On the use of derivative-free optimization methods for the minimization of empirical risk, with application to the use of neural network for database classification

(Final report)

MOUTONNET Adam* and DIOUANE Youssef‡
*ISAE-SUPAERO, Université de Toulouse, 31055 Toulouse, FRANCE
Email: adam.moutonnet@student.isae-supaero.fr
‡ISAE-SUPAERO, Université de Toulouse, 31055 Toulouse, FRANCE
Email: youssef.diouane@isae-supaero.fr

***Abstract*—When scientists come to use artificial intelligence for database classification, then often meet the classical problem of teaching the artificial intelligence how to analyze well the data. Basically, the learning phase is an optimization problem, and this problem is always solved thanks to derivatives. But they may lead to convergence issues, leading the learning to fail, and are sometimes hard to settle. The study of derivative-free optimization (DFO) methods represents a hope for scientists to avoid convergence issues and minimize empirical risks faster for small-dimension problems. In other words, it might allow scientists to be able to better reduce the risk for the artificial intelligence to be wrong on its predictions.**

I. Introduction

In a world where data are gathered on everything, the number and variety of databases is huge, and will continue to grow in the future. In order for these databases to be usable, they need to be classified. What this means is that scientists need to assign a label to each kind of data (or class) present in the database. For example, take a database full of furniture images. Pictures of tables will be assigned the label "table". A more relevant example comes with databases of bank transactions, from which we need to determine the fraudulent ones. Even if it may take time, we can imagine that the furniture database will be easy to classify for a human. But for the second example, as bank transactions may have many features to analyze in order to determine whether they are fraudulent or not, a more efficient tool than a human will be needed.

The use of artificial intelligence represents a powerful tool for data engineers because it is far more efficient than humans to analyze features of a datum. Take the bank transaction database, it would find the fraudulent one. But to achieve something that seems so beautiful, there are always drawbacks. Here it is the learning phase through which every artificial intelligence must pass before being used. In fact, an artificial intelligence needs to learn how to recognize the features, and this may be a very complex thing to do. To learn that a table will have four high legs seems far easier than to learn what makes a bank transaction fraudulent. Moreover, when the artificial intelligence manages to predict very well the labels for data on which it is trained but do not for any new data of the same type it has never analyzed before, we encounter over-fitting. In order to bring it to light, scientists always consider a training database, on which the learning is done, and a test database, on which the learning is tested. Over-fitting can be seen when the performances on the training database are good, but those on the test database are poor.

And that is when the derivatives come. In fact, as we saw in the bibliographic report, the learning phase is just a complex optimization problem, and the function to optimize will be a loss function representing the gap between what the artificial intelligence believe the class is, and what the class really is. Concretely, an artificial intelligence is a neural network that take as an argument the vector of features $x_n \in \chi$, and where each neuron of a layer is linked to an other one of the next layer with weights, by simple mathematical operations, conducting to the last layer where the value of the neurons can be interpreted as the label $\hat{t}(x_n, w)$ predicted by the neural network. Fig. 1 summarizes the concept.

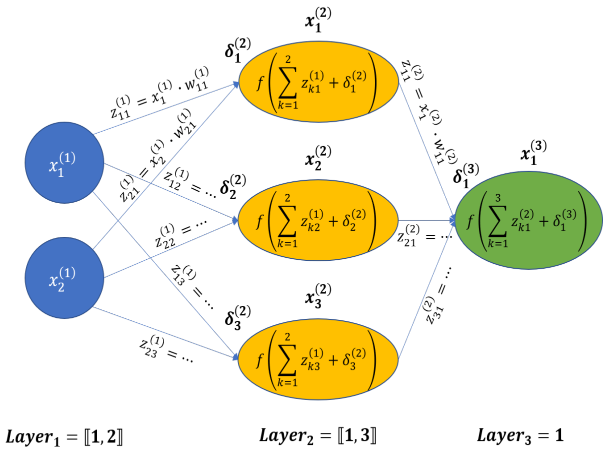
$x_1^{(2)}$
$\delta_1^{(2)}$
$f\left(\sum_{k=1}^{2} z_{k1}^{(1)} + \delta_1^{(2)}\right)$
$z_{11}^{(1)} = x_1^{(1)} \cdot w_{11}^{(1)}$
$z_{21}^{(1)} = x_2^{(1)} \cdot w_{21}^{(1)}$
$z_{11}^{(2)} = x_1^{(2)} \cdot w_{11}^{(2)}$
$x_1^{(1)}$
$x_2^{(2)}$
$z_{12}^{(1)} = \dots$
$\delta_2^{(2)}$
$f\left(\sum_{k=1}^{2} z_{k2}^{(1)} + \delta_2^{(2)}\right)$
$x_1^{(3)}$
$\delta_1^{(3)}$
$f\left(\sum_{k=1}^{3} z_{k1}^{(2)} + \delta_1^{(3)}\right)$
$z_{22}^{(1)} = \dots$
$z_{21}^{(2)} = \dots$
$x_2^{(1)}$
$z_{13}^{(1)} = \dots$
$z_{31}^{(2)} = \dots$
$x_3^{(2)}$
$\delta_3^{(2)}$
$f\left(\sum_{k=1}^{2} z_{k3}^{(1)} + \delta_3^{(2)}\right)$
$z_{23}^{(1)} = \dots$
$Layer_1 = [\![1,2]\!]$
$Layer_2 = [\![1,3]\!]$
$Layer_3 = 1$

Fig. 1. Illustration of a neural network.

The summary of this optimization problem can be formulated with the following statement:

$$\min_w L_{\mathcal{D}}(w) = \frac{1}{N} \sum_{n=1}^{N} l(t_n, \hat{t}(x_n, w)) \tag{1}$$

where $L_{\mathcal{D}}(w)$ is the empirical risk on the training database, N the number of data, $x_n \in \chi$ the vector of features, w the vector of weights, $\hat{t}$ the prediction function which will assign a predicted label to x depending on the weights, $t_n \in \tau$ the real label associated to x_n, and l the function evaluating the difference between the prediction $\hat{t}(x_n, w)$ and the real label t_n [7]. We also define $\widehat{L_{\mathcal{D}}}(w)$ which is the empirical risk on the test data set. A perfect learning without over-fitting will minimize $\widehat{L_{\mathcal{D}}}(w)$ as much as $L_{\mathcal{D}}(w)$. We easily see that the weights w will be the parameters of the neural network to optimize.

We also saw in the bibliographic report the fact that we minimize the function using batches. During a numerical iteration, we will consider an approached function of $L_{\mathcal{D}}(w)$

$$\widetilde{L_{\mathcal{D}}}(w) = \frac{1}{|\mathcal{S}_k|} \sum_{x_n \in \mathcal{S}_k} l(t_n, \hat{t}(x_n, w)) \tag{2}$$

where $\mathcal{S}_k$ is a subset called "batch" of χ. We will then realize a single update of w minimizing this approached function. The number of epochs will be the number of total passes through the whole set χ.

Commonly, this update is a single gradient-descent realized thanks to an optimization algorithm based on gradient methods. The main challenges of the learning phase are the efficiency, the convergence, the over-fitting [7] and the time allowed, and we saw in the bibliographic report that the two first challenges are sometimes hard to take up for gradient-based methods. Thus we thought of an alternative to gradient-based methods hoping that they would solve the problems often encountered when it comes to train a neural network.

During this research, we would like to study DFO algorithm, and more precisely two of them, a deterministic one and a stochastic one. We will apply these two algorithm to the learning problem of simple neural networks, dealing with some easy data sets, and some harder ones. First, in Sec. II, we will describe the different problems (databases), show the associated loss function, its particularities and the influence of the batch size on this function. Then we will go into details with the derivative-free optimizers used, their particularities and their tunable parameters in Sec. III. Sec. IV will compare them on some problems, and bring to light the superiority of a particular one, based on its speed, its accuracy, its efficiency, its capacity to analyze widely the problem and its sensitivity to the batch size. Eventually, in Sec. V, we will compare the winner of this battle with the commonly used gradient-based methods ADAM [6], and show where the DFO algorithm is better, and where the gradient-based method keeps its superiority.

II. Description of the problems

The first database we will have to classify will be a very simple binary classification problem, with two different clouds of point called "blops". We will distinguish the two blops assigning them colors, one will be blue, and one will be red. Each point of the clouds will have a 2D feature vector (its coordinate on the 2D plane), and we expect the neural network to classify every points of the 2D plane, assigning them to either the red blop or the blue blop. A visual representation can be seen on Fig. 2.

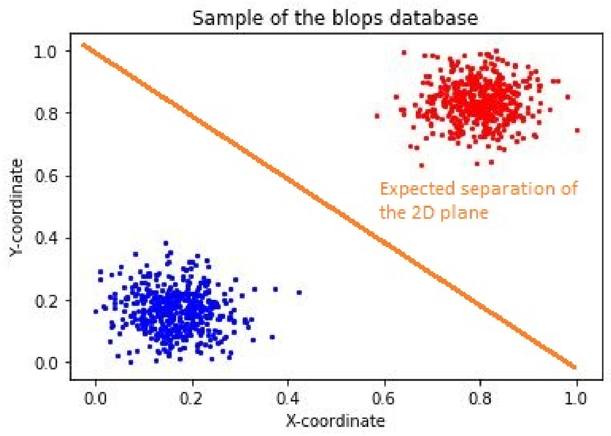

Fig. 2. Illustration of the BLOPS database.

Now we have an idea of the database, let us now deal with the particularities of the associated loss function. The problem will be of dimension 151 (we will have 151 weights to optimize), thus we will visualize the function only on two directions d_i and d_j of the orthonormal basis. Starting from a point w_0 (which is a initialization choice of the 151 weights), we will plot the following function:

$$g_{w_0,d_i,d_j} : (\alpha, \beta) \mapsto \widehat{L_{\mathcal{D}}}(w + \alpha d_i + \beta d_j) \tag{3}$$

and visualize its contour. For the blops database, it gives the contours on Fig. 3. We can easily see that the function will be composed of plateaux (on the left of the figure) on which the value of the loss function does not change much. Apart from that, this function is not multi-modal and the variations are not so abrupt. Of course, we suppose that the function has this comportment in every direction, and on every part of the space.

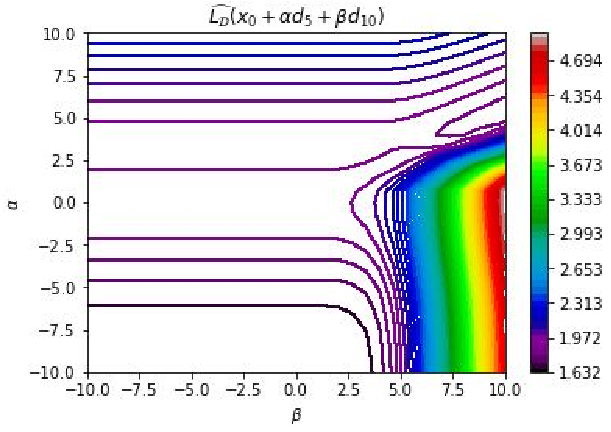

Fig. 3. Illustration of the loss function for BLOPS database.

We will also use two other similar databases, with moons clouds, and concentric circle ones, as shown on Fig. 4. Their loss function have more or less the same comportment than for the BLOPS database, with the apparition of valleys, in which the algorithm can be stuck, as shown on Fig. 5.

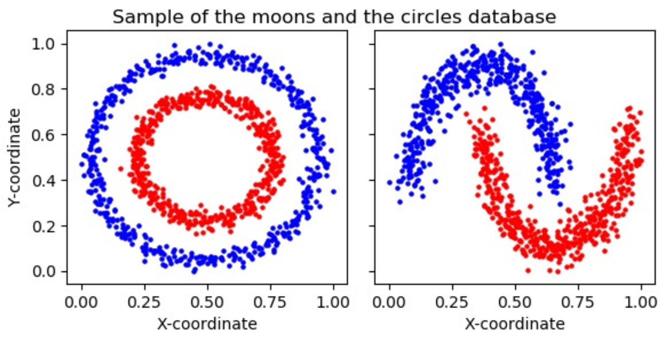

Fig. 4. Illustration of the MOONS (right) and CIRCLES (left) database.

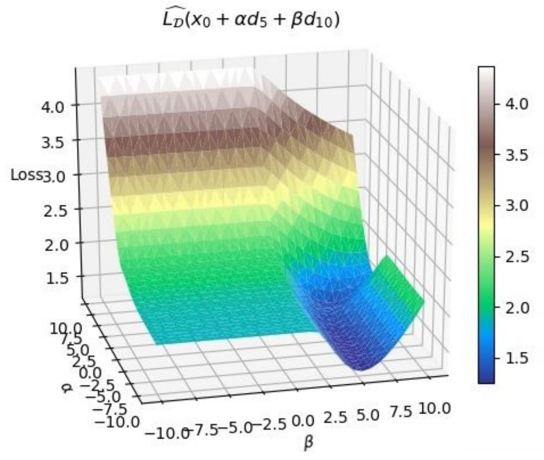

Fig. 5. Illustration of the loss function for CIRCLES database.

As harder databases now, we will use the binary MNIST database, which is composed of 15x15 black and white images of written digits, 0 and 1, and the EYES database, which composed of 24x24 black and white images of open and closed eyes. A sample of the two databases can be seen on Fig. 6.

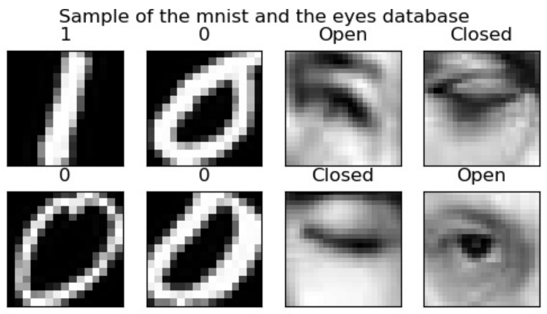

Fig. 6. Illustration of the binary MNIST (left) and the EYES (right) databases.

For the MNIST one, the purpose of the algorithm will be to determine whether it is a 1 or a 0 written. The vector of features will be the value of each pixel on a gray scale from 0 to 255, but normalized (brought into $[0,1]$, 0 is 0 and 1 is 255). The size of the feature vector will be 225, and the size of the problem (number of weights) will be 111. Regarding its associated loss function, there are also plateaux, but no more valleys, it is more about local minima, as shown on Fig. 7. Here, the hard challenge is that the high plateau is separated from the minimum by a crete, which can lead to convergence issues.

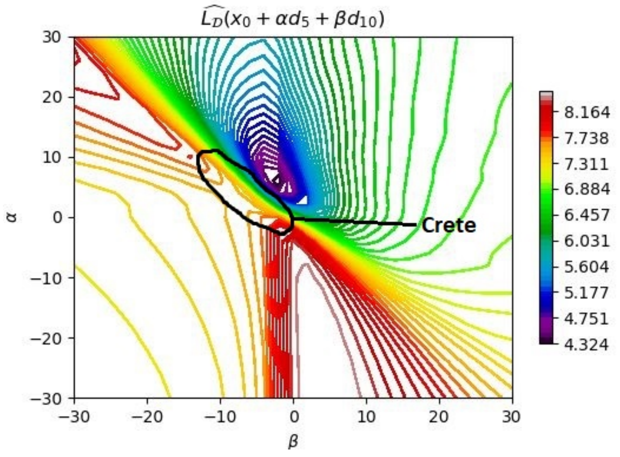

Fig. 7. Illustration of the loss function for the binary MNIST database.

For the EYES database, which is probably the hardest, the purpose of the algorithm will be to determine whether the eye is closed or not analyzing the image, the same way as for MNIST. The size of the feature vector will be 576, and the size of the problem (number of weights) will be 663. What is hard with this database is that sometimes the loss function is a lot multi-modal, as shown on Fig. 8, with a lot of little plateaux where algorithm may be stuck, and abrupt slopes.

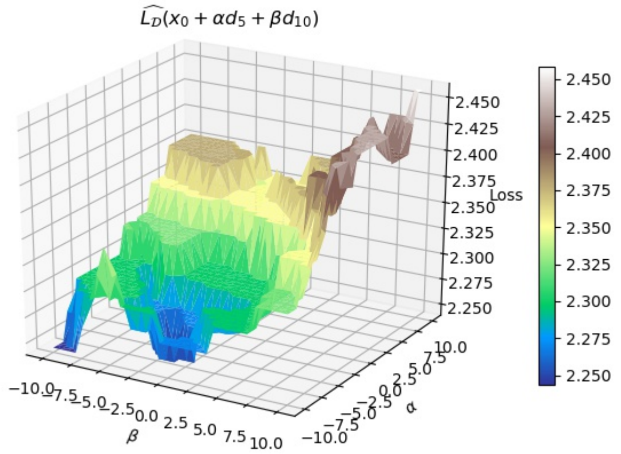

Fig. 8. Illustration of the loss function for the binary EYES database.

Finally, I would like to highlight the influence of the batch size on the loss function, because as we will minimize $\widetilde{L_{\mathcal{D}}}(w)$ and not $L_{\mathcal{D}}(w)$, this may represent an issue. Actually, it does represent an issue. On Fig. 9 we can easily see that the loss function with a large batch is a lot more smooth than the functions with a reduced batch. It leads to the apparition of more local minima, into which the algorithm will fall without any possibility of making it to a better minimum. Plus, the "best" minimum (as it may certainly not be the global one) of the function is not the same when the batch size change. That is why we will have to find a good compromise for the batch-size, because a gigantic one will cause the algorithm to be slower, and a small one will cause problems of convergence.

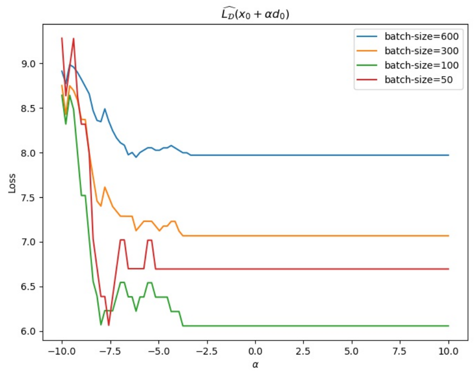

Fig. 9. Illustration of the batch influence on the loss function for the EYES database, on d_5.

III. Description of the DFO algorithms

Now we have analyzed our problems, we are going to focus on the algorithms used to solve them [2]. The first one will be a quasi-deterministic (QD) one and is described precisely on Alg. 1 [1]. Quasi-deterministic, because the set of directions used to create new polling points is created according to a normal distribution. We will establish some parameters, like $\beta = 0.9$, $\gamma = 1.1$, that will remain unchanged, and we will consider the following tunable parameters : α_0, x_0, $|\mathcal{S}|$ and $|\mathcal{D}|$. An idea of what this algorithm is really doing can be seen on Fig. 10.

The second algorithm will be a stochastic genetic one, with evolution strategy (ES), and is described precisely on Alg. 2 [5]. For this algorithm, we will consider the following tunable parameters : σ_0, x_0, λ, $|\mathcal{S}|$. The Epoch Initialization will be $\sigma = \sigma_0$ for the EYES database only, and $\sigma = \frac{\sigma_0}{2}$ for the others. An idea of what this algorithm is really doing can be seen on Fig. 11. Note that this algorithm is not globally convergent, meaning that it does not assure us that it will find a local minimum. It may be possible to make it globally convergent, as described in [3] and [4], but it reduces its capacity to explore the space, as we will see later on.

The visual representations show us clearly what are the purpose of these two algorithms, they aim to generate a new point at each iteration such that the value of the function at this point is less than the value at the precedent point. Now that we have a better idea of what these algorithms are doing, we can proceed to the comparison on a learning problem.

Algorithm 1: **A Quasi-Deterministic Method**

```
**Main Initialization:** Choose a initial vectors of weights $x_0$, an initial learning rate $\alpha_0$, $\gamma$ and $\beta$ st. $0 < \beta < 1 < \gamma$, a batch-size $|\mathcal{S}|$, and a size $|\mathcal{D}|$ of the future random set of direction, set $k$ to 0.
for $epoch = 0, 1, \ldots$ do
    **Epoch Initialization:** Multiply the learning rate $\alpha_k$ by $(\gamma + \beta)$.
    for $batch = 0, \ldots, \frac{N}{|\mathcal{S}|}$ do
        **1. Batch Step:** Randomly choose a batch $\mathcal{S}_k$ of size $|\mathcal{S}|$ within the training database.
        **2. Set point Step:** Following a normal distribution, choose a random set of directions $\mathcal{D}_k$ of size $|\mathcal{D}|$ st. for each direction belonging to it, there is also its opposite direction. If there is a $lastdir$ stored, add it at the beginning of the set.
        **3. Poll Step:** Go through $\mathcal{D}_k$. As soon as there is a $d_{ki}$ in $\mathcal{D}_k$ such that the poll point $w_k + \alpha_k d_{ki}$ leads to $f(x_k + \alpha_k d_{ki}) < f(x_k)$, then set $x_{k+1} = x_k + \alpha_k d_{ki}$, declare the iteration as successful, store the current direction as $lastdir$, and go to the next step. Otherwise, if there is no $d_{ki}$ satisfying that criterion, declare it unsuccessful, erase $lastdir$, and go to the next step.
        **4. Update the step size parameter:** If the iteration is successful, then set $\alpha_{k+1} = \gamma\alpha_k$.Otherwise, set $\alpha_{k+1} = \beta\alpha_k$. Add 1 to $k$ and go to the next batch.
    end
end
```

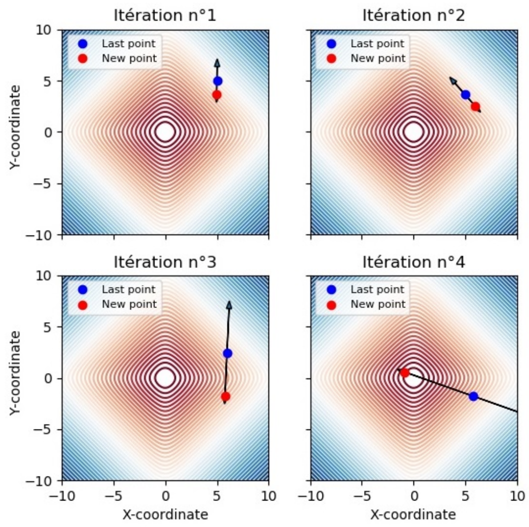

Fig. 10. Illustration of the deterministic algorithm in action.

IV. Comparison of the DFO algorithms

In this section, we will compare the two DFO algorithms, seeing which algorithm do the better with various batch sizes $|\mathcal{S}|$ and various initial set of weights x_0.

In order to measure the performance of these algorithm, we will plot the evolution of the loss function per epoch. The more the loss function will reduce with the less epoch processed, the better will be the algorithm. Plus, we will also consider the accuracy, which is the percentage of successful prediction over all predictions done on a batch. Finally, we will check that there is no over-fitting checking the predictions done on the test database, otherwise, the algorithm will be of a poor utility for a learning problem. Concretely, we want the neural network to classify as better as possible the database.

Algorithm 2: A Evolution Strategy Method

```
Main Initialization: Choose a initial vectors of weights $x_0$, an initial learning rate $\sigma_0$, a population size $\lambda$ and infer from this number other parameters of the algorithm ($\mu$,weights for recombination,...), choose a batch-size $|\mathcal{S}|$, set $k$ to 0.
for $epoch = 0, 1, \dots$ do
    Epoch Initialization: After each epoch, reinitialize the learning rate $\sigma_k$ to $\frac{\sigma_0}{2}$ or $\sigma_0$ depending on the need to explore the space. The first choice will allow the algorithm to be more precise, the second to explore better the surrounding environment.
    for $batch = 0, \dots, \frac{N}{|\mathcal{S}|}$ do
        1. Offspring Generation: Compute new sample points $Y_{k+1} = \{y^1_{k+1}, \dots, y^\lambda_{k+1}\}$ around $x_k$ under the given distribution represented by $C_k$.
        2. Selection: Reorder the $Y_{k+1} = [\tilde{y}^i_{k+1}]_{i=1,\dots,\lambda}$ such that $f(\tilde{y}^1_{k+1}) \leq \dots \leq f(\tilde{y}^\lambda_{k+1})$ to keep only the first $\mu$ elements of the reordered $Y_{k+1}$.
        3. Recombination: Process $x_{k+1} = \sum_{i=1}^{\mu} w^i_{k+1}\tilde{y}^i_{k+1}$ which is the weighted mean of the kept elements, with $[w^i_k]_{i=1,\dots,\mu}$ weights that are computed before.
        4. Adaptation: Process $C_{k+1}$ from $C_k$ depending on the new progress of the problem, the same for $\sigma_{k+1}$.
    end
end
```

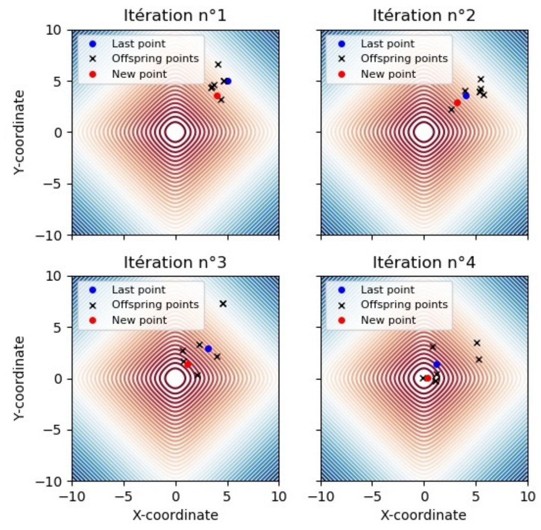

Fig. 11. Illustration of the ES algorithm in action.

An example of a good classified database can be seen on Fig. 12

But in order for the comparison to be accurate, we need to put both algorithms on the same bases, that is why we will try to choose the best size of the point set generated (and evaluated) at each iteration (λ for the ES algorithm, $|\mathcal{D}|$ for the QD one) and the best initial learning rate (α and σ) that works well for both. They will then produce their best performances, sharing some parameters, allowing us to conclude which is the best when we will compare them on the batch size criterion and on the different starting weights sets.

About the learning rate, the experiment will be done on the circles database. The batch size used will be 1000 and we will choose a starting weights set not far from zero, drawing values

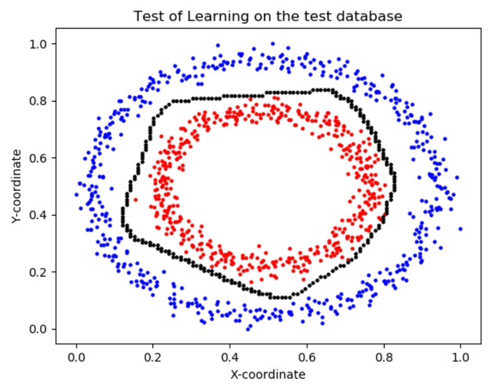

Fig. 12. Illustration of what a good learning aims to do on the circle database. The separation frontier is in black. We can see that the neural network separated well the outer points from the inner ones. The performances of this learning correspond to a loss value of 10^{-7} and an accuracy of 100%

according to a normal distribution of mean 0 and scale 1. For the points set size, we will use for the moment the following formula given in [5] $\lambda = 4 + 3 * \log(size of the problem)$ that applies for the ES algorithm. The same value will be chosen for $|\mathcal{D}|$. On Fig. 13, various evolution of the loss function and the accuracy for botch algorithms have been plotted for different values of the initial learning rate.

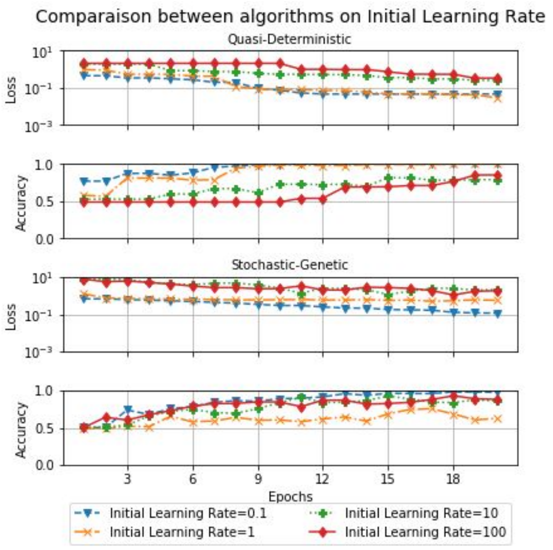

Fig. 13. Evolution of the loss function and the accuracy on 20 epochs for various initial learning rates on the CIRCLES database.

For big initial learning rates such as 10 and 100, both algorithms took a lot of epoch to start the decrease of the loss function, and even for the following epochs, the decrease is not that fast. What is surprising is that even if the deterministic algorithm reaches lower values of loss function, the accuracy it produces stay inferior to the accuracy produced by the genetic algorithm. For smaller initial learning rates now, the decrease of the loss function is higher for both algorithms, and the deterministic one do better than the genetic one. The accuracy is far better, except for the genetic algorithm with an initial learning rate of 1. The conclusion that can be drawn is that smaller learning rates work better (for this problem, and also for the BLOPS, MOONS and MNIST database) and that the accuracy may not be representative of the loss function. Hence, we will continue with an initial learning rate α_0 and σ_0 of 0.1.

Carrying on with a batch size of 1000 and an initial weights set close to zero, the parameter at stake now is the size of the points set. On Fig. 14, various evolutions of the loss function and the accuracy for both algorithms have been plotted for different values of the points set size, working on the MOONS database.

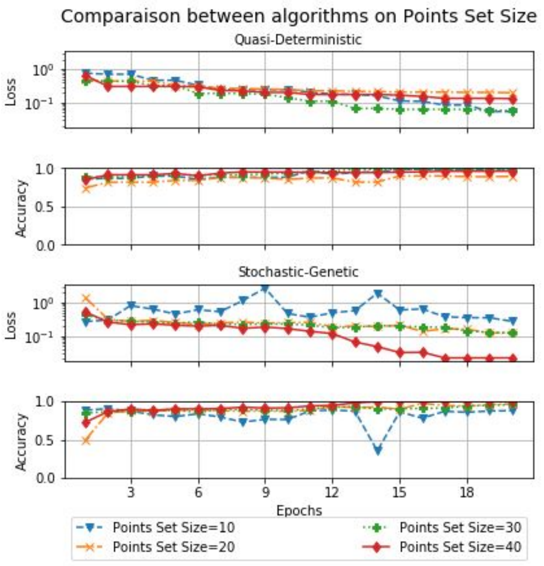

Fig. 14. Evolution of the loss function and the accuracy on 20 epochs for various points set sizes on the MOONS database.

Oddly, the ES algorithm is more sensitive to the size of points set generated at each iteration than the deterministic one. Let us not forget that the randomness may play a role in this paradox. In any case, the genetic algorithm copes better with greater points set size. For the deterministic one, we will conclude that the points set size is not really important, but logically we will better take a great one than a not sufficient one. Moreover, I would like to highlight that when this size grows, the time of computation becomes huge, and that is why from this point we will limited ourselves to a points set size of 20 for what lies ahead.

Now we have a solid common basis to compare them, both algorithm will pass the batch size test. Starting from the same set of weights once again and using the moons database, we will make the value of the batch size vary from 250 to 1000. The results can be seen on Fig. 15.

Fig. 15 clearly shows that the QD algorithm is not sensitive to the batch size, its performance does not vary much with it. Overall, the minimization of the loss function is good and the accuracy too. Regarding the ES algorithm, its performances are far better, but it is very sensitive to the batch size, as the case with the batch size 500 shows (although the accuracy

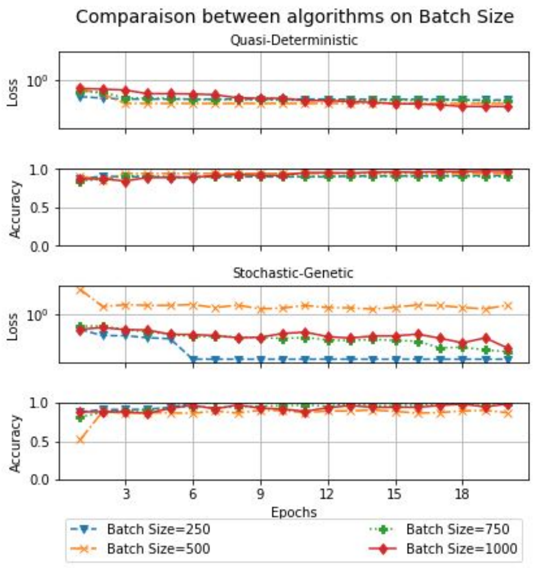

Fig. 15. Evolution of the loss function and the accuracy on 20 epochs for various batch sizes on the MOONS database.

stays good). Naturally, as these algorithms are stochastic, this result may vary from a test to an other, but this graph is representative of what both algorithms do generally.

Before drawing any conclusion, we will see the influence of the initial set of weights. For this purpose, we will draw values according to a normal distribution with a random mean between -5 and 5, and a random scale between 0 and 10. We will draw four sets of weights, and see how both algorithm are doing with each of them with a batch size of 1000 (even if 250 may be better, it is costly to compute). This time, we will use the MNIST database, as the topology of the associated loss function is harder as demonstrated in Sec. II. The results can be seen on Fig. 16.

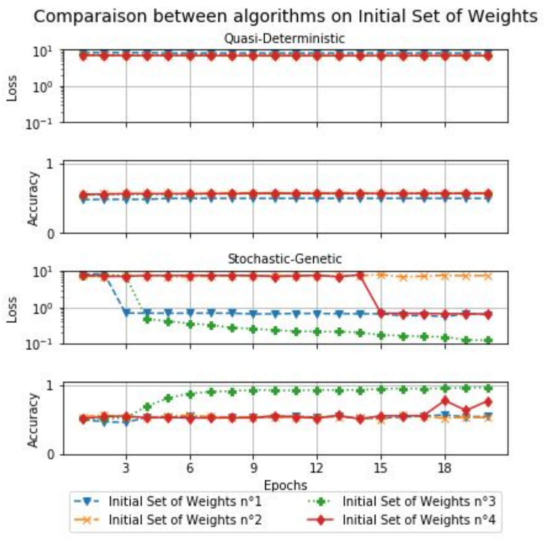

Fig. 16. Evolution of the loss function and the accuracy on 20 epochs for various initial sets of weights on the MNIST database.

The conclusion to draw is unanimous. While the deterministic algorithm did not manage to optimize the learning once, the ES algorithm manages to reduce the loss value for three of the four initial sets of weights (even if the accuracy did not really follow the trend for two of them). It clearly shows the better capacity of the ES algorithm to explore the space it faces. It might be explained by the fact that the QD algorithm is linked to its starting point, while the ES one is not as the generated population progresses at each iteration. Plus, the learning rate of the QD algorithm converges too quickly to zero. This advantage of the ES algorithm will be of a great importance for hard databases such as the EYES one. Speaking of which, let us look at Fig. 17.

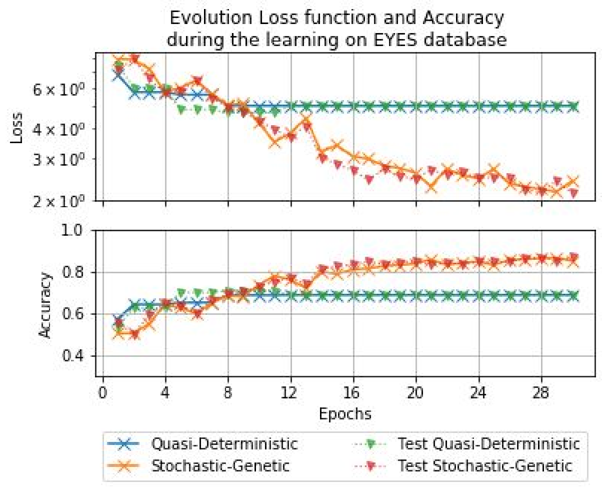

Fig. 17. Evolution of the loss function and the accuracy on 30 epochs working with the EYES database.

Here the problem of the space exploration is non negligible as the topology of the loss function is so complex as seen in Sec. II. Any algorithm with a poor exploration capacity will be stuck on a plateau, and that is what happened here for the deterministic algorithm. While its fellow manages to decrease the loss function and to increase the accuracy, the deterministic algorithm does not manage to get out the local minimum (probably a plateau) it founds around epoch number 8. Regarding the over-fitting, we can see that for both algorithm, the value of the loss function and the value of the accuracy on the test database fits very well with the ones on the train database. It means that if the learning succeed, it will be of a great quality.

Now we have brought to light the best DFO algorithm, which is without contests the ES one, we will compare in the next section its performances with the ones of a classic optimization algorithm for training neural network that is ADAM.

V. Comparison of the ES algorithms with ADAM

ADAM is a gradient based method, meaning it needs to process the slope at the current point in the space, and then take a step in the direction of the descent. That kind of method causes many issues when it comes to loss functions of a learning problem. In fact, as we have seen in Sec. II, there are a lot of plateaux, valleys, and obstacles such as cretes that can prevent gradient-based algorithm from exploring the space and find a valid minimum. And by valid, I mean a minimum close to the global one.

Take a very simple database, like the BLOPS one, and try to train the neural network with the Adam and the ES algorithm. If we start from a initial set of weights that is adequate, such as the one drawn with a normal distribution with mean 0 and scale 1, we can see on Fig. 18 that both algorithm manage to find a great minimum, with an final accuracy of 100%. Note that we are using a batch size of 1000, and the same other parameters as those decided in Sec. IV for the ES algorithm.

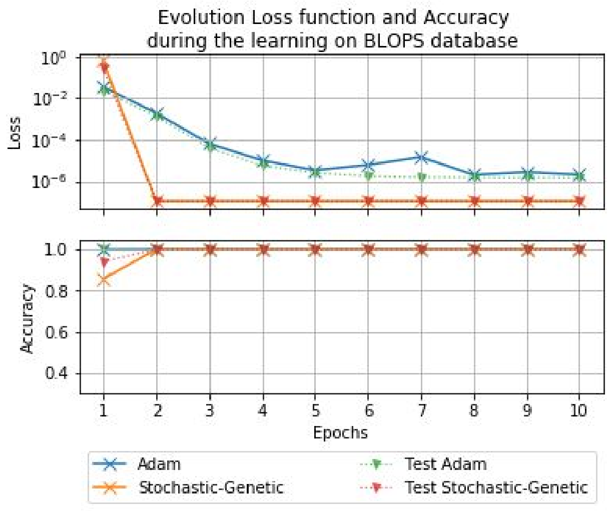

Fig. 18. Evolution of the loss function and the accuracy on 10 epochs working with the BLOPS database.

However, an inadequate starting set of weights like the one surrounding by the topology on Fig. 19, where we clearly see that for these two directions there is a huge plateau to overcome, will prevent ADAM from finding a slope. Fig. 20 shows that for this starting point, ADAM can not move from the plateau where it is stuck, whereas the ES algorithm manages to find its way out of it, and go to a proper minimum with an accuracy of 100%. And that is a first victory for the DFO algorithms.

An other kind of topology that will cause problems to ADAM will be the one associated with the EYES database, as shown on Fig. 8 in Sec. II. Whatever the starting point, ADAM will constantly fight with local minima, plateaux, and any other kind of issues. Fig. 21 shows that even after 30 epochs, ADAM did not manage to even increase the accuracy.

Unfortunately, nothing can be perfect and the ES algorithm is no exception. We have already seen in the last section that it is also sensitive to initial set of weights, even if it explores the space better, it might not reach a proper minimum. Moreover, as the DFO algorithm must evaluate a lot of point at each iteration, it is far slower than the gradient-based algorithm, as shown on Fig. 22.

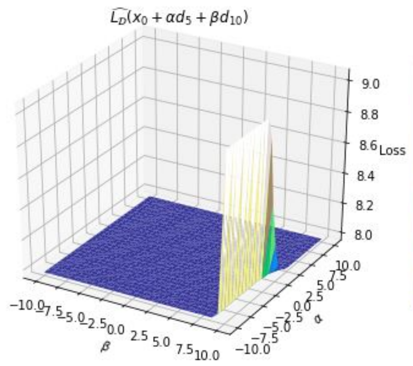

Fig. 19. Plot of the topology of the loss function in direction d_5 and d_{10} around the initial set of weights for the BLOPS database. Note the huge plateau surrounding it.

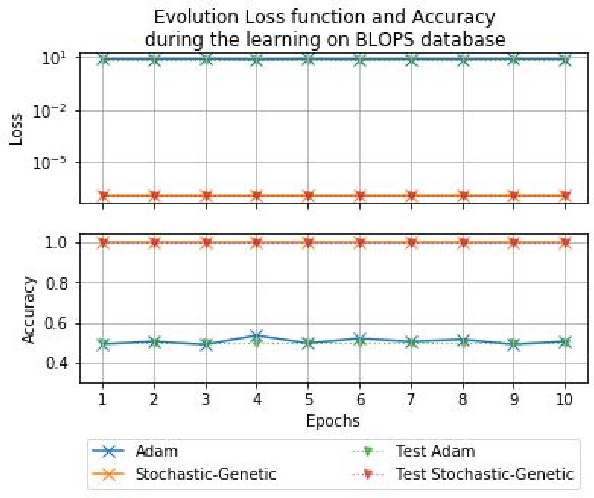

Fig. 20. Evolution of the loss function and the accuracy on 10 epochs working with the BLOPS database. Note that we can not see the evolution of the function during each epoch, explaining why the values for both algorithm are different on epoch 1, even if we started from the same initial set of weights.

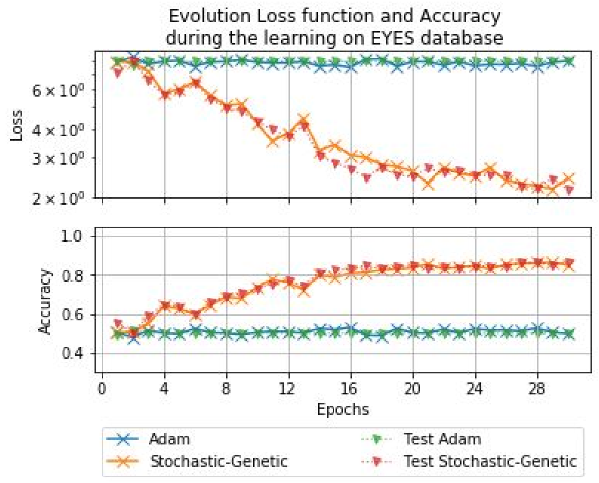

Fig. 21. Evolution of the loss function and the accuracy on 30 epochs working with the EYES database.

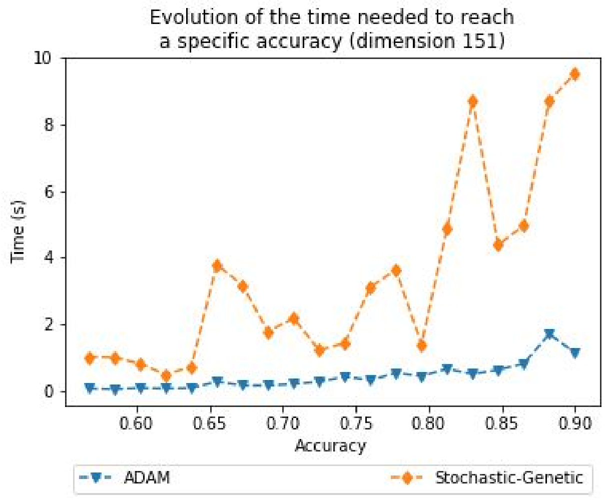

Fig. 22. Comparison of the two algorithms on the time needed to reach the given accuracy.

Here we can clearly see that ADAM is much faster than the DFO algorithm to reach a given accuracy. Even if it may take less epoch to reach it, the ES algorithm asks for more computational power and hence is slower to compute. However, if we consider the number of evaluation, ADAM will evaluate more times the function, as it needs to compute the gradients.

Finally, the greatest weakness of a DFO algorithm is the curse of dimension. The higher is the dimension, the greater the size of the points set generated at each iteration will be in order to have proper results [5]. But more points means more evaluation of the loss function, which is very costly to process. Fig. 23 shows this behaviour of the ES algorithm and the comparison with ADAM.

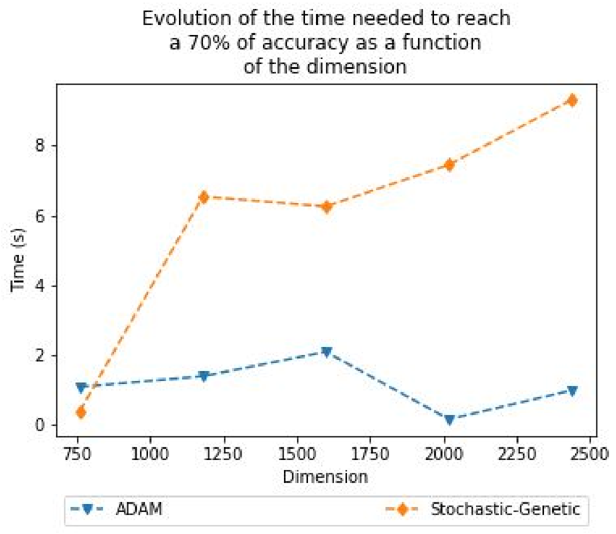

Fig. 23. Comparison of the two algorithms on the time needed to reach a 70% accuracy, as a function of the dimension.

As the time needed for ADAM to do the learning tends to grow a little when the dimension increases, it is the other way round for the ES algorithm. It means that DFO algorithms need to be used on small dimensions problems. Otherwise, it will spend to much time and take too much memory to train the neural network.

VI. Conclusion

As it is the most common method of optimization, gradient-based algorithms have always been used to train neural networks since their discovery. Globally, it works on most of the problems, it is not so costly in computation and very fast to settle. However, it is very sensitive to the initial set of weights chosen, and might get stuck in local minima, saddle points or plateaux. Although DFO algorithms are a bit harder to settle, slower, expensive in computation and do not scale very well with the size of the problem, they really represent a solution to these problems encountered by gradient-base methods. In fact, they tend to explore the space far better, and therefore are less sensitive to the issues encountered by their counterparts using derivatives. Hence they allow to find better minima, and to find them more often.

Moreover, hybrid algorithms could be a solution to the problem of slowness and expensiveness. In fact, using a gradient-based algorithm on ”easy” parts, where there is a correct slope, and a ES algorithm on ”hard” parts such as plateaux, local minima, etc... might lead to better learnings and thus better performances of neural networks for database classification. But the dimension of the neural network will be limited, and for very hard classification problem such as the ”fraudulent or not” banking transaction, where the size of the neural network can exceed tens of thousands of weights to optimize, the use of DFO algorithm may be forbidden.

Nevertheless, their use is not to be exclude for larger issues. With recent computers and high performance computing, the possibilities are large. Parallelization of the tasks can expand the dimensional limit of DFO algorithm to a far greater number, although the implementation will certainly be harder.

References

[1] Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. SIAM, Philadelphia, PA, USA, 2009.

[2] Youssef Diouane. Introduction to derivative-free optimization. 2017.

[3] Youssef Diouane, Serge Gratton, and Luis Nunes Vicente. Globally convergent evolution strategies. *Mathematical Programming*, 152(1):467–490, 2015.

[4] Youssef Diouane, Serge Gratton, and Luis Nunes Vicente. Globally convergent evolution strategies for constrained optimization. *Computational Optimization and Applications*, 62(2):323–346, 2015.

[5] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[7] Osvaldo Simeone. A brief introduction to machine learning for engineers. *CoRR*, abs/1709.02840, 2017.