Hessian-free second-order methods for statistical learning

Loïc Shi-Garrier

September 2019

Abstract

In this report, we introduce a method to efficiently compute Hessian-vector product for second-order trust region method with truncated conjugate gradient in the context of maximum likelihood estimation or non-linear least squares problems. We present numerical experiments, notably an application to the classical machine learning problem of MNIST handwritten digit recognition. The code used in this report can be found in `https://github.com/lshigarrier`.

Acknowledgments: I would particularly like to thank Professor F. Bastin to have received me as an intern in the Département d'informatique et de recherche opérationnelle (DIRO, Université de Montréal). I would also like to thank Professor M. Sbihi and Professor M. Mongeau to have suggested me this experience.

1 Introduction

1.1 Objectives and contents

Considering machine learning problems dealing with large datasets, first-order methods like the stochastic gradient descent have been privileged because of computational constraints. However, those techniques suffer several issues, in particular the large number of hyperparameters requiring fine tuning to achieve good performances. For example, the choice of the learning rate is challenging to achieve efficient convergence behaviour near the optimum (see chapter 8 in [5] or chapter 3 in [9] for more details).

In this report, we propose a computationally efficient second-order method for maximum likelihood estimation and non-linear least squares problems and we applied it to several examples including a classical example of machine learning problem: MNIST handwritten digit recognition.

In the rest of this section, we introduce our notations and recall the statement of the maximum likelihood estimation and of non-linear least squares problem. In section 2, we present the full algorithm used in this report: trust region method with truncated conjugate gradient for the computation of the trial step. Then, we introduce our proposed method as a refinement of the BHHH and Gauss-Newton methods to approximate the Hessian in the trust region model. The end of section 2 is dedicated to a brief introduction to retrospective approximation. Section 3 deals with the application and comparison of the different methods introduced in section 2 to three types of problems: a very basic non-linear least squares regression, a maximum likelihood estimation with logit model and MNIST handwritten digit recognition. Section 4 is dedicated to conclusive remarks and indications for future work.

1.2 Statement of the problems and notations

In this subsection, we will introduce our notations and briefly recall both the maximum likelihood estimation (see [8] for more details) and the least squares problem (see chapter 10 in [4] for more details).

1.2.1 Maximum likelihood estimation

Suppose we have a family of probability distributions (which can either be discrete distributions, or probability density functions in the continuous case) $P(\,\cdot\,, \beta)$ defined over $\mathbb{R}^p$ and parameterized by $\beta \in \mathbb{R}^n$. Given a set of observations $\{x_i \in \mathbb{R}^p,\ i \in [\![1, N]\!]\}$ i.i.d sampled from the distribution $P(\,\cdot\,, \beta^*)$ parameterized by an unknown parameter β^*, we consider the negative log-likelihood function:

$$LL(\beta) = \frac{1}{N}\sum_{i=1}^{N} -\ln P(x_i, \beta) \tag{1}$$

which can be seen as an estimator of $\mathbb{E}[-\ln P(x_i, \beta)]$, computed using *sample average approximation* [6] over the set of observations.

The maximum likelihood estimator is then defined as:

$$\hat{\beta} = \operatorname*{arg\,min}_{\beta \in \mathbb{R}^n} LL(\beta)$$

Note that $\hat{\beta}$ does not necessarily exist, neither is necessarily unique if it exists. It can be shown that if β^* is identified, then β^* is the unique minimizer of $\mathbb{E}[-\ln P(x_i, \beta)]$ (see Lemma 2.2 in [8]). With some supplemental assumptions, we have that $\hat{\beta}$ exists and is consistent (Theorem 2.5 and Theorem 2.7 in [8]).

The score of the i-th observation is defined as :

$$s_i(\beta) = -\nabla \ln P(x_i, \beta) \tag{2}$$

Hence, the gradient of the log-likelihood can be written as:

$$\nabla LL(\beta) = \frac{1}{N}\sum_{i=1}^{N} s_i(\beta) \tag{3}$$

The Hessian matrix of the i-th observation at the point β will be denoted :

$$H_i(\beta) = \nabla s_i(\beta) \tag{4}$$

1.2.2 Non-linear least squares

Suppose we have a set of observations i.i.d. $\{(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R},\ i \in [\![1, N]\!]\}$ and a non-linear model function $g(\,\cdot\,, \beta)$ defined over $\mathbb{R}^p$ and parameterized by $\beta \in \mathbb{R}^n$. For all i, let the residual be $r_i(\beta) = g(x_i, \beta) - y_i$. We note $R(\beta) = \begin{bmatrix} r_1(\beta) & \cdots & r_N(\beta) \end{bmatrix}^T$. The residual sum of squares is:

$$RSS(\beta) = \frac{1}{2N} R(\beta)^T R(\beta) \tag{5}$$

which can be seen as an estimator of $\mathbb{E}[(g(x_i, \beta) - y_i)^2]$.

The non-linear least square estimator (if it exists and is unique, see Theorem 2.1 and section 2.2.2 in [8]) is then defined as:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^n} RSS(\beta)$$

The gradient of $RSS(\beta)$ can be expressed as:

$$\nabla RSS(\beta) = \frac{1}{N} \sum_{i=1}^{N} r_i(\beta) \nabla r_i(\beta)$$

The Hessian of $RSS(\beta)$ is:

$$\nabla^2 RSS(\beta) = \frac{1}{N} \sum_{i=1}^{N} \nabla r_i(\beta) \nabla r_i(\beta)^T + \frac{S(\beta)}{N} \tag{6}$$

where

$$S(\beta) = \sum_{i=1}^{N} r_i(\beta) \nabla^2 r_i(\beta) \tag{7}$$

For more details, we refer the reader to chapter 10 of [4].

2 Algorithms

In this section, we first present the basic framework of our methods. This includes the basic trust region algorithm along with the truncated conjugate gradient method as the trial step calculation method. Then, we introduce the new method HOPS (Hessian-free Outer Product of the Scores) as a refinement of the BHHH method for the maximum likelihood estimation and of the Gauss-Newton method for the non-linear least squares problem.

In the last subsection, we briefly introduce the retrospective approximation framework that will be hybridized with the HOPS method.

2.1 Basic Trust Region

In this subsection, we present the basic trust region algorithm. Since trust region methods are not the objective of this report, we will just give some insights and refer the reader to [3] and to chapter 4 in [10] for in-depth discussion of the method.

We first consider the following unconstrained optimization problem:

$$\min_{\beta \in \mathbb{R}^n} f(\beta) \tag{8}$$

where f is supposed twice continuously differentiable.

The idea of the trust region method is to approximate the objective function f by a model m_k around the current iteration point β_k for each iteration k, and thereby defined a new optimization problem:

$$\begin{aligned} \min_{s \in \mathbb{R}^n} & \; m_k(\beta_k + s) \\ s.t & \; \|s\| \leq \Delta_k \end{aligned} \tag{9}$$

where the step length $\|s\|$ is constrained by a trust region of radius Δ_k. If the diminution in the objective function is sufficient, the next iterate is defined as $\beta_{k+1} = \beta_k + s_k$, where s_k is an (approximate) solution of (9).

In the following, we will consider the quadratic model defined in (10) using the gradient of f evaluated at β_k and an approximation of the Hessian. The full basic trust region algorithm as implemented for the applications in section 3 is detailed in Algorithm 1.

Algorithm 1 (Basic Trust Region)**.**

Step 0: Initialization
An initial point β_0 and an initial trust region radius Δ_0 are given.
The constants η_1, η_2, γ_1, γ_2 and Δ_{max} are given with $0 < \eta_1 \leq \eta_2 < 1$ and $0 < \gamma_1 \leq \gamma_2 < 1$.
Compute $f(\beta_0)$
Let $k = 0$

While stopping criterion not met:

Step 1: Model definition

Let the quadratic model be defined by:

$$m_k(\beta_k + s) = m_k(\beta_k) + g_k^T + \frac{1}{2} s^T H_k s \tag{10}$$

over the following set (trust region):

$$\mathcal{B}_k = \{s,\ \|s\|_2 \leq \Delta_k\}$$

with:

$$\begin{aligned} m_k(\beta_k) &= f(\beta_k) \\ g_k &= \nabla f(\beta_k) \end{aligned}$$

H_k is an approximation of the Hessian. Several methods can be used to obtain it. This is the topic of section 2.2.

Step 2: Trial step calculation

Calculation of the trial step s_k as an approximate solution of (9). This approximate solution will be obtained with the truncated conjugate gradient method presented in Algorithm 2.

Step 3: Acceptance of the trial point

Compute $f(\beta_k + s_k)$
Compute:

$$\rho_k = \frac{f(\beta_k) - f(\beta_k + s_k)}{m_k(\beta_k) - m_k(\beta_k + s_k)} \tag{11}$$

If $\rho_k \geq \eta_1$ then $\beta_{k+1} = \beta_k + s_k$ and the iteration is said to be successful.
Else $\beta_{k+1} = \beta_k$

Step 4: Trust region radius update

If $\rho_k \geq \eta_2$ then $\Delta_{k+1} = \min(\Delta_{max}, \max(\Delta_k, 4\|s_k\|_2))$ and the iteration is said to be very successful.
If $\rho_k \in [\eta_1, \eta_2)$ then $\Delta_{k+1} = \gamma_2 \Delta_k$.
If $\rho_k < \eta_1$ then $\Delta_{k+1} = \gamma_1 \Delta_k$.
Increment k by 1 and go to step 1.

The ratio ρ_k defined in (11) aims to measure how good is the decrease in the objective function in regards of the actual decrease in the model. If the model fits the function well, ($\rho_k \geq \eta_2$), the trust region can be expanded. On the other hand, if the model differs greatly from the function, that means that the trust region was too large and hence should be reduced. If ρ_k is very small ($\rho_k < \eta_1$), the step is rejected and the radius of the trust region is decreased.

In the section 3, we use the following classical values for the constants: $\eta_1 = 0.01$, $\eta_2 = 0.9$, $\gamma_1 = \gamma_2 = 0.5$ and $\Delta_{max} = 10^{12}$.

2.1.1 Truncated Conjugate Gradient

To compute the trial step s_k in the step 2 of Algorithm 1, we use the truncated conjugate gradient method. The conjugate gradient is a method used to efficiently minimize quadratic problems since it converges in n iterations where n is the size of the problem. The method can also be applied to more general problems (see chapter 5 in [10] for more information). The truncated conjugate gradient is a refinement of the conjugate gradient to cope with the constrains of the trust region.

Moreover, we do not need the exact minimizer of the model since the model itself is just an approximation of the objective function. The conjugate gradient quickly converges to very good approximations of the minimizer and hence could be stopped early.

More details about the truncated conjugate gradient can be found in [14] and [15], in particular, note that for simplicity we do not have used any preconditioning method here.

The complete algorithm of the truncated conjugate gradient is detailed in Algorithm 2, where we denote the iterations by i, while k is the current iteration of the trust region algorithm.

Algorithm 2.

Initialization

g_k, H_k and Δ_k are given in step 1 of Algorithm 1.
χ and θ are given constants with $0 < \chi < 1$ and $\theta \geq 0$.

```
Let:

$$\begin{aligned}\tilde{g}_0 &= g_k\\ d_0 &= -\tilde{g}_0\\ \tilde{s}_0 &= 0\\ i &= 0\end{aligned}$$

While $\|\tilde{\mathbf{g}}_\mathbf{i}\|_\mathbf{2} > \|\tilde{\mathbf{g}}_\mathbf{0}\|_\mathbf{2} \min\{\chi, \|\tilde{\mathbf{g}}_\mathbf{0}\|_\mathbf{2}^\theta\}$ or $\mathbf{i} \leq \mathbf{n}$:

Let $\kappa_i = d_i^T H_k d_i$.
If $\kappa_i \leq 0$ then:
        Compute $\sigma_i$ as the positive root of $\|\tilde{s}_i + \sigma d_i\|_2 = \Delta_k$
        Let $s_k = \tilde{s}_i + \sigma_i d_i$.
        Stop.
Else compute $\alpha_i = \frac{\|\tilde{g}_i\|_2^2}{\kappa_i}$.
If $\|\tilde{s} + \alpha_i d_i\|_2 \geq \Delta_k$ then:
        Compute $\sigma_i$ as the positive root of $\|\tilde{s}_i + \sigma d_i\|_2 = \Delta_k$
        Let $s_k = \tilde{s}_i + \sigma_i d_i$.
        Stop.
Else:

$$\begin{aligned}\tilde{s}_{i+1} &= \|\tilde{s} + \alpha_i d_i\|_2\\ \tilde{g}_{i+1} &= \tilde{g}_i + \alpha_i H_k d_i\\ d_{i+1} &= -\tilde{g}_{i+1} + \frac{\|\tilde{g}_{i+1}\|_2^2}{\|\tilde{g}_i\|_2^2} d_i\end{aligned}$$

Increment $i$ by 1.
```

Note that the positive root of $\|\tilde{s}_i + \sigma d_i\|_2 = \Delta_k$ can be obtained with :

$$\sigma_i = \frac{-\tilde{s}_i^T d_k + \sqrt{(\tilde{s}_i^T d_i)^2 - \|d_i\|_2^2(\|\tilde{s}_i\|_2^2 - \Delta_i^2)}}{\|d_i\|_2^2}$$

where the various terms are computed as follow :

$$\|d_i\|_2^2 = \|\tilde{g}_i\|_2^2 + \frac{\|\tilde{g}_i\|_2^2}{\|\tilde{g}_{i-1}\|_2^2}^2 \|d_{i-1}\|_2^2$$

$$\tilde{s}_i^T d_i = \frac{\|\tilde{g}_i\|_2^2}{\|\tilde{g}_{i-1}\|_2^2} (\tilde{s}_{i-1}^T d_{i-1} + \alpha_{i-1}\|d_{i-1}\|_2^2)$$

$$\|\tilde{s}_{i+1}\|_2^2 = \|\tilde{s}_i\|_2^2 + 2\alpha_i \tilde{s}_i^T d_i + \alpha_i^2 \|d_i\|_2^2$$

In the section 3, we use the following values for the constants: $\chi = 0.1$ and $\theta = 0.5$.

2.1.2 Convergence of BTR

In this paragraph, we present a result on the convergence of Algorithm 1, using Algorithm 2 to compute the trial step. Let the Cauchy point be defined as the minimizer of the model along the direction of steepest descent subjected to the trust region constraint:

$$s_k^C = \arg\min_{t\in[0,1]} m_k \left(\beta_k - t\frac{\Delta_k}{\|g_k\|_2} g_k \right)$$

Under mild assumptions, it can be shown that when the iteration is successful, the decrease in the model obtained with the truncated conjugate gradient is at least as good as the decrease obtained with the Cauchy point. Indeed, the first iterate of the truncated conjugate gradient is the Cauchy point and the following iterates lead to a greater decrease in the model (see Corollary 6.3.2 in chapter 6 of [3] and Theorem 2.1 in [14]). This is a condition for the following convergence result:

Theorem 1.

$$\lim_{k\to\infty} \|\nabla f(\beta_k)\|_2 = 0$$

Proof. See Theorem 6.4.6 in chapter 6 of [3]. □

2.2 Approximation of the Hessian

In this subsection, we first present the BHHH method to compute an approximation of the Hessian corresponding to the H_k term in the model constructed in the step 1 of Algorithm 1. Then the HOPS method is introduced as a refinement of the BHHH method, and we show how it can be used as a refinement of the Gauss-Newton method in the non-linear least squares framework.

2.2.1 BHHH

We will present here the BHHH method (see [1]) for approximating the Hessian in the maximum likelihood estimation. This method is originally used as a quasi-Newton method, see chapter 8 of [16] for more details. The justification of this method is based on the information matrix identity, stated below:

Theorem 2 (Information matrix identity)**.** Let $s_i(\beta)$ be the score defined in (2) and $H_i(\beta)$ the Hessian matrix defined in (4). Suppose the model is well defined and let β^* be the true value of parameters. Then we have:

$$\mathbb{E}[H_i(\beta^*)] = \mathbb{E}[s_i(\beta^*)s_i(\beta^*)^T]$$

Proof. See paragraph 8.7 in chapter 8 of [16]. $\square$

This means that, at the true parameter, the average Hessian in the population is equal to the average product of the scores in the population (that is the covariance matrix of the scores in the population). Hence, one approximation of the Hessian at any point β_k could be the "outer product of the gradient":

$$H_k^{BHHH} = \frac{1}{N}\sum_{i=1}^{N} s_i(\beta_k)s_i(\beta_k)^T$$

which is the BHHH approximation of the Hessian. Intuitively, we could expect that this approximation became more and more accurate as we approach the true value of the parameters and the sample size grows to infinity. Note that the calculation of the scores is required to compute the gradient as in (3).

2.2.2 HOPS

Some issues make the BHHH approximation (and more generally the quasi-Newton methods) impracticable when dealing with large datasets. One such issue is that it requires to store the entire Hessian approximation which have a memory impact of $\mathcal{O}(n^2)$ where n is the number of parameters.

Another issue is that the computation of this approximation requires the computation of N outer product of size $n \times n$ where N is the number of observations. Then, the BHHH matrix is only used in matrix-vector product to evaluate the model m_k and also during the trial step computation with the truncated conjugate gradient.

The idea of the HOPS method is to only store the scores of the observations $s_i(\beta_k)$. Hence, the impact memory is lower if the number of parameters is greater than the number of observations, as it is the case in some modern machine learning models like deep learning [5]. The computation impact is reduced as there is no more need to compute any outer product. The matrix-vector products are computed as the sum of the scores weighted by dot products, such that given a $v \in \mathbb{R}^n$:

$$H_k^{BHHH}\, v = \frac{1}{N}\sum_{i=1}^{N}(s_i(\beta_k)^T v)\, s_i(\beta_k)$$

The HOPS method can also be applied in non-linear least squares as it shows similar structures to approximate the Hessian as the maximum likelihood estimation. Consider the Hessian expression given in (6). The $S(\beta)$ term, given in (7), is costly to compute since we need the second-order derivative of each residual $\nabla^2 r_i(\beta)$. One might think to discard this term and to compute an approximation of the Hessian as:

$$H_k^{LS} = \frac{1}{N}\sum_{i=1}^{N}\nabla r_i(\beta_k)\nabla r_i(\beta_k)^T \tag{12}$$

This approximation corresponds to the approximate Hessian used in the Gauss-Newton algorithm. More details can be found in chapter 10 of [4]. The $\nabla r_i(\beta_k)$ terms can be treated as the scores in the maximum likelihood estimation and lead to similar practical improvements as stated above.

2.3 Retrospective approximation

At the beginning of the trust region algorithm, the iterates might be far from an optimum. Instead of directly using the full dataset, the idea of retrospective approximation (RA) is to solve a sequence of sample-path problems where the objective function is estimated with a sample average approximation over a subsample of the full dataset. More precisely, a sample-path problem of sample size $N_l \leq N$ is defined as:

$$\min_{\beta\in\mathbb{R}^n} f_{N_l}(\beta) = \frac{1}{N_l}\sum_{i=1}^{N_l} Y(x_i, \beta) \tag{13}$$

where Y is a function of the i.i.d observations $\{x_i\}$ and of the parameters β, such that $f(\beta) = \mathbb{E}[Y(x_i, \beta)]$ is the objective function of (8).

The first sample-path problems have small subsamples and are solved up to a low accuracy, so that they converge quickly and lead to quick improvements as the iterates are far from an optimum. Each sample-path problem uses the final iterate of the previous problem as an initial iterate ("warm start"). The subsamples are progressively increased by generating independent observations, as the accuracy of the solving of each problem is refined. The procedure is stopped when the subsamples reach a certain limit (for example all the available observations) or when a limit computation time is reached. A review of the RA techniques can be found in [6]. The RA framework is described in Algorithm 3, where l denotes the iterations of the method.

```
**Algorithm 3** (Retrospective approximation).

**Initialization**
The initial sample size $N_0$, the initial error-tolerance $\epsilon_0$ and the initial point $\beta_0$ are given.
Let $l = 0$.

**While stop criterion not met:**
    Generate a sample-path problem of size $N_l$ as defined in (13).
    Initialize Algorithm 1 with $\beta_l$ and solve the sample-path problem up to a precision $\epsilon_l$ to obtain a point $\beta_{l+1}$.
    Generate $N_{l+1}$ and $\epsilon_{l+1}$ such that the sequence of sample sizes $\{N_j\}$ is tending to infinity and the sequence of error-tolerances $\{\epsilon_j\}$ is tending to zero.
    Increment $l$ by 1.
```

Note that an accuracy of error-tolerance ϵ_l is defined as $|f_{N_l}(\beta_k) - f^*_{N_l}| \leq \epsilon_l$ where β_k is the current iterate and $f^*_{N_l}$ is the optimal value of the function f_{N_l} defined in (13). In the section 3, we will use a multiplicative sample size policy, such that $N_{l+1} = KN_l$ for a fixed K. The choice of this policy is mainly empirical, while some justifications could be taken from [11].

3 Applications

In this section, we apply the HOPS method to some classical examples and compare its performances with other methods and with several stopping criteria. The algorithms presented in the previous section are implemented in Julia. The trust region implementation with the truncated conjugate gradient is taken from the GERALDINE repository in [2] and the logit model implementation from the AMLET repository also in [2]. The test are conducted on a server with 16 Intel® Xeon® Silver 4110 CPUs (2.10 GHz) and 64 Go of memory.

3.1 Basic non-linear least squares problems

The first example is a very basic example of non-linear least squares with observations x_i and parameters β^* generated randomly with [2] (repository RDST). The y_i are then computed as:

$$y_i = \beta_1^* + \sum_{j=1}^{n-1} \exp\left(\beta_{j+1}^*\, x_{i,j}\right) + u_i$$

with $\beta^* = \begin{bmatrix}\beta_1^* & \cdots & \beta_n^*\end{bmatrix}^T$ and $x_i = \begin{bmatrix}x_{i,1} & \cdots & x_{i,n-1}\end{bmatrix}^T$, and where the u_i are i.i.d. according to a centered normal law of standard deviation 0.01. The model function is:

$$g(x_i, \beta) = \beta_1 + \sum_{j=1}^{n-1} \exp\left(\beta_{j+1}\, x_{i,j}\right)$$

For various numbers of parameters, we compare the performances of HOPS with three other methods. The first method uses the same approximation as HOPS but with the explicit computation and storage of H_k^{LS} as stated in (12), and will be referred as GN (for Gauss-Newton). The second method is the BFGS approximation (see chapter 6 of [10] for the details). The third method relies on storing the true Hessian (TH), computed using forward mode automatic differentiation (see [12]). The number of observations generated is 20 times the number of parameters. The stopping criteria used in the trust region algorithm is a robust test of the first order necessary condition, such that for all j:

$$\left| \frac{\partial RSS(\beta)}{\partial \beta_j} \cdot \frac{\max(|\beta_j|, 1)}{\max(|RSS(\beta)|, 1)} \right| \leq \epsilon_1 \tag{14}$$

where we have used a value of $\epsilon_1 = 10^{-4}$.

The results are summarized in the table 1 where n is the number of parameters and where the values are averaged over ten runs. We provide the computation time, the number of iterations of the trust region algorithm (Algorithm 1) and the memory allocations.

	TH	BFGS	GN	HOPS
$n = 25$				
Computation time (s)	0.48	0.17	0.083	0.045
Nb of iterations	6	26	6	6
Allocations (MiB)	794	196	143	57
$n = 50$				
Computation time (s)	5.71	1.10	0.58	0.26
Nb of iterations	8	32	7	7
Allocations (MiB)	12108	1269	1042	334
$n = 100$				
Computation time (s)	110	31.7	7.21	2.49
Nb of iterations	11	148	18	18
Allocations (MiB)	253097	38522	14513	4045

Table 1: Basic non-linear least squares

Note that the four methods always converge toward the true values of parameters. For these problems, HOPS seems to perform better than the three other methods. In particular, the comparison between HOPS and GN tends to show that, when the size of the problem increases, HOPS improves its performances over GN. The ratio in computation times of GN over HOPS is 1.84 for $n = 25$ whereas it is 2.90 for $n = 100$. Since the number of parameters is less than the number of observations and the problems are small, we expected that GN performs better but it is not the case. An explanation could be than the computation of the outer products of the scores ($n \times n$ matrices) is costly both in terms of memory allocations and in terms of running time and hence compensate the memory impact of the storage of the scores in HOPS ($20n$ vectors of size n).

3.2 Maximum likelihood estimation with logit model

This second example is a maximum likelihood estimation performed on real data. The objective function has the same form than (1) with $P(x_i, \beta)$ a logit model as stated here (more information in chapter 3 of [16]):

$$P(x_i, \beta) = \frac{e^{\beta^T x_{i,0}}}{\sum_{j=1}^{k_i} e^{\beta^T x_{i,j}}} \tag{15}$$

where $x_i = \begin{bmatrix} x_{i,0} & \cdots & x_{i,k_i} \end{bmatrix}$ is a matrix where each column is a vector of observed variables relating to one of the k_i alternatives faced by the i-th individual. The first column $x_{i,0}$ is the chosen alternative. This example includes $N = 3000$ individuals and $n = 19$ parameters.

We compare the performances of HOPS with BFGS and BHHH. The values are averaged over several runs and presented in the table 2.

	BFGS	BHHH	HOPS
Computation time (s)	18.0	18.1	11.6
Nb of iterations	57	33	33
Allocations (GiB)	29.12	31.22	16.92

Table 2: Maximum likelihood estimation with logit model

In this example of maximum likelihood, HOPS seems to perform better than both BFGS and BHHH (whose performances are similar) in terms of running time as far as in terms of memory allocation. Similarly as GN in the previous subsection, the low performances of BHHH could be explained by the cost of the computation of the outer products of the scores (19×19 matrices).

3.3 MNIST handwritten digit recognition

The MNIST database is a famous problem of handwritten digit recognition (the datasets can be found in [7]). It is composed of a training set of 60000 images and a test set of 10000 images. Each image is a 28×28 pixels grayscale handwritten digit. The training set will be used to train the model described below and the test set to get a measure of the accuracy obtained.

3.3.1 Description of the model

We will briefly describe the model used during the testings. More details can be found in [9] or in chapter 6 of [5]. Since the accuracy of the model is not the objective of this report and since we try to avoid long computation times, we use a very simple multilayer perceptron with no hidden layer (except for the last testings where we use some hidden layers). The input layer is made of 784 neurons (28×28) and the ouput layer is made of 10 neurons (one for each digit). The number of parameters, including the biases, is hence $n = 7850$. The activation function used for the output layer is a generalisation of the logit function defined in (15) called the softmax function:

$$Softmax(\hat{y}) = \begin{bmatrix} \frac{e^{\hat{y}_1}}{\sum_{j=1}^{10} e^{\hat{y}_j}} & \cdots & \frac{e^{\hat{y}_{10}}}{\sum_{j=1}^{10} e^{\hat{y}_j}} \end{bmatrix}$$

where $\hat{y} \in \mathbb{R}^{10}$ is the input of the output layer.

In the final testings of this subsection, we use the ReLU function as the activation function for the hidden layers:

$$ReLU(z) = \max(0, z)$$

where z is the input of a hidden neuron.

The objective function (also referred as the loss function) is the cross-entropy as stated below:

$$L(\beta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{10} -y_{i,j} \ln(\Phi(x_i, \beta)_j) \tag{16}$$

where $x_i \in \mathbb{R}^{784}$, $\beta \in \mathbb{R}^{7850}$ and y_i is the one-hot encoding of the expected output such that $y_{i,j} = 1$ if the expected output is the digit j and $y_{i,j} = 0$ otherwise. The function Φ represents the forward propagation in the network while $\Phi(x_i, \beta)_j$ is the j-th component of the output vector.

Since the output layer's activation function is the softmax function, the value $\sum_{j=1}^{10} -y_{i,j} \ln(\Phi(x_i, \beta)_j)$ can be seen as $-\ln[P(x_i, \beta)]$ where $P(x_i, \beta)$ is the probability to obtain the expected digit. Hence, we can see $L(\beta)$ as a negative log-likelihood function defined in (1).

In the following testings, we use three different stopping criteria. The first one is the same as described in (14) and will be referred as criterion 1. The second criterion tests the decrease in the objective function between two successful iterations of the trust region algorithm, and will be referred as criterion 2. This criterion is explained below.

After each successful iteration of Algorithm 1, we compute the following statistic:

$$T_k = \frac{(f(\beta_{k-1}) - f(\beta_k)) - \epsilon_2 f(\beta_{k-1})}{\sqrt{\sigma^2/N}} \tag{17}$$

where we see $f(\beta_{k-1}) - f(\beta_k) = \frac{1}{N} \sum_{i=1}^{N} Y_k$ as an estimator of the expectation of a random variable $Y_k = X_{k-1} - X_k$ whose realisations are the differences of cross-entropy of the form $X_{k,i} = \sum_{j=1}^{10} -y_{i,j} \ln(\Phi(x_i, \beta)_j)$. Since $\sigma^2 = \operatorname{Var}(Y_k) = \operatorname{Var}(X_{k-1}) + \operatorname{Var}(X_k) - 2\operatorname{Cov}(X_{k-1}, X_k)$, we can estimate σ^2 with the classical estimators of the variance and the covariance using the realizations $\{X_{k-1,i}\}$ and $\{X_{k,i}\}$. The statistic T_k is then compared with the $(1-\alpha)$-th quantile of the Student t-distributions with $N-1$ degrees of freedom, denoted $q_{1-\alpha}$. If $T_k \leq q_{1-\alpha}$, the algorithm is stopped. The parameter ϵ_2 aims at quantifying what we mean by "insufficient decrease in the objective function" by comparing the actual decrease with a value proportional to the last value of the objective function: $\epsilon_2 f(\beta_{k-1})$. In the case of the RA framework, we will use the value $\epsilon_2 = 10^{-4}$ except for the last iteration (with all the observations) where we use a value of $\epsilon_2 = 0$ meaning that we stop if there is no decrease. In the other cases, we use $\epsilon_2 = 0$. We also use a value of $\alpha = 0.05$.

The third stopping criterion is the cross-validation (criterion 3). If the RA is used, this criterion will only be applied for the last iteration while the criterion 2 is applied for the other iteration. For the cross-validation, the 70000 images are divided into a training set of 50000 images, a test set of 10000 images and a validation set of 10000 images. After each iteration of the trust region algorithm, the loss function is evaluated over the validation set. If the value obtained over the validation test increase between two iterations of Algorithm 1, the algorithm is stopped. The objective is to avoid overfitting by stopping the algorithm when there is no more improvement over the validation set.

3.3.2 Results

Our first test aims at comparing the three criteria explained above, using the HOPS method to approximate the Hessian. The results are summarized in the table 3. For criterion 2 and criterion 3, the values are averaged over two runs where, for each run, the 70000 images are randomly divided between the training set and the test set (and the validation set for criterion 3) and the initial value of parameters β_0 is randomly chosen according to a centered normal law of standard deviation equal to the inverse square root of the size of the input layer.

	Comp. time (s)	Nb of iter.	Alloc. (GiB)	Accuracy over the train. set (%)	Accuracy over the test set (%)
crit. 1	34174	983	46184	93.59	92.60
crit. 2	668	19	1101	92.42	91.92
crit. 3	571	23	1033	92.36	91.88

Table 3: Stopping criteria

The criterion 1 needs a very high number of iterations to stop the algorithm but reaches a high accuracy. The algorithm quickly reaches good accuracy but then requires a long running time with few improvements of the accuracy before reaching the stopping criterion 1. The criteria 2 and 3 aims at introducing an early stopping. These last two criteria give similar performances with a fast convergence and a good accuracy close to that of criterion 1. In the following, we use the criterion 2.

In the table 4, we compare the HOPS method with the BFGS method over the 60000 examples of the training set, using the stopping criterion 2.

	BFGS	HOPS
Computation time (s)	12459	553
Nb of iterations	161	14
Allocations (GiB)	16174	926
Accuracy over the training set (%)	93.60	92.37
Accuracy over the test set (%)	92.69	92.12

Table 4: Comparison between BFGS and HOPS

With this stopping criterion, BFGS converges slowly but achieves high accuracy. HOPS converges very quickly wile the accuracy remains close to the result of BFGS. It is possible that BFGS reaches quickly good accuracy but carries on significant decrease in the objective function with only small improvements in the accuracy, while with HOPS, the objective function quickly shows no more significant decrease and then the algorithm stops.

We cannot obtain results from the BHHH methods because it is far too slow here. The reason is that the construction of the Hessian approximation with the sum of outer products of scores is extremely slow as it deals with $7850^2 = 61622500$ matrix elements for each outer product. With our implementation, the computation of one outer product (for only one image) requires 500 ms, so the method is unusable over the 60000 images. The point is that we use here a very naive approach to deal with matrices. For example, it should be possible to take advantage of the sparse property of the outer product of scores to obtain more efficient computation of the Hessian approximation, however, this idea is not investigated here. It is also possible to use only an subsample of the 60000 images to compute the Hessian approximation. This technique has not been investigated with the BHHH method but some testings have been conducted with the HOPS methods and can be found below.

In the following testings, we use the HOPS method within the RA framework (see section 2.3) with the stopping criterion 2. For the increase in sample size in the RA, we use $N_{l+1} = KN_l$ with $K = 4$ and $N_0 = 3750$ ($N_0 = 3125$ when applying the cross-validation so that the algorithm uses the full sample at the last iteration). Those values are chosen empirically by making a trade-off between the computing time, the final accuracy and a limited overfitting. The overfitting can be visualized by the increase in the objective function between two iterations of the RA (see figure 1 in the annex). Note that since we have a limited number of observations available, the samples used at each iteration of the RA algorithm are not independent (in fact each sample contains the sample of the previous iteration). Note also that the $\{\epsilon_l\}$ introduced in Algorithm 3 can be linked with the statistic defined in (17) used in the stopping criterion 2: as the subsample size N increases, the value of T_k is increased and the test is more easily rejected, such that a better accuracy is needed to stop the trust region algorithm.

We present in the table 5 some results about the use of a subsample for the computation of the scores, while the full sample is used for the computation of the objective function and for the gradient. The coefficient c is, for each iteration of the RA, the proportion of the current sample used for the computation of the scores. While it is possible to use a different subsample at each iteration of the trust region algorithm so that the full sample is covered after a certain number of iteration, we use here the more simple method of splitting the sample according to the coefficient c and to use the same subsample at each iteration to compute the score. The results are averaged over two runs with the same characteristics as used for criteria 2 and 3 in the table 3. Note that the number of iterations is the total number of the trust region algorithm iterations over all the iterations of the RA algorithm.

	$c = 1$	$c = 0.75$	$c = 0.5$	$c = 0.25$
Computation time (s)	616	517	302	124
Nb of iterations	31.5	40.5	52.5	60
Accuracy over the training set (%)	93.29	93.50	93.19	92.03
Accuracy over the test set (%)	92.23	92.29	92.20	91.63

Table 5: Subsampling for the approximate Hessian calculation

By comparing the first column ($c = 1$) with the criterion 2 of the table 3, we first note that this implementation of the RA framework leads to some improvements in the accuracy, within a similar computing time. Otherwise, the results tends to show that a subsampling in the computation of the scores leads to significant running time decrease while the loss in accuracy is negligible except for high subsampling (here $c = 0.25$). Another testing not performed here could be to fix the available computing time and to measure the accuracy reached according to the value of c.

We now present a final testing where we use the HOPS method (in the RA framework) using a multilayer perceptron with one or two hidden layers. We use the stopping criterion 2. The aim is to test the behaviour of the method when the number of parameters increase. We briefly compare several values of the size and the number of hidden layers, and in the coefficient of subsampling c. We also conduct a run with the criterion 3 (cross-validation). Finally, we present a run with L2 regularization (see paragraph 3.2 of [9]) achieving the best accuracy. L2 regularization consists in modifying the loss function introduced in (16) as follows:

$$L_{L2}(\beta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{10} -y_{i,j}\ln(\Phi(x_i,\beta)_j) + \frac{\lambda}{2N}\sum_{\beta_i \in \mathscr{W}} \beta_i^2$$

where λ is the regularization parameter (we use $\lambda = 0.1$) and $\mathscr{W}$ is a subset of the parameters containing only the weights of the network (not the biases). The gradient and the scores of L_{L_2} are computed by adding the gradient of $\frac{\lambda}{2N}\sum_{\beta_i \in \mathscr{W}} \beta_i^2$ to the former gradient and scores of the cross-entropy defined in (16). Since the objective function is no longer a log-likelihood function, the justification for the approximation of the Hessian presented in section 2.2 do not hold here, but in this case it seems that we nonetheless achieve better performances with regularization than without. The results are summarized in the table 6. The first column is a reference run, the changing parameters in each other column are emphasized.

By comparing Test 1 with Test 4, it seems that the performances slightly increase when c decreases, which is counter-intuitive. This trend seems also to appear in the table 5 between the first and the second column. This might be linked with overfitting. We see that increasing the number of neurons in the hidden layer (Test 2) improve the performances but not increasing the number of layers (Test 3). This may be due to the characteristics of the model and not to the optimization method. In Test 5, the cross-validation criterion has stopped the algorithm significantly early than in Test 1, while improvements in the accuracy were still possible since the accuracy in Test 1 is higher than in Test 5. Note that these testings have been conducted

Test n°	1	2	3	4	5	6
Nb hidden layers	1	1	*2*	1	1	1
Size hidden layers	100	*30*	100	100	100	100
Subsampling coeff. c	0.3	0.3	0.3	*0.5*	0.3	*0.8*
Stopping criterion	2	2	2	2	*3*	2
L2 regularization	no	no	no	no	no	*yes*
Comp. time (s)	4137	1153	8984	4144	2716	54357
Nb of iterations	148	137	179	71	118	47
Acc. train. set (%)	98.20	96.25	97.05	97.67	96.09	99.03
Acc. test set (%)	97.00	95.70	96.52	96.55	95.37	97.55

Table 6: Multilayer perceptron with hidden layers

over one run with a specific division between the training set and the test set and a specific initial point, so more experiments are required to bring out the real behaviour of the method.

4 Conclusion

In this report, we have presented a Hessian-free method to efficiently approximate the Hessian without storing any approximation matrix, by using directly the scores to compute matrix-vector product. This approximation is used in a trust region method framework, with the truncated conjugate gradient as the method to compute the trial point. This method is applicable in maximum likelihood estimation as far as in non-linear least squares regression. The first empirical results seems to show that this method could be used on problems with large datasets while other similar methods are generally less efficient in terms of computation time and memory allocations and cannot be efficiently applied when the number of observations is too large.

However, several testings are required to confirm these first impressions. Since our implementation of BHHH do not rely on the sparse property of the outer product of scores, BHHH might still give performances similar to HOPS. Some comparisons are also possible with other quasi-Newton method (like SR1, see chapter 6 in [10]) and also with the stochastic gradient descent and its variants (see chapter 8 in [5]). Another comparison framework not presented in this report may be to compare the efficiency of the methods by the accuracy reached among a fixed computation time. In terms of implementation, all the testings presented here have been relying only on the CPU, while in this type of framework, parallel computing using the GPU is highly pertinent. Finally, the HOPS method should be tested on larger instances, in particular when the number of parameters is greater than the number of observations.

Annex

In the figure 1, the evolution of the objective function in the RA algorithm is plotted for two values of N_0 and K (see 2.3) using the criterion 2 (see 3.3).

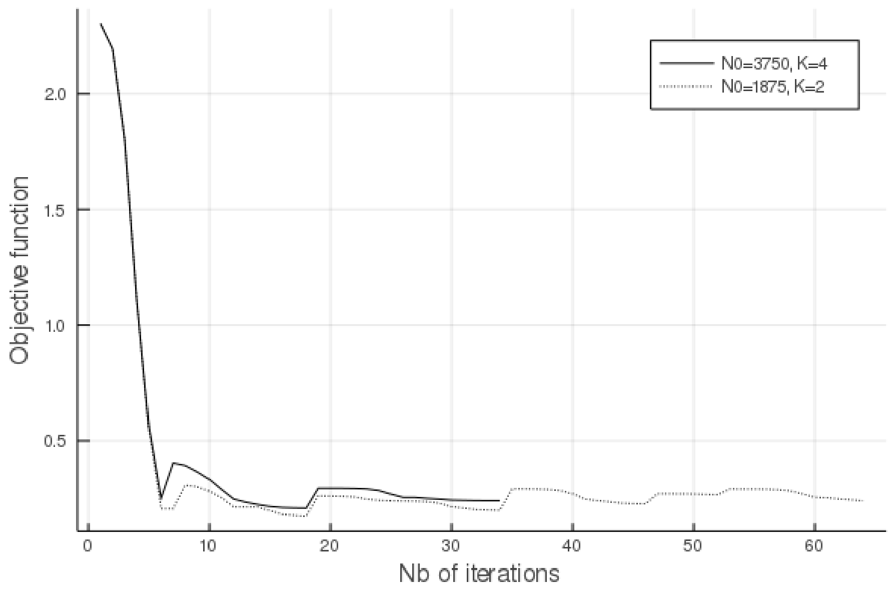

Figure 1: Evolution of the objective function during the retrospective approximation algorithm

In section 2.3, we introduced a multiplicative sample size policy for the RA algorithm. We give here the result of a test where we use another method for the increase of the sample size. This method, called S-SSCP, has been proposed in [13]. Instead of using a error-tolerance up to which the sample-path problem should be solved, S-SSCP defines and solves a control problem at the beginning of each iteration of the RA algorithm to determine the sample size and the number of iterations that should be performed at this iteration. The control problem is defined in such a way that it optimizes a criteria to balance the computation time with the convergence speed, more details in [13]. Table 7 gives the result for a run with the simple perceptron model defined in section 3.3 and with an initial sample size of $N_0 = 1000$. The other hyperparameters are set to the same values as proposed in [13]. The stopping criterion used is a cross-validation where we compute the accuracy over a validation set after each iteration of the RA algorithm and stop when the accuracy decreases.

Comp. time (s)	Nb of iter.	Accuracy over the train. set (%)	Accuracy over the test set (%)
4474	310	93.78	92.43

Table 7: S-SSCP

The accuracy is slightly higher than HOPS in the second column of table 4, but the computing time is multiplied by 8. In fact, the S-SSCP method quickly increases the sample size up to the limit (whereas the method is designed to rely on limitless independent observations) and so may have needed larger datasets to achieve better performances.

References

[1] E. K. Berndt, B. H. Hall, R. E. Hall, and J. A. Hausman. Estimation and inference in nonlinear structural models. *Annals of Economic and Social Measurement*, 3(4):653–665, 1974.

[2] J. L. Chartrand. `https://github.com/JLChartrand/`.

[3] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*. SIAM, 2000.

[4] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, 1996.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] S. Kim, R. Pasupathy, and S. G. Henderson. A guide to sample average approximation. In M. C. Fu, editor, *Handbook of Simulation Optimization*, chapter 8, pages 207–243. Springer, 2015.

[7] Y. LeCun, C. Cortes, and C. J. C. Burges. http://yann.lecun.com/exdb/mnist/.

[8] W. K. Newey and D. McFadden. Large sample estimation and hypothesis testing. In R. Engle and D. McFadden, editors, *Handbook of econometrics*, chapter 36, pages 2111–2245. Elsevier, 1994.

[9] M. A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015. http://neuralnetworksanddeeplearning.com/.

[10] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, second edition, 2006.

[11] R. Pasupathy. On choosing parameters in retrospective-approximation algorithms for stochastic root finding and simulation optimization. *Operations Research*, 58(4):889–901, 2010.

[12] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*, 2016.

[13] J. O. Royset. On sample size control in sample average approximations for solving smooth stochastic programs. *Computational Optimization and Applications*, 55(2):265–309, 2013.

[14] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, June 1983.

[15] Ph. Toint. Towards an efficient sparsity exploiting newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88. Academic press, 1981.

[16] K. Train. *Discrete Choice Methods with Simulation.* Cambridge University Press, 2002.