Algorithms for Big Data (VI)

Chihao Zhang

Shanghai Jiao Tong University

Oct. 25, 2019

Review

Review

We learnt AMS algorithm to estimate $\|\mathbf{f}\|_k^k$ for $k \geq 2$ using $O\left(kn^{1-1/k}(\log m + \log n)\right)$ bits.

Review

We learnt AMS algorithm to estimate $\|\mathbf{f}\|_k^k$ for $k \geq 2$ using $O\left(kn^{1-1/k}(\log m + \log n)\right)$ bits.

An ad-hoc algorithm for $\|\mathbf{f}\|_2^2$ costs $O\left(\log m + \log n\right)$.

Review

We learnt AMS algorithm to estimate $\|\mathbf{f}\|_k^k$ for $k \geq 2$ using $O\left(kn^{1-1/k}(\log m + \log n)\right)$ bits.

An ad-hoc algorithm for $\|\mathbf{f}\|_2^2$ costs $O(\log m + \log n)$.

- Pick $h : [n] \to \{-1, 1\}$ from a 4-universal family;
- On input (j, Δ), $x \leftarrow x + \Delta \cdot h(j)$;
- Output x^2.

An Algebraic View

An Algebraic View

It is instructive to view the Tug-of-War algorithm from linear algebra.

An Algebraic View

It is instructive to view the Tug-of-War algorithm from linear algebra.

Assume that we run the algorithm k times (to apply the averaging trick), each time with function h_i.

An Algebraic View

It is instructive to view the Tug-of-War algorithm from linear algebra.

Assume that we run the algorithm k times (to apply the averaging trick), each time with function h_i.

Consider the matrix $A = (a_{ij})_{i \in [k], j \in [n]}$ where $a_{ij} = h_i(j)$.

An Algebraic View

It is instructive to view the Tug-of-War algorithm from linear algebra.

Assume that we run the algorithm k times (to apply the averaging trick), each time with function h_i.

Consider the matrix $A = (a_{ij})_{i\in[k],j\in[n]}$ where $a_{ij} = h_i(j)$.

Let $\mathbf{x} = A\mathbf{f}$, we know that $\mathbf{E}\left[x_i^2\right] = \|f\|_2^2$. Our algorithm outputs $\frac{\sum_{i=1}^k x_i^2}{k} = \frac{\|\mathbf{x}\|_2^2}{k}$.

An Algebraic View

It is instructive to view the Tug-of-War algorithm from linear algebra.

Assume that we run the algorithm k times (to apply the averaging trick), each time with function h_i.

Consider the matrix $A = (a_{ij})_{i\in[k],j\in[n]}$ where $a_{ij} = h_i(j)$.

Let $\mathbf{x} = A\mathbf{f}$, we know that $\mathbf{E}\left[x_i^2\right] = \|f\|_2^2$. Our algorithm outputs $\frac{\sum_{i=1}^k x_i^2}{k} = \frac{\|\mathbf{x}\|_2^2}{k}$.

The 2-norm of the vector $\frac{\mathbf{x}}{\sqrt{k}}$ is close to that of $\mathbf{f}$!

Dimension Reduction

Dimension Reduction

Suppose $k \ll n$, what the matrix A does is to map a vector in $\mathbb{R}^n$ to a vector in $\mathbb{R}^k$ without changing its norm much.

This operation is often referred as dimension reduction or metric embedding.

Dimension Reduction

Suppose $k \ll n$, what the matrix A does is to map a vector in $\mathbb{R}^n$ to a vector in $\mathbb{R}^k$ without changing its norm much.

This operation is often referred as dimension reduction or metric embedding.

The algorithm we met is similar to one important dimension reduction technique - Johnson-Lindenstrauss transformation.

Johnson-Lindenstrauss transformation

Johnson-Lindenstrauss transformation

Theorem

For any $0 < \varepsilon < \frac{1}{2}$ and any positive integer m, consider a set of m points $S \subseteq \mathbb{R}^n$. There exists an matrix $A \in \mathbb{R}^{k \times n}$ where $k = O\left(\varepsilon^{-2} \log m\right)$ satisfying

$$\forall \mathbf{x}, \mathbf{y} \in S, \quad (1-\varepsilon)\|\mathbf{x}-\mathbf{y}\| \leq \|A\mathbf{x} - A\mathbf{y}\| \leq (1+\varepsilon)\|\mathbf{x}-\mathbf{y}\|.$$

Johnson-Lindenstrauss transformation

Theorem

For any $0 < \varepsilon < \frac{1}{2}$ and any positive integer m, consider a set of m points $S \subseteq \mathbb{R}^n$. There exists an matrix $A \in \mathbb{R}^{k \times n}$ where $k = O\left(\varepsilon^{-2} \log m\right)$ satisfying

$$\forall \mathbf{x}, \mathbf{y} \in S, \quad (1-\varepsilon)\|\mathbf{x}-\mathbf{y}\| \leq \|A\mathbf{x}-A\mathbf{y}\| \leq (1+\varepsilon)\|\mathbf{x}-\mathbf{y}\|.$$

We construct A by drawing each of its entry from $\mathcal{N}(0, \frac{1}{k})$ independently.

Gaussian Distribution

Gaussian Distribution

Recall the density function of a variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Gaussian Distribution

Recall the density function of a variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The distribution function is

$$F_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \, \mathrm{d}\, t.$$

Gaussian Distribution

Recall the density function of a variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The distribution function is

$$F_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \,\mathrm{d}\,t.$$

Assume $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then

$$aX_1 + bX_2 \sim \mathcal{N}(a\mu_1 + b\mu_2, a^2\sigma_1^2 + b^2\sigma_2^2).$$

Proof of JL

Proof of JL

The statement is equivalent to

$$1 - \varepsilon \leq \frac{\|A(\mathbf{x} - \mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|} \leq 1 + \varepsilon.$$

Proof of JL

The statement is equivalent to

$$1 - \varepsilon \le \frac{\|A(\mathbf{x} - \mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|} \le 1 + \varepsilon.$$

We only need to show that for every unit length vector $\mathbf{f}$,

$$\mathbf{Pr}\left[\left|\|A\mathbf{f}\| - 1\right| > \varepsilon\right] \le 1 - \delta.$$

Proof of JL

The statement is equivalent to

$$1 - \varepsilon \le \frac{\|A(\mathbf{x} - \mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|} \le 1 + \varepsilon.$$

We only need to show that for every unit length vector $\mathbf{f}$,

$$\mathbf{Pr}\left[\left|\|A\mathbf{f}\| - 1\right| > \varepsilon\right] \le 1 - \delta.$$

Assume $\mathbf{x} = A\mathbf{f}$, then $x_i = \sum_{j \in [n]} a_{ij} \cdot f_j \sim \mathcal{N}(0, \frac{1}{k})$.

Proof of JL

The statement is equivalent to

$$1 - \varepsilon \le \frac{\|A(\mathbf{x} - \mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|} \le 1 + \varepsilon.$$

We only need to show that for every unit length vector $\mathbf{f}$,

$$\mathbf{Pr}\left[|\|A\mathbf{f}\| - 1| > \varepsilon\right] \le 1 - \delta.$$

Assume $\mathbf{x} = A\mathbf{f}$, then $x_i = \sum_{j \in [n]} a_{ij} \cdot f_j \sim \mathcal{N}(0, \frac{1}{k})$.

We need a concentration inequality for squared sum of Gaussians:

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{k} x_i^2 - 1\right| \ge \varepsilon\right] \le 1 - \delta.$$

Concentration

Concentration

Theorem

Assume $X_1, X_2, \ldots, X_k$ be i.i.d $\mathcal{N}(0,1)$, then for $0 < \varepsilon < 1$,

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{k} X_i^2 - k\right| \geq \varepsilon k\right] < 2e^{-\frac{\varepsilon^2 k}{8}}.$$

Concentration

Theorem

Assume $X_1, X_2, \ldots, X_k$ be i.i.d $\mathcal{N}(0,1)$, then for $0 < \varepsilon < 1$,

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{k} X_i^2 - k\right| \geq \varepsilon k\right] < 2e^{-\frac{\varepsilon^2 k}{8}}.$$

The proof is similar to the proof of the Chernoff bound we met before.

Estimate F_2 from JL

Estimate F_2 from JL

We can use JL to estimate F_2:

Estimate F_2 from JL

We can use JL to estimate F_2:

Algorithm JL Transformation

Init:

$Z_1, \ldots, Z_n$ from $\mathcal{N}(0,1)$.

$x \leftarrow 0$.

On Input (y, Δ)**:**

$x \leftarrow x + \Delta \cdot Z_y$

Output:

Output x^2.

Estimate F_2 from JL

We can use JL to estimate F_2:

Algorithm JL Transformation

Init:

$Z_1, \ldots, Z_n$ from $\mathcal{N}(0,1)$.

$x \leftarrow 0$.

On Input (y, Δ)**:**

$x \leftarrow x + \Delta \cdot Z_y$

Output:

Output x^2.

The algorithm is neither friendly to implement nor efficient, but it is inspiring.

Estimate F_2 from JL

We can use JL to estimate F_2:

Algorithm JL Transformation

Init:

$Z_1, \ldots, Z_n$ from $\mathcal{N}(0,1)$.

$x \leftarrow 0$.

On Input (y, Δ)**:**

$x \leftarrow x + \Delta \cdot Z_y$

Output:

Output x^2.

The algorithm is neither friendly to implement nor efficient, but it is inspiring.

The core property we used to prove its correctness is that $\sum_{j=1}^{n} Z_j \cdot f_j$ has the same distribution as $\|\mathbf{f}\|_2 Z$ where $Z \sim \mathcal{N}(0,1)$.

The property generalizes to $p < 2$.

The property generalizes to $p < 2$.

For some distribution $\mathcal{D}_p$, if $Z_j \sim \mathcal{D}_p$, then $\sum_j Z_j \cdot f_j$ has the same distribution as $\|\mathbf{f}\|_p Z$ where $Z \sim D_p$.

The property generalizes to $p < 2$.

For some distribution $\mathcal{D}_p$, if $Z_j \sim \mathcal{D}_p$, then $\sum_j Z_j \cdot f_j$ has the same distribution as $\|\mathbf{f}\|_p Z$ where $Z \sim D_p$.

The distribution is called p-stable.

The property generalizes to $p < 2$.

For some distribution $\mathcal{D}_p$, if $Z_j \sim \mathcal{D}_p$, then $\sum_j Z_j \cdot f_j$ has the same distribution as $\|\mathbf{f}\|_p Z$ where $Z \sim D_p$.

The distribution is called p-stable.

We can use them to estimate F_p. Many technical issue of the algorithm is beyond the scope of this course.

Graph Stream

Graph Stream

We have a graph with vertex set $[n]$, but its edges are unknown.

Graph Stream

We have a graph with vertex set $[n]$, but its edges are unknown.

The edge are given in a streaming fashion, namely each time we reveal an edge (u, v).

Graph Stream

We have a graph with vertex set $[n]$, but its edges are unknown.

The edge are given in a streaming fashion, namely each time we reveal an edge (u, v).

Can we compute graph properties using small bits of memories?

Graph Stream

We have a graph with vertex set $[n]$, but its edges are unknown.

The edge are given in a streaming fashion, namely each time we reveal an edge (u, v).

Can we compute graph properties using small bits of memories? Say in $O\left(n \cdot \text{poly}(\log n)\right)$.

Connectedness

Connectedness

A basic graph property is whether the graph is connected.

Connectedness

A basic graph property is whether the graph is connected.

We can maintain a spanning forest F of G:

Init:

$F \leftarrow \varnothing$,

$X \leftarrow 0$.

On Input (u, v)**:**

if $X = 0$ and $F \cup \{(u, v)\}$ has no cycle **then**

  $F \leftarrow F \cup \{(u, v)\}$;

  if $|F| = n - 1$ **then** $X \leftarrow 1$

  end if

end if

Output:

Output X.

Connectedness

A basic graph property is whether the graph is connected.

We can maintain a spanning forest F of G:

Init:
$F \leftarrow \varnothing$,
$X \leftarrow 0$.

On Input (u, v)**:**
if $X = 0$ and $F \cup \{(u, v)\}$ has no cycle **then**
  $F \leftarrow F \cup \{(u, v)\}$;
  if $|F| = n - 1$ **then** $X \leftarrow 1$
  end if
end if

Output:
Output X.

Bipartiteness

Bipartiteness

The following algorithm decides whether G is bipartite.

Bipartiteness

The following algorithm decides whether G is bipartite.

Init:

$F \leftarrow \emptyset$,

$X \leftarrow 1$.

On Input (u, v):

if $X = 1$ **then**

  if $F \cup \{(u, v)\}$ has no cycle **then**

    $F \leftarrow F \cup \{(u, v)\}$;

  else if $F \cup \{(u, v)\}$ has an odd cycle **then** $X \leftarrow 0$

  end if

end if

Output:

Output X.

Bipartiteness

The following algorithm decides whether G is bipartite.

Init:

$F \leftarrow \varnothing$,

$X \leftarrow 1$.

On Input (u, v)**:**

if $X = 1$ **then**

  if $F \cup \{(u, v)\}$ has no cycle **then**

    $F \leftarrow F \cup \{(u, v)\}$;

  else if $F \cup \{(u, v)\}$ has an odd cycle **then** $X \leftarrow 0$

  end if

end if

Output:

Output X.