DAAD Project: Optimization of hyper-parameters in Scientific Machine Learning

Ognjen Stefanovic

August 2023

Abstract

Using machine learning to model fluids in physical systems is an emerging research method. Instead of using numerical models, Scientific machine learning looks to utilize neural networks to give good results with less cost. These neural networks have hyper-parameters(e.g. learning rate) which are model dependant. Performance of such models is sensitive to choice of hyper-parameters. Hence, we explored hyper-parameter optimization using the learning to learn framework by [4] from the field of neuroscience, which is based on meta-learning [3]. The optimization was done on neural network models and datasets from PDEBench [2]. PDEBench is a set of time-dependant simulations based on partial differential equations. It provides datasets and code to benchmark performance of various neural networks. In this work, we showed that there is significant room for improvement. We used Evolution Strategy as the optimizer and the Unet as the optimizee in the learning to learn framework. In the end we provide useful suggestions on how to take the project further.

1 Introduction

The field of Scientific machine learning has seen rapid growth over the past few years. In this work, we focus our attention on the application of neural networks to solve partial differential equations(PDE) using high-performance computing. Traditional numerical simulations of PDEs are accurate but have an issue of being too computationally expensive to run. Instead, neural networks trained on the outputs of such simulations can learn the underlying properties of the original numerical models in a fast and effective manner. They can act as surrogates to replace these models.

The performance of neural networks is susceptible to the choice of hyper-parameters which are set by the engineer before the training starts. Hyper-parameters can be the architecture of the neural network and the optimizer related parameters (learning rate, batch size, period of learning rate decay, multiplicative factor γ of learning rate decay etc.).

We looked into exploring this hyper-parameter space using the learning to learn framework by [4]. We focused our efforts on optimizing for the benchmark metrics achieved in [2].

2 Problem Definition

Let $\vec{u} : T \times S \times \Theta \to \mathbb{R}^n$ be a vector solution to a PDE, where T is temporal space, S is a spacial domain and Θ is a function-valued parameter-space.

The mapping of the state space from the solution at one timestep to the next one, $\mathscr{F}_\theta : \vec{u}_\theta\left(t, \cdot\right) \to \vec{u}_\theta\left(t+1, \cdot\right)$ is called the forward propagator.

Finding such a forward propagator can often be hard, expensive and time consuming. Instead, Scientific machine learning looks to find a surrogate model, called an emulator. It is an approximation to the forward propagator $\hat{\mathscr{F}}_\theta \approx \mathscr{F}_\theta$.

We focused our effort mainly on the 1D Advection equation. The dataset in [2] considered pure advection behaviour without non-linearity whose expression is:

$$\partial_t u(t,x) + \beta \partial_x u(t,x) = 0, x \in \left(0,1\right), t \in \left(0,2\right], \tag{1}$$

$$u\left(0,x\right) = u_0\left(x\right), x \in \left(0,1\right) \tag{2}$$

where β is a constant advection speed. Multiple dataset with various β values are provided by [2]. We focused our effort on the $\beta = 4$ case. Note that the exact solution of the system is given as: $u\left(t,x\right) = u_0\left(x - \beta t\right)$.

The dataset only considered a periodic boundary condition. For more details on this please refer to appendix D.1 of [2].

3 Related Work

In the field of neuroscience, models have a high number of degrees of freedom and only some (unknown) parameter regions are of interest. [4] tackles the problem of finding these regions efficiently using meta-learning [3]. It uses machine learning to get high throughput hyper-parameter optimization at scale and utilizes parallelization of HPC systems. [4] provides a goodybag of neuroscience models which can explore the hyper-parameter space. It also gives various example applications of the framework.

Figure 1 showns the general L2L framework. The *optimizer* performs hyper-parameter optimization and spawns multiple *optimizees* with various hyper-parameter values. Each optimizee is e.g. an instance of a neural network, which performs training and tries to optimize some performance metric. We then map the performance metric to a *fitness* value. All the fitnesses from each optimizee get forwarded to the optimizer and based on those, hyper-parameter optimization is performed.

We chose as the optimizer evolutionary strategy by [1] (code implemented by [4]) and as the optimizee the Unet applied on predicting the solution of the advection equation.

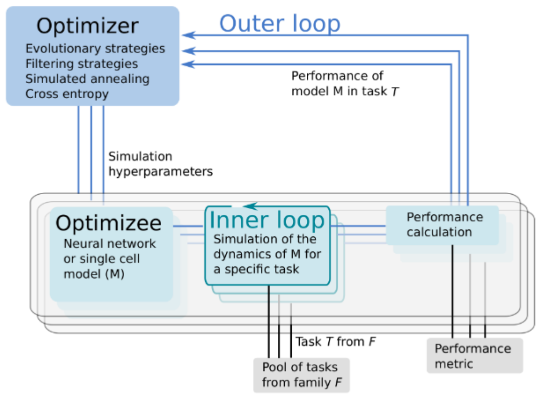

Optimizer
Evolutionary strategies
Filtering strategies
Simulated annealing
Cross entropy
Outer loop
Performance of model M in task T
Simulation hyperparameters
Optimizee
Neural network or single cell model (M)
Inner loop
Simulation of the dynamics of M for a specific task
Performance calculation
Task T from F
Pool of tasks from family F
Performance metric

Figure 1: Learning 2 learn framework

[2] provides a benchmark collection of time-dependent simulations which are based on Partial Differential Equations (PDEs). It provides open source reference for evaluating performance of new ML models on PDEs. The datasets provided span a wide range of PDEs. They also provide machine learning models such as Unet, PINN and FNO whose performance is evaluated on various performance metrics. We attempt to improve the performances of these networks using the learning to learn framework [4].

4 Experimental Setup and Installation

My repository consists of two independent parts, one is for my first month of work for replicating the metrics achieved in [2] and the second for combining the L2L framework with the PDEBench code.

First clone the repository and follow further instructions in the readme.

Here we will summarise the most important commands that we worked on during the internship.

Inside of `pdebench-daad/L2L/bin/` you can find l2l-pdebench-es.py and there the JUBE command:

```
jube_params = {"exec": "srun -n 1 -c 10 --mem-per-cpu 8000
  --gpu-bind=per_task:1 --exact python"}
```

It tells the program that each individual (which is one Unet that trains on the advection dataset) should run on one GPU. This training should be indepenent from other GPUs.

It is important that there are as many individuals in a generation as there are available GPUs. This is set in the `launch_l2l_pdebench_Unet_es.py` script with a command like:

```
    COMMAND="l2l-pdebench-es.py --lr 0.005
```

```
--pop-size 31 --n-iteration 17 --noise-std 0.001
--epochs 500 --batch-size 1000
--optimizer-lr 0.000001 --config config_Adv.yaml"
```

The –pop-size 31 is shorthand for *population size* and it says that there will be spawned 32 individuals in one generation(due to unknown reasons it spawns always one more). So on e.g. JURECA-DC we would need to set the number of nodes to 8, and with 4 GPUS per node, this would give us 32 available GPUs.

As for the other arguments, `--lr 0.005` says that the initial mean of the learning rate is $\mu = 0.005$ and `--noise-std 0.001` says that $\sigma = 0.001$ is the standard deviation.

For example, in the first generation, there are 32 individuals, and each individual gets a learning rate α which is sampled from $\alpha \sim \mathcal{N}\left(0.005, 0.001^2\right)$. In the next generation, the mean might shift to e.g. $\mu = 0.006$ and the individual learning rates would be sampled from $\alpha \sim \mathcal{N}\left(0.006, 0.001^2\right)$.

For the training of neural networks [2] used 500 epochs in Unets, PINNs and FNOs. Hence, we used the same, but the training time per one individual was big. For example, in the advection dataset, an individual which is a Unet or FNO took $1h$ to train while a PINN takes only $5 - 10mins$.

5 Results

To show that this line of work has potencial, we look at the following results. We set the optimizee as a Unet for the advection dataset. The optimizer is set to be Evolutioary strategy as in [1]. We chose to optimize for the learning rate of the Unet as a hyper-parameter. Figures 2, 3 and 4 all belong to the same experiment. We chose the fitness to be the average of the 4 most important metrics for the advection equation, $fitness = \frac{RMSE+nRMSE+bRMSE+cRMSE}{4}$. The goal of the optimizer is to get the fitness as low as possible. For more details about each of these RMSE metric please look at Appendix B of [2].

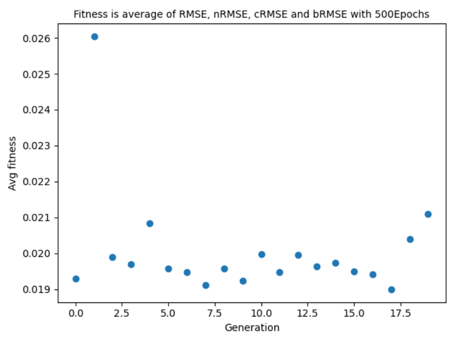
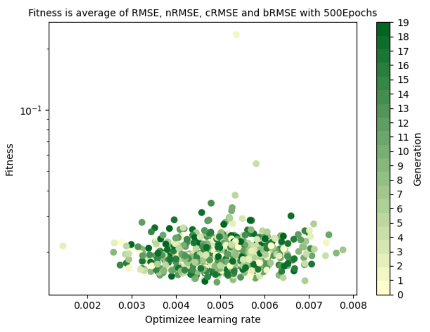
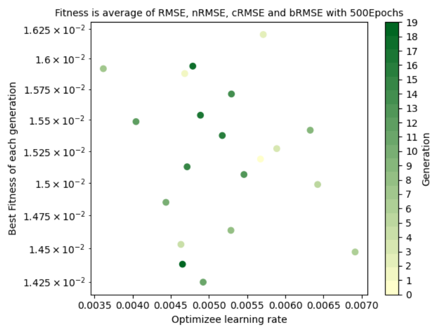


Figure 2: Average fitness in each generation



(a) Fitness of every individual in every generation



(b) Best individuals from fig. 3a in each generation with lowest fitness

Figure 3: Plots of fitness for each generations

In figure 2 we expected the average fitness to go down, but it did not. In some smaller scale tests, we did get the fitness to go down, but here that did not seem to happen.

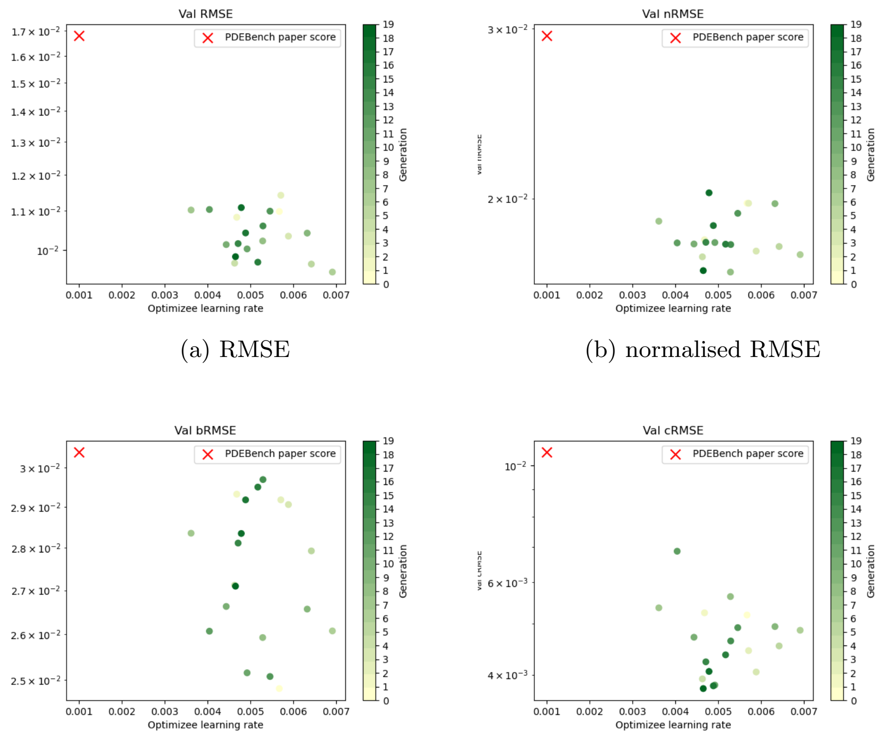
In figures 3 and 4 we expected that the light green dots appear further up the y axis and the dark green dots further down, but again this did not happen.

On the positive side we are outperforming the results achieved in [2], which can be seen in figure 4 because all green dots are below the red cross. This gives hope for taking the project further.

The issue with using a Unet in the Advection equation dataset was that training a single Unet individual took too long. Hence, this would make the total number of generations too small (up to 25 generations). A good amount of generations should be at least 80 so that the optimizer learning algorithm has time to learn good hyper-parameter values. With only 25 generations, the success of the optimization depends mostly on the initial values of the hyper-parameters set by the engineer (here it was $\mu = 0.005$ for the learning rate and $\sigma = 0.001$ for the noise) and somewhat on the optimizer learning algorithm. This issue cannot be fixed by the type of optimizer we choose, because it is the training time of the optimizee that is the issue.

Another issue found with [1] is how the initial value of the hyper-parameters need to be set to a good value. The point of performing hyper-parameter optimization in an unknown landscape becomes questionable, since we initially had a quite good value. This is quite worrying for taking the project forward. Perhaps this is something worth discussing with researchers from neuroscience or just to change the optimizer that is being used to e.g. genetic algorithm, cross-entropy etc.

Another problem is how would this scale up to multiple hyper-parameters. If we have two hyper-parameters with different orders of magnitude, [1] only permits to have one value of noise standard deviation σ for the multivariate Gaussian which we are sampling from. For example, lets say we tried to optimize



(a) RMSE

(b) normalised RMSE

(c) boundary RMSE

(d) conserved RMSE

Figure 4: Various RMSE metrics for the individuals from figure 3b

for the learning rate α and its scheduler decay of learning rate γ. We know that the optimal learning rate lies in $\alpha^* \in \left(10^{-5}, 10^{-3}\right)$, while $\gamma^* \in \left(10^{-2}, 0.5\right)$, but we can only use e.g. $\sigma = 0.001$ when sampling for various individuals. We did not have time during the project to try and resolve this issue.

One of the answers could be that we are applying [1] incorrectly or we shouldn't even apply [1] to our problem, because the scales of the hyper-parameters differ by large magnitudes and perhaps [1] was not mean to be used for this kind of problem.

My inherit feeling after reading the applications in various problemsets in [4] is that these learning algorithms are not used to fine tune hyper-parameters and get excellent values starting from good values, but to get rather decent/good hyper-parameter values starting from bad ones in an unknown parameter space.

Learning to learn did not show to scale well with the number of nodes. We managed to get good GPU utilization of about 70% when using 2, 4 and 8 nodes. But when using 32 nodes the utilization dropped down significantly to only 20% which is bad. We did not look into why does this happen.

6 Conclusion and Future Work

We tried to apply methods used in neuroscience to our problem of hyper-parameter optimization. We focused most of our attention towards the Unet and optimizing for the learning rate. The results show that there is significant room for improvement of the benchmarks recorded in [2]. The evolutionary strategy as in [1] gave some improvement, but the issue was the training time of each Unet individual was too long, so only a small amount of generations (up to 25) could be trained, which was realized late in the project.

One way of going forward is to reevaluate if it is worth trying to pursue this approach as is, simply because of the training time required on one Unet on one GPU. Perhaps the best way to continue this line of work is to look how to paralelize the code to use multiple GPUs when training and to try and improve the GPU utilization.

A simpler approach would be to look into using PINNs as optimizees due to their low training and inference time.

Another suggestion is to read more into detail which of the optimizer algorithms from [4] would be suitable. I feel as if we took Evolution Strategy by [1] without looking into much detail about the other optimizers available.

As far as evolutionary strategy goes, we had to fine tune the noise standard deviation σ by hand (and the fine tuning of σ depends on the initial value of μ). This means that we had to optimize for σ which is then used to optimize for the hyper-parameter α, the learning rate. So there was implicitly another layer of hyper-parametrization. Perhaps there is not such an issue with the other optimizer techniques available in [4].

To summarise, in this project we looked into improving the benchmarks achieved in [2] using high-performance computing. This was done using the learning to learn library. We showed that this line of work has great potential.

References

[1] Tim Salimans et al. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning.* 2017. arXiv: `1703.03864 [stat.ML]`.

[2] Makoto Takamoto et al. *PDEBENCH: An Extensive Benchmark for Scientific Machine Learning.* 2023. arXiv: `2210.07182 [cs.LG]`.

[3] Sebastian Thrun and Lorien Pratt. *Learning to Learn.* Springer Science & Business Media, 2012.

[4] Alper Yegenoglu et al. "Exploring Parameter and Hyper-Parameter Spaces of Neuroscience Models on High Performance Computers With Learning to Learn". In: *Frontiers in Computational Neuroscience* 16 (May 2022). DOI: `10.3389/fncom.2022.885207`. URL: `https://doi.org/10.3389%2Ffncom.2022.885207`.