# COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2022.
Lecture 9

# Logistics

- Problem Set 2 is due next Friday, 10/14 at 11:59pm.
- The midterm is the following Thursday, 10/20 in class.
- Many students want some more time to go over Distinct Elements/Median trick/LogLog algorithm.
- I will plan to cover less material on high dimension geometry before the midterm and review this material instead.
- If we have time, I'll also go over some more practical use cases of distinct elements counting. Also see Lecture 9 slides.

# Summary

Last Class:

- Analysis of distinct elements counting vis MinHashing.
- The Median Trick to boost success probability.
- High-level overview of practical distinct elements algorithms (see posted slides for more info if you are interested).

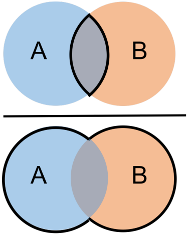
This Class:

- Introduction of Jaccard similarity and the similarity search problem.
- Locality sensitive hashing for fast similarity search.
- MinHashing for Jaccard similarity search.

# Another Fundamental Problem

**Jaccard Index:** A similarity measure between two sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}.$$



Natural measure for similarity between bit strings – interpret an $n$ bit string as a set, containing the elements corresponding the positions of its ones. $J(x,y) = \frac{\#\text{ shared ones}}{\text{total ones}}$.

# Search with Jaccard Similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}.$$

**Want Fast Implementations For:**

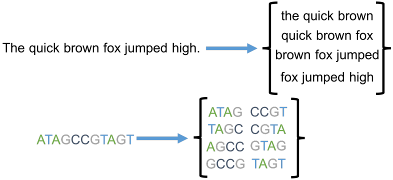
- **Near Neighbor Search:** Have a database of $n$ sets/bit strings and given a set $A$, want to find if it has high Jaccard similarity to anything in the database. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have $n$ different sets/bit strings and want to find all pairs with high Jaccard similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

Will speed up via randomized locality sensitive hashing.

# Application: Document Similarity

## Document Similarity:

- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. ($n$-grams, $k$-mers)



The quick brown fox jumped high.

the quick brown
quick brown fox
brown fox jumped
fox jumped high

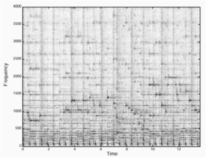
ATAGCCGTAGT

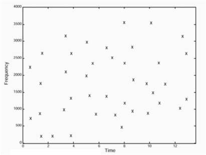
ATAG CCGT
TAGC CGTA
AGCC GTAG
GCCG TAGT

# Application: Audio Search

## Audio Fingerprinting:

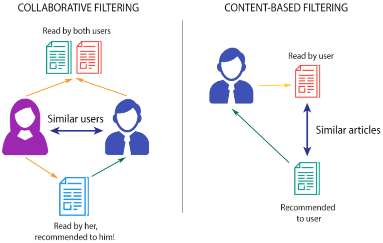
- E.g., in audio search (Shazam), Earthquake detection.
- Represent sound clip via a binary ‘fingerprint’ then compare with Jaccard similarity.





# Application: Collaborative Filtering

Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.

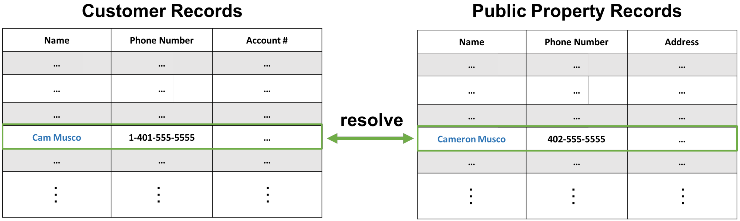


- Twitter: represent a user as the set of accounts they follow. Match users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.
- Netflix: look at sets of movies watched. Amazon: look at products purchased, etc.

# Application: Entity Resolution

**Entity Resolution Problem:** Want to combine records from multiple data sources that refer to the same entities.

- E.g. data on individuals from voting registrations, property records, and social media accounts. Names and addresses may not exactly match, due to typos, nicknames, moves, etc.
- Still want to match records that all refer to the same person using all pairs similarity search.



**Customer Records**

| Name | Phone Number | Account # |
|---|---|---|
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| Cam Musco | 1-401-555-5555 | ... |
| ... | ... | ... |
| ⋮ | ⋮ | ⋮ |

**Public Property Records**

| Name | Phone Number | Address |
|---|---|---|
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| Cameron Musco | 402-555-5555 | ... |
| ... | ... | ... |
| ⋮ | ⋮ | ⋮ |

See Section 3.8.2 of *Mining Massive Datasets* for a discussion of a real world example involving 1 million customers. Naively this would

# Application: Spam and Fraud Detection

Many applications to spam/fraud detection. E.g.

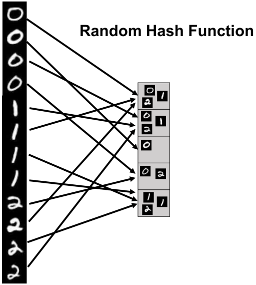
- Fake Reviews: Very common on websites like Amazon. Detection often looks for (near) duplicate reviews on similar products, which have been copied. 'Near duplicate' measured with shingles + Jaccard similarity.
- Lateral phishing: Phishing emails sent to addresses at a business coming from a legitimate email address at the same business that has been compromised.
    - One method of detection looks at the recipient list of an email and checks if it has small Jaccard similarity with any previous recipient lists. If not, the email is flagged as possible spam.

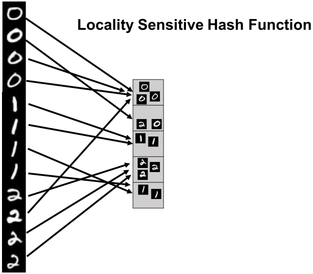
# Locality Sensitive Hashing

**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Locality sensitive hashing (LSH).

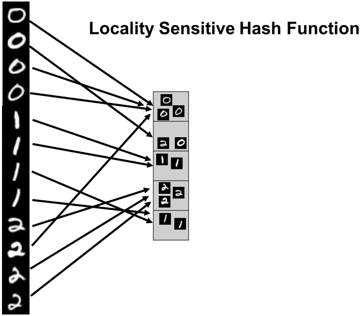
- Design a hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)





# LSH For Similarity Search

How does locality sensitive hashing (LSH) help with similarity search?



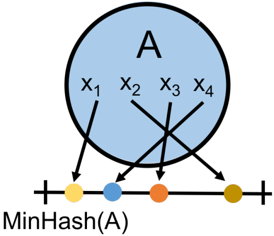
- **Near Neighbor Search:** Given item $x$, compute $\mathrm{h}(x)$. Only search for similar items in the $\mathrm{h}(x)$ bucket of the hash table.
- **All-pairs Similarity Search:** Scan through all buckets of the hash table and look for similar pairs within each bucket.

# MinHashing

**An Example:** Locality sensitive hashing for Jaccard similarity.

**MinHash(A):** [Andrei Broder, 1997 at Altavista]

- Let $\mathbf{h} : U \rightarrow [0, 1]$ be a random hash function
- $\mathbf{s} := 1$
- For $x_1, \ldots, x_{|A|} \in A$
    - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_k))$
- Return $\mathbf{s}$

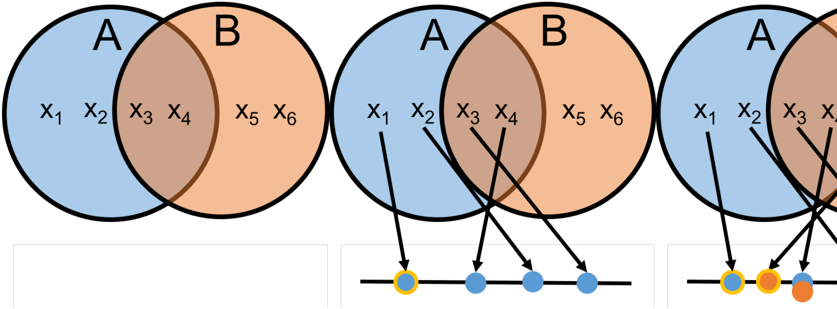


Identical to our distinct elements sketch!

# MinHash Analysis

For two sets $A$ and $B$, what is $\Pr(MinHash(A) = MinHash(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $\mathbf{h}(x) = \mathbf{h}(y)$ for $x \neq y$ (i.e., no spurious collisions)



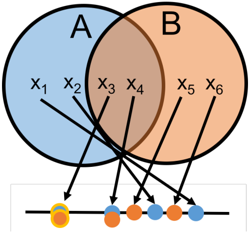
A
B
$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$
A
B
$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$
A
$x_1$ $x_2$ $x_3$

- $MinHash(A) = MinHash(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.

# MinHash Analysis

For two sets $A$ and $B$, what is $\Pr(MinHash(A) = MinHash(B))$?

**Claim:** $MinHash(A) = MinHash(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.



$$\Pr(MinHash(A) = MinHash(B)) =? \frac{|A \cap B|}{\text{total } \# \text{ items hashed}}$$
$$= \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$

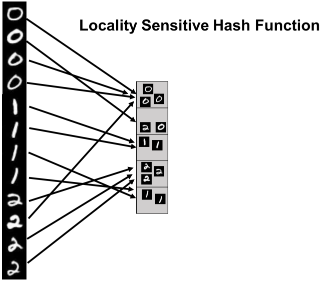
Locality sensitive: the higher $J(A, B)$ is, the more likely $MinHash(A), MinHash(B)$ are to collide.

# Similarity Search with MinHash

**Goal:** Given a document $y$, identify all documents $x$ in a database with Jaccard similarity (of their shingle sets) $J(x, y) \geq 1/2$.

**Our Approach:**

- Create a hash table of size $m$, choose a random hash function $\mathrm{g} : [0, 1] \to [m]$, and insert every item $x$ into bucket $\mathrm{g}(MinHash(x))$. Search for items similar to $y$ in bucket $\mathrm{g}(MinHash(y))$.



- What is $\Pr[\mathrm{g}(MinHash(x)) == \mathrm{g}(MinHash(y))]$ assuming

# Reducing False Negatives

With a simple use of MinHash, we miss a match $x$ with $J(x, y) = 1/2$ with probability $1/2$. How can we reduce this false negative rate?

**Repetition:** Run MinHash $t$ times independently, to produce hash values $MH_1(x), \ldots, MH_t(x)$. Apply random hash function $\mathbf{g}$ to map all these values to locations in $t$ hash tables.

- To search for items similar to $y$, look at all items in bucket $\mathbf{g}(MH_1(y))$ of the $1^{st}$ table, bucket $\mathbf{g}(MH_2(y))$ of the $2^{nd}$ table, etc.
- What is the probability that $x$ with $J(x, y) = 1/2$ is in at least one of these buckets, assuming for simplicity $\mathbf{g}$ has no collisions? $1-$ (probability in *no* buckets) $= 1 - \left(\frac{1}{2}\right)^t \approx .99$ for $t = 7$.
- What is the probability that $x$ with $J(x, y) = 1/4$ is in at least one of these buckets, assuming for simplicity $\mathbf{g}$ has no collisions? $1-$ (probability in *no* buckets) $= 1 - \left(\frac{3}{4}\right)^t \approx .87$ for $t = 7$.

Potential for a lot of false positives! Slows down search time.

# Balancing Hit Rate and Query Time

We want to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)

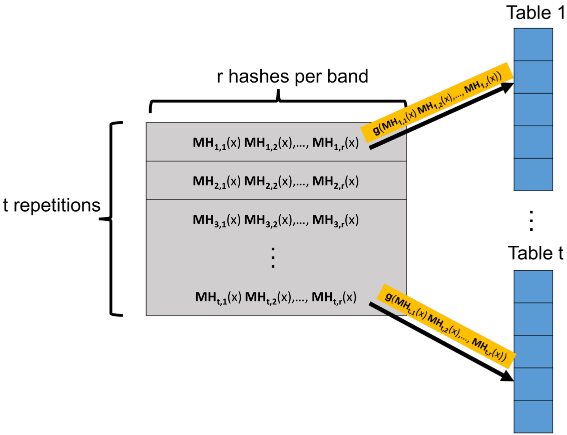


Table 1
r hashes per band
$g(MH_{1,1}(x)\ MH_{1,2}(x),\dots,\ MH_{1,r}(x))$
$MH_{1,1}(x)\ MH_{1,2}(x),\dots,\ MH_{1,r}(x)$
$MH_{2,1}(x)\ MH_{2,2}(x),\dots,\ MH_{2,r}(x)$
t repetitions
$MH_{3,1}(x)\ MH_{3,2}(x),\dots,\ MH_{3,r}(x)$
Table t
$MH_{t,1}(x)\ MH_{t,2}(x),\dots,\ MH_{t,r}(x)$
$g(MH_{t,1}(x)\ MH_{t,2}(x),\dots,\ MH_{t,r}(x))$

Create *t* hash tables. Each is indexed into not with a single MinHash value, but with *r* values, appended together. A length *r* signature.

# Balancing Hit Rate and Query Time

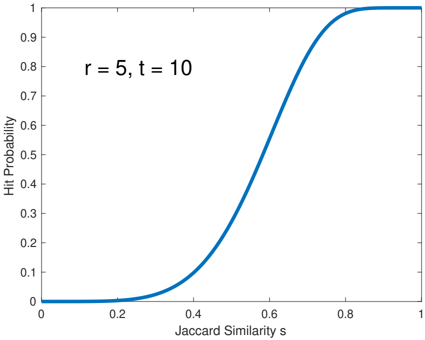
Consider searching for matches in $t$ hash tables, using MinHash signatures of length $r$. For $x$ and $y$ with Jaccard similarity $J(x, y) = s$:

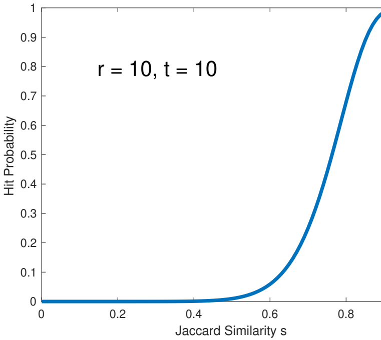
- Probability that a single hash matches. $\Pr\left[MH_{i,j}(x) = MH_{i,j}(y)\right] = J(x, y) = s$.
- Probability that $x$ and $y$ having matching signatures in repetition $i$. $\Pr\left[MH_{i,1}(x), \ldots, MH_{i,r}(x) = MH_{i,1}(y), \ldots, MH_{i,r}(y)\right] = s^r$.
- Probability that $x$ and $y$ don't match in repetition $i$: $1 - s^r$.
- Probability that $x$ and $y$ don't match in *all repetitions*: $(1 - s^r)^t$.
- Probability that $x$ and $y$ match in at least one repetition:

$$\text{Hit Probability: } 1 - (1 - s^r)^t.$$

# The $s$-curve

Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x,y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.





$r$ and $t$ are tuned depending on application. 'Threshold' when hit probability is 1/2 is $\approx (1/t)^{1/r}$. E.g., $\approx (1/30)^{1/5} = .51$ in this case.

# $s$-curve Example

**For example:** Consider a database with $10,000,000$ audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x,y) \geq .9$.

- There are 10 true matches in the database with $J(x,y) \geq .9$.
- There are $10,000$ near matches with $J(x,y) \in [.7,.9]$.

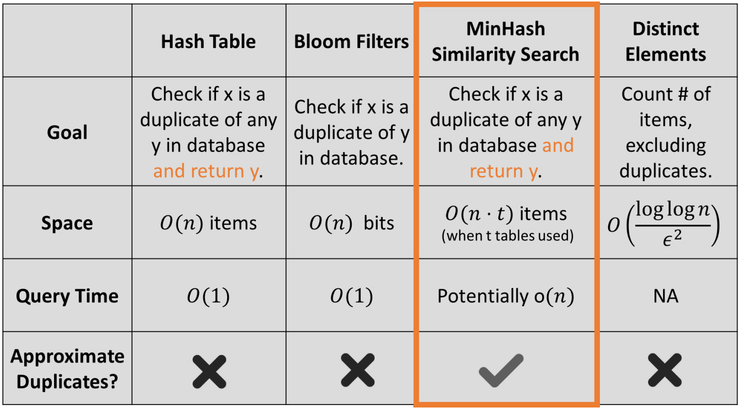
With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x,y) = s$ is $1-(1-s^{25})^{50}$.

- Hit probability for $J(x,y) \geq .9$ is $\geq 1-(1-.9^{25})^{50} \approx .98$
- Hit probability for $J(x,y) \in [.7,.9]$ is $\leq 1-(1-.9^{25})^{50} \approx .98$
- Hit probability for $J(x,y) \leq .7$ is $\leq 1-(1-.7^{25})^{50} \approx .007$

**Expected Number of Items Scanned:** (proportional to query time)

$$\leq 10 + .98 * 10,000 + .007 * 9,989,990 \approx 80,000 \ll 10,000,000.$$

# Hashing for Duplicate Detection



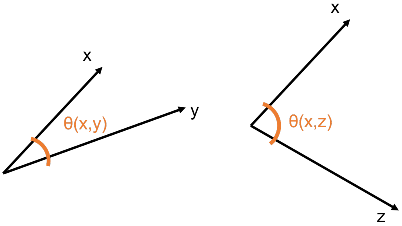
| | Hash Table | Bloom Filters | MinHash Similarity Search | Distinct Elements |
|---|---|---|---|---|
| **Goal** | Check if x is a duplicate of any y in database and return y. | Check if x is a duplicate of y in database. | Check if x is a duplicate of any y in database and return y. | Count # of items, excluding duplicates. |
| **Space** | $O(n)$ items | $O(n)$ bits | $O(n \cdot t)$ items<br>(when t tables used) | $O\left(\frac{\log\log n}{\epsilon^2}\right)$ |
| **Query Time** | $O(1)$ | $O(1)$ | Potentially o$(n)$ | NA |
| **Approximate Duplicates?** | ✗ | ✗ | ✓ | ✗ |

All different variants of detecting duplicates/finding matches in large datasets. An important problem in many contexts!

# Generalizing Locality Sensitive Hashing

Repetition and $s$-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

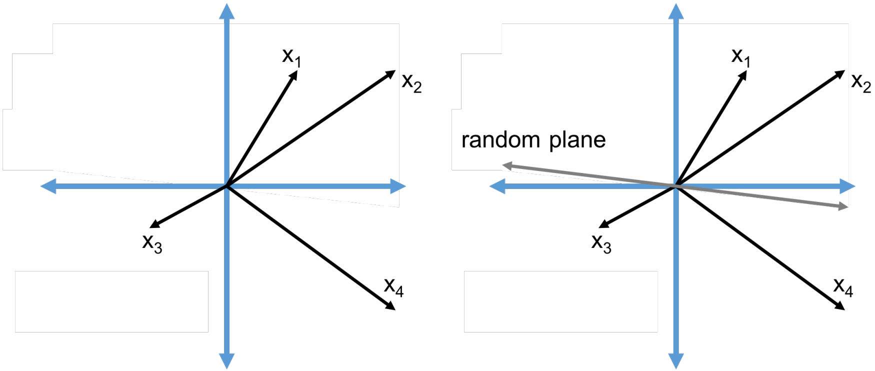


**Cosine Similarity:** $\cos(\theta(x,y)) = \frac{\langle x,y\rangle}{\|x\|_2 \cdot \|y\|_2}$.

- $\cos(\theta(x,y)) = 1$ when $\theta(x,y) = 0°$ and $\cos(\theta(x,y)) = 0$ when $\theta(x,y) = 90°$, and $\cos(\theta(x,y)) = -1$ when $\theta(x,y) = 180°$

# SimHash for Cosine Similarity

**SimHash Algorithm:** LSH for cosine similarity.

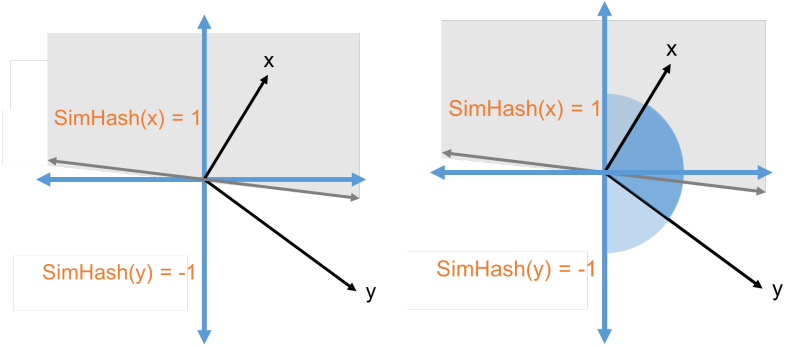


$SimHash(x) = \mathrm{sign}(\langle x, t\rangle)$ for a random vector $t$.

# SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

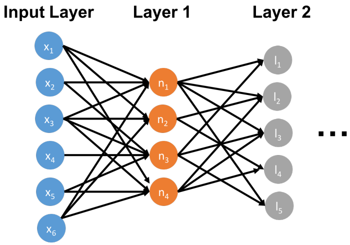
$SimHash(x) \neq SimHash(y)$ when the plane separates $x$ from $y$.

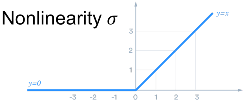


- $\Pr[SimHash(x) \neq SimHash(y)] = \frac{\theta(x,y)}{\pi}$
- $\Pr[SimHash(x) = SimHash(y)] = 1 - \frac{\theta(x,y)}{\pi} \approx \frac{\cos(\theta(x,y))+1}{2}$

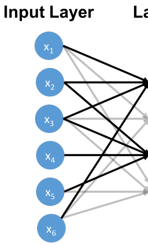
# Hashing for Neural Networks

Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).



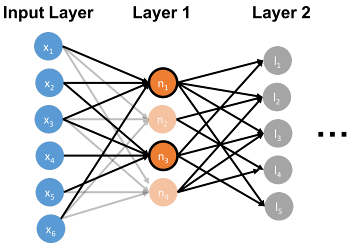


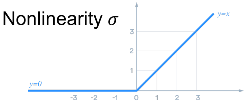
$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i) \cdot x_j\right) = \sigma(\langle w_i, x\rangle)$$



- Evaluating $\mathcal{N}(x)$ requires $|x| \cdot |\text{layer } 1| + |\text{layer } 1| \cdot |\text{layer } 2| + \ldots$ multiplications if fully connected.
- Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with high activation when $x$ is presented.

# Hashing for Neural Networks





$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i) \cdot x_j\right) = \sigma(\langle w_i, x\rangle)$$

- Important neurons have high activation $\sigma(\langle w_i, x\rangle)$.
- Since $\sigma$ is typically monotonic, this means large $\langle w_i, x\rangle$.
- $\cos(\theta(w_i, x)) = \frac{\langle w_i,\ x\rangle}{\|w_i\|\|x\|}$. Thus these neurons can be found very quickly using LSH for cosine similarity search.
- Store each weight vector $w_i$ (corresponding to each node) in a set of hash tables and check inputs $x$ for similarity to these stored vectors.