Optimization Methods

Supervisor:	Posted:	Course:
Dr. Tsui-Wei Weng	Nov 9, 2022	DSC 210 FA’22 Numerical Linear Algebra

Team:

- Ryan Hammonds
  PID: A59018983
  HDSI
- Gabriel Riegner
  PID: A59017990
  HDSI
- Benjamin Pham
  PID: A12903873
  HDSI

1. Introduction

In statistics and machine learning, optimization is the process of tuning model parameters so that the estimated outputs or predictions are as close to the target values as possible. To quantify how well a model fits the data generating process, we define a **loss function**, which is a function $L(\mathbf{y}, \hat{\mathbf{y}})$ that measures the difference between the predictions $\hat{\mathbf{y}}$ and the targets $\mathbf{y}$. Optimization problems try to minimize this loss function with respect to tunable model parameters, averaged over repeated model trainings.

1.1 Applications:

Optimization methods can optimize the parameters across many machine learning algorithms, including deep neural networks. And the applications of these machine learning methods are numerous – computer vision, natural language processing, bioinformatics, and computational neuroscience to name a few. However, apart from using iterative optimization algorithms for minimizing loss functions, these optimization approaches can be used to identify minima and maxima across a range of high dimensional data problems. For example, the Newton method has been applied to routing phone calls, by finding the shortest path that connects a set of phone callers with a set of receivers. Additionally, the schedule of airport flights involves finding flight times that minimize costs like fuels, flight crew, etc. while maximizing the number of passengers. It soon becomes clear the utility that iterative optimization algorithms have in science and the industrial world.

2. Problem Formulation

2.1 Relation to numerical linear algebra:

To solve an optimization problem, one can use an exact solution or an approximation. Both types of methods include solving matrix inversion, linear systems of equations, matrix decomposition, and eigenvalue problems.

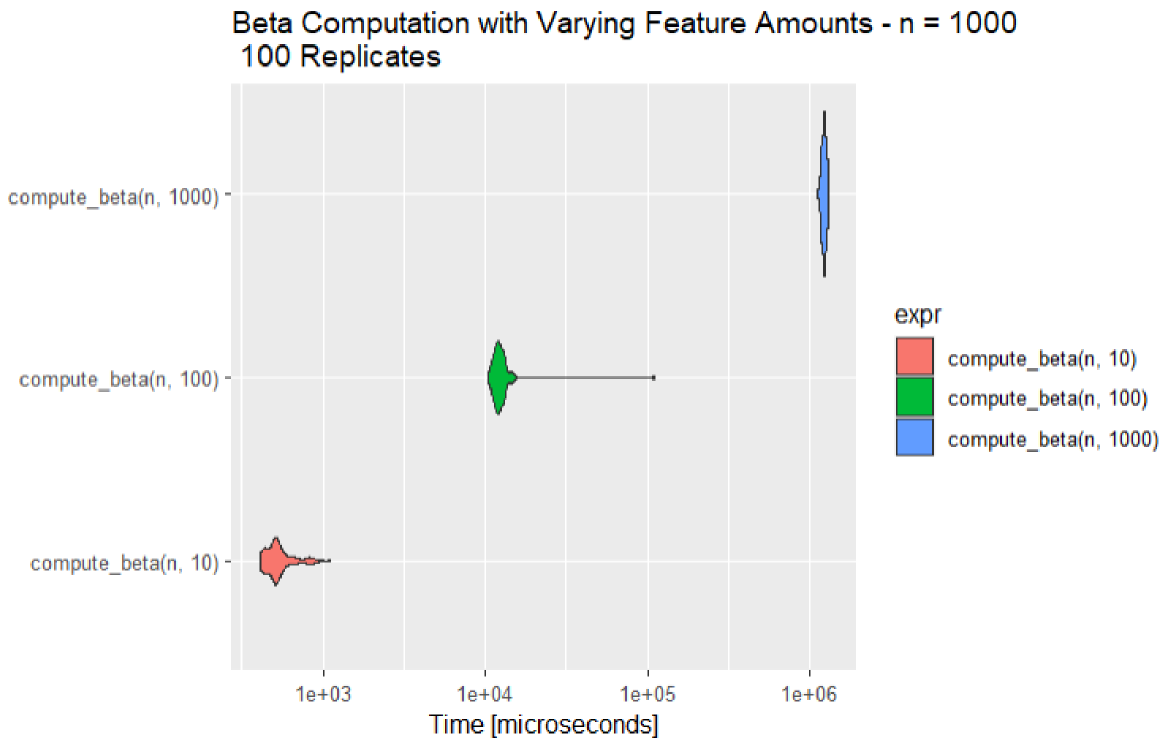
The relationship between numerical linear algebra and iterative optimization is bidirectional. On one hand, optimization algorithms can approximate linear algebra solutions when the computational complexity becomes too large to solve closed-form equations, as seen in the example of least squares. In this case, an iterative approach speeds up computation but sacrifices accuracy of the result. Additionally, when no algebraic solutions exist for optimization, iterative algorithms can often still find approximate solutions. On the other hand, the iterative algorithms like the Newton's method are built from numerical methods in linear algebra, including solving systems of equations and matrix inversions.

2.2 Approach description:

Objective

$$f(\mathbf{x}) = loss(\hat{\mathbf{y}}, \mathbf{y})$$

The objective of optimization typically involves minimizing a loss function, such as mean squared error or mean absolute error. The goal is to find a vector of unknown model parameters, $\mathbf{x}$, that predicts the known $\mathbf{y}$ as $\hat{\mathbf{y}}$ with minimal error. Few algorithms have closed-form solutions to this optimization problem. An unusual example is linear regression, which has an algebraic least squares solution that minimizes its loss function: *residual sum of squares* $RSS = \Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$. The exact solution is $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$, which conveniently finds the optimal coefficients. However, this is often not used in practice because of the computational complexity required to invert a matrix with many features. On average over 100 simulations, the time to compute $\hat{\beta}$ increases as the number of features increase in the design matrix $\mathbf{X}$.



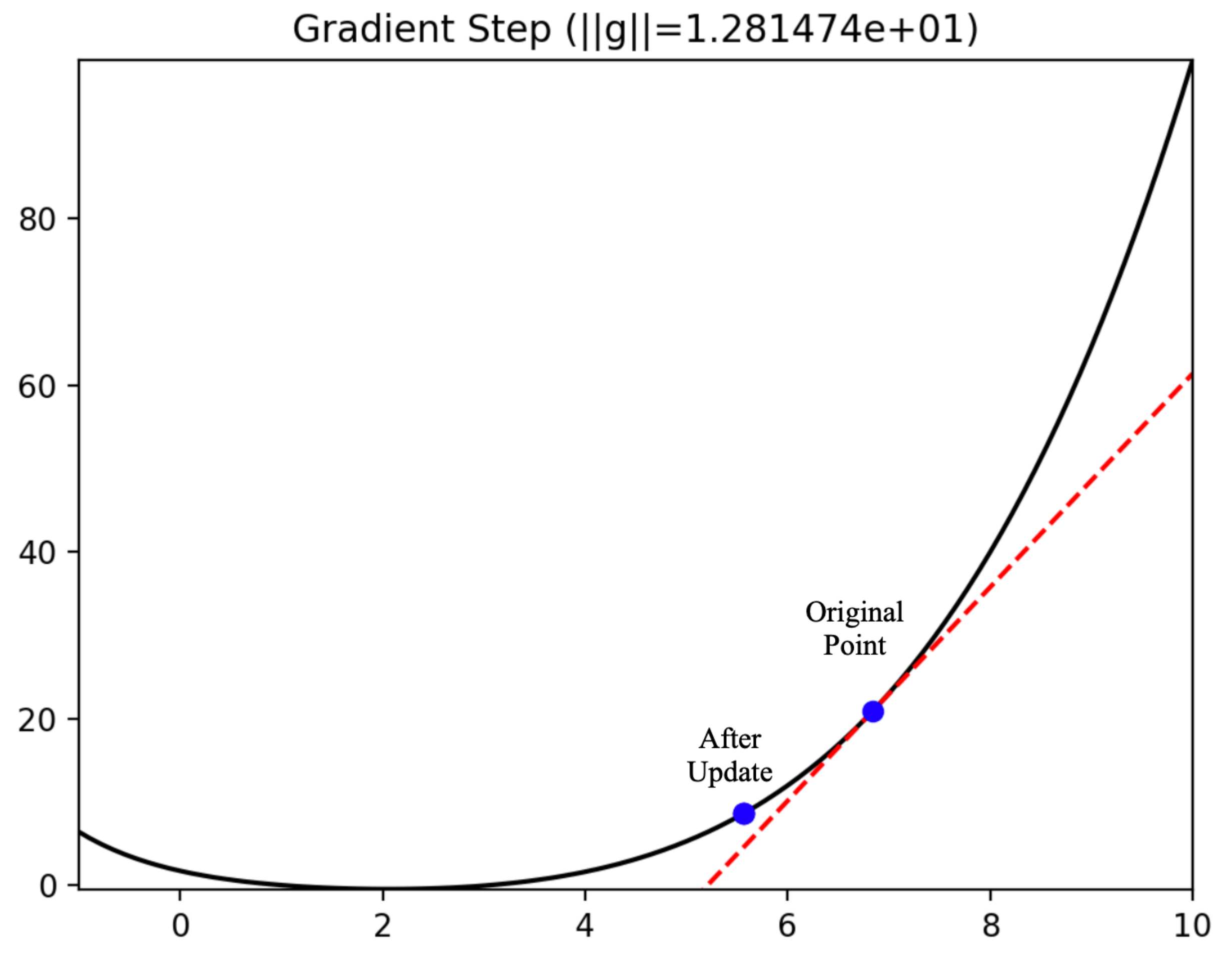
Another possibility is to use an iterative approach to find the optimal parameters that minimize the RSS metric. Applying an update rule over and over to gradually converge to the minimum of this loss function is known as an iterative optimization algorithm**.** The general scheme of these iterative algorithms is to update the vector of unknown parameters by subtracting it by a scaled gradient. The algorithm could stop early if there is very little difference between the current step and the previous step.

Gradient Descent

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \nabla f(\mathbf{x})$$

Gradient descent iteratively updates the vector of unknown parameters, $\mathbf{x}$. At each step, $\mathbf{x}_k$ is updated by subtracting the gradient of the loss function, $\nabla f(\mathbf{x})$. The gradient is a vector of first-order partial derivatives that defines the slope of the loss function. The goal is to descend down the slope of the gradient to find values of $\mathbf{x}$ that minimizes loss. The learning rate or step size, η_k, scales the gradient and determines how fast to descend along the gradient. In the context of linear regression, gradient descent would start by randomly selecting model coefficients β_0 (the intercept) and β_i (the weights) and then take steps in the direction of the error function's minimum, eventually converging on a solution. The computational complexity of this solution can be greatly reduced by using stochastic gradient descent to randomly subset the data into mini-batches to reduce the total number of calculations.

Although stochastic gradient descent is a widely used iterative optimization algorithm, it can be inefficient and does not work for every loss function. The crux of this problem is that gradients (first order derivatives in the univariate setting) are effective at pointing the algorithm to the minimum in the direction opposite the gradient, but ineffective when deciding a step size between iterations of the algorithm. Because of this, gradient descent can take many more iterations than necessary to find a solution.

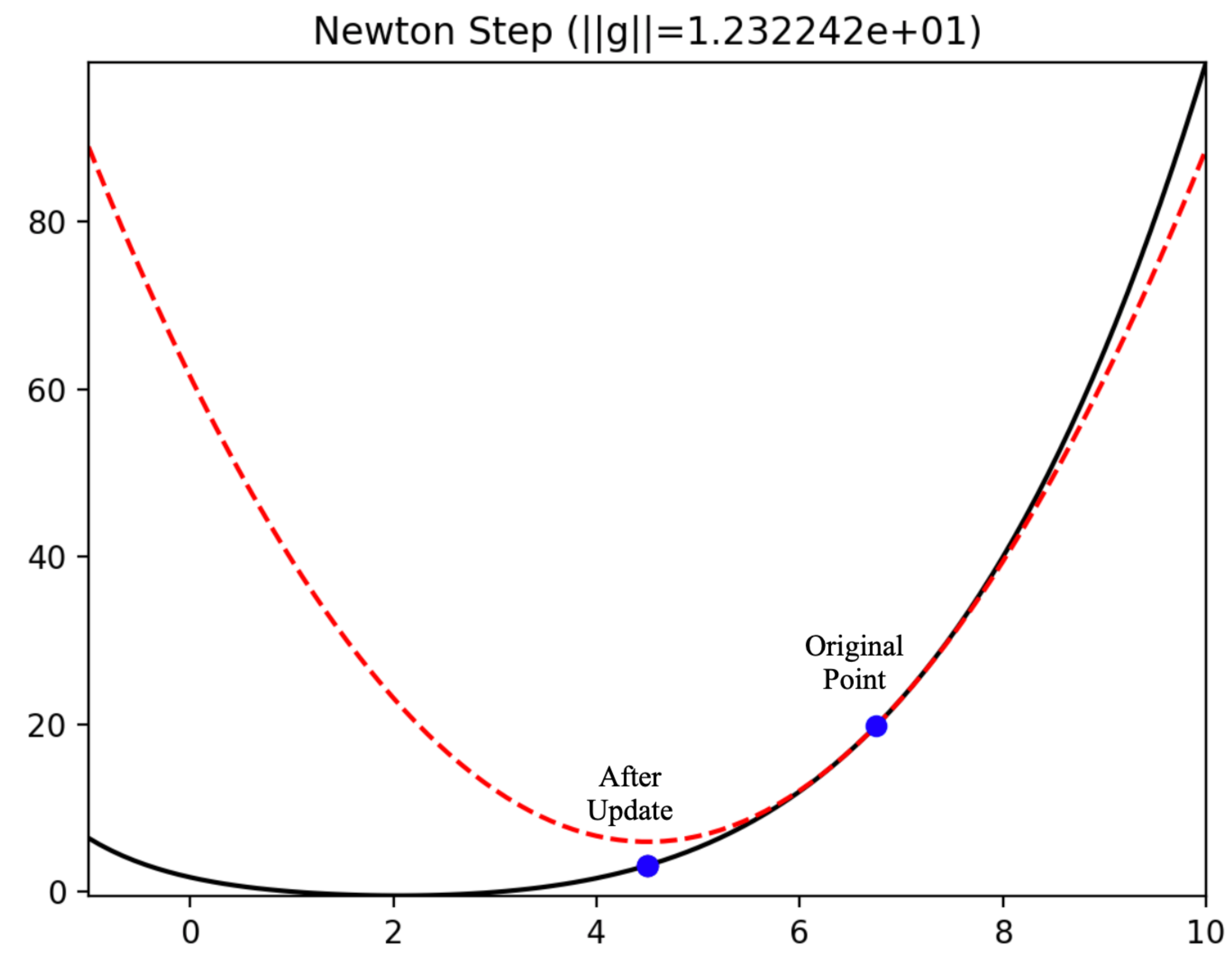


Source: *Gradient Descent (and Beyond)*.

Newton's Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x})^{-1}\nabla f(\mathbf{x})$$

Newton's method is a second order optimization algorithm. This approach introduces second order partial derivatives stored in a Hessian matrix. A vector of unknown parameters is iteratively updated with the previous step being scaled by the dot product of the gradient and the inverse of a Hessian Matrix. The use of the inverse Hessian matrix allows for quadratic approximations around minimization iterations, thus requiring only function evaluations. At each point on the loss function, a quadratic function is approximated using Taylor series, and the solution of the minimizer of this function is used as the next iterator. This non-arbitrary choice of step size allows the Newton method to converge much more quickly to the minima of the loss function. This method does not scale well to large $\mathbf{n}$ because of the computational complexity needed to compute and invert the Hessian matrix.



Source: *Gradient Descent (and Beyond)*.

3. State-of-the-art (SOTA)

3.1 SOTA Approach description:

Quasi-Newton Method

$$\Delta \mathbf{x} = \underbrace{-\mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)}_{\text{Newton's Method}} = \underbrace{-\mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)}_{\text{Quasi Newton Method}}$$

In Newton’s method, computing the inverse Hessian is a computationally costly operation. Despite descending more directly to the minimum, Newton’s method is much slower to compute per iteration. Quasi Newton methods find a middle ground between direct descent and computation complexity. The quasi-Newton method avoids computing the Hessian and its inverse to allow it to scale to much larger data problems. Instead, it utilizes an approximation to the Hessian matrix. The Hessian approximation matrix, $\mathbf{B}$, is initialized as a diagonal matrix, $\beta\mathbf{I}$. Most quasi Newton methods directly estimate $\mathbf{B}^{-1}$ to avoid computational costs associated with matrix inversion. At the end of each iteration, $\mathbf{B}_k^{-1}$ is updated to satisfy the secant equation:

$\mathbf{B}_{k+1}^{-1}\Delta\mathbf{x} = \nabla f(\mathbf{x}_{k+1}) - \ \nabla f(\mathbf{x}_k)$.

This system of equations is underdetermined and has either no solution or infinite solutions, thus additional constraints require $\mathbf{B}_{k+1}$ to be symmetric, “close” to $\mathbf{B}_k$, and positive-definite. Methods to solve $\mathbf{B}_{k+1}$ include: BFGS, Broyden, and SR1.

3.2 Why SOTA?

Modern quasi Newton methods aim to 1) decrease iterations to convergence compared to first-order methods, 2) reduce the computational complexity per iteration, and 3) reduce memory demands. The first aim is solved with Newton’s methods and the second is solved using quasi Newton methods, such as BFGS. The last aim related to memory is highlighted as the number of model parameters, $\mathbf{x}$, increases. Modern deep neural networks often have millions of parameters to optimize and storing an $n \times n$ Hessian approximation matrix is intractable in such cases. For example, if n is one-million, the $\mathbf{B}$ matrix will require 8 TB of memory assuming float64. To solve this, methods such as limited-memory BFGS (Liu and Nocedal, 1989), K-FAC (Martens and Grosse, 2015), and K-BFGS (Goldfrab et al., 2020) have been proposed.

---

4. Experimental setup:

To implement iterative optimization algorithms, we will use Python and numerical computing libraries like Numpy, SciPy, and PyTorch which provide methods for minimizing objective functions. Specifically, *scipy.optimize.minimize()* includes implementations of state-of-the-art quasi-Newton methods and PyTorch includes implementations of gradient descent. Each method will be evaluated under controlled conditions (in the case of simulation) in their ability to converge to the true minima of loss functions given an increasing number of parameters (1-1000). We will evaluate performance across these different datasets using total convergence step number, runtime, and accuracy of convergence. Using estimators from Scikit-learn, we will test optimization performance using small and large datasets, and evaluate total step size and runtime. After confirming our expected outcomes with the simulations, we will benchmark the efficiency of these algorithms under both simulated loss functions and data included with Scikit-learn, with examples in regression and classification. The dataset with the smallest number of features available in Scikit-learn that we will investigate is the “Linnerud dataset” (20 observations, 3 features) while the dataset with the largest number of features that we will investigate is the “Breast cancer wisconsin (diagnostic) dataset” (569 observations, 30 features). When possible, we will compare against closed-form solutions from numerical linear algebra. Matplotlib will be our primary tool for visualizing performance on these metrics.

5. Conclusion

So far in this project, we have explored the use of iterative optimization algorithms and how they can extend methods from linear algebra to solve statistical and machine learning problems. We have planned experiments based on researching the theoretical foundations of these algorithms, given their mathematical implementations and computing costs. We expect that the size of the problem will dictate the best algorithm to use, where state-of-the-art methods will have the greatest yield when dealing with high-dimensional data and models that require estimating a large number of parameters. These types of problems are becoming increasingly important to the fields of data science and machine learning, where the size of data and number of model parameters continues to grow.

6. References

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1–3), 503–528. https://doi.org/10.1007/BF01589116

Martens, J., & Grosse, R. (2015). *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*. https://doi.org/10.48550/ARXIV.1503.05671

Ren, Y., Bahamou, A., & Goldfarb, D. (2021). *Kronecker-factored Quasi-Newton Methods for Deep Learning*. https://doi.org/10.48550/ARXIV.2102.06737

Hardt, M., & Recht, B. (2022). Patterns, predictions, and actions: A story about machine learning. Princeton University Press.

Boyd, S., & Vandenberghe, L. (2004). Convex optimization. Cambridge university press.

Gradient Descent (and Beyond). Lecture 7: Gradient descent (and beyond). (n.d.). Retrieved November 9, 2022, from https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote07.html