Randomized Linear Algebra

Krishna Balasubramanian

University of California, Davis

STA 243: Spring 2020

Notation

For a matrix $\mathbf{A}$ a matrix of size $m \times n$,

- $\mathbf{A}_{i,:} \in \mathbb{R}^n$ denotes the i-th row
- $\mathbf{A}_{:,j} \in \mathbb{R}^m$ denotes the j-th column
- $A_{i,j}$ denote the entry at the (i, j)-th position.
- Matrix norms satisfy following properties:
  - $\|\mathbf{A}\| \geq 0$ and $\|\mathbf{A}\| = 0$ if and only if $A = 0$ (positivity)
  - $\|\alpha \mathbf{A}\| = |\alpha| \|\mathbf{A}\|$ (homogeneity)
  - $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$
- Frobenius norm: $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \mathbf{A}_{i,j}^2}$

Notation

- Induced norm: Given a vector norm $\|\cdot\|$, we can define the corresponding **induced norm** by

$$\begin{aligned}\|\mathbf{A}\| &= \sup_x \{\|\mathbf{A}x\| : \|x\| = 1\} \\ &= \sup_x \left\{ \frac{\|\mathbf{A}x\|}{\|x\|} : x \neq 0 \right\}\end{aligned}$$

- Operator norm: $\|\mathbf{A}\|_2 = \sup_{x\neq 0} \frac{\|\mathbf{A}x\|_2}{\|x\|_2} = \sigma_{max}(\mathbf{A})$ (the maximum singular value). In this notes, we use $\|\mathbf{A}\|$ to denote $\|\mathbf{A}\|_2$, the operator norm.

Notation

Fact

Let $\mathbf{A} = ab^\top$ be a rank-1 matrix. Then $\|\mathbf{A}\|_F = \|\mathbf{A}\|_2 = \|a\|_2 \|b\|_2$. Prove it.

Justification: For the operator norm, we have:

$$\begin{aligned} \|ab^\top\|_2 &= \sup_x \left\{ \|ab^\top x\|_2 : \|x\|_2 = 1 \right\} \\ &= \sup_x \left\{ |b^\top x| \|a\|_2 : \|x\|_2 = 1 \right\} \\ &= \|a\|_2 \|b\|_2 \end{aligned}$$

where the last equality follows because the direction that maximizes the inner product is the same as b.

Randomized Matrix Multiplication

Randomized Matrix Multiplication

Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$ the problem we consider is to compute or approximate $\mathbf{C} = \mathbf{A}\mathbf{B}$.

Randomized Matrix Multiplication

A naive way of multiplying two matrices is given in Algorithm 1.

Algorithm 1 Vanilla Matrix Multiplication Algorithm

```
for $i = 1, \ldots, m$ do
    for $j = 1, \ldots, n$ do
        $\mathbf{C}_{i,j} = \mathbf{A}_{i,:}\mathbf{B}_{:,j} (= \sum_{k=1}^{n} \mathbf{A}_{i,k}\mathbf{B}_{k,j})$
    end for
end for
Return $\mathbf{C}$
```

Randomized Matrix Multiplication

Exact Computation

Assuming $m = p = n$, the computational complexity of the above naive algorithm is $\mathcal{O}(n^\omega)$ with $\omega = 3$. Over the years, several researchers have worked to bring the exponent ω down 2.3728639. See Figure 1 for an overview. Such algorithms typically care about **exactly** computing the matrix $\mathbf{C}$.

Randomized Matrix Multiplication

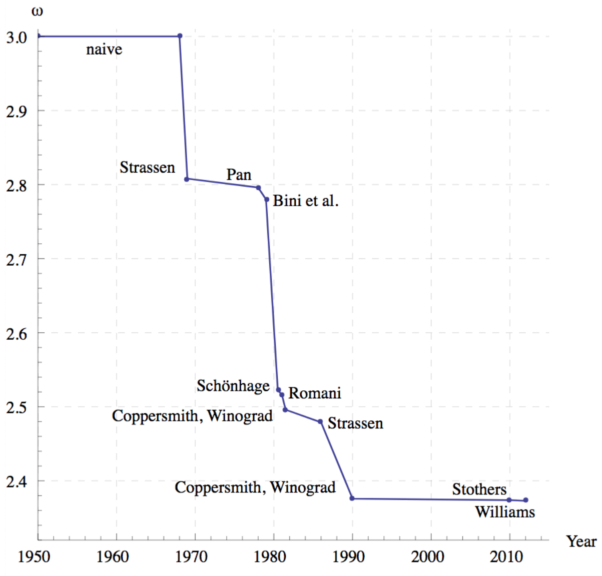

Figure 1: The bound on ω over time. Source; Wikipedia.

Randomized Matrix Multiplication

Approximate Computation

What if we are OK with **approximately** computing the matrix $\mathbf{C}$? It turns out, one can develop fast algorithms in this case. Its based on the following crucial observation: Algorithm 1 is based on inner-product operations, but another way to do matrix multiplication directly is to view it as based on outer-product operations.

Randomized Matrix Multiplication

$$\begin{aligned}
\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} &= \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} \begin{pmatrix} a & d \end{pmatrix} + \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} \begin{pmatrix} b & e \end{pmatrix} + \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} \begin{pmatrix} c & f \end{pmatrix} \\
&= \begin{pmatrix} 1a & 1d \\ 4a & 4d \\ 7a & 7d \end{pmatrix} + \begin{pmatrix} 2b & 2e \\ 5b & 5e \\ 8b & 8e \end{pmatrix} + \begin{pmatrix} 3c & 3f \\ 6c & 6f \\ 9c & 9f \end{pmatrix} \\
&= \begin{pmatrix} 1a + 2b + 3c & 1d + 2e + 3f \\ 4a + 5b + 6c & 4d + 5e + 6f \\ 7a + 8b + 9c & 7d + 8e + 9f \end{pmatrix}.
\end{aligned}$$

Figure 2: Outer Product view of matrix multiplication

Randomized Matrix Multiplication

View $\mathbf{C} = \mathbf{AB}$ as sum of rank-one matrices (or outer products) as follows (see also figure 2):

$$\mathbf{C} = \mathbf{AB} = \sum_{i=1}^{n} \mathbf{A}_{:,i}\mathbf{B}_{i,:}$$

Note that each outer product in the above summation is a rank-1 matrix. Based on this, the idea is to *randomly* select r rank-one components, which leads to the randomized algorithm for matrix multiplication given in Algorithm 2.

Randomized Matrix Multiplication

Algorithm 2 Randomized Matrix Multiplication Algorithm

```
for $l = 1, \ldots, r$ do
    Pick $i_l \in \{1, \ldots, n\}$ i.i.d. with probability $\mathbb{P}\{i_l = k\} = p_k$
end for
Return
    $$\mathbf{M} = \sum_{l=1}^{r} \frac{1}{r p_{i_l}} \mathbf{A}_{:,i_l} \mathbf{B}_{i_l,:}$$
```

Randomized Matrix Multiplication

Alternative representation: Let

$$X_l = \sum_{k=1}^{n} \frac{1}{rp_k} \mathbf{A}_{:,k} \mathbf{B}_{k,:} \mathbb{1}\{i_l = k\}$$

Then we have $\mathbf{M} = \sum_{l=1}^{r} \mathbf{X}_l$.

Randomized Matrix Multiplication

- First, note that $\mathbf{M}$ is *not a deterministic* matrix, it is a *random matrix*.
- From a statistical point of view, $\mathbf{M}$ is an **estimator** of the true matrix $\mathbf{C}$.
- Note that we have

$$\begin{aligned}\mathbb{E}[\mathbf{M}] &= \sum_{l=1}^{r} \sum_{k} \mathbb{P}\{i_l = k\} \frac{1}{rp_k} \mathbf{A}_{:,k} \mathbf{B}_{k,:} \\ &= \sum_{k} \mathbf{A}_{:,k} \mathbf{B}_{k,:} \\ &= \mathbf{AB} = \mathbf{C}\end{aligned}$$

So, $\mathbf{M}$ is an **unbiased** estimator of the true matrix $\mathbf{C}$.

Randomized Matrix Multiplication

- The error, for example $\|\mathbf{M} - \mathbf{C}\|_F$ depends on $\{p_k\}$. The probabilities $\{p_k\}$ are called as *importance sampling probabilities*.
- So how to compute p_k ?
- Is there an *optimal* way of doing so ?

Randomized Matrix Multiplication

- We can uniformly randomly select the outer products. In this case, the values are set to

$$p_k = \frac{1}{n}.$$

- One can also do non-uniform sampling, where we set the values to

$$p_k = \frac{\|\mathbf{A}_{:,k}\|_2 \|\mathbf{B}_{k,:}\|_2}{\sum_l \|\mathbf{A}_{:,l}\|_2 \|\mathbf{B}_{l,:}\|_2} \tag{1}$$

Intuition

$$58 = (5 * 10) + (3 * 2) + (1 * 2)$$

Randomized Matrix Multiplication

In this case, p_k can be computed using one pass and $\mathcal{O}(n)$ memory. Intuitively this biases the sampled rank-1 components towards "large" rank-1 matrix. Here by "large" we mean the components that contribute most to towards the sum. There are two results we can provide about Algorithm 2 with the choice of importance sampling probabilities in Equation (1) which makes the use of this algorithm appealing in practice.

Randomized Matrix Multiplication

The choice of importance sampling probabilities in Equation (1), minimizes $\mathbb{E}\left[\|\mathbf{M} - \mathbf{AB}\|_F^2\right]$. To see that, note since $\mathbb{E}[\mathbf{M}] = \mathbf{C}$, we have

$$\mathbb{E}\left[\|\mathbf{M} - \mathbf{AB}\|_F^2\right] = \mathbb{E}\left[\sum_{i,j}(\mathbf{M}_{i,j} - \mathbf{A}_{i,:}\mathbf{B}_{:,j})^2\right] = \sum_{i,j} \text{Var}\left[\mathbf{M}_{i,j}\right] \tag{2}$$

$$= \frac{1}{r}\sum_k \sum_{i,j} \frac{\mathbf{A}_{i,k}^2 \mathbf{B}_{k,j}^2}{p_k} - \frac{1}{r}\sum_{i,j}(\mathbf{A}_{i,:}\mathbf{B}_{:,j})^2 \tag{3}$$

$$= \frac{1}{r}\sum_k \frac{1}{p_k}\|\mathbf{A}_{:,k}\|_2^2 \|\mathbf{B}_{k,:}\|_2^2 - \frac{1}{r}\|\mathbf{AB}\|_F^2$$

Randomized Matrix Multiplication

In addition, by Cauchy-Schwarz inequality, we have for sequences α_k and p_k,

$$\left(\sum_k \sqrt{\alpha_k}\right)^2 \leq \left(\sum_k p_k\right)\left(\sum_k \frac{\alpha_k}{p_k}\right)$$

with equality attained if $p_k \propto \sqrt{\alpha_k}$. Note that, we have by definition, $\sum_k p_k = 1$. Hence, we have

$$\begin{aligned} & \frac{1}{r}\sum_k \frac{1}{p_k}\|\mathbf{A}_{:,k}\|_2^2\|\mathbf{B}_{k,:}\|_2^2 - \frac{1}{r}\|\mathbf{AB}\|_F^2 \\ \geq\ & \frac{1}{r}\left(\sum_k \|\mathbf{A}_{:,k}\|_2\|\mathbf{B}_{k,:}\|_2\right)^2 - \frac{1}{r}\|\mathbf{AB}\|_F^2 \end{aligned}$$

where the lower bound is attained when $p_k \propto \|\mathbf{A}_{:,k}\|_2\|\mathbf{B}_{k,:}\|_2$. Together with the constraint that $\sum_k p_k = 1$, we have the choice of p_k in Equation (1).

Randomized Matrix Multiplication

Next, we will show that the algorithm has less error, with **very high probability**.

Theorem

Suppose

$$p_k = \frac{\|\mathbf{A}_{:,k}\|_2 \|\mathbf{B}_{k,:}\|_2}{\sum_l \|\mathbf{A}_{:,l}\|_2 \|\mathbf{A}_{l,:}\|_2}.$$

Then, as long as

$$r = C \log n$$

we have

$$\frac{\|\mathbf{M} - \mathbf{AB}\|_F}{\|\mathbf{A}\|_F \|\mathbf{B}\|_F} \leq \sqrt{\frac{1}{C}} \tag{4}$$

with probability $1 - \mathcal{O}(n^{-10})$.

Randomized Matrix Multiplication

Remark

Lets say we require the relative error in the RHS of equation 4 to be 10^{-1} with high-probability. Then we just pick $C = 100$. For large matrices, that is n being order of millions, the above algorithm practically is very fast and give high-accuracy solution with probability ≈ 1.

Randomized Matrix Multiplication

Before proving Theorem 2, you are required to understand the notation and the result in Theorem 3.
The proof involves calculating the quantity V and R in Theorem 3 and applying the result in Equation 5.

Randomized Matrix Multiplication

Theorem (Matrix Bernstein Inequality (Thm 3))

Let $\{\mathbf{X}_l\}_{l=1}^{L} \in \mathbb{R}^{d_1 \times d_2}$ *be a sequence of independent zero-mean random matrices. Assume that each matrix satisfies* $\|X_l\| \leq R$*. Let*

$$V = \max\left\{\left\|\mathbb{E}\left[\sum_{l=1}^{L} \mathbf{X}_l \mathbf{X}_l^\top\right]\right\|, \left\|\mathbb{E}\left[\sum_{l=1}^{L} \mathbf{X}_l^\top \mathbf{X}_l\right]\right\|\right\}.$$

Then we have

$$\mathbb{P}\left\{\left\|\sum_{l=1}^{L} \mathbf{X}_l\right\| \geq \tau\right\} \leq (d_1 + d_2) \exp\left(\frac{-2\tau^2/2}{V + R\tau/3}\right)$$

Randomized Matrix Multiplication

To understand the above concentration inequality, answer the following two questions:

1. What happens when τ is not too large ? Can you see we have $\exp\left(-\tau^2/V\right)$
2. What happens when τ is large ? Can you see we have $\exp\left(-\tau/R\right)$

Randomized Matrix Multiplication

Fact

From the above theorem, one can show that we also have, with probability $1 - \mathcal{O}\left((d_1 + d_2)^{-10}\right)$

$$\left\| \sum_{l=1}^{L} \mathbf{X}_l \right\| \leq \sqrt{V \log(d_1 + d_2)} + R \log(d_1 + d_2) \tag{5}$$

Proof of Theorem 2

Step 1: Calculating V and R:
Let

$$X_l = \sum_{k=1}^{n} \frac{1}{rp_k} \mathbf{A}_{:,k} \mathbf{B}_{k,:} \mathbb{1}\{i_l = k\}$$

Then we have $\mathbf{M} = \sum_{l=1}^{r} \mathbf{X}_l$. Using Fact 1, we also have

$$\begin{aligned} \|\mathbf{X}_l\| &\leq \max_k \left\{ \frac{1}{rp_k} \|\mathbf{A}_{:,k}\|_2 \|\mathbf{B}_{k,:}\|_2 \right\} \\ &= \frac{1}{r} \sum_{k=1}^{n} \|\mathbf{A}_{:,k}\|_2 \|\mathbf{B}_{k,:}\|_2 \\ &= R \end{aligned}$$

Proof of Theorem 2 (continued)

Next, again using Fact 1, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{l=1}^{r} \|\mathbf{X}_l\|^2\right] &= r\sum_{k=1}^{n} \frac{\mathbb{P}\{i_l = k\}\|\mathbf{A}_{:,k}\|_2^2\|\mathbf{B}_{k,:}\|_2^2}{r^2 p_k^2} \\
&\leq \frac{\left(\sum_{k=1}^{n} \|\mathbf{A}_{:,k}\|_2\|\mathbf{B}_{k,:}\|_2\right)^2}{r} \\
&= V
\end{aligned}$$

Note that R and V differ only by a square factor in the numerator.

Proof of Theorem 2 (continued)

Step 2: Applying the result in Equation 5:

Based on the result in Step 1, by Equation 5, we have

$$\begin{aligned}
\|\mathbf{M} - \mathbf{C}\|_F &\le \sqrt{V \log n} + R \log n \\
&\le \sqrt{\frac{\log n}{r}} \left(\sum_{k=1}^{n} \|\mathbf{A}_{k,:}\|_2 \|\mathbf{B}_{k,:}\|_2 \right) \\
&\le \sqrt{\frac{\log n}{r}} \|\mathbf{A}\|_F \|\mathbf{B}\|_F
\end{aligned}$$

where the last step follows by Cauchy-Schwarz.

More Information

You are not required to understand the Matrix Bernstein Inequality in full detail or its proof. Make sure you understand Equation (5) and how it is used in the proof of Theorem 2. If you are curious about the result, you can refer to [Tro15]. You are also encouraged to first read scalar version of Bernstein's inequality from wikipedia: scalar Bernstein's inequality.

More Information

Matrices (and tensors) play a crucial role in large-scale data analysis. A crucial aspect of dealing with such large-scale matrices is the use of randomization for performing even routine operations that otherwise become computationally prohibitive. For more details, refer to [Mah16].

Randomized Algorithms for Eigenvector Computation

Power Method

- Eigenvalues and Eigenvectors are uniformly popular across almost all disciplines from probability, statistics, applied mathematics, computer science, etc. because of their wide applicability, for both theoretical and practical purposes.
- For simplicity, we will focus only on positive semidefinite matrices in this notes. Recall that a square matrix $\mathbf{A} \in \mathbb{R}^d$ is positive semidefinite, if $\forall\ \mathbf{b} \in \mathbb{R}^d$, we have the quadratic form satisfy $\mathbf{b}^\top \mathbf{A}\mathbf{b} \geq 0$. Furthermore, recall that the Frobenius norm could also be represented using the trace as follows:

$$\|\mathbf{A}\|_F^2 = \mathrm{tr}(\mathbf{A}\mathbf{A}^\top).$$

Power Method

Power method is one of the oldest method to compute the eigenvectors. Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be a **deterministic** positive semi-definite matrix. That means that we have the following decomposition

$$\mathbf{A} = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top$$

for some (unit-norm) eigenvectors $\mathbf{v}_i \in \mathbb{R}^d$ and eigenvalues λ_i.

Fact

Eigenvectors $\mathbf{v}_i$, $i = 1, \ldots, n$ form a basis for the Euclidean space $\mathbb{R}^d$.

Power Method

- We assume that $\lambda_1 > \lambda_i$ for all $i = 2, \ldots, d$. Such an eigenvalue is then an **unique/isolated** top eigenvalue and the corresponding eigenvector $\mathbf{v}_1$ is called as the top eigenvector.
- Note that it is not guaranteed that every positive semi-definite matrix has such a top eigenvector – **this is an assumption!**
- The problem, we consider is given a matrix $\mathbf{A}$ that has the above structure, find a vector $\mathbf{c} \in \mathbb{R}^n$ that is an approximation to $\mathbf{v}_1$.

Power Method

We also note the following alternative way of representing eigenvectors:

$$\begin{aligned}
\mathbf{v}_1 &= \underset{\mathbf{v}:\ \|\mathbf{v}\|_2=1}{\arg\min} \left\{ \|\mathbf{A} - \lambda_1 \mathbf{v}\mathbf{v}^\top\|_F^2 = \|\mathbf{A}\|_F^2 + \|\lambda_1 \mathbf{v}\mathbf{v}^\top\|_F^2 - 2\lambda_1 \mathrm{tr}(\mathbf{A}\mathbf{v}\mathbf{v}^\top) \right\} \\
&= \underset{\mathbf{v}:\ \|\mathbf{v}\|_2=1}{\arg\min} \left\{ \|\mathbf{A}\|_F^2 + \|\lambda_1 \mathbf{v}\mathbf{v}^\top\|_F^2 - 2\lambda_1 \mathrm{tr}(\mathbf{v}^\top \mathbf{A}\mathbf{v}) \right\} \\
&= \underset{\mathbf{v}:\ \|\mathbf{v}\|_2=1}{\arg\max} \quad \mathrm{tr}(\mathbf{v}^\top \mathbf{A}\mathbf{v})
\end{aligned}$$

Power Method

The power method is described in Algorithm 3.

Algorithm 3 Power Method

```
**Input:** Unit-vector $\mathbf{c}^{(0)} \in \mathbb{R}^d$ (Randomly generated) and number of iterations $T$.
for $t = 1, \ldots, T$ do
    $\mathbf{c}^{(t)} = \mathbf{A}\mathbf{c}^{(t-1)}$
    $\mathbf{c}^{(t)} = \frac{\mathbf{c}^{(t)}}{\|\mathbf{c}^{(t)}\|_2}$        (Normalization step)
end for
```

Power Method

- Note that, if we ignore the normalization step, we have $\mathbf{c}^{(t)} = \mathbf{A}^t \mathbf{c}^{(0)}$.
- Note that any vector could be written as a linear combination of the d eigenvectors $\mathbf{v}_i$.
- Specifically, the random vector $\mathbf{c}^{(0)}$ given as input to Algorithm 3 could be written as follows, for some constants $\gamma_1, \ldots, \gamma_d$.

$$\mathbf{c}^{(0)} = \gamma_1 \mathbf{v}_1 + \gamma_2 \mathbf{v}_2 + \ldots + \gamma_d \mathbf{v}_d$$

Power Method

- Now, look at the iterates of Algorithm 3, without normalization step. We then have

$$\begin{aligned}\mathbf{c}^{(1)} = \mathbf{A}\mathbf{c}^{(0)} &= \gamma_1 \mathbf{A}\mathbf{v}_1 + \gamma_2 \mathbf{A}\mathbf{v}_2 + \ldots + \gamma_d \mathbf{A}\mathbf{v}_d \\ &= \gamma_1 \lambda_1 \mathbf{v}_1 + \gamma_2 \lambda_2 \mathbf{v}_2 + \ldots + \gamma_d \lambda_d \mathbf{v}_d.\end{aligned}$$

- Similarly, we have

$$\begin{aligned}\mathbf{c}^{(2)} = \mathbf{A}\mathbf{c}^{(1)} &= \gamma_1 \lambda_1 \mathbf{A}\mathbf{v}_1 + \gamma_2 \lambda_2 \mathbf{A}\mathbf{v}_2 + \ldots + \gamma_d \lambda_d \mathbf{A}\mathbf{v}_d \\ &= \gamma_1 \lambda_1^2 \mathbf{v}_1 + \gamma_2 \lambda_2^2 \mathbf{v}_2 + \ldots + \gamma_d \lambda_d^2 \mathbf{v}_d.\end{aligned}$$

- For the general case, we have

$$\mathbf{c}^{(t+1)} = \mathbf{A}\mathbf{c}^{(t)} = \gamma_1 \lambda_1^{t+1} \mathbf{v}_1 + \gamma_2 \lambda_2^{t+1} \mathbf{v}_2 + \ldots + \gamma_d \lambda_d^{t+1} \mathbf{v}_d.$$

Power Method

- Recall that, we have assumed λ_1 is larger than other other eigenvalues. So, when it is raised to the power $(t+1)$, the vector $\mathbf{c}^{(t+1)}$ is most correlated (or aligned) in the direction of $\mathbf{v}_1$.
- But if λ_2 is very close to λ_1, then t might have to be really large to get $\mathbf{c}^{(t+1)}$ correlated with $\mathbf{v}_1$.

Power Method

- Now, suppose instead of normalizing with the $\|\mathbf{c}^{(t)}\|_2$, we normalize with λ_1. [Note: λ_1 is typically not known and might be hard to calculate. This is done, just for understanding purpose.]
- In this case, we have $\mathbf{c}^{(t+1)} =$

$$\gamma_1 \left(\frac{\lambda_1}{\lambda_1}\right)^{t+1} \mathbf{v}_1 + \gamma_2 \left(\frac{\lambda_2}{\lambda_1}\right)^{t+1} \mathbf{v}_2 + \ldots + \gamma_d \left(\frac{\lambda_d}{\lambda_1}\right)^{t+1} \mathbf{v}_d$$

$$= \gamma_1 \mathbf{v}_1 + \gamma_2 \left(\frac{\lambda_2}{\lambda_1}\right)^{t+1} \mathbf{v}_2 + \ldots + \gamma_d \left(\frac{\lambda_d}{\lambda_1}\right)^{t+1} \mathbf{v}_d$$

Power Method

- Note that, in this case, $\mathbf{c}^{(t+1)}$ will be correlated with $\mathbf{v}_1$ even within a few iterations (small value of t). But obviously, we might not know λ_1.
- A compromise is to estimate just to direction of eigenvector $\mathbf{v}_1$.
- So, we assume that $\mathbf{v}_1$ is an unit vector, all the magnitude information is absorbed into the eigenvalue λ_1.
- Thus, in Algorithm 3, we just normalize with the magnitude of $\mathbf{c}$ vectors to estimate the direction.

Power Method

Note that from the power method, we can also compute the eigenvalue in each iteration as

$$\hat{\lambda}_1^{(t)} = [\mathbf{c}^{(t-1)}]^\top \mathbf{c}^{(t)}.$$

Hence, define the relative error of eigenvalue estimation after T steps as

$$e_T = \frac{\lambda_1 - \hat{\lambda}_1^{(T)}}{\lambda_1}.$$

Power Method

Theorem

Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be a PSD matrix. Let the starting vector $\mathbf{c}^{(0)}$ be drawn from $N(0, I_d)$. Then, after T iterations of the power method, we have

$$\mathbb{E}[e_T] \leq 0.871 \, \frac{\log d}{T-1}, \quad \forall T \geq 2.$$

Furthermore, if we let $\varkappa = \frac{\lambda_1 - \lambda_2}{\lambda_1}$ be the relative spectral gap. Then we have

$$\mathbb{E}[e_T] \leq 1.254 \, \sqrt{d}\varkappa \, \exp^{-T\varkappa}, \quad \forall T \geq 1.$$

Proof and more details in [KW92].

Power Method

- In some applications, the eigenvector might by a sparse vector, i.e., only k of the d co-ordinates are non-zero.
- The rest of the co-ordinates are zero. The above power method in Algorithm 3 could then be easily modified to this situation, with a simple truncation argument in each step.
- The detailed algorithm is given in Algorithm 4.

Power Method

Algorithm 4 Truncated Power Method

```
Input: Unit-vector $\mathbf{c}^{(0)} \in \mathbb{R}^d$ (Randomly generated) and number of iterations $T$.
for $t = 1, \ldots, T$ do
    $\mathbf{c}^{(t)} = \mathbf{A}\mathbf{c}^{(t-1)}$
    $\mathbf{c}^{(t)} = \frac{\mathbf{c}^{(t)}}{\|\mathbf{c}^{(t)}\|_2}$        (Normalization step)
    Let $F^{(t)} \subset \{1, \ldots, d\}$ be the set of co-ordinates of $\mathbf{c}^{(t)}$ corresponding to top $k$ absolute values.
    Keep the values of $\mathbf{c}^{(t)}$ in the co-ordinates index by the set $F^{(t)}$ as such and set the rest of the coordinates to zero and re-normalize.
end for
```

Power Method

- For both the power method and truncated power method, the difference between λ_1 and λ_2 determines the rate of convergence. Furthermore, its crucial how the initialization vector is generated.

Statistical Application: PCA

- Principal component analysis (PCA) is arguably the most popular dimensionality reduction technique in statistics.
- In linear PCA, we try to represent each data sample as linear combination of some fixed basis vector. That is[1],

$$X^{(i)} = \sum_{j=1}^{p} \mathbf{v}_j \beta_j^{(i)} = \mathbf{V}\beta^{(i)}$$

where $\mathbf{V} \in \mathbb{R}^{d \times p}$ is the matrix with the basis vectors $\mathbf{v}_j$ as its columns and $\beta_j^{(i)}$ denote the j^{th} coordinate of the vector $\beta^{(i)}$.

[1]Note that in the previous section, we used the superscript (t), to denote the number of iteration of the power method. Now, we use it to denote the samples.

Statistical Application: PCA

- Note that the basis vectors are fixed and does not change with i and the coefficients change with i.
- The reasoning behind this is that, you want to represent each sample as a different linear combination of the same basis vector.
- For example, if the $X^{(i)}$ represent human faces, then $\mathbf{v}_j$ represents “template" human faces. Then PCA posits that every human face is indeed some linear combination of some template faces. This reasoning is motivated by empirical studies in human face anatomy.
- Furthermore, this reasoning also appears to be true for several different fields, which made PCA widely applicable in practice.

Statistical Application: PCA

- In the PCA model above, note that without any other assumptions on $\mathbf{V}$, the model is not unique. Hence we assume that the matrix $\mathbf{V}$ is such that $\mathbf{V}^\top \mathbf{V} = I$.
- That is, the basis vectors $\mathbf{v}_j$ are orthogonal. Given this, PCA boils down to the to task of estimating the vector $\{\hat{\beta}^{(i)}\}_{i=1}^n$ and the matrix $\hat{\mathbf{V}}$ that minimizes certain reconstruction error, given the data $\{X^{(i)}\}_{i=1}^n \in \mathbb{R}^d$.
- That is ,we have

$$\{\hat{\beta}^{(i)}\}_{i=1}^n, \hat{\mathbf{V}} = \underset{\substack{\mathbf{V} \text{ such that } \mathbf{V}^\top \mathbf{V} = I \\ \{\beta^{(i)}\}_{i=1}^n \in \mathbb{R}^p}}{\arg\min} \sum_{i=1}^n \|X^{(i)} - \mathbf{V}\beta^{(i)}\|_2^2$$

Statistical Application: PCA

- The above problem is a non-convex problem. Fortunately, one can use Eigenvalue decomposition to get the global solution. In order to see that, first assume that we want to find $\hat{\beta}^{(i)}$ for each i, given the optimal basis matrix $\hat{\mathbf{V}}$.
- In order to do that, we have

$$\hat{\beta}^{(i)} = \arg\min_{\beta} \|X^{(i)} - \hat{\mathbf{V}}\beta^{(i)}\|_2^2$$

  for all $i = 1, \ldots, n$.
- This is because the objective function for finding $\hat{\beta}^{(i)}$ does not depend on any other terms involving $\hat{\beta}^{(k)}$, where $k \neq i$.
- Note that the above problem is a linear least-squares problem and so we have

$$\hat{\beta}^{(i)} = (\hat{\mathbf{V}}^\top \hat{\mathbf{V}})^{-1} \hat{\mathbf{V}}^\top X^{(i)} = \hat{\mathbf{V}}^\top X^{(i)},$$

  as the optimal basis matrix is orthogonal.

Statistical Application: PCA

- The main message here is that the optimal $\beta^{(i)}$ is going to have form $\hat{\mathbf{V}}^\top X^{(i)}$ for each i, if we know $\hat{\mathbf{V}}$. But we don't know $\hat{\mathbf{V}}$.
- In order to find $\hat{\mathbf{V}}$, substitute $\beta^{(i)} = \mathbf{V}^\top X^{(i)}$ in the original objective function to get:

$$\hat{\mathbf{V}} = \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\min} \sum_{i=1}^{n} \|X^{(i)} - \mathbf{V}\mathbf{V}^\top X^{(i)}\|_2^2$$

Statistical Application: PCA

The above problem could be expanded out as

$$\hat{\mathbf{V}} = \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\min} \sum_{i=1}^{n} \left({X^{(i)}}^\top X^{(i)} - {X^{(i)}}^\top \mathbf{V}\mathbf{V}^\top X^{(i)} \right)$$

$$\underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\min} \sum_{i=1}^{n} \left({X^{(i)}}^\top X^{(i)} - \operatorname{trace}\left({X^{(i)}}^\top \mathbf{V}\mathbf{V}^\top X^{(i)} \right) \right)$$

Statistical Application: PCA

Note that the above minimization problem is equivalent to the following maximization problem:

$$\begin{aligned}
\hat{\mathbf{V}} = & \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\max} \sum_{i=1}^{n} \left(\operatorname{trace}\left(X^{(i)^\top} \mathbf{V}\mathbf{V}^\top X^{(i)} \right) \right) \\
& \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\max} \sum_{i=1}^{n} \left(\operatorname{trace}\left(X^{(i)} X^{(i)^\top} \mathbf{V}\mathbf{V}^\top \right) \right) \\
& \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\max} \sum_{i=1}^{n} \left(\operatorname{trace}\left(\mathbf{V}^\top X^{(i)} X^{(i)^\top} \mathbf{V} \right) \right) \\
& \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\max} \left(n \cdot \operatorname{trace}\left(\mathbf{V}^\top \left(\frac{1}{n} \sum_{i=1}^{n} X^{(i)} X^{(i)^\top} \right) \mathbf{V} \right) \right) \\
& \underset{\mathbf{V} \text{ such that } \mathbf{V}^\top V = I}{\arg\max} \operatorname{trace}\left(\mathbf{V}^\top \left(\hat{\Sigma}_n \right) \mathbf{V} \right)
\end{aligned}$$

Statistical Application: PCA

- In the above steps, we have used two properties of trace: (i) $\text{trace}(A + B) = \text{trace}(A) + \text{trace}(B)$ and (ii) $\text{trace}(ABC) = \text{trace}(CAB) = \text{trace}(BCA)$.
- Also, in the last step, we dropped n as it plays no part in the maximization. The above problem involves the sample covariance matrix $\hat{\Sigma}_n = \frac{1}{n}\sum_{i=1}^{n} X^{(i)}{X^{(i)}}^\top$.

Statistical Application: PCA

- Lets try to interpret the above problem. Assume that $p = 1$. Hence $\mathbf{V} = \mathbf{v}_1 \in \mathbb{R}^d$. In this case, the above problem becomes

$$\begin{aligned}\hat{\mathbf{v}}_1 &= \underset{\mathbf{v}\ :\mathbf{v}^\top \mathbf{v}=1}{\arg\max}\ \operatorname{trace}\left(\mathbf{v}^\top \left(\hat{\Sigma}_n\right) \mathbf{v}\right) \\ &= \underset{\mathbf{v}\in\mathbb{R}^{\mathbf{d}}\ :\mathbf{v}^\top \mathbf{v}=1}{\arg\max}\ \mathbf{v}^\top \left(\hat{\Sigma}_n\right) \mathbf{v}\end{aligned}$$

- We see that $\hat{\mathbf{v}}_1$ is nothing but the top eigenvector of the sample covariance matrix. We can use the power method in Algorithm 3 to calculate them. Specifically, we take $\mathbf{A} = \hat{\Sigma}_n$.

Statistical Application: PCA

- The above discussion of linear PCA implicitly that $n > d$. Only then, the sample covariance matrix will convergence to the true covariance matrix and as a consequence, the top eigenvectors of the sample covariance matrix will convergence correspondingly to the top eigenvectors of the true covariance matrix.
- When $n < d$, we see that the sample covariance matrix is not full-rank. Hence it will not converge to the true covariance matrix and as a consequence the sample eigenvectors won't converge to the truth.
- So we enforce sparsity constraint on the $\mathbf{v}_1$ vector and use Algorithm 4 instead. Specifically, we have the following problem:

$$\hat{\mathbf{v}}_1 = \underset{\mathbf{v_1} \in \mathbb{R}^{\mathbf{d}} \;:\; \mathbf{v}_1^\top \mathbf{v_1} = 1 \text{ and } \|\mathbf{v_1}\|_0 = s}{\arg\max} \; \mathbf{v}_1^\top \left(\hat{\Sigma}_n \right) \mathbf{v}_1$$

Statistical Application: PCA

- Here the constraint $\|\mathbf{v}_1\|_0 = s$ ensures that there are only s non-zero entries in the d dimensional vector $\mathbf{v}_1$. This is a NP-hard combinatorial optimization problem (a.k.a *bad* problem) and takes exponential time to be solved.
- It turns out that, if one is willing to spend the exponential time and solve the above problem, then $d > n > s \log(d)$ samples are required for the sample eigenvector to converge to the true eigenvector.
- But if we want an efficient algorithm (that is an algorithm that runs in polynomial time rather than exponential time), then any such algorithm **must** require $d > n > s^2 \log(d)$ to work.
- The truncated power method, outlined in 4 with $\mathbf{A} = \hat{\Sigma}_n$ achieves this sample complexity requirement.
- The intriguing phenomenon with sparse PCA is even if you use other *complicated algorithms*, they too require the same number of samples $n > s^2 \log(d)$ to work. This is often times the case in several problems.

Statistical Application: PCA

- Coming back to regular (non-sparse) PCA, recall that we use Algorithm 3 with $\mathbf{A} = \hat{\Sigma}_n$. In each iteration, we are re-using the set of n samples again and again.
- Can we do better ? It turns out that, we can formulate an algorithm that uses only 1 sample at a time.
- Hence, in this case, the number of iterations of the algorithm and the number of samples used by the algorithm becomes the same!

Statistical Application: PCA

- In order to provide the algorithm, first note that the population version of the PCA problem is given by

$$\mathbf{v}_1 = \underset{\mathbf{v}\in\mathbb{R}^{\mathbf{d}}\,:\,\mathbf{v}^\top\mathbf{v}=1}{\arg\max}\ \mathbf{v}^\top(\Sigma)\,\mathbf{v} = \underset{\mathbf{v}\in\mathbb{R}^{\mathbf{d}}\,:\,\mathbf{v}^\top\mathbf{v}=1}{\arg\max}\ f(\Sigma,\mathbf{v})$$

  where $\Sigma = \mathbb{E}[XX^\top] = \int_{\mathbb{R}^d} XX^\top P(X)$.
- In practice, we don't observe $P(X)$ but we will be able to obtain a sequence of observations sampled from $P(X)$.

Statistical Applications: PCA

- Furthermore, note that $\nabla_{\mathbf{v}} f(\Sigma, \mathbf{v}) = \mathbf{\Sigma v} \in \mathbb{R}^d$.
- The algorithm is then based on calculating the gradient using one sample in each iteration and is provided in Algorithm 5.
- Note that the algorithm is very fast to implement and uses only 1 sample in each iteration as opposed to the Algorithm 3 with $\mathbf{A} = \hat{\Sigma}_n$.
- Finally, although we have not provided rates of convergence of the above algorithms, rigorous rates have been established in the recent past for these algorithms.

Statistical Applicatins: PCA

Algorithm 5 Stochastic/Online Algorithm for PCA

```
**Input:** Unit-vector $\mathbf{v}^{(0)} \in \mathbb{R}^d$ (Randomly generated) and step-size $\beta$.
**for** $t = 1, \ldots, T$ **do**
    Sample $X^{(t)}$ from the distribution P(X).
    Let $\mathbf{v}^{(t)} = \mathbf{v}^{(t-1)} + \beta X^{(t)} {X^{(t)}}^\top \mathbf{v}^{(t-1)}$
    $\mathbf{v}^{(t)} = \frac{\mathbf{v}^{(t)}}{\|\mathbf{v}^{(t)}\|_2}$
**end for**
```

Statistical Applications: PCA

- Note that the Algorithm 5 is very fast to implement and uses only 1 sample in each iteration as opposed to the Algorithm 3 with $\mathbf{A} = \hat{\Sigma}_n$.
- Although we have not provided rates of convergence of the above algorithms, rigorous rates have been established in the recent past for these algorithms.
- Finally, Algorithm 5 is an example of Stochastic/Online algorithms. Later in this course, we will study this in more details in a general context.

Randomized Algorithms for Least Squares

Least Squares Regression

Now, we will study randomized algorithm for computing ordinary least squares and ridge regression. OLS and Ridge regression are popular techniques for studying *linear relationships* between the covariates ($X^{(i)} \in \mathbb{R}^d$) and response ($Y^{(i)} \in \mathbb{R}$) given n samples. We denote the data matrix, with $X^{(i)}$ as its column, by $\mathbf{X} \in \mathbb{R}^{n\times d}$. Similarly, we denote the response vector, with $Y^{(i)}$ as its entries, by the vector $\mathbf{y} \in \mathbb{R}^n$. Then the OLS estimate of the linear relationship between $\mathbf{X}$ and $\mathbf{y}$ is given by the following problem:

$$\mathbf{b} = \arg\min_{\mathbf{c}\in\mathbb{R}^d} \|\mathbf{X}\mathbf{c} - \mathbf{y}\|_2^2. \tag{6}$$

Least Squares Regression

In statistics, typically the $\mathbf{X}$ is assumed to be random matrix and a linear model assumption is explicitly made between $\mathbf{X}$ and $\mathbf{y}$ to study the statistical properties of the estimator $\mathbf{b}$. Our discussion below proceeds by conditioning on all the randomness in the data matrix and the response. That is, we assume $\mathbf{X}$ and $\mathbf{y}$ to be deterministic for the purpose of discussion.

Least Squares Regression

- The solution of the problem 6 satisfies the following normal equations:

$$\mathbf{X}^\top \mathbf{X}\mathbf{b} = \mathbf{X}^\top \mathbf{y}. \tag{7}$$

- When $\mathbf{X}$ has full column-rank, then we can write the solution of 6 in closed form as follows:

$$\mathbf{b} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}.$$

  If we have the following SVD for $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top$, then we have

$$\mathbf{b} = \mathbf{V}\Lambda^{-1}\mathbf{U}^\top \mathbf{y}.$$

- Furthermore, the residual vector $\mathbf{r}_\mathbf{b} = \mathbf{y} - \mathbf{X}\mathbf{b} \in \mathbb{R}^n$ is orthogonal to the column space of $\mathbf{X}$. That is, we have

$$\mathbf{r}^\top \mathbf{X} = (\mathbf{y} - \mathbf{X}\mathbf{b})^\top \mathbf{X} = 0$$

Least Squares Regression

- Direct methods, that use the closed form solution have a complexity of $\mathcal{O}(nd^2)$.
- Iterative methods for directly solving the optimization problem above also have similar complexity.
- When considering the regime of of d being small but $d^2 \leq n \leq e^d$, these methods are computationally prohibitive.

Least Squares Regression

Sketching

The idea of sketching is simple: One can potentially construct a matrix $\Phi \in \mathbb{R}^{r \times n}$ and solve the following **sketched-OLS** problem instead of OLS:

$$\mathbf{b}_s = \arg\min_{\mathbf{c} \in \mathbb{R}^d} \|\Phi(\mathbf{X}\mathbf{c} - \mathbf{y})\|_2^2 = \arg\min_{\mathbf{c} \in \mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{c} - \Phi\mathbf{y}\|_2^2$$

Similar to the previous case, we have the closed form expression,

$$\mathbf{b}_s = \left((\Phi\mathbf{X})^\top \Phi\mathbf{X} \right)^{-1} (\Phi\mathbf{X})^\top \Phi\mathbf{y}. \tag{8}$$

Least Squares Regression

- The matrix Φ is called as the **sketching matrix**. It reduces the dimensionality of the problem with respect to the sample size n.
- The other alternative to sketching is the trivial method: just **throw away** part of the samples.
- When we instead use the sketching idea and if the matrix Φ is constructed **carefully**, we would have a huge computational saving with very little loss in accuracy of the obtained solution.
- A popular way to do Sketched-OLS is described in Algorithm 6.

Algorithm 6 The **Sketched-OLS** algorithm

```
Input: $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^n$, and an error parameter $\epsilon \in (0, 1)$.
Output: $\mathbf{b}_s \in \mathbb{R}^d$.
  1. Let $r \approx \frac{d \log(n)}{\epsilon}$.
  2. Let $\mathbf{S}$ be an empty matrix.
  3. For $t = 1, \ldots, r$ (i.i.d. trials with replacement) select uniformly at random an integer from $\{1, 2, \ldots, n\}$.
       ▶ If $i$ is selected, then append the column vector $\left(\sqrt{n/r}\right) \mathbf{e}_i$ to $\mathbf{S}$, where $\mathbf{e}_i \in \mathbb{R}^n$ is the $i$-th canonical vector.
  4. Let $\mathbf{H} \in \mathbb{R}^{n \times n}$ be the normalized Hadamard transform matrix.
  5. Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be a diagonal matrix with
       $$\mathbf{D}_{ii} = \begin{cases} +1 & \text{, with probability } 1/2 \\ -1 & \text{, with probability } 1/2 \end{cases}$$
  6. Compute and return $\mathbf{b}_s = \left(\mathbf{S}^T \mathbf{HDX}\right)^{\dagger} \mathbf{S}^T \mathbf{HDy}$, where for a matrix $\mathbf{A}$, $\mathbf{A}^{\dagger} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top$.
```

Least Squares Regression

- Basically, the sketching matrix in this algorithms is defined by $\Phi = \mathbf{S}^T\mathbf{H}\mathbf{D}$ where $\mathbf{S} \in \mathbb{R}^{n\times r}$ and $\mathbf{H}, \mathbf{D} \in \mathbb{R}^{n\times n}$.
- Notice that the algorithm fixes $r \approx d\log(n)/\epsilon$. Hence, even when $n \approx e^d$, we have $r \approx d^2/\epsilon$ which provides the computational benefits.
- Specifically, the sketched-OLS in Equation 8 could be computed in this case in $\mathcal{O}(rd^2) \approx d^4/\epsilon$ running time.

Least Squares Regression

- The matrix $\mathbf{S}$ and $\mathbf{D}$ are described in Algorithm 6. The matrix $\mathbf{H}$ is called as a *normalized* Hadamard transform matrix. First, the Hadamard transformation matrix is defined recursively as follows.

$$\tilde{\mathbf{H}}_2 = \begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix} \qquad \text{and} \qquad \tilde{\mathbf{H}}_n = \begin{pmatrix} \tilde{\mathbf{H}}_{n/2} & \tilde{\mathbf{H}}_{n/2} \\ \tilde{\mathbf{H}}_{n/2} & -\tilde{\mathbf{H}}_{n/2} \end{pmatrix}$$

- Then, the normalized Hadamard transformation matrix is defined as $\mathbf{H} = (1/\sqrt{n})\tilde{\mathbf{H}}_n$.

Least Squares Regression

- The matrix **HD** is called as **randomized Hadamard transform** as **D** is random and has nice properties. Specifically, the matrix-vector multiplication **HDe** could be done in $n \log(n)$ time for any vector **e**.
- **Note: A naive implementation might take $\mathcal{O}(n^2)$ time but the fast Hadamard transform approach outlined in** `https://en.wikipedia.org/wiki/Fast_Walsh-Hadamard_transform` **has the above mentioned complexity**.

Least Squares Regression

- The matrix $\mathbf{S}$ is a *sub-sampling* matrix which also contributes to the randomness in the algorithm. Recall that, we assume that $\mathbf{X}$ and $\mathbf{y}$ are deterministic in our discussion.
- That is, you can think as follows: all of the discussion above is done condition on $\mathbf{X}$ and $\mathbf{y}$ if they are random.
- So in the algorithm as such, the the only source of randomness in through $\mathbf{D}$ and $\mathbf{S}$.
- So, when you get to Fact 1 and encounter the phrase, **with high probability** we refer to this source of randomness only.

Least Squares Regression

What is a *good* sketching matrix ?

Basically, any matrix Φ that satisfies the two assumptions stated in Lemma 6 is called as a good sketching matrix. Instead of stating the assumption upfront, we outline the proofs from which we can see where the stated assumptions arise from. To fix notation, we denote the residual vectors with respect to OLS solution as $\mathbf{r_b} = \mathbf{y} - \mathbf{Xb}$ and the residual with the estimator $\mathbf{b}_s$ as $\mathbf{r}_{\mathbf{b}_s} = \mathbf{y} - \mathbf{X}\mathbf{b}_s$.

Least Squares Regression

Theorem

Let $\mathbf{b}$ *be the OLS solution and* $\mathbf{b}_s$ *be the Sketched-OLS solution. Let the sketching matrix* Φ *satisfy the two conditions stated in Lemma 6. Then we have*

$$\begin{aligned} \|\mathbf{r}_{\mathbf{b}_s}\|_2^2 &\leq (1+\epsilon)\|\mathbf{r}_{\mathbf{b}}\|_2^2 \\ \|\mathbf{b}-\mathbf{b}_s\|_2^2 &\leq \frac{\|\mathbf{r}_{\mathbf{b}}\|_2^2\epsilon}{\sigma_{\min}^2(\mathbf{X})} \end{aligned}$$

Least Squares Regression

Lemma

Consider the optimal value of the following two optimization problems:

$$\min_{\mathbf{c}\in\mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{c} - \Phi\mathbf{y}\|_2^2 \qquad \min_{\mathbf{e}\in\mathbb{R}^d} \|\Phi\mathbf{U}\mathbf{e} - \Phi\mathbf{r}_{\mathbf{b}}\|_2^2.$$

Then both values are the same, i.e.,

$$\min_{\mathbf{c}\in\mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{c} - \Phi\mathbf{y}\|_2^2 = \min_{\mathbf{e}\in\mathbb{R}^d} \|\Phi\mathbf{U}\mathbf{e} - \Phi\mathbf{r}_{\mathbf{b}}\|_2^2.$$

Furthermore, if a vector $\boldsymbol{\theta}$ satisfies:

$$\mathbf{U}\boldsymbol{\theta} = \mathbf{X}(\mathbf{b}_s - \mathbf{b})$$

then, it achieves the minimum of the optimization problem in the right hand size.

Least Squares Regression I

Proof: To see the first statement, note that

$$\begin{aligned}
&\min_{\mathbf{c}\in\mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{c} - \Phi\mathbf{y}\|_2^2 \\
=& \min_{\mathbf{c}\in\mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{c} - \Phi(\mathbf{X}\mathbf{b} + \mathbf{r}_{\mathbf{b}})\|_2^2 \\
=& \min_{\mathbf{d}\in\mathbb{R}^d} \|\Phi\mathbf{X}(\mathbf{b} + \mathbf{d}) - \Phi(\mathbf{X}\mathbf{b} + \mathbf{r}_{\mathbf{b}})\|_2^2 \\
=& \min_{\mathbf{d}\in\mathbb{R}^d} \|\Phi\mathbf{X}\mathbf{d} - \Phi\mathbf{r}_{\mathbf{b}}\|_2^2 \\
=& \min_{\mathbf{e}\in\mathbb{R}^d} \|\Phi\mathbf{U}\mathbf{e} - \Phi\mathbf{r}_{\mathbf{b}}\|_2^2
\end{aligned}$$

In the above derivation, in the last step, we are allowed to replace $\mathbf{X}$ with $\mathbf{U}$ as singular vectors in $\mathbf{U}$ form a basis for the column space of $\mathbf{X}$. Hence, we just have different combination of the vectors in $\mathbf{U}$ instead of vectors in $\mathbf{X}$. This proves the first statement.

Least Squares Regression II

To see the next statement, take the objective function of the optimization problems in the right hand side and note that

$$\begin{aligned}
\|\Phi \mathbf{U}\boldsymbol{\theta} - \Phi \mathbf{r}_{\mathbf{b}}\|_2^2 &= \|\Phi \mathbf{X}(\mathbf{b}_s - \mathbf{b}) - \Phi \mathbf{r}_{\mathbf{b}}\|_2^2 \\
&= \|\Phi \mathbf{X}\mathbf{b}_s - \Phi \mathbf{X}\mathbf{b} - \Phi \mathbf{r}_{\mathbf{b}}\|_2^2 \\
&= \|\Phi \mathbf{X}\mathbf{b}_s - \Phi(\mathbf{X}\mathbf{b} + \mathbf{r}_{\mathbf{b}})\|_2^2 \\
&= \|\Phi \mathbf{X}\mathbf{b}_s - \Phi \mathbf{y}\|_2^2 \\
&= \min_{\mathbf{c}\in\mathbb{R}^d} \|\Phi \mathbf{X}\mathbf{c} - \Phi \mathbf{y}\|_2^2 \\
&= \min_{\mathbf{e}\in\mathbb{R}^d} \|\Phi \mathbf{U}\mathbf{e} - \Phi \mathbf{r}_{\mathbf{b}}\|_2^2
\end{aligned}$$

Hence, we have shown that the vector $\boldsymbol{\theta}$ indeed is indeed the vector that achieves the minimum value of the optimization problem on the right hand side.

Least Squares Regression

Lemma

Assume the following

- *Isometry assumption 1:*

$$\sigma_{\min}^2(\Phi \mathbf{U}) \geq \frac{1}{\sqrt{2}}$$

- *Isometry assumption 2:*

$$\|(\Phi \mathbf{U})^\top \Phi \mathbf{r_b}\|_2^2 \leq \frac{\epsilon \|\mathbf{r_b}\|_2^2}{2}$$

Then, we have the following bound for $\boldsymbol{\theta}$, from Lemma 6.

$$\|\boldsymbol{\theta}\|_2^2 \leq \epsilon \|\mathbf{r_b}\|_2^2 \tag{9}$$

Least Squares Regression

Proof: Because $\boldsymbol{\theta}$ is the solution of the optimization problem $\min_{\mathbf{e}\in\mathbb{R}^d} \|\Phi\mathbf{U}\mathbf{e} - \Phi\mathbf{r_b}\|_2^2$, it should satisfy the normal equation:

$$(\Phi\mathbf{U})^\top \Phi\mathbf{U}\boldsymbol{\theta} = (\Phi\mathbf{U})^\top \Phi\mathbf{r_b}$$

Hence, under our assmption, we have

$$\frac{\|\boldsymbol{\theta}\|_2^2}{2} \leq \|(\Phi\mathbf{U})^\top \Phi\mathbf{U}\boldsymbol{\theta}\|_2^2 = \|(\Phi\mathbf{U})^\top \Phi\mathbf{r_b}\|_2^2 \leq \frac{\epsilon\|\mathbf{r_b}\|_2^2}{2}$$

Least Squares Regression

Proof of Theorem, first claim:

$$\begin{aligned}
\|\mathbf{r}_{\mathbf{b}_s}\|_2^2 &= \|\mathbf{y} - \mathbf{X}\mathbf{b} + \mathbf{X}\mathbf{b} - \mathbf{X}\mathbf{b}_s\|_2^2 \\
&= \|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2 + \|\mathbf{X}\mathbf{b} - \mathbf{X}\mathbf{b}_s\|_2^2 \\
&= \|\mathbf{r}_{\mathbf{b}}\|_2^2 + \| - \mathbf{U}\boldsymbol{\theta}\|_2^2 \\
&\leq \|\mathbf{r}_{\mathbf{b}}\|_2^2 + \epsilon\|\mathbf{r}_{\mathbf{b}}\|_2^2 && [\textit{by Equation } 9]. \\
&\leq (1+\epsilon)\|\mathbf{r}_{\mathbf{b}}\|_2^2.
\end{aligned}$$

Least Squares Regression

Remark 1

Any sketching matrix Φ that satisfies the two issometry assumptions is called as a "good" sketching matrix. The numbers in the assumptions are fixed just arbitrarily, i.e., instead of $\sigma^2_{\min}(\Phi \mathbf{U}) \geq \frac{1}{\sqrt{2}}$, we can have $\sigma^2_{\min}(\Phi \mathbf{U}) > 0$ and things will still work.

Least Squares Regression

Remark 2

The assumptions are called as **isometry** assumptions because of a reason. A matrix or a transformation is called as isometric, if it preserves norms. That is, $\|\mathbf{Ae}\|_2^2 \approx \|\mathbf{e}\|_2^2$ for all $\mathbf{e}$ or **most** vectors $\mathbf{e}$. Sketching is exactly doing something like this, while reducing the dimension as well. The two assumptions are basically a way to characterize when the sketching matrix is isometric.

Least Squares Regression

Remark 3

The Sketching matrix $\Phi = \mathbf{S}^T\mathbf{HD}$ constructed in the previous section, satisfies the properties of a good sketching matrix **with high probability** (as the matrix is a random matrix). If you are curious how to show this fact, you may refer to [Mah16].

References

- Jacek Kuczyński and Henryk Woźniakowski, *Estimating the largest eigenvalue by the power and lanczos algorithms with a random start*, SIAM journal on matrix analysis and applications **13** (1992), no. 4, 1094–1122.
- Michael W Mahoney, *Lecture notes on randomized linear algebra*, arXiv preprint arXiv:1608.04481 (2016).
- Joel Tropp, *An introduction to matrix concentration inequalities*, Foundations and Trends® in Machine Learning **8** (2015), no. 1-2, 1–230.