Benchmarking Optimization-based Meta Learning Algorithms

Krishna Garg
kgarg8@uic.edu

Abstract

Moving forward past the era of deep learning where solving a singleton task would require massive datasets and enormous computational power, few-shot learning has gained a lot of momentum, where the objective is to make predictions with smaller training data. Few-shot learning particularly finds applications in robotic training, learning from small medical datasets, etc. Meta-Learning is a nice paradigm proposed in recent works to solve the few-shot learning problem where we learn some priors by training on similar tasks with again few samples and use them to generalize on new/ unseen tasks.

The report presents the overview of the Meta-Learning problem and benchmarks some of the popular optimization-based algorithms, viz., MAML, Reptile, First Order MAML on the two standard datasets used for the problem namely Omniglot and MiniImagenet. The Pytorch-based code will soon be released on Github[1].

1 Overview of Meta-Learning

1.1 Difference between Meta-Supervised Learning and Supervised Learning

Few-Shot learning is a problem where we have very few samples to learn a model to make predictions for the new samples. Consider the example of 3 paintings each by Braque and Cezanne as the training samples and we want to predict the correct label for a new painting whether it is painted by Braque or Cezanne. In general, our model will either overfit or would not perform well given such a small number of samples. This is in contrast to supervised learning where we have lots of labeled samples in our training dataset to learn a model that can generalize well to the new samples in the testing dataset.

Let us try to formalize the difference between Supervised Learning and Meta-Learning problem. The goal of supervised learning is to learn a mapping between the inputs and outputs. More formally, given a dataset D and a prior distribution over the parameters ϕ of a model (representing mapping function between inputs and outputs), we can find the point estimate of parameters ϕ that maximizes the posterior distribution of ϕ using the equation below:

Supervised Learning

$$\phi^* = \underset{\phi}{argmax} \log p(\phi|D) \tag{1}$$

But this requires lots of training data to generalize well and in some cases, huge training data is not available. In order to bridge this gap, Meta-Learning works by incorporating some additional data to train the neural network model in hope that fine-tuning on the few samples in D can lead to good generalization for the testing samples. The additional data is also called *meta-training dataset* and let it be denoted by $D_{meta-train}$. The meta-learning problem can be formulated as finding the optimal parameters ϕ given datasets D and $D_{meta-train}$ using the following equation:

[1] https://github.com/kgarg8/optim-meta

Meta Learning

$$\phi^* = \underset{\phi}{argmax} \log p(\phi|D, D_{meta-train}) \tag{2}$$

This additional data is not the same as the samples in D but is structurally related. The authors in the original paper denote structural similarity using a distribution over tasks but the idea is not very clear. I infer that distribution refers to the plot of the count of pixels against different intensity values and the distribution can be e.g. Normal, Gaussian, etc.

If we want to get rid of the meta-dataset $D_{meta-train}$ during testing time, we can condense all the knowledge learnt from the dataset into parameters θ^*. Thus the goal of meta-learning can be re-formulated as learning good parameters θ^* for the model using which generalization becomes feasible even with small samples. Equation 2 can be re-written like below (refer Appendix for the derivation):

$$\phi^* = \underset{\phi}{argmax} \log p(\phi|D, \theta^*) = f_{\theta^*}(D) \tag{3}$$

where f represents a model which takes in dataset D and is parametrized with θ^* which is given by equation below:

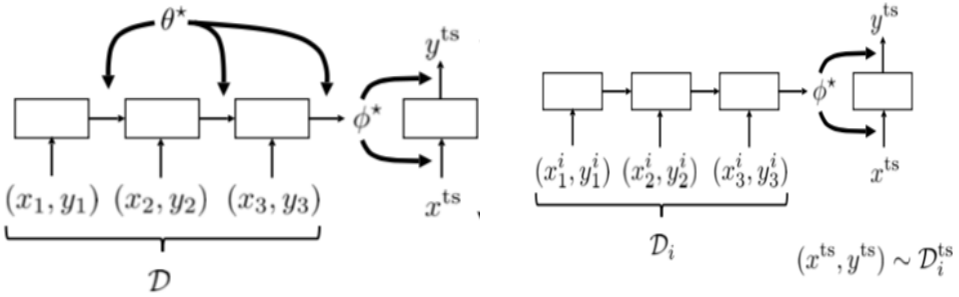
$$\theta^* = \underset{\theta}{argmax} \log p(\theta|D_{meta-train}) \tag{4}$$

Equation 4 is the **meta-learning problem** and and equation 3 refers to the **fine-tuning** or **adaptation** with respect to few samples in the meta-test set. Equation 4 can also be seen as *meta-training* and equation 3 can also be seen as *meta-testing*. θ^* are also called *meta-parameters*.

1.2 Meta-Learning Setup & Terminologies

Meta-testing: This takes the dataset D as input for the few-shot learning setup and training is done on a neural network parametrized with meta-parameters θ^* learnt from the additional dataset $D_{meta-train}$. After training on D, the model outputs the parameters ϕ^* (refer figure 1a).

Meta-training: [Vinyals *et al.*, 2016] reiterate a simple machine principle that *test and train conditions must match*. To make the similar conditions during meta-training, $D_{meta-train}$ is split into datasets D_i^{tr} where $i = 1, ..., n$. Additionally, a small test set D_i^{ts} is reserved corresponding to each D_i^{tr}. Sometimes in the literature, D_i^{tr} is also referred as *Support Set* and D_i^{ts} as *Query Set*. The pair {Support Set, Query Set} is also known as a *task*. Note that D_i^{ts} and D_i^{tr} are disjoint datasets so as to avoid neural network from simply memorizing the labels.



(a) Meta-testing (b) Meta-training

Figure 1: Figures depicting the setups for Meta-testing and Meta-training [Finn and Levine, 2019 accessed May 9 2020]

Table 1: Dataset splits

	Train	Val	Test
Omniglot	4800	-	1692
MiniImagenet	38400	9600	12000

To mimic the testing conditions, meta-training on D_i^{tr} is made to output the parameters ϕ_i for each task (refer figure 1b). If f represents a neural network for the dataset D_i^{tr}, then ϕ_i can be given by the equation below:

$$\phi_i = f_\theta(D_i^{tr}) \tag{5}$$

This is very similar to equation 3. Finally the meta-parameters θ^* can be computed by maximizing the posterior distribution of parameters ϕ_i for all the query sets D_i^{ts} as given in the below equation:

$$\theta^* = \underset{\theta}{argmax} \sum_{i=1}^{n} \log p(\phi_i | D_i^{ts}) \tag{6}$$

So, the equation 6 is finally the formulation of meta-learning problem. In summary, in absence of many training samples, we incorporate the knowledge from meta-dataset and meta-learning problem is basically to learn good meta-parameters from the meta-dataset.

N-WAY K-SHOT BENCHMARK IN META-LEARNING:
N-way denotes that there are N-classes used for the task. *K-shot* denotes that there are K-samples for each class in the Support Set. *Q-query* denotes that there are Q-queries for each class in the Query Set.

2 Datasets & Architecture

2.1 Datasets

OMNIGLOT is a dataset proposed by Lake et al. [Lake *et al.*, 2015] for learning human-like algorithms. It is a collection of 1623 character images belonging to 50 different classes. Each sample in the dataset was drawn by 20 Amazon Mechanical Turk workers. Every image was rotated by 90, 180, 270 degrees and thus the dataset was augmented with thrice the initial number of images. The splits are mentioned in the table 1.

MINIIMAGENET was first proposed by Vinyals et al. [Vinyals *et al.*, 2016], a subset of the popular ImageNet [Russakovsky *et al.*, 2015], ILSVRC-2012 dataset. The dataset contains 60,000 images categorized into 100 classes. The splits used for the experiments were different from the original splits and were borrowed from Ravi et al. [Ravi and Larochelle, 2016]. Refer table 1 for more details on the splits.

2.2 Architecture

A convolutional neural network (CNN) based deep architecture is used to learn the mapping of the images to an embedding space. The architecture stacks four convolution blocks. Each block contains a convolutional layer with a set of 64 filters of size 3*3, followed by a batch normalization layer, a ReLU [Nair and Hinton, 2010] non-linearity layer and a max-pooling layer with filters of size 2*2. The input to this architecture is an image and the output is a k-dimensional vector, where k = 64 for Omniglot images and 1600 for miniImageNet images.

Algorithm 1 MAML Algorithm

Input $\rho(T)$: distribution over tasks, β: step-size hyperparameter, k:gradient descent steps
Output θ: parameters of the model

Randomly initialize θ
while not converge **do**
  Sample batch of tasks from $\rho(T)$
  for all T_i from the batch **do**
    Evaluate $\triangledown L_{T_i}(\theta)$ for K examples sampled from T_i
    Compute $\theta_i = GD(L_{T_i}, k, \theta)$ ▷ Take k steps of Gradient Descent
  end for
  Update $\theta \leftarrow \theta$ - $\beta \triangledown_\theta \sum_{T_i \sim \rho(T)} L_{T_i}(\theta_i)$
end while

3 Optimization-based Meta-Learning Algorithms

3.1 Model-Agnostic Meta-Learning (MAML) Algorithm

3.2 First-Order MAML (FOMAML) Algorithm

Algorithm 2 FOMAML Algorithm

Input $\rho(T)$: distribution over tasks, β: step-size hyperparameter, k:gradient descent steps
Output θ: parameters of the model

Randomly initialize θ
while not converge **do**
  Sample batch of tasks from $\rho(T)$
  for all T_i from the batch **do**
    Evaluate $\triangledown L_{T_i}(\theta)$ for K examples sampled from T_i
    Compute $\theta_i = GD(L_{T_i}, k, \theta)$ ▷ Take k steps of Gradient Descent
  end for
  Update $\theta \leftarrow \theta$ - $\beta \triangledown_{\theta_i} \sum_{T_i \sim \rho(T)} L_{T_i}(\theta_i)$
end while

3.3 Reptile Algorithm

Algorithm 3 Reptile Algorithm

Input $\rho(T)$: distribution over tasks, β: step-size hyperparameter, k:gradient descent steps
Output θ: parameters of the model

Randomly initialize θ
while not converge **do**
  Sample batch of tasks from $\rho(T)$
  for all T_i from the batch **do**
    Compute $\theta_i = GD(L_{T_i}, k, \theta)$ ▷ Take k steps of Gradient Descent
  end for
  Update $\theta \leftarrow \theta + \frac{\beta}{n} \sum_{i=1}^{n} (\theta_i - \theta)$
end while

4 Results

The results on the datasets MiniImagenet and Omniglot are shown in the tables 2 and 3 respectively.

Table 2: Benchmarking results (accuracy) on MiniImageNet dataset (15-query samples)

MiniImagenet	MAML	FOMAML	Reptile
1-shot, 5-way	**0.46**	0.42	0.30
5-shot, 5-way	**0.65**	0.59	0.42

Table 3: Benchmarking results (accuracy) on Omniglot dataset (15-query samples)

Omniglot	MAML	FOMAML	Reptile
1-shot, 5-way	**0.86**	**0.86**	0.54
5-shot, 5-way	0.92	**0.97**	0.77
1-shot, 10-way	**0.74**	0.28	0.35
5-shot, 10-way	0.82	**0.94**	0.55

4.1 Training and Hyperparameters details

The models were trained until convergence which took roughly 100-400 epochs for each model. MAML was the most time-consuming and each model took at least a day on GeForce RTX 2080Ti. The results are reported on the test set and occasionally validation set (due to shortage of time), averaged over 150 batches.

4.2 Discussion on Results

- It can be observed that as we increase the number of shots keeping the number of classes the same, the accuracy increases. This is the expected behavior because a more number of shots better train the model and lead to better adaptation/ finetuning during validation/ testing.
- On MiniImagenet, MAML performs the best whereas on Omniglot, FOMAML [Nichol *et al.*, 2018] and MAML compete with each other on different scenarios. Except for 1-shot, 10-way on Omniglot dataset, the results for MAML and FOMAML are always comparable. This is the expected behavior because FOMAML just ignores the second-order derivative term while computing the gradients sometimes at the cost of accuracy but makes the training many times faster. During training, FOMAML was more than 4-5 times faster than MAML.
- Results for the Reptile algorithm could not be reproduced properly due to lack of time.

5 Conclusion

The Few Shot Learning via Meta-Learning has been clearly defined and the attempt was made to reproduce the results for the three popular algorithms, namely, MAML, FOMAML and Reptile algorithms. However, the results could be successfully reproduced for the first two (MAML and FOMAML) algorithms and the code will be open-sourced. The developers are invited to help improve the code, particularly, for Reptile.

Table 4: Hyperparameters

Hyperparameters	Values
Epochs for training	100-400
Inner Optimizer	SGD
Outer Optimizer	Adam
Inner_lr	1e-2
Inner steps	5
Outer_lr	1e-3
Scheduler	ReduceLROnPlateau

References

[Finn and Levine, 2019 accessed May 9 2020] Chelsea Finn and Sergey Levine. Meta-learning: from few-shot learning to rapid reinforcement learning, 2019 (accessed May 9 , 2020).

[Lake *et al.*, 2015] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[Nichol *et al.*, 2018] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[Ravi and Larochelle, 2016] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[Vinyals *et al.*, 2016] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

A Derivation of Meta-learning problem

Recall Eqn. 2:

$$
\begin{aligned}
\phi^* &= \underset{\phi}{argmax} \log p(\phi|D, D_{meta-train}) \\
&= \underset{\phi}{argmax} \log \int_\theta p(\phi|D,\theta) p(\theta|D_{meta-train}) d\theta \\
&\quad [\textit{Assuming } \phi \textit{ is conditionally independent of } (D_{meta-train}|\theta)] \\
&\approx \underset{\phi}{argmax} \log p(\phi|D,\theta^*) + \underbrace{\log p(\theta^*|D_{meta-train})}_{meta-learning} \qquad (\theta^* = \log p(\theta|D_{meta-train})) \\
\phi^* &\approx \underbrace{\underset{\phi}{argmax} \log p(\phi|D,\theta^*)}_{adaptation}
\end{aligned}
$$