# Deep Meta-Learning
AML Assignment 2

## 1 Introduction

Deep neural networks have achieved enabled great successes in various areas. However, they are notoriously data-hungry: they require a lot of data to achieve a good level of performance. Naturally, this raises the question of how we can let them learn more quickly (from fewer data) just like humans.

Meta-learning is one approach that could achieve this by learning how to learn. In this assignment, you will study, implement, and investigate two popular meta-learning techniques, namely Prototypical Network (PNET) [5] and model-agnostic meta-learning (MAML) [3].

## 2 Few-shot learning

To investigate and study these techniques, we will use the few-shot learning setup, where techniques have to learn new tasks from a limited amount of data. More specifically, we will use the $N$-way $k$-shot image classification setting. Here, every task consists of a support set (the training data) and a query set (the test data) following these criteria:

- For a given task, the classes in the support and query set are the same (we want to measure how well we have learned on the support set, by evaluating on new inputs from the same classes)
- There are exactly $N$ classes in the support and query set per task
- The support set contains exactly $k$ examples per class
- The query set may contain a variable number of examples per class

An example of a task following the $N$-way $k$-shot setting is displayed in Figure 1.

### Omniglot

We will be using Omniglot [4]. The dataset consists of 1 623 classes (distinct characters) and contains 20 examples (black-white images) per class. This dataset is frequently used as a few-shot learning benchmark in the literature.

Meta-learning has 3 stages:

- Meta-training
- Meta-validation
- Meta-test

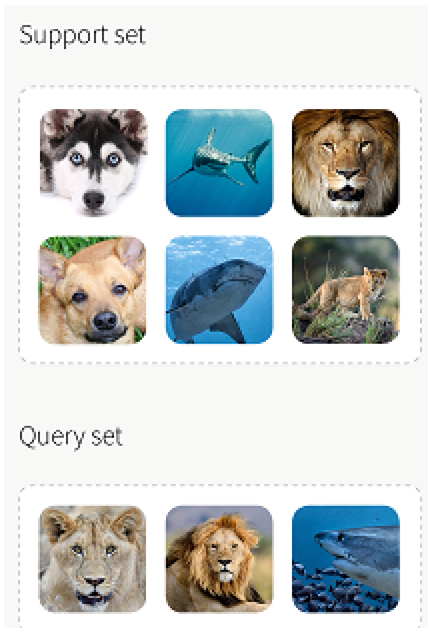

Figure 1: Example of a $N$-way $k$-shot task. Note that there are exactly $N$ classes and $k$ examples per class in the support set. Image source: [1].

The classes that are used for these 3 stages are distinct to ensure that we measure how well our techniques can learn new concepts/classes.

**Setup of data**

We provide the Omniglot dataset through Brightspace for you to download. Extract the data.zip such that you have a folder called `data` in the **same** directory as your Python files.

## 3 Methods

Below, we give a short summary of the methods that we will investigate as well as give some pointers to help you correctly implement the techniques.

### 3.1 Prototypical network

Prototypical network is a metric-based meta-learning techniques that computes predictions for new query input embedding $g_\theta(\hat{\mathbf{x}})$ by comparing it to class centroids $\mathbf{m}_n = \frac{1}{|X_n|}\sum_{\mathbf{x}_i \in X_n} g_\theta(\mathbf{x}_i)$ using a kernel $k_\theta(\hat{\mathbf{x}}, \mathbf{m}_n)$. Note that in theory (as discussed during the lecture) we could two separate networks: one to embed the query images, and one to embed the support images, but for simplicity, we use just one network $g_\theta$. The used kernel/similarity measure is the negative squared Euclidean distance, but other options exist. The class of the centroid closest to the query embedding $g_\theta(\hat{\mathbf{x}})$ is then predicted. In order to compute the kernel between the query input embedding and every centroid, you may find `torch.cdist` useful.

### 3.2 Model-agnostic meta-learning

Model-agnostic meta-learning, or MAML, aims to learn good initialization parameters $\theta$ such that new tasks can be learned within a few gradient update steps. This requires performing backpropagation through an optimization trajectory. In Pytorch, this can be a bit tricky as it needs to recognize that the task-specific parameters originate from the initialization. Below are some hints to help you with the implementation:

- Make a copy of the initial network weights by using `param.clone()`, where `param` is a tensor from the list of parameters. PyTorch then knows that the copy (called *fast weights*) originated from the initialization parameters. You can then adjust this copy using gradient update steps.
- Make sure to pass these fast weights when computing predictions, i.e., `preds = network(inputs, weights=fast_weights)`
- Make sure to study this resource before implementing
- Do not call `loss.backward()` in the code other than the call that is already in there as this can interfere with the meta-gradient signals.

### 3.3 Hyperparameters

Use the following hyperparameters in your experiments, unless explicitly requested otherwise.

- inner learning rate: 0.4
- outer learning rate: 0.001
- meta-batch size: 1
- outer optimizer: Adam
- number of inner gradient updates: 1
- number of meta-iterations: 40 000
- validation interval: 500 episodes (meta-batches)
- backbone: convolutional neural network (Conv4 in `networks.py`)
- For other hyperparameters that are not included in this list, use the defaults as provided in the code (unless mentioned otherwise)

## 4 Theory questions

Below are the questions that should be answered in your report. Answering these questions will also help you with the implementation of the methods.

1. Explain how from a given dataset with classes $\mathcal{Y}$, we can create/sample training, validation, and test tasks in an $N$-way $k$-shot manner. When and why do we use validation tasks? When and why do we use the test tasks?

| Support set | | |
|---|---|---|
| Identifier (i) | $g_\theta(\mathbf{x}_i)$ | Class |
| 1 | $[1, 2]$ | 1 |
| 2 | $[1, 3]$ | 1 |
| 3 | $[-1, -2]$ | 2 |
| 4 | $[-1, -3]$ | 2 |

Table 1: Support set for theory question 1. The identifier column corresponds to the subscript index of the support examples. Thus the example with identifier 1 can be written as $g_\theta(\mathbf{x}_1)$.

2. Suppose we have a binary classification task with a support set $D^{tr}_{\mathcal{T}_j}$ and a new query image $\hat{\mathbf{x}}$. Suppose that the support set is given in Table 1. Furthermore, suppose that $g_\theta(\hat{\mathbf{x}}) = [-1, 0.5]$. Show **step by step** how Prototypical network computes the prediction vector $\mathbf{y}_\theta(\hat{\mathbf{x}}) \in \mathbb{R}^2$. What is its prediction for class 1? And what is the prediction for class 2? Use the negative squared Euclidean distance as similarity measure (kernel).
3. Explain why Prototypical network is a meta-learning algorithm.
4. When is the meta-gradient $\nabla_\theta \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta'_j) = \nabla_\theta \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta^{(T)}_j)$ in MAML zero equal to zero? Interpret your findings: when does MAML update its initialization parameters $\theta$ and when does it not?
5. You have meta-trained MAML and ProtoNet on tasks from a single dataset. You now encounter a new task that is distant from the training task distribution. Which method do you expect to work better under the assumption that both techniques work equally well on tasks from the training distribution? Explain why.
6. In Prototypical network, the score of class $n$ of a new query input is proportional to the negative squared Euclidean distance $\mathbf{y}_\theta(\hat{\mathbf{x}})_n \propto -||g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_n||_2^2$. Suppose that we want to mimic this nearest-centroid type of behavior using a linear classifier of the form $f(\hat{\mathbf{x}}) = \mathbf{W} g_\theta(\hat{\mathbf{x}}) + \mathbf{b}$ where $\mathbf{W}$ is a weight matrix and $\mathbf{b}$ the bias vector. How do we have to set the weights of the $n$-th row of $\mathbf{W}$ and the $n$-th entry of the bias vector $b_n$ such that the predictions of the linear model are proportional to those of the Prototypical network?

## 5 Empirical questions

In your report, investigate the following empirically. Make sure to use the hyperparameter settings from from subsection 3.3 unless explicitly mentioned otherwise.

1. Show a plot of the meta-validation and meta-training losses over time in the 5-way, 1-shot setting (using 15 query examples/shots per class) for ProtoNet, first-order MAML, and second-order MAML. Interpret your findings.
2. Investigate the effect of the number of inner gradient update steps on the performance of first-order and second-order MAML in the 5-way 1-shot setting. Interpret your results.
3. Investigate the effect of increasing the number of classes per task on the performance of ProtoNet and first- and second-order MAML. Interpret your results.

Make sure to design your experiments in a rigorous way (e.g., how do you deal with randomness?)

## 6 Bonus (to be eligible for a 10)

Without doing this bonus component, the maximum grade is a 9. You are eligible to obtain a 10 if you also do at least one of the following:

- Investigate the influence of the similarity measure on the performance of ProtoNet. How well does the cosine similarity work instead of the negative squared Euclidean distance?
- Investigate the effect of the meta-batch size on the performance of MAML and ProtoNet. What do you find?

If you do the bonus, you have 1 page extra compared with the default page limit.

## 7 Work distrbution

Make sure to include the distribution of work in the assignment (who did what for the assignment). This way, we can weigh the grades based on individual contributions, if there is a large discrepancy between the work done by different members of the group. This section will not count towards the page limit.

## 8 Provided materials

We provide you with the file `a2.zip` which contains the following files:

- `main.py`: The main script to run for experiments. You do **not** have to change any code here. This code performs meta-training, meta-validation, and meta-testing. It writes the results to the ./results/ directory.
- `networks.py` Code that contains the two backbone networks: a feed-forward classifier and a convolutional one. In the experiments, we use the convolutional backbone. You do not need to change the file.
- `dataloaders.py` The file containing the data loader code (you should not change this file)
- `pnet` The place to implement Prototypical network. You only have to complete the apply function, which learns the task and makes predictions on the query inputs.
- `maml.py` The place to implement MAML. You only have to complete the apply function, which learns the task and makes predictions on the query inputs.

**Setup** For this assignment, we use PyTorch and Torchmeta [2]. We used the following versions:

- Torchmeta==1.8.0

- Torch==1.10.1+cu113

Make sure to install Torchmeta using `pip install torchmeta`.

## 9 Requirements/grading

The following requirements hold for this project:

- Do not plagiarize code or text. There will be a strict plagiarism check on both the code and the report that you submit.
- It is required that you implement the algorithms with PyTorch.
- You will write a report **of at most 4 pages (excluding references) using the provided template**. You are not allowed to deviate from this template (it is also not allowed to change the margins or font size in the template). The report should answer all questions above. We use the following criteria for grading
  - Correctness
  - Completeness
  - Rigorous experimental design that is suited for answering the empirical research questions posed in this assignment.
  - Detailed and clear description of the experiments, rendering them reproducible.
  - Good data visualization (think carefully about how to represent results!!–what type of plot, use of caption, use of legend, etc.)
  - Display of understanding: good analysis and interpretation of the experimental results. Clear and logical conclusions.
- You are **NOT** allowed to use existing libraries/repositories for the implementation of Prototypical network and MAML with the exception of the imports we have already done in the provided files.
- The report should be concisely written and void of spelling and grammar mistakes (consider using the Grammarly Chrome extension)
- To make a submission, upload all your code and the report to Brightspace through TurnitIn.
- The project deadline is due **25 November 2022 23:59 CET**

## 10 Helpful resources

You may find the resources below useful:

- `https://lilianweng.github.io/posts/2018-11-30-meta-learning/`
- `https://link.springer.com/content/pdf/10.1007/978-3-030-67024-5.pdf`

- `https://pytorch.org/tutorials/beginner/basics/intro.html`
- `http://luiz.hafemann.ca/libraries/2018/06/22/pytorch-doublebackprop/`
- `https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html`
- `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html`

And of course, the original papers [3, 5].

## References

[1] Tutorial few-shot learning. `https://www.borealisai.com/en/blog/tutorial-2-few-shot-learning-and-meta-learning-i/`. Accessed: 04-19-2022.

[2] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. Torchmeta: A Meta-Learning library for PyTorch, 2019. Available at: https://github.com/tristandeleu/pytorch-meta.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, ICML'17, pages 1126–1135. PMLR, 2017.

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

[5] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.