Casus ubique valet; semper tibi pendeat hamus:
Quo minime credas gurgite, piscis erit.
[Luck affects everything. Let your hook always be cast.
Where you least expect it, there will be a fish.]

— Publius Ovidius Naso [Ovid], *Ars Amatoria*, Book III (2 AD)

There is no sense being precise
when you don't even know what you're talking about.

— Attributed to John von Neumann

15 Streaming Algorithms and Dimensionality Reduction

Randomness is an essential tool for designing streaming algorithms, which is a different model of computation that we will see in this lecture. A **data stream** is an extremely long sequence of items from a universe that can only be read once, in order. Some examples of data streams are packets passing through a network router, a sequence of google search queries, New York Stock Exchange trades.

Standard algorithms are not suitable for computation in a streaming setting because there is simply too much data to store, and it arrives too quickly for complex computations. Ideally, one would like to compute properties of the data stream with a small memory (and time as well). In particular, given an input stream $a_1, a_2, \ldots, a_m$ where each a_i comes from a universe $\mathcal{U}$ of n elements, the holy grail in the streaming setting is an algorithm that uses $\text{poly}(\log m, \log n)$ bits of memory. Note that $\log m$ bits are needed to remember which element of the stream we are processing and $\log n$ bits are needed to remember the current element in the stream. Typically, the length of the stream is not known, and in practice one either assumes some upper bound on the length or design algorithms that are oblivious to the length of the stream.

Streaming algorithms are sometimes used even in non-streaming settings. One particularly important example of this is to process data in massive data centers. The amount of data is typically gigantic — at a scale of petabytes — and is stored on hard disks which are slow to read and write. A low-memory algorithm is desired in such settings since the data relevant for performing the computation can then be stored in the RAM where faster read and write operations are available.

Examples. Let us look at some basic properties of the input stream one would want to compute to get an idea of how one would design a streaming algorithm.

Sum (or Average). Suppose each item a_i of the stream is an $O(\log(n))$-bit number, and we want to store the sum (or average) of the items seen so far. The streaming algorithm just has to remember the sum (or average) of the items seen thus far. Since, the the sum of m $O(\log(n))$-bit numbers is at most $2^{O(\log n)}m$, the space requires is $O(\log n + \log m)$ bits.

Minimum or Maximum. Consider the same setting as above where now we want to maintain the minimum or maximum of all the numbers seen so far. This only requires $O(\log n)$ bits of space, since one only has to keep the current maximum or minimum in the memory.

Median. Now suppose we want to maintain the median of all the numbers seen so far. It is not obvious how to design a small space streaming algorithm for this, since one has to remember

the bottom or top half of all the elements to determine the median at the end of the stream. In fact, computing the median exactly requires $\Omega(n)$ bits of space in the streaming setting. The proof of this fact is not difficult but we omit it here, since we will see a similar proof soon.

In-class Exercises. The following are some puzzles to get used to designing algorithms in a streaming setting.

1. Suppose the input stream is $a_1, a_2, \ldots, a_n$ where each $a_i \in [n+1]$ and the a_i's are distinct. Find the missing value using $O(\log n)$ space.

   Store sum of all the elements of the stream so far. Output $\sum_{i\in[n+1]} i - \sum_{i\in[n]} a_i$.

2. Given a stream $a_1, a_2, \ldots, a_m$ of distinct elements, sample a uniformly random element from all the elements seen so far, using only $O(\log n + \log m)$ bits of space.

   Let S be the sample we store in the memory. Consider the following algorithm:

   - *Initialize $S \leftarrow a_1$.*
   - *When a_i arrives, with probability $\frac{1}{i}$, set $S \leftarrow a_i$.*

   Why is S uniformly distributed? Consider the situation when the item a_i arrives. Note that $\mathbb{P}[S = a_i] = \frac{1}{i}$ and for all $j \neq i$, we have $\mathbb{P}[S = a_j] = \left(1 - \frac{1}{i}\right) \cdot \frac{1}{i-1} = \frac{1}{i}$. Thus, S is uniformly distributed among $a_1, \ldots, a_i$.

3. Given a stream $a_1, a_2, \ldots, a_m$ with distinct elements, sample a uniformly random set of k elements (without replacement) from all the elements seen so far, using $O(k(\log n + \log m))$ bits of space.

   Left as exercise.

Distinct Element Estimation

The main problem we will look in this lecture is the distinct element estimation problem: given a stream $a_1, a_2, \ldots, a_m$ where each $a_i \in \mathcal{U}$ for a universe $\mathcal{U}$ of n elements, count the number of distinct elements in the stream. Typically, the number of distinct elements is denoted F_0.

Here are some naive streaming algorithms one can come up with for the above problem.

1. Store an indicator vector which records the elements of the universe we have seen thus far. This requires storing an n-bit vector at each step.
2. Store the set of all the elements we have seen so far in the stream. This requires space $O(m \log n)$ bits.

Both of these algorithms require space that is linear in either m or n. This leads us to the very natural question: can we design a poly$(\log m, \log n)$ space algorithm for this problem?

It turns out that both randomization and approximation are necessary to solve this problem. In particular, one can show that every deterministic algorithm requires $\Omega(n)$ bits, even for a 0.1

approximation (i.e., computing a number D such that $0.9F_0 \leq D \leq 1.1F_0$) and every randomized algorithm that computes the exact number of distinct elements F_0 requires $\Omega(m)$ bits of space. We will only prove a lower bound on the space complexity of exact deterministic algorithms here. Proving that both approximation and randomization are necessary as mentioned above requires some more techniques that we will not have time to cover in this course.

Lemma. *Assume that $n \geq 2m$, where n is the size of the universe and m is the length of the stream. Then, any deterministic algorithm that exactly counts the number of distinct elements requires $\Omega(m)$ space.*

Proof. Suppose the algorithm uses s bits of memory, then there are at most 2^s possibilities for the contents of the memory just before the final item is received. Without loss of generality, let us focus on the input streams where the first $m-1$ elements are all distinct. There are $\binom{n}{m-1}$ sets of distinct items that the algorithm could observe among the first $m-1$ elements of the stream.

We claim that if the algorithm always exactly computes the number of distinct elements, then it the number of possible memory configurations must be more than the number of possible sets of distinct items among the first $m-1$ elements. More concretely,

$$2^s \geq \binom{n}{m-1} \geq \binom{2m}{m-1},$$

which implies that $s = \Omega(m)$.

To prove this we argue by contradiction: suppose $2^s < \binom{n}{m-1}$, then by the pigeonhole principle, there must be two different sets S and T of $m-1$ distinct items that lead to the same memory configuration. Since S and T are distinct and of equal size, there exists elements x and y such that $x \in S \setminus T$ and $y \in T \setminus S$. Now, because the memory configuration is the same for both S and T before the final element is received, the answer of the algorithm is a deterministic function of the final element of the stream and the memory contents. In particular, the answer of the algorithm on $S \cup \{x\}$ is the same as the answer on $T \cup \{x\}$. However, the number of distinct element in the two cases differ, so the algorithm must err on one of them. □

Randomized Algorithm for Distinct Element Estimation

Thus, the only possibility in this setting is to allow for both approximation and randomness. In particular, given a stream $a_1, \ldots, a_m$, we would like to design a randomized algorithm that outputs a number D such that:

$$\mathbb{P}[(1-\epsilon)F_0 \leq D \leq (1+\epsilon)F_0] \geq 1-\delta,$$

for parameters $0 < \epsilon, \delta < 1$.

The best algorithm for this problem is due to Kane, Nelson and Woodruff from 2010. This algorithm uses $O\left(\left(\epsilon^{-2} + \log n\right) \cdot \log \frac{1}{\delta}\right)$ space, which is optimal in terms of space complexity. Here we shall see a simpler algorithm due to Chakraborty, Vinodchandra and Meel from 2023 that uses $O\left(\epsilon^{-2} \cdot \log n \cdot \log \frac{m}{\delta}\right)$ space.

The basic idea behind the algorithm is the following: Suppose we randomly sample a set X where each distinct element in the stream is included with probability p independently. Then, denoting by $|X|$ the size of the set X, we have that $\mathbb{E}[|X|] = pF_0$ and thus $\mathbb{E}[\frac{|X|}{p}] = F_0$. Furthermore, by Chernoff bounds, for any $0 < \epsilon$, we have that

$$\mathbb{P}\left[\left||X| - \mathbb{E}[|X|]\right| \geq \epsilon \mathbb{E}[|X|]\right] \leq e^{-\frac{\epsilon^2 \cdot \mathbb{E}[|X|]}{2+\epsilon}},$$

which implies that

$$\mathbb{P}\left[\left|\frac{|X|}{p} - F_0\right| \geq \epsilon F_0\right] \leq e^{-\epsilon^2 \cdot pF_0}.$$

Note that the probability of being far from F_0 is small if pF_0 which is the expected size of X is not too small. Thus, we can randomly sample a set X as above, divide its size by p, and hope to approximate F_0, provided the size of the set is not too small. In particular, if we set $p = \frac{100}{\epsilon^2 F_0}\log\left(\frac{m}{\delta}\right)$, then the probability is at most $\frac{\delta}{4m}$ and the expected size of X is $\frac{100}{\epsilon^2}\log\left(\frac{m}{\delta}\right)$. Let us define the parameter $\textbf{thresh} := \frac{100}{\epsilon^2}\log\left(\frac{m}{\delta}\right)$.

However, there are two problems to carry out the above plan: first, how can we independently sample each distinct element with probability p? And second, how do we set the sampling rate p? For the latter, the Chernoff bound calculation says that we do not want p to be too small, but we do not want p to be too large either. For example, if $p = 1$, then X is the set of all distinct items and storing it in the memory would be too costly. Ideally, we want $p \approx \textbf{thresh}/F_0$ so that the expected size of X is close to **thresh**, but this would require us knowing F_0!

Let's see how to resolve these problems one by one.

Sampling from Distinct Elements. Consider the following streaming algorithm that maintains a random subset X of all the distinct elements seen thus far. Let X be the current set stored in the memory and a_i be the next item, then the algorithm does the following:

Algorithm 1 Sampling from the set of distinct elements with probability p

1: **if** $a_i \in X$ **then**
2: Remove a_i from X
3: **end if**
4: Add a_i to X with probability p

It is easy to check by a simple case analysis that each distinct element is included in X independently with probability p when executing the above algorithm.

Rate of Sampling. The key idea to get around the fact that we do not know the correct rate of sampling is to try all rates $p_k = 2^{-k}$ for different values of k. In particular, for $k \in \mathbb{N}$, let X_k be a random subset of the set of distinct elements where each distinct element is sampled at rate 2^{-k} independently. In particular, X_0 is the set of all the distinct elements, X_1 is obtained by sampling each distinct element with probability $1/2$, X_2 is obtained by sampling each distinct element with probability $1/4$ and so on. Note that one can also obtain X_{k+1} by subsampling[1] each element of X_k independently with probability $1/2$.

Consider the following thought experiment, where we maintain all sets $X_0, X_1, X_2, \ldots, X_{k_{\max}}$ in the memory for some value of $k_{\max}$. As long as the set $X_{k_{\max}}$ is not too small, we can use any of the sets $X_0, \ldots, X_{k_{\max}}$ with the corresponding rate to estimate F_0. However, as you might already have noticed that storing all the sets $X_0, X_1, X_2, \ldots, X_{k_{\max}}$ is every costly. Even just storing X_0 is already very costly. So we cannot do this, if we want to design a low-space algorithm.

[1] To ensure independence, this needs to be done carefully. See the original paper: Distinct Elements in Streams: An Algorithm for the (Text) Book. Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. In Proceedings of European Symposium of Algorithms (ESA) 2022.

The main observation to overcome this obstacle is that we only need one of the sets $X_0, X_1, X_2, \ldots, X_{k_{\max}}$ with the corresponding rate in order to estimate F_0. In particular, we keep a threshold for the size of our current set given by the parameter **thresh**. If the current set exceeds this size, we throw it away and move to the next one and keep track of the fact that the value of p has halved.

The overall algorithm is the following.

Algorithm 2 Estimate F_0 on input stream $a_1, \ldots, a_m$

```
Initialize: p ← 1, X ← ∅
for i = 1 to m do
    ▷ Sample from distinct elements at the current rate p
    if a_i ∈ X then
        Remove a_i from X
    end if
    Add a_i to X with probability p

    if |X| ≥ thresh then
        ▷ Subsample half the elements and decrease the rate by half
        Throw away each element of X with probability 1/2
        p ← p/2
    end if
end for
Output |X|/p
```

The above algorithm uses $O\left(\epsilon^{-2} \cdot \log n \cdot \log \frac{m}{\delta}\right)$ space. Furthermore, the following lemma shows that the expected size of the set maintained by the above algorithm at the end is large enough to get a $1 \pm \epsilon$ approximation of F_0 using the Chernoff bounds.

Lemma. *The probability that $p < \frac{\textbf{thresh}}{4F_0}$ at any point during the execution of the algorithm is at most δ.*

In particular the above implies that the expected size of the set X at the end is at least $\textbf{thresh}/4$ with probability at least $1 - \delta$.

Proof. Suppose at some iteration $i \in [m]$, the probability decreases from $2^{-\ell}$ to $2^{-\ell-1}$ where $2^{-\ell-1} < \frac{\textbf{thresh}}{4F_0} \leq 2^{-\ell}$. This can only happen when the subsampled set X at the rate $2^{-\ell}$ has reached its maximum allowed size. However, note that $\frac{\textbf{thresh}}{8} < \mathbb{E}[|X|] \leq \frac{\textbf{thresh}}{4}$. Since, in X each of the F_0 distinct elements is included independently with probability $2^{-\ell-1}$, by Chernoff bounds, the probability that its size is a constant times larger than its expectation is very small. In particular,

$$\mathbb{P}[|X| \geq \textbf{thresh}] \leq e^{-(2/9)\cdot \textbf{thresh}} \leq \delta/m.$$

By union bound over all the m iterations, the probability that p decreases below $\frac{\textbf{thresh}}{4F_0}$ at any point during the execution is at most δ. □

Dimensionality Reduction

How do we deal with data in high dimensions? We often visualize data and algorithms in one, two, or three dimensions, e.g., a graph or a 3D plot. But high-dimensional space does not behave like low-dimensional space, as we will see in the first part of this lecture, so such visualization is not very informative.

In the second part of the lecture, we will ignore our own advice and look at sketching, also known as dimensionality reduction techniques. In particular, we will see that the properties of high-dimensional spaces that seem counterintuitive also give us a lot of utility.

High-dimensional spaces are strange! Recall that the inner product of two vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ in d dimensions is given by:

$$\langle x, y \rangle = x^\top y = y^\top x = \sum_{i=1}^{d} x_i y_i = \|x\| \cdot \|y\| \cdot \cos\theta,$$

where θ is the angle between the two vectors and $\|x\|$ denotes the Euclidean length of x. One can ask how many mutually orthogonal unit vectors $x_1, x_2, \ldots, x_d$ can we find in d dimensions? The answer is there are exactly d such vectors. But if we relax this condition a little bit, for instance, suppose we want to find the largest number of **nearly mutually orthogonal** unit vectors $x_1, \ldots, x_t$ in d dimensions, i.e. $|\langle x_i, x_j \rangle| \leq 0.01$ for all pairs $i \neq j$. It turns out that there can be $\exp(\Theta(d))$ such vectors. In general, if we want the inner product to be at most ϵ, then there can be $\exp(\Theta(\epsilon d))$ such vectors. Thus, in high-dimensional spaces, we can find exponentially many vectors that are all far apart from each other. This is one example of the *curse of dimensionality* that occurs when solving computational problems involving high-dimensional data. For example, solving the nearest neighbor problem is intractable in high-dimensions if our data is truly random, because we will need an exponential amount of data before we start seeing even a single point that is close. In fact, the existence of some kind of low-dimensional structure is often the only reason we can hope to solve such computational problems.

Let's look at another example of the strangeness of high-dimensional spaces. Consider the unit ball in d dimensions $B_d = \{x \in \mathbb{R}^d \mid \|x\| \leq 1\}$. One can wonder what fraction of the volume of the ball resides in an ϵ-shell around its surface, i.e. in the region $\{x \in \mathbb{R}^d \mid 1 - \epsilon \leq \|x\| \leq 1\}$. In 2 or 3 dimensions, the fraction is $O(\epsilon)$ which is very small if ϵ is small say 0.01. However, it turns out that in d dimensions, almost the entire volume resides in the ϵ-shell, in particular, the fraction of the volume of the ball in the shell is $1 - \exp(-\Theta(\epsilon d))$. Similarly, one can ask, how much of the volume is within an ϵ band of the equator, i.e. in the region $\{x \in \mathbb{R}^d \mid |x_1| \leq \epsilon\}$ where $x = (x_1, \ldots, x_d)$ in its coordinate representation. Again, our 2- or 3-dimensional intuition would suggest that the fraction of volume in the band is small if ϵ is small, but it turns out that almost all of the volume is in this band if d is large. Note that we can choose any equator and the same holds true. To summarize, most of the volume of the d-dimensional ball is close to its surface and close to any equator. In a nutshell, the high-dimensional ball looks nothing like the 2- or 3-dimensional ball.

Sketching or Dimensionality Reduction

Despite the fact that low-dimensional space behaves nothing like high-dimensional space, we can still leverage its strangeness to our advantage.

Suppose we have data $x_1, X_2, \ldots, x_N \in \mathbb{R}^d$ and we want to find some way of making it low-dimensional, say we want to convert it to a lower dimensional space $\mathbb{R}^k$ where $k \ll d$. This is a form of data compression and of course, we should not expect lossless compression. Regardless, we would like to preserve the geometry of our data in some of way. Here, we will look at (approximately) preserving the pairwise distances between points.

Such compression has lots of applications. One particular area where it is very helpful is computational geometry where it allows one to approximately solve high-dimensional problems by moving to lower dimensions. As a concrete example consider the **k-means clustering problem**: given a set of points $x_1, \ldots, x_N \in \mathbb{R}^d$ and an integer $k > 1$, we want to find $y_1, \ldots, y_k \in \mathbb{R}^d$ such that

$$\sum_{i=1}^{N} \min_{j \in [k]} \|x_i - y_j\|^2,$$

is minimized. One can show that this problem boils down to partitioning the input points into k clusters, with each cluster having a center, and we want to minimize the sum of squared distances between the points and their closest center. Since this problem only considers pairwise distances between the input points, if we can find a way to make our problem low-dimensional while approximately preserving the pairwise distances, we can solve k-means approximately much faster by working in the low-dimensional space. A similar situation arises for other problems like nearest neighbor search.

Johnson-Lindenstrauss Lemma. Our key tool to convert high-dimensional data to low-dimensional data is by a powerful result called the Johnson-Lindenstrauss Lemma proven in 1984 by Johnson and Lindenstrauss. This lemma gives a way to embed N data points from $\mathbb{R}^d$ in $\mathbb{R}^n$ where $n = C \log N$, for a universal constant C while approximately preserving the pairwise distances. Note that one can always assume that if we have N points in d-dimensional space, then $d \leq N$ by restricting to the hyperplane that contains all the points which is atmost N-dimensional. So, applying the above lemma gives an exponential reduction in the dimension of the space, since it goes down from N to $O(\log N)$.

The embedding is done via a linear map, aka, a linear transformation that maps $\mathbb{R}^d$ to $\mathbb{R}^n$. In other words, we can find a $n \times d$ dimensional matrix A and our embedding will map the point $x \in \mathbb{R}^d$ to the n-dimensional vector Ax while approximately preserving the distances.

$$A : \mathbb{R}^d \to \mathbb{R}^k$$

Theorem. *For any $x_1, \ldots, x_N \in \mathbb{R}^d$, there exists an integer $n \leq C \log N$ for a universal positive constant C and a matrix $A \in \mathbb{R}^{n \times d}$, such that the following holds for every $i, j \in [N]$,*

$$(1-\epsilon)\|x_i - x_j\| \leq \|Ax_i - Ax_j\| \leq (1+\epsilon)\|x_i - x_j\|$$

How do we find such a matrix A? In fact, we will see the power of randomness in this example again — picking a random matrix will work with high probability. It will also turn out that the random matrix does not even depend on the data points and can be sampled ahead of time.

In order to prove the above lemma, we will need to work with the Gaussian distribution (also called the Normal distribution), so we take a small detour to look at its properties.

Gaussian or Normal Distribution. We will work with continuous probability distributions, such as distributions on the real line or in the d-dimensional real space $\mathbb{R}^d$. Continuous distributions have a probability density function (pdf) which gives the "weight" the distribution gives to a particular region. For example, in 1-dimensional space, the pdf is a function $p : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and the probability of an interval $I \subset \mathbb{R}$ is given by $\int_I p(x)dx$.

Gaussian distribution is one of the most useful probability distributions. The probability density function (pdf) of a 1-dimensional **standard** Gaussian is given by:

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

The mean of the 1-dimensional standard Gaussian G is given by

$$\mu = \mathbb{E}[G] = \int_{\mathbb{R}} x p(x) dx = 0.$$

Note that this is analogous to the expression we saw for the expectation of a discrete random variable X where $\mathbb{E}[X] = \sum_x x \cdot \mathbb{P}[X = x]$.

Another important quantity is the variance of the standard gausssian:

$$\sigma^2 = \mathbb{E}[(G-\mu)^2] = \int_{\mathbb{R}} x^2 p(x) dx = 1.$$

The 1-dimensional standard Gaussian is denoted by $N(0,1)$. In general, one can have a Gaussian with a different mean μ and variance σ^2. Such a Gaussian is denoted by $N(\mu, \sigma^2)$ and its pdf for parameters $\mu, \sigma \in \mathbb{R}$ is given by

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Properties of the Gaussian distribution The normal distribution has several important and unique properties, which make it widely useful in probability and statistics.

Tail Bounds. Suppose we toss n independent coins, where $X_1, \ldots, X_n$ are the outcomes of these tosses. Each $X_i \in \{-1, 1\}$ with $\mathbb{P}[X_i = 1] = \mathbb{P}[X_i = -1] = \frac{1}{2}$. Let $S_n = \sum_{i=1}^n X_i$.

Note that $\mathbb{E}[S_n] = 0$. Also, the Chernoff bound provides an exponential decay in the probability of large deviations from the mean, implying that:

$$\mathbb{P}\left[\left|\frac{S_n}{\sqrt{n}}\right| \geq t\right] \leq 2e^{-t^2/2}.$$

In fact, something stronger is true: as n increases, the distribution of S_n approaches a Gaussian distribution. This is the essence of the **Central Limit Theorem**, which implies that for large n, the distribution of $S_n/\sqrt{n}$ is approximately $N(0,1)$.

A tail inequality of the form

$$\mathbb{P}[|G| > t] \leq 2e^{-t^2/2},$$

is called a Gaussian tail bound because it holds when G is $N(0,1)$. In particular, it shows that the probability of observing a large value from a Gaussian distribution decreases exponentially

with the square of the value. The proof of the above tail bound for a standard Gaussian follows from basic calculus and we omit it here.

Sum and scaling of Gaussians. Suppose $G_1 \sim N(\mu_1, \sigma_1^2)$ and $G_2 \sim N(\mu_2, \sigma_2^2)$. Then the sum of these two independent Gaussian variables is also a Gaussian:

$$G_1 + G_2 \sim N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$$

This property generalizes to the sum of many independent Gaussian variables, each with potentially different means and variances.

Similarly, if $G \sim N(\mu, \sigma^2)$, then for any scalar a, the random variable aG is also Gaussian:

$$aG \sim N(a\mu, a^2\sigma^2)$$

This shows that scaling a Gaussian random variable changes its mean by a factor of a and variance by the factor of a^2.

Multivariate Gaussian distribution. A *multivariate Gaussian distribution* in d dimensions is a vector $G = (G_1, \ldots, G_d)$ where each coordinate G_i is an independent $N(0, 1)$ random variable. Denoting the pdf of the one dimensional standard Gaussian by p, the probability density function (pdf) of this distribution is a function $p_d : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ given by:

$$p_d(x) = p(x_1) \cdot p(x_2) \cdot \cdots \cdot p(x_d) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d} x_i^2\right) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\|x\|^2\right),$$

where $x = (x_1, \ldots, x_d)$ in its coordinate representation.

Pictorially, the 2-dimensional pdf looks like a bell-shaped surface in three-dimensional space, where the height represents the probability density at that point.

Thin shell phenomenon. A remarkable property of high-dimensional Gaussians is the *thin shell phenomenon*. Although, based on the probability density function, it seems like most of the probability mass of the d-dimensional standard Gaussian is near the origin, this is not true in high dimensions. In fact, if we sample points from a d-dimensional standard Gaussian, most of them will lie close to the surface of a ball of radius $\sqrt{d}$, even though the probability density is highest at the origin.

Concretely, the thin shell theorem states that for a standard Gaussian vector $G \in \mathbb{R}^d$, the norm $\|G\|$ is concentrated around $\sqrt{d}$:

$$\mathbb{P}\left[0.99\sqrt{d} \leq \|G\| \leq 1.01\sqrt{d}\right] \geq 1 - e^{-cd},$$

for some constant c.

This is similar to the fact mentioned earlier about the unit ball in high dimensions, where most of the volume is concentrated near the surface rather than at the center.

Proof of the Johnson-Lindenstrauss Lemma. We now have all the tools to prove the Johnson-Lindenstrauss Lemma, which we restate below.

Theorem (Johnson-Lindenstrauss '84)**.** *For any $x_1, \ldots, x_N \in \mathbb{R}^d$, there exists an integer $n \leq C \log N$ for a universal positive constant C and a matrix $A \in \mathbb{R}^{n \times d}$, such that the following holds for every $i, j \in [N]$,*

$$(1-\epsilon)\|x_i - x_j\| \leq \|Ax_i - Ax_j\| \leq (1+\epsilon)\|x_i - x_j\|$$

Proof. We will choose the matrix $A = \frac{1}{\sqrt{n}} \cdot G$ where G is an $n \times d$ Gaussian random matrix, i.e. each entry G_{ij} is independent and $N(0,1)$. We will show that with high probability, such a matrix satisfies the lemma.

Let us first understand what this matrix does to a fixed vector $z \in \mathbb{R}^d$.

Fact. *Let $z = (z_1, \ldots, z_d)$ be a unit vector in $\mathbb{R}^d$, i.e., $\|z\| = 1$. Then, Gz is a standard multivariate Gaussian in n dimensions. In other words, each coordinate of Gz is a Gaussian random variable $N(0,1)$, and all coordinates are independent.*

Proof of Fact. Each coordinate i of the vector Gz is given by:

$$(Gz)_i = \sum_{j=1}^{d} G_{ij} z_j$$

Note that each term $G_{ij}z_j$ is $N(0, z_j^2)$ since it is a scaling of a standard Gaussian. Hence, the sum of all these Gaussians is distributed as

$$N(0, z_1^2 + z_2^2 + \cdots + z_d^2) = N(0, \|z\|^2) = N(0,1),$$

where the last equality follows since z is a unit vector. Furthermore, all coordinate of Gz are independent since to compute $(Gz)_i$ we compute the product of the ith row of G with z and all the rows of G are independent. □

Now, consider a pair of points x_i and x_j. We want to show that the distances between these points are approximately preserved. Define

$$z = \frac{x_i - x_j}{\|x_i - x_j\|}.$$

Then, Gz is a standard n-dimensional Gaussian and by the thin shell theorem we have that

$$\mathbb{P}\left[0.99\sqrt{n} \leq |||G||| \leq 1.01\sqrt{n}\right] \geq 1 - e^{-cn}.$$

Rearranging the above gives us that

$$\mathbb{P}\left[0.99\|x_i - x_j\| \leq \left\|\left(\frac{G}{\sqrt{n}}\right)x_i - \left(\frac{G}{\sqrt{n}}\right)x_j\right\| \leq 1.01\|x_i - x_j\|\right] \geq 1 - e^{-cn}.$$

Since $A = \frac{1}{\sqrt{n}} \cdot G$, it follows that the probability that the bad event

$$E_{ij} = \left\{\left|\|Ax_i - Ax_j\|\right| \notin [0.99\|x_i - x_j\|, 1.01\|x_i - x_j\|]\right\},$$

holds for a fixed pair i, j is at most $\exp(-cn)$.

By a union bound the probability that there is some pair i, j for which the above event E_{ij} holds is atmost

$$\mathbb{P}[\cup_{i,j} E_{ij}] \leq \sum_{ij} \mathbb{P}[E_{ij}] \leq N^2 \exp(-cn).$$

If we choose $n \geq C \log N$ for a large enough constant C, the probability above is at most N^{-100}. Thus, with probability at least $1 - N^{-100}$, the matrix A satisfies the desired property. This completes the proof. □

Prologue. If high dimensional geometry is so different from low-dimensional geometry, why is dimensionality reduction possible? Does Johnson-Lindenstrauss Lemma not tell us that high-dimensional geometry can be approximated in low dimensions? Note that the Johnson-Lindenstrauss Lemma only preserves distances between the N data points $x_1, \ldots, x_N \in \mathbb{R}^d$, not all points in $\mathbb{R}^d$. In particular, we are now using the strange behavior of high-dimensional spaces to our advantage — previous we saw that d-dimensional space can contain $2^{\Theta(d)}$ mutually orthogonal unit vectors. When $d = C \log N$ for a large enough C, this is enough to embed N unit vectors that might be mutually *almost* orthogonal.

One final remark for the interested reader: one may wonder whether we can sample a Gaussian random matrix in practice since it is a continuous distribution. In fact, there are various practical ways of generating samples from a Gaussian distribution that are already widely used (Matlab already does this, for instance). However, it turns out that, with more work, one can even work with a simpler discrete distribution. It is possible to show that even choosing G to be a random ± 1 matrix, i.e. a matrix where each entry is chosen independently to be an unbiased ± 1 random variable, works with high probability. In fact, one can remove the randomness from the above construction in some sense. After a long line of work, there are now even fast deterministic algorithms to find the Johnson-Lindenstrauss matrix A given the data set as input.