Student: Sebastianelli Nicola

Inexact Newton Method Implementation

Distributed Optimization and Games Project

Abstract: Optimization is one of the most important part of the machine learning, and the most famous used algorithms are the 1st order gradient-based methods. Recently, second order methods have become attractive, however those methods can be not so efficient due to the difficulties related to the computation of the Hessian. In this project we are going to study and analyze one of those second order methods, the Inexact Newton Method and implement one of its variants, the Subsampled Hessian-Free Newton Method.

Introduction: The simplest algorithm used to perform iterative minimization is the gradient descent. This method is part of the first order methods due to the fact that it take advantage of the gradient properties in order to reach a local minimum. Sometimes this could be not enough and impose significant limitations in particular because the first order methods don't take into account the curvature of the error function causing a slow model training. Another reason that could lead us to consider second order methods is the fact that first order methods are not scale invariant and so different scale of data, also affect the behavior of the algorithm. The approximation performed by the second order methods is a quadratic approximation because it take into account the curvature of the error surface

Inexact Newton Method: One of the most famous second order method is the Newton method. The Newton method is a method for finding iteratively better approximation of the roots of a given function. This method allows to update the parameters considering also the second derivative: $w_{k+1} = w_k - \alpha_k \frac{F\prime(w_k)}{F\prime\prime(w_k)}$ in one dimensional case and $w_{k+1} = w_k - \alpha_k \left(\nabla^2 F(w_k)\right)^{-1} \nabla F(w_k)$ in a multidimensional case. The update can also be seen as $\omega_{k+1} \leftarrow \omega_k + \alpha_k s_k$, where s_k satisfies $\nabla^2 F(\omega_k) s_k = -\nabla F(\omega_k)$. Thanks to the scale invariance and quadratic rate convergence properties, Newton method is a good candidate to be an optimization algorithm. Problems raise with this method when it is applied to a multidimensional problem. The calculation and inversion of the second derivative of the error function that create the Hessian matrix is computationally expensive. To face this problem, some variants of this method that allow to scale the computation while maintaining the benefits of the second order methods has been studied. One of those method is called Inexact Newton Method, that allow to find the s_k in an inexact way using the Hessian-vector with the conjugate gradient method (described below). The benefit of this approach is that is not required to access the Hessian matrix but is just needed the Hessian-vector. The family of algorithm that take advantage of the Hessian-vector is called Hessian-free. To better understand this method, we are going to take into account the following example:
Parameter vector $\omega = (\omega_1, \omega_2)$, $F(\omega) = \omega_1^2 \omega_2^2$ we will have $\phi(\omega, d) = \nabla F(\omega)^T s = 2\omega_1 \omega_2^2 s_1 + 2\omega_2 \omega_1^2 s_2$ and then $\nabla_\omega \phi(\omega, s) = \nabla^2 F(\omega) s = \begin{bmatrix} 2\omega_2^2 s_1 + 4\omega_2 \omega_1 s_2 \\ 4\omega_2 \omega_1 s_1 + 2\omega_1^2 s_2 \end{bmatrix}$. Using so $\nabla_\omega \phi(\omega, s)$ inside the CG instead of calculate the entire hessian give us a significant optimization in term of computation. [1][2][3]

Conjugate gradient method: The Conjugate gradient (CG) is an algorithm to find numerical solutions of x on the form $Ax = b \rightarrow \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$, where A is a $n\,x\,n$ positive defined matrix, x is the vector we want to approximate and b is a known vector. In our case, $Ax = \nabla^2 F(\omega) s$ and $b = -\nabla F(\omega_k)$. Conjugate gradient assumptions are that the matrix A has to be symmetric and positive defined. The iterative method of the conjugate gradient method is implemented as followed:

Conjugate Gradient algorithm:
$r_0 \coloneqq b - Ax_0$
$if\ r_0\ sufficiently\ small, then\ return\ \ x_0\ as\ result$

$p_0 \coloneqq r_0$
$k \coloneqq 0$
$repeat\ \ \alpha_k \coloneqq \frac{r_k^T r_k}{p_k^T A p_k}$
$x_{k+1} \coloneqq x_k - \alpha_k p_k$
$r_{k+1} \coloneqq r_k - \alpha_k A p_k$
$if\ r_{k+1}\ sufficiently\ small, then\ exit\ loop$
$\beta_k \coloneqq \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
$p_{k+1} \coloneqq r_{k+1} + \beta_k p_k$
$k \coloneqq k + 1$
$end\ repeat$
$return\ x_{k+1}$

Let r_k be the residual, p_k be the conjugate vector, α_k a step size chosen such that r_{k+1} is orthogonal to r_k and β_k a step size such that p_{k+1} is conjugated to p_k. This algorithm is will return the update of the parameter vector, so it will be a core part for the computation of the subsampled Hessian-free Newton method. [4][5]

Subsampled Hessian-Free Newton Method: The Conjugate gradient method is a flexible solution because allow us to control the number of the CG iteration, but if we don't have preconditions this method can be still expensive because one Hessian-vector product is as same cost as computing one gradient. A proposed, more efficient method is the Subsampled Hessian-Free Newton Method. This method is based on the fact that computing the Hessian-vectors in all the dataset is still computationally expensive. Instead we will use a smaller sample $\mathcal{S}_k$. This subsample should be taken small enough so that the computation cost can be significantly reduced but on the other hand it should be large enough in order to have a good quality of the curvature information obtained by the Hessian-vector. One of the main difficulties of this method is to find a proper value of $\mathcal{S}_k$ that can provide a good compromise between computation cost and quality of the curvature. Following the proposed idea, we will have:

$$\nabla f_s(w_k; \xi_k) = \frac{1}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \nabla f_s(w_k; \xi_{k,i})$$

And the stochastic Hessian estimate be

$$\nabla^2 f_s(w_k; \xi_k^H) = \frac{1}{|\mathcal{S}_k^H|} \sum_{i \in \mathcal{S}_k^H} \nabla^2 f_s(w_k; \xi_{k,i})$$

Where the sample of the Hessian estimation $\mathcal{S}_{\mathrm{k}}^{\mathrm{H}}$ is uncorrelated to $\mathcal{S}_{\mathrm{k}}$ such that $|\mathcal{S}_{\mathrm{k}}^{\mathrm{H}}| < |\mathcal{S}_{\mathrm{k}}|$. From this, we can compute

<u>Subsampled Hessian-Free Algorithm:</u>

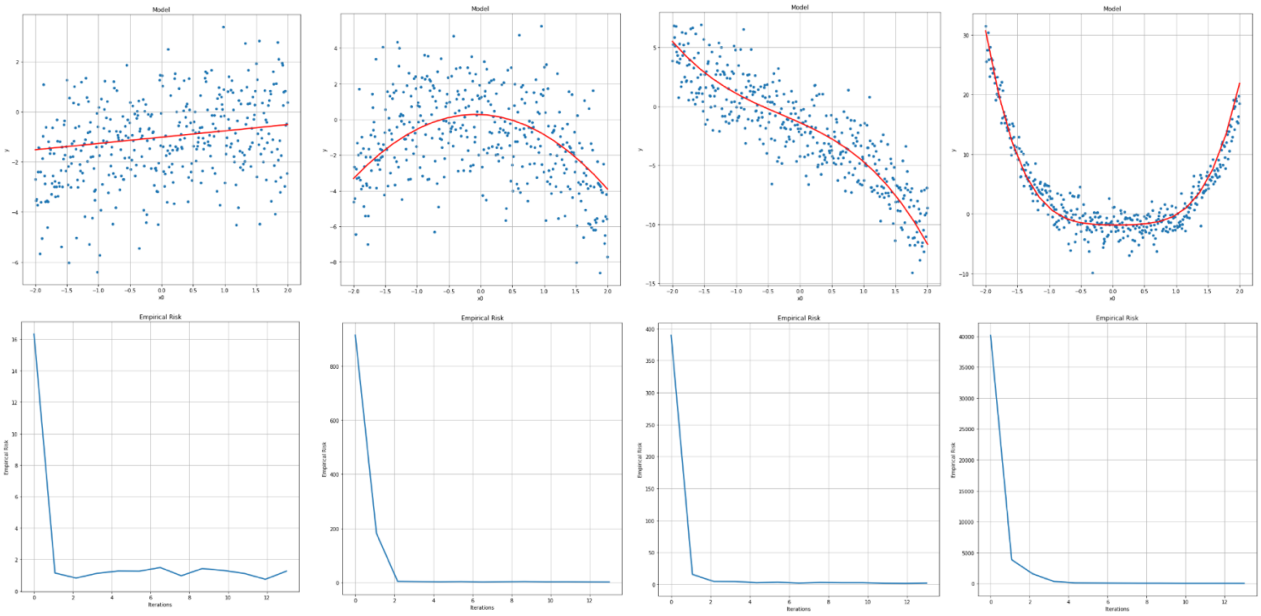
$Chose\ an\ initial\ iterate\ w_1$
$Chose\ constant\ \rho \in (0,1), \gamma \in (0,1), \eta \in (0,1)\ and\ max_{cg}\ \in \mathbb{N}$
$for\ k = 1,2, \dots do$
$generate\ realization\ of\ \xi_k and\ \xi_k^H\ corresponding\ to\ \mathcal{S}_{\mathrm{k}}\ and\ \mathcal{S}_{\mathrm{k}}^{\mathrm{H}}$
$compute\ s_k by\ applying\ Hessian\ free\ CG\ to\ solve$
$$\nabla^2 f_s(w_k; \xi_k^H) s_k = -\nabla f_s(w_k; \xi_k)$$
$until\ max_{cg}\ iterations\ have\ been\ performed\ or\ a\ trial\ solution\ yelds$
$set\ \omega_{k+1} \leftarrow \omega_k + \alpha_k s_k\ \ where\ \alpha_k \in \{\gamma^0, \gamma^1, \gamma^2, \dots\}$ is the largest element with

$$f_{s_k}(w_{k+1}; \xi_k) \leq f_{s_k}(w_k; \xi_k) + \eta \alpha_k \nabla f_{s_k}(w_k; \xi_k)^T s_k$$

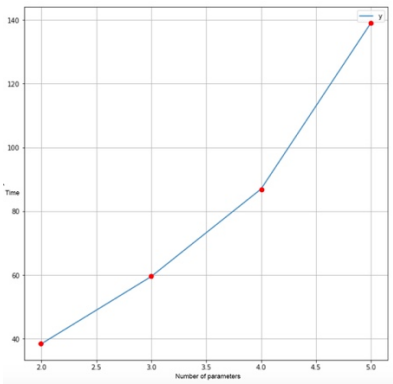
As we can see, this algorithm computes the Hessian-vector in a subsample instead of the entire dataset and dynamically adapt the learning rate selecting the biggest one that respect the above given constraint to have an optimal step for the current iteration. The constraint tells us that the learning rate that we want to take is the largest that maintain the future error is smaller than the current error plus the gradient of the current error multiplied by the current found step, learning rate α_k and a constant $\eta \in (0,1)$ [1][2][3]

Implementation & Tests: In the implementation phase I decided to use sympy python library in order to perform differentiation and maintain a symbolic form of the functions. I have developed the previously mentioned algorithm: Conjugate Gradient algorithm and Subsampled Hessian-Free Algorithm creating also an adaptation if the conjugate gradient for the Newton case that take as input the Hessian-vector and the gradient. I have tested both the CG algorithms to decide how many iterations were needed for obtaining an optimal approximation of the predicted vector. Picking as max number of internal iterations the number of elements that compose the parameter vector is enough for obtain a really good approximated result. In the Subsampled Hessian-Free Algorithm instead is possible to tune the size of the samples $\mathcal{S}_k$ and $\mathcal{S}_k^H$. So after tuning the CG method and Subsampled Hessian-Free Algorithm, for testing the behavior of this algorithm I have built a method that generate a function from a given wanted number of parameters. That function is then used to generate the dataset with the introduction of some noise. Once generated the datasets I tested the Subsampled Hessian-Free Algorithm for different number of parameters obtaining as result:

2 used parameters | 3 used parameters | 4 used parameters | 5 used parameters



As we can see from the empirical risk graph, this method obtains really good accuracy after just some iteration that are less than the number of used parameters showing the error tolerance and benefits obtained of the second order methods. Analyzing the computation time then, we can see that increasing the parameter number, the computation time grow exponentially making this algorithm not suited for really large number of parameters.



# of parameters	2	3	4	5
Computation time in seconds	38.401	59.503	86.967	139.062

Bibliography:

[1] L. Bottou, F. E. Curtis, J. Nocedal. « Optimization Methods for Large-Scale Machine Learning ». [Online]. Available: https://arxiv.org/pdf/1606.04838.pdf

[2] R. Bollapragada, R. Byrd, J. Nocedal. « Exact and Inexact Subsampled Newton Methods for Optimization ». [Online]. Available: https://arxiv.org/pdf/1609.08502.pdf

[3] R. H. Byrd, G. M. Chin, W. Neveitt, J. Nocedal. « On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning ». [Online]. Available: http://users.iems.northwestern.edu/~nocedal/PDFfiles/ssnewt.pdf

[4] S. Bubeck, Microsoft Research. « Convex Optimization: Algorithms and Complexity ». [Online]. Available: https://arxiv.org/pdf/1405.4980.pdf

[5] M. R. Hestenes, E. Stiefel . « Methods of Conjugate Gradients for Solving Linear Systems Complexity ». [Online]. Available: http://users.iems.northwestern.edu/~nocedal/PDFfiles/ssnewt.pdf