# CS330 Autumn 2021 Homework 2
# Prototypical Networks and Model-Agnostic Meta-Learning

Due Monday October 18, 11:59 PM PST

SUNet ID:
Name:
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

## Overview

In this assignment, you will experiment with two meta-learning algorithms, prototypical networks (protonets) [1] and model-agnostic meta-learning (MAML) [2], for few-shot image classification on the Omniglot dataset [3], which you also used for Homework 1. You will

1. Implement both algorithms (given starter code).
2. Interpret key metrics of both algorithms.
3. Investigate the effect of the number of classes per task during protonet training.
4. Investigate the effect of different ways of specifying the inner loop learning rate in MAML.
5. Investigate the performance of both algorithms on evaluation tasks that have more support data than training tasks.

## Preliminaries

### Notation

- $x$: Omniglot image
- $y$: class label
- $N$ (way): number of classes in a task
- $K$ (shot): number of support examples per class
- $Q$: number of query examples per class
- $c_n$: prototype of class $n$
- $f_\theta$: neural network parameterized by $\theta$
- $\mathcal{T}_i$: task $i$
- $\mathcal{D}_i^{\text{tr}}$: support data in task $i$
- $\mathcal{D}_i^{\text{ts}}$: query data in task $i$
- $B$: number of tasks in a batch
- $\mathcal{J}(\theta)$: objective function parameterized by $\theta$

### Computation environment

- We expect you to develop your solutions locally (e.g. make sure your model can run for a few training iterations), but to use Azure to run GPU-accelerated training for your results.

### Expectations

- Submit to Gradescope
  1. a `.zip` file containing your modified version of `hw2/starter/`
  2. a `.pdf` report containing your responses
- Deliverables (text responses, figures, tables, etc.) are signposted with **Submit**.
- You are welcome to use TensorBoard screenshots for your plots. Ensure that individual lines are labeled, e.g. using a custom legend, or by text in the figure caption.
- Figures and tables should be numbered and referred to.

## Part 1: Prototypical Networks (Protonets) [1]

### Algorithm Overview

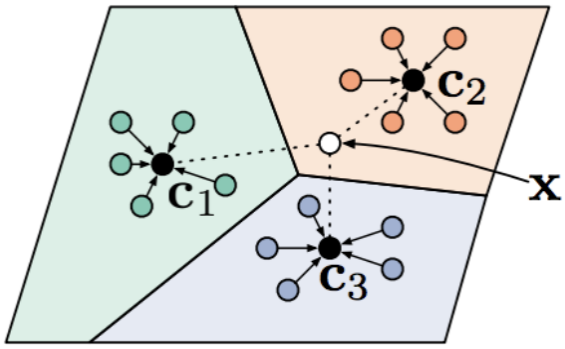


Figure 1: Prototypical networks in a nutshell. In a 3-way 5-shot classification task, the class prototypes $c_1, c_2, c_3$ are computed from each class's support features (colored circles). The prototypes define decision boundaries based on Euclidean distance. A query example $x$ is determined to be class 2 since its features (white circle) lie within that class's decision region.

As discussed in lecture, the basic idea of protonets is to learn a mapping $f_\theta(\cdot)$ from images to features such that images of the same class are close to each other in feature space. Central to this is the notion of a *prototype*

$$c_n = \frac{1}{K} \sum_{(x,y) \in \mathcal{D}_i^{\text{tr}} : y = n} f_\theta(x), \tag{1}$$

i.e. for task $i$, the prototype of the $n$-th class $c_n$ is defined as the mean of the $K$ feature vectors of that class's support images. To classify some image $x$, we compute a measure of distance $d$ between $f_\theta(x)$ and each of the prototypes. We will use the squared Euclidean distance:

$$d(f_\theta(x), c_n) = \|f_\theta(x) - c_n\|_2^2. \tag{2}$$

We interpret the negative squared distances as logits, or unnormalized log-probabilities, of $x$ belonging to each class. To obtain the proper probabilities, we apply the softmax operation:

$$p_\theta(y = n|x) = \frac{\exp(-d(f_\theta(x), c_n))}{\sum_{n'=1}^{N} \exp(-d(f_\theta(x), c_{n'}))}. \tag{3}$$

Because the softmax operation preserves ordering, the class whose prototype is closest to $f_\theta(x)$ is naturally interpreted as the most likely class for $x$. To train the model to generalize, we compute prototypes using support data, but minimize the negative log likelihood of

the query data

$$\mathcal{J}(\theta) = \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T}),(\mathcal{D}^{\text{tr}},\mathcal{D}^{\text{ts}})\sim\mathcal{T}}\left[\frac{1}{NQ}\sum_{(x^{\text{ts}},y^{\text{ts}})\in\mathcal{D}^{\text{ts}}} -\log p_\theta(y = y^{\text{ts}}|x^{\text{ts}})\right]. \tag{4}$$

Notice that this is equivalent to using a cross-entropy loss.

We optimize $\theta$ using Adam [4], an off-the-shelf gradient-based optimization algorithm. As is standard for stochastic gradient methods, we approximate the objective (4) with Monte Carlo estimation on minibatches of tasks. For one minibatch of size $B$, we have

$$\mathcal{J}(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\left[\frac{1}{NQ}\sum_{(x^{\text{ts}},y^{\text{ts}})\in\mathcal{D}_i^{\text{ts}}} -\log p_\theta(y = y^{\text{ts}}|x^{\text{ts}})\right]. \tag{5}$$

**Problems**

1. In the `protonet.py` file, complete the implementation of the `ProtoNet._step` method, which computes (5) along with accuracy metrics. Pay attention to the inline comments and docstrings.
2. Assess your implementation on 5-way 5-shot Omniglot. To do so, run

   ```
   python protonet.py
   ```

   with the appropriate command line arguments. These arguments have defaults specified in the file. To specify a non-default value for an argument, use the following syntax:

   ```
   python protonet.py --argument1 value1 --argument2 value2
   ```

   Use 15 query examples per class per task. Depending on how much memory your GPU has, you may need to reduce the batch size. You should not need to adjust the learning rate from its default of $0.001$.

   As the model trains, model checkpoints and TensorBoard logs are periodically saved to a `log_dir`. The default `log_dir` is formatted from the arguments, but this can be overriden. You can visualize logged metrics by running

   ```
   tensorboard --logdir logs/
   ```

   and navigating to the displayed URL in a browser. If you are running on a remote computer with server capabilities, use the `--bind_all` option to expose the web app to the network. Alternatively, consult the Azure guide for an example of how to tunnel/port-forward via SSH.

To resume training a model starting from a checkpoint at {some_dir}/state{some_step}.pt, run

```
python protonet.py --log_dir some_dir --checkpoint_step some_step
```

If a run ended because it reached num_train_iterations, you may need to increase this parameter.

To assess a trained model on the test split, run

```
python protonet.py --test
```

appropriately specifying log_dir and checkpoint_step.

**Submit** a plot of the validation query accuracy over the course of training.
**Hint**: you should obtain a query accuracy on the validation split of at least $97\%$.

3. 4 accuracy metrics are logged. For the above run, examine these in detail to reason about what the algorithm is doing. **Submit** responses to the following questions:

   (a) What do you notice about the train_support and val_support accuracy? What does this suggest about where the protonet places support examples of the same class in feature space?

   (b) Compare train_query and val_query. Is the model generalizing to new tasks? If not, is it overfitting or underfitting?

4. Train on 5-way 1-shot tasks. **Submit** a table comparing test performance on 5-way 1-shot tasks, with 95% confidence intervals, between training on 5-way 1-shot vs. 5-way 5-shot tasks. **Submit** responses to the following questions: How did you choose which checkpoint to use for testing for each model? What do you notice about the test performance? If there is a difference, what could explain this difference?

## Part 2: Model-Agnostic Meta-Learning (MAML) [2]

### Algorithm Overview

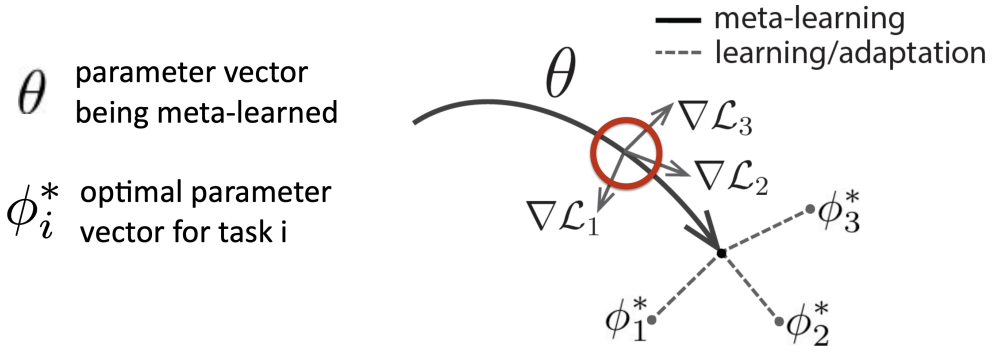


Figure 2: MAML in a nutshell. MAML tries to find an initial parameter vector $\theta$ that can be quickly adapted via task gradients to (near-)optimal parameter vectors.

As discussed in lecture, the basic idea of MAML is to (meta-)learn parameters $\theta$ that can be quickly adapted via gradient descent to a given task. To keep notation clean, define the loss $\mathcal{L}$ of a model with parameters $\phi$ on the data $\mathcal{D}_i$ of a task $\mathcal{T}_i$ as

$$\mathcal{L}(\phi, \mathcal{D}_i) = \frac{1}{|\mathcal{D}_i|} \sum_{(x^j, y^j) \in \mathcal{D}} -\log p_\phi(y = y^j | x^j) \tag{6}$$

Adaptation is often called the *inner loop*. For a task $\mathcal{T}_i$ and $L$ inner loop steps, adaptation (that follows the gradient descent algorithm) looks like the following:

$$\begin{aligned} \phi^1 &= \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}) \\ \phi^2 &= \phi^1 - \alpha \nabla_{\phi^1} \mathcal{L}(\phi^1, \mathcal{D}_i^{\text{tr}}) \\ &\vdots \\ \phi^L &= \phi^{L-1} - \alpha \nabla_{\phi^{L-1}} \mathcal{L}(\phi^{L-1}, \mathcal{D}_i^{\text{tr}}) \end{aligned} \tag{7}$$

Notice that only the support data is used to adapt the parameters to $\phi^L$. (In lecture, you saw $\phi^L$ denoted as $\phi_i$.) To optimize $\theta$ in the *outer loop*, we use the same loss function applied on the adapted parameters and the query data.

For this homework, we will further consider a variant of MAML [5] that proposes to additionally learn the inner loop learning rates $\alpha$. Instead of a single scalar inner learning rate for all parameters, there is a separate scalar inner learning rate for each parameter group (convolutional kernel, weight matrix, or bias vector). Adaptation remains the same as in vanilla MAML except with appropriately broadcasted multiplication between the inner

loop learning rates and the gradients with respect to each parameter group. The objective is

$$\mathcal{J}(\theta, \alpha) = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T}), (\mathcal{D}^{\text{tr}}, \mathcal{D}^{\text{ts}}) \sim \mathcal{T}} \left[ \mathcal{L}(\phi^L, \mathcal{D}^{\text{ts}}) \right] \tag{8}$$

Like before, we will use minibatches to approximate $(8)$ and use the Adam optimizer.

**Problems**

1. In the `maml.py` file, complete the implementation of the `MAML._inner_loop` and `MAML._outer_step` methods. The former computes the task-adapted network parameters (and accuracy metrics), and the latter computes the MAML objective (and more metrics). Pay attention to the inline comments and docstrings.
   **Hint**: the simplest way to implement `_inner_loop` involves using `autograd.grad`.
   **Hint**: read the documentation for the `create_graph` argument of `autograd.grad`.

2. Assess your implementation of vanilla MAML on 5-way 1-shot Omniglot. Comments from the previous part regarding arguments, checkpoints, TensorBoard, resuming training, and testing all apply. Use 1 inner loop step with a **fixed** inner learning rate of 0.4. Use 15 query examples per class per task. You should not need to adjust the outer learning rate from its default of $0.001$. Note that MAML generally needs more time to train than protonets.
   **Submit** a plot of the `val` post-adaptation query accuracy over the course of training.
   **Hint**: you should obtain a query accuracy on the validation split of at least $93\%$.

3. 6 accuracy metrics are logged. Examine these in detail to reason about what MAML is doing. **Submit** responses to the following questions:

   (a) What do you notice about the `train_pre_adapt_support` and `val_pre_adapt_support` accuracies? Why does this make sense given the task sampling process?

   (b) What can you infer about the model from comparing the `train_pre_adapt_support` and `train_post_adapt_support` accuracies? And the corresponding `val` accuracies?

   (c) What about by comparing the `train_post_adapt_support` and `train_post_adapt_query` accuracies? And the corresponding `val` accuracies?

4. Try MAML with a fixed inner learning rate of $0.04$. **Submit** a plot of the validation post-adaptation query accuracy over the course of training with for the two inner learning rates $(0.04, 0.4)$. **Submit** a response to the following question: Why would these different values affect training?

5. Try MAML with learning the inner learning rates. Initialize the inner learning rates with $0.4$. **Submit** a plot of the validation post-adaptation query accuracy over the course of training for learning and not learning the inner learning rates, initialized at $0.4$. **Submit** a response to the following question: What is the effect of learning the inner learning rates?

## Part 3: More Support Data at Test Time

In practice, we usually have more than 1 support example at test time. Hence, one interesting comparison is to train both algorithms with $5$-way $1$-shot tasks (as you've already done) but assess them using more shots.

1. Use the protonet trained with 5-way 1-shot tasks, and the MAML trained with **learned** inner learning rates initialized at 0.4. Try $K = 1, 2, 4, 6, 8, 10$ at test time. **Submit** a plot of the test accuracies for the two models over these values of $K$ with the 95% confidence intervals as error bars or shaded regions. **Submit** a response to the following question: How well is each model able to use additional data in a task without being explicitly trained to do so?

## A Note

You may wonder why the performance of these implementations don't reach the numbers reported in the original papers. One major reason is that the original papers used a different version of Omniglot few-shot classification, in which multiples of $90^\circ$ rotations are applied to each image to obtain 4 times the total number of images and characters. Another reason is that these implementations are designed to be pedagogical and therefore straightforward to implement from equations and pseudocode as well as trainable with minimal hyperparameter tuning. This sacrifices some performance.

## References


[1] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

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

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.