CoOpt: Comparing Optimizers

Gurnoor Singh Khurana
gurnoor.khurana@epfl.ch

Aayush Kumar
aayush.kumar@epfl.ch

Pradhit CR
pradhit.canchirangam@epfl.ch

***Abstract*—Two very new innovations in the field of optimisation are Sophia and Lion. The Lion optimizer is a memory efficient optimizer, which stores only the momentum values. Unlike traditionally used adaptive optimizers, the Lion optimizer has an update of same magnitude for all the parameters. Sophia, on the other hand, is a second-order optimizer. Generally, using second-order optimizers is very costly as they involve the calculation of the Hessian matrix. In this project, we compare the performance of these recent optimizers with the widely used Adam and SGD optimizers.**

I. Introduction

Optimizers are an essential component in training machine learning models. The process of training a machine learning model boils down to a minimization problem. Over the years, various optimizers have been developed and have become standard such as SGD, Adam, etc. Most of these optimizers are first-order, that is, they use the value of the first derivative of their objective function to make updates to the parameters. Furthermore, some of these optimizers, such as Adam, are adaptive optimizers which means that the parameters are updated with a different value in each step.

Recently, two new optimizers have been proposed: Lion [1] and Sophia [2]. Lion optimizer has been developed by researchers at Google, and unlike adaptive optimizers, its update has the same magnitude in each step. It is a first-order optimizer.

Sophia on the other hand, is a second-order optimizer. Second-order optimizers are seldom used, owing to the large per-step overhead, which makes them very slow and practically unusable. Therefore, instead of using second derivatives, often an approximation of Hessian is used. Sophia, in particular, uses an estimate of the diagonal Hessian as the preconditioner.

In this project, we compare the traditional and established optimizers such as SGD and Adam with the recently introduced Lion and Sophia optimizers.

II. Models and Methods

We originally intended our project to contain an implementation of Newton's method and some of its variants such as damped Newton, Secant and damped Secant. While we did implement these methods, they have several practical limitations.

Firstly, it is not scalable. We implemented Newton and Quasi-Newton methods on the `Concrete_Data.csv` dataset for MSE loss. The pseudocode for this implementation can be found in Algorithm 1. The optimization function for MSE Loss is $\frac{1}{N}||Ax - b||^2$. The hessian of this function is given by $A^T A$. As we can see, the Hessian matrix has the dimensions $m \times m$ where m is the number of parameters in our model. Since Newton's method involves calculating the inverse of the Hessian, which is typically $O(m^3)$, this method can only be applied in very limited scenarios.

Algorithm 1 Damped Newton's Optimizer for MSE loss

```
Require: Data, max_iters, $\lambda$, lr
 1: $iter \leftarrow 0$
 2: $x \leftarrow 0$
 3: while $iter < max\_iter$ do
 4:     $ind \leftarrow$ sample a random batch of indices from data
 5:     $A \leftarrow Data[ind]$
 6:     $b \leftarrow target[ind]$
 7:     $H \leftarrow A^T A$
 8:     $grad \leftarrow A^T(Ax - b)$
 9:     $x \leftarrow x - lr * (H + \lambda I) * grad$
10: Output: $x$
```

Secondly, the expression for Hessian in the case of a multi-layered CNN is not trivial. In the case of a single-layer perceptron, the dependency of the loss function on the parameters (W) is clear: $loss = \frac{1}{N}||WX - Y||^2$. So, the hessian can be easily calculated. However, in case of a multi layered CNN, as in [3], this dependency between parameters and the loss function is not so trivial. Hence, calculation of hessian is a tough task in itself. We also tried using PyTorch's `torch.autograd.functional.hessian`, but this does not solve the issue either.

Owing to these issues, direct implementation of Newton's optimizer was not scalable, and thus could not be applied on any real world datasets. Therefore, we resort to using pre-implemented second-order optimizers, such as the Sophia optimizer.

In our experiments, we compare the performance of various optimizers on two tasks: image classification on the CIFAR-10 dataset and regression on the `Concrete_Data.csv` from lab05. We chose these tasks for the following reasons:

- **Task structure:** Image classification and regression, have different objectives and cover a wide range of machine learning problems. Image classification focuses on identifying patterns in pixels to predict objects, while regression involves modeling a scalar based on specific features. By comparing optimizer performance on these tasks, we can assess their robustness and adaptability to different task structures.

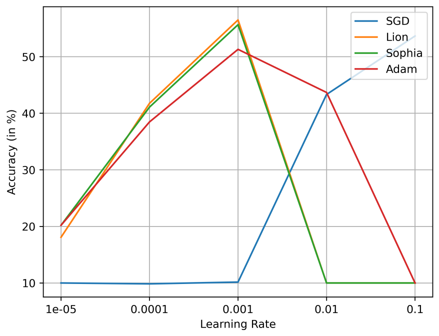

Fig. 1: Plot of accuracy vs learning rate for various optimizers on the CIFAR-10 dataset with batch_size 256

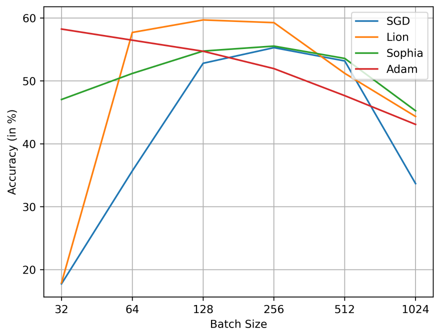

Fig. 2: Accuracy vs batch size on the CIFAR-10 dataset with learning rates SGD : $1e^{-1}$, Lion: $1e^{-3}$, Sophia: $1e^{-3}$, Adam: $1e^{-3}$

- **Model Structure:** Image classification tasks typically rely on using CNNs. Regression, on the other hand, makes use of a single-layer perceptron. Since CNNs and SLPs have very different model architectures, with SLPs being fairly simple, optimisers tend to have varying degrees of effects on these architextures.
- **Hyperparameter Sensitivity:** Since optimizers are sensitive to hyperparameters, we found it worthwhile to check the sensitivity of hyperparameters based for these different tasks.

To measure the performance of an optimizer on each task, we compare the effect of the following parameters:

- **Learning Rate:** Given a fixed batch size, we compare the accuracy by varying the learning rate.
- **Batch Size:** Given a fixed learning rate for each optimizer, we compare the accuracy by varying batch size.
- **Time taken:** Given a fixed number of steps, we calculate the time taken by each optimizer. This helps us determine the efficiency of each step of the optimizer.

For the sake of simplicity, we use simple models for our tasks. For the image classification task, we use a Convolutional Neural Network in our experiments. This model architecture has been taken from the official CIFAR-10 Pytorch tutorial [3]. For the regression task, we use a single-layer perceptron.

All the experiments have been performed on Google Colab using the GPU for the CIFAR-10 dataset, and locally using CPU for the regression task.

III. Results

A. Image Classification

Figures 1 and 2 show a plot of accuracy vs learning rate and accuracy vs batch size for various optimizers.

Optimizer	SGD	Adam	Lion	Sophia
Avg. time taken (s)	30.36	31.03	30.35	30.26

TABLE I: Training time (s) on the CIFAR-10 dataset

From figure 1, we observe that the Lion and Sophia optimisers are able to give the best accuracy overall, with Lion edging just slightly ahead of Sophia. Lion, Sophia, and Adam all perform best for the same learning rate ($1e^{-3}$), and while Adam does not perform as well as Lion or Sophia for this learning rate, it is able to accommodate a learning rate of $1e^{-2}$ without the accuracy falling as drastically as it does for Lion or Sophia. This suggests that Adam is less sensitive to learning rates near its peak performance.

Another interesting observation is that the Lion and Sophia optimizers have very similar performances throughout.

While the graphs for the performances of Lion, Sophia, and Adam are somewhat similar, the performance of SGD follows a completely different pattern, and seems to suit higher learning rates much better. Its best performance, at a learning rate of 0.1, is almost similar to the best performance of the Adam optimiser.

The literature on the Lion optimiser [1] points out that the Lion optimizer requires a smaller learning rate than Adam, since the magnitude of update in Lion optimizer is larger. Figure 1 contradicts this since the peak accuracy of Adam, Lion and Sophia optimizers is obtained at the same learning rate.

The variation of accuracy with variation in batch size (Figure 2) showed much more irregular results. While the performances of the Lion, Sophia, and SGD optimizers seems to peak for a particular batch size and then fall, the performance of the Adam optimiser monotonically deteriorates as the batch size increases.

Overall, we find that Lion has a notably better performance than the other optimizers. The performance of Lion peaks at a batch size of 128 and then decays in a much slower rate than the decay with variation in learning rates. SGD peaks at a batch size of 256 and has similar behaviour, though its performance decays very steeply for batch sizes not around the optimal batch size of 256.

Sophia also peaks at a batch size of 256, and has a remarkably stable performance for batch sizes around this peak. While it does not deliver the best performance, it is much more robust to variations in batch size.

The Adam optimiser peaks at a batch size of 32 and its performance almost linearly decays with an increase in batch size. This decay is also not as steep as the same for variation in learning rates.

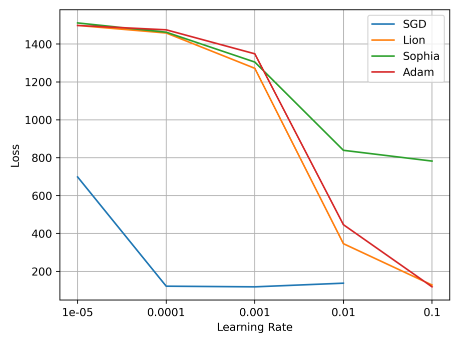

Fig. 3: Plot of loss vs learning rate for various optimizers on the `Concrete_Data.csv` dataset with batch_size 8

We also tested the average training time over 10 instances of training the model for each optimiser, with a learning rate of $1e^{-3}$ and a batch size of 512 (Table I). While Adam has a slightly higher time on average, the time taken is almost identical between all optimisers and does not make a case as an important factor for choosing an optimizer for a particular task. It is important to note that the actual execution time can vary due to factors such as CPU model, running processes, etc., making it difficult to reproduce these specific time values.

B. Regression

Figure 3 and 4 show a plot of loss vs learning rate and loss vs batch size for various optimizers.

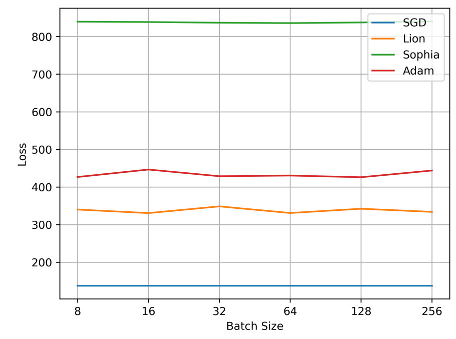

Fig. 4: Plot of loss vs batch size for various optimizers on the `Concrete_Data.csv` dataset with learning rate $1e^{-2}$

In figure 3, we can observe a somewhat similar pattern for the graphs of Lion, Sophia, and Adam, while SGD shows unique behaviour. Overall, SGD shows a better performance for the regression task, and for learning rates greater than $1e^{-3}$ has a small, almost constant loss. While SGD shows a drastic decay in loss from a learning rate of $1e^{-5}$ to $1e^{-3}$, the other optimizers show the biggest change when the learning rate increases from $1e^{-3}$, to $1e^{-2}$,.
Adam and Lion behave almost identically for this task across all learning rates, with the Lion optimizer performing marginally better at each step.
Sophia behaves similarly to both Lion and Adam till the learning rate of $1e^{-3}$, but following this, the loss for Sophia does not decay as quickly as it does for Lion and Adam and it performs worse overall.

In the plot for change in loss for different batch sizes (figure 4), all the optimizers show an approximately similar pattern but with different absolute values of loss. The loss for each optimiser is practically invariant with chnages in batch size. However, in terms of magnitude, we can see that SGD performs the best while Sophia performs the worst.

IV. Discussion

On the whole, we observe that the Lion optimizer behaves quite similarly to the Adam optimizer, but shows slightly better results. By appropriately identifying the learning rate and batch size, using Lion is a better option than using Adam.
SGD performs much better than all the other optimisers on the regression task, and it is clear why it is the most popular optimiser for regression. It also performs well on the image classification task, but using other optimizers could be preferred.
The Sophia optimiser is a second-order optimizer, and keeping that in mind, the fact that it takes a similar amount of time to train is quite impressive. This can be explained from the following text taken from the literature on Sophia ([2]): *Sophia only estimates the diagonal Hessian every handful of iterations, which has negligible average per-step time and memory overhead.* However, this approximation perhaps reflects in the results as it does not provide significantly better accuracy than the first-order methods. In fact, overall it has the worst performance on the regression task, and it does not have the best results for the image classification task. Thus, in most contexts, it is likely for another optimizer to be preferred, as the benefits of being a second-order method are not reflected in the results of the Sophia optimizer for these tasks.

V. Summary

In this project, we compare the effect of different optimizers on the tasks of image classification and regression. We observe that the newly developed Lion and Sophia optimizers can compete with the performance of established and popular optimisers Adam and SGD, and that the Lion optimizer is perhaps an improvement over the Adam optimizer. In the coming years, it is likely to see these new optimizers being used regularly alongside the currently popular ones, especially Lion. We also first-hand learnt the practical difficulties of using Newton's optimizer in machine learning, despite its very fast theoretical convergence.

VI. Acknowledgements

We acknowledge that we have used various external sources in our project. Specifically, we would like to point out to the official PyTorch tutorial on CIFAR-10 [3], implementation of the Lion optimizer on github [4], implementation of the Sophia optimizer on github [5], the `Concrete_Data.csv` from lab05 [6].

References

[1] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le, "Symbolic discovery of optimization algorithms," 2023.

[2] H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma, "Sophia: A scalable stochastic second-order optimizer for language model pre-training," 2023.

[3] "Pytorch cifar-10 tutorial," https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/4e865243430a47a00d551ca0579a6f6c/cifar10_tutorial.ipynb.

[4] "Pytorch implementation of lion optimizer," https://github.com/google/automl/blob/master/lion/lion_pytorch.py.

[5] "Pytorch implementation of sophia optimizer," https://github.com/Liuhong99/Sophia/blob/main/sophia.py.

[6] "Dataset from lab05 of cs439 at epfl," https://github.com/epfml/OptML_course/blob/master/labs/ex05/template/Concrete_Data.csv.