AutoML - Few-shot learning

LanGe Chen - LIACS, Leiden University

1 Introduction

To achieve a good performance, deep neural networks specifically require a large amount of data and heavy computation which may demand a long time for training. It raises questions how to effectively train such models when only small datasets are available and how to achieve robust generalization in such case. **transferlearning** and **metalearning** are potential techniques to address this issue. In this assignment, two approaches namely **Fine-tuning** and **Reptile** [1] will be implemented for training the neural network to solve the **Sine wave regression** problem, where the model shall be able to quickly adapt its function to a new set of task parameters. To compare their influence on the performance of the model, experiments are conducted to investigate whether they can enable the networks to learn more efficiently with fewer data.

2 Background

Fine-tuning is closely linked with the term transfer learning which refers to the process that extracts features from a distribution of task and leverages them on a new, similar task. It can be done by freezing the hidden layers of the model, and then only training the output layer from the source model and incrementally adapting those pre-trained features to the new similar data. Reptile is a meta-learning algorithm that repeatedly iterates over a set of tasks, performs stochastic gradient descent on it and updates the initial parameters towards the final parameters learned from the task. The neural network can finally generalize based on a very small dataset. Both Fine-tuning and Reptile aim to cut down on the training time while still being able to obtain good accuracy. However, they work differently in terms of how they optimize and utilize the weights from the trained model.

3 Experiments

The goal is to fit a neural network to a randomly generated sine curve for a specific set of task parameters after having learned from a large distribution of different sine wave functions. [1] The experiments are conducted by utilizing Fine-tuning and Reptile to optimize the neural network, and observing the performance of the model on unseen tasks, only using a small set of points in the support set and evaluating against the query set. Particularly the choice for the size of the support set is evaluated and which impact it may have on the few-shot transfer tasks.

[1] https://github.com/LanGe-Chen/AutoML/tree/main/Few-shot_Learning

3.1 Fine-tuning

For training the source model for Fine-tuning, the dataset is generated using the non-episodic mode of the dataloader. For each training iteration over the entire distribution of sine wave tasks, the source model is trained on the sine data including 100 *samples* with *minibatch_size=10*. The *step_size* is configured as 0.01 and *SGD* is used as the optimizer. For Fine-tuning the target model, the hidden layer parameters of the source model are copied and frozen, and the weights for the output layer are re-initialized for training. The evaluation is designed by testing the target model for a fixed number of training epochs (epochs=32) on the support set of the new task and calculating the MSE loss for the query set over the last 100 iterations. The support set has a size of 20 and the query set has a size of 80. Fig. 1 shows the training and testing at different stages (left) and the learning curves (right) for evaluation. The MSE within 30000 iterations for training and testing does not improve but instead keeps oscillating. The model achieves a mean error of 5.412 on the training set before Fine-tuning, and a mean error of 5.280 on the test set after Fine-tuning respectively.

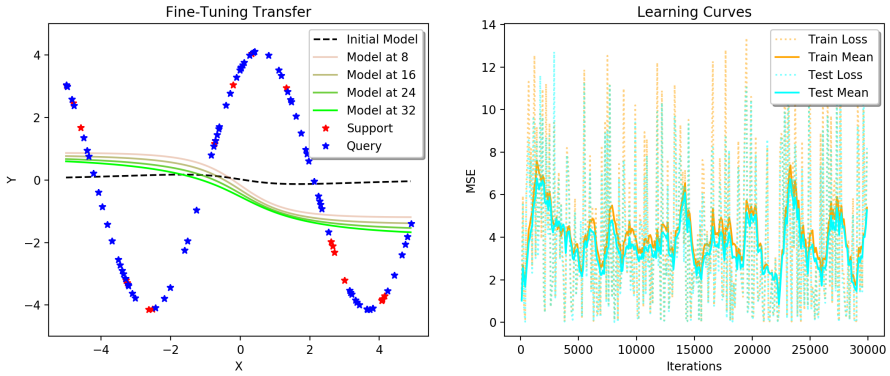

Figure 1: Visualization of the model using Fine-Tuning and corresponding learning curve

3.2 Reptile

The network is trained and evaluated using the dataset generated by different Sine wave regressions using *episodic* mode of the dataloader. The number of *innerepochs* for the inner-loop is determined as 3. The *minibatch_size* for the inner-loop training is set to 5. To avoid the model converge on each batch B of tasks too quickly which would lead to unstable optimization, the *step size* at each iteration is set to 0.01 for the inner loop. The initial outer step size is set to 0.1 and gradually decreased. The *SGD* optimizer is used for both inner- and outer-loops. First, the initial weights θ of the model are stored as a copy at the beginning of each outer loop iteration. The model weights are then trained inside the inner-loop for each task τ_J using both the training and test set. The initial weights θ of the outer-loop are finally updated after the inner-loop training using $\theta = \theta + \epsilon \frac{1}{J} \sum_{\tau_J \in B} (\theta_j^{(T)} - \theta)$, and the outer step size is downscaled over each iteration depending on the total number of iterations. Next, the model is tested for 32 training epochs on the *support_set* and evaluated against *query_set*. Fig. 2 shows the performance of the model using different support sizes ($n = 5, 10, 20, 50$) with a query set including 100 samples for evaluation. It can be observed that Reptile performs generally better than using Fine-tuning. Table. 1 further indicates that the more support samples are given, the better and more stable performance can be achieved.

Support Size	5	10	20	50
Test Mean	0.927	0.573	0.129	0.044

Table 1: The mean MSE errors over the last 100 iterations using different support size

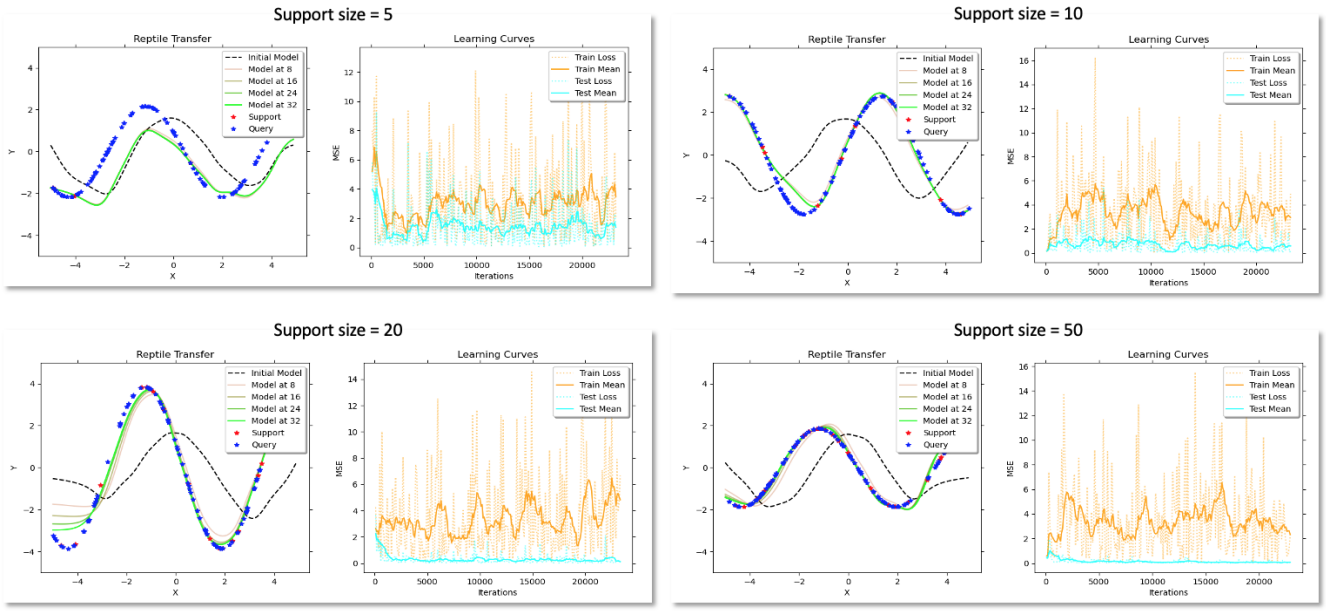

Figure 2: Visualization of the model using Reptile with different sizes of support set for evaluation

4 Discussion and conclusions

Fine-tuning and Reptile work differently in terms of how they update the weight space, where Reptile performs an outer loop weight update separately from the inner loop training. The intuition is to gradually move towards a set of source weights that can quickly adapt to new tasks, slightly updating the weights through iteration over a set of tasks, instead of greedily updating the model in direction of each individual task or the entire distribution of tasks. This leads to a much better learning efficiency of Reptile as it can find suitable weights in much less time, even though the learning rate of the outer-loop for Reptile is decreased over iterations. In contrast, Fine-tuning can perform computations faster during transfer since many of the parameters are frozen, but does not succeed to robustly find a suitable set of weights that can quickly adapt to a new unseen task, even after long training times. However, the big limitation of both approaches is that they only work if the initial and target tasks are similar and relevant enough for the pre-training in order to train on the new target. If not, *negative transfer* can easily occur, making the target problems more difficult to solve. Especially for Fine-tuning, the source model was trained with a straightforward purpose of being generic by only retraining the final weights of the network, but which may not work optimally after transferring to a new task. However, according to the results of the experiments on the Sine-wave regression problem, the performance of Reptile – even though not complicated and task-agnostic – performs much better than Fine-tuning.

References

[1] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv: Learning*, 2018.