CS 330 Autumn 2019/2020 Homework 2
Model-Agnostic Meta-Learning and Prototypical Networks
Due Wednesday October 23, 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 we will experiment with two meta-learning algorithms, model-agnostic meta-learning (MAML) [1] and prototypical networks [2], for few-shot classification. You will
(1) Implement and train MAML. You also need to experiment with different choices of the key hyperparameter of the MAML algorithm, the inner gradient step size, and implement a variant of MAML that learns the inner step size automatically.
(2) Implement and train prototypical networks.

Similar to Homework 1, we will work with the Omniglot dataset [3], which contains 1623 different characters from 50 different languages. For each character there are 20 28x28 images. We are interested in training models for K-shot, N-way classification.

Submission: To submit your homework, submit one pdf report and one zip file to GradeScope, where the report will contain answers to the deliverables listed below and the zip file contains your code (`run_maml.py, models/maml.py, run_ProtoNet.py, models/ProtoNet.py, load_data.py`) with the filled in solutions.

Code Overview: The code consists of five main files:

- `load_data.py`: code to load batches of images and labels, which does not need to be edited.

- `run_maml.py`: training and testing script for running MAML.

- `models/maml.py`: network architecture and computation graph construction of MAML.

- `run_ProtoNet.py`: training and testing script for running prototypical networks.

- `models/ProtoNet.py`: network architecture/loss of prototypical networks.

There is also the `omniglot_resized` folder with the data. *You should not modify this folder.*

Dependencies: We expect code in Python 3.5+ with `Pillow, scipy, numpy, tensorflow` installed. If you use `Anaconda`, run `conda env create -f environment.yml` to build a virtual environment containing packages required for the homework.

$$\underset{\text{[test-time]}}{\overset{\textbf{Fine-tuning}}{}} \quad \phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\mathrm{tr}})$$

pre-trained parameters

training data for new task

$$\textbf{Meta-learning} \quad \min_\theta \sum_{\textbf{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\mathrm{tr}}), \mathcal{D}_i^{\mathrm{ts}})$$

Figure 1: For each task i, MAML computes inner gradient updates on training datapoints $\mathcal{D}_i^{\mathrm{tr}}$ and evaluates the loss on test datapoints $\mathcal{D}_i^{\mathrm{ts}}$. Averaging over all tasks, the outer loop loss function is optimized w.r.t. the original model parameter θ to learn an initialization that can quickly adapt to new tasks during meta-test time.

Note: Even though convolutional networks will be used for MAML and Prototypical Networks, the code should be able to be run with CPU.

Problem 1: Model-Agnostic Meta-Learning (MAML) [1]

We will first attempt few-shot classification with MAML. As introduced in the class, during meta-training phase, MAML operates in two loops, an inner loop and an outer loop. In the inner loop, MAML computes gradient updates using examples from each task and calculates the loss on test examples from the same task using the updated model parameters. In the outer loop, MAML aggregates the per-task post-update losses and performs a meta-gradient update on the original model parameters. At meta-test time, MAML computes new model parameters based a few examples from an unseen class and uses the new model parameters to predict the label of a test example from the same unseen class. The main idea of MAML is shown in Figure 1. The data processing will be done in the same way as in Homework 1. In the `run_maml.py` and `models/maml.py` files:

1. Fill in the data processing parts in the `meta_train` and `meta_test` functions in `run_maml.py`, which should call the data generator provided in `load_data.py` to generate a batch of images and their corresponding labels. You should partition the batch into two parts, `inputa, labela` and `inputb, labelb`, where `inputa, labela` are used to compute gradient updates in the inner loop and `inputb, labelb` are used to get the task losses after the gradient update. *Hint: You need to fill in data processing parts for meta-training, meta-validation and meta-test, but they should be fairly similar.*

2. Fill in the function called `task_inner_loop` in the `models/maml.py` file which takes inputs `inputa, labela, inputb, labelb` and computes the inner loop updates in the main MAML algorithm. Feel free to use `self.loss_func` to compute the losses. Your main work should be calculating the gradient updates of each weight variable stored in the `weights` dictionary and passing the updated weights to `forward_conv` in
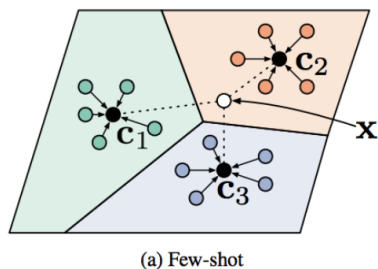
$$\mathbf{c}_k = \frac{1}{|\mathcal{D}_i^{\mathrm{tr}}|} \sum_{(x,y)\in\mathcal{D}_i^{\mathrm{tr}}} f_\theta(x)$$

$$p_\theta(y = k|x) = \frac{\exp(-d(f_\theta(x), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\theta(x), \mathbf{c}_{k'}))}$$

(a) Few-shot

Figure 2: Prototypical networks compute the prototypes of all tasks using training datapoints $\mathcal{D}_i^{\mathrm{tr}}$. Then by comparing the query example x to each of the prototype, the model makes prediction based on the softmax function over the distance between the embedding of the query and all prototypes.

order to get the new prediction. Note that you need to do multiple gradient updates as `num_inner_updates` could be greater than 1. *Hint: You may need* `tf.gradients` *to calculate the inner gradients instead of using* `tf.train.Optimizer`

3. Run `python run_maml.py --n_way=5 --k_shot=1 --inner_update_lr=0.4 --num_inner_updates=1`. Also try with `inner_update_lr` being 0.04 and 4.0. For each configuration, submit a plot of the validation accuracy over iterations as well as the number of the average test accuracy. Can you briefly explain why different values of `inner_update_lr` would affect meta-training?

4. Tuning `inner_update_lr` could turn out to be tricky when running MAML for different datasets. A variant of MAML [4] proposes to automatically learn the `inner_update_lr`. Try to learn separate `inner_update_lr` per `num_inner_update` per weight variable. Specifically, for each inner gradient update, for each weight variable stored in the `weights` dictionary, initialize one `inner_update_lr` variable and learn it using back-propagation. Plot the meta-validation accuracy over meta-training iterations and state how it compares to the MAML with fixed `inner_update_lr`.

Problem 2: Prototypical Networks [2]

Now we will try a non-parametric meta-learning algorithm, prototypical networks. As discussed in lecture, the basic idea of prototypical networks resembles weighted nearest neighbors. It computes the prototype of each class using a set of support examples and then calculates the distance between the query example and all the prototypes. The query example is classified based on the label of the prototype it's closest to. See Figure 2 for an overview.

1. Similar to Problem 1, fill in the data processing parts in `run_ProtoNet.py`, which should also call the data generator provided in `load_data.py`. You should partition

the sampled batch into `support`, i.e. the per-task training data, and `query`, i.e. the per-task test datapoints. The `support` will be used to calculate the prototype of each class and `query` will be used to compute the distance to each prototype. You also need to get `labels` of the `query` examples in order to compute the cross-entropy loss for training the whole model.

2. Fill in the function called `ProtoLoss` in the `models/ProtoNet.py` file which takes the embeddings of the support and query examples as well as the one-hot label encodings of the queries and computes the loss and prediction accuracy based on the main algorithm of the prototypical networks.

3. Run `python run_ProtoNet.py ./omniglot_resized/` and plot the validation accuracy over iterations. Report the average test accuracy along with its standard deviation.

Problem 3: Comparison and Analysis

After implementing both meta-learning algorithms, we would like to compare them a bit. In practice, we usually have limited amount of meta-training data but relatively more meta-test datapoints. Hence one interesting comparison would be meta-training both algorithms with 5-way 1-shot regime but meta-testing them using 4-shot data. Specifically, run the following to evaluate Prototypical Networks:

```
python run_ProtoNet.py ./omniglot_resized/ --n-way=5 --k-shot=1 --n-query=5
--n-meta-test-way=5 --k-meta-test-shot=4 --n-meta-test-query=4
```

For evaluating MAML, first do meta-training by running:
```
python run_maml.py --n_way=5 --k_shot=1 --inner_update_lr=0.4
--num_inner_updates=1
```

Then restore the weights to get meta-test performance by running:
```
python run_maml.py --n_way=5 --k_shot=4 --inner_update_lr=0.4
--num_inner_updates=1 --meta_train=False --meta_test_set=True --meta_train_k_shot=1
```

Try $K = 4, 6, 8, 10$ at **meta-test** time. Compare the meta-test performance between MAML and Prototypical Networks by plotting the meta-test accuracies over different choices of K.

References

[1] 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.

[2] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 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] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.