Adversarially Robust Streaming Algorithms
Classic Streaming Algorithms

Modeled by updates to a large vector $x \in \mathbb{R}^n$

- At time $t = 1, 2, ..., $ receive update $(i_t, \Delta_t)$, causing change

  $$x_{i_t} \leftarrow x_{i_t} + \Delta_t$$

- If for all $t$, $\Delta_t \geq 0$, the stream is called insertion-only

- At any point, algorithm computes function $f(x)$ using small space

- E.g. $f(x) = F_2 = |x|_2^2 = \sum_i x_i^2$ or $f(x) = F_0 = |\{i \text{ such that } x_i \neq 0\}|$

- Algorithm stores a small sketch $S(x)$ of the data, much smaller than $n$ bits
Example: $F_2$ — Estimation

- Output $\overline{F_2}$ for which $(1 - \epsilon) \cdot F_2 \leq \overline{F_2} \leq (1 + \epsilon) \cdot F_2$ (recall $F_2 = \sum_i x_i^2$)

- Choose a random matrix $S \in \{-\epsilon, \epsilon\}^{1/\epsilon^2 \times n}$
  - Entries can be 4-wise independent

- Maintain $S \cdot x$ in the stream
  - Given an update $x_{it} \leftarrow x_{it} + \Delta_t$, set $S \cdot x \leftarrow S \cdot x + \Delta_t \cdot S_{*, it}$

- Use $|S \cdot x|_2^2$ to estimate $|x|_2^2$
  - $E_S[|S \cdot x|_2^2] = |x|_2^2$ and $\text{Var}_S[|S \cdot x|_2^2] = O(\epsilon^2 |x|_2^4)$

- $O(\frac{\log n}{\epsilon^2})$ bits of memory

$S = \epsilon \cdot \begin{bmatrix}
-1 & 1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & 1 & 1 \\
-1 & 1 & 1 & -1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$
Example: $F_0 \rightarrow$ Estimation in Insertion Streams

- Output $\widehat{F_0}$ with $(1 - \epsilon) \cdot F_0 \leq \widehat{F_0} \leq (1 + \epsilon) \cdot F_0$ (recall $F_0 = |\{i \text{ with } x_i \neq 0\}|$)

- Choose a hash function $h: \{1, 2, ..., n\} \rightarrow \{0, 1, 2, ..., M\}$, where $M = O(n^2)$
  - With good probability, no collisions

- Maintain the smallest $t = \frac{100}{\epsilon^2}$ hash values in the stream

- Output $Z = tM/v$, where $v$ is the $t$-th smallest hash value
  - Smallest hash value about $\frac{M}{F_0}$, so $v$ should be about $\frac{tM}{F_0}$

- $O(\epsilon^{-2} \cdot \log n)$ bits of memory. Can improve to $O(\epsilon^{-2} + \log n)$ bits
Tracking Algorithms

• $x^{(t)} :=$ the stream vector after updates 1, 2, ... $t$

• Algorithm must output $R^{(t)}$ so

$$R^{(t)} = (1 \pm \epsilon)f(x^{(t)})$$
Tracking Algorithms

• $x^{(t)} := \text{the stream vector after updates } 1, 2, \ldots, t$
• Algorithm must output $R^{(t)}$ so

$$R^{(t)} = (1 \pm \epsilon)f(x^{(t)})$$
Tracking Algorithms

- $x^{(t)} := \text{the stream vector after updates } 1, 2, \ldots, t$
- Algorithm must output $R^{(t)}$ so

$$R^{(t)} = (1 \pm \epsilon)f(x^{(t)})$$
Tracking Algorithms

• $x^{(t)} :=$ the stream vector after updates $1, 2, ... t$
• Algorithm must output $R^{(t)}$ so

$$R^{(t)} = (1 \pm \epsilon)f(x^{(t)})$$

$(i_{t+1}, \Delta_{t+1})$
Tracking Algorithms

• $x^{(t)} :=$ the stream vector after updates $1,2, \ldots, t$

• Algorithm must output $R^{(t)}$ so

$$R^{(t)} = (1 \pm \epsilon)f(x^{(t)})$$
Adversarial Streams

• **Classic streams:** Data is fixed before algorithm starts: future data does not depend on outputs $R^{(t)}$!

  • Future data often depends on past decisions: no known guarantees for streaming algorithms in this case!

• **Adversarial Streams:** Adversary controls stream updates: sees $R^{(t)}$, then gets to choose $(i_{t+1}, \Delta_{t+1})$. 
Adversarial Streams

Modeled by 2-player game.

$(i_1, \Delta_1)$
Adversarial Streams

Modeled by 2-player game.
Adversarial Streams

Modeled by 2-player game.
Adversarial Streams

Modeled by 2-player game.
Adversarial Streams

Modeled by 2-player game.
Adversarial Streams

Modeled by 2-player game.
Adversarial Streams

Modeled by 2-player game.

(i_3, \Delta_3)

R^{(1)}, R^{(2)}
Adversarial Streams

Goal of Adversary: Make Algorithm fail to output an $\epsilon$-approximation:

• Adversary wants: $R^{(t)} \neq (1 \pm \epsilon)f(x^{(t)})$ at some time $t$

• Adversary has unbounded computational power, knows entire history of outputs $R^{(1)}, R^{(2)}, \ldots, R^{(t-1)}$ at time $t$

• Deterministic algorithms are adversarially robust, however most streaming algorithms provably must be randomized
Classic Streaming Algorithms Not Robust!

**Theorem:** the classic **AMS Sketch** (Alon, Matias, Szegedy ‘96) for estimating $F_2$ is not adversarially robust!

- Even in insertion only streams, meaning $\Delta_t \geq 0$ for all $t$. 
Classic Streaming Algorithms Not Robust!

**Theorem:** the classic **AMS Sketch** (Alon, Matias, Szegedy ‘96) for estimating $F_2$ is not adversarially robust!

- Even in insertion only streams, meaning $\Delta_t \geq 0$ for all $t$.

We need new algorithms!
Give two generic methods to transform any streaming algorithm $A$ into an adversarially robust algorithm $A'$, with a mild space overhead:
Generic Transformations

Give two **generic** methods to **transform** any streaming algorithm $\mathcal{A}$ into an *adversarially robust* algorithm $\mathcal{A}'$, using small space overhead:

**Sketch Switching**

**Computation Paths**
Generic Transformations

**Sketch Switching**

Useful for exploiting algorithms $\mathcal{A}$ which provide tracking better than one shot + a union bound over all time steps $t$.

**Computation Paths**

Useful for exploiting algorithms $\mathcal{A}$ with better dependence on failure probability $\delta$ than multiplicative $O(\log 1/\delta)$.
Flip Number

**Definition** (informal): For a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$, define the $\epsilon$-flip number $\lambda_\epsilon(f)$ to be the maximum number of times $f(x^{(t)})$ can change by a factor of $(1 + \epsilon)$ after poly$(n)$ updates.
Flip Number

**Definition** (informal): For a function $f: \mathbb{R}^n \to \mathbb{R}$, define the $\epsilon$-flip number $\lambda_\epsilon(f)$ to be the maximum number of times $f(x^{(t)})$ can change by a factor of $(1 + \epsilon)$ after $\text{poly}(n)$ updates.

**Example:** $f(x) = \|x\|_p^p = \sum_i |x_i|^p$, then for insertion only streams

1. $\|x^{(0)}\|_p^p = 0$
2. $\|x^{(\text{poly}(n))}\|_p^p \leq \text{poly}(n)$

So $\lambda_\epsilon(f) = \log_{(1+\epsilon)}(\text{poly}(n)) = O\left(\frac{1}{\epsilon \log n}\right)$
**Sketch Switching**

1. Keep multiple ($\lambda_\epsilon (f)$ many) independent sketches concurrently.

2. Only use output of one sketch $S_i$ at a time.

3. Once estimate $R^{(t)}$ of $S_i$ changes by $(1 + \epsilon)$, info about $S_i$ is leaked, throw $S_i$ away!
**Sketch Switching**

1. Keep multiple ($\lambda_\varepsilon(f)$ many) independent sketches concurrently.
2. Only use outputs of one sketch $S_i$ at a time.
3. Once estimate $R^{(t)}$ of $S_i$ changes by $(1 + \varepsilon)$, info about $S_i$ is leaked, throw $S_i$ away!

**Computation Paths**

Streaming algorithms can be made robust by setting failure probability $\delta$ small enough:

$$\delta' = \delta \cdot n^{-O(\lambda_\varepsilon(f))}$$

1. Only need to change output $\lambda_\varepsilon(f)$ times!
2. Stream is length $\text{poly}(n)$, and output is $O(\log n)$ bits, so only $n^{O(\lambda_\varepsilon(f))}$ possible “computation paths” between algorithm and adversary.
3. Setting $\delta'$ small enough, can union bound over all of them!
**Theorem** (informal): For $f: \mathbb{R}^n \to \mathbb{R}$, let $\mathcal{A}$ be any algorithm which $\epsilon$-tracks $f(x^{(t)})$. Then there is an adversarially robust algorithm $\mathcal{A}'$ for $\epsilon$-tracking $f(x^{(t)})$ using space $O(\lambda_\epsilon(f) \cdot \text{Space}(\mathcal{A}))$.

**Theorem** (informal): Let $\mathcal{A}$ be any algorithm which $\epsilon$-tracks $f(x^{(t)})$ with probability $1 - \delta$ using space $\text{Space}(\mathcal{A}, \delta)$. Then there is an adversarially robust algorithm $\mathcal{A}'$ that uses $\text{Space}(\mathcal{A}, \delta')$ where

$$\delta' = \delta \cdot n^{-O(\lambda_\epsilon(f))}$$
Sketch Switching

**Theorem** (informal): For $f: \mathbb{R}^n \rightarrow \mathbb{R}$, let $\mathcal{A}$ be any algorithm which $\epsilon$-tracks $f(x^{(t)})$. Then there is an adversarially robust algorithm $\mathcal{A}'$ for $\epsilon$-tracking $f(x^{(t)})$ using space $O(\lambda_\epsilon(f) \cdot \text{Space}(\mathcal{A}))$. 
Application of Sketch Switching

**Theorem** (Blasiok SODA ‘18): There is a streaming algorithm that $\varepsilon$-tracks the number of distinct elements in an insertion only stream, defined by $\|x(t)\|_0 = |\{ i : x_i \neq 0 \}|$, using space $O \left( \frac{\log \log n}{\varepsilon^2} + \log n \right)$.
Application of Sketch Switching

**Theorem (Blasiok SODA ‘18):** There is a streaming algorithm that $\epsilon$-tracks the number of **distinct elements** in an insertion only stream, defined by $\|x^{(t)}\|_0 = |\{i : x_i \neq 0\}|$, using space $O\left(\frac{\log \log n}{\epsilon^2} + \log n\right)$.

**Theorem:** There is an adversarially robust streaming algorithm for $\epsilon$-tracking the number of **distinct elements** using space:

$$O\left(\frac{\log n}{\epsilon} \left(\frac{\log \log n}{\epsilon^2} + \log n\right)\right)^*$$

In this case can improve the leading $\log n$ factor to $\log(\frac{1}{\epsilon})$.
Sketch Switching

High level idea:
• Adversary wants to learn about your sketch and randomness to fool it
• We carefully reveal information about our sketches
• As soon as we reveal any new information, immediately make this information irrelevant
Sketch Switching:

1. Create $O(\lambda \frac{\epsilon}{12} (f))$ independent sketches $S_1, ..., S_k$, each providing a $(1 \pm \frac{\epsilon}{10})$-approximation. Set $i = 1$ and $R^{(0)} = 0$

2. At time $t$, if Estimate $(S_i) \notin \left(1 \pm \frac{\epsilon}{3}\right) R^{(t-1)}$
   1. Set $R^{(t)} = \text{Estimate}(S_i)$ and throw out $S_i$
   2. $i \leftarrow i + 1$

Otherwise set $R^{(t)} = R^{(t-1)}$
Sketch Switching Proof

• Can assume adversary is deterministic by averaging

• This **fixes** the part of the stream the adversary gives $S^i$ after $S^{i-1}$ returned an answer Out
  • The stream does not depend on $S^i$ though it may depend on $S^1, S^2, ..., S^{i-1}$
  • So, $S^i$ is correct at all positions in new stream

• $S^i$ outputs old value Out until its value $Out' \notin \left[\left(1 - \frac{\epsilon}{3}\right)\text{Out}, \left(1 + \frac{\epsilon}{3}\right)\text{Out}\right]$, at which point we switch to $S^{i+1}$

• Might worry you learn something about $S^i$ until it outputs $Out'$, and you do, but $S^i$ is correct on whatever fixed stream you choose until $S^i$ outputs $Out'$
Sketch Switching Proof

• If $S^i$ provides a $(1 \pm \frac{\epsilon}{10})$-approximation, then if $\text{Out}' \notin \left[\left(1 - \frac{\epsilon}{3}\right)\text{Out}, \left(1 + \frac{\epsilon}{3}\right)\text{Out}\right]$, necessarily $f$ has changed by a $(1 \pm \frac{\epsilon}{12})$ factor

• Number of sketches we need is bounded by $\lambda_{\epsilon}(f)$

• Conversely, if $f$ changes by a $(1 \pm \epsilon)$ factor, necessarily $\text{Out}' \notin \left[\left(1 - \frac{\epsilon}{3}\right)\text{Out}, \left(1 + \frac{\epsilon}{3}\right)\text{Out}\right]$

• So we are correct at all times
Computation Paths

**Theorem (informal):** For $f: \mathbb{R}^n \rightarrow \mathbb{R}$, let $\mathcal{A}$ be any algorithm which $\epsilon$-tracks $f(x(t))$ with probability $1 - \delta$ using space $L(\epsilon, \delta)$. Then there is a *robust* algorithm $\mathcal{A}'$ for using space $L(\epsilon/10, \delta')$, where

$$\delta' = \delta \cdot n^{-O(\lambda_{\epsilon/12}(f))}$$

- Streaming algorithms can be made robust by setting failure probability $\delta$ to be small!
Computation Paths: High Level Proof

Streaming algorithms can be made robust by setting failure probability to be

$$\delta' = \delta \cdot n^{-O(\lambda_{\epsilon/12}(f))}$$

1. Only need to change the output $\lambda_{\epsilon/12}(f)$ times

2. Stream is length poly(n), and output is $O(\log n)$ bits, so $n^{O(\lambda_{\epsilon/12}(f))}$ possible streams a deterministic adversary can create

3. Setting $\delta'$ small enough, can union bound over all of them
Results of [BJWY], instantiating $\lambda_\varepsilon(f) = O(\varepsilon^{-1} \log n)$

<table>
<thead>
<tr>
<th>Problem</th>
<th>Non-Adversarial</th>
<th>Adversarial ([BJWY])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Elements ($F_0$)</td>
<td>$\tilde{O}(\varepsilon^{-2} + \log n)$</td>
<td>$\tilde{O}(\varepsilon^{-3} + \log n)$</td>
</tr>
<tr>
<td>$F_p$ estimation, $p \in (0,2] \setminus {1}$</td>
<td>$\tilde{O}(\varepsilon^{-2} \log n)$</td>
<td>$\tilde{O}(\varepsilon^{-3} \log n)$</td>
</tr>
<tr>
<td>$F_p$ estimation, $p &gt; 2$</td>
<td>$O\left(n^{1-\frac{2}{p}}(\varepsilon^{-3} \log^2 n + \varepsilon ^{-\frac{6}{p}} \log^{4+1} n)\right)$</td>
<td>Same when $\delta = O(n^{-\log \frac{\log n}{\varepsilon}})$</td>
</tr>
<tr>
<td>Heavy Hitters</td>
<td>$O(\varepsilon^{-2} \log^2 n)$</td>
<td>$\tilde{O}(\varepsilon^{-3} \log^2 n)$</td>
</tr>
<tr>
<td>Entropy Estimation</td>
<td>$O(\varepsilon^{-2} \log^3 n)$</td>
<td>$O(\varepsilon^{-5} \log^6 n)$</td>
</tr>
</tbody>
</table>

For adversaries with **bounded computation + Cryptographic Assumptions**, can improve some of above:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Adversarial ([BJWY])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Elements ($F_0$)</td>
<td>$\tilde{O}(\varepsilon^{-2} + \log n)$</td>
</tr>
<tr>
<td></td>
<td>optimal even with no adversary</td>
</tr>
<tr>
<td>Entropy Estimation</td>
<td>$O(\varepsilon^{-5} \log^4 n)$</td>
</tr>
</tbody>
</table>
Polynomially Bounded Adversaries

• Recall non-robust $F_0$-estimation algorithm:
  
  • Choose a hash function $h: \{1, 2, \ldots, n\} \rightarrow \{0, 1, 2, \ldots, M\}$, where $M = O(n^2)$
  
  • Maintain smallest $t = \frac{100}{\epsilon^2}$ values $h(i)$ found when processing stream
  
  • State of the algorithm is exactly the same if you insert the same item twice
    • Breaking this algorithm requires breaking the hash function $h$
  
  • Assumption: for any $c > 0$ there is a $d > 0$ and a family of $n^d$ hash functions that can be evaluated in $O(\log n)$ memory such that any $n^c$-time Adversary cannot break this
    • Exponentially secure pseudorandom function (in practice, AES or SHA256)
Improvements

1. [Hassidim, Kaplan, Mansour, Matias, Stemmer]
   1. Use differential privacy

2. Improve the [BJWY] bounds of $\tilde{O}(\epsilon^{-3} \log n)$ to $\tilde{O}(\epsilon^{-2.5} \log^4 n)$ for $F_0$, $F_2$, and many other streaming tasks

2. [W, Zhou]
   1. Introduce “Difference Estimators”

2. Improve the [BJWY] bounds of $\tilde{O}(\epsilon^{-3} \log n)$ and the $\tilde{O}(\epsilon^{-2.5} \log^4 n)$ bounds above to $\tilde{O}(\epsilon^{-2} \log n)$ for $F_0$, $F_2$, and many other streaming tasks

3. Non-robust algorithms for these problems require $\Omega(\epsilon^{-2})$ bits, so our memory is optimal in $\epsilon$ (and often matches non-robust log n factors)
Difference Estimators

• Do we really need to switch our sketch whenever the output changes by 1+ε?

• Maybe? Unclear what the adversary is learning.

• If the last output Out was a $(1 \pm \epsilon)$-approximation to function value $f(x)$, and $f(x)$ changes to $f(x')$ with $f(x') \in (1 \pm O(\epsilon))f(x)$, do we need a brand new $1 \pm \epsilon$ approximation to $f(x')$?

• Seems wasteful. We've fixed Out - maybe we can use Out for something?

• Difference Estimator: approximate $f(x') - f(x)$ up to an $O(1)$-factor, and add it to Out!
Difference Estimators

• Need to approximate \( f(x') - f(x) \) up to additive error \( \epsilon f(x) \) given that \( f(x') - f(x) = O(\epsilon) f(x) \)

  • Can’t afford to approximate each of \( f(x') \) and \( f(x) \) up to relative \( 1 \pm \epsilon \)

  • Approximating each of \( f(x') \) and \( f(x) \) up to relative \( O(1) \) error won’t give \( O(\epsilon) f(x) \) additive error

• Design the first difference estimators for streams!
  • Example: \( |x'|^2 - |x|^2 = |x' - x|^2 + 2 < x' - x, x > \)
  • Approximate terms up to \( O(\epsilon)|x|^2 \) error – uses \( |x' - x|^2 = O(\epsilon|x|^2) \)
  • Only need \( 1/\epsilon \) memory to do this
Sketch-Stitching and Granularity Changing

• Suppose \( x \) is the current underlying vector

• If \( x \) grows to \( x' \) with \( f(x') \geq 2f(x) \), \( x \) must first grow to \( x^1 \) with \( f(x^1) = (1 + \epsilon)f(x) \)

• Approximate the difference \( f(x^1) - f(x) \) up to \( C/\log(\frac{1}{\epsilon}) \)-relative error for constant \( C > 0 \)

• Then \( x \) must grow to a vector \( x^2 \) where \( f(x^2) = (1 + 2\epsilon)f(x) \)
  • Approximate the difference \( f(x^2) - f(x) \) up to \( C/(2 \log(\frac{1}{\epsilon})) \)-relative error
  • Important not to use \( [f(x^1) - f(x)] + [f(x^2) - f(x_1)] \) here – errors would grow too fast

• Then \( x \) must grow to a vector \( x^3 \) where \( f(x^3) = (1 + 3\epsilon)f(x) \)
  • Approximate the difference \( f(x^2) - f(x) \) up to \( C/(2 \log(\frac{1}{\epsilon})) \)-relative error
  • Approximate the difference \( f(x^3) - f(x^2) \) up to \( C/\log(\frac{1}{\epsilon}) \)-relative error

• Additive errors add to \( O(\epsilon)f(x) \), using \( O(\log(\frac{1}{\epsilon})) \) differences in binary representation
Achieving Adversarial Robustness

• For robustness for $F_2$, sketch-switch in each of $\log 1/\epsilon$ levels in a binary tree

  • Top level uses memory $\frac{1}{\epsilon^2}$ but only need to sketch-switch $O(\log n)$ times

  • Bottom level uses memory $\frac{1}{\epsilon}$ but needs to sketch-switch $\frac{1}{\epsilon}$ times

• Overall memory bound is a sum over levels
Further Work / Open Questions

• Tight bounds in terms of flip number [Kaplan, Mansour, Nissim, Stemmer]

• Improvements for small stream length [Ben-Eliezer, Eden, Onak]

• For streams with negative updates, can one prove strong lower bounds?

• Other uses of cryptography for data streams?