

CSCSE 689: Special Topics in Modern Algorithms for Data Science

Lecture 27

Samson Zhou

Presentation Schedule

- **November 27:** Chunkai, Jung, Galaxy AI
- **November 29:** STMI, Anmol, Jason
- **December 1:** Bokun, Ayesha, Dawei, Lipai

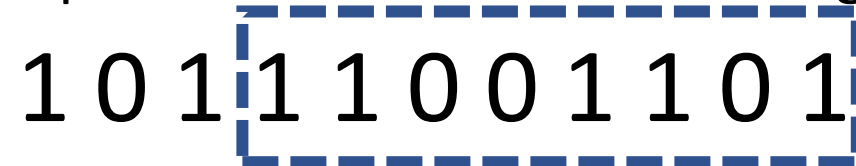
Last Time: Linear Sketch

- Suppose stream S induces a frequency vector f
- Algorithm framework:
 - Generate a random matrix A and maintain $A \cdot f$
 - Apply a post-processing function $g(A \cdot f)$ as the output
- What algorithms have we discussed that fit this framework?

Last Time: Sliding Window Model

- **Input:** Elements of an underlying data set S , which arrives sequentially
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S
- **Sliding Window:** “Only the m most recent updates form the underlying data set S ”
 - Emphasizes recent interactions, appropriate for time sensitive settings

1 0 1 1 1 0 0 1 1 0 1

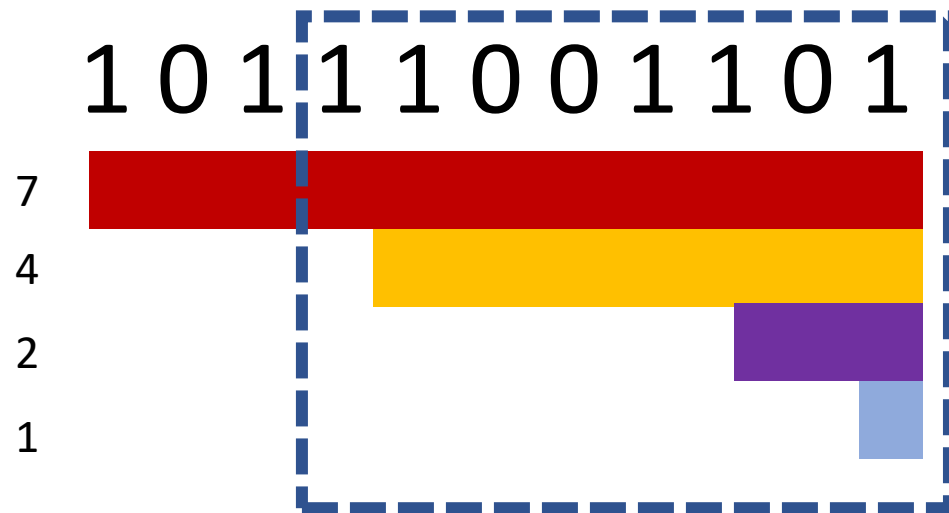


Last Time: Smooth Histogram

- Converts a streaming algorithm for a smooth function into a sliding window algorithm

Last Time: Smooth Histogram

- Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- Each time there are three instances that report “close” values, delete the middle one
- Use different checkpoints to “sandwich” the sliding window

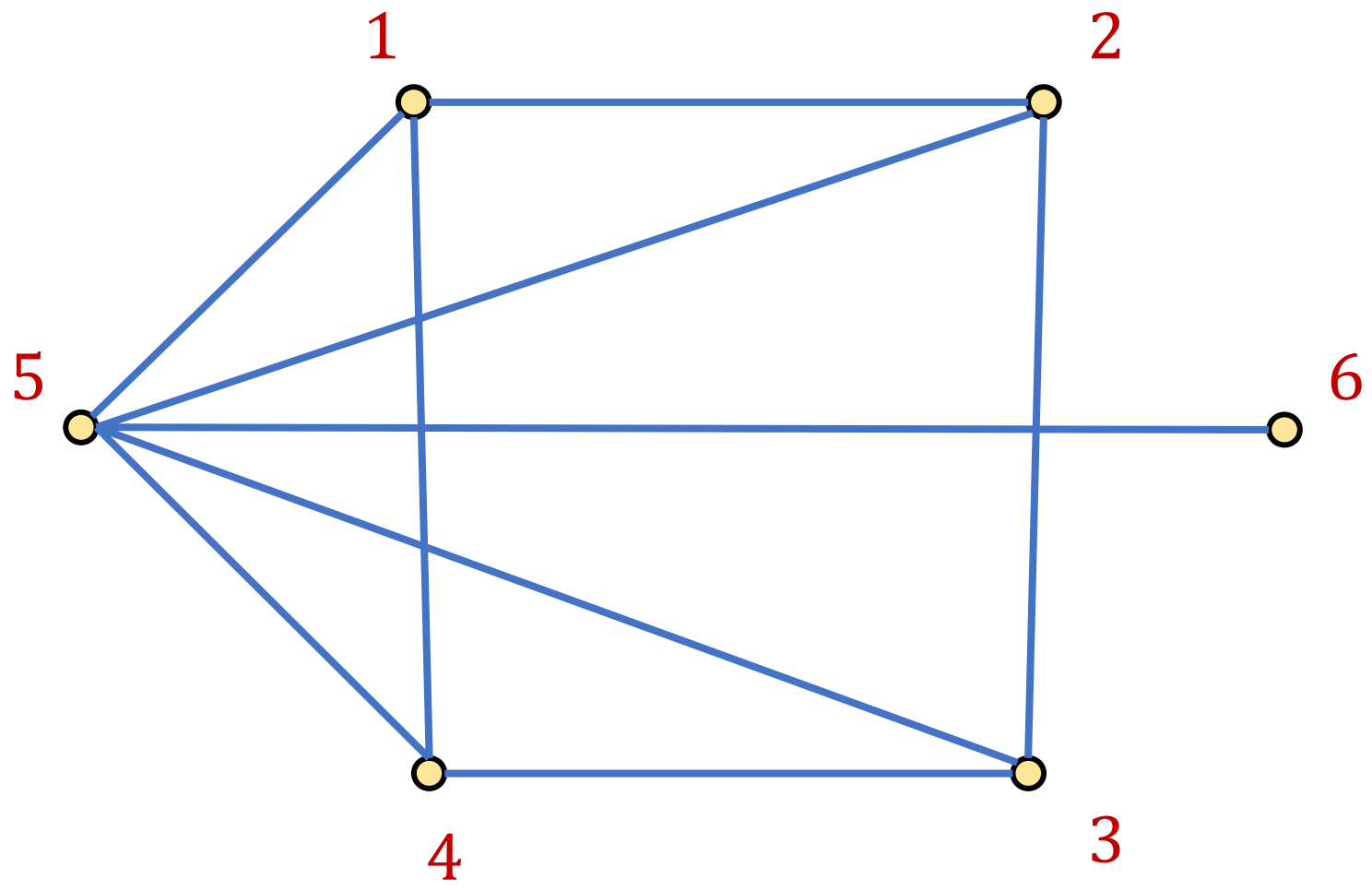


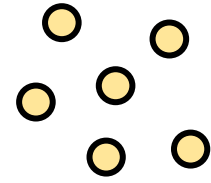
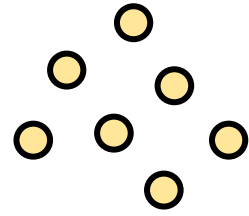
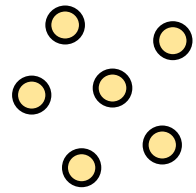
- **Example:** Number of ones in sliding window (2-approximation)
- Number of ones in sliding window is at least 4 and at most 7
- 4 is a good approximation

Streaming Model

- **Input:** Elements of an underlying data set S , which arrives sequentially
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S

47 72 81 10 14 33 51 29 54 9 36 46 10





Streaming Model

- We inherently assume the input is fixed in advance and thus *independent* of the algorithm
- What if we need to interact with the algorithm multiple times?

Case Study #1

- Suppose we run the same algorithm on multiple datasets. Do we know how to handle this?

Case Study #1

- Suppose we run the same algorithm on multiple datasets. Do we know how to handle this? **YES (union bound)**

Case Study #2

- Suppose I have a batch of queries for a randomized database algorithm. Do we know how to handle this?

Case Study #2

- Suppose I have a batch of queries for a randomized database algorithm. Do we know how to handle this? **YES, if the batch of queries is fixed in advance**

Case Study #3

- Suppose I ask a sequence of queries for a randomized database algorithm. Do we know how to handle this?

Case Study #3

- Suppose I ask a sequence of queries for a randomized database algorithm. Do we know how to handle this? **YES, if the batch of queries is fixed in advance, NO if each query can depend on the answer to previous queries**
- In the latter case, future inputs may not be independent of the algorithm's randomness

Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S



1

1



Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S



10

1



Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S



101

2



Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S



1010

3



Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S



1010

3



Adversarially Robust Streaming

- **Input:** Elements of an underlying data set S , which arrives sequentially and *adversarially*
- **Output:** Evaluation (or approximation) of a given function
- **Goal:** Use space *sublinear* in the size m of the input S

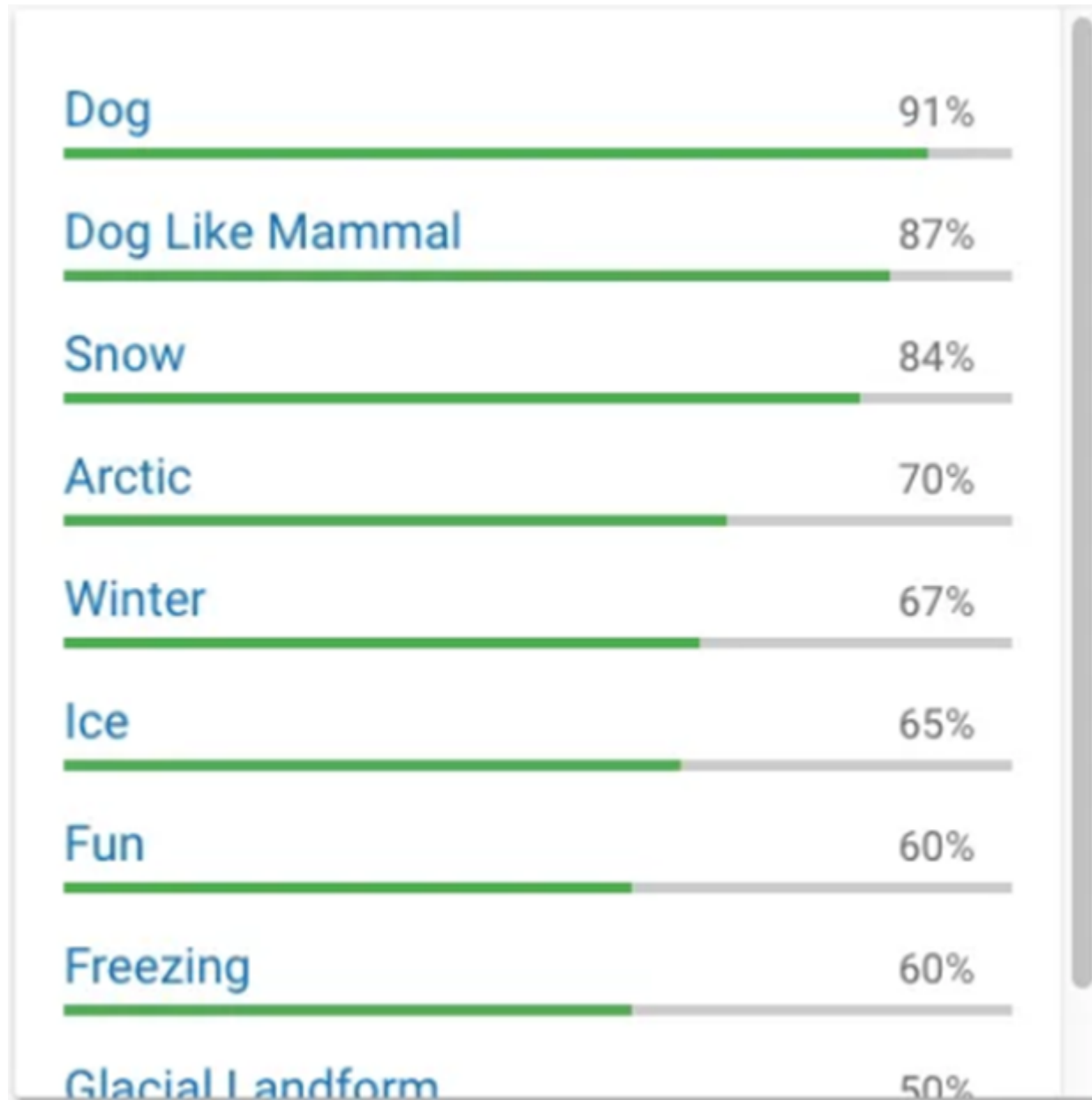
- **Adversarially Robust:** “Future queries may depend on previous queries”
- **Motivation:** Database queries, adversarial ML

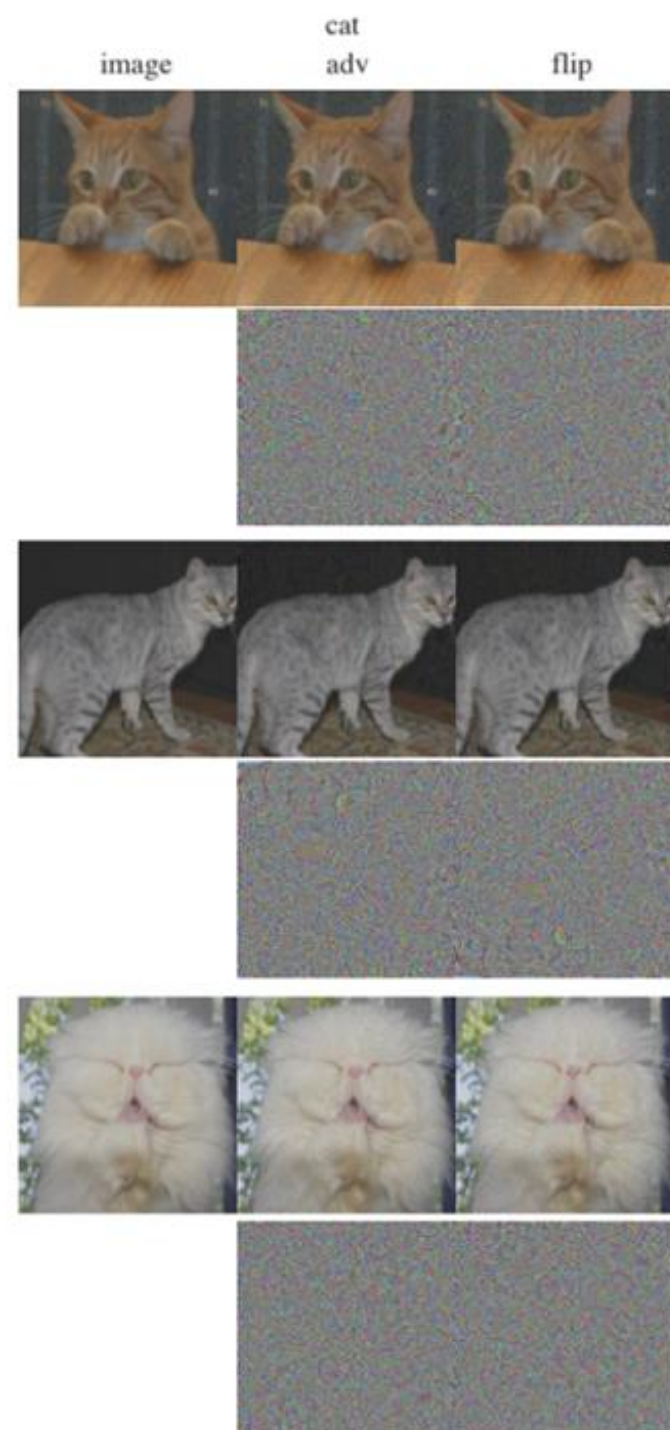
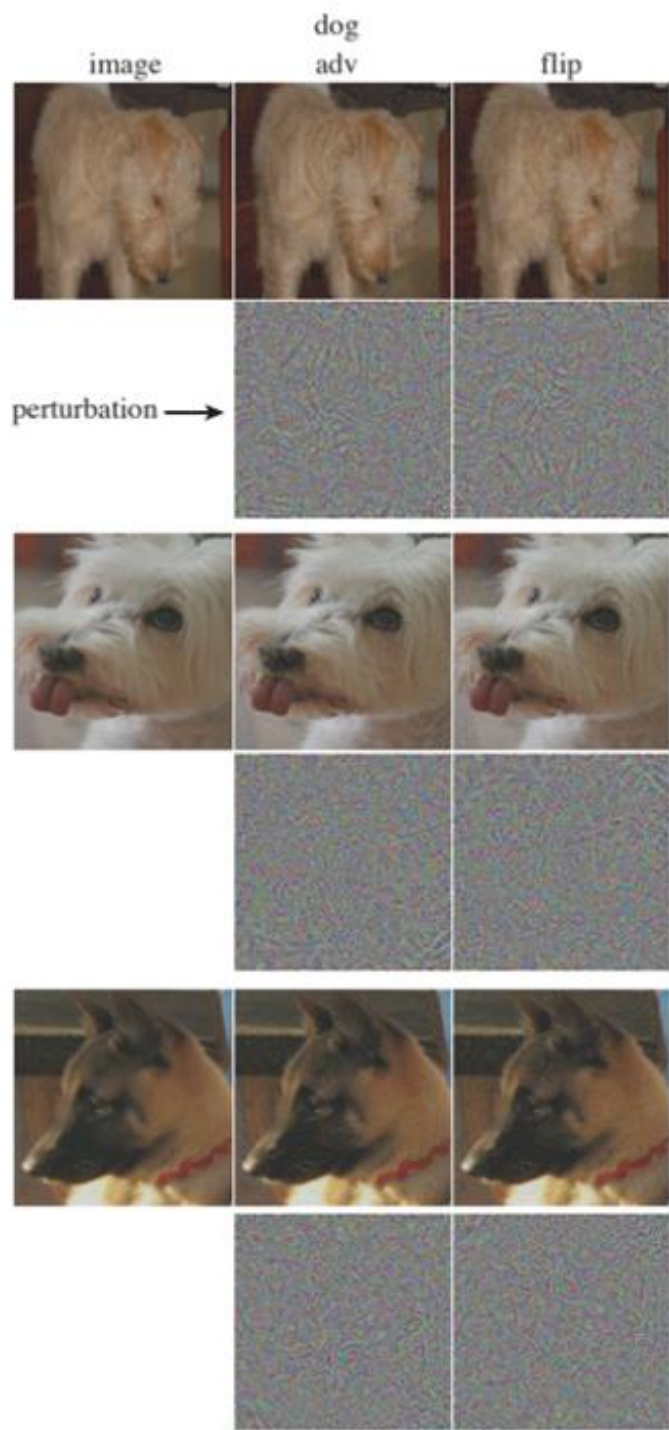


skier_adv.png

Human readers will easily identify the image as showing two men on skis. Google's Cloud Vision service reported being 91 percent certain it saw a dog. Other stunts have shown how to make stop signs invisible, or audio that sounds benign to humans but is transcribed by software as "OK Google browse to evil dot com."

LABSIX





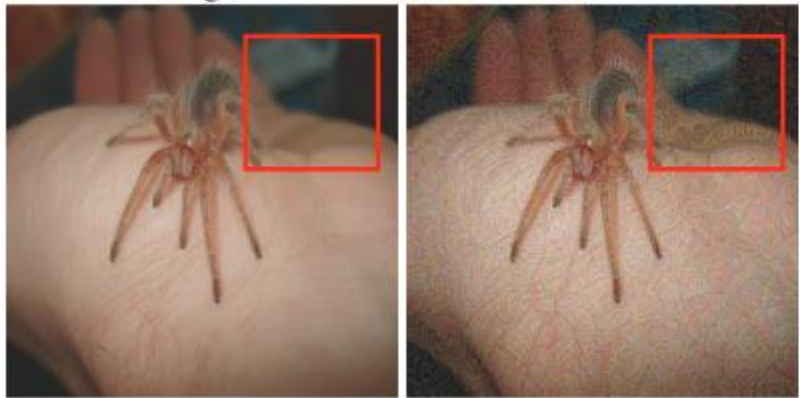
texture modification
image adv



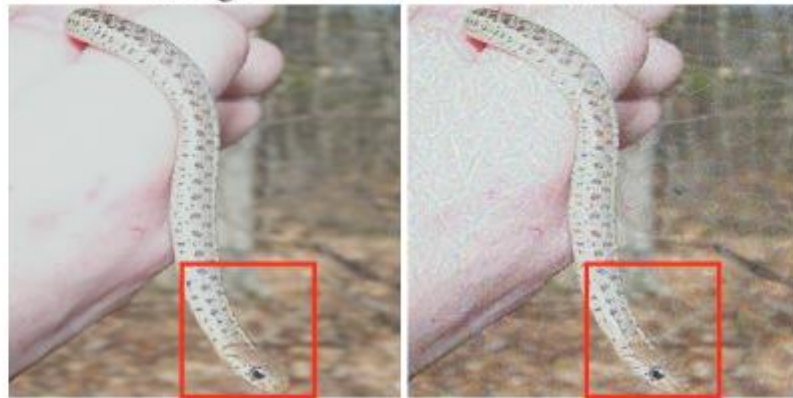
dark parts modification
image adv



edge enhancement
image adv



edge destruction
image adv



AMS F_2 Algorithm

- Let $s \in \{-1, +1\}^n$ be a sign vector of length n
- Let $Z = \langle s, f \rangle = s_1 f_1 + \dots + s_n f_n$ and consider Z^2

$$E[Z^2] = \sum_{i,j} E[s_i s_j f_i f_j] = f_1^2 + \dots + f_n^2$$

$$\text{Var}[Z^2] \leq \sum_{i,j} E[s_i s_j s_k s_l f_i f_j f_k f_l] \leq 2F_2^2$$

- Take the mean of $O\left(\frac{1}{\varepsilon^2}\right)$ inner products for $(1 + \varepsilon)$ -approximation
[AlonMatiasSzegedy99]

“Attack” on AMS

- Can learn whether $s_i = s_j$ from $\langle s, e_i + e_j \rangle$
- Let $f_i = 1$ if $s_i = s_1$ and $f_i = -1$ if $s_i \neq s_1$
- $Z = \langle s, f \rangle = s_1 f_1 + \dots + s_n f_n = m$ and $Z^2 = m^2$ deterministically
- What happened? Randomness of algorithm not independent of input

Reconstruction Attack on Linear Sketches

- Linear sketches are “not robust” to adversarial attacks, must use $\Omega(n)$ space [HardtWoodruff13]
- Approximately learn sketch matrix U , query something in the kernel of U
- Iterative process, start with V_1, \dots, V_r
- Correlation finding: Find vectors weakly correlated with U orthogonal to V_{i-1}
- **Boosting**: Use these vectors to find strongly correlated vector v
- **Progress**: Set $V_i = \text{span}(V_{i-1}, v)$

Insertion-Only Streams

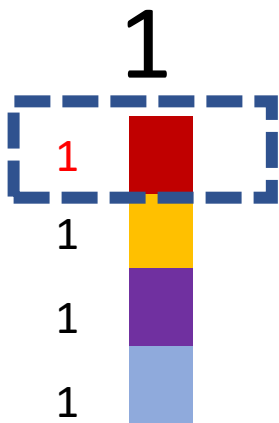
- **Key:** Deletions are needed to perform this attack
- Similar lower bounds for the sliding window model
[DatarGionisIndykMotwani02]
- Assume insertion-only updates
- How do the previous results work?

Robust Algorithms

- Suppose we are trying to approximate some given function
 - Suppose we have a streaming algorithm for this function
 - Suppose this function is monotonic and the stream is insertion-only
- Sketch switching framework [Ben-EliezerJayaramWoodruffYogev20] gives a robust for this function
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output

Sketch Switching

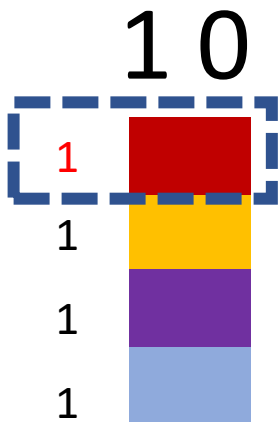
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

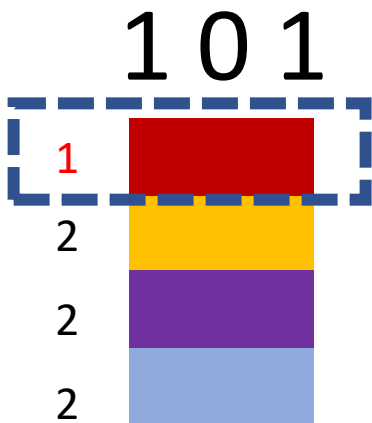
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

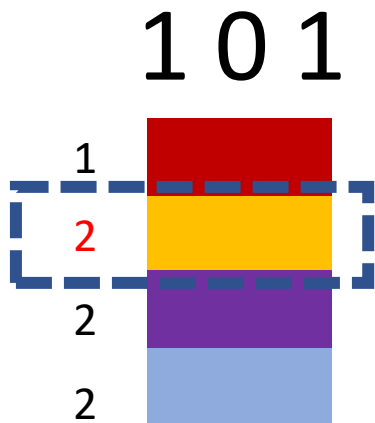
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

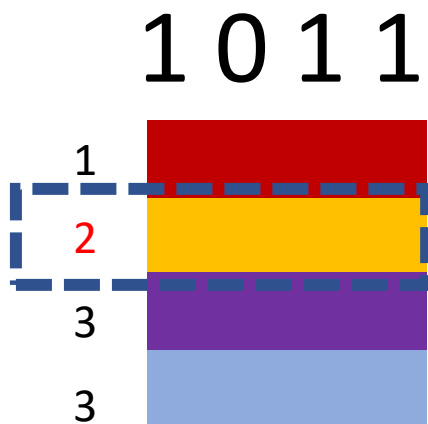
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

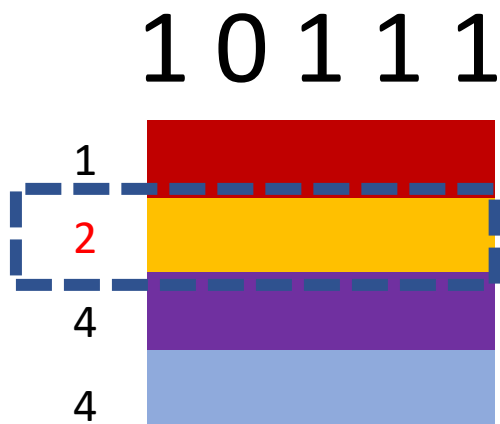
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

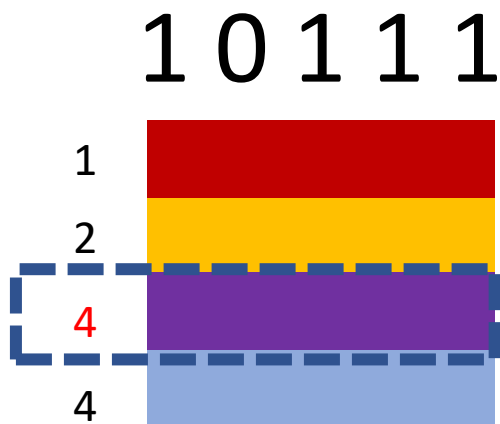
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

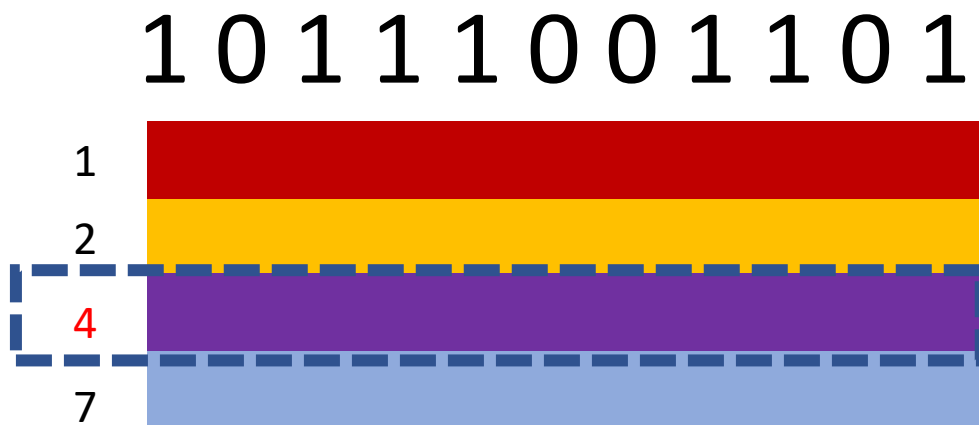
- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)

Sketch Switching

- Start many instances of the streaming algorithm at the beginning
- Use an instance of the algorithm but “freeze” the output
- Each time the next instance has value $(1 + O(\epsilon))$ more than the “frozen” output, use the next instance and “freeze” its output



- Example: Number of ones in the stream (2-approximation)
- Number of ones stream is at least 4 and at most 8
- 4 is a good approximation

Sketch Switching Summary

- Sketch switching for robust algorithms uses $\frac{1}{\varepsilon^2}$ space each time F_p increases by $(1 + \varepsilon)$ and function increases $\frac{1}{\varepsilon}$ times
- How much space do “typical” algorithms use?

$(1 + \varepsilon)$ -Approximation Streaming Algorithms

- Space $O\left(\frac{1}{\varepsilon^2} + \log n\right)$ algorithm for F_0 [KaneNelsonWoodruff10, Blasiok20]
- Space $O\left(\frac{1}{\varepsilon^2} \log n\right)$ algorithm for F_p with $p \in (0, 2]$ [BlasiokDingNelson17]
- Space $O\left(\frac{1}{\varepsilon^2} n^{1-2/p} \log^2 n\right)$ algorithm for F_p with $p > 2$ [Ganguly11, GangulyWoodruff18]
- Space $O\left(\frac{1}{\varepsilon^2} \log n\right)$ algorithm for L_2 -heavy hitters [BravermanChestnutIvkinNelsonWangWoodruff17]

Sketch Switching Summary

- Sketch switching for robust algorithms uses $\frac{1}{\varepsilon^2}$ space each time F_p increases by $(1 + \varepsilon)$ and function increases $\frac{1}{\varepsilon}$ times
- Sketch switching gives $O\left(\frac{1}{\varepsilon^3}\right)$ dependency in space for many problems