Hypernetworks for few-shot learning

Jakub Migdał

June 11, 2024

Project Description

Project Goals

The goal of the project was to generally become more acquainted with developing a machine learning project while also working on novel architectures with modern use-cases, which was a great fit for few-shot learning, a monumentally important problem in branches like computer vision, representation learning, and meta-learning.
The models in question, based on papers listed in the bibliography at the end of this doccument, are very recent developments originating from WMiI's GMUM's researchers, and the project itself is a fork of their repository with their original implementations.
Refined, nearly from-scratch implementations are the result of this project, with code focused on comprehensibility of fairly complex architectures and heuristics utilised.

Functionality

The repository provides a simple way to train a number of meta-learning models on the provided datasets.
As of now, training on the `omniglot` dataset is possible, and validation on `omniglot` and `EMNIST`.
The fully supported learning methods include `--method=maml`, `--method=hyper_maml` and `--method=bayes_hmaml`, corresponding to the `MAML`, `HyperMAML`, and `BayesHMAML` models described in the architecture section

Intended Usage

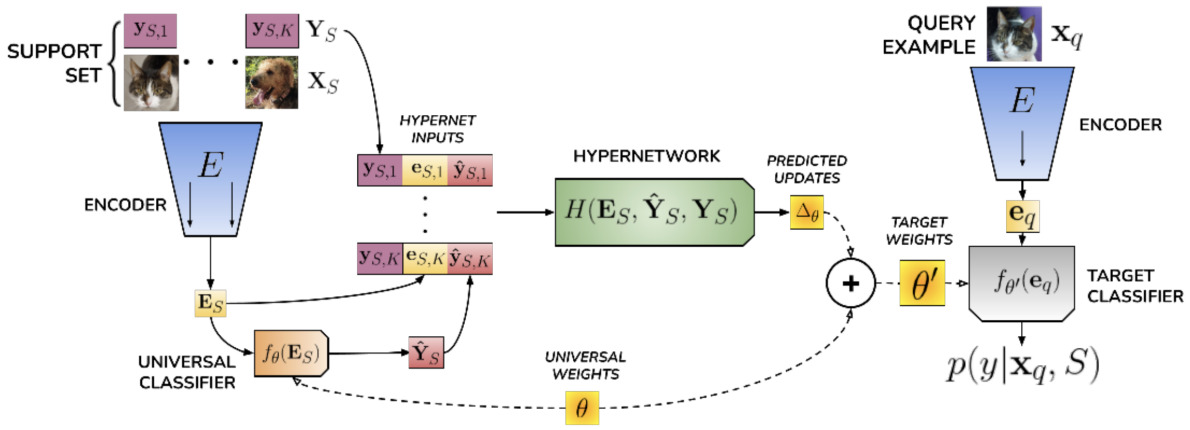
To begin with, it is recommended to create a virtual environment inside the project directory, for example using `python -m venv .venv`.
To install required dependencies, use `pip install -r requirements.txt`
To download the required repositories, there are shell scripts provided inside the `filelists` directory, one for each corresponding dataset. The dependencies to these scripts are `wget` and `unzip`.
There are two python entrypoints that will be of interest to the end user, `train.py` and `test.py`, both should be run from the CLI, preferably one that is long-session friendly like tmux.
Both provide a similar interface in terms of options, which are described by passing `-h` to the script. The required and most important ones are `--dataset` and `--method` which specify the dataset and learning method respectively. Example commands are listed in the `commands.sh` file at the project root.
For enhanced efficiency the user should also make sure `PyTorch` is configured to work with CUDA, and while a GPU is not required, whether it is visible to the runtime.

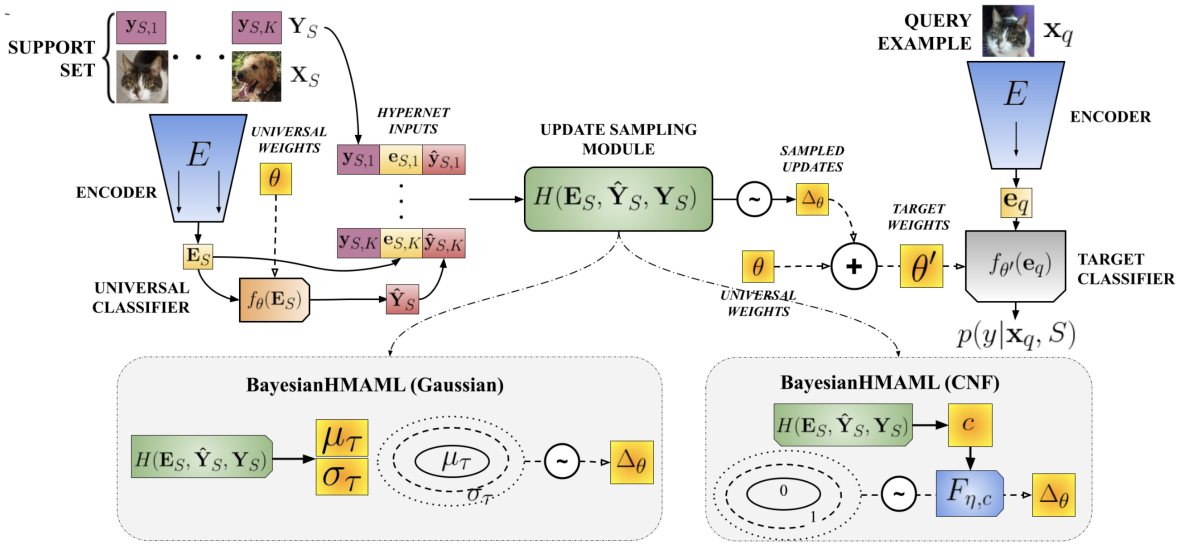
Project Architecture

Technology Stack

The utilised technologies mainly revolved around the chosen programming language - *python*. The general framework used is *PyTorch 2.23*, and to a lesser extent, *NumPy* and *torchmeta*, which allowed a paradigm shift to a more functional model structuring.

Application Architecture





The provided diagrams describe the structure of `HyperMAML` and `BayesHMAML`. Since MAML is more of an algorithm than an architecture, no diagram for its class is presented. Modules that were also used but are not the focus of the project include custom `DataLoader` classes and a number of vanilla networks made from composing standard `PyTorch` building blocks

System Design Justification

The main difference between the original design and the one employed in this project is the usage of `torchmeta.MetaModule` as the backbone for all modules, which allowed decoupling of parameters from architecture. This resulted in cleaner, more concise and idiomatic code, and in addition fixed a memory leak in the original `MAML` implementation, and a logical bug in `HyperMAML`.
The original implementation made heavy use of bootstrapping duplicate model parameters via reflection, which necessitated being mindful at all times of their current state, whether they are initialised or not, and left little transparency in terms of how memory was handled.

1 References

MAML: https://arxiv.org/pdf/1703.03400
HyperMAML: https://arxiv.org/pdf/2205.15745
BayesHMAML: https://arxiv.org/pdf/2210.02796