Studying BatchNorm Learning Rate Decay on Meta-Learning Inner-Loop Adaptation

Alexander Wang*
University of Toronto
alexw@cs.toronto.edu

Gary Leung*
University of Toronto
garyleung@cs.toronto.edu

Sasha Doubov*
University of Toronto
doubovs@cs.toronto.edu

Abstract

Meta-learning for few-shot classification has been challenged on its effectiveness compared to simpler pretraining methods and the validity of its claim of "learning to learn". Recent work has suggested that MAML-based models do not perform "rapid-learning" in the inner-loop but reuse features by only adapting the final linear layer. Separately, BatchNorm, a near ubiquitous inclusion in model architectures, has been shown to have an implicit learning rate decay effect on the preceding layers of a network. We study the impact of BatchNorm's implicit learning rate decay on feature reuse in meta-learning methods and find that counteracting it increases change in intermediate layers during adaptation. We also find that counteracting this learning rate decay sometimes improves performance on few-shot classification tasks.

1 Introduction

Meta-learning [2] offers a compelling promise of machine learning generalization. A model "learns to learn" by solving a collection of different tasks in an episodic training objective. By optimizing this challenging bi-level objective, the model learns to adapt its weights to new tasks at test time through a few simple gradient descent steps. The updates usually aim to update large portions, if not the entire network, to suit each task.

This is in contrast to pretraining methods [14] that learn a single flexible representation to solve an entire distribution of tasks by adding and training a single layer linear classifier during test-time. These approaches have been shown to effectively tackle few-shot learning tasks and are simpler to train than meta-learning methods. A natural question is to explore why meta-learning is outperformed by these pretraining techniques. [12] uncovers that while meta-learning methods perform gradient descent updates to the entire network, the inner loop updates only change the final linear layer, leaving the majority of the weights unchanged. This suggests that meta-learning methods are not actually adapting their representations for each task. Intuitively, this weakens its distinction from pre-training methods as full-network adaptability is often considered one of the strengths of meta-learning [10].

Separately, it has been shown that the usage of BatchNorm modules in neural networks causes an implicit learning rate decay proportional to weight norm of the preceding layer. As many meta-learning models make use of the 4-block convolutional architecture used in [2], they may all be subject to this decay phenomenon. This is particularly in line with ANIL [12] where only the final layer, which is not followed by a BatchNorm layer, sees substantial adaptation.

In our work, we aim to answer the question, *does BatchNorm's implicit learning rate decay negatively impact model adaptation and lead to feature reuse?* We find evidence that counteracting this decay increases learning in intermediate layers and improves performance on existing few-shot classification

*Equal contribution.

5th Workshop on Meta-Learning at NeurIPS 2021, Sydney, Australia.

benchmarks. We also find that enabling this increased adaptation lead to notable performance gains over other methods for tasks sampled from a different distribution.

2 Related Work

Representation adaptation in meta-learning MetaOptNet [8] is the first method to meta-learn a fixed representation and then only adapt a final convex classification layer during task-specific training. ANIL [12] studies rapid-learning vs. feature reuse in meta-learning by studying individual layer adaptation. They demonstrate that representations hardly change during inner loop optimization for MAML [2] and that training only the final layer during the inner-loop preserves most of the method's performance. More recently, BOIL [10] does the opposite of ANIL and only optimizes the body of the network and also proposes evaluating a model on different test-time datasets. [1] proposes a method of creating few-shot tasks that are more challenging and observe improved performance of full-model adaptation methods over ANIL and pretraining approaches.

Normalization and learning rate decay [4] first identifies that in CNNs, the norm of the weight vectors grows logarithmically with the number of gradient descent steps. [16] then identifies that the usage of normalization schemes influences the effective learning rate based on the magnitude of the layer weights that precede and normalization module. [9] then proposes an exponentially growing learning rate for neural network training to improve model performance.

3 Studying the Inner-Loop Adaptation Procedure

3.1 Implicit Learning Rate Decay in Inner-Loop Adaptation

We observe that all layers except for the final Dense layer in the 4-block convolutional architecture is affected by BatchNorm and its implicit learning rate decay. This was previously studied in [16] and we reference [3] in our explanation below. This is due to the scale invariance of functions from BatchNorm and other normalization schemes. We focus on BatchNorm as it is the most prevalent normalization method. Due to the operation being scale invariant, the network function and cost are also scale invariant when viewed from the incoming weights $\mathbf{w}_j$. If a function $g(x)$ is scale invariant with respect to x with any scale γ, the following holds true:

$$g(\gamma x) = g(x) \tag{1}$$

$$\nabla g(\gamma x) = \gamma^{-1} \nabla g(x) \tag{2}$$

We crucially observe that while the function value is now unaffected by scale due to BatchNorm, the gradient of the function is not. We can study the dynamics this has on training by looking at our stochastic gradient descent update. Because the scale of $\mathbf{w}_j$ is invariant, we canonicalize $\mathbf{w}_j$ to the unit vector $\hat{\mathbf{w}} = \mathbf{w}_j / ||\mathbf{w}_j||$ to use as a reference point to calculate an effective learning rate. The gradient update is approximated as:

$$\hat{\mathbf{w}}_j^{(k+1)} = \frac{\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)})}{||\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)})||} \approx \frac{\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)})}{||\mathbf{w}_j^{(k)}||} = \hat{\mathbf{w}}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^{-2} \nabla \mathcal{J}(\hat{\mathbf{w}}_j^{(k)}) \tag{3}$$

From this equation, we can interpret $\hat{\alpha} = \alpha ||\mathbf{w}_j^{(k)}||^{-2}$ as the effective learning rate. Thus, the learning rate for each layer is downscaled by the square of the norm of its weights. We note that as the final Dense layer is not succeeded by BatchNorm, it does not experience any decay effect. It is also important to note that as training progresses, the weight norm $||\mathbf{w}_j^{(k)}||$ will also increase, resulting in a further gradual decrease of learning rate. This is shown empirically in Figure 1, with the mathematical explanation in Appendix A.3.

3.2 Counteracting Implicit Learning Rate Decay

We aim to study the effect that implicit learning rate decay has on model adaptation by counteracting this effect during the inner-loop training procedure. We present two solutions – Scaled learning rate

update and Unit norm weight update. For each method, we outline the update rule and the resulting new effective learning rate and include a toy example demonstrating each in Appendix A.2.

Scaled learning rate update multiplies the inner-loop learning rate for each layer except for the classification head by its squared weight norm.

Update rule: We apply the following gradient update (with changes shown in color):

$$\mathbf{w}_j^{(k+1)} = \mathbf{w}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^2 \nabla \mathcal{J}(\mathbf{w}_j^{(k)}) \tag{4}$$

Effective learning rate:

$$\hat{\mathbf{w}}_j^{(k+1)} = \frac{\mathbf{w}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^2 \nabla \mathcal{J}(\mathbf{w}_j^{(k)})}{||\mathbf{w}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^2 \nabla \mathcal{J}(\mathbf{w}_j^{(k)})||} \approx \frac{\mathbf{w}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^2 \nabla \mathcal{J}(\mathbf{w}_j^{(k)})}{||\mathbf{w}_j^{(k)}||} = \hat{\mathbf{w}}_j^{(k)} - \alpha \nabla \mathcal{J}(\hat{\mathbf{w}}_j^{(k)}) \tag{5}$$

As seen above, the effects of BatchNorm's implicit learning rate decay have been counteracted by our scaled learning rate update. Note that our weights are not actually normalized, rather this update is to correct the learning rate scaling from the perspective of a normed weight vector.

Unit norm weight update ensures that all the weights for each layer in the network other than the classification head have a weight norm of 1 throughout training.

Update rule: We normalize each layer by its norm after every step of gradient descent, such that $||\mathbf{w}_j^{(k)}|| = 1$ for all k. We apply the following gradient update (with changes shown in color).

$$\mathbf{w}_j^{(k+1)} = \frac{\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)})}{||\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)}||} \tag{6}$$

Effective learning rate: As seen below, the effects of BatchNorm's implicit learning rate decay have been counteracted by our unit norm weight update.

$$\mathbf{w}_j^{(k+1)} = \hat{\mathbf{w}}_j^{(k+1)} \approx \frac{\mathbf{w}_j^{(k)} - \alpha ||\mathbf{w}_j^{(k)}||^{-1} \nabla \mathcal{J}(\hat{\mathbf{w}}_j^{(k)})}{||\mathbf{w}_j^{(k)}||} = \mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\hat{\mathbf{w}}_j^{(k)}) \tag{7}$$

4 Experiments

4.1 Experimental setup

We perform our experiments on several classification benchmarks for few-shot learning. We use the *mini*ImageNet dataset proposed by [13]. We also perform experiments on the Caltech-UCSD Birds 200 (CUB) dataset [17] which is a fine-grained classification task for birds and on the Fewshot-CIFAR100 (FC-100) dataset [11], which is based on the widely-used CIFAR100 dataset.

For all experiments, we use the standard Conv4 architecture for meta-learning outlined by [2]. We use 64 filters per layer and max-pooling, as was done previously for the *mini*ImageNet benchmark. We perform 5-way classification with 1 or 5 shots denoted as 5x1 and 5x5 classification, respectively.

We show results from MAML [2], ANIL [12] and our two methods. We also include a simple baseline in several experiments, labelled "3X LR update", which refers to the Vanilla MAML algorithm with the inner loop learning rate scaled by a factor of 3. This is used to test whether rapid adaptation in the network can occur simply with an increased learning rate.

We provide details on hyperparameter settings for our experiments in Appendix A.1.

	*mini*ImageNet	
	5-way 1-shot	5-way 5-shot
MAML	47.8±1.0	63.9±1.0
MAML 3X	48.6±0.9	62.9±1.1
ANIL	47.3±0.9	61.0±0.8
Scaled LR Update	49.6±0.9	67.2±1.0
Unit Norm Update	35.8±0.7	49.6±1.0

Table 1: Accuracy Results on the *mini*Imagenet.

4.2 Evaluation Metrics

In addition to reporting accuracies, we are interested in measuring the layer-wise similarity between the *slow weights*, the weights of the network prior to performing the inner-loop optimization, and the *fast weights*, the weights of the network after the inner-loop optimization.

We use the centered kernel alignment (CKA) metric [5], which is also used to motivate ANIL in [12]. CKA measures the similarity between two neural network representations by comparing the neuron activations over a set of data. We use the output activations of each convolutional layer in the Conv4 network and the final dense linear layer for our comparisons.

4.3 Weight norm throughout training

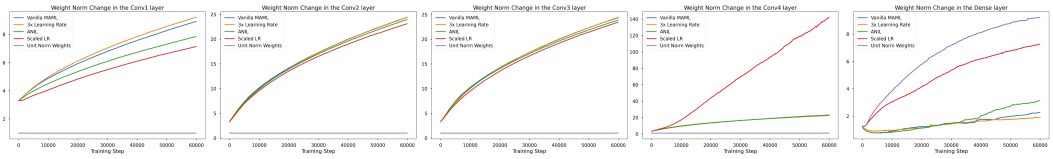

Figure 1: Weight norms layers, conv1-4 and dense layer throughout training on *mini*ImageNet 5x1 task

We visualize how the weight norm changes throughout the training process across the learning algorithms in 1. The growth in the weight norm confirms that learning rate decay occurs and begins quite early in training. We also see that the 3x learning rate method follows a similar weight norm trajectory compared to vanilla MAML updates. This suggests that increasing the learning rate does not change the weight trajectory. Curiously, we see that the Scaled LR norm increases dramatically in the Conv4 block, though the underlying mechanism behind this change requires further study.

4.4 Feature reuse due to batch normalization

In Figure 2, we display the CKA similarity between the fast and slow weights per layer in the Conv4 network on the *mini*ImageNet 5x1 tasks. This is performed after training the network for 60k iterations. Here, the feature reuse phenomenon of MAML can be observed, as the dense layer is the only one that changes meaningfully after inner loop optimization. ANIL has similar behaviour by construction, as its earlier layers do not perform any weight updates. We also see that the 3x Learning Rate only has significant updates in the last layer, which suggests that this effect is not due to the inner loop learning rate being too small.

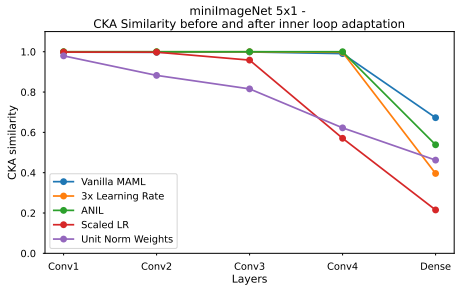

Figure 2: CKA Similarity for *mini*ImageNet 5x1

Interestingly, we see that the unit norm weight update and scaled learning rate update have different behaviours, despite the results in Section 3.2 showing their equivalence. We see that the Unit weight update deviates significantly from its slow weights in the 2nd and 4th layers, while the scaled learning rate update drops off at the 3rd layer of the network.

From these plots, it is evident that our update methods behave differently when applied to MAML's bi-level optimization. Furthermore, we see that the unit norm weight update counteracts the feature

re-use phenomenon more significantly than its scaled learning rate counterpart, as there is less feature reuse occurring earlier within the network.

Despite the promising results for the unit norm weight update in its ability to encourage learning in earlier layers of the network, it performs poorly, achieving lower accuracy on the *mini*ImageNet benchmark, as seen in Table 1.

The scaled learning rate update outperforms the vanilla MAML algorithm on the *mini*ImageNet benchmark. The 3X Learning Rate achieves similar performance to the scaled Learning Rate update on *mini*ImageNet. It isn't entirely clear why MAML with higher learning rate performs better, this could be explained with further hyperparameter tuning in future work.

Lastly, in Figure 3 we see that the CKA similarity metrics do not change significantly throughout the training process for each learning rule, implying that the feature phenomenon occurs very quickly into the training procedure.

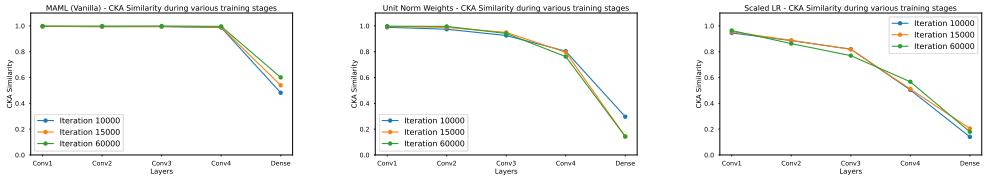

Figure 3: CKA similarity over model training on *mini*ImageNet 5x1

4.5 Feature reuse and batch normalization in diverse tasks

We wish to verify how batch normalization affects MAML's performance in more diverse settings. To test this, we measure the CKA metric on the CUB, and Fewshot-CIFAR100 datasets, as well as the *mini*ImageNet dataset, Figure 4. As in Section 4.4, we train models using unit norm weight update, scaled learning rate update and the vanilla MAML algorithm.

In Figure 4, we observe that feature reuse continues to occur across most datasets for the vanilla MAML algorithm as CKA does not register any changes other than the final layer. Comparatively, we see considerable adaptation over all datasets trained using our two update methods.

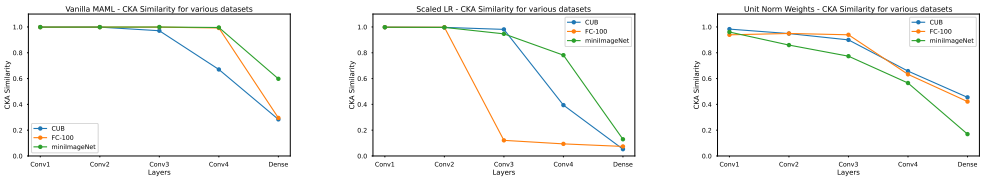

Figure 4: CKA similarity for the CUB, FC-100, and *mini*ImageNet datasets.

4.6 Novel Dataset Generalization

To further study how meta-learning behaves in different data settings, we draw tasks from different datasets during test time. Changing class domains and image settings may demand more adaptation from the model. Here, we evaluate our *mini*ImageNet5x1 model on two downstream tasks of CUB and FC-100, in addition to its training set, *mini*ImageNet as a baseline comparison. We present a single-graph comparing our different methods trained on *mini*ImageNet and tested on FC-100 in Figure 5. We also show Figure 6 showing CKA results for each method across different datasets. MAML continues to re-use features and surprisingly performs even less adaptation on the transfer CUB task than the model trained on CUB. This could be

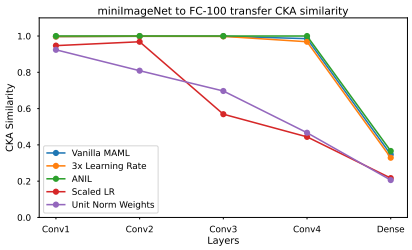

Figure 5: CKA Similarity for *mini*ImageNet 5x1 model on FC-100

Evaluation Dataset	Model (trained on *mini*ImageNet)	Setting	
		5way-1shot	5way-5shot
CUB	MAML	45.3±0.8	61.8±0.6
	3x LR MAML	43.1±0.8	61.9±0.9
	ANIL	40.6±0.8	58.4±0.6
	Scaled LR Update	48.1±0.9	67.8±0.9
	Unit Norm Update	39.9±0.8	55.5±0.8
FC-100	MAML	39.1±0.7	49.3±0.9
	3x LR MAML	37.9±0.7	48.7±0.9
	ANIL	37.5±0.7	45.8±0.6
	Scaled LR Update	40.9±0.8	48.9±0.9
	Unit Norm Update	40.8±0.8	49.2±1.1

Table 2: Accuracy Results of the *mini*Imagenet trained model on the CUB and FC-100 dataset

due to a greater breadth of features learned by training on *mini*ImageNet. Once again, our methods adapt more across all tasks.

We present accuracy evaluations in Table 2, we notice similar trends in the CUB evaluation as in the *mini*ImageNet evaluation - we observe improvements in accuracy for the model with scaled learning rate update as compared to vanilla MAML and a general decrease in accuracy in the unit norm weight update case. Interestingly, in the FC-100 evaluations, the unit norm weight update shows surprisingly competitive results, performing similarly to the scaled learning rate update and MAML in both 5 way-1 shot and 5 way-5 shot.

Furthermore, in Figure 5, we see that the 3X learning rate update performs similarly to the Vanilla MAML and ANIL updates, and fails to update the intermediate layers significantly. This again reinforces the point that the underlying effect we are studying is *not* caused by a small learning rate.

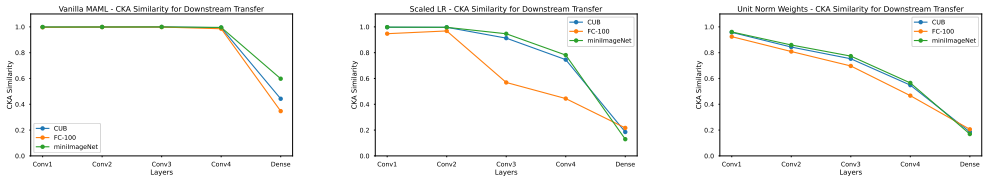

Figure 6: CKA similarity for vanilla MAML, scaled learning rate and unit norm weight update *trained* on *mini*ImageNet and evaluated on the CUB, FC-100, *mini*ImageNet datasets

5 Conclusion and future work

As previous works have noted, meta-learning methods often do not "adapt" to new tasks and instead train a single-layer network to re-use a meta-learned representation. In our work, we study the impact of implicit learning rate decay due to batch normalization on model adaptation within the test time inner loop. Using CKA [6] as our metric for model adaptation at each layer of the network, we find that counteracting the batch norm learning rate decay increases intermediate layer representation change throughout the entirety of meta-learning training. This suggests that there is increased model adaptation to each new task.

One of our methods does also show some modest improvement in classification accuracy for some tasks and settings. The normed weights approach, struggles, but we expect this to improve and require entirely new hyperparameter tuning compared to unnormalized weight vectors. Furthermore, we believe that the minor gain could be due to existing benchmarks being relatively simple. Other future work would include testing on more challenging tasks with greater class diversity or different image statistics to observe any possible benefits. Diverse tasks from Meta-Dataset [15] or artificially generated ones from ATG [1] may benefit from this increased adaptability and capacity to learn.

References

[1] S. M. R. Arnold and F. Sha. Embedding adaptation is still needed for few-shot learning. *CoRR*, abs/2104.07255, 2021.

[2] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[3] R. Grosse. Csc2541 - chapter 5 adaptive gradient methods, normalization, and weight decay, February 2021.

[4] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1731–1741, 2017.

[5] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited, 2019.

[6] S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. *CoRR*, abs/1905.00414, 2019.

[7] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[8] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10657–10665. Computer Vision Foundation / IEEE, 2019.

[9] Z. Li and S. Arora. An exponential learning rate schedule for deep learning. *CoRR*, abs/1910.07454, 2019.

[10] J. Oh, H. Yoo, C. Kim, and S. Yun. Does MAML really want feature reuse only? *CoRR*, abs/2008.08882, 2020.

[11] B. N. Oreshkin, P. Rodriguez, and A. Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning, 2019.

[12] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of MAML. *CoRR*, abs/1909.09157, 2019.

[13] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[14] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola. Rethinking few-shot image classification: A good embedding is all you need? In A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIV*, volume 12359 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2020.

[15] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[16] T. van Laarhoven. L2 regularization versus batch and weight normalization. *CoRR*, abs/1706.05350, 2017.

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

A Appendix

A.1 Hyperparameter Details

We train a single model for each N-way K-shot problem and test it on all 3 datasets (*mini*ImageNet, CUB and FC-100). The MAML baseline uses the same hyperparameters as [2], making the 3x inner-loop learning rate variation use a rate of 0.03 for both training and validation. For the 5x1 setting, our methods used a batch size of 4, 6 inner-loop steps and an inner-loop learning rate of 0.005 and outer loop 0.0005. For our 5x5 setting, our methods used a batch size of 2 and 6 inner loops steps. Our scaled learning rate update used 0.001 for both inner and outer learning rates while our the unit norm weight update used 0.005 and 0.0005 respectively.

A.2 Unit weights vs. layer-wise learning rate update

We perform a simple toy experiment to confirm the parity between our two methods of scaling the learning rate for each layer as well as normalizing layer weights. We implement both methods and train the same convolutional model as discussed in Section 4.1. This is in a regular supervised learning (*not* meta-learning) problem where we train the model to classify CIFAR10 [7]. We see in Figure 7 that the unit weight and scaled learning rate methods are roughly the same. While not identical, this could be due to the approximation described in equation 5.

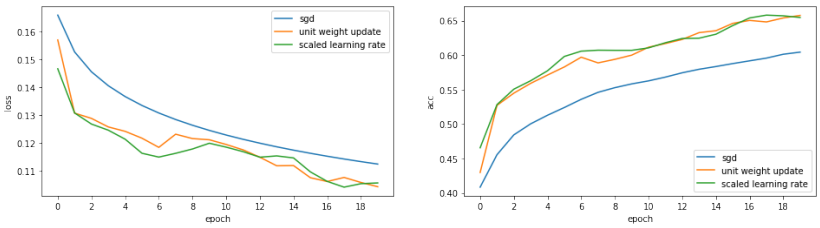

Figure 7: Validation of our methods

A.3 Mathematical Explanation of Increasing Weight Norm during Training

We reference our math from [16] and [3]. If the function $g(x)$ is homogeneous of degree k with respect to x with any scale γ, the following holds true:

$$g(\gamma x) = \gamma^k g(x) \tag{8}$$

$$\nabla g(\gamma x) = \gamma^{k-1} \nabla g(x) \tag{9}$$

The function $g(x)$ is said to be scale-invariant if this holds for the degree $k = 0$. One corollary required for Equation 9, which we left out for brevity in the main text, is that if the function g is scalar-valued (which in our case holds true), Euler's homogeneous function theorem tells us that:

$$x^\top \nabla g(x) = k g(x) \tag{10}$$

In the scale invariant case where $k = 0$, then x is orthogonal to $\nabla g(x)$. Using Pythagorean's Theorem, we can now derive the update for our weight norm:

$$||\mathbf{w}_j^{(k+1)}||^2 = ||\mathbf{w}_j^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}_j^{(k)})||^2 = ||\mathbf{w}_j^{(k)}||^2 + \alpha^2 ||\nabla \mathcal{J}(\mathbf{w}_j^{(k)})||^2 \tag{11}$$

As shown in the equation, our weight norm increases monotonically according to $||\mathcal{J}(\mathbf{w}_j^{(k)})||^2$.