Accelerating Newton's Method using Krylov Subspace Methods

Anonymous Authors[1]

Abstract

Traditional applications of Newton's Method are computationally expensive due to the necessity of computing and inverting the Hessian matrix, which becomes increasinly more computationally expensive as the feature dimensionality increases. We assess various Krylov subspace methods and matrix approximation techniques to hopefully accelerate Newton's method using both convex and nonconvex loss functions.

We assess the effectiveness of various Krylov subspace methods compared to baselines such as Exact Newton's Method on the a9a dataset using the binary logistic regression loss function. We also aim to assess the effectiveness of second-order optimization methods such as approximated Newton's Method and Gradient Descent on deep neural networks. To assess this, we train a deep neural network on the CIFAR10 dataset using various second order methods and compare their effectiveness to baselines such as gradient descent. When we train the deep neural network, we don't form the exact Hessian. Rather, we approximate the Hessian based on its' largest eigenvalue and corresponding eigenvector. We found that for the binary logistic loss function, Krylov subspace methods such as Conjugate Residual and GMRES are more effective than standard Newton's Method. For deep neural networks, we found that updating the weights layer by layer with Newton's Method results in some increase in model accuracy, but not to the extent of simply running gradient descent on the network. This paper serves to highlight the strengths and weaknesses of second-order optimization methods in

0.1. Newton's Method

In Newton's Method, we have the following weight update step: $w_{t+1} = w_t - \alpha(\nabla^2\ell(w))^{-1}\nabla\ell(w)$, where $(\nabla^2\ell(w))$ represents the Hessian matrix, where each $(\nabla^2\ell(w))_{ij} = \frac{\partial^2\ell}{\partial w_i \partial w_j}$. However, taking the inverse of this Hessian matrix is an $O(d^3)$ operation, where each $w_t \in R^d$. Notice that we are not necessarily always interested in finding the exact inverse of the Hessian. We are interested in solving the linear system: $(\nabla^2\ell(w))p = \nabla(w)$ for the Newton method update p, so that we can do the following weight update $w_{t+1} = w_t - \alpha p$. We now turn to using Krylov Subspace Methods to efficiently solve this linear system.

0.2. Krylov Subspace Methods

When solving the linear system $Ax = b$ where $A \in R^{n\times n}$ and $b \in R^n$, when n is large, we often use iterative methods to solve for the solution x^*. Standard iterative methods for solving linear systems often start with an initial guess x_0 and then keep generating a sequence of iterates x_i such that each x_i minimizes the residual $b - Ax_i$.

A Krylov Subspace is defined in the following manner: $K_m(A, b) = \text{span } \{b, Ab, Ab^2, \ldots, A^{m-1}b\}$, where $m \leq n$. Krylov methods apply the matrix A repeatedly to the iterate x_i to generate iterates $x_{i+1} \in \text{span } \{b, Ab, Ab^2, \ldots, A^{m-1}b\}$ such that the residual $b - Ax_{i+1}$ is minimized. The Krylov Subspace methods that we will test in this project include the GMRES (Generalized Minimal Residual Method) and the Conjugate Residual method. We utilize these methods primarily due to their effectiveness on all kinds of matrices A and due to their theoretical runtime being faster than $O(d^3)$, where $A \in R^{d\times d}, b \in R^d$.

0.2.1. GMRES

The Generalized Minimal Residual Method works well for linear systems $Ax = b$, where the matrix A is not necessarily symmetric. Even though we do expect the hessian $\nabla^2\ell(w)$ to be symmetric due to our $\ell(w)$ being twice differentiable, as $\ell(w)$ is the binary logistic regression loss function for the convex case, we wanted a suitable method in the event that any floating point errors on our machines caused $\nabla^2\ell(w) \neq \nabla^2\ell(w)^T$. Below, we present the GMRES al-

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Algorithm 1 GMRES Algorithm

```
  **Input:** $A$, $b$, intial iterate $x_0$, initial residual $r_0 = b - Ax_0$, $v_1 = r_0/||r_0||$, number of iterations $m$
  **for** $j = 1$ **to** $m$ **do**
    $h_{i,j} =< Av_j, v_i >$, for $i = 1, \ldots, j$.
    $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
    $h_{j+1,j} = ||\hat{v}_{j+1}||_2$
    $v_{j+1} = \frac{\hat{v}_{j+1}}{h_{j+1,j}}$
  **end for**
  $y_m = min_y ||\beta e_1 - \overline{H}_m y||_2$, get required matrices with Arnoldi Iteration
  $x_m = x_0 + V_m y_m$
**output** $x_m$
```

Algorithm 2 Generalized Conjugate Residual Algorithm

```
**Input:** $A$, $b$, initial iterate $x_0$
$r_0 \leftarrow b - Ax_0$ Initialize residual
$p_0 \leftarrow r_0$ Initialize search direction
**for** $j = 0, 1, 2, \ldots$ until convergence **do**
  $\alpha_j \leftarrow \frac{(r_j, Ap_j)}{(Ap_j, Ap_j)}$ Compute step length
  $x_{j+1} \leftarrow x_j + \alpha_j p_j$ Update solution
  $r_{j+1} \leftarrow r_j - \alpha_j Ap_j$ Update residual
  **for** $i = 0, 1, \ldots, j$ **do**
    $\beta_{ij} \leftarrow \frac{(Ar_{j+1}, Ap_i)}{(Ap_i, Ap_i)}$ Orthogonalization coefficients
  **end for**
  $p_{j+1} \leftarrow r_{j+1} - \sum_{i=0}^{j} \beta_{ij} p_i$ Update search direction
  $Ap_{j+1} \leftarrow Ar_{j+1} - \sum_{i=0}^{j} \beta_{ij} Ap_i$ Update $Ap_{j+1}$ using previous vectors
**end for**
**Output:** $x_{j+1}$ Approximate solution
```

gorithm originally presented by Youcef Saad and Martin H. Schultz in "GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems". We can see that the time complexity of m iterations of the GMRES algorithm is $O(mn^2)$, where $O(mn^2) << O(n^3)$ for $n >> m$. We present the following algorithm presented by ($Youcef\ Saad$, 1986)

0.2.2. Conjugate Residual

Above, we also present the pseudocode for the Generalized Conjugate Residual (GCR) algorithm originally introduced by Eisenstat, Elman and Schultz in 1983 (S.C. Eisenstat, 1983).

The Generalized Conjugate Residual (GCR) algorithm is an iterative method for solving non-symmetric linear systems $Ax = b$, where A is a general non-symmetric matrix. It constructs an orthogonal basis $p_0, p_1, \ldots, p_m$ for the Krylov subspace $\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$, where $r_0 = b - Ax_0$ is the initial residual.

At each iteration j, the algorithm computes the next basis vector p_{j+1} as a linear combination of the current residual r_{j+1} and the previous basis vectors $p_0, \ldots, p_j$, enforcing the orthogonality condition $(Ap_i, Ap_j) = 0$ for $i \neq j$. Both the basis vectors p_i and the products Ap_i are stored, doubling the storage requirements compared to GMRES. However, GCR has the advantage of a simpler implementation. A restarted version GCR(m) can be defined to limit the maximum subspace dimension.

0.3. Experiment Plan

In our experiments, we assess various Krylov methods in both convex and nonconvex scenarios.

0.3.1. Convex Case

For our first set of experiments, we assess the GMRES and Conjugate Residual methods compared to Newton's Method and Gradient Descent on the a9a dataset, which involves classifying whether a person's annual income exceeds 50000 based on certain features. Our loss function $\ell(w) = \frac{1}{n}\sum_{i=1}^{n} \log\ (1 + exp(-w^T x_i y_i)) + \frac{\lambda}{2}||w||^2$. We used $\lambda = 0.01$ in our experiments.

We used Gradient Descent as an initial baseline in order to find a minimum of $\ell(w)$. We used a constant step size α, where α was the largest eigenvalue of $\nabla^2 l(w)$ at $w = 0$. We find that the $\nabla^2 l(w)$ at $w = 0$ is $\frac{1}{4}XX^T + \lambda I$, where $X \in R^{n \times d}$, where n is the number of training examples and d is the dimension of each training example.

Let λ_1 be the maximal eigenvalue of $\frac{1}{4}XX^T + \lambda I$. We set $\alpha = \frac{1}{\lambda_1}$, and we find the minimal loss up to a threshold of $\delta = 10^{-16}$. We randomly choose an initial vector w_0 and run Newton's Method, GMRES, and Conjugate Residual with this initialization w_0 for 10 epochs. For calculating the step sizes for each of these algorithms, we use a line search that follows the Armijo conditions that is shown in the figure above. We measure the accuracies, losses, and suboptimalities of all these methods. When we measure suboptimality, we use the loss calculated by the initial gradient descent run as the benchmark. We also measure the wall clock time taken for all of these algorithms for 10 epochs.

0.3.2. Nonconvex Case

For the nonconvex case, we train a neural network on the CIFAR10 dataset using Newton's Method using the cross entropy loss function. The neural network will be of the following architecture: two 2D Convolutional Layers, one 2D MaxPool Layer, two Dropout Layers, and two Linear

Layers. We applying Newton's Method layerwise to the neural network. Due to the computational intensity, we don't apply Newton's Method to all layers of the neural network. Rather, we apply Newton's Method on one layer at a time, and observe which layers' training had the greatest effect on the training accuracy of the model.

To do the Newton's Method Update, we use the pyhessian library to find the eigenvector v corresponding to the largest eigenvector of each layer's Hessian matrix. Then, our hessian approximation is $\nabla^2 \ell(w) = \lambda_{max} v v^T + \tau I$, where τI tries to make the Hessian positive definite, and λ_{max} denotes the largest eigenvalue of the $\nabla^2 \ell(w)$. By using the `get_params_grad` method in the pyhessian library, we obtain the gradient $\nabla f(w)$. We set a constant step size of $\alpha = 0.01$, and perform the weight update. Due to the size of the neural network and our computational constraints, we only run Newton's Method for 4 epochs on each layer.

1. Previous Works

1.0.1. Enhancing Optimization Convergence with Nesterov's Acceleration and Approximate Newton Methods

There is considerable research interest in improving the convergence performance of optimization methods, particularly in scenarios where it is challenging to approximate the Hessian matrix well and efficiently. One of the key works in this area is the paper titled "Nesterov's Acceleration For Approximate Newton" (Haishan Ye, 2017). In this paper, the authors explore the use of Nesterov's acceleration technique to enhance the convergence of a class of second-order optimization methods known as approximate Newton methods. The authors of the paper argue that while stochastic second-order methods have gained attention due to their low computational cost per iteration, these algorithms may perform poorly when faced with difficulties in approximating the Hessian matrix. They propose the use of Nesterov's acceleration technique as an effective approach to addressing this challenge. Theoretical analysis presented in the paper demonstrates that Nesterov's acceleration technique can indeed improve the convergence performance of approximate Newton methods, akin to its impact on first-order methods. As a result, the authors propose an accelerated regularized sub-sampled Newton algorithm. Experimental results in the paper show that the accelerated algorithm outperforms the original regularized sub-sampled Newton method, thus validating the theoretical findings. Moreover, the accelerated regularized sub-sampled Newton method demonstrates performance comparable to or even better than classical algorithms.

1.0.2. Accelerating Optimization with Sparse Newton and Krylov Methods

Several recent works have also explored techniques to accelerate optimization algorithms by leveraging second-order information and Newton-type methods. One area where this has shown promise is in computing optimal transport distances between distributions, which has many applications in machine learning. Specifically, (Xun Tang†, 2024) proposed the Sinkhorn-Newton-Sparse (SNS) algorithm, which aims to accelerate the classical Sinkhorn algorithm for approximating optimal transport distances. The Sinkhorn algorithm uses iterative matrix scaling to maximize a concave potential, but can suffer from slow convergence in practice. SNS extends Sinkhorn by adding an early stopping criteria for the matrix scaling steps, followed by a Newton-type subroutine to achieve super-exponential convergence rates. The key insight is that the Hessian matrix of the potential function is approximately sparse, allowing an $O(n^2)$ per-iteration complexity similar to Sinkhorn. Through analyzing the approximate sparsity of the Hessian, the authors derive rigorous bounds on the convergence rate of SNS. Their empirical results demonstrate that SNS can converge orders of magnitude faster than the original Sinkhorn algorithm across various optimal transport problems like computing Wasserstein distances between distributions. Our work shares similarities in that we also aim to accelerate a classical optimization algorithm (Newton's method) by leveraging Krylov subspace techniques to efficiently approximate and invert the Hessian matrix. While SNS uses sparse Newton iterations specifically for optimal transport, our approach explores Krylov methods as a general acceleration strategy for Newton-type optimization across different loss functions and high-dimensional problems like training deep neural networks.

1.0.3. Dynanewton method

Another notable work that explores accelerating Newton's method is the Dynanewton algorithm proposed by ETH Zurich Computer Science group (Hadi Daneshmand, 2016).

In their work, [authors] identify two key properties specific to empirical risk minimization problems that can be leveraged to accelerate Newton's method: subsampling training data and increasing strong convexity through regularization. They propose a novel continuation method that defines a family of objectives over increasing sample sizes and decreasing regularization strength. The solutions along this path are tracked such that the minimizer of the previous objective is guaranteed to be within the quadratic convergence region of the next objective to be optimized. This approach ensures that every Newton iteration achieves super-linear contractions with respect to the chosen objective, which becomes a moving target.

The Dynanewton algorithm provides a theoretical analysis that motivates the proposed method and characterizes its speed of convergence. Experimental results on a wide range of datasets and problems consistently confirm the computational savings achieved by the algorithm.

While the Dynanewton algorithm focuses on accelerating Newton's method for empirical risk minimization problems, our project aims to assess the effectiveness of various Krylov subspace methods in accelerating Newton's method for high-dimensional optimization tasks, including both convex and nonconvex loss functions. Krylov subspace methods have been explored as alternatives to directly computing and inverting the Hessian matrix, which becomes computationally expensive as the feature dimensionality increases.

2. Results

2.1. Convex Case

We first begin by presenting the suboptimality plots for Newton's Method, GMRES, and Conjugate Residual.

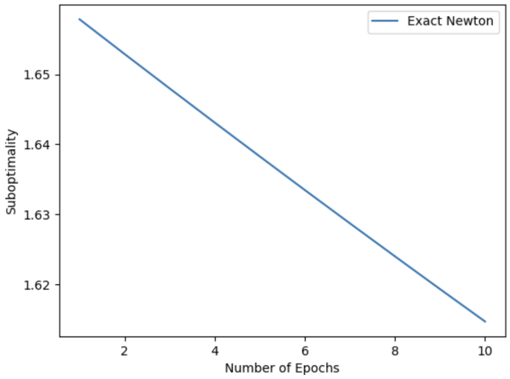

Figure 1. Exact Newton's Method Suboptimality

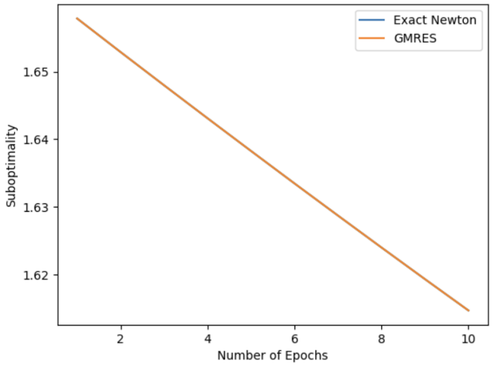

Figure 2. GMRES Suboptimality

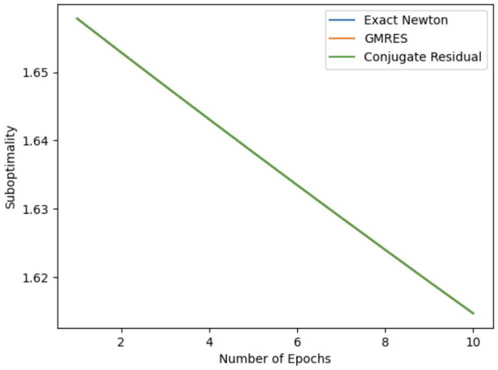

Figure 3. Conjugate Residual Suboptimality

The suboptimality plots displayed in Figure 1, 2, and 3 illustrate the performance of the Conjugate Residual, Exact Newton, and GMRES methods in terms of their suboptimality values across different numbers of epochs. As observed from these plots, the suboptimality values for all three methods exhibit a similar decreasing trend as the number of epochs increases, indicating that they converge to the optimal solution at a comparable rate.

Upon closer inspection of the plots, we can observe that the Exact Newton method (shown in blue) has a slightly steeper descent in suboptimality compared to the GMRES (orange) and Conjugate Residual (green) methods. This suggests that while all three methods converge at a similar rate, the Exact Newton method may achieve a lower suboptimality value for a given number of epochs, potentially leading to a more accurate solution.

As mentioned earlier, the wall clock times for the GMRES and Conjugate Residual methods are lower than the Exact Newton method for 10 epochs. This implies that although the Exact Newton method may converge to a slightly lower suboptimality value, the GMRES and Conjugate Residual methods are more computationally efficient, particularly when a larger number of epochs is required.

The trade-off between convergence rate, accuracy, and computational efficiency is also worth to consider. While the Exact Newton method may provide a more accurate solution, the GMRES and Conjugate Residual methods offer a better balance between accuracy and computational efficiency, making them more suitable for scenarios where computational resources are limited or when the problem size is large.

2.2. Nonconvex Case

For the nonconvex case, we train Newton's Method on a deep neural network consisting of the following four main layers: 2 2D convolutional layers and 2 linear Layers. We had some other layers in the neural network that we didn't run Newton's Method on. These other layers were mentioned in the Experiment Plan section of this report.

To train the deep neural network with Newton's method, we updated the weights, layer by layer. Due to the computational cost needed to run Newton's Method on this network, we only updated the weights of the first, second, and fourth layers of the neural network (so both 2D Convolutional Layers and the last linear layer). In Figures 4, 5, 6, 7, we present our training accuracies for running Newton's Method on the various layers as well as running gradient descent.

We can notice that gradient descent was far more effective at improving the model's training accuracy over the course of 4 epochs. However, training Newton's Method only the first layer did result in an approximately 15% accuracy over 4 epochs, and training only on the fourth layer resulted in an approximately 14% accuracy over 4 epochs. Interestingly, training only one layer with Newton's Method resulted in a greater training accuracy than training multiple layers with Newton's Method. Also, the wall clock time for training these three layers with Newton's Method was about 2 hours and 47 minutes, wheras the wall clock times for the other experiments were roughly 20 minutes.

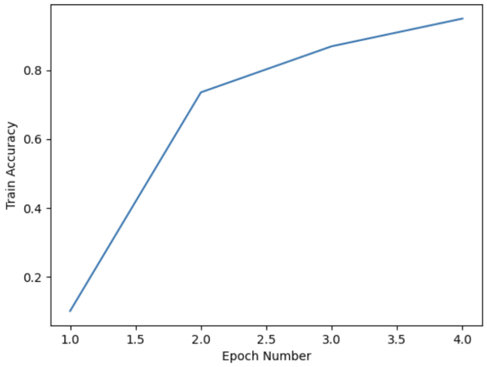

Figure 4. Train Accuracy with Gradient Descent

3. Conclusions

We assess the effectiveness of Newton's Method and approximated Newton's Method in various settings. In the convex setting, we ran binary class logistic regression on the a9a dataset, and demonstrated that while Exact Newton, GMRES, and Conjugate Residual acheive similar

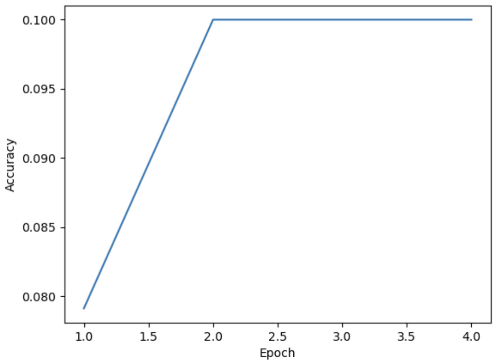

Figure 5. Newton's Method All But Third Layer

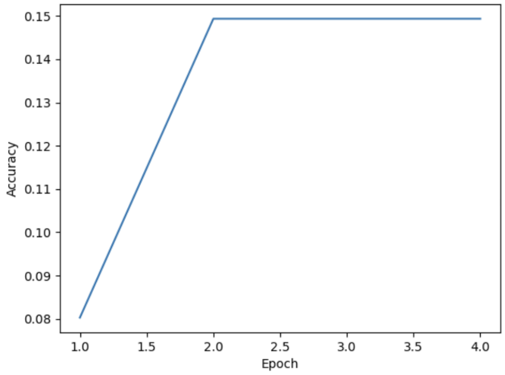

Figure 6. Newton's Method Only First Layer

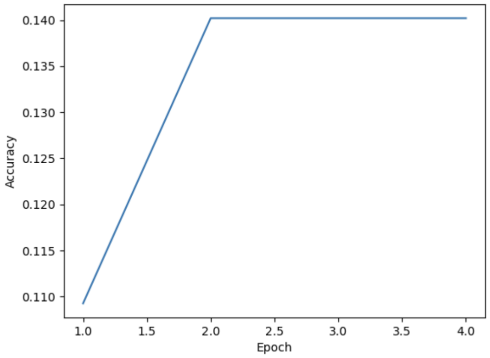

Figure 7. Newton's Method Only Fourth Layer

training accuracies, the wall clock time for the GMRES and Conjugate Residual methods are lower over 10 epochs compared to Exact Newton due to not explicitly inverting the Hessian. We feel that it is reasonable that over a larger number of epochs, the difference in the time needed to run GMRES and Conjugate Residual compared to Exact Newton's Method will be greater.

When training a deep neural network on CIFAR10, we observed that gradient descent is much less costly and results in higher training accuracy than even using Newton's Method by sparsely forming the Hessian using its' greatest eigenvalue and corresponding eigenvector. However, we did note that running Newton's Method with a sparsely formed Hessian did lead to an approximately 150% improvement in the accuracy compared to the untrained model when done with one layer at a time instead of across multiple layers in the model.

Second-order methods seem to be more optimal when dealing with convex settings than nonconvex settings both with regards to hardware constraints such as wall-clock times and accuracies.

Software and Data

Our code for this project is located at the following Github repository: https://github.com/AnantShyam/CS6787FinalProject

Acknowledgements

We would like to thank Professor Chris De Sa, Cathy, and TC for a wonderful semester in CS 6787 (Advanced Machine Learning Systems) as well as for all of their help throughout this project. We were fortunate to have multiple insightful discussions about our project with all of them, and we learned a lot throughout this entire semester!

References

Hadi Daneshmand, Aurelien Lucchi, T. H. Dynanewton accelerating newton's method for machine learning. *arxiv*, 2016.

Haishan Ye, Z. Z. Nesterov's acceleration for approximate newton. *JMLR*, 2017.

S.C. Eisenstat, H.C. Elman, M. S. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM Journal of Numerical Analysis*, 1983.

Xun Tang†, Michael Shavlovsky, H. R. E. T. K. K. T. T. X. L. Y. Accelerating sinkhorn algorithm with sparse newton iterations. *ICLR*, 2024.

Youcef Saad, M. H. S. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 1986.