Understanding Train-Validation Split in Meta-Learning with Neural Networks

Xinzhe Zuo[1] Zixiang Chen[2] Huaxiu Yao[3] Yuan Cao[4,5]
Quanquan Gu[2]

[1]Department of Mathematics, UCLA

[2]Department of Computer Science, UCLA

[3]Department of Computer Science, Stanford

[4]Department of Statistics and Actuarial Science, University of Hong Kong

[5]Department of Mathematics, University of Hong Kong

April 28, 2023

Model Agnostic Meta-Learning[1]

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
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 descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
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**

Figure: MAML Algorithm by Finn et al. [1]. Usually, the samples used for the inner optimization steps (line 6) are different from samples used for the outer optimization steps (line 8). This is known as the **train-validation split**.

Motivation

Why study train-validation split?

- There exist algorithms (Reptile [2] and Meta-MinibatchProx [3]) that use all per task data for training and perform well on benchmark tasks.
- Since each task does not contain many samples, splitting each task into training set and validation set might hurt data efficiency.
- Bai et al. [4] showed that in the linearly realizable case, train-train method can aymptotically achieve better MSE than the train-validation method in a linear model.

Data model

We need to establish

1. the relation among different tasks
2. the data distribution for each specific task

To achieve this, we suppose that the data distribution $\mathcal{D}_k$ for the k-th task is defined based on a vector $\boldsymbol{\nu}_k$, i.e., $\mathcal{D}_k = \mathcal{D}(\boldsymbol{\nu}_k)$, and that the vectors $\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_K$ are independently drawn from a distribution Π.

Definition 1 (Distribution of tasks)

Let $\boldsymbol{\nu}, \mathbf{z}_1, \ldots, \mathbf{z}_M \in \mathbb{R}^d$ be fixed vectors, where $\mathbf{z}_1, \ldots, \mathbf{z}_M$ are orthogonal to $\boldsymbol{\nu}$. A vector $\tilde{\boldsymbol{\nu}}$ is generated from Π by (i) randomly pick a vector $\mathbf{z}$ from $\{\mathbf{z}_1, \ldots, \mathbf{z}_M\}$, and (ii) let $\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu} + \mathbf{z}$.

Data model

Definition 2 (Distribution of data)

Given a vector $\boldsymbol{\nu}_k \in \mathbb{R}^d$, each data point $(\mathbf{x}, y)$ with $\mathbf{x} = [\mathbf{x}^{(1)\top}, \mathbf{x}^{(2)\top}]^\top \in \mathbb{R}^{2d}$ and $y \in \{-1, 1\}$ is generated from $\mathcal{D}(\tilde{\boldsymbol{\nu}})$ as follows:

1. The label y is assigned as $+1$ or -1 with equal probability.
2. A noise vector $\boldsymbol{\xi}$ is generated from $\mathcal{N}(\mathbf{0}, \sigma_\xi^2 \cdot (\mathbf{I} - \mathbf{P}))$, where $\mathbf{P} \in \mathbb{R}^{d \times d}$ is the projection operator onto $\mathrm{span}(\{\boldsymbol{\nu}, \mathbf{z}_1, \ldots, \mathbf{z}_M\})$.
3. One of $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ is randomly selected and assigned as $y \cdot \boldsymbol{\nu}_k$; the other is assigned as $\boldsymbol{\xi}$.

For $k \in [K]$, we denote by $\mathcal{S}_k = \{(\mathbf{x}_{k,i}, y_{k,i})\}_{i=1}^n$ the set of independent samples from the k-th observed task.

Neural Network model

We study a two-layer CNN with m hidden layer neurons whose second layer weights are frozen as ± 1's. Let $\mathbf{W}$ represent the collection of all weights of our network. For a data input $\mathbf{x} = [\mathbf{x}^{(1)\top}, \mathbf{x}^{(2)\top}]^\top$, we consider the convolutional neural network $f(\mathbf{W}, \mathbf{x}) = F_{+1}(\mathbf{W}_{+1}, \mathbf{x}) - F_{-1}(\mathbf{W}_{-1}, \mathbf{x})$, where

$$F_j(\mathbf{W}_j, \mathbf{x}) = \sum_{r=1}^{m} \sum_{p=1}^{2} \sigma(\langle \mathbf{w}_{j,r}, \mathbf{x}^{(p)} \rangle), \ j \in \{-1, 1\}.$$

Here, for $j \in \{+1, -1\}$ and $r \in [m]$, we use $\mathbf{w}_{j,r}$ to denote the r-th convolution filter with second layer weight j, and use $\mathbf{W}_j$ to denote the collection of $\mathbf{w}_{j,1}, \ldots, \mathbf{w}_{j,m}$. The activation function $\sigma(\cdot)$ is the Huberized-ReLU function.

Neural Network model

We consider cross-entropy loss. The loss at a data point $(\mathbf{x}, y)$ is given as $\mathcal{L}(\mathbf{W}, \mathbf{x}, y) = \ell[y \cdot f(\mathbf{W}, \mathbf{x})]$, where $\ell(z) = \log(1 + \exp(-z))$. For a set of data points $\mathcal{S}$, we also define

$$\mathcal{L}(\mathbf{W}, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(\mathbf{x}, y) \in \mathcal{S}} \mathcal{L}(\mathbf{W}, \mathbf{x}, y). \tag{0.1}$$

Following the data model given in Definitions 1 and 2, We define the test loss achieved by a CNN with weights $\mathbf{W}$ as

$$\mathcal{L}_{\text{test}}(\mathbf{W}) := \mathbb{E}_{\tilde{\boldsymbol{\nu}} \sim \Pi, (\mathbf{x}, y) \sim \mathcal{D}(\tilde{\boldsymbol{\nu}})} \ell\left(y \cdot f(\mathbf{W}, \mathbf{x})\right). \tag{0.2}$$

Train-train method

Train-train: for each task k, we use all of the samples for adapting the parameter in the inner-loop updates. Specifically, the meta objective is to minimize

$$\widehat{\mathcal{L}}^{\text{tr-tr}}(\mathbf{W}, \{\mathcal{S}_k\}_{k=1}^{K}) = \frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(\widetilde{\mathbf{W}}(\mathbf{W}, \mathcal{S}_k), \mathcal{S}_k)\,, \tag{0.3}$$

where $\widetilde{\mathbf{W}}(\mathbf{W}, \mathcal{S}_k)$ represents the weights of the network after J gradient descent steps (w.r.t. loss $\mathcal{L}(\cdot, \mathcal{S}_k)$) starting from $\mathbf{W}$ with step size γ. The FOMAML algorithm updates the CNN weights using the following update rule:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \cdot \frac{1}{K}\sum_{k=1}^{K}\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}, \mathcal{S}_k)\big|_{\mathbf{W}=\widetilde{\mathbf{W}}(\mathbf{W}^{(t)}, \mathcal{S}_k)}\,. \tag{0.4}$$

Train-validation method

Train-validation: for each task k, we split the data with index sets $\mathcal{I}_k^{\mathrm{tr}} = \{1, \ldots, n_1\}$ and $\mathcal{I}_k^{\mathrm{val}} = \{n_1 + 1, \ldots, n\}$. We then use $\mathcal{S}_k^{\mathrm{tr}} = \{(\mathbf{x}_{k,i}, y_{k,i})\}_{i \in \mathcal{I}_k^{\mathrm{tr}}}$ as the training data set, and $\mathcal{S}_k^{\mathrm{val}} = \{(\mathbf{x}_{k,i}, y_{k,i})\}_{i \in \mathcal{I}_k^{\mathrm{val}}}$ as the validation data set. The meta objective of the train-validation method is to minimize

$$\widehat{\mathcal{L}}^{\text{tr-val}}(\mathbf{W}, \{\mathcal{S}_k\}_{k=1}^K) = \frac{1}{K} \sum_{k=1}^K \mathcal{L}(\widetilde{\mathbf{W}}(\mathbf{W}, \mathcal{S}_k^{\mathrm{tr}}), \mathcal{S}_k^{\mathrm{val}}), \tag{0.5}$$

where $\widetilde{\mathbf{W}}(\mathbf{W}, \mathcal{S}_k^{\mathrm{tr}})$ represents the weights of the network after J gradient descent steps (w.r.t. loss $\mathcal{L}(\cdot, \mathcal{S}_k^{\mathrm{tr}})$) starting from $\mathbf{W}$ with step size γ. For the train-validation method, the FOMAML algorithm implements the following outer-loop update rule to train the network:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \cdot \frac{1}{K} \sum_{k=1}^K \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathcal{S}_k^{\mathrm{val}})|_{\mathbf{W} = \widetilde{\mathbf{W}}(\mathbf{W}^{(t)}, \mathcal{S}_k^{\mathrm{tr}})}. \tag{0.6}$$

Additional Conditions

Condition 4.1

There exists $\sigma_s > 0$ *such that* $(1/2) \cdot \sigma_s \sqrt{d} \leq \|\mathbf{z}_i\|_2 \leq (3/2) \cdot \sigma_s \sqrt{d}$ *for* $i \in [M]$, *and* $\langle \mathbf{z}_i, \mathbf{z}_j \rangle \leq \mathcal{O}(\sigma_s^2 \cdot \sqrt{d \log(d)})$ *for all* $i \neq j$.

Condition 4.2

$\|\boldsymbol{\nu}\|_2 = 1$, $\sigma_\xi = d^{-1/2} \cdot \mathrm{polylog}(d)$, $\sigma_s = d^{-1/2}/\mathrm{polylog}(d)$, $n = \Theta(1)$, $K = \mathrm{polylog}(d)$, $m = \mathrm{polylog}(d)$, $\Omega(d^{1/2}) \leq M \leq d/2$.

Additional Conditions

Condition 4.3

We initialize the CNN weights $\mathbf{W}^{(0)}$ *by Gaussian random initialization with standard deviation* $\sigma_0 = d^{-1/2}$*. We set the inner-loop step size* $\gamma = \mathrm{polylog}(d)$*, and the outer-loop step size* $\eta = 1/\mathrm{polylog}(d)$*. We run* $T = \mathrm{poly}(d)$ *outer-loop iterations, and within each outer-loop iteration, we run* $J = 5$ *inner-loop gradient descent steps.*

Main results

Theorem 4.4

Under Conditions 4.1, 4.2 and 4.3, suppose that one uses the train-train method to train the neural network. Then with probability at least $1 - (Kn)^{-10}$,

1. *the training loss is small:*

$$\min_{t \in [T]} \widehat{\mathcal{L}}^{tr\text{-}tr}(\mathbf{W}^{(t)}, \{\mathcal{S}_k\}_{k=1}^{K}) \leq \mathcal{O}\left(\frac{1}{\mathrm{poly}(d)}\right).$$

2. *the test loss is large:*

$$\min_{t \in [T]} \mathcal{L}_{\mathrm{test}}(\mathbf{W}^{(t)}) = \Omega(1).$$

Main results

Theorem 4.5

Under Conditions 4.1, 4.2 and 4.3, suppose that one uses the train-validation method to train the neural network. Then with probability at least $1-(Kn)^{-10}$,

1. *the training loss is small:*

$$\min_{t\in[T]} \widehat{\mathcal{L}}^{tr\text{-}val}(\mathbf{W}^{(t)}, \{\mathcal{S}_k\}_{k=1}^{K}) \leq \mathcal{O}\left(\frac{1}{\mathrm{poly}(d)}\right).$$

2. *the test loss is also small: there exists a constant* $c>0$ *such that*

$$\mathcal{L}_{\mathrm{test}}(\mathbf{W}^{(T)}) = \mathcal{O}(\exp(-K^c)).$$

Proof intuitions

Lemma 6.1 (informal)

Under the train-train method, the inner loop **amplifies the noise inner products more than it does to the feature inner products**.

Lemma 6.2 (informal)

Under the train-validation method, the inner loop will have an **amplifying effect on the feature inner products**. *On the other hand, the inner loop* **does not change the noise inner products by a lot**.

The main reason the above lemma holds is because we assumed the **noise vector has a larger norm than the feature** (recall that $\|\boldsymbol{\xi}\|_2 = \Theta(\mathrm{polylog}(d))$, whereas $\|\boldsymbol{\nu}\|_2 = 1$).

Experiments

Table: Performance comparison of the number of optimization steps in the inner-loop.

# of Inner Steps	Setting	RainbowMNIST	miniImagenet
		Acc $\uparrow$	Acc $\uparrow$
1 step	Train-Train	$79.76 \pm 0.41\%$	$25.09 \pm 1.11\%$
	Train-Validation	$85.83 \pm 0.25\%$	$25.17 \pm 1.04\%$
5 steps	Train-Train	$65.32 \pm 0.54\%$	$25.93 \pm 1.10\%$
	Train-Validation	$87.52 \pm 0.20\%$	$46.15 \pm 1.36\%$

Experiments

Table: Performance w.r.t. the inner-loop learning rate and the outer-loop learning rate.

Setting	Learning Rate	RainbowMNIST	miniImagenet
		Acc ↑	Acc ↑
outer-lr $>$ inner-lr	Train-Train	$64.82 \pm 0.48\%$	$20.00 \pm 0.00\%$
	Train-Validation	$66.13 \pm 0.31\%$	$20.00 \pm 0.00\%$
outer-lr $<$ inner-lr	Train-Train	$65.32 \pm 0.54\%$	$20.00 \pm 0.10\%$
	Train-Validation	$87.52 \pm 0.20\%$	$46.15 \pm 1.36\%$

Reference - thank you!

C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

P. Zhou, X. Yuan, H. Xu, S. Yan, and J. Feng, "Efficient meta learning via minibatch proximal update," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

Y. Bai, M. Chen, P. Zhou, T. Zhao, J. Lee, S. Kakade, H. Wang, and C. Xiong, "How important is the train-validation split in meta-learning?," in *International Conference on Machine Learning*, pp. 543–553, PMLR, 2021.