Stochastic Optimization with Adaptive Batch Size: Discrete Choice Models as a Case Study

Gael Lederrey

Virginie Lurkin

Tim Hillel

Michel Bierlaire

Transportation and Mobility Laboratory, ENAC, EPFL

May 2019

STRC | 19th Swiss Transport Research Conference
Monte Verità / Ascona, May 15 – 17, 2019

Contents

Transportation and Mobility Laboratory, ENAC, EPFL

Stochastic Optimization with Adaptive Batch Size: Discrete Choice Models as a Case Study

Gael Lederrey
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
phone: +41-21-693 25 32
gael.lederrey@epfl.ch

Virginie Lurkin
Department of Industrial Engineering
Innovation Sciences
Eindhoven University of Technology
5612 AZ Eindhoven, Netherlands
v.j.c.lurkin@tue.nl

Tim Hillel
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
phone: +41-21-693 24 29
tim.hillel@epfl.ch

Michel Bierlaire
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
phone: +41-21-693 25 37
michel.bierlaire@epfl.ch

May 2019

Abstract

The 2.5 quintillion bytes of data created each day brings new opportunities, but also new stimulating challenges for the discrete choice community. Opportunities because more and more new and larger data sets will undoubtedly become available in the future. Challenging because insights can only be discovered if models can be estimated, which is not simple on these large datasets.

In this paper, inspired by the good practices and the intensive use of stochastic gradient methods in the ML field, we introduce the algorithm called Window Moving Average - Adaptive Batch Size (WMA-ABS) which is used to improve the efficiency of stochastic second-order methods. We present preliminary results that indicate that our algorithms outperform the standard second-order methods, especially for large datasets. It constitutes a first step to show that stochastic algorithms can finally find their place in the optimization of Discrete Choice Models.

Keywords

Keywords; Optimization, Discrete Choice Models, Stochastic Algorithms, Adaptive Batch Size

1 Introduction

The 2.5 quintillion bytes of data created each day brings new opportunities, but also new stimulating challenges for the discrete choice community. Opportunities because more and more new and larger data sets will undoubtedly become available in the future. In addition, these new data set comes the possibility to uncover new insights into customersâ behaviors. Challenging because insights can only be discovered if models can be estimated, which is not simple on these large data sets. State-of-the-art algorithms, but also standard practices regarding model specification might no longer be adapted to these large data sets. Indeed, three state-of-the-art discrete choice softwares (Pandas Biogeme (Bierlaire, 2018), PyLogit (Brathwaite *et al.,* 2017), and Larch (Newman *et al.,* 2018)) are using the standard optimization algorithm available in the Python package `scipy`, *i.e.* the BFGS algorithm and some variants.

In contrast, extracting useful information from big data sets is at the core of Machine Learning (ML). Primarily interested in achieving high prediction accuracy, ML algorithms (and especially Neural Networks) have proved to be successful on models involving a huge number of parameters. Thus, large-scale machine learning models often involve both large volumes of parameters and large data sets. As such, first-order stochastic methods are a natural choice for large-scale machine learning optimization. Due to the high cost of computing the full-Hessian, the second-order methods have been much less explored. And yet, algorithms exploiting second-order information can provide faster convergence.

For the sake of interpretability, discrete choice models usually have a more restricted set of parameters than models typically investigated in the ML community. We, therefore, argue that it is possible to use second-order information to estimate these models. In this paper, inspired by the good practices and the intensive use of stochastic gradient methods in the ML field, we introduce the algorithm called Window Moving Average - Adaptive Batch Size (WMA-ABS) which is used to improve the efficiency of stochastic Iterative Optimization Algorithms (IOAs). The objective of this paper is to investigate the convergence of our algorithms by benchmarking it against standard IOAs using primarily Discrete Choice Models (DCMs) on different data sets. We also show that the WMA-ABS improves different IOAs, ranging from first-order to complex second-order, in terms of optimization speed. We conclude with a parameters study on the WMA-ABS.

2 Related Work

While the principle of optimization is pretty easy to understand, *i.e.* finding a maximum or a minimum value that can be subject to some constraints, there are many ways to achieve the optimum. Some of the major subfields are Convex Programming, Integer Programming, Combinatorial Optimization, and Stochastic Optimization. For this thesis, we are interested in Iterative Optimization Algorithms (IOA) with the addition of stochasticity. Thus, we will not discuss other types of optimization. We recommend the reader to read the book named "Optimization: Principles and Algorithms" of Bierlaire (2015) for a good mathematical introduction on the principle of optimization and some algorithms.

The common ancestor of all IOA has to be the Gradient Descent (Cauchy, 1847). Its principle is straightforward. We define a function $f(x) : \mathbb{R}^n \to \mathbb{R}$ such that it is defined and derivable in a neighborhood of a point a. We know that f will decrease the fast along its negative gradient. Thus, if

$$x_{n+1} = x_n - \gamma \nabla f(x_n) \tag{1}$$

with γ the step size small enough, then $f(x_n) \geq f(x_{n+1})$. From this very principle, many algorithms have been created. One of the branches deriving from this algorithm is the Stochastic Algorithms. One of the first algorithm to appear is Stochastic Gradient Descent[1] (SGD). The principle is almost the same as in Equation 5. Let us define a function F such that it is a finite sum of other functions.

$$F(x) = \frac{1}{n} \sum_{=1}^{n} f_i(x) \tag{2}$$

Each function f_i is generally associated with the i-th observation in the dataset. Then, we can simply replace $f(x_n)$ in Equation 5 by $f_j(x_n)$ where j is a random index. This definition corresponds to the SGD. This algorithm inspired many researchers. Instead of providing you with a list of algorithms arising from SGD, we give you a schematic representation of the inspiration for many first-order stochastic algorithms in Figure 1. If you want more information about the algorithms in Figure 1, we recommend the reader to have a look at the paper of Ruder (2016).

[1] We do not have a precise date for its emergence. Many sources are dating its origin back to the 1940's.

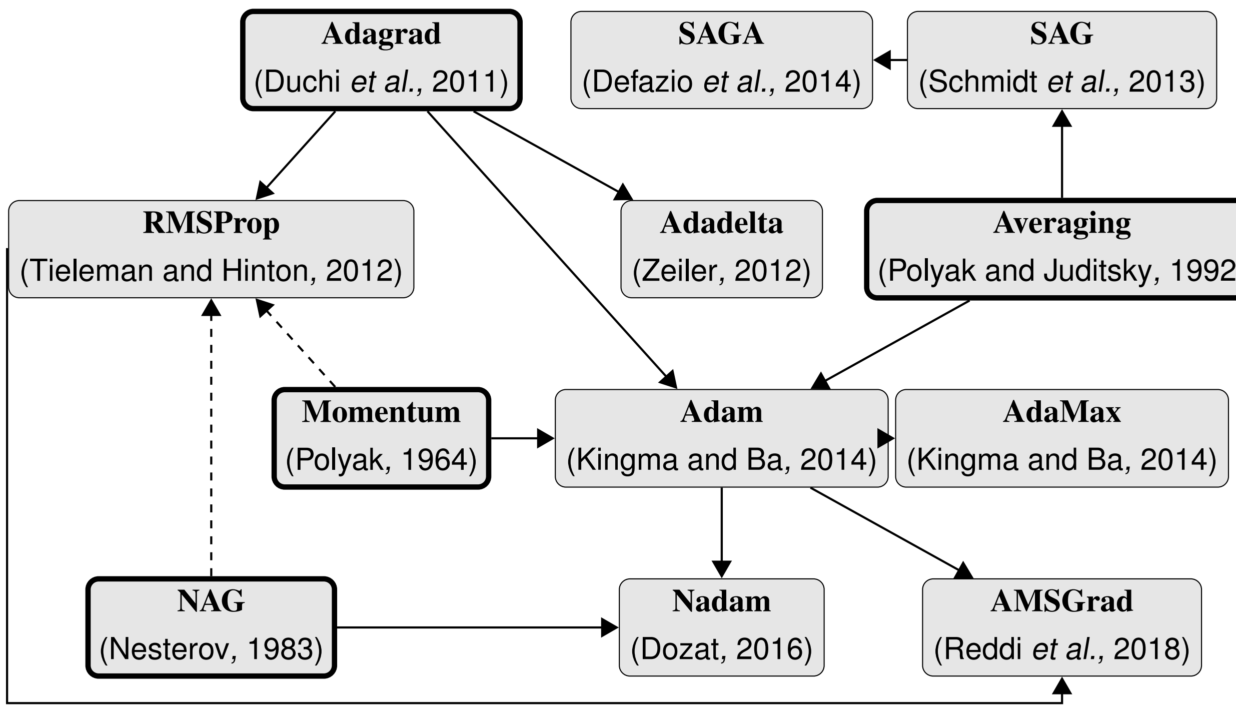

Adagrad (Duchi et al., 2011)
SAGA (Defazio et al., 2014)
SAG (Schmidt et al., 2013)
RMSProp (Tieleman and Hinton, 2012)
Adadelta (Zeiler, 2012)
Averaging (Polyak and Juditsky, 1992)
Momentum (Polyak, 1964)
Adam (Kingma and Ba, 2014)
AdaMax (Kingma and Ba, 2014)
NAG (Nesterov, 1983)
Nadam (Dozat, 2016)
AMSGrad (Reddi et al., 2018)

Figure 1: Historical development of the first-order iterative methods. Each algorithm with a bold border have the Stochastic Gradient Descent (SGD) as their common ancestor.

All the algorithms given in Figure 1 do not represent every first-order stochastic method ever created. However, it gives a good overview of the work that has already been done. The reason behind all of these needed algorithms is Machine Learning. Indeed, this field has been trending for the last twenty to thirty years with the quick development of computer technologies. People were able to process more data since they had more and more power on their computers. In the last fifteen years, researchers have been working a lot on Neural Networks. We recommend the reader to have a look at the book "An Introduction to Neural Networks" from Gurney (2014) for more information about these models. Apart from being very complex biologically-inspired models, Neural Networks are good at working with huge datasets and an important number of features. Thus, if we want to optimize such a model, it may be difficult, or even impossible, to compute the Hessian of this model due to the number of features. Thus, researchers have mainly focused on developing first-order methods to solve this particular problem.

However, researchers have not neglected second-order and quasi-Newton methods. They have been developing these techniques for a specific purpose. For example, Gower *et al.* (2018) improved the BFGS algorithm specifically for solving Matrix Inversion problems. They are using a stochastic version of the BFGS algorithm in their case. Since our computers are much more powerful nowadays, researchers have started to work with second-order methods. However, they try to avoid to compute the Hessian since it is quite a heavy work. Martens (2010), for example, developed a Hessian-free optimization technique specifically for Deep Learning. Other researchers have worked on Hessian-free optimization such as Kiros (2013), and Wang *et al.* (2014). Some researchers have been working on modifying the BFGS algorithms in a stochastic way which corresponds to Hessian-free optimization. We can cite the articles of Mokhtari and Ribeiro (2014), Keskar and Berahas (2016), and Bordes *et al.* (2009*,* 2010). The literature on second-order stochastic methods is sparser compared to first-order and quasi-Newton methods. Nevertheless, we can cite the article of Byrd *et al.* (2011) who are trying to get some information from the Hessian to improve the optimization process. More recently, Agarwal *et al.* (2016) developed a second-order stochastic method specifically for Machine Learning, *i.e.* using the finite-sum objective function.

In our case, we are interested in techniques that can be helpful for any kind of IOAs, first-order or second-order. For example, all algorithms in Figure 1 have one characteristic in common: they all use static batch size. For a long time, researchers have been only trying to find better learning rates such as the algorithm Adagrad. However, in recent years, thanks to the availability of more advanced neural networks architecture and more power, researchers have been interested in Adaptive Batch Size (ABS). We can cite the article of Balles *et al.* (2016) who are coupling the adaptive batch size with the learning rate. They show the mathematical motivation to couple both of these parameters. However, they do not explore the performance improvement. Devarakonda *et al.* (2017), on the other hand, demonstrate that they can achieve a speedup of up to 1.5 times in terms of optimization speed for different neural networks. Following these articles, multiple researchers have been working on different versions of ABS. Bollapragada *et al.* (2018) have developed an ABS for L-BFGS specifically for Machine Learning. They even propose a convergence analysis of their algorithm. Shallue *et al.* (2018) makes use of new hardware technologies for parallelizing the optimization process. In their article, they study the relationship between batch size and the number of training steps, which is related to the optimization speed. They do not present a new ABS method but they show that the theory behind this method works and is worth exploring in more details.

3 Window Moving Average - Adaptive Batch Size

We present in this section the algorithm named Window Moving Average - Adaptive Batch Size (WMA-ABS). This algorithm has been developed with the idea in mind that if a stochastic algorithm is struggling to improve, it can be fixed by updating the batch size. However, due to the high fluctuation of the loss function, it is difficult to distinguish, at the early sign of lack of improvement, if this behavior is due to the stochasticity of the algorithm or the prior. We thus need a way to smooth the loss function and the improvement. We used the Window Moving Average algorithm often used in Economics and Computer Vision to smooth our functions.

We start by defining the improvement of the loss function as follows:

$$\Delta_i = \frac{\mathcal{L}_{i-1} - \mathcal{L}_i}{\mathcal{L}_{i-1}} \tag{3}$$

Where i is the current iteration, and $\mathcal{L}_i$ is the value of the log likelihood at iteration i. We can now define the Window Moving Average (WMA) for an arbitrary function f. For a window of n values at the current iteration M, the WMA at M is defined by:

$$\text{WMA}_{M,n} = \frac{nf_M + (n-1)f_{M-1} + \cdots + 2f_{M-n+2} + f_{M-n+1}}{n + (n-1) + \cdots + 2 + 1} \tag{4}$$

Where f_i corresponds to the value of the function f at iteration i. Using the smoothed value of the objective function, it becomes easier to interpret the improvement. Indeed, if the window is large enough, we will not see any spikes in the improvement due to the stochasticity of the optimization algorithm.

The WMA-ABS is an algorithm that has been designed such that it can be used with any standard Stochastic Iterative Optimization Algorithm (SIOA). Algorithm 1 shows the pseudo code for the standard IOA. Line 4 is the most important line where much can change between algorithms. Indeed, to compute the direction of descent, we can either use the gradient where p_k is defined as:

$$p_k = -\nabla f(x_k) \tag{5}$$

In this case, we only use the gradient of the function f. If we compute the direction using the Newton step, we also use the Hessian of the function f. To find the search direction, we need to solve the following equation:

$$\nabla^2 f(x_k) \cdot p_k = -\nabla f(x_k) \tag{6}$$

For stochastic IOA, the only difference is the rule with which we will compute the direction. Indeed, most of them require to draw a sample of n observations and compute the gradient or the Hessian on these observations only. The rest of these algorithms does not change much from the pseudocode in Algorithm 1.

As stated earlier, the WMA-ABS is a method that can be plugged to any type of SIOA. The idea is to simply have a look at the improvement of the loss function at the end of the iteration and

Algorithm 1 Pseudocode for standard IOA

Input: function (f), initial parameters (x_0), the maximum number of iterations ($I_{\max}$), and the stopping criterion (ε^*)

Output: optimal parameters (x^*)

```
1: function OPTIMIZE
2:     Define the current parameters $x_k = x_0$, Number of iteration $k = 0$, Stopping criterion $\varepsilon = \infty$
3:     while $k < I_{\max}$ && $\varepsilon < \varepsilon^*$ do
4:         Compute the search direction $p_k$ in function of $f(x_k)$, $\nabla f(x_k)$, and $\nabla^2 f(x_k)$
5:         Compute the step size $\alpha$ using a line search algorithm
6:         Update the parameters: $x_{k+1} = x_k + \alpha p_k$
7:         Update the other variables: $k = k + 1$ and $\varepsilon = ||\nabla f(x_{k+1})||$
       return $x^* = x$, the optimal parameters
```

decide if the algorithm is improving enough or not. To decide if the improvement was good enough, we need to use the value of the objective function for the previous iterations. We can store the WMA of the objective function in an array and look at its improvement between the current iteration and the previous one. Then, we can simply argue that if the current improvement is lower than a given threshold, the algorithm is not improving enough. If we want to be more conservative, we can ask the method to wait until a certain number of iteration with a low improvement has been reached before updating the batch size. To update the batch size, we can define a factor by which the batch size will be increased. We thus define the four parameters used in the method WMA-ABS:

- **W** corresponds to the size of the window. It is the same as n in the explanation of the WMA, see Equation 4.
- $\boldsymbol{\Delta}$ corresponds to the threshold under which the batch size will be increased.
- **C** corresponds to the maximum number of times the improvement can be under the threshold Δ. If the number of time is bigger than C, the batch size will be increased.
- $\boldsymbol{\tau}$ corresponds to the factor multiplying the current batch size. For example, if $\boldsymbol{m}$ corresponds to the current batch size, the next one will be equal to $\boldsymbol{\tau} \cdot \boldsymbol{m}$.

We can now present the pseudocode of the WMA-ABS. It is given in Algorithm 2. As you can see, we start the current function value in a global array named $\mathcal{F}$. We then compute the WMA and store it in another array $\mathcal{A}$. Then, we check that it is not the first iteration to avoid an issue with the computation of the improvement. If it is not the case, we can compute it using Equation 3. Then, we check if the current batch size is smaller than the full size. If it is the case, it means that we can increase it. We then check if the current improvement is smaller than the threshold Δ. If it is the case, we update the counter c. If the counter achieves the defined value of C, we decide to increase the batch size. We simply multiply the current batch size by the update factor τ and return it.

Algorithm 2 Window Moving Average - Adaptive Batch Size (WMA-ABS)

```
Input: Current iteration index ($M$), function value at iteration $M$ ($f_M$), batch size ($n$), and full
    size ($N$)
Output: New batch size ($n'$)
 1: function WMA-ABS
 2:     Store $f_M$ in a list $\mathcal{F}$
 3:     Compute WMA$_{M,W}$ using $\mathcal{F}$ and store it in a list $\mathcal{A}$
 4:     if $M > 0$ then          ▷ We need at least two values to compute the improvement.
 5:         Compute $\Delta$ the improvement as in Equation 3 using the list $\mathcal{A}$ and store it in a list $\mathcal{I}$
 6:         if $n < N$ then
 7:             if $\mathcal{I}_M < \Delta$ then          ▷ Improvement under the threshold
 8:                 $c = c + 1$
 9:             else
10:                 $c = 0$          ▷ We restart the counter
11:             if $c == C$ then          ▷ We will update the batch size
12:                 $c = 0$          ▷ We restart the counter
13:                 $n' = \tau \cdot n$
14:                 if $n' >= N$ then          ▷ The batch size is too big now
15:                     $n' = N$
16:             else
17:                 $n' = n$
        return $n'$
```

As shown above, the WMA-ABS method uses four different parameters. While it is possible to optimize these hyperparameters, it would be counterproductive since we want to speed up the optimization process of SIOA. We thus suggest some set of parameters given in Table 1.

Table 1: Suggested parameters for the WMA-ABS algorithm

	Simple	Complex
W	10	10
Δ	1%	1%
C	1	2
τ	2	2

The window has a value of 10 to allow some smoothing. We argue that this parameter does not have a huge influence on the speed of the optimization and we show it in the results. The threshold is set low enough such that we do not update the batch size while the algorithm can still show some significant improvements. The multiplication factor is set such that it gives a significant boost in terms of batch size but keep it small at the beginning of the optimization process. The only parameter that changes is the count before updating the batch size. We give two different values for two types of models: simple and complex. While the differentiation between these two types of model is difficult to do, the reason why we change this parameter is easier to understand. Indeed, for a simple model, the objective function is generally less complex. Thus, any IOA should be able to optimize it faster and without issue. Therefore, we can be confident about the computation of the improvement of our algorithm while using WMA-ABS. For the complex, on the contrary, we need to make sure that the algorithm is struggling to improve itself. We thus prefer to wait and make sure that the algorithm is not able to improve after two consecutive steps before updating the batch size.

4 Case study

As a case study, we decided to use multiple IOA and create their stochastic counterparts. We also define multiple discrete choice models on multiple data sets. Discrete Choice Models (DCMs) are a fascinating case study for the study of optimization. Indeed, DCMs are built such that they keep their interpretability. We thus cannot make models with thousands of parameters as seen in Deep Learning. In addition, we can use second order optimization algorithms such as Newton Method since the computation of the Hessian is analytically possible. All the models have been created using Pandas Biogeme (Bierlaire, 2018). The algorithms have been written in Python to work with Pandas Biogeme. In this section, we will first present the algorithms and their stochastic counterparts. We will then present the different models and their data set.

4.1 Algorithms

We selected four different standard IOA. The IOAs are defined in Algorithm 1. In this section, we present the similarities and differences between the selected IOAs. The first algorithm we selected is Gradient Descent (Cauchy, 1847). If we define the function f being the objective function we are trying to optimize, then the search direction p_k is given by Equation 5, *i.e.* $p_k = d$. There are much more advanced stochastic first-order optimization algorithms. However, the effect of the WMA-ABS algorithm may be less important due to the improvements already made. However, in this specific case study, we work with DCMs. Thus, we can use algorithms at a higher order. We thus decided to select the algorithm BFGS (Fletcher, 1987). To find the search direction, we need to solve the following Newton equation:

$$B_k p_k = -\nabla f(x_k) \tag{7}$$

where B_k is an approximation of the Hessian matrix at iteration k. The quasi-Newton condition used to update B_k is given by:

$$B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k) \tag{8}$$

From this condition, we can derive the update B_{k+1} using different substitutions. We recommend the reader to have a look at the literature for more details on the BFGS algorithm. The final two algorithms that were chosen are the Newton method and the Trust-Region algorithm. The search direction for the Newton method is given in Equation 6. The Trust-Region algorithm does not have a search direction defined in only one equation. Indeed, the idea of this algorithm is to define a subproblem around the current iteration. This subproblem is written as such:

$$\begin{aligned} \min \quad & m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \\ \text{s.t.} \quad & \|p\| \leq \Delta_k \end{aligned} \tag{9}$$

Where Δ_k corresponds to the radius of the current subproblem. Then, we have to check the model validity using the following formula:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \tag{10}$$

Where p_k is the optimal solution in Equation 9. Then, depending on the value of ρ_k, we will decide if the radius Δ_k has to change and if the current iteration x_k can be updated using p_k as the search direction. The most complex part of this method is to solve the Trust-Region subproblem. In our case, we used the Conjugated Gradient Steihaugâs Method (Steihaug, 1983). We refer the reader to this article for more details about this algorithm.

4.2 Models

In this section, we present the different models that are used for the case study. Most of the models presented are DCMs and were built with Pandas Biogeme. The rest was created using the Python package `scikit-learn` Pedregosa *et al.* (2011). We choose a different type of models, more or less complex, and with different data set size. Table 2 give a summary of all the model presented in this article. More details on the different models are given below.

Table 2: Summary of the models used for the case study

Name	Type	#Parameters	Data set	#Observations
`MNL-SM`	MNL	4	Swissmetro	6'768
`Nested-SM`	Nested	5	Swissmetro	6'768
`LogReg-BS`	Logistic Regression	12	Bike Sharing	16'637
`small MNL-CLT`	MNL	100	London Passenger Mode Choice	27'478
`MNL-CLT`	MNL	100	London Passenger Mode Choice	54'766

`MNL-SM` is a simple Multinomial Logit Model using the Swissmetro data set (Bierlaire *et al.*, 2001). The Swissmetro data set contains 10'728 observations. This model has four different parameters, two for the constants and one for both time and cost. This model can be found on Biogeme website[2] under the name `logit01.py`. A few observations were removed due to not having the choice displayed or other factors. In the end, this model uses 6'768 observations.

`Nested-SM` is a simple Nested Logit Model using the Swissmetro data set. It has exactly the same parameters as the model `MNL-SM` with the addition of the parameter for the nest. This model can also be found on Biogeme website[1] under the name `09nested.py`. The same observations were removed for a total of 6'768 observations used.

`LogReg-BS` is a Logistic Regression using the Bike Sharing data set[3]. This regression is using the 12 parameters available which correspond to all parameters except the parameter `cnt`. Indeed, this model is trying to predict the number of bikes that have been rented every hour. The data set has 16'637 observations. We use the first two years as a training set and the remaining third year as a test set.

`small MNL-CLT` is a Multinomial Logit Model using the London Passenger Mode Choice data set[4] Hillel *et al.* (2018). The model has 100 parameters and is available in Hillel (2019). The data set spans over three years, from 2012 to 2014. For this specific model, we only used the year 2012 for a total of 27'478 observations.

`MNL-CLT` is exactly the same model as `small MNL-CLT` except that we used the years 2012-2013 to train the model for a total of 54'766 observations.

[2]http://biogeme.epfl.ch/examples_swissmetro.html

[3]It can be found on UCI: https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset.

[4]Contact `tim.hillel@epfl.ch` for more information about this data set.

5 Results

In this section, we present the results obtained with the proposed method WMA-ABS. We start by showing that it effectively improve the optimization compared to stochastic IOAs. We then show in details the results obtained with the different models and algorithms presented in the case study. We conclude this section by showing a study on the suggested parameters and explain them in more details.

5.1 Effect of WMA-ABS

If we want to study the effect of WMA-ABS on the optimization of DCMs, we need to start with a baseline. We thus decided to use the simplest model, the `MNL-SM`, and optimize it with the four algorithms presented in the case study. The results are given in Figure 2.

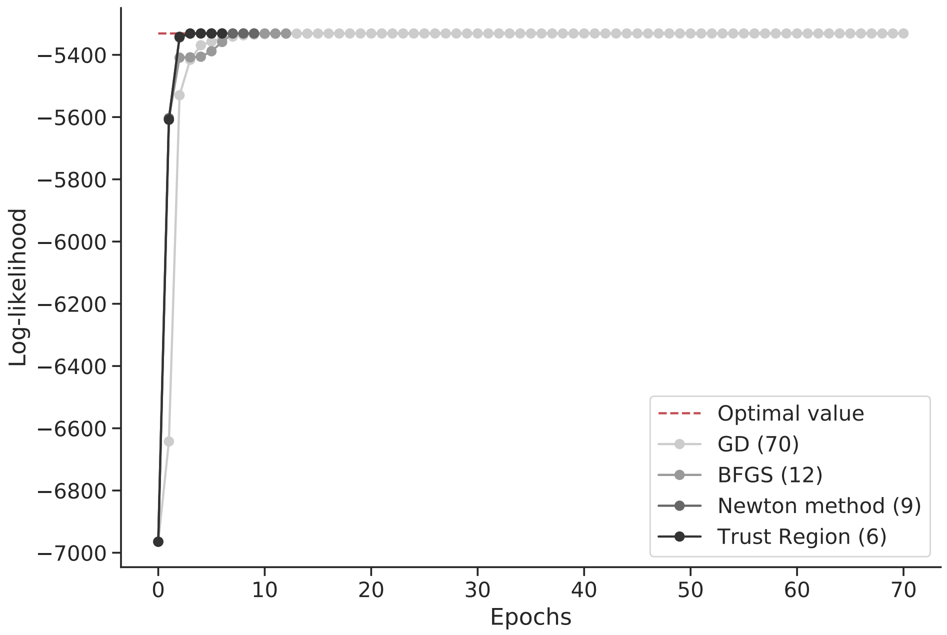


Figure 2: Optimization of the model `MNL-SM` with the Gradient Descent, the Newton method, the BFGS, and the Trust-Region algorithms. The number in parentheses in the legend gives the number of epochs needed until convergence.

Based on the literature, the results obtained in Figure 2 are expected. Indeed, Gradient Descent takes the most number of epochs to optimize the model, while Trust-Region took the least. We can thus first conclude that this model can be globally optimized. The next step is to use a stochastic IOA to optimize this model. Lederrey *et al.* (2018) developed the Stochastic Newton Method (SNM) and we applied it to this model. The results are given in Figure 3.

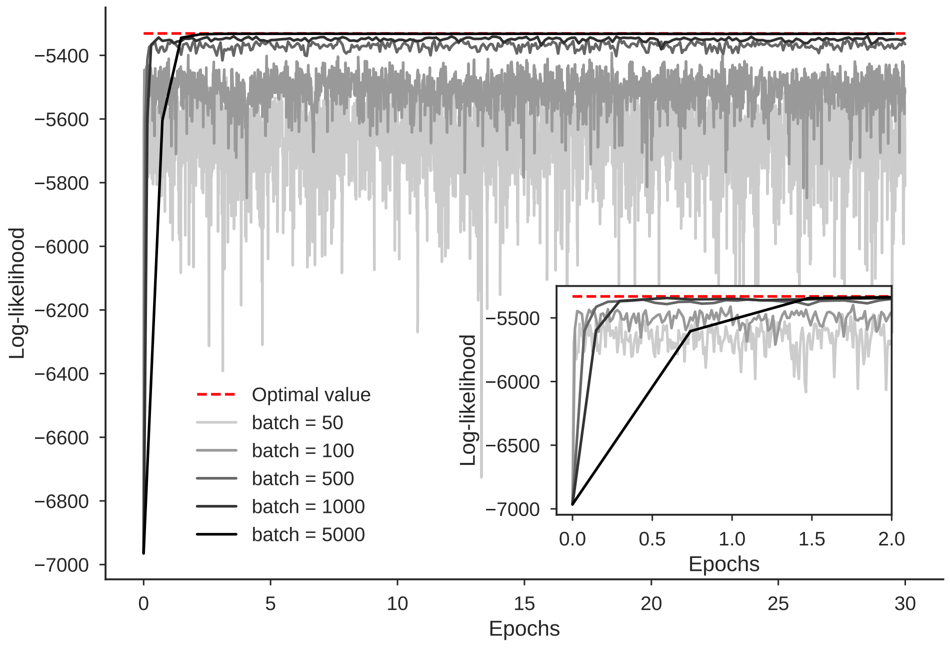


Figure 3: Optimization of the model `MNL-SM` with the Stochastic Newton Method. The lines correspond to the average values over 10 runs.

From Figure 3, we can extract some useful information:

- The value of the objective function tends to be extremely noisy, especially with small batch size.
- The SNM is not able to converge in a reasonable number of epochs no matter the batch size. Indeed, we used a batch size of up to half the size of the data set and the algorithm seems to be stuck close to the optimum. However, it is not able to fully converge.
- Using smaller batch size increase the speed of the optimization at the beginning of the optimization process, thus comforting the idea of starting with small batch size and updating it later on.

We can now address the first two issues. Indeed, the use of the WMA is justified by the fact that we need to have a precise idea on the improvement of the objective function. Thus, its smoothing will greatly help to interpret its change of value. Concerning the lack of convergence, Lederrey *et al.* (2018) gives a detailed discussion. In our case, we can easily address this issue by finishing the optimization process with a full batch, *i.e.* using all the available data. Finally, by using the third point, it is almost certain that we can speed up the optimization process.

Now that we have all the information needed from the experiment with stochastic algorithms, we can have a look at the behavior of the WMA-ABS algorithm. As a starting point, we used the stochastic Newton method and added the WMA-ABS method using the parameters for simple models provided in Table 1. We optimized the model `MNL-SM` for the comparison with a starting batch size of 100.

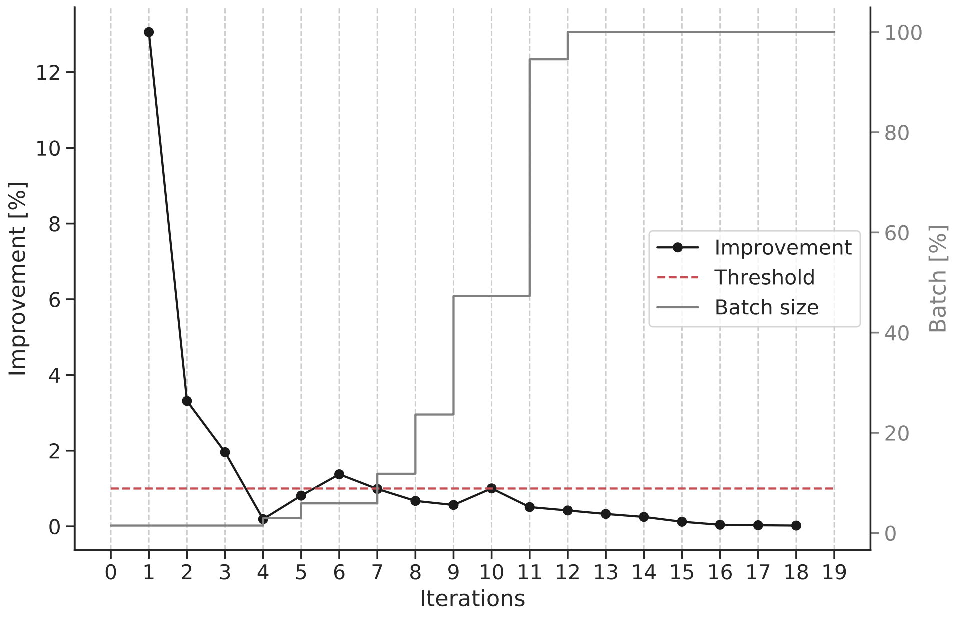


Figure 4: Improvement of the log likelihood and update of the batch size using the WMA-ABS algorithm with the stochastic Newton method on the optimization of the model `MNL-SM`.

In Figure 4, the 19 iterations corresponds to 9.45 epochs which is more than the 9 epochs that NM needed to optimize the model `MNL-SM`. We cannot make the conclusion that the WMA-ABS is not good enough yet[5]. Next section gives a more detailed comparison including this one. We should, therefore, concentrate on the improvement curve. At first, we see that the algorithm makes a few steps with the starting batch size. Once the improvement goes under 1%, the batch size is updated and we can see that it spikes once outside of the threshold zone. So, the WMA-ABS is able to make better improvement. Logically, the improvement becomes smaller again. But due to multiple updates of the batch size, the algorithm is able to improve better than the threshold once more. It finishes the optimization process by some iterations at the full batch size.

The interesting facts about the WMA-ABS algorithm are that it is definitely able to create better improvements on the objective function. As shown with this specific example, the WMA-ABS method may not be able to always be faster than its standard counterpart. However, this kind

[5] If it was the case, this paper would not exist.

of behavior is expected on simple models with a small data set. We will thus continue the investigation on the other models with the other algorithms.

5.2 Comparison between algorithms

Now that we know that the WMA-ABS method is working, we want to study in more depth its efficiency. We start with the model `MNL-SM`. The goal is to optimize it using the WMA-ABS algorithm in addition to the four stochastic IOAs presented in the case study. We use the suggested parameters for a simple model and start with a batch size of 100. We optimized the model 20 times and show the number of epochs required to optimize the model in Figure 5.

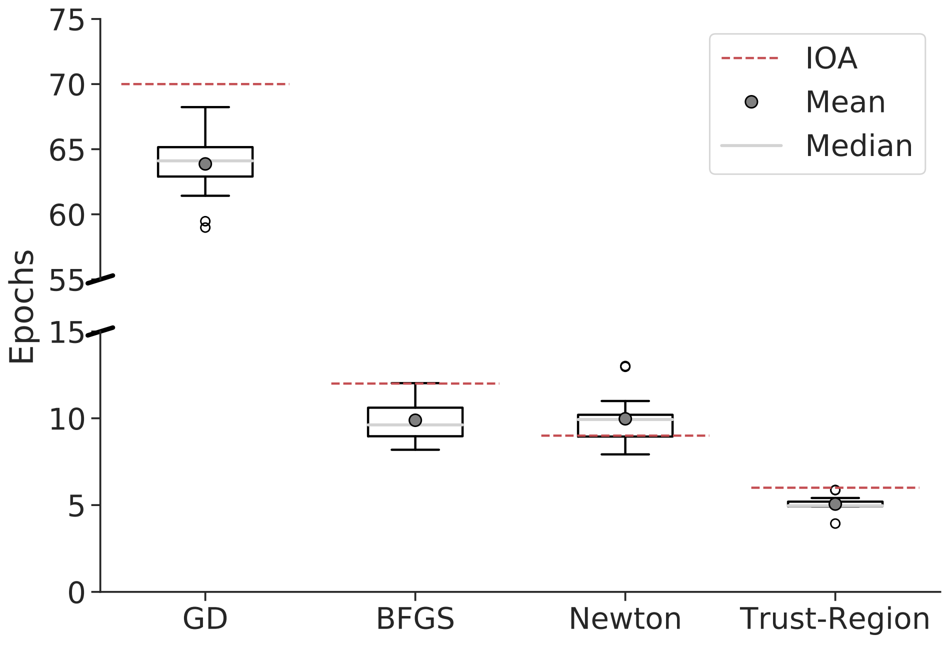


Figure 5: Comparison of WMA-ABS on the four optimization methods for the model `MNL-SM`. The red dotted lines correspond to the number of epoch required for the corresponding IOA to optimize the model.

In Figure 5, the red dotted lines are equivalent to the values given in parentheses in Figure 2. The goal of the WMA-ABS is to be under this value. We can see here that it is the case for all algorithms except the Newton method. So, even for a simple model with few data, the WMA-ABS algorithm is able to beat its standard counterpart. We can do now the same comparison with the model `Nested-SM`. We do not change the set of parameters for WMA-ABS nor the starting batch size. The results are given in Figure 6.

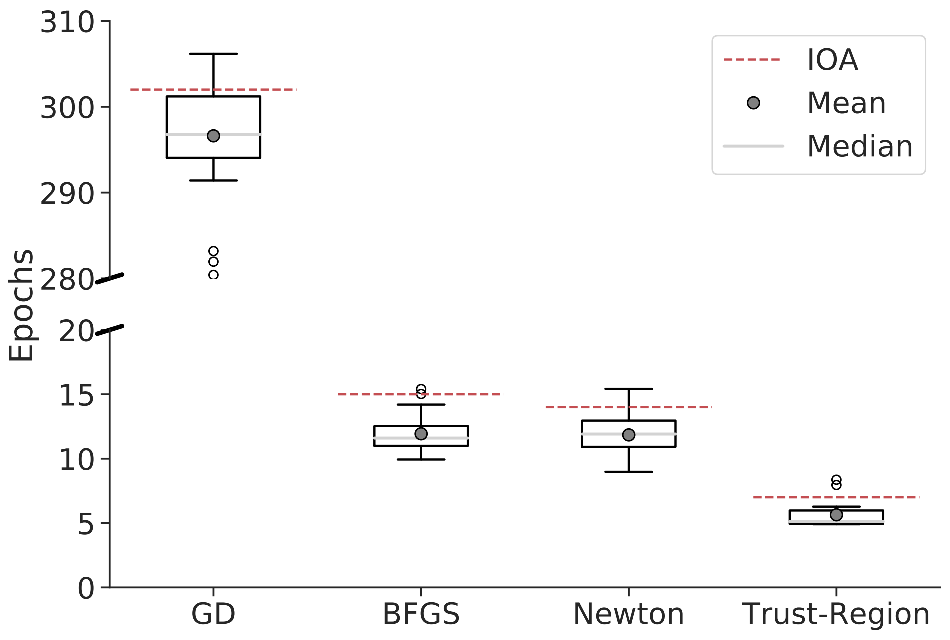


Figure 6: Comparison of WMA-ABS on the four optimization methods for the model `Nested-SM`. The red dotted lines correspond to the number of epoch required for the corresponding IOA to optimize the model.

For this model, we can see that the number of epochs required to optimize the model with the Gradient Descent increased greatly. For the other algorithms, the optimization process is only a little bit longer compared to the model `MNL-SM`. However, it is interesting to note that this time, the WMA-ABS algorithm was able to beat all of its counterparts. The difference is less significant for the Gradient Descent. It is also interesting to see that BFGS and Newton Method are close. This shows that the approximation of the Hessian is sometimes more than enough. In the end, we obtain an improvement of up to 20.44% for BFGS, 15.38% for NM, and 19.40% for Trust-Region. We can now do the same study for the most complex model with the most data in our case study: the `MNL-CLT`. However, we will not show the results for the Gradient Descent since it would surely skyrocket and take too much time to optimize. We also changed the set of parameters to use the ones suggested for complex models. Finally, we started with a batch size of 1'000 observations to avoid issues with batches that are too small. The results are given in Figure 7

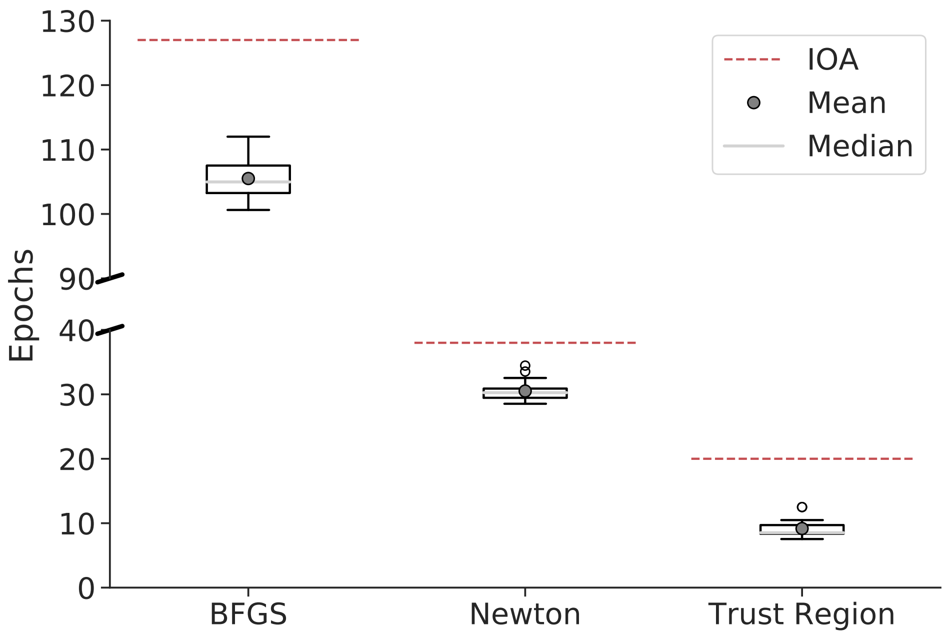


Figure 7: Comparison of WMA-ABS on the four optimization methods for the model `MNL-CLT`. The red dotted lines correspond to the number of epoch required for the corresponding IOA to optimize the model.

The results show that WMA-ABS is able to optimize faster this model than its counterparts. We also see that BFGS is struggling to optimize it. It is possible that it comes from the fact that the Hessian has now a size of 100×100 which is quite big to simply approximate it. The improvement over the standard IOAs is of 16.93% for BFGS, 19.70% for Newton Method, and 54.14% for Trust-Region. The last result is impressive and can be explained by the fact that Trust-Region may be struggling at the beginning of the optimization with a full batch. Thus, it helps a lot to only select part of the data and uses more and more data the closer we get to the optimum solution.

We finally present two tables to summarize the results obtained thus far. Table 3 present the number of epochs used to optimize the models `MNL-SM`, `Nested-SM`, and `LogReg-BS` for the four algorithms presented in the case study. The line named IOA corresponds to the values that the WMA-ABS has to beat. We used both suggested sets of parameters as well as optimized parameters. These optimized parameters have been found using the Python package `hyperopt` (Bergstra *et al.*, 2015). The values for some of these parameters are given in Figure 5 and are discussed in the following section. The first conclusion we can draw from Table 3 is that, except for the model `MNL-SM` with the Newton method, the WMA-ABS algorithm is always able to beat its counterpart. For the Gradient Descent, the set of parameters does not seem to change anything in the length of the optimization. However, the set of parameters for simple models is better by quite a margin for the other algorithms. If we optimize the WMA-ABS parameters, we only obtain a small improvement for the BFGS and Newton method. For Trust-Region, improvement is more important.

Algorithms		`MNL-SM`	`Nested-SM`	`LogReg-BS`
GD	IOA	70	302	/
	Simple	63.74 ± 3.32	296.61 ± 8.53	/
	Complex	**63.15 ± 2.36**	**295.27 ± 7.02**	/
	optimized	/	/	/
BFGS	IOA	12	15	23
	Simple	9.81 ± 1.10	**11.93 ± 1.53**	**20.10 ± 0.97**
	Complex	10.83 ± 1.12	12.50 ± 1.15	21.03 ± 0.80
	optimized	**9.25 ± 1.21**	/	20.32 ± 1.17
Newton	IOA	**7**	14	27
	Simple	9.97 ± 1.26 $*$	**11.85 ± 1.65**	25.31 ± 2.39
	Complex	11.70 ± 1.42 $*$	12.74 ± 1.30	**24.53 ± 2.14**
	optimized	9.55 ± 0.96 $*$	/	24.89 ± 2.11
Trust-Region	IOA	6	7	7
	Simple	5.06 ± 0.39	**5.64 ± 0.97**	4.86 ± 0.10
	Complex	6.52 ± 0.48 $*$	6.76 ± 0.31	6.35 ± 0.11
	optimized	**3.55 ± 0.22**	/	**3.95 ± 0.51**

Table 3: Number of epochs required to optimize the first three models. The numbers corresponds to the average value and the standard deviation over 20 optimization process. The bold values indicates the best set of parameters. Values with an asterisk correspond to the WMA-ABS method being worse than its counterpart.

Table 4 present the number of epochs used to optimize the models `small MNL-CLT` and `MNL-CLT` for three of the algorithms presented in the case study. For these two models, the best set of parameters is the one suggested for complex models. The difference is especially important for the Trust-Region algorithm. We achieve here improvements between 15% and 54% with the right set of parameters. It is interesting to note that the size of the data set does not play an important role in the optimization of this model. Indeed, we see that the number of epochs required to optimize the models to convergence is quite similar between the algorithms. The fact that Trust-Region is faster to optimize the model `MNL-CLT` rather than the model `small MNL-CLT` is difficult to explain. It is possible that it is due to the fact that there is more correlation in the year 2013 of the data. This needs to be investigated in the future.

Algorithms		`small MNL-CLT`	`MNL-CLT`
BFGS	IOA	118	127
	Simple	106.77 ± 3.91	107.55 ± 3.17
	Complex	**101.11 ± 2.94**	**105.50 ± 2.96**
Newton	IOA	40	38
	Simple	30.40 ± 1.39	30.64 ± 1.47
	Complex	**30.17 ± 1.10**	**30.51 ± 1.62**
Trust-Region	IOA	20	20
	Simple	11.65 ± 1.92	10.38 ± 2.31
	Complex	**10.35 ± 0.95**	**9.17 ± 1.41**

Table 4: Number of epochs required to optimize the last two models. The numbers corresponds to the average value and the standard deviation over 20 optimization process. The bold values indicates the best set of parameters.

5.3 Parameters study

Now that we have shown that the WMA-ABS algorithm works well and is able to optimize faster than its counterpart, we need to have a look at the parameters. Indeed, it would be counterproductive to spend a lot of time finding the right set of parameters to optimize the model 20% faster. We would thus prefer to use generic parameters that would allow us to optimize the model faster, even if it is slower than with optimized parameters. As shown in Table 3, the models `MNL-SM` and `LogReg-BS` have been compared with optimal set of parameters. These optimal values are given in Figure 5. At first, we can see that these optimum parameters are quite different from the suggested one except for C. W is the parameter with the most deviation. However, it seems that if it takes a value close or bigger than 10 is optimal. Interestingly, the threshold Δ takes values a bit higher than the one suggested, same for the factor τ. Since these models are simple with a global optimum, there is no need to spend a lot of time in the small batch size. Indeed, if we can quickly get closer to the solution and then go to the full batch, it will be faster. In addition, it seems that the algorithm is confident that the improvement of the objective function is less highly correlated to the noise of the stochasticity, thus the high value for the threshold.

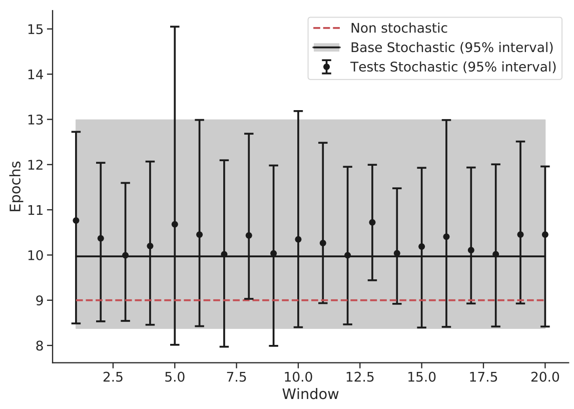
Model	Algorithm	W	Δ [%]	C	τ
`MNL-SM`	BFGS	29	3.0	1	1.8
	NM	18	4.4	1	6.2
	Trust-Region	9	3.9	1	5.1
`LogReg-BS`	BFGS	9	5.1	2	5.5
	NM	19	2.4	4	5.6
	Trust-Region	30	7.9	1	4.6

Table 5: Optimized parameters found using the package `hyperopt` for the models `MNL-SM` and `LogReg-BS` optimized by Newton method, BFGS, and Trust-Region.

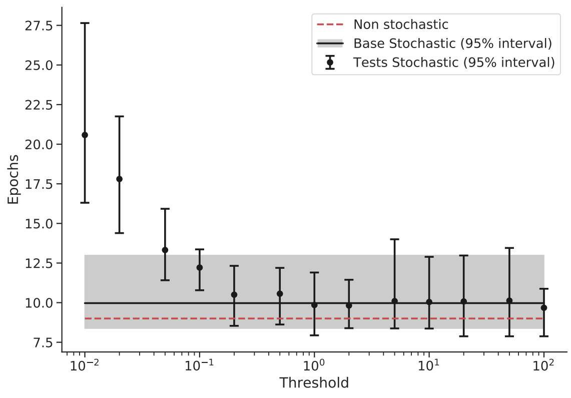
By analyzing the values in Table 5, we could think that the suggested parameters are not good enough. However, as shown in Table 3, the improvement with the optimized parameters is quite low compared to the set of parameters for simple models, except for the algorithm Trust-Region. Nonetheless, it is important to understand the effect of the parameters on the speed of the optimization of these models. As explained earlier, we do not want to find and use the optimal parameters. We thus decided to start with the suggested parameters and explore from this starting set. The idea is to vary each parameter while the others are fixed to identify which parameters are the most important. We started with the model `MNL-SM` and the Newton method. The results are similar for the other algorithms. Results are given in Figure 8.

The first interesting parameter is the Window. Indeed, we see in Figure 7(a) that this parameter does not influence the number of epochs required. Thus, we can use any value for this parameter. We recommended here to use the value 10 which is big enough to smooth the objective function. The threshold, on the other hand, cannot be too small, see Figure 7(b). This is expected since we are waiting too long that the algorithm is close to convergence. The primary goal of the algorithm is to advance quickly at the beginning. It is thus required to get closer to the optimum but quickly switch to higher batch size. The count has an interesting linear relationship in Figure 7(c). For such a simple model, the lack of improvement is mostly coming from the algorithm struggling to optimize. Thus, we can confidently update the batch size when the improvement is low enough. Thus, the best value is definitely 1. For the last parameter, we can see that the factor has to be bigger or equal to 2, see Figure 7(d). While it is obvious that a factor too small will slow the optimization process, it is quite interesting to see that using a value bigger than 2 for the factor is not significantly improving the speed of the optimization. We did the same study with the model `LogReg-BS` and the BFGS algorithms. The results are given in Figure 9. For this model, we also used the suggested set of parameters for a simple model since it is quite small.

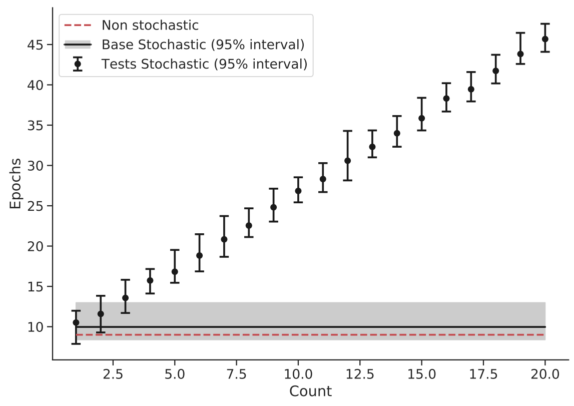
((a)) Window W



((b)) Threshold Δ



((c)) Count C



((d)) Factor τ

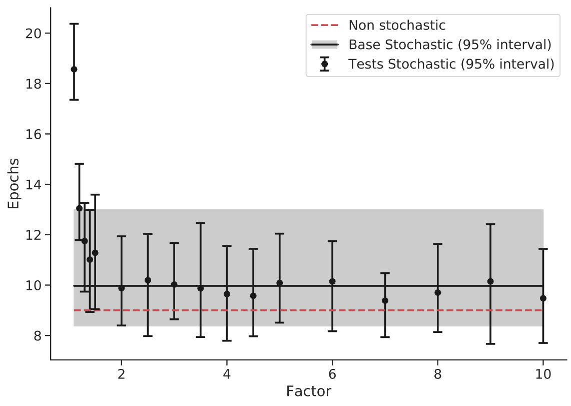


Figure 8: Evaluation of different parameters values for the model `MNL-SM` and the Newton method. The other parameters were set to the values suggested for simple models. Each set of parameters was used to optimize the model 20 times. The red dotted line corresponds to the optimization with the non-stochastic version of the algorithm. The gray area corresponds to the 95% interval trained with the suggested parameters.

By comparing Figures 8 and 9, we see that the curves have not changed for the parameters W and Δ. Indeed, the window seems that it does not have any influence on the speed of the optimization. However, it is possible that it has more importance for bigger models such as the model `MNL-CLT`. Nevertheless, using a value of 10 does not harm the speed of the optimization and it will surely help quite a lot when the objective function becomes noisier. For the threshold, the results simply indicate that we need to use a value that is not too small. Higher values do not show any significant improvement in the speed of the optimization process. Thus, 1% seems to be a reasonable choice in this case. For the factor, shown in Figure 8(d), the results are different from the model `MNL-SM`. Indeed, it seems that using a bigger factor tends to make the optimization slower. This is surely due to the fact that the algorithm is not exploiting the gain of speed due to the use of small batch size. It is thus better to use a slightly smaller value for the factor. Looking at the results, a value of around 1.5-2 would be the best, thus supporting the

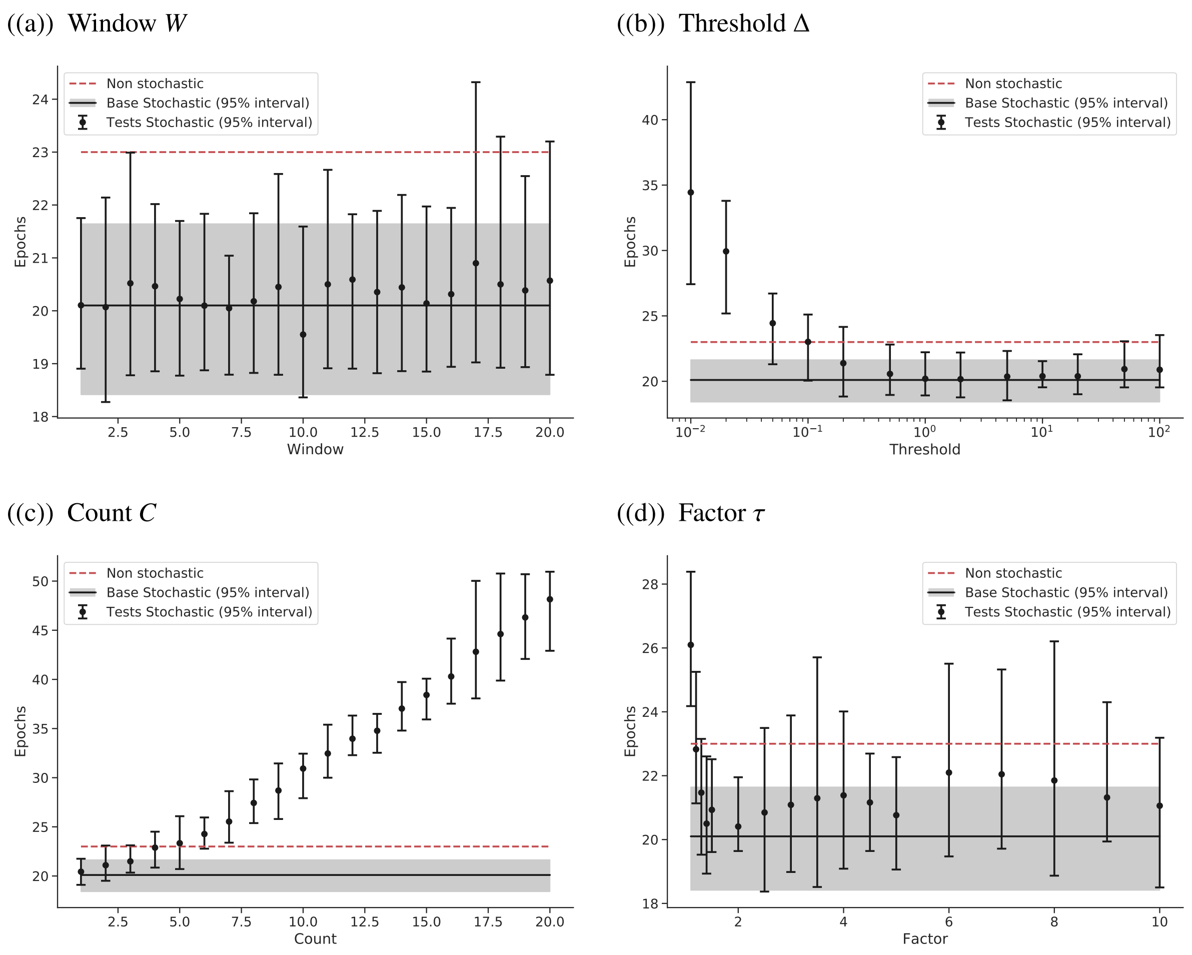


Figure 9: Evaluation of different parameters values for the model `LogReg-BS` and the BFGS algorithm. The other parameters were set to the values suggested for simple models. Each set of parameters was used to optimize the model 20 times. The red dotted line corresponds to the optimization with the non-stochastic version of the algorithm. The gray area corresponds to the 95% interval trained with the suggested parameters.

suggested value. Finally, for the parameter count, we see that the linear relationship is slowly fading for the small values as shown in Figure 8(c). Indeed, for higher values of the count, we keep the same linear relationship between the value of this parameter and the speed of the optimization. However, for lower values, the curve is slightly flattened. It thus suggests us that a higher value for this parameter is required for a more complex model as suggested when presenting the parameters.

6 Conclusion & Future Work

In this paper, we have presented an adaptive batch size algorithm called WMA-ABS. The central idea is to look at the improvement and increase the batch size when the improvement is too low using some smoothing technique to be more accurate. We show that it works well with different second-order IOAs and quasi-Newton method. We have confirmed that the use of stochastic algorithms is more interesting for models having a larger data set and more parameters. While the WMA-ABS algorithm has some parameters that can be optimized, it can also be used with the recommended sets of parameters given in Table 1 and show satisfying results. This algorithm has shown up to 50% of speed up in the optimization process with the standard parameters. It also shows that the more complex a model, the most improvement it can bring.

For future research, we can perform more tests on many different models and data sets. The adaptive batch size could also be used in addition to other improvements found in the literature. Secondly, we would like to update the WMA-ABS method such that it has more control over the size of the batch. Currently, it can only update it by the factor τ predefined in the parameters. However, it would be interesting to define heuristics able to automatically choose the next batch size, being either bigger or smaller.

7 References

Agarwal, N., B. Bullins and E. Hazan (2016) Second-Order Stochastic Optimization for Machine Learning in Linear Time, *arXiv:1602.03943 [cs, stat]*, February 2016. ArXiv: 1602.03943.

Balles, L., J. Romero and P. Hennig (2016) Coupling Adaptive Batch Sizes with Learning Rates, *arXiv:1612.05086 [cs, stat]*, December 2016. ArXiv: 1612.05086.

Bergstra, J., B. Komer, C. Eliasmith, D. Yamins and D. D. Cox (2015) Hyperopt: a Python library for model selection and hyperparameter optimization, *Computational Science & Discovery*, **8** (1) 014008, ISSN 1749-4699.

Bierlaire, M. (2015) *Optimization: Principles and Algorithms*, EPFL Press, ISBN 978-2-940222-78-0.

Bierlaire, M. (2018) Pandasbiogeme: a short introduction, *Technical Report*.

Bierlaire, M., K. Axhausen and G. Abay (2001) The acceptance of modal innovation: The case of Swissmetro, *Swiss Transport Research Conference 2001*, March 2001.

Bollapragada, R., D. Mudigere, J. Nocedal, H.-J. M. Shi and P. T. P. Tang (2018) A Progressive Batching L-BFGS Method for Machine Learning, *arXiv:1802.05374 [cs, math, stat]*, February 2018. ArXiv: 1802.05374.

Bordes, A., L. Bottou and P. Gallinari (2009) SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent, *J. Mach. Learn. Res.*, **10**, 1737–1754, December 2009, ISSN 1532-4435.

Bordes, A., L. Bottou, P. Gallinari, J. Chang and S. A. Smith (2010) Erratum: SGDQN is Less Careful than Expected, *Journal of Machine Learning Research*, **11** (Aug) 2229–2240, ISSN ISSN 1533-7928.

Brathwaite, T., A. Vij and J. L. Walker (2017) Machine Learning Meets Microeconomics: The Case of Decision Trees and Discrete Choice, *arXiv:1711.04826 [stat]*, November 2017. ArXiv: 1711.04826.

Byrd, R., G. Chin, W. Neveitt and J. Nocedal (2011) On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning, *SIAM Journal on Optimization*, **21** (3) 977–995, July 2011, ISSN 1052-6234.

Cauchy, A. (1847) Mã©thode gã©nã©rale pour la rã©solution des systemes dââ©quations simultanã©es, *Comp. Rend. Sci. Paris*, **25** (1847) 536–538.

Defazio, A., F. Bach and S. Lacoste-Julien (2014) SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives, in Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence and K. Q. Weinberger (eds.) *Advances in Neural Information Processing Systems 27*, 1646–1654, Curran Associates, Inc.

Devarakonda, A., M. Naumov and M. Garland (2017) AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks, *arXiv:1712.02029 [cs, stat]*, December 2017. ArXiv: 1712.02029.

Dozat, T. (2016) Incorporating Nesterov Momentum into Adam, February 2016.

Duchi, J., E. Hazan and Y. Singer (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine Learning Research*, **12** (Jul) 2121–2159, ISSN ISSN 1533-7928.

Fletcher, R. (1987) *Practical Methods of Optimization; (2Nd Ed.)*, Wiley-Interscience, New York, NY, USA, ISBN 978-0-471-91547-8.

Gower, R. M., F. Hanzely, P. RichtÃ¡rik and S. Stich (2018) Accelerated Stochastic Matrix Inversion: General Theory and Speeding up BFGS Rules for Faster Second-Order Optimization, *arXiv:1802.04079 [cs, math]*, February 2018. ArXiv: 1802.04079.

Gurney, K. (2014) *An introduction to neural networks*, CRC press.

Hillel, T. (2019) Understanding travel mode choice: A new approach for city scale simulation, Ph.D. Thesis, University of Cambridge, Cambridge, January 2019.

Hillel, T., M. Z. E. B. Elshafie and Y. Jin (2018) Recreating passenger mode choice-sets for transport simulation: A case study of London, UK, *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, **171** (1) 29–42, March 2018.

Keskar, N. S. and A. S. Berahas (2016) adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs, paper presented at the *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, 1–16, September 2016, ISBN 978-3-319-46127-4 978-3-319-46128-1.

Kingma, D. P. and J. Ba (2014) Adam: A Method for Stochastic Optimization, *arXiv:1412.6980 [cs]*, December 2014. ArXiv: 1412.6980.

Kiros, R. (2013) Training Neural Networks with Stochastic Hessian-Free Optimization, *arXiv:1301.3641 [cs, stat]*, January 2013. ArXiv: 1301.3641.

Lederrey, G., V. Lurkin and M. Bierlaire (2018) SNM: Stochastic Newton Method for Optimization of Discrete Choice Models, *IEEE - ITSC'18*, November 2018.

Martens, J. (2010) Deep learning via Hessian-free optimization, *ICML*, **27**, 735–742.

Mokhtari, A. and A. Ribeiro (2014) RES: Regularized Stochastic BFGS Algorithm, *IEEE Transactions on Signal Processing*, **62** (23) 6089–6104, December 2014, ISSN 1053-587X.

Nesterov, Y. E. (1983) A method for solving the convex programming problem with convergence rate O (1/k^ 2), paper presented at the *Dokl. Akad. Nauk SSSR*, vol. 269, 543–547.

Newman, J. P., V. Lurkin and L. A. Garrow (2018) Computational methods for estimating multinomial, nested, and cross-nested logit models that account for semi-aggregate data, *Journal of Choice Modelling*, **26**, 28–40, March 2018, ISSN 1755-5345.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay (2011) Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, **12**, 2825–2830.

Polyak, B. and A. Juditsky (1992) Acceleration of Stochastic Approximation by Averaging, *SIAM Journal on Control and Optimization*, **30** (4) 838–855, July 1992, ISSN 0363-0129.

Polyak, B. T. (1964) Some methods of speeding up the convergence of iteration methods, *USSR Computational Mathematics and Mathematical Physics*, **4** (5) 1–17.

Reddi, S. J., S. Kale and S. Kumar (2018) On the Convergence of Adam and Beyond, February 2018.

Ruder, S. (2016) An overview of gradient descent optimization algorithms, *arXiv:1609.04747 [cs]*, September 2016. ArXiv: 1609.04747.

Schmidt, M., N. L. Roux and F. Bach (2013) Minimizing Finite Sums with the Stochastic Average Gradient, *arXiv:1309.2388 [cs, math, stat]*, September 2013. ArXiv: 1309.2388.

Shallue, C. J., J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig and G. E. Dahl (2018) Measuring the Effects of Data Parallelism on Neural Network Training, *arXiv:1811.03600 [cs, stat]*, November 2018. ArXiv: 1811.03600.

Steihaug, T. (1983) The Conjugate Gradient Method and Trust Regions in Large Scale Optimization, *SIAM Journal on Numerical Analysis*, **20** (3) 626–637, June 1983, ISSN 0036-1429.

Tieleman, T. and G. Hinton (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, *COURSERA: Neural networks for machine learning*, **4** (2) 26–31.

Wang, X., S. Ma and W. Liu (2014) Stochastic Quasi-Newton Methods for Nonconvex Stochastic Optimization, *arXiv:1412.1196 [math]*, December 2014. ArXiv: 1412.1196.

Zeiler, M. D. (2012) ADADELTA: An Adaptive Learning Rate Method, *arXiv:1212.5701 [cs]*, December 2012. ArXiv: 1212.5701.