Conjugate Gradient Methods

Dr. Amiri

دانشگاه صنعتی شریف

Electrical Engineering department

Mazdak Teymorian	401101495
Kiarash Joolaie	400100949
Kooshan Fattah	401102191

Project

January 27, 2025

Conjugate Gradient Methods

Project

Mazdak Teymorian	401101495
Kiarash Joolaie	400100949
Kooshan Fattah	401102191

A Review of the Conjugate Gradient Methods and Their Usage in Hessian-Free Optimization

Abstract

It's well known that for optimizing the objectives that exhibit a pathological curvature (meaning that the objective changes direction abruptly, making it difficult for algorithms to predict the optimal direction of movement or that the minimum lies within a very narrow and elongated region, making it challenging to converge efficiently) the gradient descent method is not suitable. For these type of objectives, we use 2^{nd}-order optimization that model the curvature and correct for it. The reason that methods like gradient descent are unsuitable for such problems is that these methods are curvature-blind and thus it is impossible for them to successfully navigate. One of best methods to use is the Hessian-Free Optimization which, as its main backbone and advantage, uses the conjugate gradient method and its variants.

Introduction

The main point of this paper is studying the advantages and disadvantages of the Hessian-Free optimization method which is much faster than the Newton method, which is the canonical 2^{nd}-order optimization algorithm. We first review the Newton's method and why it is not a good choice for large models due to the quadratic relation between the size of the Hessian and the model parameters,and then we introduce the Hessian-Free method and the conjugate gradient methods affiliated with it.

Newton's method

The main idea behind the Newton's method is that f can be approximated to the 2^{nd} degree around the point θ by the quadratic equation :

$$f(+\alpha \cdot p) \approx q_\theta(p) = f(\theta) + \nabla f(\theta)^T p + \frac{1}{2} p^T B p$$

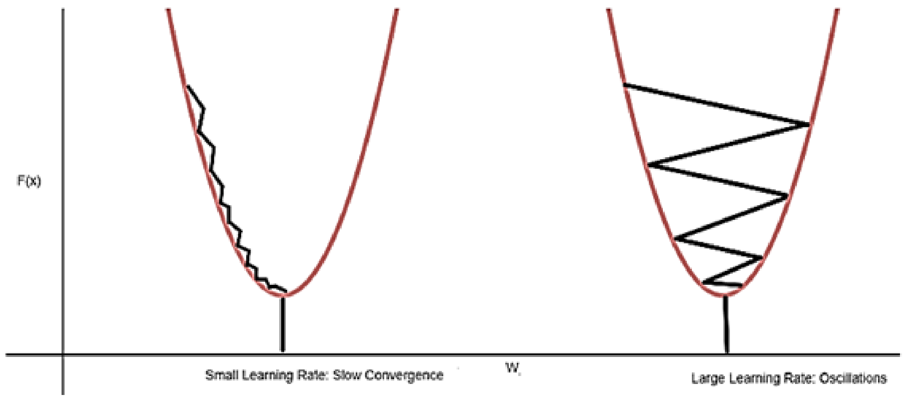
where $B = H(\theta)$ is the hessian of f at θ. Also p denotes the search direction. **Here we encounter the first disadvantage** , where the Hessian matrix may be indefinite so that this quadratic may not have a global minimum. Also for large p , this approximation may not be a good approximation of f. It's common to **damp** the Hessian and to use $B' = H + \lambda I$ instead of the original Hessian matrix.
Imagine you're hiking down a mountain. The "curvature" represents how steeply the slope changes.

- Gentle Slope (Low Curvature): If the slope changes gradually, you can confidently take long strides in that direction, even if the initial descent isn't very steep.

- Steep Slope (High Curvature): If the slope changes dramatically, it's wiser to take shorter steps to avoid overshooting and potentially going uphill.

Newton's method cleverly calculates the "ideal" step size by considering both the steepness of the descent and the rate of change in the slope. It essentially determines how far to go in a particular direction before the slope starts to incline again.

- Bouncing: Taking overly large steps in areas of high curvature can cause the algorithm to oscillate back and forth, like a ball bouncing down a bumpy slope. This is often addressed by slowing down the learning rate (taking smaller steps overall).
- Slow Exploration: If the most promising descent directions have low curvature, the algorithm may proceed very slowly, as it cautiously takes small steps. This can make the optimization process extremely slow, even if the algorithm is not truly stuck at a local minimum


F(x)
Small Learning Rate: Slow Convergence
W
Large Learning Rate: Oscillations

truncated-Newton

In the traditional Newton's method, optimizing $q_\theta(p)$ involves computing the $N \times N$ matrix B and solving the system $Bp = -\nabla f(\theta)$. This becomes computationally prohibitive for large N, a scenario often encountered even in moderately sized neural networks. The HF method addresses this by utilizing two key ideas. The first is that for an N-dimensional vector d, the product Hd can be efficiently approximated using finite differences. This approximation requires only one additional gradient evaluation, as expressed by the identity:

$$Hd = \lim_{\epsilon \to 0} \frac{\nabla f(\theta + \epsilon d) - \nabla f(\theta)}{\epsilon}$$

The second idea involves the linear conjugate gradient algorithm (CG), which is particularly effective for optimizing quadratic objectives such as $q_\theta(p)$. This algorithm only requires matrix-vector products with B. While CG may require up to N iterations to converge in the worst case, it often makes significant progress in minimizing $q_\theta(p)$ within a much smaller number of iterations. The fundamental structure of the HF method is detailed in Algorithm 1.

Algorithm 1 HF Optimization Method

```
1: for $n = 1, 2, \ldots$ do
2:     $g_n \leftarrow \nabla f(\theta_n)$                          ▷ Compute the gradient
3:     Compute/Adjust $\lambda$ by some method                       ▷ Adjust the damping parameter
4:     $\mathbf{B_n}(\mathbf{d}) = \mathbf{H}(\theta_n)d + \lambda d$   ▷ Define the modified Hessian-vector product
5:     $p_n \leftarrow$ CG-Minimize($\mathbf{B_n}, -g_n$)             ▷ Minimize using Conjugate Gradient
6:     $\theta_{n+1} \leftarrow \theta_n + p_n$                     ▷ Update the parameters
7: end for
```

An appealing feature of the HF approach is the efficiency of the CG method. Unlike the non-linear CG method (NCG) frequently used in machine learning, linear CG takes advantage of the quadratic nature of the optimization problem to iteratively generate a set of "conjugate directions" d_i. These directions satisfy $d_i^\top A d_j = 0$ for $i \neq j$, enabling independent and precise optimization along each direction. Specifically, the step size along each direction is given by the reduction divided by the curvature, i.e., $-\nabla f^\top d_i / d_i^\top A d_i$, a result that follows from the conjugacy property. In contrast, the non-linear CG method, which operates directly on f rather than q_θ, tends to lose the conjugacy of directions quickly, and the line search is often performed inexactly and at a higher computational expense.
Now we discus the main point of this report ; the conjugate gradient method.

Conjugate gradient

The main outline of the linear CG method is shown below :

Algorithm 2 Linear Conjugate Gradient Algorithm

```
1: Result: Estimate an approximate value of the minimizer $x^*$ and determine $f(x^*)$
2: Define $f(x)$;                                   ▷ Define the objective function
3: Define the initial iterate $x$, the symmetric positive definite matrix $A$, the vector $b$, and the
   positive tolerance $\epsilon$;
4: $r \leftarrow Ax - b$;                           ▷ Initialize the residual
5: $\delta \leftarrow -r$;                          ▷ Initialize the descent direction
6: while TRUE do
7:     if $\|r\| \leq \epsilon$ then
8:         Break;
9:     end if
10:    Return $x^* \leftarrow x$;                   ▷ Return the minimizer
11:    Return $f(x^*)$;                             ▷ Return the function value at the minimizer
12:    Calculate $\beta \leftarrow -\frac{r^T\delta}{\delta^T A\delta}$;
13:    Calculate $x \leftarrow x + \beta\delta$;    ▷ Generate the new iterate
14:    Calculate $r \leftarrow Ax - b$;             ▷ Generate the new residual
15:    Calculate $\chi \leftarrow \frac{r^T A\delta}{\delta^T A\delta}$;
16:    Calculate $\delta \leftarrow \chi\delta - r$;  ▷ Generate the new descent direction
17: end while
```

Now we delve into the details and see how the parameters are defined. The Conjugate Gradient (CG) method is an iterative algorithm used to solve systems of linear equations of the form $Ax = b$, where A is a symmetric positive definite (SPD) matrix. The method minimizes the quadratic function:

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

where x is the vector of unknowns, b is a known vector, and c is a constant. The gradient of this function is:

$$\nabla f(x) = Ax - b$$

The goal of the CG method is to find the minimizer x^* of $f(x)$, which corresponds to the solution of the linear system $Ax = b$.

▬ *Step 1: Initialization*

The algorithm begins with an initial guess x_0 for the solution. The initial residual r_0 is computed as:

$$r_0 = b - Ax_0$$

The residual r_0 represents the error in the initial guess. The initial search direction p_0 is set equal to the residual:

$$p_0 = r_0$$

▬ *Step 2: Iterative Update*

At each iteration k, the algorithm performs the following steps:

1. **Compute the Step Length α_k:**

   The step length α_k is chosen to minimize $f(x)$ along the search direction p_k. This is achieved by setting the derivative of $f(x_k + \alpha_k p_k)$ with respect to α_k to zero:

   $$\frac{d}{d\alpha_k} f(x_k + \alpha_k p_k) = 0$$

   Substituting $f(x)$ into the equation and simplifying, we obtain:

   $$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

2. **Update the Solution Estimate x_{k+1}:**

   The new estimate of the solution is computed by moving in the direction p_k with step length α_k:

   $$x_{k+1} = x_k + \alpha_k p_k$$

3. **Update the Residual r_{k+1}:**

   The residual is updated to reflect the new solution estimate:

   $$r_{k+1} = r_k - \alpha_k A p_k$$

4. **Compute the New Search Direction p_{k+1}:**

   The new search direction p_{k+1} is computed using the residual r_{k+1} and the previous search direction p_k. The parameter β_k is chosen to ensure that p_{k+1} is conjugate to all previous search directions:

   $$p_{k+1} = r_{k+1} + \beta_k p_k$$

   The parameter β_k is computed as:

   $$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

Step 3: Termination

The algorithm terminates when the residual r_k is sufficiently small, indicating that the current estimate x_k is close to the true solution x^*. The convergence criterion is typically based on the norm of the residual:

$$\|r_k\| \leq \epsilon$$

where ϵ is a small tolerance.

Mathematical Justification

The CG method is derived from the idea of minimizing the quadratic function $f(x)$ by iteratively minimizing it along conjugate directions. Two vectors p_i and p_j are said to be conjugate with respect to A if:

$$p_i^T A p_j = 0 \quad \text{for} \quad i \neq j$$

This property ensures that the minimization along each direction p_i does not interfere with the minimization along previous directions, leading to efficient convergence.

The step length α_k is chosen to minimize $f(x)$ along the direction p_k. This is equivalent to solving the one-dimensional optimization problem:

$$\min_{\alpha_k} f(x_k + \alpha_k p_k)$$

Substituting $f(x)$ into the equation and setting the derivative to zero, we obtain:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

The parameter β_k is chosen to ensure that the new search direction p_{k+1} is conjugate to all previous search directions. This is achieved by setting:

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

Practical Considerations

In practice, the CG method is often preconditioned to improve convergence. Preconditioning involves transforming the original system $Ax = b$ into an equivalent system $M^{-1}Ax = M^{-1}b$, where M is a matrix that approximates A and is easy to invert. Common preconditioners include diagonal (Jacobi) preconditioning, incomplete Cholesky factorization, and multigrid methods.

implementation of the linear conjugate gradient method

We consider a symmetric positive definite (SPD) matrix A and a vector b. The goal is to solve the linear system $Ax = b$ using both the Conjugate Gradient method and the Steepest Descent method. We compare the number of iterations required for each method to converge to a solution within a specified tolerance.

Problem Setup

Let A be a 100×100 SPD matrix, and b be a vector of size 100. We generate A and b as follows:

$$A = 2I - \text{tridiag}(-1, 2, -1), \quad b = \mathbf{1}$$

The initial guess x_0 is a vector of zeros. The tolerance for convergence is set to $\epsilon = 10^{-6}$.

Implementation of Conjugate Gradient Method

The Conjugate Gradient method is implemented per algorithm 2 , which was shown earlier.

Implementation of Steepest Descent Method

The Steepest Descent method is implemented as follows:

Algorithm 3 Steepest Descent Method

```
1: Initialize $x = x_0$, $r = b - Ax$
2: for $i = 1$ to max_iter do
3:     Compute $\alpha = (r^T r)/(r^T A r)$
4:     Update $x = x + \alpha r$
5:     Update $r = r - \alpha A r$
6:     if $\|r\| < \epsilon$ then
7:         Break
8:     end if
9: end for
```

Results and Comparison

After running both methods, we observe the following results:

- **Conjugate Gradient Method**: Converges in approximately 50 steps.
- **Steepest Descent Method**: Converges in approximately 500 steps.

This demonstrates that the Conjugate Gradient method is significantly more efficient than the Steepest Descent method for this problem. The CG method takes advantage of the conjugate directions to converge much faster, especially for large-scale problems.

Explanation of Results

The Conjugate Gradient method converges faster because it uses conjugate directions to ensure that each step is optimal and does not interfere with previous steps. In contrast, the Steepest Descent method follows the direction of the gradient at each step, which can lead to slow convergence, especially for ill-conditioned matrices.

The number of steps required for convergence is a key metric for comparing the efficiency of iterative methods. In this example, the CG method requires fewer steps to achieve the same level of accuracy, making it a more powerful tool for solving large linear systems.

Nonlinear Conjugate Gradient Method : Fletcher-Reeves Algorithm

The Fletcher-Reeves algorithm is a nonlinear conjugate gradient method used to solve unconstrained optimization problems. It extends the linear conjugate gradient method to general nonlinear functions by iteratively generating search directions that are approximately conjugate.

Key Idea

The algorithm minimizes a nonlinear function $f(x)$ by generating a sequence of search directions d_k using the Fletcher-Reeves parameter:

$$d_k = -\nabla f(x_k) + \beta_k d_{k-1}$$

where β_k is given by:

$$\beta_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}$$

This ensures that the search directions remain approximately conjugate.

Advantages

- Memory-efficient: Only requires gradient information and the previous search direction.
- Scalable: Suitable for large-scale optimization problems.
- Global convergence: Converges to a local minimum for smooth functions with an accurate line search.

Limitations

- Sensitive to line search accuracy.
- Performance degrades for highly nonlinear functions.

It's implementation is as follows :

Algorithm 4 Fletcher-Reeves Algorithm

```
1: Result: Estimate an approximate value of the minimizer $x^*$ and determine $f(x^*)$
2: Define $f(x)$;                                        ▷ Define the objective function
3: Define $\nabla f(x)$;                     ▷ Define the gradient of the objective function
4: Initialize the starting experimental point $x_j$, and the tolerance $\epsilon$;
5: $\delta_j \leftarrow -\nabla f(x_j)$;           ▷ Initialize the Fletcher-Reeves descent direction
6: while TRUE do
7:     Perform the one-dimensional minimization task to find the step length $\beta_j$ at the $j^{th}$ step;
8:     $x \leftarrow x_j + \beta_j \delta_j$;                 ▷ Generate a new updated experimental point
9:     if $\|\nabla f(x)\| < \epsilon$ then                                ▷ Termination condition
10:        Break;
11:    end if
12:    Return $x^* \leftarrow x$;                                       ▷ Return the minimizer
13:    Return $f(x^*)$;                      ▷ Return the function value at the minimizer
14:
15:    $x_j \leftarrow x$;                                          ▷ Generate the new iterate
16:    $\chi_j \leftarrow \frac{\|\nabla f(x_j)\|^2}{\|\nabla f(x_{j-1})\|^2}$;
17:    $\delta_j \leftarrow -\nabla f(x_j) + \chi_j \delta_j$;             ▷ Generate the new descent direction
18:
19: end while
```

The implementation of the algorithms are done in the jupyter notebook.