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 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 in transportation, marketing, health, or economics, among others. Estimating the parameters of 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 optimization 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 [1] or Larch [2]) can be 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 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 a Stochastic Newton Method (SNM) for the estimation of discrete choice models parameters. The objective of this paper is to investigate the convergence of our algorithm by benchmarking it against standard first-order methods and quasi-newton methods using a simple logit model on a small data set. We present preliminary results that indicate that our method outperforms (stochastic) first-order and quasi-newton methods. This 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 remainder of this paper is structured as follows. In section II, we present some related works about first and second-order optimization methods.

CHANGE THIS!

In Section **??**, 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 V shows the results and we present our concluding remarks and future works in section VI.

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 [3].

To remedy this situation, a momentum term is usually introduced in SGD algorithms (see [4]). Momentum-based

methods help accelerate gradients vectors in the right directions, leading to faster convergence. Other kind of techniques such as preconditioning can be used to solve this particular problem. Other first-order methods adapt the step size (or learning rate) to the parameters, such as Adagrad [5]. 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 [6] gives a good overview of first-order methods, from SGD up to complex and recent first-order algorithms such as Nadam [7] or AMSGrad [8].

More recently, thanks to the growth in computing power, researchers went beyond the first-order methods to consider quasi-newton methods that collect information about the curvature. The key motivation is to benefit from Newton's method speed without having to compute the exact second derivatives at each iteration. The central idea is therefore to build up, iteratively, an approximation of the Hessian matrix from the function and gradient values computed at previous step(s).

Much progress have been made lately toward developing Stochastic BFGS algorithms such as RES-BFGS [9], a regularized stochastic BFGS. Nowadays, several 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.* [10] have implemented an alternative version of BFGS for matrix inversion. Keskar *et al.* [11] have implemented adaQN, an adaptative quasi-newton method that is specifically designed for training Recurrent Neural Networks. Some researchers, such as Ye and Zhang [12], have got inspiration from the progress on first-order methods to improve second-order methods and Byrd *et al.* [13] 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. [14–17]), while little work has been done regarding the second-order methods.

III. Stochastic Newton Method (SNM)

The central idea behind our algorithm is to compute a stochastic Hessian instead of a full Hessian, *i.e.* use all observations to compute the Hessian. Indeed, conside a Choice model $\mathcal{P}(i|x_n, \beta, \mathcal{C}_n)$ that gives the probability that individual n chosses alternative i within choice set $\mathcal{C}_n$, given the values of the features x_n, and the parameters β. Now, consider a sample of individuals. For each individual n in the sample, the following data is available:

- The set $\mathcal{C}_n$ of available alternatives.
- The observed choice, characterized by the vector y_n, whose elements are defined as $y_{in} = 1$ is individual n has been observed to choose alternative i, and 0 otherwise. Note that $\sum_i y_{in} = 1 \forall n$.
- The vector of features x_n.

The log likelihood of the sample is a function of the unknown parameters β defined as

$$\mathcal{L}(\beta) = \sum_n \sum_i y_{in} \log \mathcal{P}(i|x_n, \beta, \mathcal{C}_n) \tag{1}$$

The estimation of the β parameters amounts to solve the following optimization problem

$$\max_\beta \mathcal{L}(\beta) \tag{2}$$

The structure of the objective function, in Equation (1), as a sum over entries in the database suggest the use of this stochastic approach.

We now present our algorithm named Stochastic Newton Method (SNM)[1]. . The input parameters are the following:

- β_0: The initial parameters used to start the optimization process.
- $\mathcal{D}$: Data. For the Python implementation, we use a `pandas.DataFrame`.
- f: Objective function. It corresponds to the log likelihood in Equation (1). It is a function that takes the parameters β and the data $\mathcal{D}$ and returns the function value.
- ∇f: Gradient of the objective function. It takes the parameters β and the data $\mathcal{D}$ and returns the gradient of f. This function has to work with batches, meaning that an additional parameters for the batch has to be provided.
- $\nabla^2 f$: Hessian of the objective function. It takes the parameters β and the data $\mathcal{D}$ and returns the Hessian of f. This function has to work with batches, meaning that an additional parameters for the batch has to be provided.
- N_{ep}: Maximum number of epochs. In our case, it is the only stopping criterion.
- N_{batch}: Batch size. It is used to compute the stochastic Hessiand and stochastic gradient on N_{batch} samples.

As outputs, SNM returns the epochs E, *i.e.* the steps, the parameters for all epochs β, and the objective function values for all epochs f_v.

The beginning of SNM is similar to all stochastic first-order algorithms. The number of samples $N_\mathcal{D}$ and the number of parameters M have to be retrieved from the data or given as parameters. Then, we can compute the number of iterations N_{iter} based on the maximum number of epochs N_{ep}, the number of samples $N_\mathcal{D}$, and the number of batches N_{batch} with the following formula:

$$N_{iter} = \lceil N_{ep} N_\mathcal{D} / N_{batch} \rceil$$

After the intialization of the output parameters, we can start the `for` loop on the iterations. We first fill the outputs with the current epoch and the current function value. The next step, on line 8, is to get the batch for the stochastic components of this algorithm. To achieve this, we draw N_{batch} indices from a uniform distribution $\mathcal{U}(0, N_\mathcal{D})$ without replacement. Then, on lines 9 and 10, we compute the stochastic gradient and the stochastic Hessian with the current parameters, denoted $\nabla f_{\texttt{idx}}(\beta[i])$ and $\nabla^2 f_{\texttt{idx}}(\beta[i])$ respectively. The next step is to decide wether we should do a gradient step or a Newton step. We do this by looking at the Hessian. Since we are trying to maximize the value of the log-likelihood, we can do a Newton

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

step if and only if the Hessian is negative definite. We can then obtain the Newton step by solving the following system

$$\nabla^2 f(\beta) \cdot p = -\nabla f(\beta)$$

Where p is the step direction we are looking for, ∇f is the gradient of a function f, and $\nabla^2 f$ is the Hessian of the same function f. If the Hessian is not negative definite, meaning than one of its eigenvalues is either 0 or positive, we cannot perform a Newton step. Thus, we can simply do a gradient step. In this case, the direction is given by the gradient itself. The next important step is to find a good step size, see line 15. We do this using a backtracking Line Search method using the Armijo-Goldstein condition [18]. The algorithm starts with $\alpha > 0$ being the maximum candidate step size. In addition, we use two search control parameters $\tau \in (0, 1)$ and $c \in (0, 1)$. In our case, $\alpha = 1$ and $\tau = c = 0.5$. Then, we can set $t = -c \cdot p^T \nabla f(\beta)$. The core loop goes as follows: until the condition $f(\beta) - f(\beta + \alpha p) \geq \alpha t$ is satistifed, set $\alpha = \tau \cdot \alpha$. Once the condition is reached, we can simply return α. In our implementation, we also added a stopping criterion to avoid being caught in infinite loop. Indeed, we stop the backtracking line search if $\alpha < 10^{-8}$.

IV. Case Study

We use the *Swissmetro* dataset [19] and build a logit model denoted by $\mathcal{M}$:

$$\begin{aligned} V_{\text{Car}} &= \text{ASC}_{\text{Car}} + \beta_{\text{TT,Car}}\text{TT}_{\text{Car}} + \beta_{\text{C,Car}}\text{C}_{\text{Car}} + \beta_{\text{Senior}}\mathbb{1}_{\text{Senior}} \\ V_{\text{SM}} &= \text{ASC}_{\text{SM}} + \beta_{\text{TT,SM}}\text{TT}_{\text{SM}} + \beta_{\text{C,SM}}\text{C}_{\text{SM}} \\ &\quad + \beta_{\text{HE}}\text{HE}_{\text{SM}} + \beta_{\text{Senior}}\mathbb{1}_{\text{Senior}} \\ V_{\text{Train}} &= \text{ASC}_{\text{Train}} + \beta_{\text{TT,Train}}\text{TT}_{\text{Train}} + \beta_{\text{C,Train}}\text{C}_{\text{Train}} + \beta_{\text{HE}}\text{HE}_{\text{Train}} \end{aligned} \tag{3}$$

where $\mathbb{1}_{\text{Senior}}$ is a feature 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 [1] 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 the parameters associated with the features normalized are multiplied by 100. The reason behind this normalization is to obtain parameters in the same order of magnitude.

	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.

	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.

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 [6] and Adagrad [5]. For the quasi-newton methods, we use BFGS algorithm [20] and RES-BFGS [9]. The main second-order algorithm is the Newton method [21]. Finally, to avoid the long and tedious search of a good step size, we run all algorithms presented above with the backtracking Line Search method using the Armijo-Goldstein condition [18] as explained at the end of Section III.

V. 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.

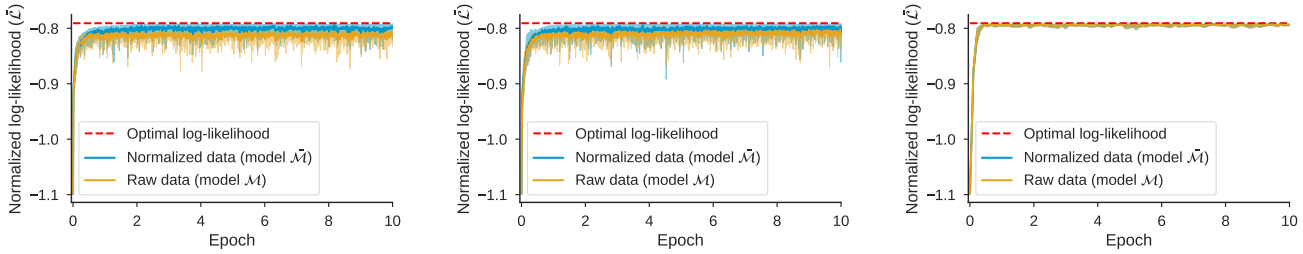
A. Raw data vs Normalized data

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 **??**, 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.

Algorithm 1 Stochastic Newton Method (SNM)

Input: Starting parameter value (β_0), data ($\mathcal{D}$), function (f), gradient (∇f), Hessian ($\nabla^2 f$), number of epochs (N_{ep}), batch size (N_{batch})
Output: Epochs (E), parameters (β), function values (f_v)

function SNM
  $(N_{\mathcal{D}}, M) = |\mathcal{D}|$ ▷ Number of samples and parameters
  $N_{iter} \leftarrow \lceil N_{ep} N_{\mathcal{D}} / N_{batch} \rceil$ ▷ Number of iterations
  Initialize E, β and f_v. Set $\beta[0] \leftarrow \beta_0$
  for $i = 0 \dots N_{iter}$ **do**
    $E[i] \leftarrow i \cdot N_{batch} / N_{\mathcal{D}}$ ▷ Store the epoch
    $f_v[i] \leftarrow f(\beta[i])$ ▷ Store the function value
    `idx` $\leftarrow N_{batch}$ indices from $\mathcal{U}(0, N_{\mathcal{D}})$ without replacement
    `grad` $\leftarrow \nabla f_{\texttt{idx}}(\beta[i])$ ▷ Gradient on the samples from `idx`
    `hess` $\leftarrow \nabla^2 f_{\texttt{idx}}(\beta[i])$ ▷ Hessian on the samples from `idx`
    if `hess` is negative definite **then**
      Solve `hess` $\cdot$ `step` $= -$`grad` to get `step` ▷ Newton step
    else
      `step` $\leftarrow$ `grad` ▷ Gradient step
    $\alpha \leftarrow$ Backtracking Line Search with `step` on the subset of data with indices from `idx`
    $\beta[i+1] \leftarrow \beta[i] + \alpha \cdot$ `step`
  $E[n_{iter}] \leftarrow N_{iter} \cdot N_{batch} / N_{\mathcal{D}}$
  $f_v[N_{iter}] \leftarrow f(\theta[N_{iter}])$
  return E, β and f_v



(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 (**??**). 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.

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 V-B and V-C, 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.

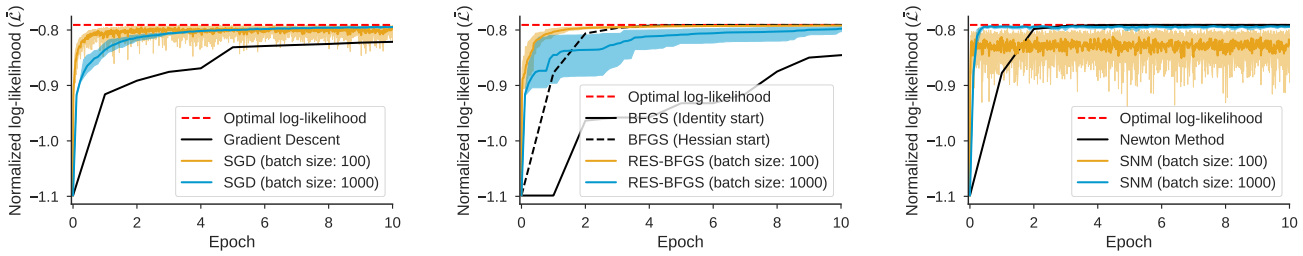
B. Comparison of the algorithms

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 colored parts show the 95% confidence interval.

	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.

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-



(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. The vertical axis corresponds to the normalized log-likelihood presented in Equation (**??**). 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.

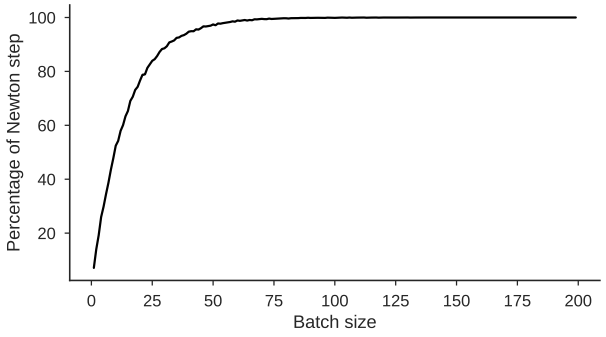


Fig. 3. Theoretical percentage of Newton step in function of the batch size for model $\tilde{\mathcal{M}}$. The percentage was computed on a thousand draws.

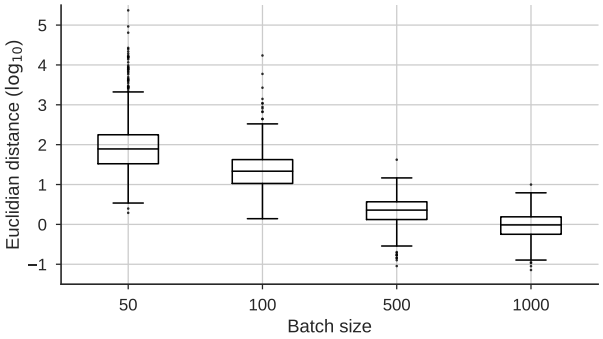


Fig. 4. Euclidian distance between the optimal parameters obtained on the full dataset and optimal parameters found on batches of the data for different batch sizes. The line in the middle represents the median. A thousand draws were coomputed for each batch size.

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 V-C, we study the possible reason behind such behavior.

C. *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, 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 3, we see that we have one binary feature: $\mathbb{1}_{Senior}$. 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 features 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.

VI. Conclusion

In this paper, we have presented a 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 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 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 use a more sophisticated way to calculate a pre-conditioning matrix using batch second order information. Also, the calculation of the step should be improved. 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.

VII. 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.

References

[1] M. Bierlaire, "BIOGEME: a free package for the estimation of discrete choice models," *Swiss Transport Research Conference 2003*, Mar. 2003. [Online]. Available: http://www.strc.ch/2003/bierlaire.pdf

[2] J. Newman, V. Lurkin, and L. Garrow, "Computational methods for estimating multinomial, nested, and cross-nested logit models that account for semi-aggregate data." *Journal of Choice Modelling*, no. 26, pp. 28–40, 2018. [Online]. Available: https://orbi.uliege.be/handle/2268/201287

[3] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," *Proceedings of Eightth Annual Conference of the Cognitive Science Society, 1986*, 1986. [Online]. Available: https://ci.nii.ac.jp/naid/10022346408

[4] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166

[5] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[6] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv:1609.04747 [cs]*, Sep. 2016, arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747

[7] T. Dozat, "Incorporating Nesterov Momentum into Adam," Feb. 2016. [Online]. Available: https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ

[8] S. J. Reddi, S. Kale, and S. Kumar, "On the Convergence of Adam and Beyond," Feb. 2018. [Online]. Available: https://openreview.net/forum?id=ryQu7f-RZ

[9] A. Mokhtari and A. Ribeiro, "RES: Regularized Stochastic BFGS Algorithm," *IEEE Transactions on Signal Processing*, vol. 62, no. 23, pp. 6089–6104, Dec. 2014.

[10] R. M. Gower, F. Hanzely, P. Richtrik, and S. Stich, "Accelerated Stochastic Matrix Inversion: General Theory and Speeding up BFGS Rules for Faster Second-Order Optimization," *arXiv:1802.04079 [cs, math]*, Feb. 2018, arXiv: 1802.04079. [Online]. Available: http://arxiv.org/abs/1802.04079

[11] N. S. Keskar and A. S. Berahas, "adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science. Springer, Cham, Sep. 2016, pp. 1–16. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46128-1_1

[12] H. Ye and Z. Zhang, "Nestrov's Acceleration For Second Order Method," *arXiv:1705.07171 [cs]*, May 2017, arXiv: 1705.07171. [Online]. Available: http://arxiv.org/abs/1705.07171

[13] R. Byrd, G. Chin, W. Neveitt, and J. Nocedal, "On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 977–995, Jul. 2011. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/10079923X

[14] R. Kiros, "Training Neural Networks with Stochastic Hessian-Free Optimization," *arXiv:1301.3641 [cs, stat]*, Jan. 2013, arXiv: 1301.3641. [Online]. Available: http://arxiv.org/abs/1301.3641

[15] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755842

[16] A. Bordes, L. Bottou, P. Gallinari, J. Chang, and S. A. Smith, "Erratum: SGDQN is Less Careful than Expected," *Journal of Machine Learning Research*, vol. 11, no. Aug, pp. 2229–2240, 2010. [Online]. Available: http://www.jmlr.org/papers/v11/bordes10a.html

[17] N. Agarwal, B. Bullins, and E. Hazan, "Second-Order Stochastic Optimization for Machine Learning in Linear Time," *arXiv:1602.03943 [cs, stat]*, Feb. 2016, arXiv: 1602.03943. [Online]. Available: http://arxiv.org/abs/1602.03943

[18] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, Jan. 1966. [Online]. Available: https://msp.org/pjm/1966/16-1/p01.xhtml

[19] M. Bierlaire, K. Axhausen, and G. Abay, "The acceptance of modal innovation: The case of Swissmetro," *Swiss Transport Research Conference 2001*, Mar. 2001. [Online]. Available: http://www.strc.ch/2001/bierlaire1.pdf

[20] R. Fletcher, *Practical Methods of Optimization; (2Nd Ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[21] J. Caswell, "A treatise of algebra, both historical and practical," Tech. Rep., 1685.