XCS330 Problem Set 3

Due Sunday, May 7 at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs330-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit only the `src/submission.py` file in the online student portal. For further details, see Writing Code and Running the Autograder below.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be foudn at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.

- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Hidden Tests

All hidden tests rely on files that are not provided to students. Therefore, the tests can only be run remotely. When a hidden test like `1a-1-hidden` is executed locally, it will produce the following result:

```
----- START 1a-1-hidden:  Test multiple instances of the same word in a sentence.
----- END 1a-1-hidden [took 0:00:00.011989 (max allowed 1 seconds), ???/3 points] (hidden test ungraded)
```

Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:

```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:

```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```


Remote Execution

Basic and hidden tests are treated the same by the remote autograder. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.

1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

Just like in the local autograder, this error caused the test to fail.

Just like in the local autograder, start your debugging in line 23 of grader.py.

1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

This error caused the test to fail.

Start your debugging in line 31 of grader.py.

Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.

1a-0-basic) Basic test case. (2.0/2.0)

1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

1 Prototypical Networks (Protonets) [1]

Goals: In this assignment, you will experiment with a meta-learning algorithms - prototypical networks (protonets) [1] for few-shot image classification on the Omniglot dataset [2]. You will:

1. Implement the protonets algorithms (given starter code).
2. Interpret key metrics of the algorithm.
3. Investigate the effect of task composition during protonet training on evaluation.
4. Investigate the performance of the algorithm on meta-test tasks that have more support data than training tasks do.

Expectations

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

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_θ: neural network parameterized by θ
- $\mathcal{T}_i$: 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 θ

For this assignment, please submit the following files to gradescope to recieve points for coding questions:

- `submission.pdf`
- `src/submission/__init__.py`
- `src/submission/maml.py`
- `src/submission/protonet.py`

Protonets Algorithm Overview

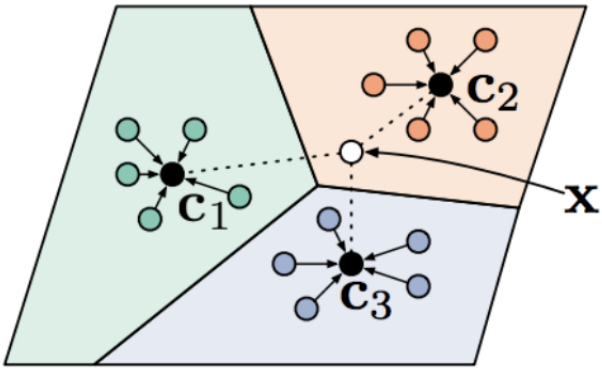


Figure 1: Prototypical networks in a nutshell. In a 3-way 5-shot classification task, the class prototypes c_1, c_2, c_3 are computed from each class's support features (colored circles). The prototypes define decision boundaries based on Euclidean distance. A query example x is determined to be class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

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

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

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

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

We optimize θ using Adam [3], 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 with B tasks, we have

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

1.a **[6 points (Written)] No Suffle Required**

We have provided you with `omniglot.py`, which contains code for task construction and data loading.

Recall that for training black-box meta-learners in the previous homework we needed to shuffle the query examples in each task. This is not necessary for training protonets. Explain why.

1.b **[8 points (Coding)] Implement Step**

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.

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 want to adjust the batch size. Do not 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/
```

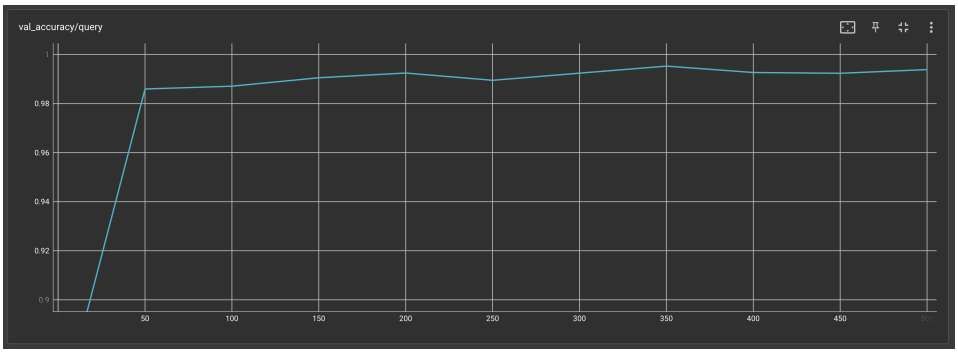
and navigating to the displayed URL in a browser. If you are running on a remote computer with server capabilities, use the `--bind_all` option to expose the web app to the network. Alternatively, consult the Azure guide for an example of how to tunnel/port-forward via SSH.

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.

Your plot of the validation query accuracy over the course of training should look like the figure below.



Hint: you should obtain a query accuracy on the validation split of at least 99%.

1.c **[6 points (Written)] Evaluating Accuracy Metrics**

4 accuracy metrics are logged. For the above run, examine these in detail to reason about what the algorithm is doing.

(i) Is the model placing support examples of the same class close together in feature space or not? Support your answer by referring to specific accuracy metrics.

(ii) Is the model generalizing to new tasks? If not, is it overfitting or underfitting? Support your answer by referring to specific accuracy metrics.

1.d **[6 points (Written)] 5-way 1-shot Comparison** We will now compare different settings at training time. Train on 5-way 1-shot tasks with 15 query examples per task.

(i) Compare your two runs (5-way 1-shot training and 5-way 5-shot training) by assessing test performance on 5-way 1-shot tasks. To assess a trained model on test tasks, run

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

appropriately specifying `log_dir` and `checkpoint_step`. Submit a table of your results with 95% confidence intervals.

(ii) How did you choose which checkpoint to use for testing for each model?

(iii) Is there a significant difference in the test performance on 5-way 1-shot tasks? Explain this by referring to the protonets algorithm.

2 Model-Agnostic Meta-Learning (MAML) [4]

A Note: You may wonder why the performance of these implementations don't match 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° 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. Finally, with our use of batch statistics for batch normalization during test (see code), we are technically operating in the *transductive* few-shot learning setting.

MAML Algorithm Overview

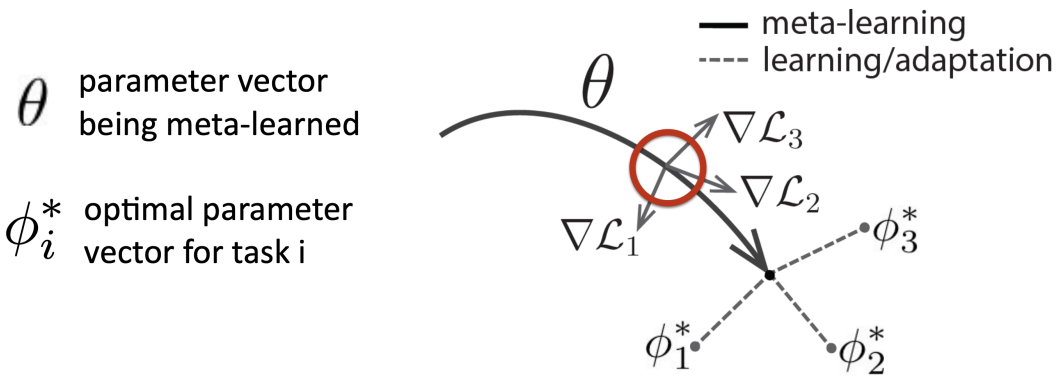


Figure 2: MAML in a nutshell. MAML tries to find an initial parameter vector θ that can be quickly adapted via task gradients to task-specific optimal parameter vectors.

As discussed in lecture, the basic idea of MAML is to meta-learn parameters θ 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 ϕ on the data $\mathcal{D}_i$ of a task $\mathcal{T}_i$ as

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

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

$$\begin{aligned} \phi^1 &= \phi^0 - \alpha \nabla_{\phi^0} \mathcal{L}(\phi^0, \mathcal{D}_i^{\text{tr}}) \\ \phi^2 &= \phi^1 - \alpha \nabla_{\phi^1} \mathcal{L}(\phi^1, \mathcal{D}_i^{\text{tr}}) \\ &\vdots \\ \phi^L &= \phi^{L-1} - \alpha \nabla_{\phi^{L-1}} \mathcal{L}(\phi^{L-1}, \mathcal{D}_i^{\text{tr}}) \end{aligned} \tag{7}$$

where we have defined $\theta = \phi^0$.

Notice that only the support data is used to adapt the parameters to ϕ^L. (In lecture, you saw ϕ^L denoted as ϕ_i.) To optimize θ in the *outer loop*, we use the same loss function (6) applied on the adapted parameters and the query data:

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

For this homework, we will further consider a variant of MAML [5] that proposes to additionally learn the inner loop learning rates α. Instead of a single scalar inner learning rate for all parameters, there is a separate scalar inner learning rate for each parameter group (e.g. 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 full MAML objective is

$$\mathcal{J}(\theta, \alpha) = \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T}), (\mathcal{D}_i^{\text{tr}}, \mathcal{D}_i^{\text{ts}}) \sim \mathcal{T}_i} \left[\mathcal{L}(\phi^L, \mathcal{D}_i^{\text{ts}}) \right] \tag{9}$$

Like before, we will use minibatches to approximate (9) and use the Adam optimizer.

2.a **[12 points (Coding)] Implement Inner Loop**

In the `maml.py` file, complete the implementation of the `MAML._inner_loop` which computes the task-adapted network parameters (and accuracy metrics). Pay attention to the inline comments and docstrings.
Hint: the simplest way to implement `_inner_loop` involves using `autograd.grad`.

2.b **[12 points (Coding)] Implement Outer Loop**

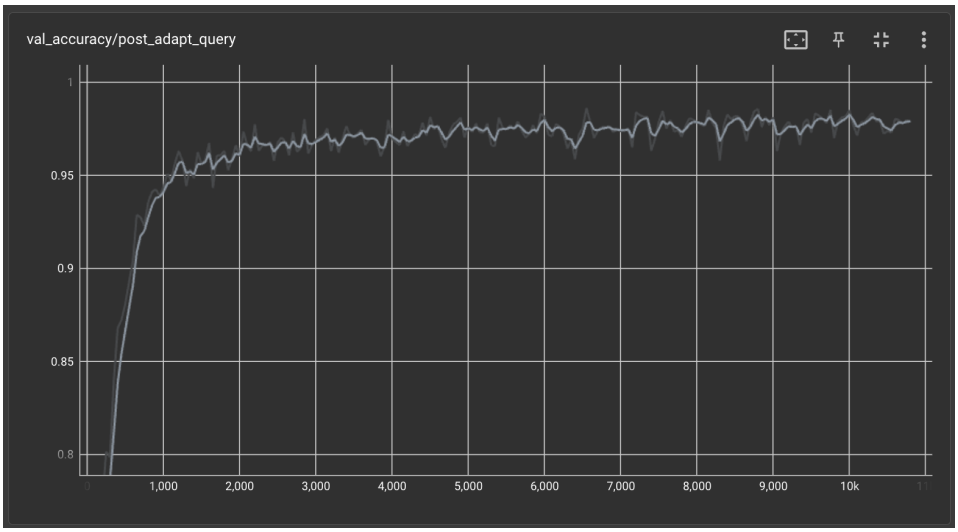
In the `maml.py` file, complete the implementation of the `MAML._outer_step` which computes the MAML objective (and more metrics). Pay attention to the inline comments and docstrings.
Hint: to understand how to use the Boolean `train` argument of `MAML._outer_step`, read the documentation for the `create_graph` argument of `autograd.grad`.

Assess your implementation of vanilla MAML on 5-way 1-shot Omniglot.

You can run the experiment by executing: `python maml.py`

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. Do not adjust the outer learning rate from its default of 0.001.

Your plot of the validation post-adaptation query accuracy over the course of training should look like this.



Hint: you should obtain a query accuracy on the validation split of at least 96%.

2.c **[5 points (Written)] Accuracy Metrics**

6 accuracy metrics are logged. Examine these in detail to reason about what MAML is doing.

(i) State and explain the behavior of the `train_pre_adapt_support` and `val_pre_adapt_support` accuracies. Your answer should explicitly refer to the task sampling process.
Hint: consult the `omniglot.py` file.

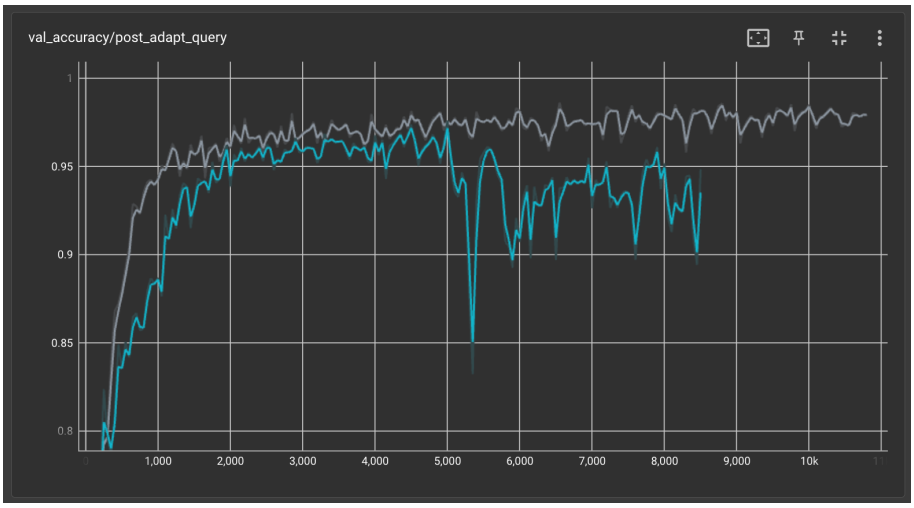
(ii) Compare the `train_pre_adapt_support` and `train_post_adapt_support` accuracies. What does this comparison tell you about the model? Repeat for the corresponding `val` accuracies.

(iii) Compare the `train_post_adapt_support` and `train_post_adapt_query` accuracies. What does this comparison tell you about the model? Repeat for the corresponding `val` accuracies.

2.d **[5 points (Written)] Experiments - Adjusting Learning Rate and Inner Loop Steps**

(i) Try MAML with the same hyperparameters as above except for a fixed inner learning rate of 0.04 by running `python maml.py --inner_lr 0.04`

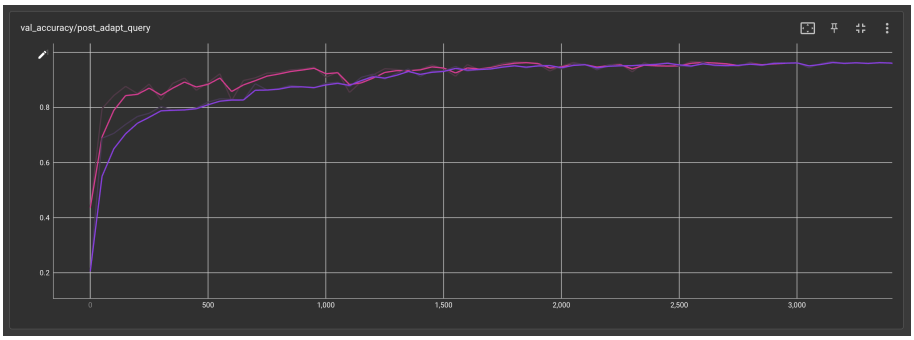
Your plot of the validation post-adaptation query accuracy over the course of training with the two inner learning rates (0.04, 0.4) should look as follows.



What is the effect of lowering the inner learning rate on (outer-loop) optimization and generalization?

(ii) Try MAML with a fixed inner learning rate of 0.04 for 5 inner loop steps by `python maml.py --inner_lr 0.04 ----num_inner_steps 5`

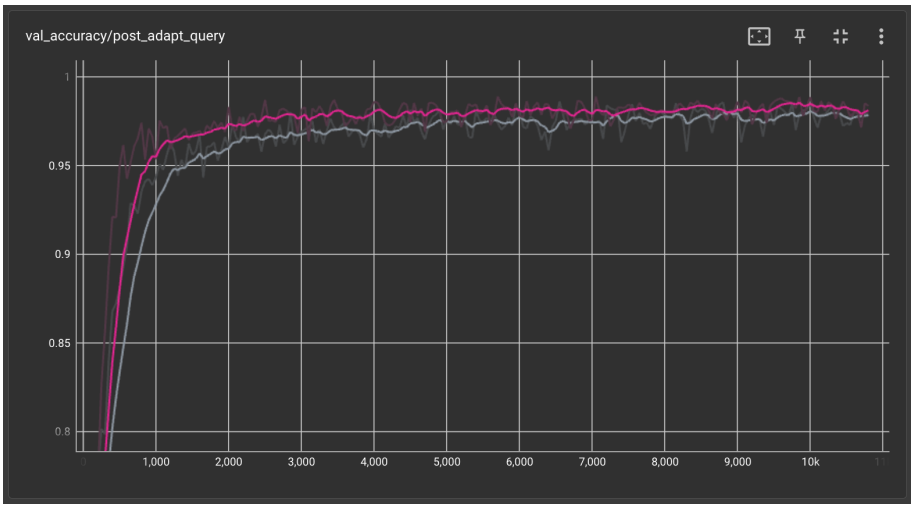
Your plot of the validation post-adaptation query accuracy over the course of training with the two number of inner loop steps (1, 5) with inner learning rate 0.04 should look as follows.



What is the effect of increasing the number of inner loop steps on (outer-loop) optimization and generalization?

(iii) Try MAML with learning the inner learning rates by running `python maml.py --learn_inner_lrs`. Initialize the inner learning rates with 0.4.

Your 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 should look as follows.



What is the effect of learning the inner learning rates on (outer-loop) optimization and generalization?

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

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

[4] 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.

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

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

1.c.i
1.c.ii

1.d.i
1.d.ii
1.d.iii

2.c.i
2.c.ii
2.c.iii

2.d.i
2.d.ii
2.d.iii