CS330 Autumn 2020 Homework 2
Model-Agnostic Meta-Learning and Prototypical Networks

Due Friday October 16, 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 the inner loop of MAML. You also need to experiment with different choices of 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]. We are interested in training models for K-shot, N-way classification.

Submission: To submit your homework, submit one PDF report to Gradescope containing written answers/plots to the questions below and the link to the colab notebook. The PDF should also include your name and any students you talked to or collaborated with.
This year, we are releasing assignments as Google Colab notebooks. The Colab notebook for this assignment can be found here:
https://colab.research.google.com/drive/1zbt2A74kM10HvcAEgEy3fGgRSNyHZQKj?usp=sharing
As noted in the notebook instructions, you will need to save a copy of the notebook to your Google Drive in order to make edits and complete your submission. **When you submit your PDF responses on Gradescope, please include a clearly-visible link to your Colab notebook so we may examine your code. Any written responses or plots to the questions below must appear in your PDF submission.**

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

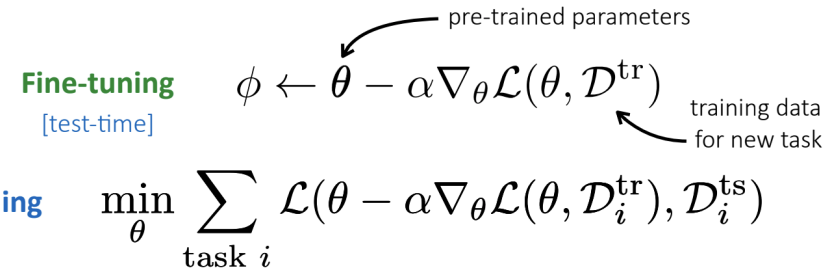


Figure 1: For each task i, MAML computes inner gradient updates on training datapoints $\mathcal{D}_i^{\text{tr}}$ and evaluates the loss on test datapoints $\mathcal{D}_i^{\text{ts}}$. Averaging over all tasks, the outer loop loss function is optimized w.r.t. the original model parameter θ to learn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

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.

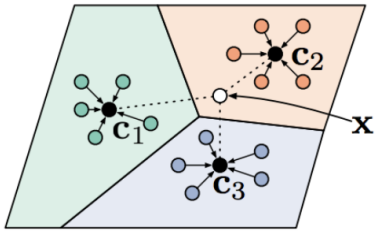
1. Fill in the data processing parts in the `meta_train_fn` and `meta_test_fn` functions, which should call the data generator provided to generate a batch of images and their corresponding labels.

   You should partition the batch into two parts, `input_tr, label_tr` and `input_ts, label_ts`, where `input_tr, label_tr` are used to compute gradient updates in the inner loop and `input_ts, label_ts` are used to get the task losses after the gradient update. *Hint: You need to fill in data processing parts in these functions, but they should be extremely similar.*

2. Fill in the function called `task_inner_loop` which takes inputs `input_tr, label_tr, input_ts, label_ts` 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 in the `weights` dictionary and passing the updated weights to the model 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 will need* `tf.GradientTape` *to calculate the inner gradients.*

3. Run a code cell with `run_maml(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?

   Hint: You should be able to achieve at least 50% *few-shot accuracy.*



(a) Few-shot

$$\mathbf{c}_k = \frac{1}{|\mathcal{D}_i^{\text{tr}}|} \sum_{(x,y)\in\mathcal{D}_i^{\text{tr}}} f_\theta(x)$$

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

Figure 2: Prototypical networks compute the prototypes of all tasks using training datapoints $\mathcal{D}_i^{\text{tr}}$. 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.

4. Tuning `inner_update_lr` can be tricky. 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 backpropagation. Run `run_maml(n_way=5, k_shot=1, inner_update_lr=0.4, num_inner_updates=1, learn_inner_update_lr=True)`. Also try with `inner_update_lr` being 0.04 and 4.0.

   Submit a plot of 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 nearest neighbors to class prototypes. It computes the prototype of each class using a set of support examples and then calculates the distance between the query example and each 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 the function `run_protonet`, which should also call the data generator provided. 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` 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 `run_protonet('./omniglot_resized/')` and **submit** a plot of 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. 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:

```
run_protonet('./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:

```
run_maml(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:

```
run_maml(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.