An implementation of Model Agnostic Meta Learning (MAML) for few-shot supervised image classification

Final Project - Continual Learning 2022/23
University of Pisa, M.Sc. Computer Science

Irene Pisani - 560104
i.pisani1@studenti.unipi.it

Project objective

1. Provide an **implementation of MAML** for few shot supervised learning using **Pytorch**.

2. Reproduce the experiments on the **Omniglot dataset.** Follow the original experimental protocol of **N-way classification with 1 or 5 shots and N equal to 5 or 20.**

3. **Performance comparison** between the obtained performance and original ones.

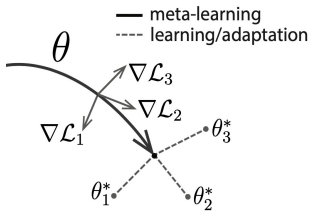
4. Analysis of the **impact of the number of inner SGD steps** during training and evaluation.

Introduction to MAML

- **Optimization-based meta-learning algorithm** compatible with any model trained with gradient descent and applicable to a variety of different learning problems (classification, regression, RL).
- A method that can **learn the parameters of any deep network via meta-learning** in order to **prepare that network for fast adaptation**.

Intuition: encouraging the learning of general-purpose internal representations applicable to a variety of tasks.

Find θ that are *sensitive* to changes in the task s.t. small changes in θ cause large improvements on the loss of any task.



MAML algorithm

Training inner loop

Sample a task Ti ~ P(T), the model is trained on support set with one or more SGD steps: θ → θi′.
The adapted model is tested on query set.

Training outer loop

Meta optimization of θ across tasks is performed via SGD and the query loss on sampled tasks serves as training loss.

Test phase

test, performance on new tasks sampled from P(T) are measured after learning from K samples.

Algorithm 2 MAML for Few-Shot Supervised 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: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$
11: **end while**

Critical points in development phase

1. **Due to initial lack of understanding, designing a metabatch sampler take considerable time.**
Other implementations of MAML available on Github helped with proper development.

2. **In the inner loop fit the model to a task and get query loss with θ' without overwriting the current θ values.** *
`functional_call (model, params, buffer, x)` allow to compute the model's output with the specified parameters.

3. **`backward()` method not exploitable during fast adaptation on support set due to θ-preserving constraint.** * Explicitly compute gradient of support loss and SGD weight update rule.

* Solved with pytorch.func functionalities

Preprocessing, architecture and Hyper-parameters

Data preprocessing:
Image resizing to 28 x 28
No image augmentations

Model's architecture replicates the original one:
4 modules with a 3 × 3 strided convolutions and 64 filters, followed by batch normalization, and ReLU non linearity.
Last hidden layer is fed into a soft-max.

- Inner optimizer: SGD
  Meta optimizer: Adam with lr = 0.001.
- Inner and meta loss: Cross Entropy.
- Due to computational constraints all the model were trained for max 100 epochs instead of 60000 iterations on CoLab GPU with **model checkpointing** based on VL loss.
- **No model selection**: hyper-parameters are the same used in the original paper.

Hyper-parameters	5-way classification	20-way classification
SGD step in TR	1 **	5**
SGD step in TS	3 **	5**
Meta batch-size	32	16
step size α	0.4	0.1

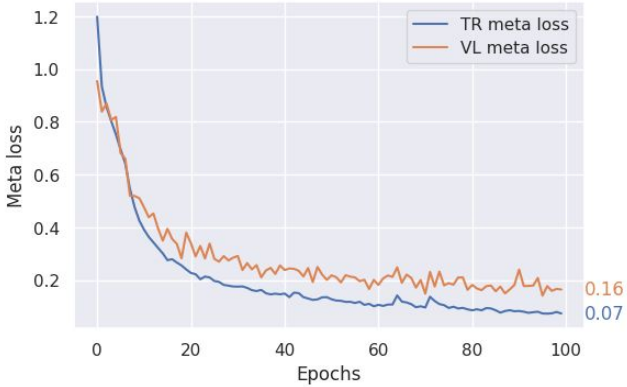
Performance comparison for 5-way classification

Small gap between original and obtained performances over TS set.

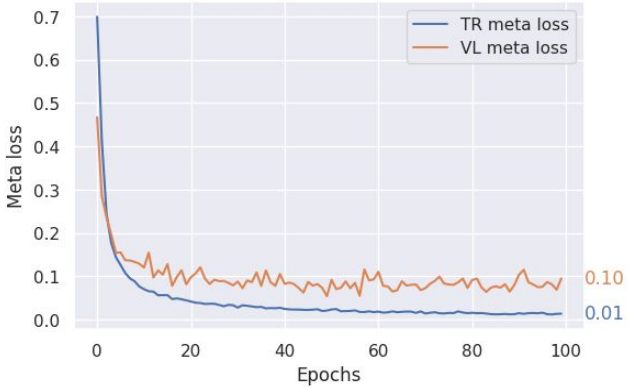
	1-shot	5-shot
MAML Pytorch	93.99 %	98.01 %
MAML (2017)	98.70 %	99.90 %

Increase the number of epochs to obtain better performance on TS set: TR and VL loss are still decreasing!

5 way
1 shot



5 way
5 shot



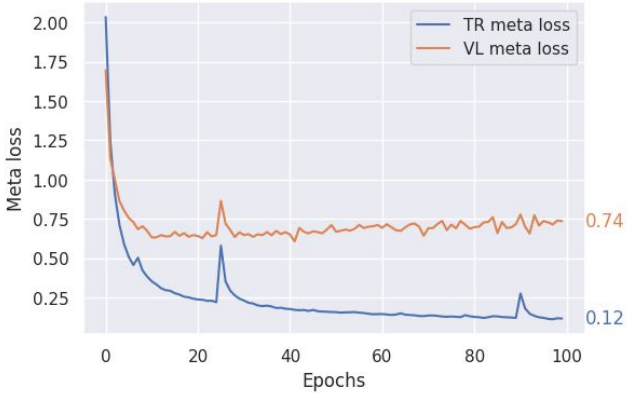
Performance comparison for 20-way classification

Significative gap between original and obtained performances over TS set.

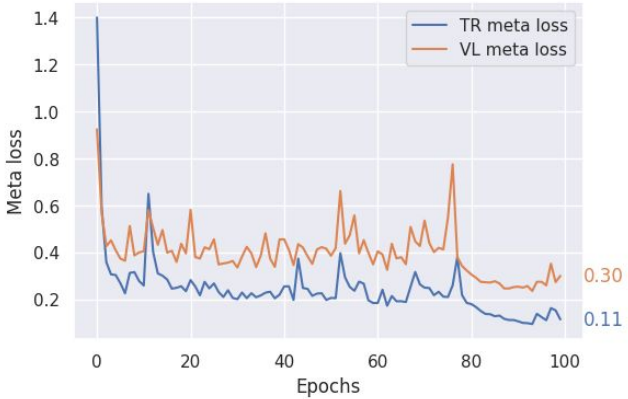
	1-shot	5-shot
MAML Pytorch	80.92 %	92.90 %
MAML (2017)	95.80 %	98.90 %

To obtain better performance increase the number of epochs, deal with overfitting introducing image augmentations, and handle instability by using gradient clipping.

20 way
1 shot



20 way
5 shot



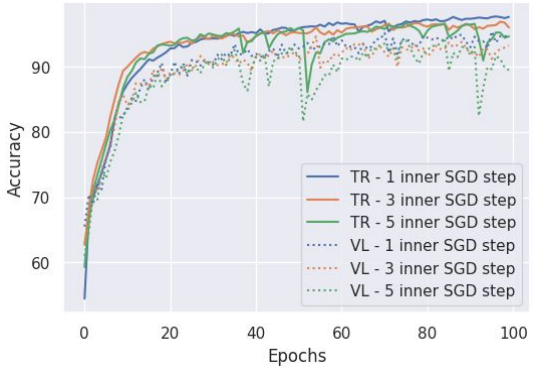
Impact of inner SGD steps during training

5 way: Increase num. of SGD step from 1 to 3 or 5 led to more training instability. The adapted parameter θ' computed in the inner loop are adapting too much to the support set → Larger loss on the query set → Bigger change in meta-update for θ parameter.

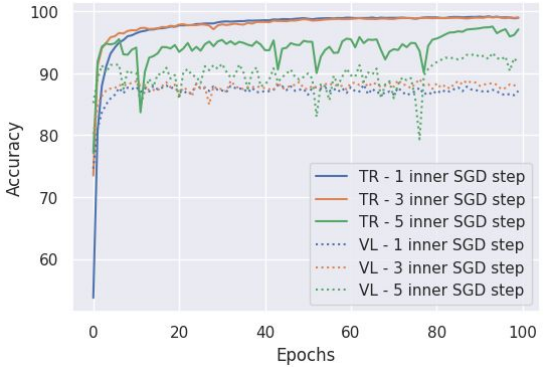
20 way: Decreasing the number of inner SGD step from 5 to 1 or 3 lead to more stable TR curves but less generalization capabilities on VL tasks.

The number of inner SGD step used in the inner loop impacts on training stability.

5 way 5 shot



20 way 5 shot



Impact of inner SGD steps during evaluation

For both 5 way and 20 way classification, as the number of inner steps used for evaluation increase TS performance increase as well.

Models continue to improve using more SGD step in evaluation, despite being trained for one or few gradient step. MAML optimizes θ to be sensitive to loss functions from the distribution of tasks p(T) rather than overfitting to θ that only improve after one step.

Increase too much the number of SGD step used for evaluation go against fast adaptation goal.

TR inner step	TS inner step	TS meta loss	TS meta accuracy
1.0	1	0.304097	90.638939
	3	0.274585	91.640863
	5	0.265397	91.937737
	10	0.262469	92.052079
3.0	1	0.936726	81.170189
	3	0.238914	92.763779
	5	0.230867	93.020831
	10	0.230742	93.009531
5.0	1	0.380145	88.146872
	3	0.262001	92.490221
	5	0.246181	92.900717
	10	0.230092	93.326519

20 way 5 shot

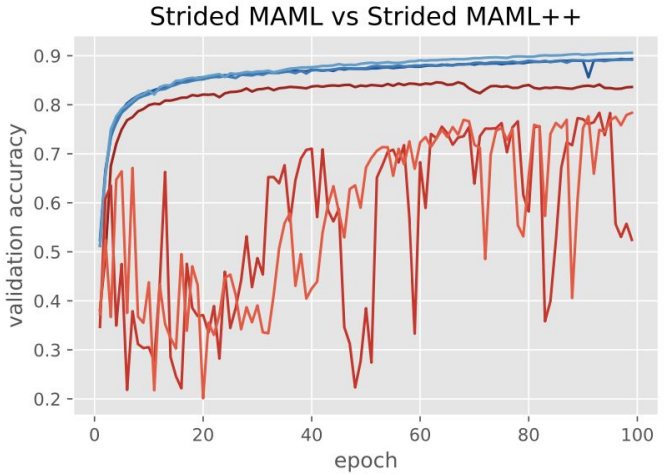
Conclusions and future works

MAML

Simple and powerful but it's **sensitive to NN architectures**, leads to training **instability**, requires **arduous hyperparameter searches** and it's **computationally expensive** at both training and inference times.

MAML ++

A modified version of MAML that **stabilize the system**, improve the **generalization performance**, **convergence speed** and **computational overhead**.



How to train your MAML

Thank you for the attention!

References

C. Finn, P. Abbeel, and S. Levine. *Model-agnostic meta-learning for fast adaptation of deep networks.* (2017).
A. Antoniou, A. Storkey, Harrison Edwards. *How to train your MAML.* (2019).