CPSC 665 : An Algorithmist's Toolkit Lecture 1 : 23 February 2015

Gradient Descent and Conjugate Gradient

Lecturer: Sushant Sachdeva *Scribe: Rachel Lawrence*

1. Introduction

Solving a linear system approximately via iteratively improving guesses is often more feasible than solving it outright. Gradient Descent and Conjugate Gradient are iterative methods for solving systems of linear equations that have symmetric, positive definite, and sparse matrices.

In particular, suppose we wish to solve the system $Ax = b$, and we know that $A \succ 0 \in \mathbb{R}^{n \times n}$

Definition 1.1 (**PSD System**). *A **PSD System** is a linear system of equations, whose matrix representation is a PSD matrix.*

2. Solving PSD Systems

Let $f(x) = \|x^* - x\|_A^2$, the "distance" from x to the optimal solution x^*. We wish to minimize $f(x)$.

Definition 2.1 (**Inner Product**). $\langle a, b \rangle_M = a^\intercal M b$

Definition 2.2 (**Norm**). $\|a\|_M = \langle a, a \rangle_M^{\frac{1}{2}}$

Remark 2.3. *$M \succ 0 \Leftrightarrow f$ is a convex function.*

So solving linear functions in PSD matrix is equivalent to solving convex programs, and we have that:

$$\begin{aligned} f(x) &= \left\|x^* - x\right\|_A^2 \\ &= \langle x - x^*, x - x^* \rangle_A \\ &= x^\intercal A x - 2x^\intercal b + \underbrace{x^{*\intercal} A x^*}_{\text{constant}} \end{aligned}$$

So $x^* = A^{-1}b$ and $\nabla(f(x^*)) = 0$

This leads us to a basic idea for minimizing convex functions: "if you're at point x on a differentiable function, you should move in a direction to minimize the function". Gradient descent is a method for solving linear systems approximately, based on the concept of moving a small amount in the opposite direction from the gradient at every step.

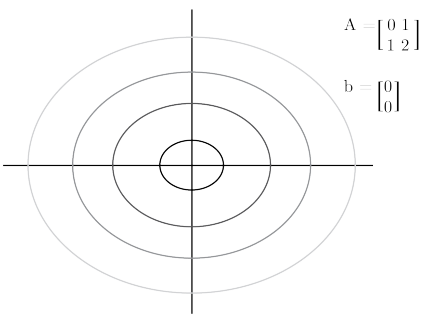

Figure 1. Solving PSD Systems

It is easy to see that if f were not convex, gradient descent could get "stuck" in a local minimum – but if it is convex, every local optimum is a global optimum.

3. Gradient Descent

The algorithm for computing the iterative approximations is as follows:

3.1. **Gradient Descent Algorithm.** First, guess $x_0 = 0$. At every subsequent step, update x_t by moving "against the gradient" by some factor α_t:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

Definition 3.1 (**Residual**). *The **residual** r_t at each step t is defined as $Ax_t - b$.*

Thus we have that

$$\nabla f(x) = 2(Ax - b) = 2r_t$$

Now, it remains to determine the best value for α_t, α_t^*, at each step. To do this, note that

$$\begin{aligned} f(x_{t+1}) &= f(x_t - 2\alpha_t r_t) \\ &= \left\|x^* - x_t + 2\alpha_t r_t\right\|_A^2 \\ &= \left\|x^* - x_t\right\|_A^2 + 4\alpha_t \langle r_t, x^* - x_t\rangle_A + 4\alpha_t^2 \left\|r_t\right\|_A^2 \end{aligned}$$

And we can minimize this expression to find α_t^* by solving for α_t when $\frac{\partial}{\partial \alpha_t} = 0$:

$$\frac{\partial}{\partial \alpha_t} = 4\langle r_t, x^* - x_t\rangle_A + 8\alpha_t \left\|r_t\right\|_A^2 = 0$$

$$\alpha_t^* = -\frac{1}{2}\frac{\langle r_t, x^* - x_t\rangle_A}{\left\|r_t\right\|_A^2}$$

Now, recalling from the definition of the inner product that

$$\begin{aligned} \langle r_t, x^* - x_t\rangle &= r_t^\intercal A(x^* - x_t) \\ &= r_t^\intercal (b - Ax_t) \\ &= -r_t^\intercal r_t \end{aligned}$$

And from this we can calculate that $\alpha_t^* = \frac{r_t^\intercal r_t}{\left\|r_t\right\|_A^2}$

3.2. **Algorithm Analysis.** Ultimately, we would like to know the runtime of this algorithm. To start, we examine how the function changes after 1 step of gradient descent. Plugging back in to $f(x_{t+1})$:

$$\begin{aligned} f(x_{t+1}) &= f(x_t) + 4\alpha_t \langle r_t, x^* - x_t\rangle_A + 4\alpha_t^2 \left\|r_t\right\|_A^2 \\ &= f(x_t) - \frac{2(-r_t^\intercal r_t)^2}{\left\|r_t\right\|_A^2} + 4(\frac{1}{4})\frac{(-r_t^\intercal r_t)^2}{\left\|r_t\right\|_A^2} \\ &= f(x_t) - \frac{(r_t^\intercal r_t)^2}{r_t^\intercal A r_t} \end{aligned}$$

Now, we are able to examine how rapidly the function value decreases, cumulatively:

$$\begin{aligned} f(x_t) &= \left\|x^* - x_t\right\|_A^2 \\ &= \left\|A^{-1}(b - Ax_t)\right\|_A^2 \\ &= r_t^\intercal A^{-1} r_t \end{aligned}$$

And so

$$f(x_{t+1}) = f(x_t)\left(1 - \frac{(r_t^\intercal r_t)^2}{f(x_t)(r_t^\intercal A r_t)}\right)$$
$$= f(x_t)\left(1 - \frac{(r_t^\intercal r_t)^2}{(r_t^\intercal A^{-1} r_t)(r_t^\intercal A r_t)}\right)$$

Fact 3.2. *Where λ_{max} and λ_{min} are the largest and smallest eigenvalues of A, $\frac{r_t^\intercal A r_t}{r_t^\intercal r_t} \leq \lambda_{max}(A)$ and $\frac{r_t^\intercal A^{-1} r_t}{r_t^\intercal r_t} \leq \lambda_{max}(A^{-1}) = \frac{1}{\lambda_{min}(A)}$*

Thus we can get an upper bound for $f(x_{t+1})$ bound independent of r_t:

$$f(x_{t+1}) \leq f(x_t)\left(1 - \frac{\lambda_{min}(A)}{\lambda_{max}(A)}\right)$$

And we can see that f decreases multiplicatively after every iteration.

Definition 3.3 (**Condition Number of A**). *The **condition number** of a matrix A is defined as $\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$.*
Equivalently, for ellipsoid $\epsilon_A = \{x | x^\intercal A^{-1} x \leq 1\}$, $\kappa(A)$ measures the "skewness" of ϵ_A:

$$\kappa(A) = \left(\frac{\textit{longest axis of ellipsoid}}{\textit{smallest axis of ellipsoid}}\right)^2$$

Fact 3.4. *Using this definition, we can modify the previous inequality to say that*

$$f(x_{t+1}) \leq f(x_t)(1 - \frac{1}{\kappa(A)})$$

Applying this inequality inductively,

$$\Rightarrow f(x_t) \leq f(x_0)e^{-t/\kappa(A)}$$

And so

$$f(x_0) = \left\|x^*\right\|_A^2$$

since the initial guess was that $x_0 = 0$.

3.3. **Runtime.**

Theorem 3.5. *After $O(\kappa log(\frac{1}{\epsilon}))$ operations, where t_A is the time required to multiply by A,the error at x_t is at most $\epsilon \left\|x^*\right\|_A$*

Proof. To find the runtime of Gradient Descent, we wish to solve for t such that

$$\left\|x^* - x_t\right\|_A \leq \epsilon \left\|x^*\right\|_A$$
$$f(x_t) \leq \epsilon^2 f(0)$$

therefore, $e^{-t}\kappa(A) = \epsilon^2$ suffices, and so

$$t = 2\kappa \log \frac{1}{\epsilon}$$

This t means we need $O(\kappa \log(\frac{1}{\epsilon}))$ iterations of Gradient Descent, and each iteration requires a constant number of matrix multiplications. □

So we can see that in the worst case, the runtime of Gradient Descent is quadratic, but it can actually be quite fast when working with sparse matrices.

4. Conjugate Gradient

It turns out, we can do better than Gradient Descent by using the Conjugate Gradient method [HS52]

Definition 4.1 (**Krylov Subspace**). *The **Krylov Subspace** of order t for matrix A and vector b is the linear subspace spanned by the images of b under the first $t-1$ powers of A:*

$$Span\{b, Ab, A^2b, \cdots, A^{t-1}b\}$$

Definition 4.2 (**Conjugate Gradient Method**). *Define $x_0 = b$ and then compute*

$$x_1 = Ax_0 - b \in Span\{b, Ab\}$$
$$x_2 \in Span\{b, Ab, A^2b\}$$
$$\vdots$$

Remark 4.3. *The t^{th} vector output from Gradient Descent lies in the Krylov subspace of order $t+1$ for A and b, but need not be the best vector in that subspace.*

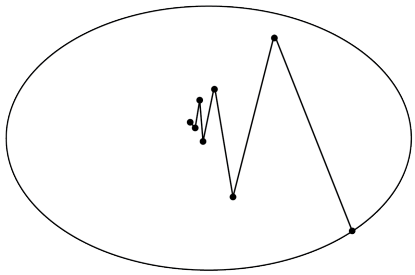

Figure 2. Gradient Descent

Theorem 4.4. *Conjugate Gradient outputs the best vector in the Krylov subspace of order t, after t iterations*

How to compute these vectors quickly?
1. Suppose $\{v_0, v_1, ..., v_t\}$ is any basis for the Krylov subspace.
2. Can we find β_i such that $\sum_i \beta_i v_i$ minimizes f?

$$f(\sum_i \beta_i v_i) = \left\| x^* - \sum_i \beta_i v_i \right\|_A^2$$
$$= \left(\sum_i \beta_i v_i \right)^\intercal A \left(\sum_i \beta_i v_i \right) - 2(Ax^*)^\intercal \left(\sum_i \beta_i v_i \right) + \text{a constant}$$

and using matrix inversion,

$$= \sum_{i,j} \beta_i \beta_j (v_i^\intercal A v_j) - 2 \sum_i \beta_i (b^\intercal v_i) + c$$

Now, as everyone knows, Conjugate Gradient works using a healthy dose of magic:

4.1. **Magic Item #1: Separability.** What if there were an orthogonality condition, so that $v_i^\intercal A v_j = 0$ for all $i \neq j$?

$\langle v_i, v_j \rangle_A = 0 \Leftrightarrow v_i, v_j$ are A-orthogonal so

$$f(\sum_i \beta_i v_i) = \sum_i \beta_i^2 (v_i^\intercal A v_i) - 2\sum_i \beta_i (b^\intercal v_i) + c$$

And so under the orthogonality assumption, the problem has been reduced to a "separable" equation, which is easily solved:

$$\beta_i^* = \frac{b^\intercal v_i}{v_i^\intercal A v_i}$$

So, we will endeavor to orthogonalize all the v_i to reduce the problem to this simpler case.

4.2. **Computing an A-orthogonal basis.** Using Gram-Schmidt Orthogonalization:
Pick $v_0 = b$; $\text{Span}\{v_0\} = \text{Span}\{b\}$
For $w_i = Av_{i-1}$,
Obtain v_i by A-orthogonalizing w_i with respect to $\{v_o, ..., v_{i-1}\}$:

$$v_i = w_i - \frac{\langle w_i, v_0 \rangle_A}{\langle v_0, v_0 \rangle_A} v_0 - \frac{\langle w_i, v_1 \rangle_A}{\langle v_1, v_1 \rangle_A} v_1 - \cdots - \frac{\langle w_i, v_{i-1} \rangle_A}{\langle v_{i-1}, v_{i-1} \rangle_A} v_{i-1}$$

Remark 4.5. *$\{v_0, v_1, ..., v_t\}$ are A-orthogonal, i.e. $v_i^\intercal A v_j = 0$ if $i \neq j$*

Lemma 4.6. *$Span\{v_0, ..., v_t\} = Span\{b, Ab, \cdots, A^t b\}$*

Proof. The argument proceeds inductively.

$$\begin{aligned}
Span\{v_0\} &= Span\{b\} \\
Span\{v_0, v_1\} &= Span\{b, Ab\} \\
&\vdots \\
Span\{v_0, \cdots, v_r\} &= Span\{v_0, \cdots, v_{r-1}, Av_{r-1}\} \\
&= Span\{b, Ab, \cdots A^{r-1}b, Av_{r-1}\} \text{ by induction} \\
&= Span\{b, Ab, \cdots, A^r b\}
\end{aligned}$$

□

4.3. **Magic Item #2: Algorithm Runtime.** We wish to know the runtime of the above procedure; however, things aren't looking good since it appears we will need to compute $O(t^2)$ inner products during Gram-Schmidt.

Theorem 4.7. *For a given j, it is only necessary to orthogonalize w_{j+1} with respect to v_j and v_{j-1}*

Proof.

$$\begin{aligned}
v_i &\in Span\{b, .., A^i b\} \\
w_{i+1} = Av_i &\in Span\{Ab.., A^{i+1} b\} \\
&\in Span\{v + 0, .., v_{i+1}\}
\end{aligned}$$

Noting that $v_j^\intercal A v_r = 0 \ \forall r < j$,

$$\begin{aligned} &v_j \text{ is A-orthogonal to } \{v_0, \cdots, v_{i+1}\} \text{ if } j > i+1 \\ &\Rightarrow (w_{i+1})^\intercal A v_j = 0 \text{ if } j > i+1 \\ &\Leftrightarrow (A v_i)^\intercal A v_j = 0 \\ &\Leftrightarrow v_i^\intercal A (A v_j) = 0 \\ &\Leftrightarrow v_i^\intercal A w_{j+1} = 0 \end{aligned}$$

And so we need to orthogonalize w_{j+1} only with respect to v_j, v_{j-1}. □

Lemma 4.8. *We can construct an A-orthogonal basis in $O(t \cdot (t_{A+n}))$ operations.*

References

[HS52] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. 1952.