SNM: Stochastic Newton Method for Optimization of Discrete Choice Models

Gael Lederrey
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
gael.lederrey@epfl.ch

Virginie Lurkin
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
virginie.lurkin@epfl.ch

Michel Bierlaire
Transport and Mobility Laboratory
École Polytechnique Fédérale de Lausanne
Station 18, CH-1015 Lausanne
michel.bierlaire@epfl.ch

***Abstract*—Optimization algorithms are rarely mentioned in the discrete choice literature. One reason may be that classic Newton-Raphson methods have been rather successful in estimating discrete choice parameters on available data sets of limited size. However, thanks to recent advances in data collection, abundant data about choice situations become more and more available and state-of-the art algorithms can be very computationally burdensome on these massive datasets. In this paper, inspired by the good practices from the machine learning field, we introduce a Stochastic Newton Method (SNM) for the estimation of discrete choice models parameters. Our preliminary results indicate that our method outperforms (stochastic) first-order and quasi-newton methods.**

***Index Terms*—Discrete Choice Models, Optimization**

I. Introduction

Discrete choice models (DCM) have become an essential operational tool in modeling individual behavior. Many success stories have been reported in the scientific studies related to the transportation field. Estimating those models requires to solve an optimization problem and yet, optimization algorithms are rarely mentioned in the discrete choice literature. One reason may be that classic nonlinear programming algorithms (i.e., Newton-Raphson method) have been rather successful in estimating discrete choice parameters on available data sets of limited size. Thanks to recent advances in data collection, abundant data about choice situations become more and more available. While offering a grand potential for a better understanding of human choices, these new data sources also brings new challenges for the community. Indeed algorithms classically embedded in state-of-the art discrete choice software's (such as Biogeme [**?**] or Larch [**?**]) can be very computationally burdensome on these massive datasets.

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 involves 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 are able to provide faster convergence.

For the sake of interpretability, discrete choice models usually have a much 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 from the ML field, we introduce a Stochastic Newton Method (SNM) for the estimation of discrete choice models parameters. We investigate the convergence of our algorithm by benchmarking it against standard first-order methods and quasi-newton methods using a simple multinomial logit model on a small data set. We present preliminary results that indicate that our method outperforms (stochastic) first-order and quasi-newton methods. However, the algorithm developed in this paper only constitutes a first step toward our final goal that is the development of an optimization method specifically designed to estimate discrete choice model parameters on big data sets. Further modifications are currently investigated to achieve faster convergence and to be able to deal with large data sets.

The remainder of this paper is structured as follows. In section II, we present some related works about first and second-order optimization methods. In Section III, we describe the discrete choice model we use; the Stochastic Newton Method algorithm that we propose to estimate its parameters, as well as the first-order methods and quasi-newton methods that we use as benchmark to evaluate our algorithm. Section IV shows the results and we present our concluding remarks and future works in section V.

II. Related Work

Optimization plays a key role in Machine Learning, especially for developing efficient and scalable algorithms suited for large datasets and complex models. In this respect, first-order methods have been extensively explored, leading to many variants of the popular Stochastic Gradient Descent (SGD) algorithm. A well-known issue of standard first-order methods is that they tend to struggle when the curvature of the objective function is not homogeneous [**?**].

To remedy this situation, a momentum term is usually introduced in SGD algorithms (see [**?**]). Momentum-based methods help accelerate gradients vectors in the right directions, leading to faster converging. Other first-order methods adapt the step size (or learning rate) to the parameters, such as Adagrad [?]. Adding a momentum and adapting the learning rate are probably the two most popular extension of the classical SGD algorithm. Then comes an iterative process between researchers trying to improve previous algorithms. Ruder [?] gives a good overview of first-order methods, from SGD up to complex and recent first-order algorithms such as Nadam [?] or AMSGrad [?].

More recently, thanks to the growth in computing power, researchers have went beyond the first-order methods to consider quasi-newton methods that use information from the curvature. The key motivation behind quasi-Newton methods is the desire to use something like Newton's method for its speed but without having to compute the exact second derivatives each time. The central idea is therefore to build up, iteratively, a good approximation of the Hessian matrix from the function and gradient values computed at previous step(s). The renowned BFGS algorithm [?] belongs to the family of quasi-newton methods. With the motivation to incorporate the curvature information for problems of big size, much progress have been made lately toward developing Stochastic BFGS algorithms such as RES-BFGS [?], a regularized stochastic BFGS. Nowadays, many researchers are trying to make use of the structure of the problem to find alternative versions of a given algorithm to perform better on this specific problem. For example, Gower *et al.* [?] have implemented an alternative version of BFGS for Matrix Inversion. Keskar *et al.* [?] have implemented adaQN, an adaptative quasi-newton method that is specifically designed for training Recurrent Neural Networks. Some researchers, such as Ye and Zhang [?], have got inspiration from the progress on first-order methods to improve second-order methods and Byrd *et al.* [?] have proposed to make use of Conjugate Gradient and stochasticity to create more effective algorithms. In definitive, the most advanced and recent methods are all based on quasi-newton methods (see e.g. [?], [?], [?], [?]), while very little work has been done regarding the second-order methods.

III. Methodology

In this section, we present our Stochastic Newton Method. Before presenting the algorithm, we introduce the choice model used to evaluate the performance of our method[1].

A. Model

We use the *Swissmetro* dataset [?] and build a multinomial logit model denoted by $\mathcal{M}$:

$$\begin{aligned} V_{\text{Car}} &= \text{ASC}_{\text{Car}} + \beta_{\text{TT,Car}}\text{TT}_{\text{Car}} + \beta_{\text{C,Car}}\text{C}_{\text{Car}} + \beta_{\text{Senior}}\mathbb{1}_{\text{Senior}} \\ V_{\text{SM}} &= \text{ASC}_{\text{SM}} + \beta_{\text{TT,SM}}\text{TT}_{\text{SM}} + \beta_{\text{C,SM}}\text{C}_{\text{SM}} \\ &\quad + \beta_{\text{HE}}\text{HE}_{\text{SM}} + \beta_{\text{Senior}}\mathbb{1}_{\text{Senior}} \\ V_{\text{Train}} &= \text{ASC}_{\text{Train}} + \beta_{\text{TT,Train}}\text{TT}_{\text{Train}} + \beta_{\text{C,Train}}\text{C}_{\text{Train}} + \beta_{\text{HE}}\text{HE}_{\text{Train}} \end{aligned} \tag{1}$$

[1]The code is on github: https://github.com/glederrey/IEEE2018_SNM.

	Value	Std err	t-test	p-value
ASC_{Car}	0	-	-	-
ASC_{SM}	$7.86 \cdot 10^{-1}$	$6.93 \cdot 10^{-2}$	11.35	0.00
$\text{ASC}_{\text{Train}}$	$9.83 \cdot 10^{-1}$	$1.31 \cdot 10^{-1}$	7.48	0.00
$\beta_{\text{TT,Car}}$	$-1.05 \cdot 10^{-2}$	$7.89 \cdot 10^{-4}$	-8.32	0.00
$\beta_{\text{TT,SM}}$	$-1.44 \cdot 10^{-2}$	$6.36 \cdot 10^{-4}$	-21.29	0.00
$\beta_{\text{TT,Train}}$	$-1.80 \cdot 10^{-2}$	$8.65 \cdot 10^{-4}$	-20.78	0.00
$\beta_{\text{C,Car}}$	$-6.56 \cdot 10^{-3}$	$7.89 \cdot 10^{-4}$	-8.32	0.00
$\beta_{\text{C,SM}}$	$-8.00 \cdot 10^{-3}$	$3.76 \cdot 10^{-4}$	-21.29	0.00
$\beta_{\text{C,Train}}$	$-1.46 \cdot 10^{-2}$	$9.65 \cdot 10^{-4}$	-15.09	0.00
β_{Senior}	-1.06	$1.16 \cdot 10^{-1}$	-9.11	0.00
β_{HE}	$-6.88 \cdot 10^{-3}$	$1.03 \cdot 10^{-3}$	-6.69	0.00

TABLE I
Parameters of the optimized model $\mathcal{M}$ by Biogeme.

where $\mathbb{1}_{\text{Senior}}$ is a boolean variable equal to one if the age of the respondent is over 65 years olds, 0 otherwise, C denotes the cost, TT the travel time, and HE the headway for the train and Swissmetro. For this model, we removed all observations with unknown choice, unkown age and non-positive travel time. This gives a total of 9,036 observations.

We first estimate the model with Biogeme [?] to obtain the optimal parameter values and verify that all parameters are significant. However, we do not use the usual log-likelihood. Instead, we are using a normalized log-likelihood which simply corresponds to the log-likelihood divided by the number of observations. Therefore, the final normalized log-likelihood is -0.7908 and the parameters are given in Table I.

We also provide a normalized model $\bar{\mathcal{M}}$ where the values of travel time, cost, and headway have been divided by 100. The parameters for this normalized model are the same as model $\mathcal{M}$ except that the values of parameters associated to the features normalized are multiplied by 100. The reason behind this normalization is to have parameters close to each other, *i.e.* in the same order of magnitude, as opposed to the values in Table I where the parameter values are in four orders of magnitude.

B. Stochastic Newton Algorithm (SNM)

As already mentioned, for Neural Networks, the number of features K can easily exceed one million, leading to a huge Hessian matrix (composed of K^2 elements). Discrete Choice Models, on the other hand, tend to have a much more reasonable number of parameters since maintaining interpretability is crucial. It follows therefrom that the primary limitation of Newton methods encountered in Neural Networks is not valid for Discrete Choice models. Yet, one problem remains: the exponential growth of data. Indeed, computing the Hessian on many data can be as tedious as computing it for many features. Hence the motivation to develop our Stochastic Newton Method (SNM).

The central idea behind our algorithm is to compute a stochastic Hessian. We show below that computing a stochastic Hessian is straightforward for a Logit model. The generalization can be applied to any finite-sum function as such as the log-likelihood of a Logit model. Let N denote the number of individuals, $\mathcal{C}$ denote the choice set and $\mathcal{C}_n$ denote the choice set, i.e. the list of available aterlatives for individual n and define

$$y_{in} = \begin{cases} 1 & \text{if individual } n \text{ chose alternative } i, \\ 0 & \text{otherwise.} \end{cases}$$

The likelihood function for such choice model is given by

$$\mathcal{L}^* = \prod_{n=1}^{N} \prod_{i \in \mathcal{C}_n} P_n(i)^{y_{in}}. \tag{2}$$

where $P_n(i)$ denotes the probability that individual n chooses alternative i. For a Logit model, this probability is given by

$$P_n(i) = \frac{e^{V_{in}}}{\sum_{j \in \mathcal{C}_n} e^{V_{jn}}}. \tag{3}$$

where V_{in} denotes the utility of alternative i for individual n. If we take the logarithm of Equation (3), we get the log-likelihood:

$$\begin{aligned} \mathcal{L} &= \sum_{n=1}^{N} \sum_{i \in \mathcal{C}_n} y_{in} \left(V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right) \\ &= \sum_{n=1}^{N} \left(\sum_{i \in \mathcal{C}_n} y_{in} V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right). \end{aligned} \tag{4}$$

The second equality is done using the fact that $\sum_{i \in \mathcal{C}_n} y_{in} = 1$. We then update the log-likehood of Equation (4) to create a normalized log-likelihood.

$$\bar{\mathcal{L}} = \frac{1}{N} \mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{i \in \mathcal{C}_n} y_{in} V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right). \tag{5}$$

This is done such that the value of the log-likelihood stays in the same magnitude of order for any subset of observations $\mathcal{I}$. Indeed, if we denote $\mathcal{L}_{\mathcal{I}}$ the log-likelihood computed on the observations from $\mathcal{I}$ and $\mathcal{N}$ the set of all observations, we see that

$$\begin{aligned} \mathcal{L}_{\mathcal{I}} &= \sum_{n \in \mathcal{I}} \left(\sum_{i \in \mathcal{C}_n} y_{in} V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right) \\ &< \sum_{n \in \mathcal{I}} \left(\sum_{i \in \mathcal{C}_n} y_{in} V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right) \\ &\quad + \sum_{n \in \mathcal{N} \setminus \mathcal{I}} \left(\sum_{i \in \mathcal{C}_n} y_{in} V_{in} - \ln \sum_{j \in \mathcal{C}_n} e^{V_{jn}} \right) \\ &= \mathcal{L}. \end{aligned} \tag{6}$$

As shown in Equation (6), the standard log-likelihood cannot be compared on different set of data if they don't contain the same number of data. However, normalizing the log-likelihood, as done in Equation (5), produces log-likelihoods of same order of magnitude, independently of the number of observations in the subset.

The first derivatives of $\bar{\mathcal{L}}$ with respect to parameter $k = 1, \ldots, K$ are given by

$$\begin{aligned} \frac{\partial \bar{\mathcal{L}}}{\partial \beta_k} &= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{i \in \mathcal{C}_n} y_{in} \frac{\partial V_{in}}{\partial \beta_k} - \sum_{i \in \mathcal{C}_n} 1 P_n(i) \frac{\partial V_{in}}{\partial \beta_k} \right) \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{i \in \mathcal{C}_n} (y_{in} - P_n(i)) \frac{\partial V_{in}}{\partial \beta_k}. \end{aligned} \tag{7}$$

The second derivatives for $k = 1, \ldots, K$ and $l = 1, \ldots, K$ are given by

$$\frac{\partial^2 \bar{L}}{\partial \beta_k \partial \beta_l} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i \in \mathcal{C}_n} P_n(i) W_{ink} W_{inl}, \tag{8}$$

where

$$W_{ink} = \left(\frac{\partial V_{in}}{\partial \beta_k} - \sum_{j \in \mathcal{C}_n} \frac{\partial V_{jn}}{\partial \beta_k} P_n(j) \right).$$

From the definition of the second derivatives in Equation (8), it is straightforward to compute the second derivative for one single observation o.

$$\left. \frac{\partial^2 \bar{L}}{\partial \beta_k \partial \beta_l} \right|_m = -\sum_{i \in \mathcal{C}_o} P_o(i) W_{iok} W_{iol} \tag{9}$$

From the definitions in Equations (8) and (9), we conclude that the Hessian on a subset of observations $\mathcal{I}$ is simply the average of the Hessians obtained for each observation $i \in \mathcal{I}$.

Algorithm 1 shows our Stochastic Newton Method (SNM). The computation of both the stochastic gradient and the stochastic Hessian are done in lines 9 and 10. One particular feature of this algorithm is the computation of the direction for the next step. Indeed, with small batches, the Hessian may be singular. For example, it could happen that a variable associated with a parameter β_k is always equal to 0 for a small batch (especially for binary variables). Therefore, the derivative of V_{in} by β_k will always be zero. Therefore, the row and column of the Hessian will be both zero for this particular parameter, making it singular. The countermeasure to this issue is to test if the Hessian is singular or not. If it is not the case, then the algorithm performs a standard Newton step with the stochastic Hessian and gradient. However, if the Hessian is singular, the algorithm performs a Stochastic Gradient Descent (SGD) step. Concerning the choice of the step size, for a given objective function, it often differs between SGD and Newton Method. Therefore, we have two possibilities: the algorithm should use two different step sizes, or we can perform a line search.

IV. Results

In this section, we show the result achieved by our algorithm, together with different benchmark algorithms. We also highligh a current main weakness of our SNM and a future way to fix it.

Algorithm 1 Stochastic Newton Method (SNM)

```
Input: Starting parameter value ($\theta_0$), data ($\mathcal{D}$), function ($f$), gradient ($\nabla f$), Hessian ($\nabla^2 f$), number of epochs ($n_{ep}$), batch size ($n_{batch}$)
Output: Epochs ($e$), parameters ($\theta$), function values ($f_v$)
 1: function SNM
 2:     $(n_\mathcal{D}, m) = |\mathcal{D}|$                                  ▷ Number of samples and parameters
 3:     $n_{iter} \leftarrow \lceil n_{ep} n_\mathcal{D} / n_{batch} \rceil$                ▷ Number of iterations
 4:     Initialize $e$, $\theta$ and $f_v$. Set $\theta[0] \leftarrow \theta_0$
 5:     for $i = 0 \dots n_{iter}$ do
 6:         $e[i] \leftarrow i \cdot n_{batch} / n_\mathcal{D}$                       ▷ Store the epoch
 7:         $f_v[i] \leftarrow f(\theta[i])$                                  ▷ Store the function value
 8:         idx $\leftarrow n_{batch}$ values from $\mathcal{U}(0, n_\mathcal{D})$ without replacement
 9:         grad $\leftarrow \nabla f_{\texttt{idx}}(\theta[i])$                        ▷ Gradient on the samples from idx
10:         hess $\leftarrow \nabla^2 f_{\texttt{idx}}(\theta[i])$                      ▷ Hessian on the samples from idx
11:         if hess is non singular then
12:             inv_hess $\leftarrow$ hess$^{-1}$
13:             step $\leftarrow -$grad $\cdot$ inv_hess
14:         else
15:             step $\leftarrow$ grad
16:         $\alpha \leftarrow$ Backtracking Line Search with step on the subset of data with indices from idx
17:         $\theta[i+1] \leftarrow \theta[i] + \alpha \cdot$ step
18:     $e[n_{iter}] \leftarrow n_{iter} \cdot n_{batch} / n_\mathcal{D}$
19:     $f_v[n_{iter}] \leftarrow f(\theta[n_{iter}])$
20:     return $e$, $\theta$ and $f_v$
```

A. Benchmark algorithms

We use several algorithms to train models $\mathcal{M}$ and $\bar{\mathcal{M}}$. These algorithms fall into three different categories: first-order methods, second-order methods, and quasi-newton methods. For first-order methods, we use mini-batch SGD [**?**] and Adagrad [**?**]. For the quasi-newton methods, we use BFGS algorithm [**?**] and RES-BFGS [**?**]. The main second-order algorithm is the Newton method [**?**]. Finally, to avoid the long and tedious search of a good step size, we run all algorithms presented above with a backtracking Line Search method using the Armijo-Goldstein condition [**?**].

B. Raw data vs Normalized data

Most of the data we can obtain are not normalized. This is often a preprocessing step required for some optimization algorithm to work. As explained in Section III-A, the optimization of the model leads to optimized parameters ranging over four orders of magnitude. Since the step size is the same for all parameters, it is difficult to find an optimal step size. Figure 1(a) and 1(b) show the optimization process of the log-likelihood for SGD and Adagrad, respectively, for the raw model $\mathcal{M}$ and the normalized model $\bar{\mathcal{M}}$. For both algorithms, the optimization was done ten times for ten epochs with a batch size of 100 observations. The lines correspond to the average while the colored part corresponds to the 95% confidence interval. The results show that these algorithms perform better on the normalized model $\bar{\mathcal{M}}$. Table II show the average value of the log-likelihood after two epochs for these two algorithms on both models.

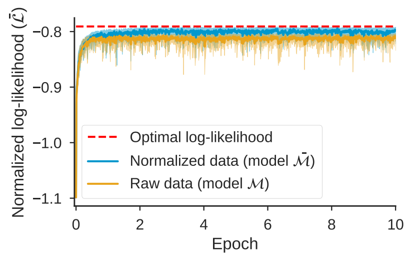
	SGD	**Adagrad**	**SNM**
on $\mathcal{M}$	-0.813107	-0.812080	-0.794219
on $\bar{\mathcal{M}}$	-0.801739	-0.801646	-0.794219
rel. diff.	1.42%	1.30%	0.00%

TABLE II
AVERAGE NORMALIZED LOG-LIKELIHOOD OVER A THOUSAND RUNS AT THE SECOND EPOCH FOR SGD, ADAGRAD AND SNM.

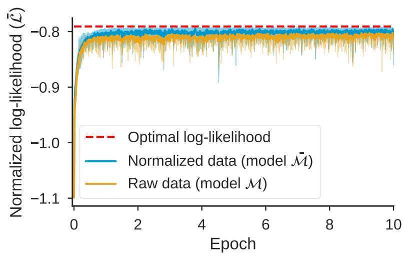
Figure 1(c) shows the results of the training on both models with SNM. We ran this algorithm with batches of 1,000 observations. In Sections IV-C and IV-D, we explain why we had to use a batch size with more observations. The qualitative results as well as the quantitative results in Table II show that second-order methods have less problem with badly conditioned optimization problem. Thus, it indicates that the information contained in the Hessian is important when the problem is ill-conditioned.

C. Comparison of the algorithms

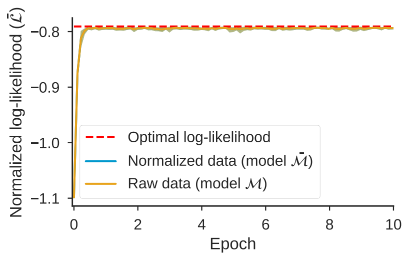
In this section, we want to compare the three main categories of algorithms: first-order methods, quasi-newton methods, and second-order methods. Figure 2(a) shows the results for SGD with batch size of 100 and 1,000 as well as gradient descent. Figure 2(b) shows the results for RES-BFGS with batch sizes of 100 and 1,000 as well as standard BFGS. Finally, Figure 2(c) shows the results for SNM with batch sizes of 100 and 1,000 as well as Newton method. For these three figures, we executed ten runs. Again, the lines give the average value for the normalized log-likelihood, and the



(a) SGD



(b) Adagrad



(c) SNM

Fig. 1. Evaluation of the algorithms on raw data (model $\mathcal{M}$) and normalized data (model $\bar{\mathcal{M}}$). The vertical axis corresponds to the normalized log-likelihood presented in Equation (5). Each time, ten runs have been executed. The lines correspond to the average value over all the runs and the colored part correspond to the 95% confidence interval. SGD and Adagrad are run with a batch size of 100 observations, SNM is run with a batch size of 1,000 observations.

	batch	**first-order**	**quasi-newton**	**second-order**
Stochastic	100	-0.801452	-0.796492	-0.828409
	1000	-0.812886	-0.837937	-0.794219
Full batch size		-0.891737	-0.963458/-0.806245	-0.798112

TABLE III
AVERAGE NORMALIZED LOG-LIKELIHOOD OVER A THOUSAND RUNS AT THE SECOND EPOCH FOR FIRST-ORDER METHODS, QUASI-NEWTON METHODS AND SECOND-ORDER METHODS.

colored parts show the 95% confidence interval.

From Figure 2, we see that first-order methods are the furthest from the optimal value. Then, we see that stochastic quasi-newton methods tend to struggle to reach the optimal value, especially with the first approximation of the Hessian being the identity matrix. Interestingly, we see that the RES-BFGS works better with smaller batch size while it tends to struggle and plateau with big batch size. Nevertheless, it can get closer to the optimal solution than SGD. However, SNM is the best algorithm out of the three. Table III gives the average value of the normalized log-likelihood for the second epoch. In this table, we report two values for the quasi-newton method and the full batch size: the first value reported is with the first approximation of the Hessian being the identity matrix, the second value corresponds to the real Hessian. The numbers confirm that SNM is the best algorithm. However, it is interesting to note that contrary to the other two algorithms, SNM runs better with bigger batch size. In the next section, see Section IV-D, we study the possible reason behind such behavior.

D. *Effect of the batch size*

As shown in Figure 2, SNM is the only algorithm for which a more significant batch size works better. This behavior is quite odd, and the explanation may come from the direction. Indeed, as explained in Section III-B, the direction is either a gradient step or a Newton step, depending on the singularity of the Hessian. Therefore, we are interested to know if this direction, *i.e.* the choice between a gradient step and a Newton step, depends on the batch size. In Figure 3, we show the percentage of Newton step that the algorithm is capable of performing in function of the batch size. This percentage is computed on a thousand draws.

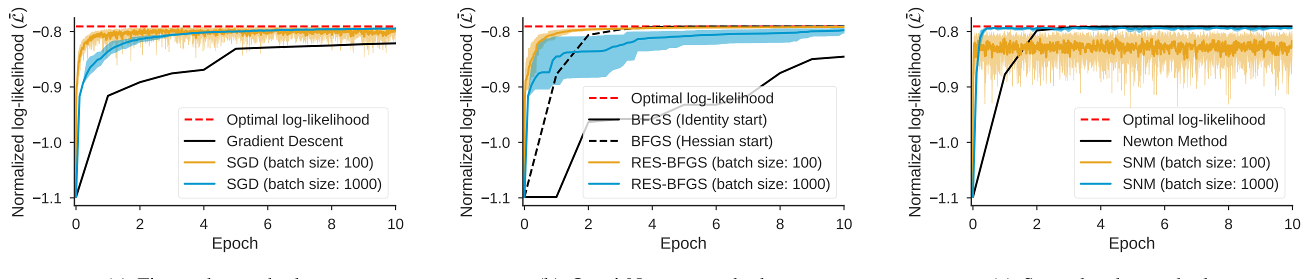
The algorithm is only capable of performing a Newton step if the Hessian is non-singular. If we take a look at model $\mathcal{M}$ in Equation 1, we see that we have one binary variable: $\mathbb{1}_{Senior}$. On our 9,036 observations, 8,406 observations have a 0 value for $\mathbb{1}_{Senior}$. In the particular case where all observations from a given batch have a 0 for $\mathbb{1}_{Senior}$, the Hessian will be singular. However, as shown in Figure 3, this percentage goes quickly to 100%. With a batch size of 100 observations, the algorithm will perform a Newton step 99.86% of the time. Thus, we see that having binary variables that are often equal to 0 is not a problem for the algorithm.

However, small batch sizes create other issues. Indeed, when computing the Hessian with small batch size, the only information we get is from a small subset of data. Therefore, for a given batch, the optimum can be different from the optimum on the whole dataset. Using the well optimized function `minimize` from the package `scipy.optimize`, we compute the optimum for different batch size. Then, we compare the euclidian distance between the optimum on the full dataset and the optimum from the different batches. Figure 4 shows the results of this experiment.

In Figure 4, we see that when taking small batches, the optimal solution is pretty far from the optimal solution on all data point. Therefore, this creates a problem in the computation of the step for SNM. Indeed, since we do not take into account previous Hessian, as opposed to RES-BFGS, the algorithm will often change direction with small batches. Indeed, every time we change the batch, the algorithm is chasing a different optimum, making difficult for it to achieve the real optimum. One way to fix this problem is to keep the information about previous Hessian and use this information to correct the direction of the algorithm.

V. Conclusion

In this paper, we presented a new stochastic second-order method called SNM. Just as SGD methods, the central idea is to compute the Hessian on a batch of observations. While not possible for machine learning algorithms that are generally used for the estimation of models including millions of parameters, we hypothezize that it is possible and desirable to include second-order information for estimating discrete



(a) First-order methods (b) Quasi-Newton methods (c) Second-order methods

Fig. 2. Comparison of the different algorithms presented in Section IV-A and III-B. The vertical axis corresponds to the normalized log-likelihood presented in Equation (5). Each time, a ten runs have been executed. The lines correspond to the average value over all the runs and the colored part correspond to the 95% confidence interval.

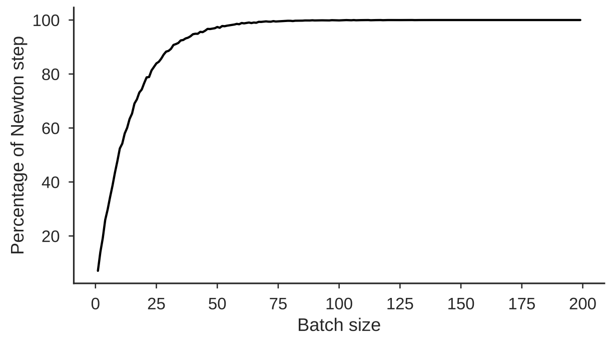


Fig. 3. Theoretical percentage of Newton step in function of the batch size for model $\bar{\mathcal{M}}$. The percentage was computed on a thousand draws.

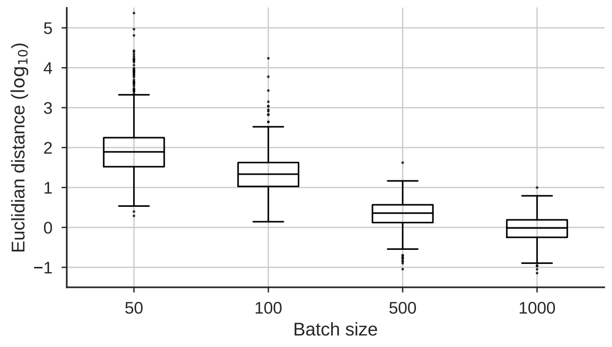


Fig. 4. Euclidian distance between the optimal parameters obtained on the full dataset and optimal parameters found on batches of the data for different batch sizes. The line in the middle represents the median. A thousand draws were coomputed for each batch size.

choice models that generally contain no more than dozens of parameters. We showed that a stochastic second-order approach was legit thanks to the finite-sum shape of the log-likelihood. We compared our algorithm with several first-order and quasi-newton benchmark algorithms using a very simple discrete choice model. Preliminary results have revealed that (stochastic) first-order methods encounter issues in estimating the parameters of such models, and that our algorithm was achieving a better performance.

Although preliminary results showed in this paper are encouraging, the current state of the algorithm only constitutes a first step toward our final goal that is the development of an optimization method specifically designed to estimate discrete choice model parameters on big data sets. The obvious next step is to improve the value of the learning rate to include second-order information from one or several previous steps. Then, the theoretical properties of our approach needs to be studied as the convergence rate of our algorithm is still unknown. Our algorithm will also have to be tested on more advanced discrete choice models (such as Nested Logit and Cross-Nested Logit models) and on much larger data sets. Regarding this latter, data sets including individuals' behavior over time have become increasingly available, and our approach seems to be particularly well suited for such data. Investigating the potential of our algorithm on panel data is therefore also an interesting avenue of future research.

VI. Acknowledgements

We would like to thank Tim Hillel for his valuable insight on the development of this method as well as his comments that significantly improved this article.