) over distance as we move to the corresponding *point* from the certain step.

accomplish this, we perform adaptation with a new learning rate set in $[0.5, 4] \times \alpha$. We set $\alpha = 0.01$ as before. Let the new learning rate set $\mathcal{A} = \{x | x = 0.5\alpha i, i \in [8]\}$, j-th learning rate of $\mathcal{A}$ be $\bar{\alpha}_j$, and $\boldsymbol{\theta}_{i,s}$ be the parameters of the adapted model to $\mathcal{T}_i$ at s-th step in the inner-loop. Then, we can compute several model parameters around $\boldsymbol{\theta}_{i,s}$ as below:

$$\boldsymbol{\theta}_{i,s}^{\bar{\alpha}_j} = \boldsymbol{\theta}_{i,s} - \bar{\alpha}_j \nabla_{\boldsymbol{\theta}_{i,s}} \mathcal{L}(\boldsymbol{\theta}_{i,s}; S_i) \tag{1}$$

Next, let B denote the batch size and S denote the number of inner-loop steps. We plot the variation in the loss and the gradient predictiveness computed by Eq. (2) and Eq. (3) in Figure 1a and Figure 1b, respectively.

$$\frac{1}{BS} \sum_{b=1}^{B} \sum_{s=1}^{S} \mathcal{L}(\boldsymbol{\theta}_{i,s}^{\bar{\alpha}_j}; S_i) \tag{2}$$

$$\frac{1}{BS} \sum_{b=1}^{B} \sum_{s=1}^{S} \| h(\boldsymbol{\theta}_{i,s}^{\bar{\alpha}_j}) - h(\boldsymbol{\theta}_i) \| \tag{3}$$

where $h(\boldsymbol{\theta}_i) = \nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta}_i; S_i)$. In Figure 1c, we calculate the values by Eq. (4) and plot them. Here, $\bar{\alpha}_j^* = \text{argmax}_{\bar{\alpha}_j \in \mathcal{A}} \| h(\boldsymbol{\theta}_{i,s}^{\bar{\alpha}_j^*}) - h(\boldsymbol{\theta}_i) \|$.

$$\frac{1}{BS} \sum_{b=1}^{B} \sum_{s=1}^{S} \frac{\| h(\boldsymbol{\theta}_{i,s}^{\bar{\alpha}_j^*}) - h(\boldsymbol{\theta}_i) \|}{\| \bar{\alpha}_j^* h(\boldsymbol{\theta}_i) \|} \tag{4}$$

Looking at Figure 1a, we can observe the loss landscape in the inner-loop as the training proceeds. In the case of MAML, the variation in loss becomes larger from approximately 5,000 iterations. It means that the loss landscape gets sharper in the vicinity of the points on it. On the contrary, in the case of CxGrad, the variation in loss is drastically reduced, implying that CxGrad efficiently and effectively smooths the loss landscape. In Figure 1b, gradient predictiveness means how far the gradients around a certain point are from the gradient at the point. The farther the distance, the lower the stability. At the beginning of training, CxGrad is more unstable than MAML in terms of gradients because the sub-network $g_{\boldsymbol{\phi}}$ doesn't learn sufficient knowledge from tasks to scale the gradients of the backbone in a task-wise manner. Nevertheless, CxGrad retains enough stability in no time. Lastly, in Figure 1c, CxGrad shows better Lipschitzness than MAML. It means that CxGrad also smooths the gradients besides the loss. As a result, CxGrad improves both the convergence speed and the performance of the model.

References

[1] Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. Meta-learning with adaptive hyperparameters. In *NeurIPS*, 2020.

[2] Sungyong Baik, Seokil Hong, and Kyoung Mu Lee. Learning to forget for meta-learning. In *CVPR*, 2020.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[7] Jaehoon Oh, Hyungjun Yoo, ChangHwan Kim, and Se-Young Yun. Boil: Towards representation change for few-shot learning. In *ICLR*, 2021.

[8] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? 2018.

[9] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.