# A Proof of (3)

Due to fundamental theorem of calculus,

$$L(\theta^{k+1}) - L(\theta^k) = \int_C \nabla L(\theta) \cdot d\theta = \int_0^1 \nabla L(\theta(t)) \cdot v(t) dt, \tag{5}$$

where $C$ is a trajectory whose start and end points are $\theta^k$ and $\theta^{k+1}$. In GD setting, because $\theta^{k+1} = \theta^k - \alpha \nabla L(\theta^k)$ for some learning rate $\alpha > 0$, we can think of the straight line trajectory joining $\theta^k$ and $\theta^{k+1}$. In this case, the velocity vector becomes $v(t) = -\alpha \nabla L(\theta^k)$ and

$$L(\theta^{k+1}) - L(\theta^k) = -\alpha \int_0^1 \nabla L(\theta(t)) \cdot \nabla L(\theta^k) dt \tag{6}$$

where $\theta(0) = \theta^k$, $\theta(1) = \theta^{k+1}$ and $\theta(t) = (1-t)\theta(0) + t\theta(1)$.

By Taylor series expansion it becomes

$$\nabla L(\theta(t)) \approx \nabla L(\theta^k) + H(\theta^k)(\theta(t) - \theta^k) = (I - \alpha t H(\theta^k)) \nabla L(\theta^k) \tag{7}$$

and combining Eq.(6) and Eq.(7) proves Eq.(3).

# B SHOT is robust to hyperparameter settings

Table 8: Test accuracy % of 4-conv using SHOT with different $\lambda$ values. The values in parentheses indicate the number of shots. The better accuracy between the baseline and SHOT is bold-faced.

| size of $\lambda$ | miniImageNet (5) |
|---|---|
| 0 (baseline, MAML) | 64.81 $\pm$ 1.63 |
| 0.1 (SHOT$_r$) | **66.86** $\pm$ 0.58 |
| $10^{-2}$ | 65.97 $\pm$ 0.93 |
| $10^{-3}$ | 66.85 $\pm$ 0.47 |
| $10^{-4}$ | 66.08 $\pm$ 0.36 |
| $10^{-5}$ | 66.81 $\pm$ 0.28 |

Table 8 shows the performance of SHOT with various hyperparameter settings. We conducted experiments with different learning rates of $\lambda$, and in all cases, SHOT improved the performance of the baseline. This suggests that SHOT is robust to hyperparameter settings.

# C Another viewpoint of ANIL and BOIL

In this section, we reconcile the opposite opinions of feature reuse versus feature adaptation in gradient-based meta-learning (GBML) with our hypothesis that reducing the impact of the Hessian in the inner loop can improve performance.

ANIL, proposed in [11], argues that feature reuse is key in the inner loop, where the feature remains invariant while only the decision boundary is adapted. On the other hand, BOIL, proposed in [12], argues that feature adaptation is key in the inner loop, where the feature is adapted while the decision boundary remains invariant.

To test their arguments, ANIL and BOIL proposed two algorithms. ANIL freezes the encoder and only updates the head in the inner loop, while BOIL freezes the head and only updates the encoder in the inner loop. The problem is that both algorithm shown good performance thereby both arguments look persuasive despite they argue exactly in the opposite ways.

Our hypothesis, which suggests that the outer loop implicitly suppresses the Hessian along the optimization trajectory, can reconcile the arguments of both ANIL and BOIL. This is because our hypothesis implies that the model acts linearly in the inner loop. ANIL and BOIL can be interpreted as algorithms that enforce linearity in the inner loop by restricting parameters and reducing the number of non-linear components between layers.

ANIL freezes the encoder and only updates the head in the inner loop, reducing the number of non-linear components in the inner loop. This enforces linearity in the inner loop, as the only non-linearity is the loss function. ANIL achieves better performance than MAML in 1-step optimization, as it is more powerful at 1-step optimization, which views the model as linear.

BOIL freezes the head and only updates the encoder in the inner loop, reducing the number of non-linear components in the inner loop. By applying BOIL, the gradient norm is predominant in the last layer of the encoder, making it a variant of ANIL that updates only the penultimate layer. This layer has much stronger performance, as it can change the feature while maintaining the linearity of the model. Table 14 of [12] shows a boosted performance when all but the penultimate layer is not frozen. By explicitly enforcing linearity in the inner loop, BOIL achieves improved performance.

# D GBML is a variant of Prototype Vector method

In this section, we provide a novel viewpoint of GBML, that GBML (Gradient-Based Meta Learning) is a varient of MBML (Metric-Based Meta Learning). This viewpoint relies on the **linearity assumption**. *i.e.*, the effect of the Hessian along the optimization trajectory is zero, thereby the model act as linear in the inner loop.

Suppose there exists a meta-learning model that satisfies the linearity assumption in the inner loop, then classifying a new classification task with a task-specific function $f(\cdot|\theta^\star)$ after an inner loop is equivalent to creating a prototype vector for each class on a specific feature map and classifying the input as the class of the most similar prototype vector.

The proof starts by defining the prototype vector at first.

**Prototype Vector** We define a prototype vector $V_c$ for class $c$ in an $N$-way $K$-shot classification task formally as

$$V_c = \sum_{i=1}^{N}\sum_{j=1}^{K} \beta_{ij}\varphi_c(X_{ij}), \quad c \in \{1, \cdots, N\}, \tag{8}$$

where $X_{ij}$ is the $j$-th input sample for the $i$-th class, $\varphi_c(\cdot) \in \mathcal{H}$ is a class-specific feature map and $\beta_{ij}$ indicates the importance of $X_{ij}$ for constituting the prototype vector $V_c$.

In other words, there exists a feature map $\varphi_c$ for each class $c$, and the support set is mapped to the corresponding feature map and then weighted-averaged to constitute the prototype vector of the corresponding class. At inference time, the classification of a given query $X$ is done by taking the class of the most similar prototype vector as follows:

$$\hat{c} = \arg\max_c \langle V_c, \varphi(X)\rangle. \tag{9}$$

Here, $\varphi : \mathcal{X} \to H$ is a non-class-specific mapping. We can also rewrite the prototype vector using $\varphi$ and by defining a projection $P_c : \mathcal{X} \to \mathcal{X}$ as

$$P_c(X) = \begin{cases} X & \text{if } y(X) = c, \\ \nu \in \mathcal{N}(\varphi), & \text{if } y(X) \neq c \end{cases} \tag{10}$$

where $y(X)$ is the ground truth class of $X$ and $\mathcal{N}$ is the null space of $\varphi$ *i.e.*, $\varphi(\nu) = 0$.

Then by defining $\varphi_c \triangleq \varphi \circ P_c$ and $\beta_{ij} \triangleq \frac{1}{K}$, it becomes

$$V_c = \frac{1}{K}\sum_{j=1}^{K} \varphi(X_{cj}). \tag{11}$$

**SGD in the inner loop** If GBML satisfies the hypothesis of linearity in the inner loop, $f$ is locally linear in $\theta$ in an inner loop. More specifically, there exists an equivalent feature map $\varphi_c : \mathcal{X} \to \mathcal{H}$ which satisfies $f_c(\cdot|\theta_c) = \langle \theta_c, \varphi_c(\cdot)\rangle$ for every $x \in \mathcal{X}$ where $f(\cdot|\theta) = [f_1(\cdot|\theta_1), \cdots, f_N(\cdot|\theta_N)]^T$.

With the loss function $L(x, y|\theta) = D(s(f(x|\theta)), y)$ for some distance measure $D$ such as cross entropy, we can formulate the inner loop of $N$-way $K$-shot meta learning by SGD as

$$\theta_c^{k+1} = \theta_c^k - \alpha \sum_{i=1}^{N} \sum_{j=1}^{K} \frac{\partial L(X_{ij}, y(X_{ij})|\theta)}{\partial \theta_c} = \theta_c^k - \alpha \sum_{i=1}^{N} \sum_{j=1}^{K} \frac{\partial D}{\partial f_c} \varphi_c(X_{ij}), \tag{12}$$

since all samples in the support set are inputted in a batch of an inner loop.

Because the model is linear in the inner loop, the batch gradient does not change. Let $\beta_{ij} = -\frac{\partial D}{\partial f_c}|_{\theta_c^0, X_{ij}}$. Then after $t$ steps, by (8), the model becomes

$$\theta_c^t = \theta_c^0 + \alpha t \sum_{i=1}^{N} \sum_{j=1}^{K} \beta_{ij} \varphi_c(X_{ij}) = \theta_c^0 + \alpha t V_c. \tag{13}$$

At the initialization step of an inner loop, there is no information about the class, even the configuration order of the class, because the task is randomly sampled. If so, the problem is solved in the inner loop. For example, if a class *Dog* is allocated to a specific index such as *Class 3*. There is no guarantee that it will have the identical index the next time the class *Dog* comes in. Thus, at a meta-initialization point $\theta^0$, the scores for different classes would not be much different, *i.e.*, $f_i(x|\theta_i^0) \simeq f_j(x|\theta_j^0)$ for $i, j \in [1, \cdots N]$.

Considering the goal of classification is achieved through relative values between $f_i(X)$'s, the value at the initialization point does not need to be considered significantly. Therefore

$$\arg\max_c f_c(X) = \arg\max_c \langle \theta_c^t, \varphi(X) \rangle = \arg\max_c \langle \theta_c^0 + \alpha t V_c, \varphi(X) \rangle \sim \arg\max_c \langle V_c, \varphi(X) \rangle \tag{14}$$

So inner loop in GBML can be interpreted as making proptotype vector with given support set. $\square$

Table 9: Test accuracy % of 4-conv network on benchmark data sets. The values in parentheses indicate the number of shots. The best accuracy among different methods is bold-faced. To differentiate the notation, we have denoted SHOT$_3$ as a model that uses 3 optimization steps in $f_r$ and 1 optimization step in $f_t$, and SHOT$_6$ as a model that uses 6 optimization steps in $f_r$ and 3 optimization steps in $f_t$.

| meta-train | miniImageNet | | | Cars | | |
|---|---|---|---|---|---|---|
| meta-test | miniImageNet | tieredImageNet | Cars | Cars | miniImageNet | CUB |
| MAML (1) | 47.88 ± 0.55 | **51.84** ± 0.24 | 34.41 ± 0.47 | 47.78 ± 0.99 | 28.67 ± 1.17 | 30.95 ± 1.41 |
| MAML + SHOT$_3$ (1) | 47.97 ± 0.71 | 51.68 ± 0.68 | 34.03 ± 1.11 | **50.44** ± 0.62 | **30.19** ± 0.50 | **31.21** ± 0.64 |
| MAML + SHOT$_6$ (1) | **48.15** ± 0.31 | 51.67 ± 0.73 | **34.79** ± 0.70 | 49.89 ± 0.17 | 28.39 ± 0.37 | 30.83 ± 0.26 |
| MAML (5) | 64.81 ± 1.63 | 67.96 ± 1.22 | 46.57 ± 0.53 | 62.24 ± 2.01 | 37.23 ± 1.95 | 41.84 ± 1.25 |
| MAML + SHOT$_3$ (5) | 66.86 ± 0.58 | 69.48 ± 0.17 | **48.42** ± 0.72 | **69.08** ± 0.31 | **40.79** ± 0.93 | **43.46** ± 0.89 |
| MAML + SHOT$_6$ (5) | 66.27 ± 0.23 | **69.60** ± 0.31 | 45.83 ± 1.82 | 66.37 ± 1.93 | 39.35 ± 0.74 | 42.63 ± 0.24 |

## E SHOT with more optimization step in the inner loop

In the main paper, we used only one step for the target model to improve computation efficiency. However, it's also important to test if SHOT works with more optimization steps in the inner loop. As a reference model, we set the number of optimization steps to 6, which is different from the main paper where we only used 3 steps (same as the baseline). As shown in Table 9, SHOT still performs better than the baseline even with more optimization steps in the inner loop.