⚠ 此測驗已重新評分；您的新分數反映了受影響的 20 個問題。

此測驗已鎖定，學生無法作答。

HW15

截止時間 7月9日 23:59　　**分數** 10　　**問題** 25
接受繳交時間 6月18日 14:20 - 7月9日 23:59 21 天　　**時間限制** 無
可作答次數 無限制

說明

請大家作答前注意以下幾個細節，請妥善檢查以免影響自身權益

1. 如同 release 作業時提及的，本次作業是以**最新一次**為最終作答並評分，也就是說**每當你送出該次作答**（COOL 顯示出一個假分數）**後之前的作答記錄就不算數了，請各位注意**
2. NTU COOL 系統會自動在 **今晚 23:59:00 強制交卷**，若同學屆時沒有作答完畢會以未完成的形式被送出，可能會因為上一條提及的規定影響成績，請同學**不要在最後時刻才作答**。（建議最晚提前約半小時就交卷以保留時間確認）
3. NTU COOL 目前顯示的正確答案**並非各題真正的答案**，請同學依照自己的判斷作答，真正的分數最後會在結束時公布，之所以採取此策略僅是為了方便同學確認自己選擇的選項。（因此嘗試利用無限次作答找出系統的「標準答案」是不會反應到最終分數上的，也請同學不要這麼做，這種以漏洞不正當取得正確答案的行為本身是有理由被判為作弊的。）

結束之後 HW15 成績與正確答案將在 明天中午 12:00 以前全部出爐，請同學在 7/11（日）23:59 以前確認成績，預計將在 7/12（一） 確定本學期所有成績與等第。

此測驗鎖定於 7月9日 23:59。

作答記錄

	作答記錄	時間	分數	已重新評分
最新的	作答紀錄 1	111 分鐘	得分：2.13；總分：10	得分：10；總分：10

⚠ 正確答案已隱藏。

此作答紀錄的分數： 得分：**10**；總分：10
已提交7月4日 16:38

此作答持續 111 分鐘。

Basic Concepts

問題 1

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

What does meta learning want to learn?

- ○ data distribution
- ○ mapping function for a task
- ○ mapping function for many tasks
- ◉ learning how to learn

問題 2

原始分數： 0.13 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Which of the following can be meta learned? (select 5 of them)

- ☑ model initialization
- ☑ model structure

- [x] model optimizer
- [x] the whole learning algorithm
- [ ] model name
- [x] feature extractor
- [ ] implementation toolkit

問題 3

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Which is NOT the reason we need meta learning?

- There maybe too many tasks to learn.
- We hope to customize the model into various scenarios.
- (•) It is a powerful tip that does not require training and therefore should be applied everywhere.
- The amount of data may not be sufficient to apply the general learning algorithm.

問題 4

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Which of the following uses meta learning?

- Reptile [Nichol et al., 2018] (https://arxiv.org/abs/1803.02999)
- DARTS [Liu et al., 2018] (https://arxiv.org/abs/1806.09055)
- Siamese Network [Koch et al., 2015] (https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf)
- All of above

問題 5

0.4 / 0.4 分

In the meta-training phase, we adapt the parameters of the model on the support set in the inner-loop. Which loss is used to update the parameters in the outer-loop?

- The loss tested on the query set of the training tasks with the adapted parameters.
- The loss tested on the support set of the training tasks with the adapted parameters.

The loss tested on the query set of the testing tasks with the adapted parameters.

The loss tested on the support set of the testing tasks with the adapted parameters.

問題 6 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

What does it mean that a classification problem is n way k shot?

In a task, there are k classes and n training data for each class.

In a task, there are n classes and k training data for each class.

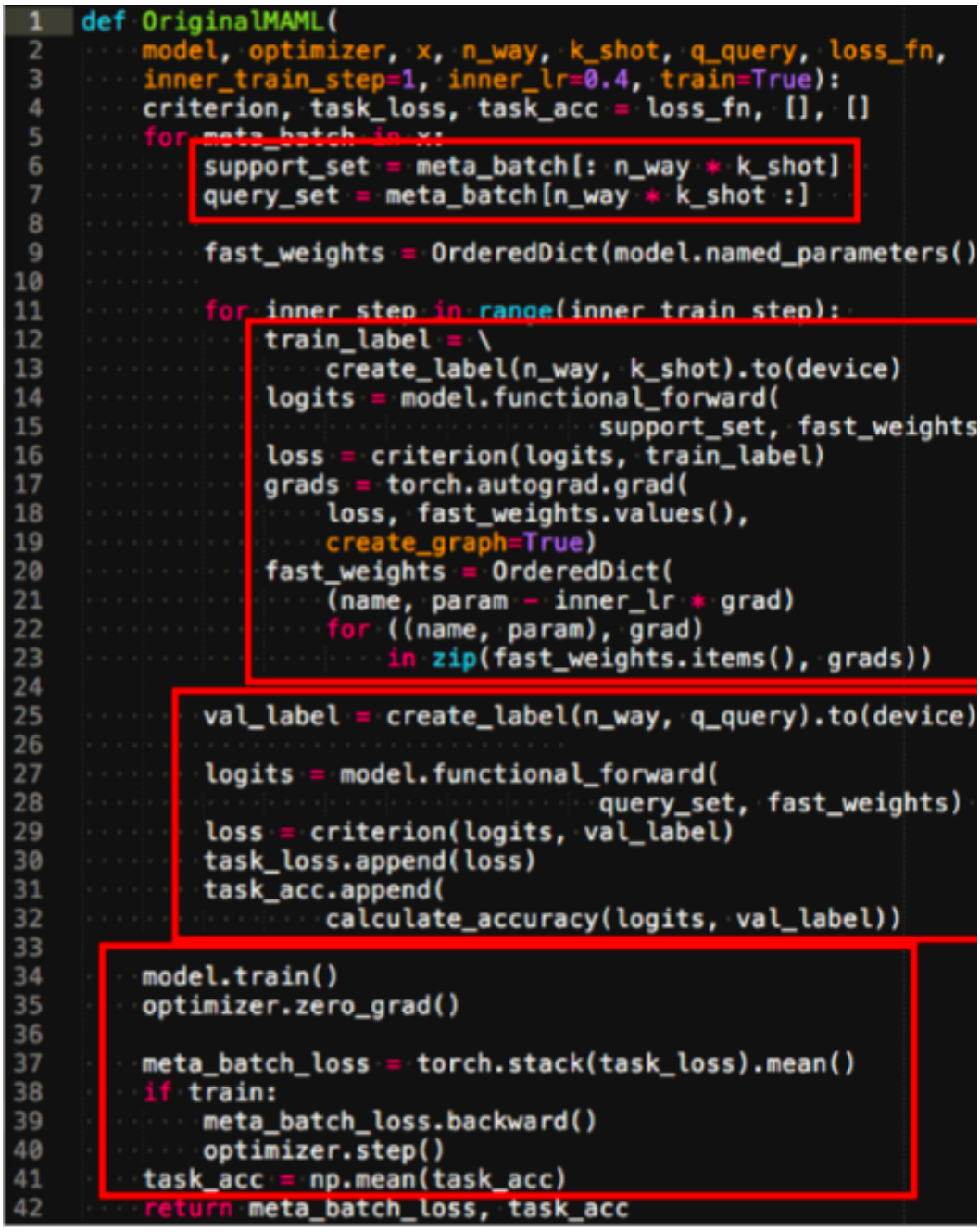
In a dataset, there are n tasks and k classes for each task.

In a dataset, there are k tasks and n classes for each task.

Implementation

問題 7 0.4 / 0.4 分

In the block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL of the `OriginalMAML()` function in the sample code, which part of the code performs the updating of the parameters in the outer-loop? (Refer to the image)



```
1  def OriginalMAML(
2      model, optimizer, x, n_way, k_shot, q_query, loss_fn,
3      inner_train_step=1, inner_lr=0.4, train=True):
4      criterion, task_loss, task_acc = loss_fn, [], []
5      for meta_batch in x:
6          support_set = meta_batch[: n_way * k_shot]
7          query_set = meta_batch[n_way * k_shot :]
8
9          fast_weights = OrderedDict(model.named_parameters())
10
11         for inner_step in range(inner_train_step):
12             train_label = \
13                 create_label(n_way, k_shot).to(device)
14             logits = model.functional_forward(
15                                 support_set, fast_weights
16             loss = criterion(logits, train_label)
17             grads = torch.autograd.grad(
18                 loss, fast_weights.values(),
19                 create_graph=True)
20             fast_weights = OrderedDict(
21                 (name, param - inner_lr * grad)
22                 for ((name, param), grad)
23                         in zip(fast_weights.items(), grads))
24
25         val_label = create_label(n_way, q_query).to(device)
26
27         logits = model.functional_forward(
28                                 query_set, fast_weights)
29         loss = criterion(logits, val_label)
30         task_loss.append(loss)
31         task_acc.append(
32                 calculate_accuracy(logits, val_label))
33
34     model.train()
35     optimizer.zero_grad()
36
37     meta_batch_loss = torch.stack(task_loss).mean()
38     if train:
39         meta_batch_loss.backward()
40         optimizer.step()
41     task_acc = np.mean(task_acc)
42     return meta_batch_loss, task_acc
```

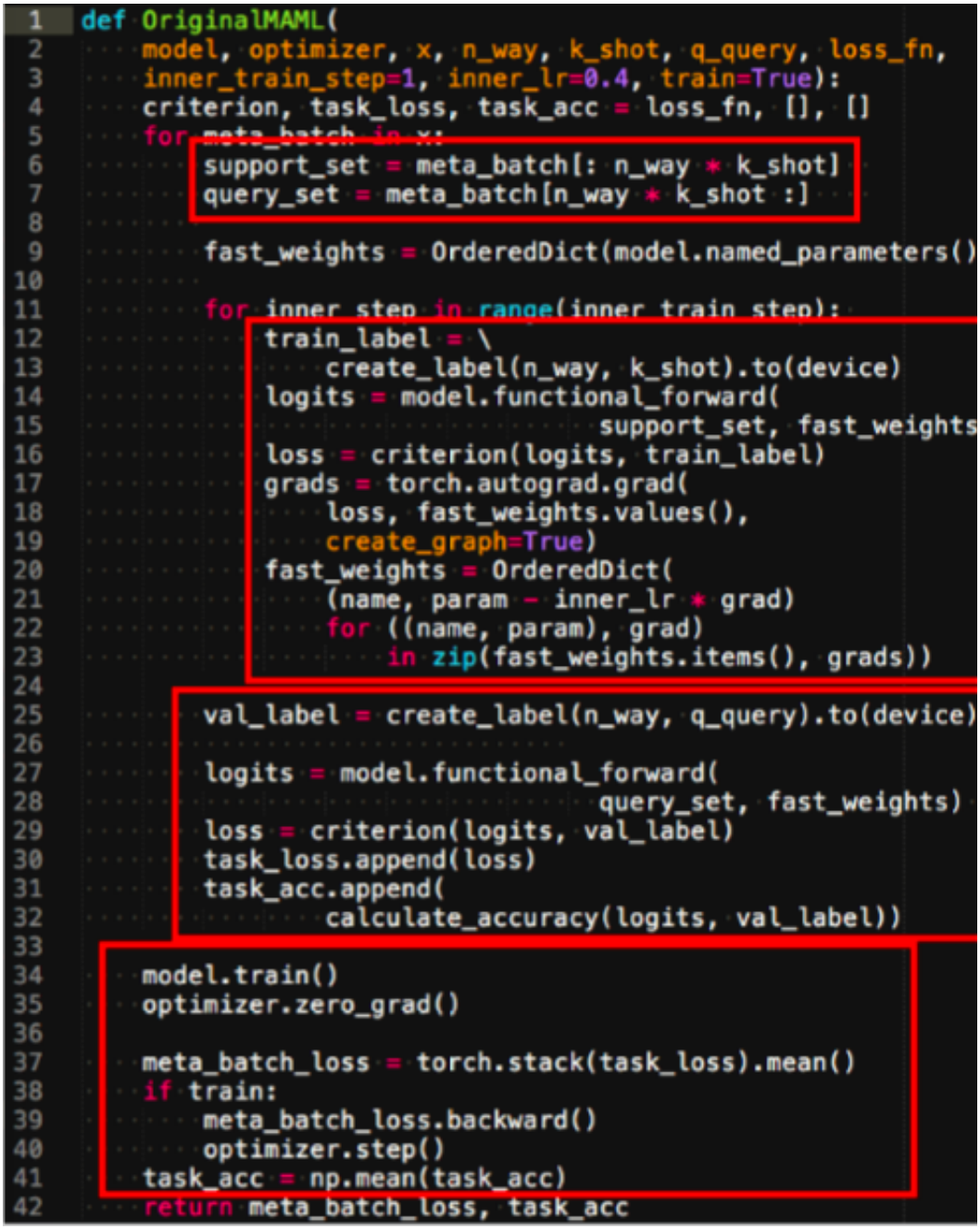
- Line 6 ~ 7
- Line 12 ~ 23
- Line 24 ~ 32

- Line 34 ~ 41

問題 8 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

In the block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL of the `OriginalMAML()` function in the sample code, which part of the code performs the updating of the parameters in the inner-loop? (Refer to the image)



```
1  def OriginalMAML(
2      model, optimizer, x, n_way, k_shot, q_query, loss_fn,
3      inner_train_step=1, inner_lr=0.4, train=True):
4      criterion, task_loss, task_acc = loss_fn, [], []
5      for meta_batch in x:
6          support_set = meta_batch[: n_way * k_shot]
7          query_set = meta_batch[n_way * k_shot :]
8
9          fast_weights = OrderedDict(model.named_parameters())
10
11         for inner_step in range(inner_train_step):
12             train_label = \
13                 create_label(n_way, k_shot).to(device)
14             logits = model.functional_forward(
15                                 support_set, fast_weights
16             loss = criterion(logits, train_label)
17             grads = torch.autograd.grad(
18                 loss, fast_weights.values(),
19                 create_graph=True)
20             fast_weights = OrderedDict(
21                 (name, param - inner_lr * grad)
22                 for ((name, param), grad)
23                     in zip(fast_weights.items(), grads))
24
25         val_label = create_label(n_way, q_query).to(device)
26
27         logits = model.functional_forward(
28                                 query_set, fast_weights)
29         loss = criterion(logits, val_label)
30         task_loss.append(loss)
31         task_acc.append(
32                 calculate_accuracy(logits, val_label))
33
34     model.train()
35     optimizer.zero_grad()
36
37     meta_batch_loss = torch.stack(task_loss).mean()
38     if train:
39         meta_batch_loss.backward()
40         optimizer.step()
41     task_acc = np.mean(task_acc)
42     return meta_batch_loss, task_acc
```

- Line 6 ~ 7
- Line 12 ~ 23

Line 24 ~ 32

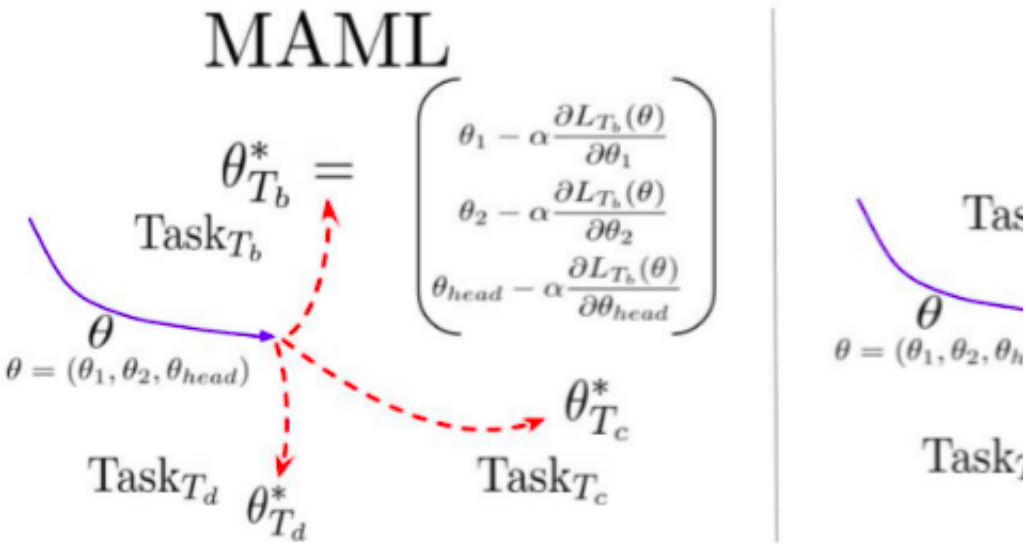
Line 34 ~ 41

問題 9

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

In the ANIL (Almost No Inner Loop) algorithm, what parameters are updated in the inner-loop?



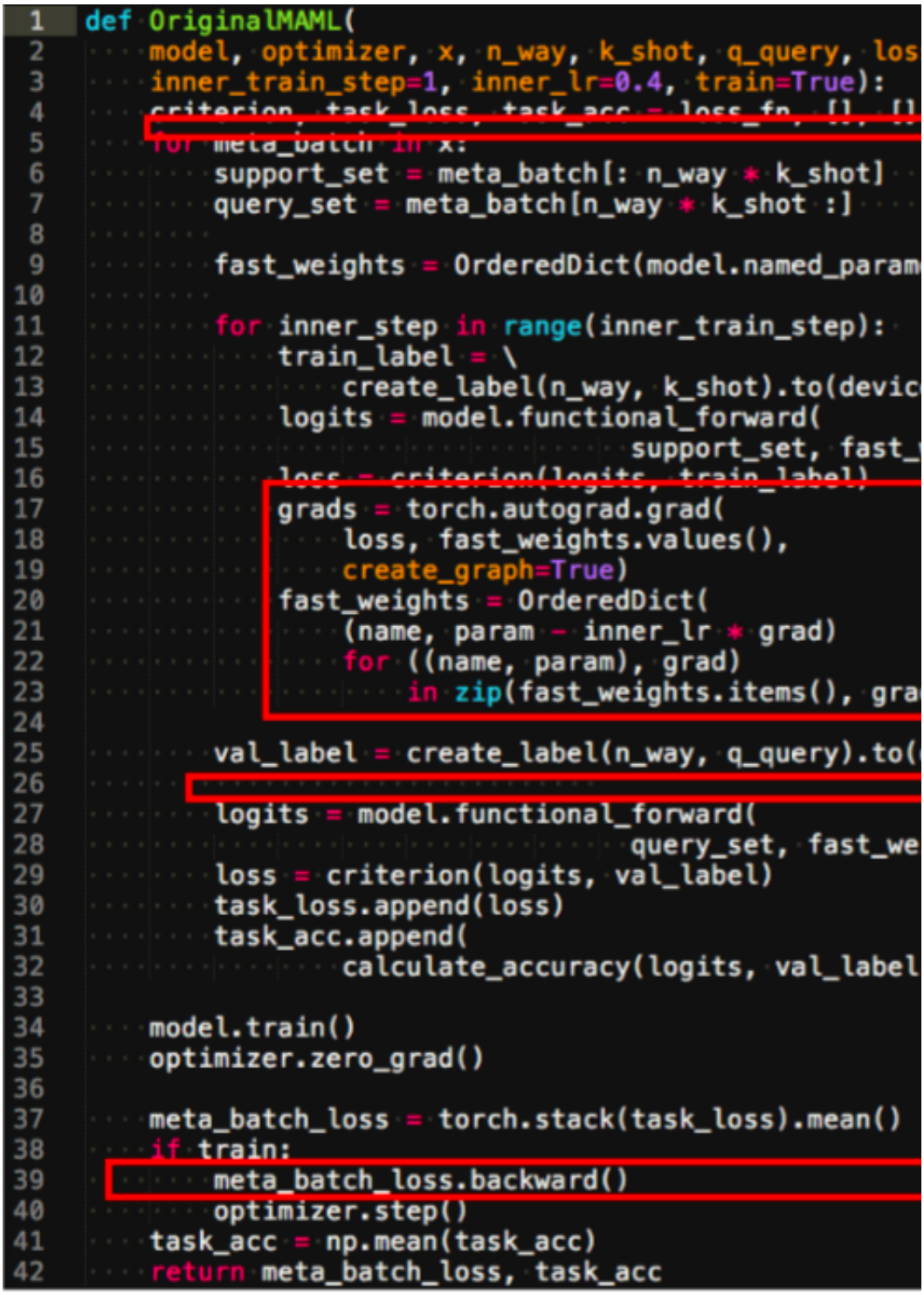
- all the model parameters
- the task specific parameters
- all the model parameters except for the task specific parameters
- none of the above

問題 10

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

In the [block of code](https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL) of the `OriginalMAML()` function in the sample code, which part of the code the MAML algorithm should we modify to achieve the first-order approximation version(FOMAML)? (Refer to the image)



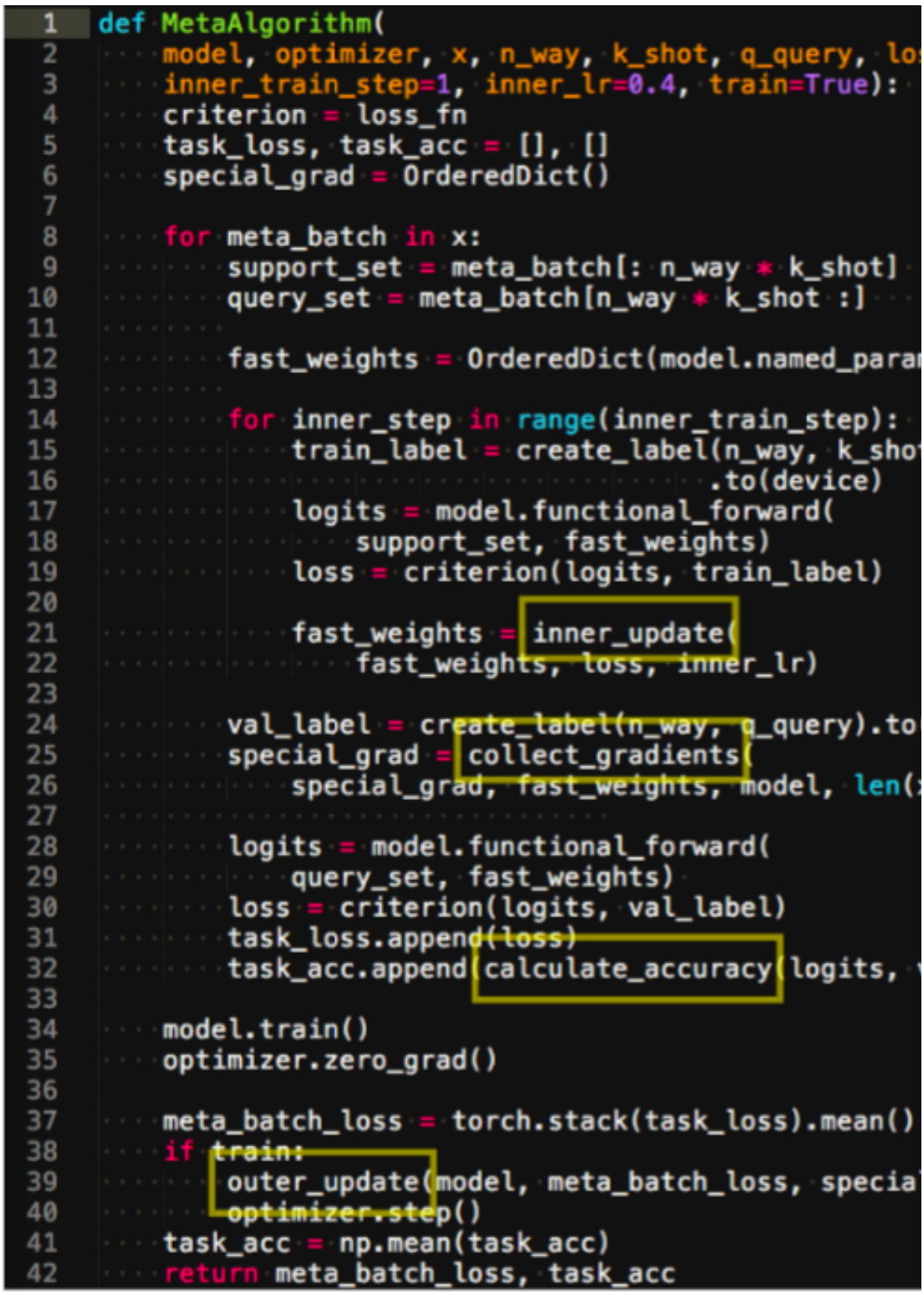
```
def OriginalMAML(
    model, optimizer, x, n_way, k_shot, q_query, los
    inner_train_step=1, inner_lr=0.4, train=True):
    criterion, task_loss, task_acc = loss_fn, [], []
    for meta_batch in x:
        support_set = meta_batch[: n_way * k_shot]
        query_set = meta_batch[n_way * k_shot :]

        fast_weights = OrderedDict(model.named_param

        for inner_step in range(inner_train_step):
            train_label = \
                create_label(n_way, k_shot).to(devic
            logits = model.functional_forward(
                                support_set, fast_
            loss = criterion(logits, train_label)
            grads = torch.autograd.grad(
                loss, fast_weights.values(),
                create_graph=True)
            fast_weights = OrderedDict(
                (name, param - inner_lr * grad)
                for ((name, param), grad)
                    in zip(fast_weights.items(), gra

        val_label = create_label(n_way, q_query).to(

        logits = model.functional_forward(
                                query_set, fast_we
        loss = criterion(logits, val_label)
        task_loss.append(loss)
        task_acc.append(
                calculate_accuracy(logits, val_label

    model.train()
    optimizer.zero_grad()

    meta_batch_loss = torch.stack(task_loss).mean()
    if train:
        meta_batch_loss.backward()
        optimizer.step()
    task_acc = np.mean(task_acc)
    return meta_batch_loss, task_acc
```

- Between line 4 ~ 5
- Lines 17 ~ 23
- Between line 25 ~ 27
- Line 39

問題 11

0.4 / 0.4 分

The block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNI of the `OriginalMAML()` function in the sample code can be abstracted into the `MetaAlgorithm()` function in `MetaAlgorithmGenerator()` .
Which function in the code the MAML algorithm should we modify to achieve the first-order approximation version(FOMAML)? (Refer to the image)



- inner_update
- collect_gradients
- calculate_accuracy
- outer_update

問題 12

0.4 / 0.4 分

The block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL of the `OriginalMAML()` function in the sample code can be abstracted into the `MetaAlgorithm()` function in `MetaAlgorithmGenerator()` .
Which function can we use to achieve FOMAML?

```
def MetaAlgorithm(
    model, optimizer, x, n_way, k_shot, q_query, lo
    inner_train_step=1, inner_lr=0.4, train=True):
    criterion = loss_fn
    task_loss, task_acc = [], []
    special_grad = OrderedDict()

    for meta_batch in x:
        support_set = meta_batch[: n_way * k_shot]
        query_set = meta_batch[n_way * k_shot :]

        fast_weights = OrderedDict(model.named_para

        for inner_step in range(inner_train_step):
            train_label = create_label(n_way, k_sho
                                       .to(device)
            logits = model.functional_forward(
                support_set, fast_weights)
            loss = criterion(logits, train_label)

            fast_weights = inner_update(
                fast_weights, loss, inner_lr)

        val_label = create_label(n_way, q_query).to
        special_grad = collect_gradients(
            special_grad, fast_weights, model, len(

        logits = model.functional_forward(
            query_set, fast_weights)
        loss = criterion(logits, val_label)
        task_loss.append(loss)
        task_acc.append(calculate_accuracy(logits,

    model.train()
    optimizer.zero_grad()

    meta_batch_loss = torch.stack(task_loss).mean()
    if train:
        outer_update(model, meta_batch_loss, specia
        optimizer.step()
    task_acc = np.mean(task_acc)
    return meta_batch_loss, task_acc
```

- inner_update_alt1

- inner_update_alt2
- collect_gradients_alt
- outer_update_alt

問題 13 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

In the [block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL](https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL) of the `OriginalMAML()` function in the sample code, which part of the code the MAML algorithm should we modify to achieve the ANIL algorithm? (Refer to the image)

```
def OriginalMAML(
    model, optimizer, x, n_way, k_shot, q_query, los
    inner_train_step=1, inner_lr=0.4, train=True):
    criterion, task_loss, task_acc = loss_fn, [], []
    for meta_batch in x:
        support_set = meta_batch[: n_way * k_shot]
        query_set = meta_batch[n_way * k_shot :]

        fast_weights = OrderedDict(model.named_param

        for inner_step in range(inner_train_step):
            train_label = \
                create_label(n_way, k_shot).to(devic
            logits = model.functional_forward(
                                support_set, fast_
            loss = criterion(logits, train_label)
            grads = torch.autograd.grad(
                loss, fast_weights.values(),
                create_graph=True)
            fast_weights = OrderedDict(
                (name, param - inner_lr * grad)
                for ((name, param), grad)
                    in zip(fast_weights.items(), gra

        val_label = create_label(n_way, q_query).to(

        logits = model.functional_forward(
                                query_set, fast_we
        loss = criterion(logits, val_label)
        task_loss.append(loss)
        task_acc.append(
                calculate_accuracy(logits, val_label

    model.train()
    optimizer.zero_grad()

    meta_batch_loss = torch.stack(task_loss).mean()
    if train:
        meta_batch_loss.backward()
        optimizer.step()
    task_acc = np.mean(task_acc)
    return meta_batch_loss, task_acc
```

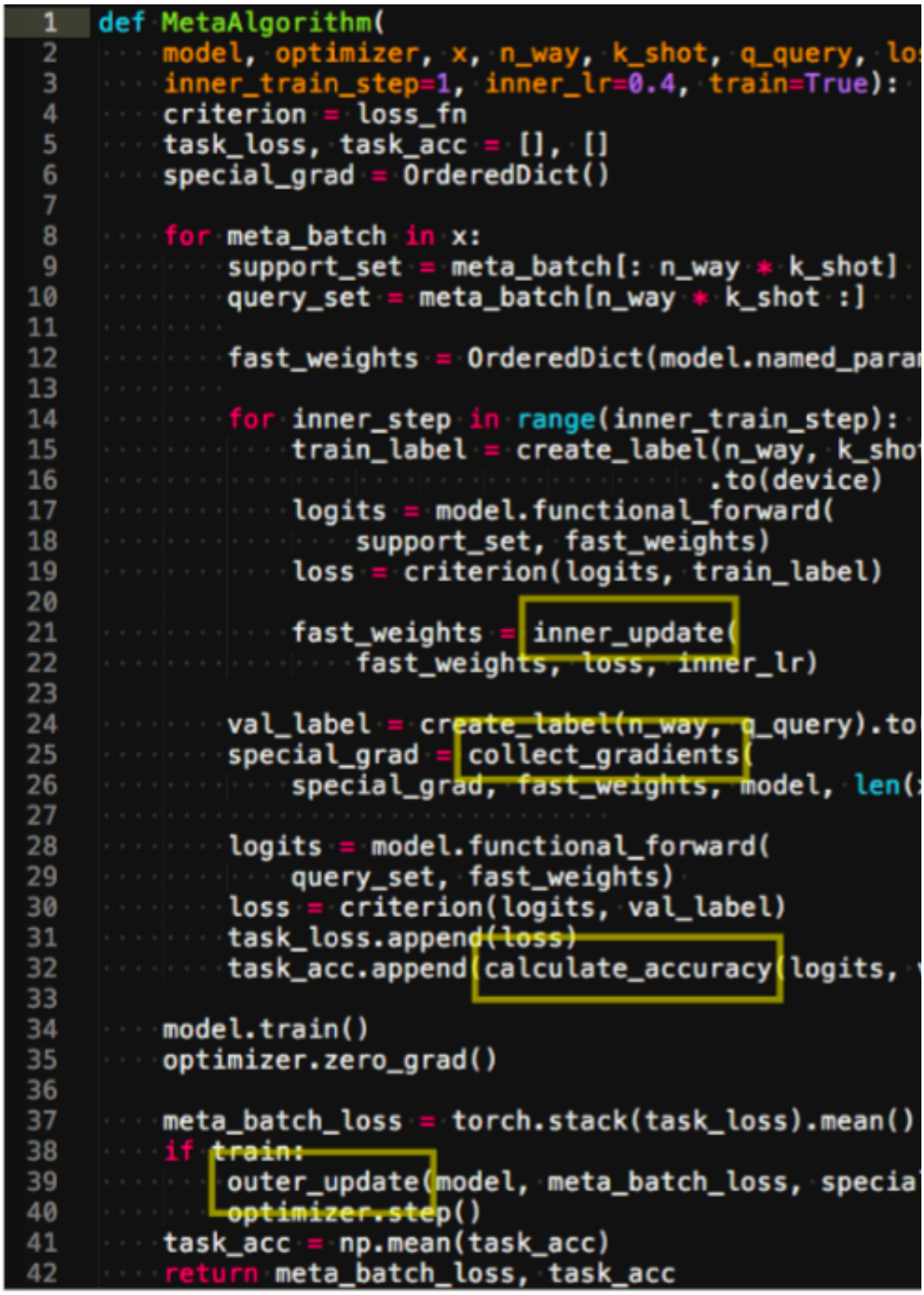
- Between line 4 ~ 5
- Lines 17 ~ 23
- Between line 25 ~ 27
- Line 39

問題 14

0.4 / 0.4 分

The block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNL of the `OriginalMAML()` function in the sample code can be abstracted into the `MetaAlgorithm()` function in `MetaAlgorithmGenerator()` .

Which function in the code the MAML algorithm should we modify to achieve the ANIL algorithm? (Refer to the image)



```
def MetaAlgorithm(
    model, optimizer, x, n_way, k_shot, q_query, lo
    inner_train_step=1, inner_lr=0.4, train=True):
    criterion = loss_fn
    task_loss, task_acc = [], []
    special_grad = OrderedDict()

    for meta_batch in x:
        support_set = meta_batch[: n_way * k_shot]
        query_set = meta_batch[n_way * k_shot :]

        fast_weights = OrderedDict(model.named_para

        for inner_step in range(inner_train_step):
            train_label = create_label(n_way, k_sho
                                       .to(device)
            logits = model.functional_forward(
                support_set, fast_weights)
            loss = criterion(logits, train_label)

            fast_weights = inner_update(
                fast_weights, loss, inner_lr)

        val_label = create_label(n_way, q_query).to
        special_grad = collect_gradients(
            special_grad, fast_weights, model, len(

        logits = model.functional_forward(
            query_set, fast_weights)
        loss = criterion(logits, val_label)
        task_loss.append(loss)
        task_acc.append(calculate_accuracy(logits,

    model.train()
    optimizer.zero_grad()

    meta_batch_loss = torch.stack(task_loss).mean()
    if train:
        outer_update(model, meta_batch_loss, specia
        optimizer.step()
    task_acc = np.mean(task_acc)
    return meta_batch_loss, task_acc
```

- inner_update

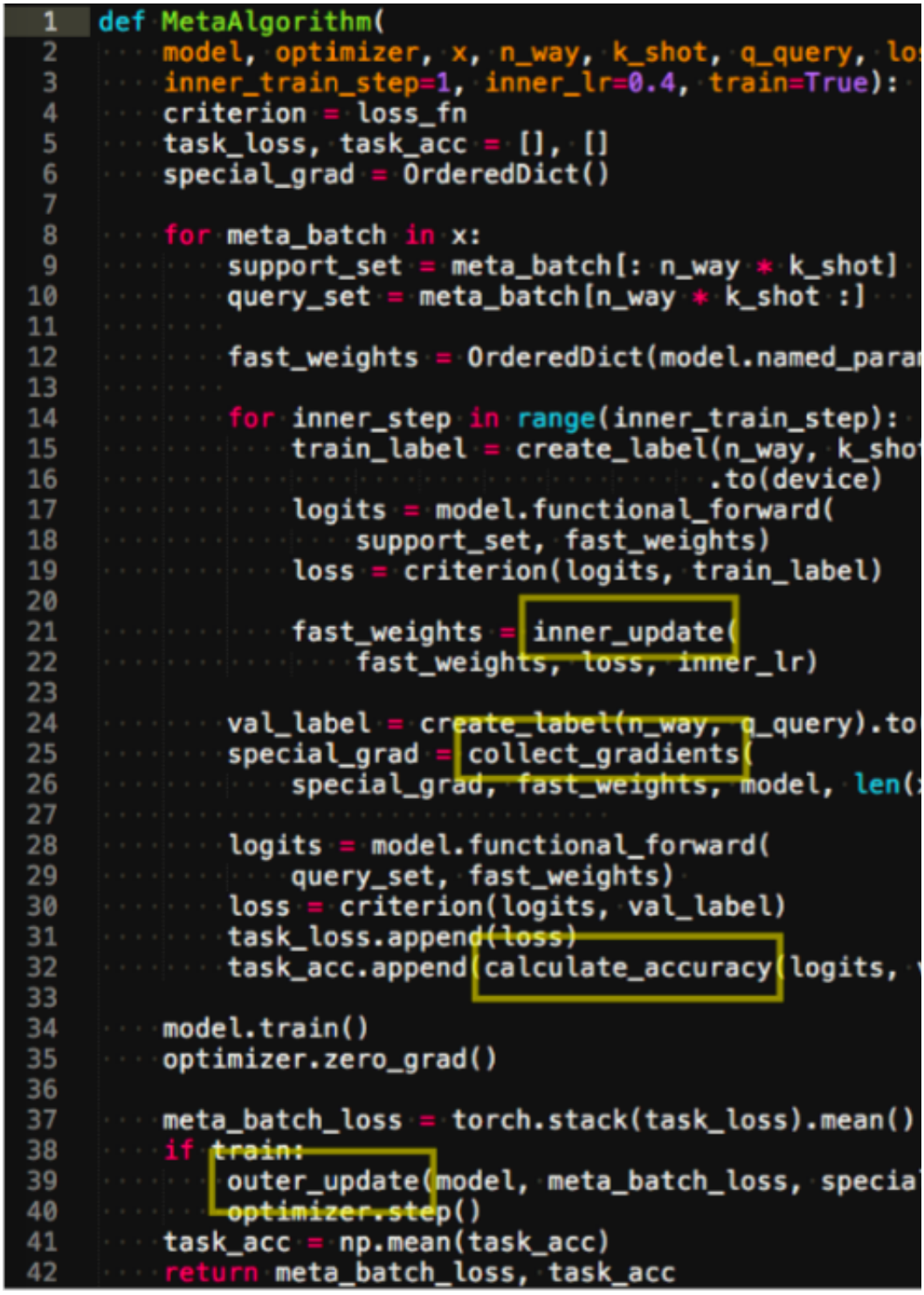
- collect_gradients
- calculate_accuracy
- outer_update

問題 15 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

The block of code (https://colab.research.google.com/drive/1N73PnZmZ3jimFe8023lX2w9yNl of the `OriginalMAML()` function in the sample code can be abstracted into the `MetaAlgorithm()` function in `MetaAlgorithmGenerator()` .

Which function can we use to achieve the ANIL algorithm?



- inner_update_alt1
- inner_update_alt2
- collect_gradients_alt
- outer_update_alt

問題 16

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

During the meta training phase, the learning rate of the inner loop and outer loop must be the same.

- 正確
- 錯誤

問題 17

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

The adaptation steps of the inner loop during the meta training phase have to be the same as the adaptation steps during the meta testing phase.

正確

錯誤

Advanced Tips

There are various modifications to MAML, for example MAML++. The following questions will be based on this paper (https://arxiv.org/pdf/1810.09502.pdf) .

問題 18 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Which one is not the issue of MAML?

Different neural network architectures may lead to wide variances of results.

It requires to update parameters during inferencing. This sometimes leads to huge computational costs compared to those non-adaptive methods.

It usually overfits when adapting the model with the support set.

Sometimes it requires difficult hyperparameter searches to stabilize training.

問題 19

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

MAML++ is an improved variant of the MAML framework that offers the flexibility of MAML along with many improvements. Which of the following is not the improvement of MAML++?

- robust and stable training
- automatic learning of the inner loop hyperparameters
- improved computational efficiency in the inference phase, but not the training phase
- improves generalization performance

問題 20

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

The paper of MAML++ points out several problems of the original MAML. Which of the following is/are correct? (select 3 choices)

- [x] It may be very unstable during training.

The first order derivative computation requires a lot more time compared to the second order derivative computation.

The way that batch normalization is used may affect the generalization performance of MAML.

In every inner loop step, the learning rate for adaptation varies.

In every outer loop step, the learning rate for adaptation is fixed.

問題 21

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Batch Normalization is a key role in batch processing and calculation. Which of the following is/are correct? (select 4 choices)

Only using the statistics of a current batch for batch normalization may affect the generalization performance of MAML.

Applying accumulating running statistics in batch normalization results to batch normalization being less effective, since the biases learned have to accommodate for a variety of different means and standard deviations instead of a single mean and standard deviation.

Batch normalization using accumulated running statistics will eventually converge to some global mean and standard deviation.

Using accumulating running statistics instead of batch statistics, may greatly increase convergence speed, stability and generalization performance as the normalized features will result in smoother optimization landscape.

☑ Batch normalization in MAML has a problem that batch normalization biases are not updated in the inner-loop; instead, the same biases are used throughout all iterations of base-models.

部分

問題 22

原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Section 4 is about a stable, automated and improved MAML training method. Which of the following is/are correct? (Select 2 choices)

☐ For a meta-batch, MAML updates the model parameters in the outer-loop after it has finished all the inner-loop steps. But with multi-Step loss optimization, it minimizes the target set loss computed by the base-network after every step towards a support set task.

☑ Learning a set of biases per-step within the inner-loop update process may help fix the shared (across step) batch normalization bias problem of the original MAML training method.

☐ Sharing running batch statistics across all update steps of the inner-loop is better than collecting statistics in a per-step regime because the latter leads to optimization issues and potentially slow down or altogether halt optimization.

☑

Learning a learning rate and direction for each layer in the network is better than learning a learning rate and gradient direction for each parameter in the base-network since the latter causes increased number of parameters and increased computational overhead.

Those chose at least 2 choices without choosing "Sharing running batch statistics across all update steps of the inner-loop is better than collecting statistics in a per-step regime because the latter leads to optimization issues and potentially slow down or altogether halt optimization." get full points.

Applications

Speech separation is a long existing problem, also known as the cocktail party problem(For more information if you want, please refer to these two videos,
ss1 (https://youtu.be/tovg5ZxNglo)

(https://youtu.be/tovg5ZxNglo)
,
ss2 (https://youtu.be/G0O1A7IONSY)

(https://youtu.be/G0O1A7IONSY)
).

The following questions will be based on this paper (https://www.researchgate.net/publication/346089726_One_Shot_Learning
.

問題 23 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

In section 3.1, how is a meta-task constructed?

Mix two utterances spoken by the same speaker to form mixture speech. Take 9 mixtures and sample a support set and a query set from them.

Mix two utterances spoken by two different speakers to form mixture speech. Take 9 mixtures and sample a support set and a query set from them.

Mix two utterances spoken by the same speaker to form mixture speech. Moreover, also mix two utterances spoken by two different speakers to form mixture speech. Take 9 mixtures of these two kinds and sample a support set and a query set from them.

Mix two utterances spoken by the same speaker to form mixture speech. Sample some mixtures of this kind for the support set. Mix two utterances spoken by two different speakers to form mixture speech. Sample some mixtures of this kind for the query set.

Speech recognition is an important task and can also trained with meta-learning. (For more information related to speech recognition if you want, please refer to these videos,
asr1 **(https://youtu.be/AIKu43goh-8)**

(https://youtu.be/AIKu43goh-8)

,
asr2 **(https://youtu.be/BdUeBa6NbXA)**

(https://youtu.be/BdUeBa6NbXA)

,
asr3 (https://youtu.be/CGuLuBaLIeI)

(https://youtu.be/CGuLuBaLIeI)

,
asr4 (https://youtu.be/XWTGY_PNABo)

(https://youtu.be/XWTGY_PNABo)

,
asr5 (https://youtu.be/5SSVra6IJY4)

(https://youtu.be/5SSVra6IJY4)

,
asr6 (https://youtu.be/L519dCHUCog)

(https://youtu.be/L519dCHUCog)

,
asr7 (https://youtu.be/dymfkWtVUdo)

(https://youtu.be/dymfkWtVUdo)
)

The following questions will be based on this paper.

The following questions will be based on this paper (https://arxiv.org/pdf/1910.12094.pdf) . There is also a short presentation for this paper
Meta-ASR (https://youtu.be/goav0eXKPwg)

(https://youtu.be/goav0eXKPwg)
.

Feel free to take a look.

問題 24 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

The idea of MAML is to learn initialization parameters from a set of tasks. In the setting of meta learning for low-resource ASR mentioned in section 2.2, which of the following is/are correct? (select 2 choices)

- [x] Different tasks have different language utterances.
- [ ] The initialization parameters obtained by MAML should only adapt to one language well. This, helps the adapted model to achieve high performance on a specific language.
- [ ] In one meta-task, the support set and query set can have different languages of spoken utterances.
- [x]

MultiASR optimizes the model on all source languages directly, without considering how learning happens on the unseen language.

Neural machine translation is a well-known task in the NLP field. Given an input sentence, a model translates the sentence into another language. The following question is based on this paper MetaNMT (https://arxiv.org/abs/1808.08437) .

問題 25 原始分數： 0 / 0.4 分 重新評分的分數： 0.4 / 0.4 分

ⓘ 此問題已經被重新評分。

Which of the following statement is most likely not true?

- The machine translation task in this paper aims to translate sentences of different languages to English.
- According to the experiments in section 5(vs. Multilingual Transfer Learning), during the meta testing phase, a subset of the source tasks is sampled for fine-tuning the model.
- According to the experiments in section 5(Impact of Validation Tasks), the choice of a validation task affects the final performance.

According to the experiments in section 5(Impact of Source Tasks), the choice of source languages has different implications for different target languages.

測驗分數： 得分：**10**；總分：10