Homework 1
CS 544 - Optimization in Computer Vision

Christian Howard
howard28@illinois.edu

Patrick Zhai
haoyuz5@illinois.edu

Rishika Agarwal
rishika2@illinois.edu

Yuxuan Zhou
yuxuanz6@illinois.edu

1 Introduction

Optimization is a crucial part to modern day computing. Numerical Optimization is not limited to familiar domains of statistical modeling and Machine Learning, but also finds itself being used in everything from Design Optimization in engineering to Optimal Control and Guidance systems on missiles, solving nonlinear equations modeling physical systems, and plenty more. Investigating the characteristics of different optimization techniques is crucial for ensuring we apply them to appropriate problems. Within this report we investigate two techniques, namely:

1. Quasi-Newton using Conjugate Gradient
2. Nonlinear Conjugate Gradient using Polak-Ribiere update

We will start the analysis by considering a toy problem that will allow us to readily investigate numerical properties of the two techniques in a variety of contexts. Following this, we will see how the two methods contrast each other across some case study problems within the context of classification in Machine Learning. Note that the optimization codes and corresponding objective functions were implemented by members of the group, so analysis completed was done with those instead of heavily optimized optimization routines in popular libraries.

2 Analysis using Toy Problem

2.1 Intro

When considering the two optimization techniques given in 1 and 2, both techniques clearly make their estimates for the optimum of a problem using different sets of information. The Quasi-Newton method benefits with explicit curvature

information while the Polak-Ribiere flavor of Nonlinear Conjugate Gradient attempts to get some curvature estimates using smart conjugate directions and then perform line searches to advance as far as possible along proposed descent directions. While Quasi-Newton gets more information, it also requires a heftier computational cost than the Polak-Ribiere method to obtain the desired search direction.

This trade-off is ultimately where one can see situations where one method might prove superior to the other. To gain some further insight, we will investigate the below toy problem.

$$\begin{aligned} \min_{\boldsymbol{x}\mathbb{R}^d} \quad & g_\alpha(\boldsymbol{x}) = (1-\alpha)f_1(x) + \alpha f_2(x) \qquad (1) \\ \text{where} \quad & f_1(\boldsymbol{x}) = \sum_{k=1}^{d} (x_k)^2 \\ & f_2(\boldsymbol{x}) = \sum_{k=1}^{d} (x_k)^4 \end{aligned}$$

We will vary this toy problem for different values of problem dimension d, as well as vary $\alpha \in [0, 1]$ to perform a convex combination of the two convex functions f_1 and f_2. This toy problem gives us analysis benefits with a known global optimal solution at $\boldsymbol{x}^* = \mathbf{0}$ as well as makes it simple to investigate how the dimensionality of the problem and the variation in second derivative magnitudes as a function of state impacts convergence.

2.2 Analysis

Within this analysis, we investigate differences in performance between Full Quasi-Newton, Sparse Quasi-Newton, and the Polak-Ribiere executed against the toy problem defined in (1). The Full Quasi-Newton is the Conjugate Gradient based Quasi-Newton method such that it uses the full, dense Hessian matrix. The Sparse Quasi-Newton method is similar but instead uses a sparse representation and matvec (matrix-vector product) form of the Hessian for use in the Conjugate Gradient algorithm.

We perform our analysis using values $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ and $d \in d_{low} \cup d_{high}$, where $d_{low} = \{10, 50, 100\}$ is the set of low dimension test values, and $d_{high} = \{1000, 2000, 3000\}$ is the set of high dimension test values. The initial condition is set to $\boldsymbol{x}_0 = \mathbf{2}$ so that we can start off with reasonably large second derivative values as α gets large. Further, convergence for this problem is defined as when $\|\nabla_x g_\alpha(\boldsymbol{x})\|_\infty \leq \tau$ for $\tau = 10^{-6}$. Using this criteria, Figures 1 through 6 represent the number of iterations and run-time results for different values of α and d for each of the three methods.

Looking at Figures 1 through 6, there are some clear trends between the various methods. Consistently, the run-time performance across both low and high dimensional problems shows that Sparse Quasi-Newton converges faster than

Polak-Ribiere, who also converges faster than Full Quasi-Newton. In the low dimensional cases, the time differences are small between the various methods but for high dimensional problems, Sparse Quasi-Newton is an order of magnitude faster than Polak-Ribiere, which is also an order of magnitude faster than the Full Quasi-Newton.

Both Quasi-Newton approaches converge in few iterations relative to the Polak-Ribiere method, but the Full Quasi-Newton method does more work per iteration than the Polak-Ribiere, which ultimately explains why the Full Quasi-Newton method is slower than the Polak-Ribiere technique. Since the Sparse Quasi-Newton method performs fast matvec operations within the Conjugate Gradient solver, its run-time is significantly smaller than the Full Quasi-Newton counterpart and that, coupled with the few iterations, explains why it is possible for this technique to show superior performance to the Polak-Ribiere technique.

Some other interesting things to note is the performance affects of changing (α, d). For the Quasi-Newton methods, there is a clear increase in run-time and number of iterations as the second derivative magnitudes, and their variation, increase. This trend holds for all d, generally speaking. For the Polak-Ribiere method, we see some generally increasing run-time and number of iterations as the second derivative magnitudes increase. However, for $d = 100$ we see some peculiar behavior where $\alpha = 0.25$ took the longest in both run-time and number of iterations. Reasons for this are not clear but suspicion lies with the choices of approximate line search technique.

Another interesting observation is that for the Quasi-Newton methods, the number of iterations they took was identical for all values of d, given some α. This observation seems to be due to the explicit use of curvature information and how this information transcends the number of dimensions for this toy problem. Polak-Ribiere, on the other hand, displayed variation in the number of iterations it requires for different values of (α, d), which was interesting. This showed us that the conjugate directions coupled with the line search really influence the convergence in a dimension and curvature dependent manner.

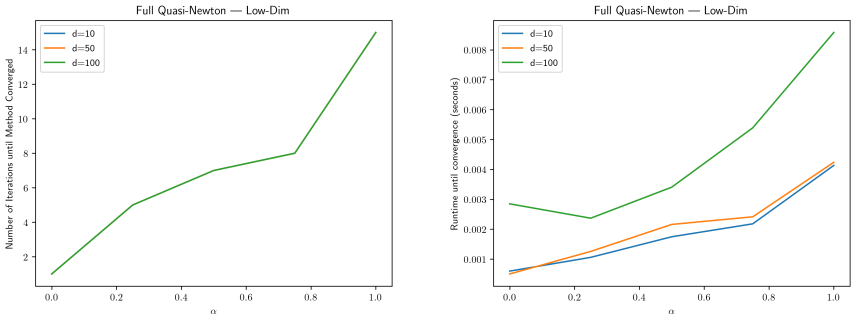

Figure 1: (α, d_{low})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Full Quasi-Newton*

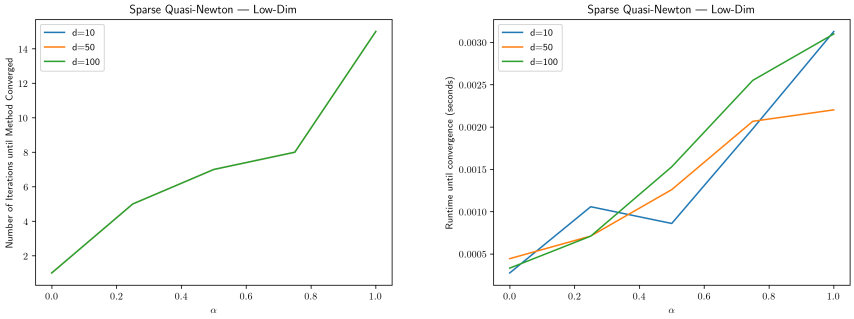

Figure 2: (α, d_{low})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Sparse Quasi-Newton*

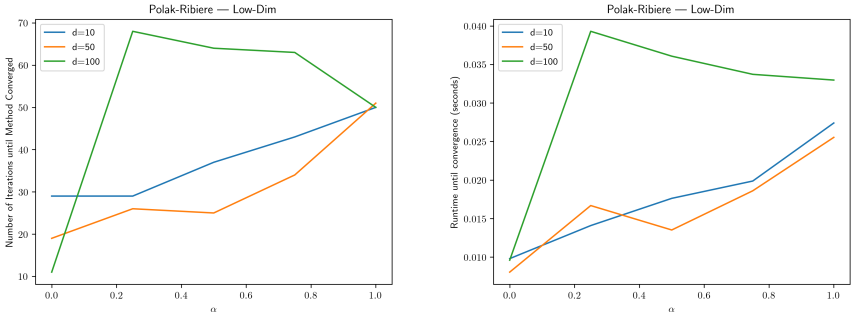

Figure 3: (α, d_{low})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Polak-Ribiere*

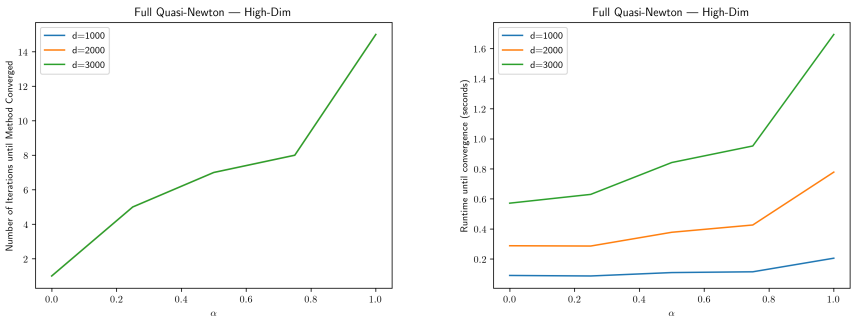

Figure 4: (α, d_{high})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Full Quasi-Newton*

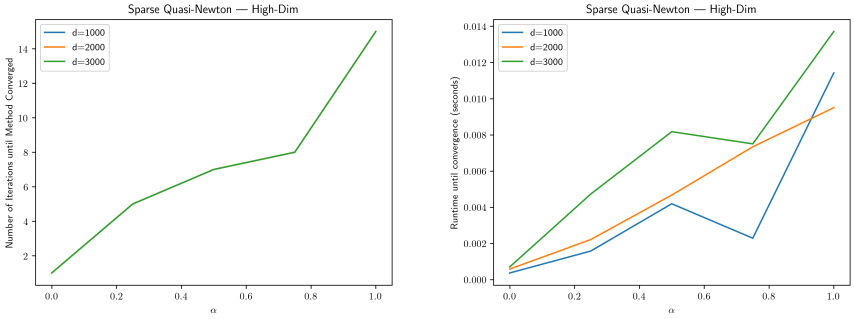

Figure 5: (α, d_{high})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Sparse Quasi-Newton*

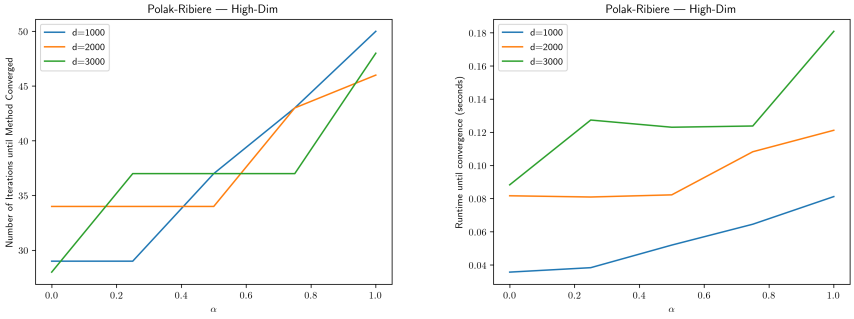

Figure 6: (α, d_{high})-study: Number of iterations until convergence (left) and Run-time until convergence (right) using *Polak-Ribiere*

3 Case Studies

3.1 Classification with Logistic Regression

To study the effect of using the two optimization methods for a practical problem, we chose to work with both the MNIST and CIFAR-10 datasets but restrict ourselves to building 1-vs-all classifiers for each dataset. We built a simple regularized logistic regression model based on the principle of classifying images into 2 classes. The probability for x_i belonging to k^{th} class is given by (2). The predicted class label is then the class with the maximum probability.

$$\begin{aligned} p(y_i = k) &= \sigma\left(\beta_0^{(k)} + \sum_j \beta_j^{(k)} x_j\right) \\ &= \frac{1}{1 + e^{-(\beta_0^{(k)} + \sum_j \beta_j^{(k)} x_j)}} \end{aligned} \tag{2}$$

where $\sigma(\cdot)$ is the Sigmoid function, $\beta^{(k)}$ is the parameter vector for class k, and $\beta_j^{(k)}$ is the j^{th} element of the vector $\beta^{(k)}$. Since we are only working with binary classification, it becomes simple enough to just compute the parameters for $k = 1$ and to use the identity $p(y_i = 0) = 1 - p(y_i = 1)$ to see what the probability is that a digit is *not* 1. Using this and a negative log-likelihood Maximum Likelihood Estimate approach with some l_2 regularization, we obtain the objective function

$$J(\boldsymbol{\beta}) = \frac{-1}{|D|} \sum_{i=1}^{|D|} y_i \log\left(\sigma(\boldsymbol{\beta}^T \boldsymbol{x}_i)\right) + (1 - y_i) \log\left(1 - \sigma(\boldsymbol{\beta}^T \boldsymbol{x}_i)\right) + \frac{\lambda}{2|D|} \sum_{i=1}^{d} \beta_i \tag{3}$$

where D is our dataset, $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ is our parameter vector being estimated, and $\lambda > 0$ is our regularization parameter. In the MNIST dataset, the input images are of size 20×20, so $d = 400$ resulting in our optimization taking place in $\mathbb{R}^{401}$. In the CIFAR-10 dataset, each image is 32×32 pixels with 3 color channels, so $d = 3 \times 1024 = 3072$ and in turn makes our optimization take place in $\mathbb{R}^{3073}$. The regularization hyper-parameter λ is kept fixed, so it is not counted as one of the parameters of the model.

The summary of the analysis can be found in Figures 7 and 8 as well as Table

Method	Iterations	Final Cost	Run-time (s)	Test Accuracy (%)
Polak-Ribiere	211	0.0188	11.465	99
Quasi-Newton	13	0.01362	9.907	99

Method	Iterations	Final Cost	Run-time (s)	Test Accuracy (%)
Polak-Ribiere	10,000	0.1722	21,243	91
Quasi-Newton	65	$2.334 \cdot 10^{-4}$	3151	85

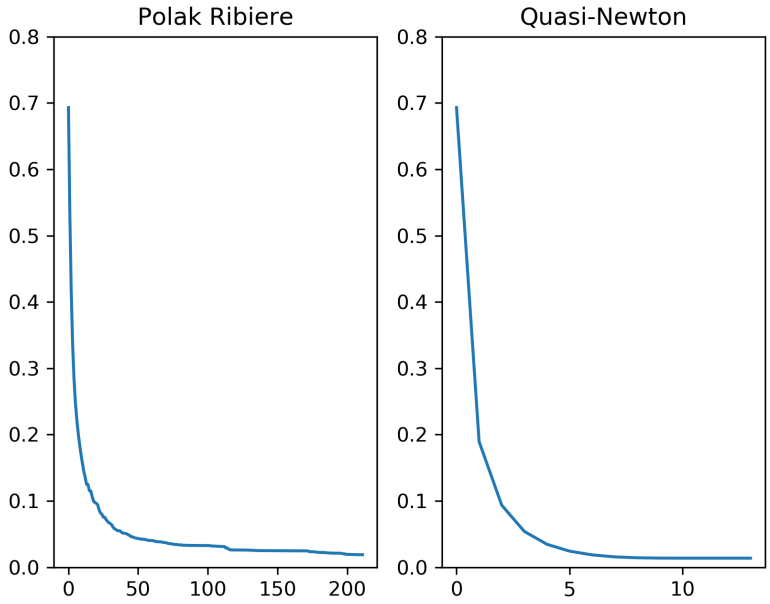

Figure 7: MNIST Objective Function Error vs Iteration Count

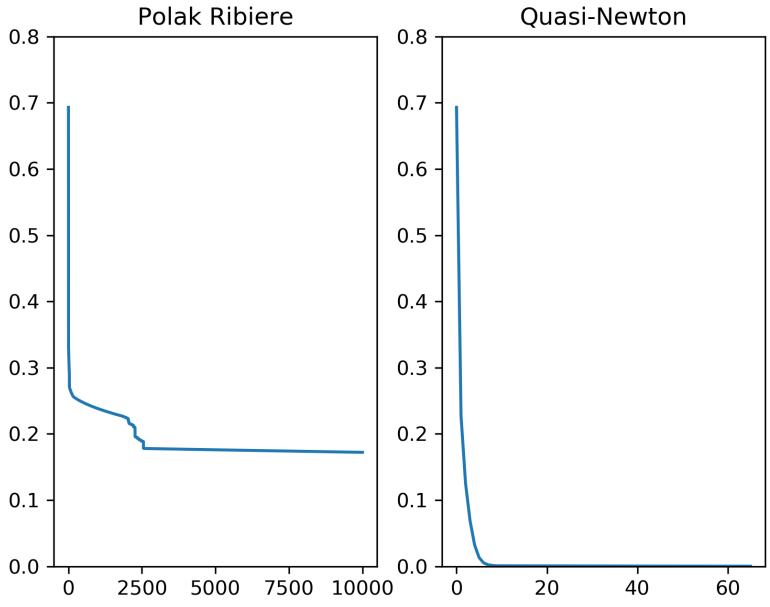

Figure 8: CIFAR-10 Objective Function Error vs Iteration Count