2021

인공지능세미나

정보찬

AILab.

2021.06.10.

주식회사 바스젠바이오

Meta Curvature

1 Keywords

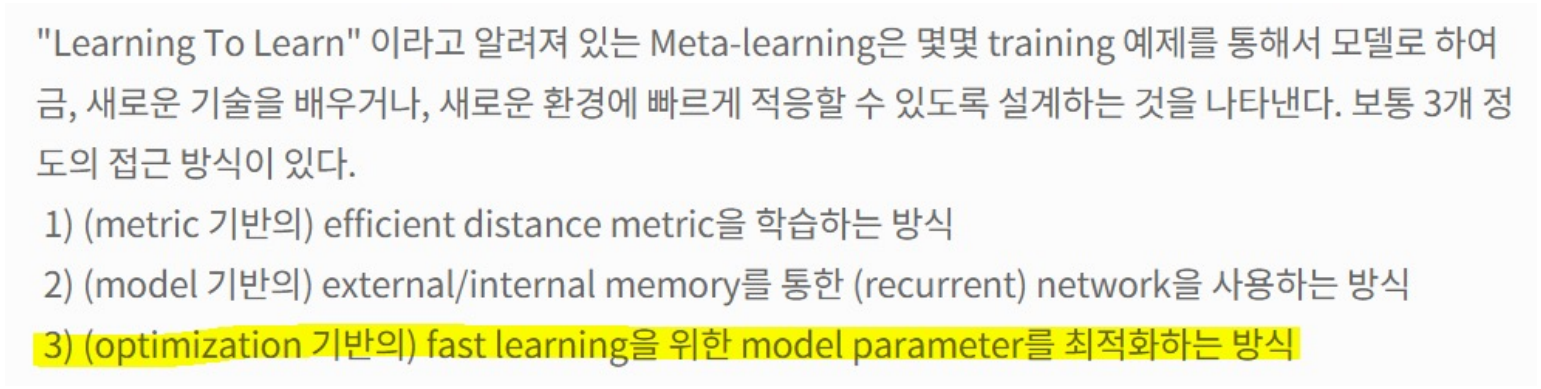
1. Meta Learning

2. Model Agnostic Meta Learning

3. Meta Curvature

1 Review

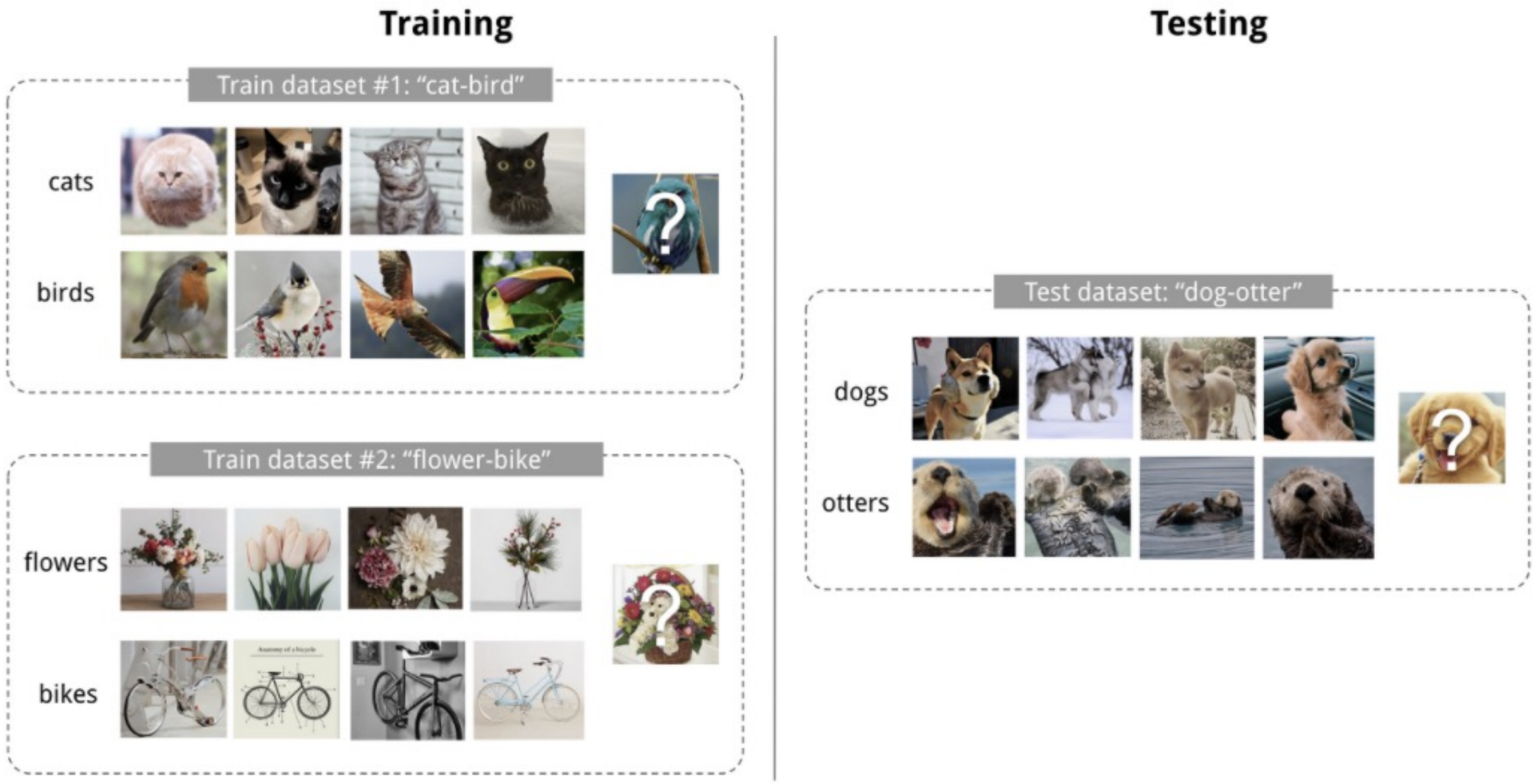
- Meta Learning이란


"Learning To Learn" 이라고 알려져 있는 Meta-learning은 몇몇 training 예제를 통해서 모델로 하여금, 새로운 기술을 배우거나, 새로운 환경에 빠르게 적응할 수 있도록 설계하는 것을 나타낸다. 보통 3개 정도의 접근 방식이 있다.

1) (metric 기반의) efficient distance metric을 학습하는 방식
2) (model 기반의) external/internal memory를 통한 (recurrent) network을 사용하는 방식
3) (optimization 기반의) fast learning을 위한 model parameter를 최적화하는 방식

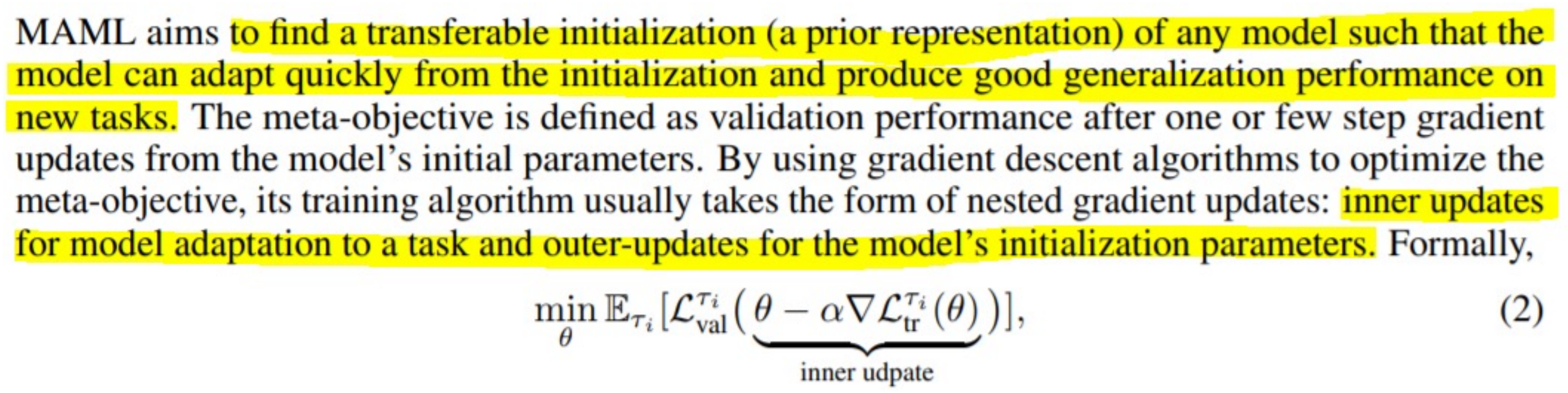
1 Review

- Meta Learning이란


Training
Testing
Train dataset #1: "cat-bird"
cats
birds
Train dataset #2: "flower-bike"
flowers
bikes
Test dataset: "dog-otter"
dogs
otters

1 Review

- Model Agnostic Meta Learning이란



MAML aims to find a transferable initialization (a prior representation) of any model such that the model can adapt quickly from the initialization and produce good generalization performance on new tasks. The meta-objective is defined as validation performance after one or few step gradient updates from the model's initial parameters. By using gradient descent algorithms to optimize the meta-objective, its training algorithm usually takes the form of nested gradient updates: inner updates for model adaptation to a task and outer-updates for the model's initialization parameters. Formally,

$$\min_{\theta} \mathbb{E}_{\mathcal{T}_i}[\mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\underbrace{\theta - \alpha\nabla\mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)}_{\text{inner udpate}})], \tag{2}$$

1 Review

• Model Agnostic Meta Learning이란

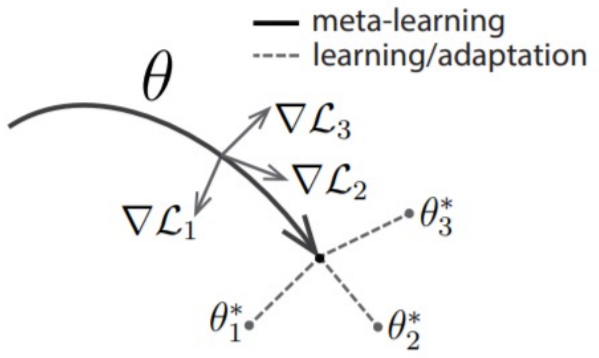
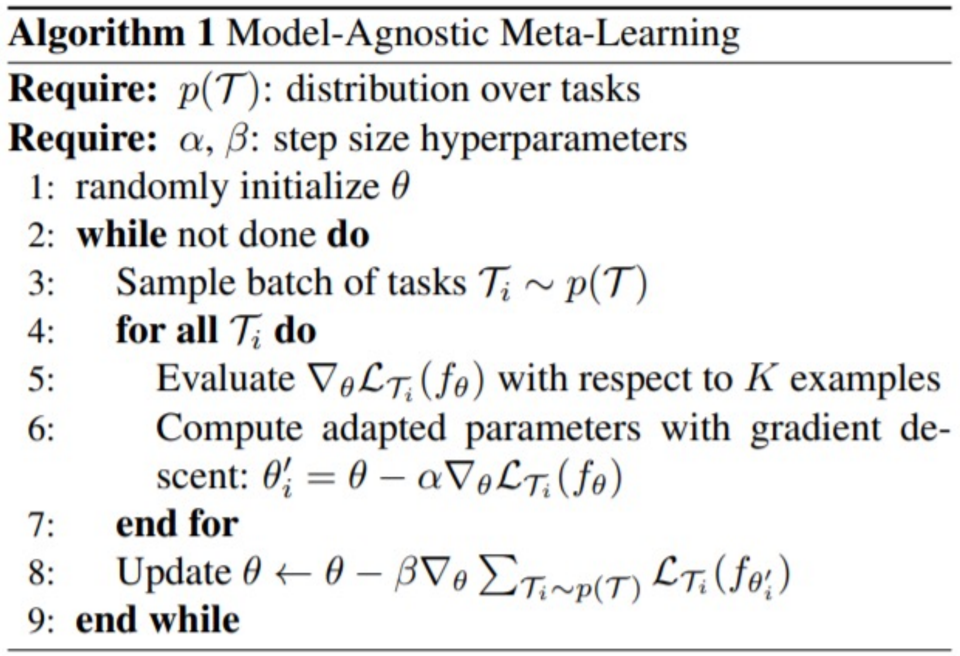


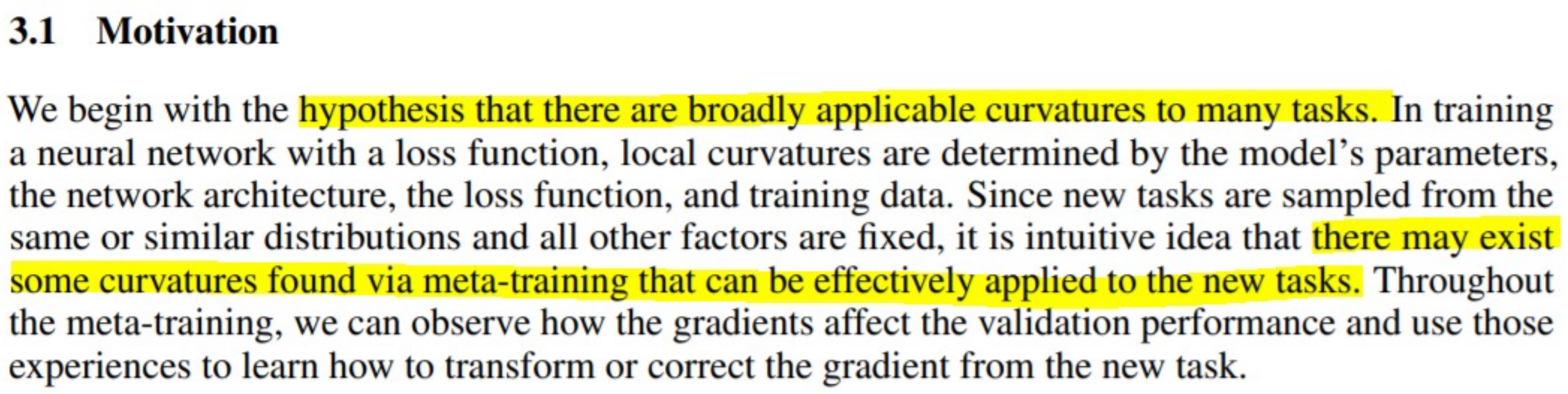
Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.



Algorithm 1 Model-Agnostic Meta-Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: while not done do
3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4:     for all $\mathcal{T}_i$ do
5:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
6:         Compute adapted parameters with gradient de-
           scent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7:     end for
8:     Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: end while
```

2 Motivation

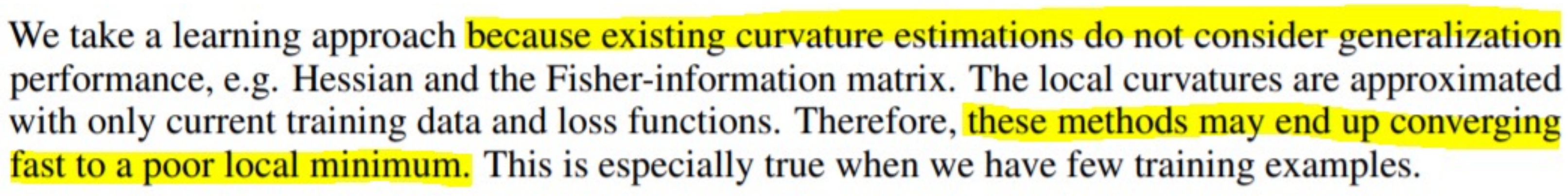


3.1 Motivation

We begin with the hypothesis that there are broadly applicable curvatures to many tasks. In training a neural network with a loss function, local curvatures are determined by the model's parameters, the network architecture, the loss function, and training data. Since new tasks are sampled from the same or similar distributions and all other factors are fixed, it is intuitive idea that there may exist some curvatures found via meta-training that can be effectively applied to the new tasks. Throughout the meta-training, we can observe how the gradients affect the validation performance and use those experiences to learn how to transform or correct the gradient from the new task.

- 다양한 task에서 활용 가능한 커베쳐가 존재할 것이라 가정

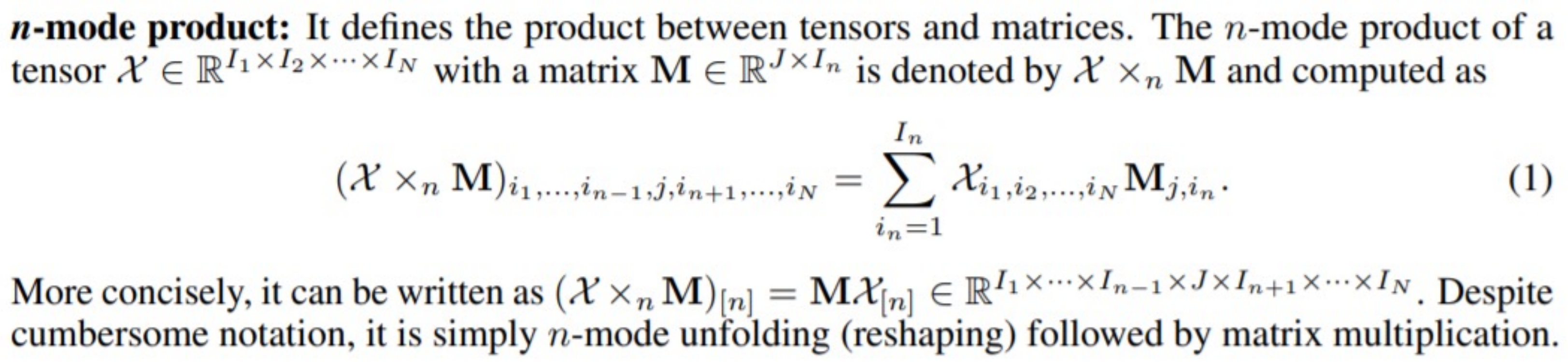
2 Motivation


We take a learning approach because existing curvature estimations do not consider generalization performance, e.g. Hessian and the Fisher-information matrix. The local curvatures are approximated with only current training data and loss functions. Therefore, these methods may end up converging fast to a poor local minimum. This is especially true when we have few training examples.

- 알려져 있는 커베쳐를 근사법은 일반화 성능을 고려하지 않았기 때문에 엉뚱한 local minimum에 빠지기 쉬움.
- 이를 해결하기 학습 가능한 커베쳐를 이용

2 Notation

- Notation



***n*-mode product:** It defines the product between tensors and matrices. The n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{M} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{M}$ and computed as

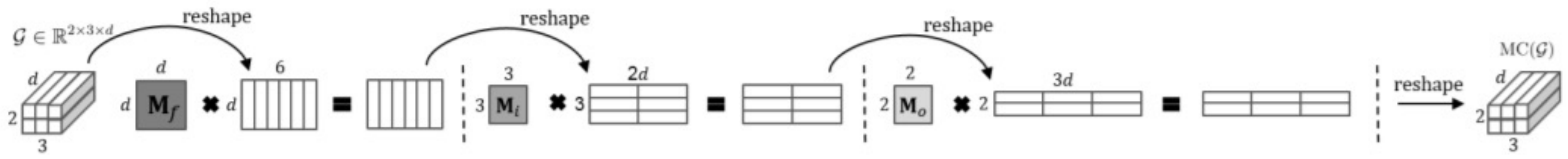
$$(\mathcal{X} \times_n \mathbf{M})_{i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N} = \sum_{i_n=1}^{I_n} \mathcal{X}_{i_1, i_2, \ldots, i_N} \mathbf{M}_{j, i_n}. \tag{1}$$

More concisely, it can be written as $(\mathcal{X} \times_n \mathbf{M})_{[n]} = \mathbf{M}\mathcal{X}_{[n]} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$. Despite cumbersome notation, it is simply n-mode unfolding (reshaping) followed by matrix multiplication.

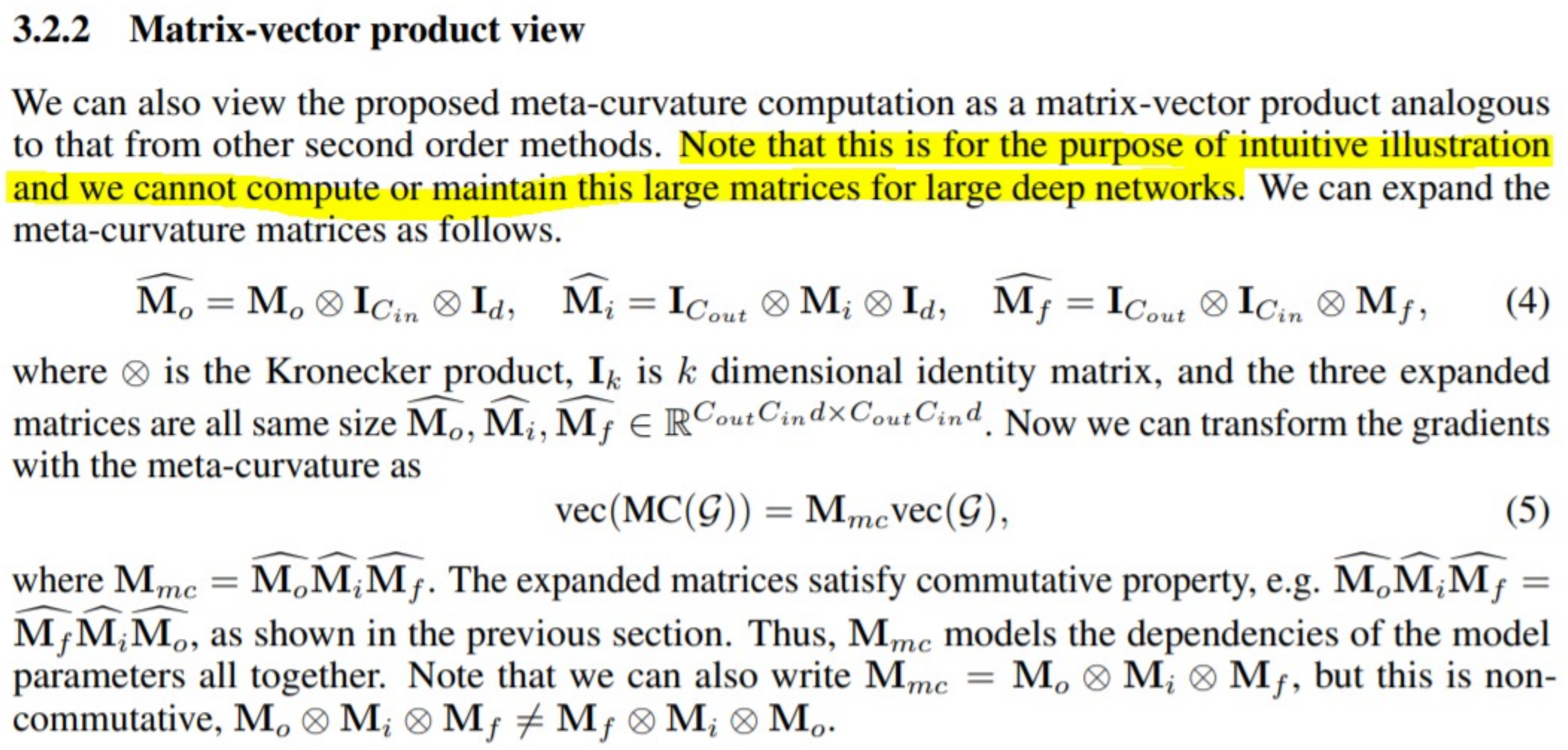
3 Meta Curvature

We consider neural networks as our models. With a slight abuse of notation, let the model's parameters $\mathcal{W}^l \in \mathbb{R}^{C^l_{out} \times C^l_{in} \times d^l}$ and its gradients of loss function $\mathcal{G}^l \in \mathbb{R}^{C^l_{out} \times C^l_{in} \times d^l}$, at each layers l. To avoid cluttered notation, we will omit the superscript l. We choose superscripts and dimensions with 2D convolutional layers in mind, but the method can be easily extended to higher dimension convolutional layers or other layers that consists of higher dimension parameters. C_{out}, C_{in}, and d are the number of output channels, the number of input channels, and the filter size respectively. d is height $\times$ width in convolutional layers and 1 in fully connected layers. We also define meta-curvature matrices, $\mathbf{M}_o \in \mathbb{R}^{C_{out} \times C_{out}}$, $\mathbf{M}_i \in \mathbb{R}^{C_{in} \times C_{in}}$, and $\mathbf{M}_f \in \mathbb{R}^{d \times d}$. Now a meta-curvature function takes a multidimensional tensor as an input and has all meta-curvature matrices as learnable parameters:

$$\mathrm{MC}(\mathcal{G}) = \mathcal{G} \times_3 \mathbf{M}_f \times_2 \mathbf{M}_i \times_1 \mathbf{M}_o. \tag{3}$$



3 Meta Curvature



3.2.2 Matrix-vector product view

We can also view the proposed meta-curvature computation as a matrix-vector product analogous to that from other second order methods. Note that this is for the purpose of intuitive illustration and we cannot compute or maintain this large matrices for large deep networks. We can expand the meta-curvature matrices as follows.

$$\widehat{\mathbf{M}_o} = \mathbf{M}_o \otimes \mathbf{I}_{C_{in}} \otimes \mathbf{I}_d, \quad \widehat{\mathbf{M}_i} = \mathbf{I}_{C_{out}} \otimes \mathbf{M}_i \otimes \mathbf{I}_d, \quad \widehat{\mathbf{M}_f} = \mathbf{I}_{C_{out}} \otimes \mathbf{I}_{C_{in}} \otimes \mathbf{M}_f, \tag{4}$$

where $\otimes$ is the Kronecker product, $\mathbf{I}_k$ is k dimensional identity matrix, and the three expanded matrices are all same size $\widehat{\mathbf{M}_o}, \widehat{\mathbf{M}_i}, \widehat{\mathbf{M}_f} \in \mathbb{R}^{C_{out}C_{in}d \times C_{out}C_{in}d}$. Now we can transform the gradients with the meta-curvature as

$$\text{vec}(\text{MC}(\mathcal{G})) = \mathbf{M}_{mc}\text{vec}(\mathcal{G}), \tag{5}$$

where $\mathbf{M}_{mc} = \widehat{\mathbf{M}_o}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_f}$. The expanded matrices satisfy commutative property, e.g. $\widehat{\mathbf{M}_o}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_f} = \widehat{\mathbf{M}_f}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_o}$, as shown in the previous section. Thus, $\mathbf{M}_{mc}$ models the dependencies of the model parameters all together. Note that we can also write $\mathbf{M}_{mc} = \mathbf{M}_o \otimes \mathbf{M}_i \otimes \mathbf{M}_f$, but this is non-commutative, $\mathbf{M}_o \otimes \mathbf{M}_i \otimes \mathbf{M}_f \neq \mathbf{M}_f \otimes \mathbf{M}_i \otimes \mathbf{M}_o$.

3 Meta Curvature

If **A** is an $m \times n$ matrix and **B** is a $p \times q$ matrix, then the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is the $pm \times qn$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix},$$

more explicitly:

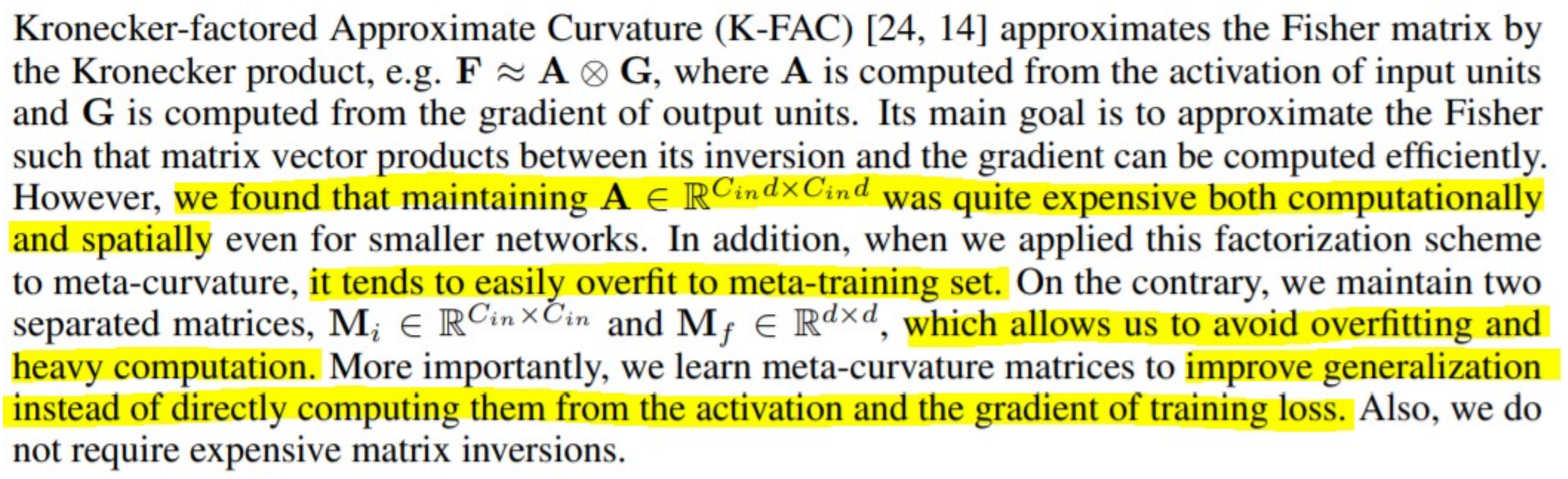
$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix}
a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\
a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\
\vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\
a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\
\vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\
a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\
a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\
\vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\
a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq}
\end{bmatrix}.$$

3 Meta Curvature

Examples

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \otimes \begin{bmatrix}0 & 5\\6 & 7\end{bmatrix} = \begin{bmatrix}1\begin{bmatrix}0 & 5\\6 & 7\end{bmatrix} & 2\begin{bmatrix}0 & 5\\6 & 7\end{bmatrix}\\3\begin{bmatrix}0 & 5\\6 & 7\end{bmatrix} & 4\begin{bmatrix}0 & 5\\6 & 7\end{bmatrix}\end{bmatrix} = \begin{bmatrix}1\times 0 & 1\times 5 & 2\times 0 & 2\times 5\\1\times 6 & 1\times 7 & 2\times 6 & 2\times 7\\3\times 0 & 3\times 5 & 4\times 0 & 4\times 5\\3\times 6 & 3\times 7 & 4\times 6 & 4\times 7\end{bmatrix} = \begin{bmatrix}0 & 5 & 0 & 10\\6 & 7 & 12 & 14\\0 & 15 & 0 & 20\\18 & 21 & 24 & 28\end{bmatrix}.$$

3 Meta Curvature


Kronecker-factored Approximate Curvature (K-FAC) [24, 14] approximates the Fisher matrix by the Kronecker product, e.g. $\mathbf{F} \approx \mathbf{A} \otimes \mathbf{G}$, where $\mathbf{A}$ is computed from the activation of input units and $\mathbf{G}$ is computed from the gradient of output units. Its main goal is to approximate the Fisher such that matrix vector products between its inversion and the gradient can be computed efficiently. However, we found that maintaining $\mathbf{A} \in \mathbb{R}^{C_{in}d \times C_{in}d}$ was quite expensive both computationally and spatially even for smaller networks. In addition, when we applied this factorization scheme to meta-curvature, it tends to easily overfit to meta-training set. On the contrary, we maintain two separated matrices, $\mathbf{M}_i \in \mathbb{R}^{C_{in} \times C_{in}}$ and $\mathbf{M}_f \in \mathbb{R}^{d \times d}$, which allows us to avoid overfitting and heavy computation. More importantly, we learn meta-curvature matrices to improve generalization instead of directly computing them from the activation and the gradient of training loss. Also, we do not require expensive matrix inversions.

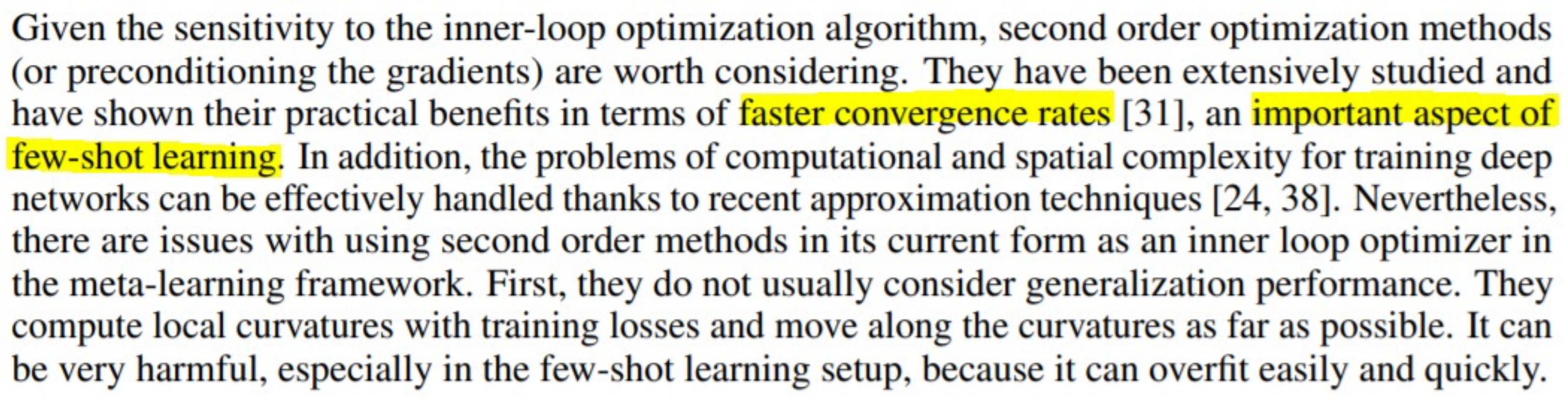
3 Meta Curvature

$$\nabla_{\mathbf{M}} \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}(\mathbf{M})) = -\alpha \nabla_{\theta^{\mathcal{T}_i}} \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}) \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)^\top$$

$$\mathbf{M}_{\mathcal{T}} = \mathbf{M} - \beta \sum_{i=1}^{|\mathcal{T}|} \nabla_{\mathbf{M}} \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}(\mathbf{M})) = \mathbf{M} + \alpha\beta \sum_{i=1}^{|\mathcal{T}|} \nabla_\theta \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}(\mathbf{M})) \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)^\top$$

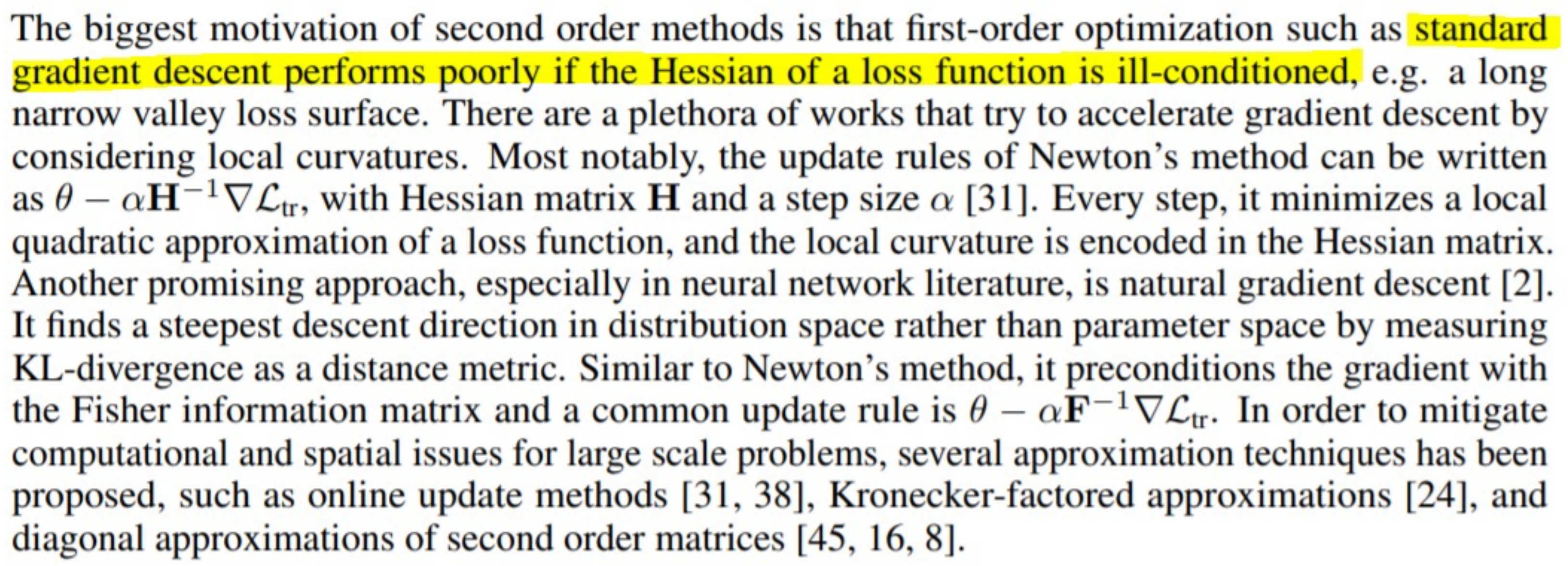
$$\begin{aligned}
\mathbf{M}_{\mathcal{T}} \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta) &= \Big(\mathbf{M} + \alpha\beta \sum_{i=1}^{|\mathcal{T}|} \nabla_\theta \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}) \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)^\top\Big) \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta) \\
&= \mathbf{M} \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta) + \beta \sum_{i=1}^{|\mathcal{T}|} \big(\nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)^\top \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta)\big) \alpha \nabla_\theta \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta^{\mathcal{T}_i}) \\
&= \mathbf{M} \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta) + \beta \sum_{i=1}^{|\mathcal{T}|} \big(\underbrace{\nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_i}(\theta)^\top \nabla_\theta \mathcal{L}_{\text{tr}}^{\mathcal{T}_{\text{new}}}(\theta)}_{\text{A. Gradient similarity}}\big)\big(\underbrace{\alpha \nabla_\theta \mathcal{L}_{\text{val}}^{\mathcal{T}_i}(\theta) + \mathcal{O}(\alpha^2)}_{\text{B. Taylor expansion}}\big).
\end{aligned}$$

3 Meta Curvature



Given the sensitivity to the inner-loop optimization algorithm, second order optimization methods (or preconditioning the gradients) are worth considering. They have been extensively studied and have shown their practical benefits in terms of faster convergence rates [31], an important aspect of few-shot learning. In addition, the problems of computational and spatial complexity for training deep networks can be effectively handled thanks to recent approximation techniques [24, 38]. Nevertheless, there are issues with using second order methods in its current form as an inner loop optimizer in the meta-learning framework. First, they do not usually consider generalization performance. They compute local curvatures with training losses and move along the curvatures as far as possible. It can be very harmful, especially in the few-shot learning setup, because it can overfit easily and quickly.

3 Meta Curvature



The biggest motivation of second order methods is that first-order optimization such as standard gradient descent performs poorly if the Hessian of a loss function is ill-conditioned, e.g. a long narrow valley loss surface. There are a plethora of works that try to accelerate gradient descent by considering local curvatures. Most notably, the update rules of Newton's method can be written as $\theta - \alpha \mathbf{H}^{-1} \nabla \mathcal{L}_{\text{tr}}$, with Hessian matrix $\mathbf{H}$ and a step size α [31]. Every step, it minimizes a local quadratic approximation of a loss function, and the local curvature is encoded in the Hessian matrix. Another promising approach, especially in neural network literature, is natural gradient descent [2]. It finds a steepest descent direction in distribution space rather than parameter space by measuring KL-divergence as a distance metric. Similar to Newton's method, it preconditions the gradient with the Fisher information matrix and a common update rule is $\theta - \alpha \mathbf{F}^{-1} \nabla \mathcal{L}_{\text{tr}}$. In order to mitigate computational and spatial issues for large scale problems, several approximation techniques has been proposed, such as online update methods [31, 38], Kronecker-factored approximations [24], and diagonal approximations of second order matrices [45, 16, 8].

4 Result

We evaluate the proposed method on a synthetic data few-shot regression task few-shot image classification tasks with Omniglot and MiniImagenet datasets. We test two versions of the meta-curvature. The first one, named as MC1, we fixed the $\mathbf{M}_o = \mathbf{I}$ Eq. 4. The second one, named as MC2, we learn all three meta-curvature matrices. We also report results on few-shot reinforcement learning in appendices.

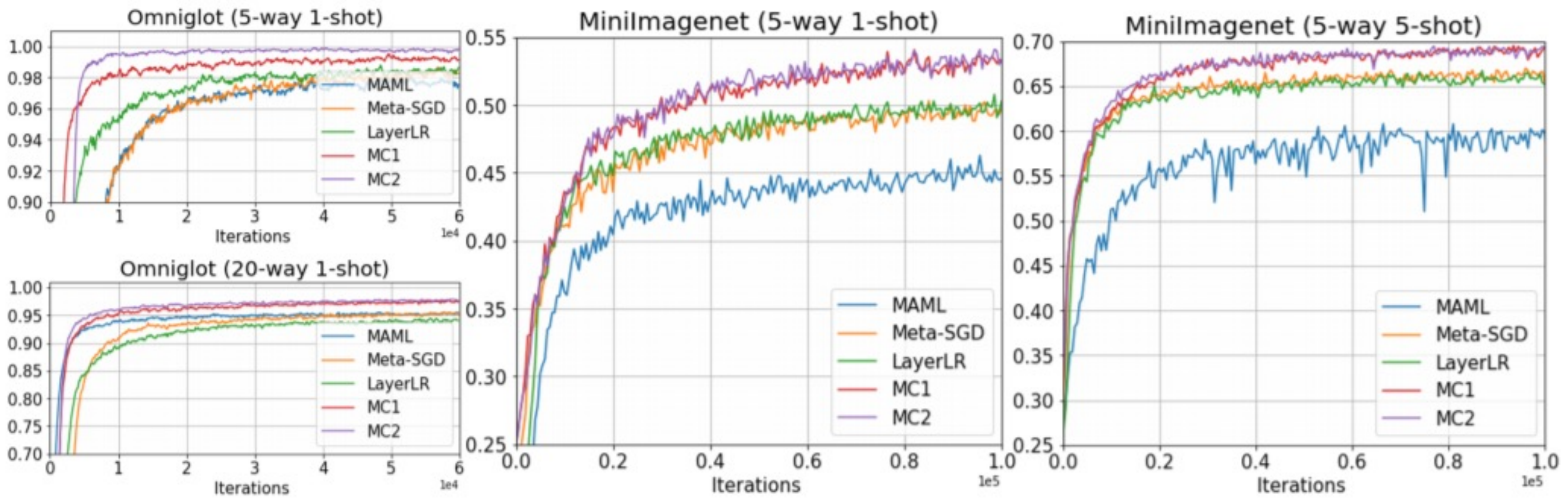


Figure 2: Few-shot classification accuracy over training iterations.

4 Result

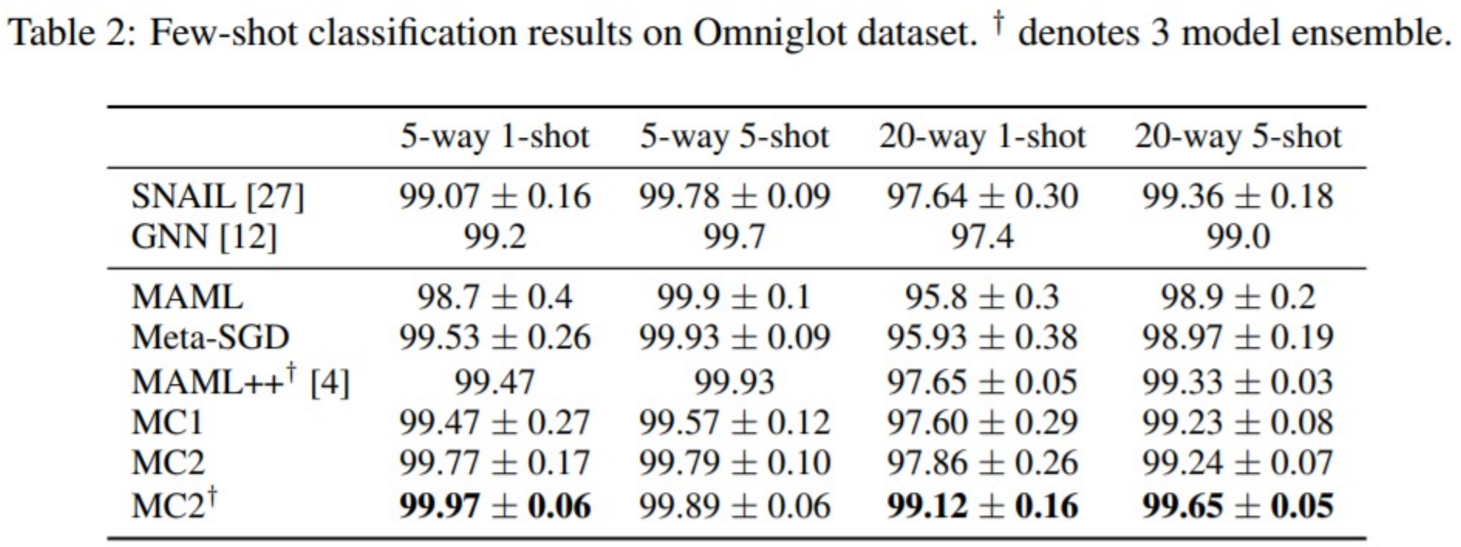


Table 2: Few-shot classification results on Omniglot dataset. † denotes 3 model ensemble.

	5-way 1-shot	5-way 5-shot	20-way 1-shot	20-way 5-shot
SNAIL [27]	99.07 ± 0.16	99.78 ± 0.09	97.64 ± 0.30	99.36 ± 0.18
GNN [12]	99.2	99.7	97.4	99.0
MAML	98.7 ± 0.4	99.9 ± 0.1	95.8 ± 0.3	98.9 ± 0.2
Meta-SGD	99.53 ± 0.26	99.93 ± 0.09	95.93 ± 0.38	98.97 ± 0.19
MAML++† [4]	99.47	99.93	97.65 ± 0.05	99.33 ± 0.03
MC1	99.47 ± 0.27	99.57 ± 0.12	97.60 ± 0.29	99.23 ± 0.08
MC2	99.77 ± 0.17	99.79 ± 0.10	97.86 ± 0.26	99.24 ± 0.07
MC2†	$\mathbf{99.97 \pm 0.06}$	99.89 ± 0.06	$\mathbf{99.12 \pm 0.16}$	$\mathbf{99.65 \pm 0.05}$