

Tested Paradigm to Include Optimization in Machine Learning Algorithms

Aishwarya Alesh

School of Computing Science and Engineering
VIT University
Vellore, India

Abstract—Optimization is considered to be one of the pillars of statistical learning and also plays a major role in the design and development of intelligent systems such as search engines, recommender systems, and speech and image recognition software. Machine Learning is the study that gives the computers the ability to learn and also the ability to think without being explicitly programmed. A computer is said to learn from an experience with respect to a specified task and its performance related to that task. The machine learning algorithms are applied to the problems to reduce efforts. Machine learning algorithms are used for manipulating the data and predict the output for the new data with high precision and low uncertainty. The optimization algorithms are used to make rational decisions in an environment of uncertainty and imprecision. In this paper a methodology is presented to use the efficient optimization algorithm as an alternative for the gradient descent machine learning algorithm as an optimization algorithm.

Keywords—BFGS; Cost Function; Data Analysis

I. INTRODUCTION

Machine Learning is a field that grew out of artificial intelligence giving new capabilities for computers. In a world of high uncertainty and imprecision the decisions should be taken which provide results in an efficient way. For this purpose machine learning algorithms are used. The amount of data in the world seems increasing and computers make it easy to save the data. As the volume of data increases, inexorably, the proportion of it that people understand decreases alarmingly [5]. When there are very large amounts of data, there is a need to use the efficient algorithms for data analysis and providing accurate results.

Depending on the types of data, machine learning algorithms are further classified into supervised learning algorithm, unsupervised learning algorithm, reinforcement learning algorithm and others. Supervised learning algorithm refers to the fact that given an algorithm a data set and the task of the algorithm is to produce the right answers. This is also called a regression problem. Regression problem provides a continuous valued output. The term classification in the supervised learning refers to the fact that the algorithm is trying to predict a discrete valued output. In a supervised learning algorithm, the data is a set of training examples with the associated correct answers. The algorithm learns to predict the correct answer from this training set. An example of this would be learning to predict whether an email is spam if given a million emails, each of which is labelled as spam or non-spam. Unsupervised learning algorithm is also called as a

clustering algorithm. Unsupervised learning algorithm breaks the data into clusters. In unsupervised learning algorithm the algorithm can find the trends in the data it is given without looking for some specific correct answer. The dataset is given as input to a learning algorithm and the hypothesis is formed. The hypothesis is a function that takes the dataset and gives the output precisely. The hypothesis can be represented as a linear function and nonlinear function depending on the type of data and the amount of datasets. A models performance is quantified by cost function. A cost function is computed using some parameters and then it is minimized to find the correct values for these parameters. Based on the values found by minimizing the cost function the hypothesis function is formed. An optimization problem is generally the problem of finding the best solution from all feasible solutions. Gradient descent and BFGS are the optimization problems. The gradient descent algorithm is used for finding the parameters. Gradient descent algorithm is an optimization algorithm used in many applications of machine learning. The aim of this paper is to explore the various stages involved in implementing optimization methods and choosing the appropriate one for a given task. From an optimization point of view learning in a neural network is equivalent to minimizing a global error function which is multivariate function that depends on the weights in the network [9].

II. GRADIENT DESCENT ALGORITHM

i. Gradient Descent Algorithm

Gradient descent algorithm offers a very good perspective for solving problems related to data analysis. It is an algorithm that is also used to minimize functions. Gradient descent algorithm when given a function with an initial set of parameter values starts the procedure iteratively and moves towards the parameter values that minimize the function. The iterative minimization is achieved by taking steps in the negative direction of the function gradient. The gradient descent can take many iterations to compute a precise local minimum. There are a large number of people still using gradient descent on neural networks and many other architectures. Gradient descent algorithm exploits the datasets given by the derivative of the function that is to be minimized. The goal of linear regression function is to fit a line corresponding to the dataset.

The datasets can be plotted on a graph using the standard line equation $y = mx+c$ where m is the slope of the line and b is the y intercept of the line. The best line that fits the given datasets can be found by finding the best set of slope m and y intercept b . A standard approach to solve this type of problem

is to define the cost function that takes the data sets as the input and returns the error value based on how well the line fits the dataset. The hypothesis function picks the input data and predicts the output for that input precisely. The sum the square of distances between each point's y value and the candidate line's y value is calculated for computing the error for the line that fits the dataset by iterating through each point in the dataset. It is always suggested to square this distance to ensure that it is positive and to make the cost function differentiable. Gradient descent can also be used to solve the system of nonlinear equations. Gradient descent is an iterative optimization procedure that uses this information to adjust a function's parameters. It takes the value of the derivative, multiplies it by a small constant called as learning rate and subtracts the result from the current parameter value. This is repeated for the new parameter value and so on until a minimum is reached. The datasets along with the standard line are pointed on a graph as shown in figure 1 and figure 2.

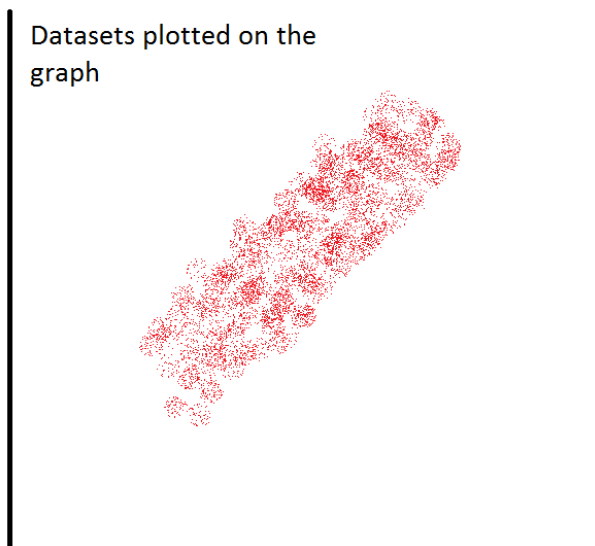


Figure 1

When the function is minimized the best line that fits the data is drawn over the dataset. The equation for gradient descent algorithm after partial differentiation can be calculated by following the above procedure and is shown below.

$$(\theta)_j := (\theta)_j - (\alpha/m) * \sum_{i=1}^m (h(\theta)(x(i)) - y(i))x_j(i) \text{ (for all } j)$$

The values of theta are to be updated simultaneously. Here m is the number of training sets or data sets and theta is the parameter. If alpha is small then the algorithm takes small steps. If it is larger the algorithm takes big steps. It is also called as batch gradient descent. Each step of the gradient descent uses all the training examples.

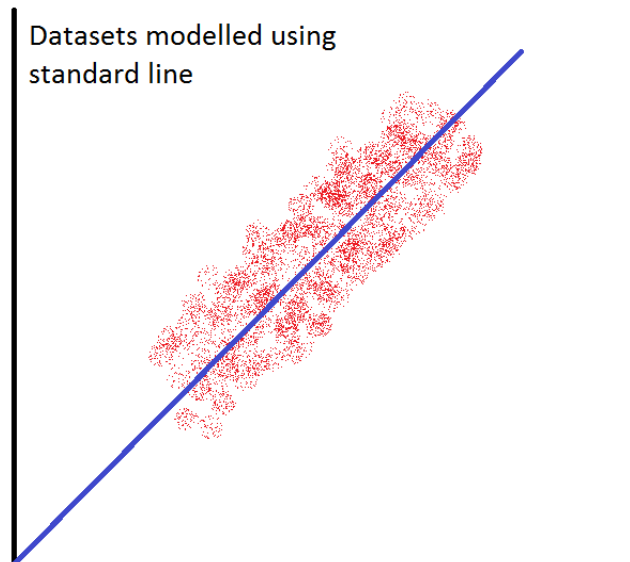


Figure 2

ii. Computational Example

The learning rate determines the step size and hence how quickly the search converges. If it is too large and the cost function has several minima, the search will overshoot and miss a minimum entirely. If it is too small, progress toward the minimum may be slow. The gradient descent can only find a local minimum. The example of the gradient descent algorithm implemented in octave and the screenshot of the code for the cost function and the values of the parameters are shown below in figure 3 and figure 4.

```

File Edit Selection Find View Goto Tools Project
costfunctionj.m x
1 function j = costfunctionj(X,y,theta)
2 m = size(X,1);
3 predictions = X*theta;
4 sqerror=(predictions-y).^2;
5 j=1/(2*m)*sum(sqerror);

```

Figure 3

Taking the values of the function as

$$X = [1 \ 1; 1 \ 2; 1 \ 3];$$

$$y = [1; 2; 3];$$

$$\theta = [0; 1];$$

The output cost function value after applying gradient algorithm is zero. The corresponding screenshot of the implementation in octave is shown in figure 4.

```

octave-3.2.4.exe:2> X=[1 1;1 2;1 3]
X =
    1    1
    1    2
    1    3

octave-3.2.4.exe:3> y=[1;2;3]
y =
    1
    2
    3

octave-3.2.4.exe:4> theta=[0;1]
theta =
    0
    1

octave-3.2.4.exe:5> j=costfunctionj(X,y,theta)
j = 0
octave-3.2.4.exe:6>

```

Figure 4

In this way the algorithm should be implemented for different values of theta. For different values of theta the corresponding cost function values are computed. All the values of cost function are noted down and the lowest value among them is considered to be the best value. The values of theta for which the cost function's value is least are the correct values for the hypothesis function that gives the gradient descent boundary.

iii. Disadvantages of Gradient Descent

There are equal disadvantages of using gradient descent algorithm on the datasets available. The disadvantages are as given below.

1. The value of step size alpha has to be experimentally chosen in the gradient descent algorithm.
2. Calculating partial derivatives of the cost function is mandatory.
3. The number of iterations should be chosen experimentally in the gradient descent algorithm.
4. Gradient descent algorithm takes time to converge to the correct point based on the value of alpha.

Gradient descent is relatively a slow algorithm when compared with the others. As the number of training examples increases the asymptotic rate of convergence becomes very slow. Gradient descent increasingly zigzags for poorly conditioned convex problems. For non-differentiable functions, gradient methods are ill-defined and difficult to find the local minimum. Gradient descent can be slow. Taking infinitesimal steps in the direction of the gradient would take a lot of time to compute, so finite step sizes must be used to compute faster. The value of precise step size is unclear. Many algorithms have been developed by people for adjusting the step size. Many of these algorithms are not robust to noise and they scale badly with the number of parameters. Gradient descent where the step size is adjusted by a simple momentum heuristic is used by many people. Gradient descent on many architectures does not result in a global optima.

III. BROYDEN-FLETCHER-GOLDFARB-SHANNO ALGORITHM

i. BFGS Algorithm

The family of Quasi-Newton methods to mimic Newton's Method, so it is called as Quasi-Newton method. The approximations of the Hessian based on gradient and past update step information are used in this method. Quasi-Newton methods are faster per every iteration than the Newton's method as they are not explicitly computing the Hessian. Newton's method is an alternative to the gradient descent methods for fast optimization and convergence. Newton's method often converges faster than gradient descent methods. In the gradient descent algorithm, if the hypothesis function is non-linear function and there are enormous amounts of datasets then the algorithm would take lot of time to compute the cost function values and also takes time in choosing the learning rate. It will become a difficult process to calculate the parameter values for the non-linear function. Considering the above disadvantages for the gradient descent algorithm for computing cost function, Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm can be used as an alternative for gradient descent algorithm. Due to their combination of computational efficiency and asymptotic convergence, the BFGS quasi-Newton method and its memory-limited LBFGS variant are considered to be the efficient algorithms for nonlinear optimization. For constructing the Hessian matrix, the BFGS method approximates the objective function locally as a quadratic without evaluating the second partial derivatives of the objective function. The Hessian matrix is approximated by the previous gradient evaluations, such that there won't be any vertical scalability issue when computing the Hessian matrix in the Newton's method. As a result, BFGS often achieves faster convergence compared with other first-order optimization techniques.

The BFGS algorithm techniques can be used in various machine learning algorithms such as Linear Regression and Logistic Regression by passing the gradient of objective function and the updater into optimizer instead of using the training application programming interfaces. Application of second order methods is not suggestible practically because computation of the objective function Hessian inverses amounts for a very high computational cost. BFGS modifies gradient descent algorithm by introducing a Hessian approximation matrix computed from finite gradient differences. The BFGS method is considered as one of the most popular and efficient algorithms of this class. The L-BFGS is a limited-memory version of BFGS algorithm and is particularly used for the problems with very large numbers of variables. The L-BFGS algorithm should be modified to handle the functions that include non-differentiable components and constraints as the BFGS is designed to minimize the functions without constraints. These methods of modifying the algorithms are called active set methods as it is based on the concept of the active set. The idea is that the function and constraints can be simplified when restricted to a small neighborhood of the current iterate.

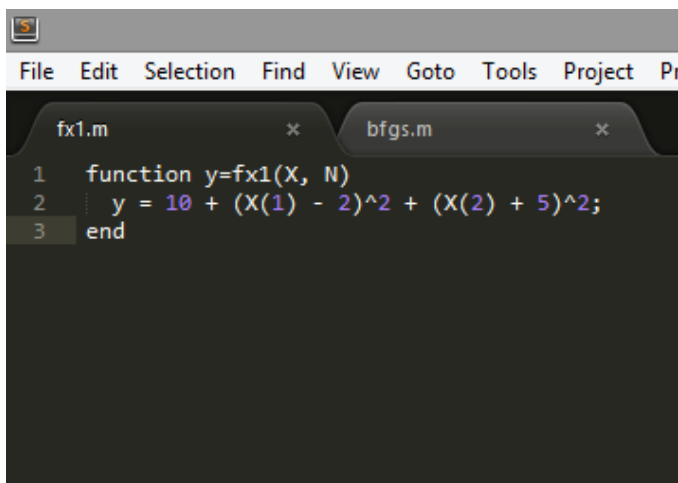
ii. Computational Example

BFGS method is based on Newton's method but performs different calculations in an efficient way.

Here is a sample session to find the optimum for the following function:

$$y = 10 + (X(1) - 2)^2 + (X(2) + 5)^2$$

The above function resides in file fx1.m. The search for the optimum 2 variables has the initial guess of [0 0] and with a minimum guess refinement vector [1e-5 1e-5]. The search employs a maximum of 100 iterations, a function tolerance of 1e-7, and a gradient tolerance of 1e-7. The corresponding screenshots of the function fx1 and the implementation of the BFGS algorithm in octave are shown below.

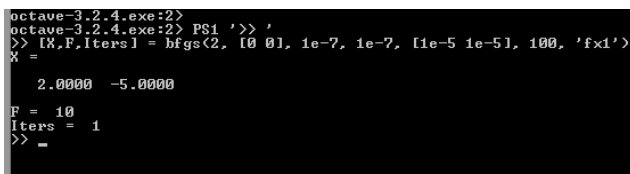


```

1 function y=fx1(X, N)
2   y = 10 + (X(1) - 2)^2 + (X(2) + 5)^2;
3 end

```

Figure 5



```

octave-3.2.4.exe:2>
octave-3.2.4.exe:2> P81 ';>>'
>> [X,F,Iters] = bfgs(2, [0 0], 1e-7, 1e-7, [1e-5 1e-5], 100, 'fx1')
X =
    2.0000   -5.0000
F = 10
Iters = 1
>>

```

Figure 6

IV. RESULTS

The BFGS algorithm has few advantages over the gradient descent algorithm such as BFGS increases the efficiency of the computation. There is no need to pick alpha manually using the BFGS algorithm. BFGS is often considered to be faster than the gradient descent. The only disadvantage with BFGS over gradient descent algorithm is that it is more complex. The BFGS method is used to train the hypothesis function properly and provide the accurate predictions for the new inputs by giving precise parameter values. Several experiments are performed and implemented using MATLAB and Octave and the results proved that the BFGS algorithm is efficient in finding the hypothesis and predictions than the Gradient Descent algorithm. The disadvantage of the optimization algorithms like BFGS and Conjugate Gradient are that they are more complex algorithms than the gradient descent algorithm and are difficult to understand and implement. BFGS helps in predicting the output for the

nonlinear second order equations. Each BFGS iteration will be more expensive but it takes fewer of the iterations to reach a local minimum.

V. CONCLUSION

BFGS is a quasi-Newton method and will converge in fewer steps than gradient descent and has a little less tendency to get stuck while performing computations. Indeed, it has been noted that in cases where BFGS does not encounter any non-smooth point, it often converges to the optimum [14]. Gradient descent algorithm computes matrix-vector products, which is useful if directional derivatives can be calculated while the BFGS performs vector-vector products. BFGS will calculate approximate Hessians using inner products of gradient information so the gradient descent analysis does not apply to BFGS. It is likely to get convergence in fewer iterations with BFGS than the gradient descent algorithm. From the above implementations of both the gradient descent and BFGS algorithms, BFGS algorithm will be faster and reaches local minimum in fewer steps. This paper described the Gradient Descent and BFGS methods in the context of linear systems and the relationships between each of the algorithms. This paper presents a future scope for the applications of optimization algorithms in the data analysis and predictions. It also gives the scope on how to use the optimization algorithms efficiently accordingly for producing more efficient and accurate results. The implementation of these algorithms demonstrate that BFGS is the best choice for well-conditioned problems because of its faster convergence to the local minimum. BFGS optimization algorithm is always to be chosen as an alternative for gradient descent accordingly for producing result as precisely as possible.

REFERENCES

- [1] Malouf, Robert (2002). "A comparison of algorithms for maximum entropy parameter estimation". Proc. Sixth Conf. on Natural Language Learning (CoNLL). pp. 49–55.
- [2] Andrew, Galen; Gao, Jianfeng (2007). "Scalable training of L₁-regularized log-linear models". Proceedings of the 24th International Conference on Machine Learning.
- [3] C.; Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge (1997). "L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization". ACM Transactions on Mathematical Software 23 (4): 550–560.
- [4] Fletcher, Roger (1987), Practical methods of optimization (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8.
- [5] Venkata Karthik Gullapalli and Aishwarya Asesh, Data Trawling and Security Strategies, ISSN – 2278-8727, IOSR Journal of Computer Engineering, Volume 16, Issue 6, Ver. 1, Nov - Dec 2014.
- [6] Danilo P Mandic, A Generalized Normalized Gradient Descent Algorithm, IEEE Signal Processing Letters, Vol. 11, No. 2, February 2004.
- [7] Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research, 4,933–969.
- [8] Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression Advances in Large Margin Classifiers, MIT Press (pp. 115–132).
- [9] Martin F. Moller, A Scaled Conjugate Gradient Algorithm for fast Supervised learning, ISSN 0105-8517, Daimi PB 339, November 1990.
- [10] D. E. Goldberg and J. H. Holland, Genetic Algorithms and machine learning, Guest Editorial, Machine Learning 3: 95-99, 1988 Kluwer Academic Publishers - The Netherlands.

- [11] R. Johnson and T. Zhang, Accelerating stochastic gradient descent using predictive variance reduction, *Adv. Neural Inf. Process. Syst.*, 26 (2013), 315–323.
- [12] Mokbnache L., Boubakeur A. (2002) “Comparison of Different Back-Propagation Algorithms used in The Diagnosis of Transformer Oil” *IEEE Annual Report Conference on Electrical Insulation and Dielectric Phenomena*, 244-247.
- [13] Charalambous C. (1992) Conjugate Gradient Algorithm for Efficient Training of Artificial Neural Networks, *IEEE Proceedings*, 139 (3), 301-310.
- [14] Jin Yu, S. V. N. Vishwanathan, Simon Gunter, Nicol N. Schraudolph, A Quasi-Newton Approach to Non Smooth Convex Optimization problems in Machine Learning, *Journal of Machine Learning Research*, March 2010.