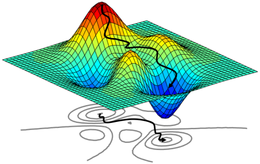


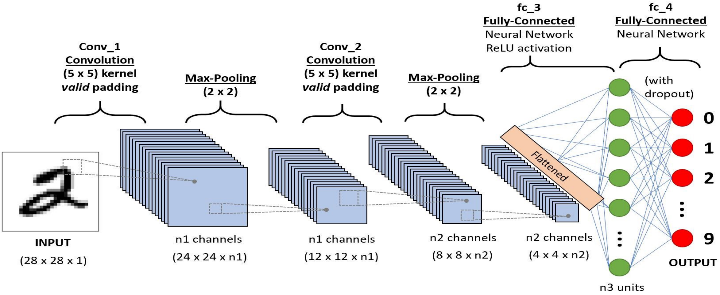
Conjugate Gradient in Large-Scale Optimization

Ronald Nap, Cristian Espinosa

April 30, 2024

Motivation

- **Universal Challenge:** Optimization seeks to find the best solution to a problem given some constraints.
- **Broad Applications:** Applications range from machine learning to data mining and image processing.
- **Challenge of Scale:** As datasets grow exponentially, traditional optimization methods struggle to keep up. This drives the need for innovative algorithms that can efficiently handle massive scales of data and high-dimensional spaces without compromising on performance.



Problem Description

Objective

The objective is to evaluate the performance of various optimization algorithms taught in class on large-scale problems, comparing their effectiveness to the Conjugate Gradient (CG) method, which is introduced as a suitable algorithm for large-dimensional data.

Quadratic Function

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

- $A \in \mathbb{R}^{5000\times 5000}$ is a symmetric positive-definite matrix.
- $b \in \mathbb{R}^{5000}$ is a vector.
- c is a scalar.

Rosenbrock Function

A non-convex function given by $g(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$

- We test with $n = 1000$ to test optimization performance.

Method

Conjugate Gradient Method

- Efficiently optimizes quadratic forms, especially when A is large and sparse.
- **Requirements for Convergence:**
  - The matrix A must be symmetric positive definite.

Pseudocode

1. Initialize x_0, compute $r_0 = b - Ax_0$, set $p_0 = r_0$
2. For $k = 0, 1, 2, \ldots$ until convergence:
   - $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
   - $x_{k+1} = x_k + \alpha_k p_k$
   - $r_{k+1} = r_k - \alpha_k A p_k$
   - $p_{k+1} = r_{k+1} + \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} p_k$

Conjugate Gradient Method: Pros and Cons

```python
def conjugate_gradient(A, b, x0, max_iter=50, tol=1e-6):
    x = x0  # Initial guess
    r = b - torch.mv(A, x)  # Initial residual
    p = r  # Initial search direction
    rs_old = torch.dot(r, r) # Square of residual norm

    for iter in range(max_iter):
        Ap = torch.mv(A, p)  # Calculate Ap

        alpha = rs_old / torch.dot(p, Ap) # Step size

        x += alpha * p  # Update the solution

        r -= alpha * Ap  # Update the residual

        # Compute the new residual norm squared
        rs_new = torch.dot(r, r)

        # Check criterion
        if torch.sqrt(rs_new) < tol:
            break

        # Update the search direction
        p = r + (rs_new / rs_old) * p

        # Set up for the next iteration
        rs_old = rs_new
    return x, iter
```

Pros

- Efficiently handles large, sparse matrices
- Only requires matrix-vector products
- No need for matrix storage
- Potentially faster than other methods for certain problems

Cons

- Convergence can be slow for ill-conditioned systems
- Performance is heavily dependent on preconditioning
- Not as robust as more modern algorithms in certain contexts

Preconditioning

Preconditioning and Condition Number

Preconditioning (1) is a technique used to transform a given problem into a form that is more amenable to numerical methods. It involves modifying the system of equations:

$$Ax = b \quad \text{to} \quad M^{-1}Ax = M^{-1}b \qquad (1) \qquad\qquad \kappa(A) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \qquad (2)$$

where M is the preconditioner. The objective is to select a preconditioner M so that $M^{-1}A$ possesses a lower condition number compared to A. The condition number given by (2) quantifies the numerical instability when solving $Ax = b$.

- **Solving $Ax = b$:** The CG method efficiently solves the equation $Ax = b$, where A is a symmetric positive definite. This method iteratively approaches the solution by minimizing the residual, which depends crucially on the characteristics of A.
- **Digits of Accuracy:** The number of digits of accuracy achievable is given by:

$$\text{Digits of Accuracy} \approx -\log_{10}(\kappa(A)) \qquad (3)$$

This formula implies that a higher condition number results in fewer digits of accuracy in the solution, underscoring the need for a low condition number.

Review of Optimization Methods

Method	Pros	Cons	Convergence
Steepest Descent	-Requires only function and gradient evaluations	Slow convergence	Linear
Newton's Method	-Quadratic convergence -Handles non-linear problems	Requires second-order derivatives; computation heavy for large systems	Quadratic
Trust Region	-Robust; -Handles non-linear problems; -Does not rely on line search	-More parameters to tune; -Requires Hessian or its approximation	Superlinear
BFGS	-Avoids Hessian computation using secant updates; -Good practical performance	-High memory needed; -Convergence not guaranteed for non-convex functions	Superlinear

Numerical Results

Function	Method	Iterations	Time (s)	Rel. Error
Quadratic	Conjugate Gradient	3	**0.0088**	5.81×10^{-7}
	Preconditioned CG	2	1.4007	3.12×10^{-9}
	Steepest Descent	50	7.9327	6.71×10^{-7}
	Newton's Method	2	1.1769	$\mathbf{3.59 \times 10^{-15}}$
	BFGS	5	87.0257	1.71×10^{-7}
	Trust Region	50	139.0355	7.32×10^{-7}
	Trust Region + CG	50	0.7868	1.44×10^{-7}
Rosenbrock	Conjugate Gradient	2	**0.0046**	0.999
	Preconditioned CG	2	0.2504	0.999
	Steepest Descent	50	19.0641	0.916
	Newton's Method	50	44.8113	0.657
	BFGS	50	29.2694	0.979
	Trust Region	50	11.8806	**0.575**
	Trust Region + CG	50	4.4343	0.726

Discussion of Numerical Results

Key Findings

- **Efficiency:** Conjugate Gradient stands out in both functions for significantly reducing the number of iterations and computational time.
- **Accuracy:** Newton's Method achieves the lowest relative error for the quadratic function and obtained the secondest lowest relative error on the rosenbrock function demonstrating high accuracy despite higher computational time.
- **Trade-offs:** BFGS and Steepest Descent show mixed results, Trust Region demonstrates efficiency and relatively low error, suggesting a balanced approach.

Implications

- The choice of optimization method should consider the specific demands of the problem, weighing the need for speed against accuracy.
- For rapid convergence in less complex or well-conditioned problems, Conjugate Gradient might be preferable, while Newton's Method is recommended for situations where high accuracy is critical.
- Trust Region + CG could be a versatile choice, offering a good compromise between convergence speed and accuracy.