Comparison of First-Order Stochastic Optimization Methods for Deep Learning Problems

Paul Guillot **Louis Latournerie** **Yanis Remmache**

Abstract

In Deep Learning, training a model involves solving an optimization problem that aims to adapt the weights in order to minimize a cost function. Therefore, optimization is a fundamental task in Deep Learning, and poor optimization of the model will lead to suboptimal performance, even with a well-suited architecture. This is why several optimization algorithms specifically tailored for Deep Learning have been invented. In this study, we first briefly introduce the main algorithms that have proven their effectiveness and are most commonly used. All the algorithms presented here have been coded from scratch, and the code is available on the corresponding GitHub repository. Subsequently, we have tested and compared the performance of these algorithms in several contexts encountered in Deep Learning: image classification, text classification, and regression. For this purpose, the datasets we are using are very classic datasets, coming either from Keras or Kaggle.

Introduction

Neural network optimization is particularly critical. In Machine Learning, some models can be formulated to be reduced to objective functions that possess advantageous properties for optimization (e.g., convex and smooth functions). Furthermore, the number of parameters to optimize is often reasonable. In this context, the use of deterministic algorithms, either first or second order, allows convergence to the unique optimum in a finite amount of time. But when it comes to Deep Learning, one cannot avoid the very general case, with non-convex, non-smooth functions and an important number of parameters to optimize. Goodfellow et al. [2016] have pinpointed the main challenges of neural network optimization. Among these challenges, we can mention local minima, poor conditioning, saddle points and flat regions, cliffs and exploding gradients, and finally, inaccurate gradients. Most of neural network objective functions combine all these difficulties. Against this background, Stochastic Gradient Descent (SGD) is frequently employed to, on the one hand, prevent convergence towards a local optimum or a saddle point by introducing noise, and on the other hand, reduce computational complexity by computing an unbiased estimator of the gradient of the objective function over a sample or a mini-batch (mini-batch SGD). Nevertheless, many challenges still need to be addressed. That's why several algorithms have been proposed to improve SGD. First, algorithms that adapt their learning rate during training were created. The idea behind adaptive learning rate is to increase the learning rate for parameters with small gradients, and vice versa (Soydaner [2020]). Among these algorithms, one can cite AdaGrad and RMS Prop. More advanced algorithms combine adaptive rate algorithms with momentum based algorithms (momentum, Nesterov momentum) : Adam, AMS Grad, Nadam and Adamax are ones of them. The common point of all these algorithms is that they are exclusively first-order algorithms, and do not use second order derivatives. In this study, we have first recoded every algorithm presented, and then compared their performances in various contexts. First, we are interested in image classification on MNIST and Fashion MNIST datasets. Then we move on to a text classification task on IMBD dataset, and finally we test the algorithms for a regression task on the Boston Housing dataset. The code of these experiments is available at our Github repository.

Advanced Machine Learning Project (2023).

1 Stochastic Gradient Descent (SGD): the cornerstone of optimization in Deep Learning

We are operating within a supervised learning framework. Let $\mathcal{H}_\theta$ be a certain class of neural networks where $\theta \in \Theta$ contains the weights and biases of the different layers. Our goal is to learn a prediction function $h_\theta \in \mathcal{H}_\theta$ such that $h_\theta : \mathcal{X} \longrightarrow \mathcal{Y}$. Let $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$ be a loss function. The loss associated with observation i, for the function h_θ, is written as $\ell(y_i, h_\theta(x_i)) := f_i(\theta)$. The loss functions used in the experiment part are differentiable. By consequence, we assume that f_i functions or differentiable [1]. Denoting n as the number of training samples, the empirical risk of h_θ is expressed as $R_n(h_\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$. As a result, the empirical minimization problem can be written as:

$$\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

SGD is an algorithm that uses a single data function f_i at each iteration to minimize $R_n(h_\theta)$. Indeed, selecting uniformly at random a single index i from $[\![1, n]\!]$, $\nabla f_i(\theta)$ is an unbiased estimate of $R_n(h_\theta)$ in the sense that $\mathbb{E}_j[\nabla f_i(\theta)] = \nabla R_n(h_\theta)$. As a result, the update equations can be written as:

$$\theta_{t+1} = \theta_t - \alpha_{t+1} \nabla f_i(\theta_t)$$

Where α_t is a hyperparameter called *learning rate*. Compared to deterministic Gradient Descent (GD), the computational cost of an iteration is $\mathcal{O}(\dim(\theta))$ instead of $\mathcal{O}(n \times \dim(\theta))$. Nonetheless, by computing the gradient on a single observation, there is an increased risk of overfitting. Thus, a middle ground between GD and SGD is often favored: mini-batch SGD. Instead of computing the gradient for each sample, we partition the training set into subsets of size m, we compute and average the gradient over these m samples, at each step. The various mentioned algorithms have been used in their "mini-batch" version. Therefore, we present them in this form.

2 Adapting the learning rate to better suit the context of Deep Learning: variants of Stochastic Gradient Descent (SGD)

For many years, SGD has been the most popular algorithm for Deep Learning tasks. However, SGD encounters several issues. Notably, it is very sensitive to the learning rate, which has to be tuned manually. On the one hand, as mentioned earlier, the number of parameters to optimize in Deep Learning is very high. Each parameter may require the use of a different learning rate. On the other hand, it can be prudent to adjust the learning rate during iterations based on the gradient values. As a result, numerous algorithms with adaptive learning rate have been invented and have become interesting alternatives to SGD. The common idea behind all these algorithms is to increase the learning rate in areas where the gradient is low, and decrease the learning rate when the gradient is high. All the algorithms that are presented in this section are first-order algorithms. They are therefore computationally efficient in the context of Deep Learning with large scale datasets.

2.1 SGD variants with momentum

As in the deterministic case, it is possible to introduce a momentum β (Qian [1999]) in the stochastic gradient algorithm. β is a damping factor that smoothens the weight variations over iterations. This fraction of the previous update accelerates gradient descent towards non-oscillating directions. Stochastic Gradient Descent with Nesterov momentum (NAG, Nesterov [1983]) is a variant of standard momentum algorithm, in which the gradient is not computed at the current value of the parameter, but slightly ahead in the direction of the momentum. The update equations of NAG can be written as follows:

$$\begin{aligned}
\tilde{\theta} &= \theta_t - \alpha_{t+1} \beta m_t \\
\hat{g}_{t+1} &= \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\tilde{\theta}) \\
m_{t+1} &= \beta m_t + (1 - \beta) \hat{g}_{t+1} \\
\theta_{t+1} &= \theta_t - \alpha_{t+1} m_{t+1}
\end{aligned}$$

[1] Otherwise, the presented algorithms would remain valid by replacing the gradient of f_i with a subgradient.

Here, $\tilde{\theta}$ helps limit excessively large steps.

2.2 Adaptative learning rates methods

AdaGrad AdaGrad (Duchi et al. [2011]) individually adapts the learning rate of each parameter. Parameters with high gradients have a learning rate that decreases rapidly, while parameters with small gradients have a small decrease in their learning rate.

$$\begin{aligned}\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t)\\ u_{t+1} &= u_t + \hat{g}_{t+1}\odot\hat{g}_{t+1}\\ \theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{u_{t+1}+\varepsilon}}\hat{g}_{t+1}\end{aligned}$$

AdaGrad has two drawbacks. Firstly, the learning rate cannot increase over iterations; it can only decrease. Additionally, the entire history of past gradients is used to adapt the learning rate.

RMS Prop RMS Prop (Tieleman and Hinton [2012]) is a variant of AdaGrad where the squared gradient accumulation is replaced by an exponentially decaying average. This allows the learning rate to increase while in the AdaGrad algorithm, the learning rate can only decrease.

$$\begin{aligned}\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t)\\ u_{t+1} &= \gamma u_t + (1-\gamma)\hat{g}_{t+1}\odot\hat{g}_{t+1}\\ \theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{u_{t+1}+\varepsilon}}\hat{g}_{t+1}\end{aligned}$$

Intuitively, a small u_t leads to horizontal acceleration of the gradient, while a large u_t results in damping vertical oscillations.

2.3 Combining the best of each method: Adam and its variants

Adam Adam (Kingma and Ba [2014]) stands for Adaptive Moment estimation. It combines momentum algorithm and RMS Prop. Thus, Adam smoothens gradient oscillations while enabling the use of a fully adaptive learning rate.

$$\begin{aligned}\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t)\\ m_{t+1} &= \beta m_t + (1-\beta)\hat{g}_{t+1}\\ u_{t+1} &= \gamma u_t + (1-\gamma)\hat{g}_{t+1}\odot\hat{g}_{t+1}\\ \theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{u_{t+1}+\varepsilon}}m_{t+1}\end{aligned}$$

AMS Grad AMS Grad (Reddi et al. [2019]) is a combination of RMS Prop and Adam. The idea is to keep the advantages of Adam, while reinforcing the convergence by using non-increasing step sizes. To do this, u_t is still computed as an exponentially decaying average of the past squared gradient, but then the current step size is normalized by the maximum of the past and current u_t, instead of normalizing by u_t as it is done in Adam.

$$\begin{aligned}\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t)\\ m_{t+1} &= \rho_1 m_t + (1-\rho_1)\hat{g}_{t+1}\\ u_{t+1} &= \rho_2 u_t + (1-\rho_2)\hat{g}_{t+1}\odot\hat{g}_{t+1}\\ \tilde{u}_{t+1} &= \max(\tilde{u}_t, u_{t+1})\\ \theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{\tilde{u}_{t+1}+\varepsilon}}m_{t+1}\end{aligned}$$

Nadam The idea behind Nadam algorithm (Dozat [2016]) is to combine Adam with Nesterov's acceleration. Nesterov momentum is theoretically and often empirically more efficient than simple momentum, therefore it makes sense to include Nesterov momentum in Adam algorithm.

$$\begin{aligned}
\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t) \\
m_{t+1} &= \mu m_t + (1-\mu)\hat{g}_{t+1} \\
u_{t+1} &= \rho u_t + (1-\rho)\hat{g}_{t+1}\odot\hat{g}_{t+1} \\
\bar{m}_{t+1} &= (1-\mu)\hat{g}_t + \mu m_{t+1} \\
\theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{u_{t+1}+\varepsilon}}\bar{m}_{t+1}
\end{aligned}$$

Adamax In Adam, the update rule for individual weights is to scale their gradients inversely proportional to an L2 norm of their individual current and past gradients. Adamax (Kingma and Ba [2014]) is a variant of Adam, based on the infinity norm instead of the L2 norm.

$$\begin{aligned}
\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t) \\
m_{t+1} &= \beta_1 m_t + (1-\beta_1)\hat{g}_{t+1} \\
u_{t+1} &= \max(\beta_2 u_t, |\hat{g}_{t+1}| + \varepsilon) \\
\theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{(1-\beta_1^{t+1})u_{t+1}}m_{t+1}
\end{aligned}$$

Nostalgic Adam At first glance, the Nostalgic Adam algorithm may seem peculiar because it gives more weight to past gradients than to present ones. This algorithm was introduced by Huang et al. [2019]. The idea behind this algorithm is that traditional Adam-like algorithms lack of long-term memory. To remedy this, in the Nostalgic Adam algorithm, the parameter is not chosen as constant as in Adam, but is allowed to vary so as to give more weight to past gradients.

$$\begin{aligned}
\hat{g}_{t+1} &= \frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\theta_t) \\
\beta_{2,t+1} &= \frac{B_t}{B_{t+1}} \\
m_{t+1} &= \beta_1 m_t + (1-\beta_1)\hat{g}_{t+1} \\
u_{t+1} &= \beta_{2,t+1}u_t + (1-\beta_{2,t+1})\hat{g}_{t+1}^2 \\
\theta_{t+1} &= \theta_t - \frac{\alpha_{t+1}}{\sqrt{u_{t+1}+\epsilon}}m_{t+1}
\end{aligned}$$

Where $B_t = \sum_{k=1}^{t} b_k \quad \text{for} \quad t \geq 1, \quad b_k \geq 0, \quad \text{and} \quad B_0 = 0.$

3 Experiments

We are now going to compare the algorithms presented in section 2 in various cases. Firstly, the algorithms are tested in the case of image classification using a convolutional neural network. Next, we look at text classification using a Multilayer Perceptron. Finally, we compare these algorithms in the case of a regression, using again a Multilayer Perceptron.

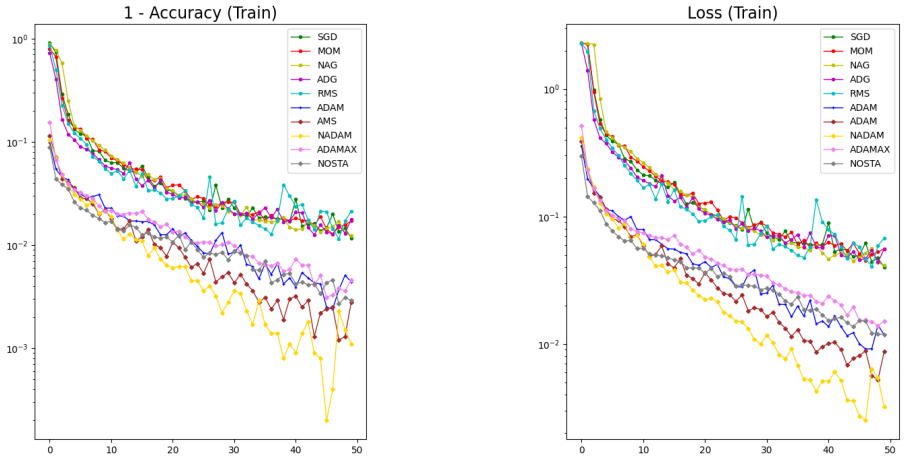
3.1 Images Classification with a Convolutional Neural Network

We are first going to compare the algorithms on image classification task with a convolutional neural network. The CNN we have implemented for this purpose is the well-known LeNet-5 architecture (LeCun et al. [1998]), which consists in two Conv2d layers with kernel 5×5 with average pooling, followed by three fully-connected layers. Because we do multi-class categorization, we use the categorical cross-entropy loss. We train the model on 50 epochs, with a batch size of 128.

MNIST Dataset. First, we compare the performances of algorithms on the basic MNIST Dataset. The dataset contains images of handwritten digits. Thus, $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The training set contains 60,000 images, whereas the test set contains 10,000 images. Both subsamples are balanced: no digit class is overrepresented compared to another.

Figure 1 shows the performances of the algorithms during training. We present the evolution of the accuracy on the MNIST train dataset, as well as the value of the loss at each epoch. For better visualization, we plot 1 - accuracy.

We can see that all the algorithms reach more than 97% accuracy on the training set after 50 epochs. The performance of the algorithms can be divided into two categories. Algorithms combining momentum and adaptive step (Adam, Nadam, Adamax, AMS Grad and Nostalgic Adam) stand out and perform better than the others. First, the loss (respectively, accuracy) associated with these algorithms is always lower (respectively, higher), regardless of the epoch considered. More precisely, these algorithms achieve an accuracy on the training set of between 99% and 99.9% after 50 epochs. On the other hand, the other more basic algorithms, either with an adaptive step (AdaGrad, RMSProp) or with momentum (NAG, Momentum) perform a little less well, achieving an accuracy on the training set of between 97% and 99% after 50 epochs. Table 1 gives the loss and the accuracy on the test set. One can observe that in terms of generalization, the best performance belongs to AMS Grad, with a test loss of 0.06 and an accuracy of 98.06%, followed by ADAMAX.



(a) 1-Accuracy (Training set)

(b) Loss (Training set)

Figure 1: Evolution of loss and accuracy on the training set during training - MNIST (Log scale)

	Algorithm	Loss (CE)	Accuracy
0	SGD	0.083920	0.9740
1	MOM	0.083247	0.9735
2	NAG	0.087975	0.9731
3	ADG	0.082869	0.9761
4	RMS	0.088023	0.9744
5	ADAM	0.112146	0.9761
6	AMS	**0.059157**	**0.9846**
7	NADAM	0.103317	0.9805
8	ADAMAX	0.077648	0.9807
9	NOSTA	0.089678	0.9763

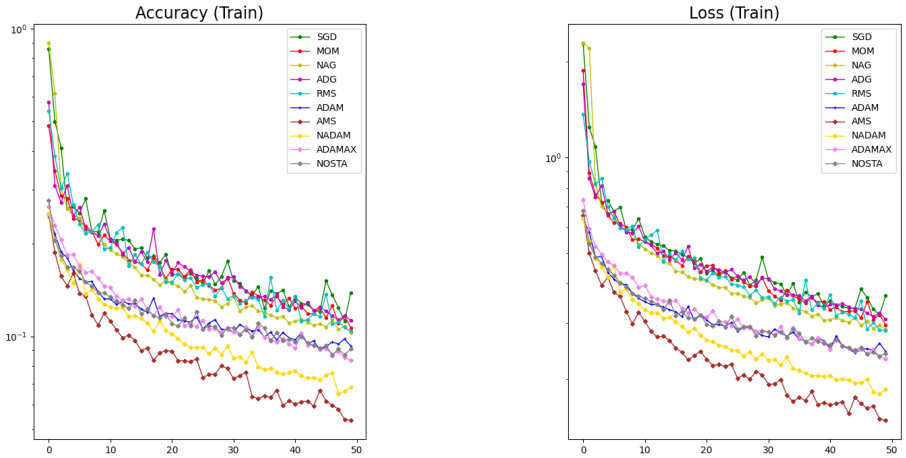
Table 1: Categorical cross entropy and Accuracy on the Test set (MNIST)

Fashion MNIST Dataset. We now turn our attention to image classification on the Fashion MNIST dataset, which is supposed to be slightly more complicated to classify than MNIST. The dataset

contains images of Zalando articles in grayscale, distributed across ten categories. The training set consists of 60,000 images, while the test set contains 10,000. The data is perfectly balanced.

We want to see if the optimization algorithms stand out more from each other. Figure 2 shows the evolution of accuracy and loss during training on the Fashion MNIST train dataset. Firstly, we can observe that indeed, classification on Fashion MNIST dataset seems less easy than classification on basic MNIST: all optimization algorithms perform less well than before. We can see that throughout training, the algorithms that perform the best are, again, Adam-like algorithms: Adam, Nadam, Adamax, AMS Grad and Nostalgic Adam. These algorithms reach an accuracy around 90% after 50 epochs on the training set, while other algorithms remain between 86% and 90% of accuracy. The categorical cross-entropy is also lower for algorithms combining momentum and adaptive step.

In terms of generalization (Table 2), Adamax provides the smallest test loss, while AMS Grad yields the highest accuracy.



(a) 1-Accuracy (Training set)

(b) Loss (Training set)

Figure 2: Evolution of loss and accuracy on the training set during training - Fashion MNIST (Log scale)

	Algorithm	Loss (CE)	Accuracy
0	SGD	0.407096	0.8567
1	MOM	0.395932	0.8657
2	NAG	0.407218	0.8631
3	ADG	0.412707	0.8557
4	RMS	0.407609	0.8615
5	ADAM	0.407077	0.8613
6	AMS	0.426442	**0.8823**
7	NADAM	0.389654	0.8809
8	ADAMAX	**0.366803**	0.8775
9	NOSTA	0.379346	0.8731

Table 2: Categorical cross entropy and Accuracy on the Test set (Fashion MNIST)

3.2 Text Classification with a Multilayer Perceptron

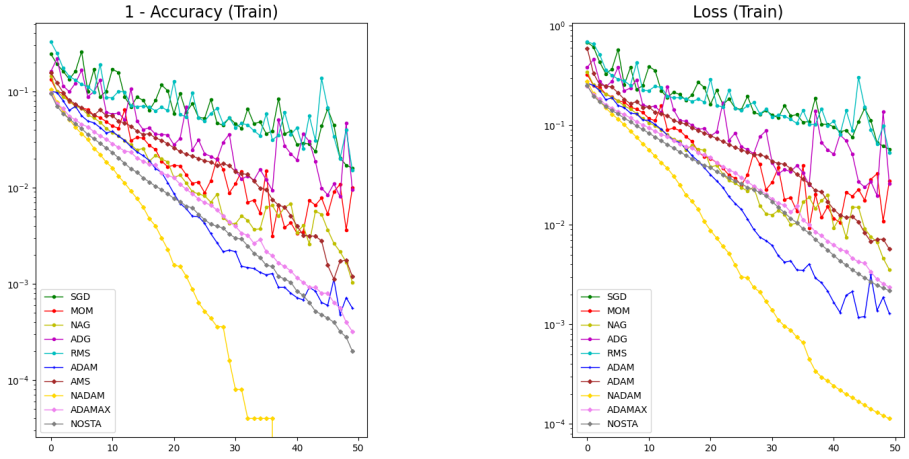
We now turn our attention to a binary classification example of movie reviews. The objective is to classify movie reviews as positive or negative, based on the content of the reviews. To do this, we use the IMBD dataset, available with `keras`.

IMBD Dataset. This dataset is composed of 50,000 reviews (very polarized) from the Internet Movie Database. We split the database into a training set of 25,000 reviews, and a test set of 25,000

reviews. For learning, we only keep the 10,000 most frequent words in the training set. To perform this classification, we implement a Multilayer perceptron, as presented by Chollet [2021]. Of course, other models could be considered as more adapted for text classification, and would get a better accuracy, but still, as we will see, this model provides rather good scores in spite of its simplicity. Because we do binary classification, we use the binary cross-entropy loss.

Figure 3 shows the evolution of the loss function and accuracy on the training set during training. Adam and variants algorithms provide the best performances in terms of loss and accuracy. In particular, the loss associated with Nadam and Adam is very close to 0 after 50 epochs. On the other hand, SGD and RMS Prop algorithms provide the worst results. We notice that for these algorithms, the loss (resp. accuracy) doesn't decrease (resp. increase) significantly through the epochs. Finally, after 30 epochs, the performance of AdaGrad and Momentum deteriorates.

Nonetheless, table 3 show that SGD yields the best results in terms of binary cross-entropy on the test set, whereas AMS Grad produces the best results in terms of accuracy.



(a) 1-Accuracy (Training set)

(b) Loss (Training set)

Figure 3: Evolution of loss and accuracy on the training set during training - IMBD (Log scale)

	Algorithm	Loss (BCE)	Accuracy
0	SGD	**0.290162**	0.88136
1	MOM	0.297365	0.87688
2	NAG	0.344889	0.87188
3	ADG	0.295658	0.88004
4	RMS	0.296404	0.88228
5	ADAM	0.404264	0.87188
6	AMS	0.360922	**0.88672**
7	NADAM	0.338802	0.87364
8	ADAMAX	0.329790	0.87632
9	NOSTA	0.322461	0.87744

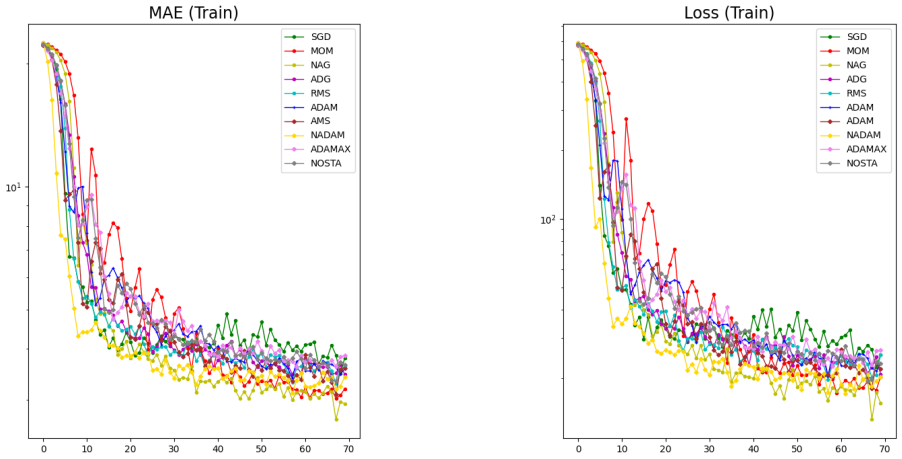
Table 3: Binary cross entropy and Accuracy on the Test set (IMBD dataset)

3.3 Regression with a Multilayer Perceptron

All the situations studied previously were classification problems. We now turn to a regression problem. We will test the algorithms on the Boston Housing Prices dataset, which is a well-known dataset for regression framework. On this dataset, the aim is to predict the median home price in the suburbs of Boston, given 13 variables. The dataset is fairly small: the training set has 404 data points, and the test set has 102 data points. Given the small size of our sample, we will train a multi-layer perceptron of moderate complexity, with two hidden layers of 64 neurons, with ReLU

activation functions. The loss used for training is the mean-squared error (MSE). We also compute the mean absolute error (MAE) to evaluate the model. Figure 4 shows the evolution of the loss and MAE during training. We can see that it is less obvious to classify the performance of the algorithms compared with the previous cases. For the first 20 epochs, Nadam is the fastest, but is then overtaken by Momentum and NAG. Momentum-based algorithms (Momentum and NAG) perform particularly well. They managed to achieve a MAE close to 3, while the other algorithms stagnated at a MAE of 4. This may be due to the fact that this tabular dataset contains less structured data, and therefore the objective function is noisier than before, which makes momentum algorithms without adaptive steps particularly well suited.

On the test set, 4 show that the best performance in terms of MSE is achieved by AMS Grad algorithm, with a score almost equal to 23.9. Nadam provides the best result in terms of MAE.



(a) 1-Accuracy (Training set)

(b) Loss (Training set)

Figure 4: Evolution of loss and accuracy on the training set during training - Boston Housing Prices (Log scale)

	Algorithm	Loss (MSE)	MAE
0	SGD	32.289402	3.9726312
1	MOM	26.128288	3.643694
2	NAG	31.632784	3.4460154
3	ADG	28.679266	3.870798
4	RMS	25.428267	3.9103708
5	ADAM	25.027214	3.6892462
6	AMS	**23.916958**	3.6705124
7	NADAM	29.326841	**3.4384363**
8	ADAMAX	33.78123	3.9228911
9	NOSTA	25.096832	3.8885157

Table 4: MSE and MAE on the Test set (Boston Housing Prices dataset)

Conclusion

The field of stochastic optimization is very wide and important for Deep Learning. Experimenting some major algorithms allowed us to have a concrete approach of this field. We observed that variants of the Adam algorithm provided the best results on the training set, especially for tasks like image or text classification. In these problems, the number of parameters to optimize is much larger than in the regression problem we addressed. Thus, algorithms combining both adaptive learning rate and momentum seem more suitable. Finally, we noted that a ranking in terms of performance is often different on the test set compared to the training set, due to the randomness in the data. Implementing

a cross-validation procedure would allow for a more robust comparison of algorithms, albeit at the expense of high computational cost.

References

F. Chollet. *Deep Learning with Python*. 2021.

T. Dozat. Incorporating nesterov momentum into adam. In *Proceedings of 4th International Conference on Learning Representations, Workshop Track*, 2016.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

H. Huang, C. Wang, and B. Dong. Nostalgic adam: Weighting more of the past gradients when designing the adaptive learning rate. 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1): 145–151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL https://www.sciencedirect.com/science/article/pii/S0893608098001166.

S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

D. Soydaner. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence*, 2020.

T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop. Technical report, COURSERA: Neural Networks for Machine Learning, 2012.