COMP6248 REPRODUCIBILITY CHALLENGE: META-LEARNING WITH WARPED GRADIENT DESCENT

Jack Dymond, Hsuan-Yang Wang & Tom Kelly
Department of Computer Science
University of Southampton
{jd5u19, hw14g11, tgk2g14}@soton.ac.uk

INTRODUCTION

Meta-Learning is often described as *learning to learn*, as such meta-learning often involves training a model on a set of tasks, and testing it on a set of separate but similar tasks. In the field, there are a variety of methods used to accomplish this. However, the leading 2 approaches are either gradient/initialisation based, or *update-based*.

In *Warped Gradient Descent* the authors take a holistic approach to the meta-learning problem, by utilising the strong inductive bias of gradient-based methods, and meta-learn using insights from update-based methods. They consider a method that updates the initialisation of the model, as well as how it makes gradient updates, in a method known as *preconditioning*. To do this, they parameterise preconditioning by defining a subspace, with layers in the network being dedicated to warping the *activation space*. [1]. This parameterises the update behaviour, while also allowing an optimised initialisation to be found in the model. Flennerhag et al. (2019)

This methodology can be applied to many machine learning problems, provided a neural network is being used. The authors demonstrate this in the paper by performing multiple few-shot learning experiments, and a maze navigation experiment using reinforcement learning. In this report, we highlight a few-shot Omniglot experiment, and the Maze navigation experiment.

In the Omniglot Experiment, they meta-train a model on many tasks; in this case, a task signifies the few-shot classification of characters in a given alphabet in the Omniglot dataset. To meta-train, they train the model on an alphabet and test it on an unseen subset of the alphabet. They then back-propagate the testing losses to the initialisation of the model, which includes the gradient guiding warp layers. They evaluate the effectiveness of the model on a separate set of alphabets, which are reserved for evaluation, and never meta-trained on. This avoids fitting to the evaluation tasks. They evaluate in the same manner as in training: They train the meta-learnt model on the alphabet and then test it on a subset of the alphabet. The difference is that the model does not see the alphabet until the evaluation stage.

In the maze navigation experiment, they define a fixed maze environment and assign a randomly selected goal coordinate within the maze. An agent is introduced and instructed to navigate to the coordinate. Upon reaching the goal, the agent is teleported to a random coordinate within the maze. The objective of an agent is to reach the goal coordinate as many times as possible in a fixed number of cycles.

In all experiments, they found a warp-layer enabled implementation would achieve the state of the art results.

WHY THIS PAPER?

We chose this paper due to the simple nature of the experiments. Omniglot is a simple dataset, and training on it typically requires little computational resource to get acceptable results. Hence, we believed replicating the experiment would be achievable on the available computational resources.

[1]The output of each layer in the neural network as a vector, the set of vectors for each layer will form a space, known as the 'activation space'

The Omniglot Experiment

Omniglot is a dataset that is traditionally used to train and evaluate few-shot learning methods. It consists of 50 separate alphabets, henceforth known as tasks, each containing 20 different images of their respective characters. For this experiment, only tasks containing 20 or more characters were used, leaving 46 tasks. The tasks were split into a test set, based on the number of pretraining examples, a validation set, and 10 were held for evaluation.

Experiment Outline

For each meta-training step, 20 of the possible training tasks were chosen, the parameters of the warp layers were frozen, and the parameters learned for the task were used for layers in the network between the warp layers. The images for the selected task were then fed into the network, with the warp layers defining the updates at each layer. This process was performed for 100 different characters within the task and collectively is known as a task-training step. The updates are stored, and the model is updated using backpropagation at the end of the meta-training step. One task-training step was performed for each task in the training set. Using this method, one meta-training step, accumulates a total of 2000 warp-layer gradient updates.

First Attempt

In the experiment outlined in the paper, this was performed 1000 times, with each meta-training step taking on average around 370 seconds on a workstation with an AMD Ryzen 3700X CPU and an Nvidia GTX 1060 GPU. We found that this performance was similar when using one of the clusters on Iridis 5, and performed worse when run on the ECS Yann server, suggesting that much of the workload of the algorithm was based on the CPU. This bottleneck would be explored further, the results of which can be found in the Results section.

Upon first reading and assessing the paper to judge how feasible it would be to recreate the paper, it was not clear what computational requirements were needed for the algorithm, with no indication of the hardware that was used to generate the results. Upon realising that to recreate the results in the paper entirely, with 1000 meta-training steps, each with 20 task-training steps would take around 4.3 days (103 hours) each. This would then need to be carried out for 7 times for different numbers of pretraining tasks, and this, in turn, would need to be repeated in order to find an average.

Our first attempt demonstrates the tremendous amount of computation power and time required to reproduce the Omniglot experiment fully. For practical reasons, we decided to reduce the scope of our experiment and only focus on replicating the Omniglot experiment. It was clear that in order to reproduce the results, we would need to reduce the parameters of the experiment or optimise the code base, prompting an investigation in both.

Bottlenecks in the Code

When looking into the performance of the model, two main bottlenecks were found. We found that one-third of the time for a meta-training step was spent training the model and accumulating the warp gradient updates, while the other two-thirds of the time was spent iterating through the saved updates for the model. This is mainly due to the number of updates at each meta-training step. The algorithm needs to load each update from memory, which is saved as a set of data points. This process was parallelised using a multithreading library. The parallelisation results in a 5% decrease in the time taken for each meta-training step.

The second main bottleneck in the system came from the fact that each task-training step was performed in serial. While an attempt was made to parallelise each task-training step onto multiple GPUs, the overhead of moving data between GPUs resulted in each meta-step taking much longer. Instead of pushing data for one task-training step onto multiple GPUs, one way to overcome this issue may be to execute different task-training steps on multiple GPUs.

Reducing Test Parameters

The default configurations of the experiment provided optimal performance. To achieve faster iteration cycles, we needed to change some parameters. We aim to discover the tradeoff between performance and efficiency of the code, our objectives being to identify which hyper-parameters will affect results more than others. The two main hyper-parameters we investigate are tasking-training parameters and meta-training parameters. Task-training controls how task-adaptable parameters get tuned to the task (given some meta-parameters). For instance, meta-parameters can be the initialisation from where task-training starts, or fixed warp-layers (not tuned). Reducing task batch size will result in faster iterations at the cost of lower performance. The number of adaptation steps can also be set separately. Meta-training governs the update of meta-parameters (such as the initialisation or warp-layers). With fewer meta-learning steps, the more influential the warp-layers, hence the faster they converge. Reducing the meta batch size gives a linear speedup.

We undertook 2 main experiments. In both cases, we reduce the meta-batch size from 20 to 5, giving a linear speed-up of $4\times$. In the first experiment, we reduce the meta-training steps, thereby decreasing the number of updates made to meta-learnt warp layers. We reduce the meta-training steps from 1000 to 250. In the second experiment, we decrease the number of task training/validation steps while keeping the number of meta-training steps at 1000. The task training/validation steps both defaults to 100, and are decreased to 25 in this experiment.

The motivation of our first experiment is to analyse the effects of reducing the number of updates, but keeping the *quality* of them the same. In contrast, our second experiment aims to investigate the effects of reducing the meta-update quality, but keeping the number of updates the same.

We argue the meta-update quality is reduced in fewer training/validation steps, as the task-trained instantiations of the model have been allowed fewer training updates. Thereby making them perform poorer in training. This means the task trained models can provide less information to the warp layers, making the meta-update less effective, and the meta-learning worse. So in our experiment, we examine the effects of the quality of meta-updates, against the number of meta-updates.

Results

We find in our experiments that a reduced number of meta-training steps has little effect on the performance of the model in testing. This is shown in the left-hand plot in fig. 1, where the performance of models is compared with varying number of tasks in meta-training set. The benchmark, shown in red, follows a very similar trend to our experiment. We do notice however that the variance in our model increases with increasing meta-training set size, this can be seen more precisely in table 1. In the second experiment, there was a significant decrease in performance as shown in the right-hand plot of fig. 1. The variance however, remains relatively consistent. In both cases we see a larger variance in our experiments.

The increased variance can be attributed to the faster convergence allowed in fewer meta-training steps. More powerful warp layers converge faster during training. This could result in them overfitting to the training/validation tasks, explaining an increased variance during testing. The consistently poor performance in the second experiments suggests effective task training is imperative in the method. In hindsight this should be expected, as if the task-trained instantiations of the model are never allowed proper training time, they can never converge on tasks. Consequently, meaningful information is never supplied in the meta-updates, giving poorer results. The larger variance in both cases, can be attributed to the decreased batch size, making updates faster but worse across both experiments, leading to worse generalisation in effective meta-parameters.

These findings are specific to the Omniglot experiment; as such, they could be an artefact of the Omniglot dataset itself. Hence, any conclusions drawn should only be considered in the domain of the Omniglot experiment.

In both experiments, we found that each model took ~ 7 hours to train/test. Compared to the 103 hour test on the baseline model. This is a speedup of $\sim 15\times$ in both cases. This suggests the additional changes to the model provided a speedup of roughly 3-4$\times$, on top of the $4\times$ increase provided by the reduced meta-batch size. Suggesting the reduction of hyper-parameters in meta-training steps and task training/validation steps have the same effect on computation time.

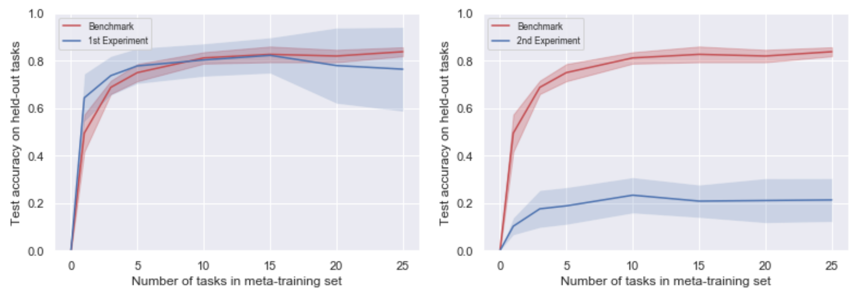

Figure 1: Omniglot test accuracies on held-out tasks after meta-training on a varying number of tasks. Shading represents standard deviation across 5 independent runs. *Left*: Warp-Leap with reduced meta learning steps. *Right*: Warp-Leap with reduced task learning steps.

Table 1: Mean test error on held out evaluation tasks for different methods.

Method	Benchmark	Reduced meta_train_steps	Reduced task_train_steps
No. Meta-training tasks			
1	49.5 ± 7.8	64.4 ± 10.1	9.9 ± 3.5
3	68.8 ± 2.8	73.7 ± 8.2	16.0 ± 7.9
5	75.0 ± 3.6	77.8 ± 7.4	18.3 ± 7.7
10	81.2 ± 2.4	80.3 ± 6.9	21.2 ± 8.2
15	82.7 ± 3.3	82.3 ± 7.5	20.2 ± 7.4
20	82.0 ± 2.6	78.0 ± 16	20.7 ± 10.7
25	83.8 ± 1.9	76.3 ± 17.7	19.7 ± 9.2

Conclusion

We reproduce a state of the art paper from the field of meta-learning. Meta-learning is the process of *learning to learn*, meaning many models are trained and tested on in the process. This oversight meant that we underestimated the computational requirements of the paper. We reduced the scope of our work to reproducing just one experiment within the paper, instead choosing to investigate the effects of model parameters on experimental results. We also attempt to optimise the codebase.

The importance of meta-update quality is compared against the number of meta-updates. We find that decreasing the meta-update quality decreases performance. It should be noted such results are specific to the Omniglot dataset. To draw more general conclusions, experiments using richer data domains should be pursued, like those using *mini* or *tiered* Image-Net. Such experiments are made in the paper, but weren't pursued in reproducing, due to the increased compute required. Two sources of inefficiency were found in the code base provided by the authors. The first is that each task in a meta-batch is run sequentially. The other, is that the losses have to be loaded from memory, which are accumulated from the meta-training step. We attempted to parallelise both processes.

If machine learning is to be moved to the usable domain, more efficient machine learning should be pursued in all cases. Reproducing a single experiment, with reduced parameters, was the most feasible task given the timescale. Consequently, we argue that the paper is not reproducible using computational resources that are typically available.

References

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent, 2019.

GitHub repository containing code used in the report: https://github.com/COMP6248-Reproducability-Challenge/Meta-Learning-with-Warped-Gradient-Descent-Reproducability-Study

We also acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.