Comparison of optimization algorithms for image classification

Edouard Dufour, Yassamine Saidi, Monika Stoilova
School of Computer and Communication Sciences, EPFL

***Abstract*—Deep neural networks have achieved remarkable performance in many applications. However the considerable success of deep learning is accompanied by constant improvements in the optimization algorithms used for the training process of neural networks on large datasets. Stochastic gradient descent (SGD) is one of the canonical optimization algorithms used in training neural networks. However, recent advances have also considered other variants of the standard algorithms such as momentum-based techniques, as well as algorithms with adaptive learning rates. In this paper, we focus on the task of image classification using convolutional neural networks (CNNs) on the popular dataset MNIST. The CNNs are trained using optimizers such as SGD, Momentum SGD, Adam and the more recent Lion optimizer and their performance is compared.**

I. Introduction

In recent times, significant progress has been made in the realm of optimization algorithms, in particular in relation to the Stochastic Gradient Descent (SGD). These advancements have led to the emergence of a wide range of its alternative variants and proved their efficiency in the neural network training. In this study, we focused on the task of image classification and we compared the performance of SGD, momentum SGD, ADAM and Lion algorithms after training convolutional neural networks (CNNs) on the widely recognized MNIST dataset. At first, we present the different methods and their pseudo-codes: Why do they have these specific structures? What are the hyperparameters and their role? how do we explain the basic idea behind every step? It is important to mention that there are two types of Lion algorithms: The one that we use in the results, is discovered recently (last version may 2023) and based on sign update and first-order momentum tracking. The other lion optimization algorithm is a nature-inspired algorithm discovered in 2015 and based on the evolution of lion populations. (See B in the Appendix for more details). Next, we introduce our model and its CNN architecture, the dataset employed, and the hyperparameter values used from the different optimizers during the training phase. Finally, we report the numerical results and analyze the plots in order to facilitate a performance comparison of the algorithms within the designated task.

II. Algorithms

Optimization algorithms are defined by an update rule and their behaviour is controlled by a set of hyperparameters. In this paper, we will use and compare first-order iterative optimization methods. In particular, let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable loss function whose gradient is given by $\nabla f(\mathbf{x})$ and $\mathbf{x} \in \mathbb{R}^d$ represents the vector of model parameters. In the deep learning framework, the loss function f is usually computed by a neural network over an entire dataset. The goal of the optimization task is to find the point $\mathbf{x}^\star$ that minimizes f. The basic idea is to construct a set of iterates $\mathbf{x}_k$ that converges to the minimum. Given an initial starting point $\mathbf{x}_0$, the sequence $\{\mathbf{x}_k\}$ is constructed by an update rule, the latter being controlled by a set of hyperparameters and used to compute $\mathbf{x}_{k+1}$ based on the previous iterate $\mathbf{x}_k$, its loss function $f(\mathbf{x}_k)$ and the corresponding gradient $\nabla f(\mathbf{x}_k)$.

A. SGD

One of the most common algorithms used for optimization is the stochastic gradient descent (SGD) method. It has many strengths that make it very popular in the machine learning field. It is computationally efficient as it computes the gradient of the loss function using only one randomly selected data point at each step, or most often a random subset of the data (mini-batch). This approach significantly reduces the computational complexity of the actual gradient, which is computed using the whole dataset at each step. SGD is typically used with a decreasing step size α_k.

Algorithm 1 SGD

```
Step size $\alpha$.
Initialise $\mathbf{x}_0$.
for $k = 1, 2, ...$ do
    $\left\{ \begin{array}{l} \text{Select } i \in \{1, ..., n\} \text{ uniformly at random} \\ \mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \nabla f_i(\mathbf{x}_{k-1}) \end{array} \right.$
end for
```

Furthermore, because of the importance of momentum in deep learning, we will also implement SGD with momentum, which takes into account past values of gradients when updating the iterates. Momentum accelerates gradients in the right direction. The update rule for SGD with momentum is given in Appendix A.

B. Adam

The ADAptive Moment estimation algorithm [2], also known as ADAM, is an extended version of SGD based on an adaptive estimation of first-order and second-order moments.

The main steps performed by ADAM are first-order and second-order moment estimation of the gradients, bias correction compensating for the initialization bias and adaptive learning rates used for update of model parameters.

Algorithm 2 ADAM

```
Step size $\alpha$.
Exponential decay rates $\beta_1, \beta_2 \in [0, 1)$.
Objective function $f(\mathbf{x})$ with parameters $\mathbf{x}$.
Set $\mathbf{x}_0 = 0, \mathbf{m}_0 = 0, \mathbf{v}_0 = 0$.
for $k = 1, 2, ...$ do
    $\begin{cases} \mathbf{g}_k &= \nabla f(\mathbf{x}_{k-1}) \\ \mathbf{m}_k &= \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}_k \leftarrow \text{Momentum} \\ \mathbf{v}_k &= \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)\mathbf{g}_k^2 \leftarrow \text{Adaptive term} \\ \hat{\mathbf{m}}_k &= \mathbf{m}_k/(1-\beta_1^k) \leftarrow \text{bias correction} \\ \hat{\mathbf{v}}_k &= \mathbf{v}_k/(1-\beta_2^k) \leftarrow \text{bias correction} \\ \mathbf{H}_k &= \sqrt{\hat{\mathbf{v}}_k} + \epsilon \\ \mathbf{x}_k &= \mathbf{x}_{k-1} - \alpha \hat{\mathbf{m}}_k / \mathbf{H}_k \end{cases}$
end for
```

Note that all operations in Algorithm 2, when applied to vectors, are applied element-wise.

The hyperparameters used by ADAM need to be tuned carefully, depending on the problem and the dataset at hand, in order to optimize the performance of the algorithm. The learning rate α controls the magnitude of the parameter updates, whereas the hyperparameters β_1 and β_2 aim to control the decay rate of the updates. Finally, a constant ϵ is added to the denominator of the update step for numerical stability purposes. In fact, the square root of the second moment estimate ($\hat{\mathbf{v}}_k$) may take a value close to zero and dividing by it directly would lead to numerical instability, while adding the small constant epsilon solves this issue. Typical values of these parameters are $\alpha = 0.001, \beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

C. Lion (EvoLved SIgn MOmeNtum)

Algorithm 3 Lion

```
Learning rate $\alpha$.
Exponential decay rates $\beta_1, \beta_2 \in [0, 1)$ for momentum tracking.
Decoupled weight decay (strength) $\lambda$.
Objective function $f(\mathbf{x})$ with parameters $\mathbf{x}$.
Set $\mathbf{x}_0 = 0, \mathbf{m}_0 = 0$.
for $k = 1, 2, ...$ do
    $\begin{cases} \mathbf{g}_k &= \nabla f(\mathbf{x}_{k-1}) \\ \mathbf{c}_k &= \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}_k \\ \mathbf{c}_k &= \text{sign}(\mathbf{c}_k) + \lambda \mathbf{x}_{k-1} \leftarrow \text{update factor} \\ \mathbf{x}_k &= \mathbf{x}_{k-1} - \alpha \mathbf{c}_k \\ \mathbf{m}_k &= \beta_2 \mathbf{m}_{k-1} + (1-\beta_2)\mathbf{g}_k \leftarrow \text{Momentum} \end{cases}$
end for
```

The lion optimizer, appeared in the recently published paper "Symbolic Discovery Of Optimization Algorithms" by Google Brain [1], is a simple and efficient method based on the track of momentum to solve optimization problems and train deep neural networks. In fact, Lion (EvoLved SIgn MOmeNtum) uses the sign-operator to define a uniform magnitude across all dimensions during the update. This is actually "an element-wise binary update ± 1", if we ignore the term of the weight decay defined as the product of the decoupled weight decay of strength λ and the learning rate α. Due to the large norm produced by this sign-function, the algorithm needs a very small α and hence a large value of λ. The main output of the algorithm is the updated weight $\mathbf{x}_k$. Unlike ADAM, Lion requires only one variable to estimate the first moment. This variable $\mathbf{m}$, initialized at zero before the training, is used to collect information during the training and therefore involved in the update rule. It depends on the hyperparameters β_1 (interpolating factor, usually $= 0.9$) and β_2 (the default EMA factor $= 0.99$). Because Lion only needs to save the momentum $\mathbf{m}$, it is faster than the other optimizers and more efficient while training big models.

III. Model and Method

A. Dataset

The performance of the optimization algorithms presented in SectionII is evaluated on the MNIST image dataset [3], which contains a training set of 60'000 examples and a test set of 10'000 examples. Each example is a 28×28 pixel gray-scale image of single handwritten digits between "0" and "9".

B. Model structure: CNN architecture

Convolutional neural networks (CNN) are a class of artificial neural networks most commonly used in image classification tasks. In fact, they are designed to deal with the extraction of features from two-dimensional input data, such as pixel images, and therefore outperform many other techniques applied on image recognition tasks.

The building block of a typical CNN consists of an input layer connected to multiple blocks of convolutional and sub-sampling layers, and finally a block of one or several fully connected layers which produce the final output layer.

The CNN architecture that we use includes one convolutional layer, one max-pooling layer and two fully connected layers. The convolutional layer applies 10 different 5×5 filters to the input image with one channel (grayscale image) and produces 10 feature maps as output. The goal of the convolution layer is to extract local features of the input images and capture simple or more complex patterns. Then, a rectified linear unit (ReLu) activation function is applied. The purpose of the activation function is to introduce non-linearity in the network and allow the CNN to capture non-linear relationships between features. Then a pooling layer is applied, which means down sampling of the image. The input of this layer is the output of the convolutional layer and it acts on it by sub-sampling small regions of it to produce one single sample. In our model, we use one max-pooling layer with kernel size 2×2 and stride 2×2, which takes the largest pixel value of a 2×2 sample of the convolutional output. Finally, two fully connected layers are added: the first fully connected layer takes the flattened output of the convolutional layers and maps it to a 500-dimensional feature space before applying a ReLu activation function. The second fully connected layer maps the 500-dimensional output to a 10-dimensional output since the number of output classes is 10, i.e. $\text{Card}(\{0, 1, \ldots, 9\})$. The output is then passed through the logarithmic softmax function to produce the class probabilities predicted by the model.

The loss function used in our model is the negative log-likelihood.

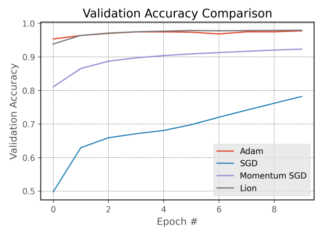
C. Optimizers and Hyperparameters

For the optimization algorithms, we use default values of the hyperparameters that control their update rules. In particular, we use a learning rate $\alpha = 10^{-3}$ for SGD. As for ADAM, we use the typical values of the hyperparameters presented in II-B, i.e. a learning rate $\alpha = 10^{-3}$, exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and a numerical stability constant $\epsilon = 10^{-8}$. The Lion otpimizer is run with a learning rate of 10^{-4}, with a weight decay $\lambda = 10^{-2}$ and exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We set the batch size equal to 64 and perform 50 epochs in total.

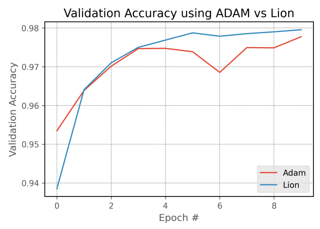
Fine-tuning the hyperparameters of these optimizers is a crucial step as it can improve the overall performance of each optimizer individually. However, in this paper, we focus on comparing their performance when using the default values, widely used by the scientific community and larger.

IV. Results

This section presents the numerical results obtained after training our CNN using the three different optimization algorithms presented previously. Figure 1 compares the accuracy obtained on the validation set using all considered optimizers. Figure 2 compares the validation to the training loss for all four optimization algorithms.



(a) Validation accuracy comparison between all optimizers.



(b) Validation accuracy comparison between ADAM and Lion.

Fig. 1: Accuracy obtained on validation set using different optimizers for CNN training.

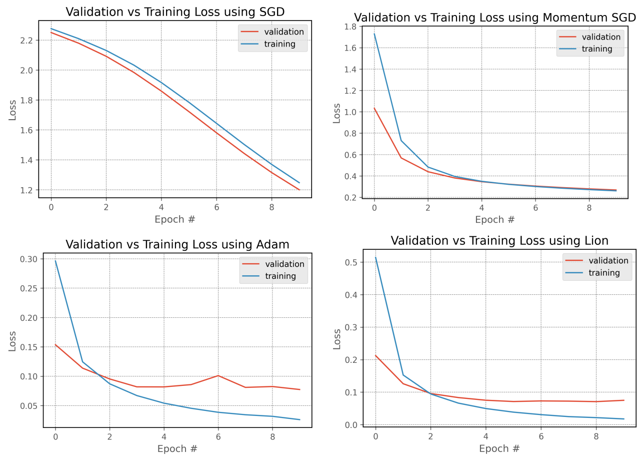


Fig. 2: Comparison of validation vs. training loss obtained using SGD, Momentum SGD, ADAM and Lion.

The final accuracy obtained on validation set is reported in Table I.

	SGD	Momentum SGD	ADAM	Lion
Accuracy [%]	78.2	92.3	97.8	98.0

TABLE I: Accuracy obtained on validation set after 10 epochs using all considered optimizers.

V. Discussion

As shown in Figure 1a and Table I, under this setting, the SGD is outperformed by all other algorithms. In particular, the accuracy increases very slowly in the beginning, compared to the other algorithms (78.2% accuracy after 10 epoch). The momentum SGD improves the performance of SGD ($\sim 92.3\%$) but is outperformed by both Lion ($\sim 98.0\%$) and Adam ($\sim 97.8\%$). However, it is not clear which of ADAM and Lion performs better. Figure 1b shows that, after one epoch, the performance of Lion, in terms of accuracy (and loss, see Figure 4 in Appendix C), is better than the accuracy obtained with ADAM. Finally, in our experiments (Figure 2), we find that the generalization performance of SGD and SGD with momentum remains superior, compared to the ADAM and Lion optimizers, as its loss on the validation set continues to decrease, while for ADAM and Lion it is not the case, suggesting that a slightly more serious overfitting occurs with those. It might be more useful to proceed to a comparison with a larger number of epochs, although more computationally expensive, in order to make valuable observations.

VI. Conclusion

In conclusion, this paper sheds light on the advancements in the different optimization algorithms by comparing the effectiveness of SGD, momentum SGD, ADAM, and Lion in training CNNs for image classification and providing valuable insights into the performance characteristics of these algorithms. Based on the results, Lion turns out to be one of the most memory-efficient and powerful algorithms to solve optimization problems. Further experiments would be changing the hyperparameters of the optimizers, the datasets, the CNN architecture (e.g. multiple layers) or even the task (e.g. computer vision, masked language modeling). However, for some tasks or, for example, when reducing the batch size ($\lesssim 64$), the performance gain of Lion decreases remarkably or stays similar to the performance of Adam.

References

[1] Xiangning Chen et al. “Symbolic Discovery Of Optimization Algorithms”. In: *Google Brain* (2023).

[2] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[3] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[4] Maziar Yazdani and Fariborz Jolai. “Lion Optimization Algorithm (LOA): A nature-inspired metaheuristic algorithm”. In: *Journal of Industrial Design and Engeneering 3* (2015), pp. 1–13.

APPENDIX

A. Momentum SGD

Algorithm 4 Momentum SGD

```
Step size $\alpha$.
Momentum $\mu$.
Initialise $\mathbf{x}_0, \mathbf{v} = 0$.
for $k = 1, 2, ...$ do
    $\begin{cases} \text{Select } i \in \{1, ..., n\} \text{ uniformly at random} \\ \mathbf{v}_k = \mu \mathbf{v}_{k-1} + \nabla f_i(\mathbf{x}_{k-1}) \\ \mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \mathbf{v}_k \end{cases}$
end for
```

B. Lion Optimization Algorithm (LOA)

LOA is a nature-inspired algorithm based on the lifestyle of lions and their way to interact with each other. Since its discovering in 2015, it has shown remarkable performance in many complex optimization problems [4]. We won't use this Algorithm in our results due to the complexity of its details but in this section, we will explain how this algorithm got inspired from the nature.

First, we generate our initial population of lions N_pop randomly over the solution space [a,b]. Every lion is a solution which is evaluated using an objective function f ("fitness function") to maximize. The lion population has two types of social organisation. [4].From the previous generated lions, we choose $N\%$ that are nomad, the rest are residents and live in prides P. Nomads live either singularly or in pairs. The pairs are basically males, who were excluded at maturity from their prides and they are less powerful than the males of the residents. A pride is composed of 5 females, their cubs (females and males) and one or two males and has its own territory.

For each pride, we select randomly ($N_{hunters}$) female lions that go hunting and for the others we choose a "safe" place among the territory by tournament selection [4]. The males on the other hand roams in the pride territory ($R\%$) are visited by each lion. And every time, if the discovered new position is better than the lion's best position, we update its personal best visited solution. At the end of the roaming, we move the lion to this best position. In the algorithm there are more details such as immigration (with rate $I\%$), "war" between nomads and residents and mating ($Ma\%$ of the females), which enables the system to keep the stability of its population.

At the end of the iteration, the lions are ordered according to their fitness values. The best females will be distributed among the prides and the weakest nomad males will be removed.The condition to stop the algorithm may be the maximum number of iterations or maximum number of iterations without noticing an improvement.

C. Additional graphs

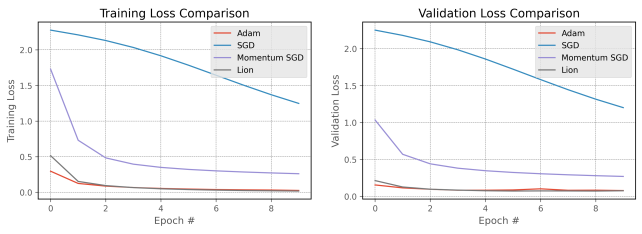


Fig. 3: Convergence of the algorithms as measured by the loss function.

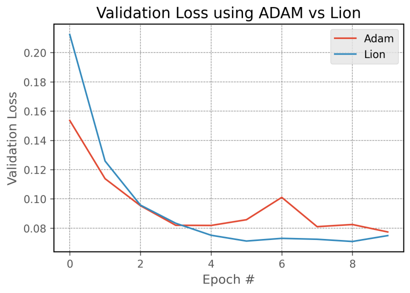


Fig. 4: Validation loss using ADAM vs. Lion.