

Self-Evaluation Test: Solutions

Instructors: *Alex Andoni, Edo Liberty*

This is a self-evaluation test for you to confirm that you have the sufficient background for the class, and identify potential parts to brush up before the class. You are expected to understand and solve all problems on this test. You do not have to turn in solutions.

The main prerequisite for the class is mathematical maturity, i.e., being able to follow and write rigorous mathematical proofs, of both combinatorial and analytical flavors, at the level of COMS 4231 (Analysis of Algorithms). In terms of concrete topics, you should be comfortable with: linear algebra (eigenvalues/eigenvectors, matrix norms, SVD); probability (linearity of expectation, variance, concentration bounds); and basic algorithmic notions (order notation for runtime, correctness proofs, NP-hardness). No machine learning or coding background is assumed. As a result, the problems below are partitioned by category.

Questions marked with $\star$ are a bit harder — they will be covered in the class (or at that level) but briefly, and hence prior understanding would be very much helpful.

1 Math

1.1 Norms

For a vector $x \in \mathbb{R}^n$, and a real $p \in [1, \infty]$, the p -norm of x is defined as $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$.¹ Prove the following:

- $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$, using Cauchy–Schwarz if needed. For which vectors x is each of the inequalities tight?

– **For $\|x\|_2 \leq \|x\|_1$:**

Let e^i denote the vector of 0s with a unique 1 in the i th coordinate. So, for $x = (x_1, \dots, x_n)$ we have $x = \sum_{i=1}^n x_i \cdot e^i$. Moreover for each product in the sum, $\|x_i \cdot e^i\|_2 = \sqrt{\sum_{j=1}^n |x_i \cdot e_j^i|^2} = \sqrt{0 + \dots + 0 + |x_i|^2 + 0 + \dots + 0} = |x_i|$.

Then, by the triangle inequality:

$$\begin{aligned} \|x\|_2 &\leq \sum_{i=1}^n \|x_i \cdot e^i\|_2 \\ &= \sum_{i=1}^n |x_i| \\ &= \|x\|_1 \end{aligned}$$

¹For $p = \infty$, $\|x\|_\infty$ is formally defined to be $\max_{i \in [n]} |x_i|$, which can be seen as the limit as $p \rightarrow \infty$.

(Alternatively: $\|x\|_1^2 = \sum_i |x_i|^2 + \sum_{i \neq j} |x_i||x_j| \geq \|x\|_2^2$.) The second form shows the inequality is an equality iff $\sum_{i \neq j} |x_i||x_j| = 0$, i.e., iff x has at most one non-zero coordinate, e.g., $x = (a, 0, \dots, 0)$.

– **For** $\|x\|_1 \leq \sqrt{n}\|x\|_2$:

By Cauchy–Schwarz (applied for the third line below):

$$\begin{aligned}\|x\|_1^2 &= \left(\sum_{i=1}^n |x_i| \right)^2 \\ &= \left(\sum_{i=1}^n 1 \cdot |x_i| \right)^2 \\ &\leq \left(\sum_{i=1}^n 1^2 \right) \left(\sum_{i=1}^n |x_i|^2 \right) \\ &= n\|x\|_2^2\end{aligned}$$

Therefore, $\|x\|_1 = \sqrt{\|x\|_1^2} \leq \sqrt{n\|x\|_2^2} = \sqrt{n}\|x\|_2$. Cauchy–Schwarz is an equality iff the two vectors are parallel, i.e., iff $(|x_1|, \dots, |x_n|)$ is a multiple of $(1, \dots, 1)$: the inequality is tight exactly when all $|x_i|$ are equal, e.g., $x = (a, \dots, a)$, for which $\|x\|_1 = n \cdot a$ and $\|x\|_2 = \sqrt{n \cdot a^2} = \sqrt{n} \cdot a$.

• when $p = \log_2 n$, $\|x\|_p$ is a 2 approximation to $\|x\|_\infty$ (i.e., $\|x\|_\infty \leq \|x\|_p \leq 2\|x\|_\infty$).

– **For** $\|x\|_\infty \leq \|x\|_p$:

By definition $\|x\|_\infty = \max_i |x_i|$. Note that $\max_i |x_i|^p \leq \sum_{i=1}^n |x_i|^p$ because all summands are non-negative. Therefore, $(\max_i |x_i|^p)^{1/p} \leq (\sum_{i=1}^n |x_i|^p)^{1/p}$ because the p th root function is monotonically increasing on non-negative numbers. Putting it all together we get,

$$\|x\|_\infty = \left(\max_i |x_i|^p \right)^{1/p} \leq \left(\sum_{i=1}^n |x_i|^p \right)^{1/p} = \|x\|_p.$$

Again, to see this is tight consider $x = (a, 0, \dots, 0) \in R^n$ for any $a > 0$.

– **For** $\|x\|_p \leq 2\|x\|_\infty$:

Notice that $\sum_{i=1}^n |x_i|^p \leq \sum_{i=1}^n (\max_j |x_j|)^p = n\|x\|_\infty^p$. Also, by the definition of p we have $n^{1/p} = (2^{\log_2 n})^{1/p} = (2^p)^{1/p} = 2$.

So,

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p} \leq (n\|x\|_\infty^p)^{1/p} = n^{1/p}\|x\|_\infty = 2\|x\|_\infty$$

★ In the spirit of above, how can you relate the 10-norm of x to the 5-norm of x ?

$\|x\|_{10} \leq \|x\|_5 \leq n^{1/10}\|x\|_{10}$. In general, to prove relations between p -norms, use Hölder’s inequality, a generalization of Cauchy–Schwarz. In this particular case we can just apply the ℓ_1 and ℓ_2 relation for the vector y where $y_i = |x_i|^5$: indeed $\|y\|_1 = \|x\|_5^5$ and $\|y\|_2 = \|x\|_{10}^5$, so $\|x\|_{10}^5 \leq \|x\|_5^5 \leq \sqrt{n}\|x\|_{10}^5$; now take 5th roots.

1.2 Order notation: runtime of Merge-Sort

Merge-Sort is an algorithm for sorting n numbers, which proceeds as follows. Suppose the input is an array $A[1 \dots n]$ of length n (which, for simplicity, assume is a power of 2). We partition the array into two, $A[1 \dots n/2]$ and $A[n/2 + 1 \dots n]$. We sort each part recursively and then merge the 2 (sorted) parts. It is known that the latter merge operation can be accomplished in time $O(n)$.² The base case of the Merge-Sort is when $n = 1$, in which case we do nothing (the array is already sorted).

Here, you are to determine whether the following statement/proof is correct or not (why or why not).

Claim 1. *The runtime of the above MergeSort algorithm is $O(n)$.*

Proof. Let $T(n)$ be the runtime of Merge-Sort on an array of length n . Then, by the above description we have that:

$$T(n) = 2 \cdot T(n/2) + O(n),$$

corresponding to the two recursive calls and the merge operation respectively.

We prove the claim that $T(n) = O(n)$ by induction. Indeed, we have that:

$$T(n) = 2 \cdot T(n/2) + O(n) = 2 \cdot O(n/2) + O(n) = O(n).$$

This completes the proof of the claim. □

The claim and the proof are wrong (in fact, $T(n) = \Theta(n \cdot \log n)$ for Merge-Sort). The error is due to a sloppy use of the $O(\cdot)$ notation.

Using the definition of $O(n)$, the claim asks to prove that: there exist n_0, c such that $T(n) \leq cn$ for $n \geq n_0$. So suppose this were true inductively for $T(n/2)$. In particular, whenever $n \geq 2n_0$, we have $T(n/2) \leq cn/2$. Now using the recurrence we have $T(n) = 2T(n/2) + O(n) \leq 2cn/2 + O(n)$, where $O(n)$ is the runtime of the merge operation, which we take to be, say, $\leq 10n$. While the above sum is indeed $\leq cn + 10n = (c + 10)n = O(n)$, this is not enough for the inductive hypothesis for n ! In particular, the inductive hypothesis is that $T(n) \leq cn$, with the *same* constant c . What the argument really shows is that the implicit constant of the $O(\cdot)$ notation grows with the number of inductive steps — by 10 per level, and there are $\log_2 n$ levels, which is where the $n \log n$ comes from.

2 Linear algebra

2.1 Quadratic forms

Let A be a n -by- n symmetric matrix whose maximum eigenvalue is 42. What is the minimum and maximum possible value for the quadratic form $x^T A x$, where $x \in \mathbb{R}^n$ are unit-norm vectors? What about $x^T A^2 x$?

Recall that a symmetric matrix has a spectral decomposition $A = U \Lambda U^T$, where U is orthogonal ($U^T U = U U^T = I$) and Λ is diagonal, with the (real) eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ on the diagonal; here $\lambda_1 = 42$. For any unit-norm x , the vector $u = U^T x$ is also unit-norm (and any unit-norm u is realizable, as $x = U u$). Therefore, $x^T A x = u^T \Lambda u = \sum_{i=1}^n \lambda_i u_i^2$ is a weighted average of the eigenvalues, with weights $u_i^2 \geq 0$ summing to 1. Hence the maximum value is $\max_i \lambda_i = 42$ (attained at the top

²Remember that $O(f(n))$ is the set of functions $g : \mathbb{N} \rightarrow \mathbb{N}$ such that there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that, for $n \geq n_0$, $g(n) \leq c \cdot f(n)$.

eigenvector, $u = e^1$), and the minimum value is $\min_i \lambda_i = \lambda_n$, which for an arbitrary such A can be any number ≤ 42 (in particular it can be as negative as desired).

For the second part, note that $A^2 = U\Lambda^2U^T$, so $x^T A^2 x = \sum_i \lambda_i^2 u_i^2$ and the same argument applies with the eigenvalues λ_i^2 . The minimum is $\min_i \lambda_i^2 \geq 0$, and the lowest it can ever be is 0 (when A is singular). The maximum is $\max_i \lambda_i^2 = \max\{42^2, \lambda_n^2\}$: it is at least 42^2 , but it can be arbitrarily large, if A has a very negative eigenvalue (e.g., $\lambda_n = -100$ gives a maximum of 10^4).

2.2 Matrix norms and the SVD

Let $A \in \mathbb{R}^{n \times d}$ be a matrix of rank r , with singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$. Recall the singular value decomposition (SVD): $A = \sum_{i=1}^r \sigma_i u_i v_i^T$, where $u_1, \dots, u_r \in \mathbb{R}^n$ are orthonormal and $v_1, \dots, v_r \in \mathbb{R}^d$ are orthonormal. Two standard matrix norms are the spectral (operator) norm $\|A\|_2 = \max_{x \in \mathbb{R}^d: \|x\|_2=1} \|Ax\|_2$, and the Frobenius norm $\|A\|_F = \left(\sum_{i,j} A_{ij}^2\right)^{1/2}$.

- Express $\|A\|_2$ and $\|A\|_F$ in terms of the singular values of A . (Hint: $\|A\|_F^2 = \text{tr}(A^T A)$.)

$$\|A\|_2 = \sigma_1 \text{ and } \|A\|_F = \sqrt{\sum_{i=1}^r \sigma_i^2}.$$

For the spectral norm: for any x , using orthonormality of the u_i 's, $Ax = \sum_i \sigma_i \langle v_i, x \rangle u_i$ and hence $\|Ax\|_2^2 = \sum_i \sigma_i^2 \langle v_i, x \rangle^2 \leq \sigma_1^2 \sum_i \langle v_i, x \rangle^2 \leq \sigma_1^2 \|x\|_2^2$, where the last step uses that the v_i 's are orthonormal (Bessel's inequality; the components of x orthogonal to all v_i 's are in the kernel of A). So $\|A\|_2 \leq \sigma_1$, with equality at $x = v_1$.

For the Frobenius norm: $\|A\|_F^2 = \sum_{i,j} A_{ij}^2 = \text{tr}(A^T A)$, and $A^T A = \sum_{i,j} \sigma_i \sigma_j v_i u_i^T u_j v_j^T = \sum_i \sigma_i^2 v_i v_i^T$ by orthonormality of the u_i 's. Since $\text{tr}(v_i v_i^T) = \|v_i\|_2^2 = 1$, we get $\|A\|_F^2 = \sum_i \sigma_i^2$. (In other words: $\|A\|_F$ is the ℓ_2 norm, and $\|A\|_2$ the ℓ_∞ norm, of the vector of singular values.)

- Prove that $\|A\|_2 \leq \|A\|_F \leq \sqrt{r} \|A\|_2$. When is each inequality tight?

By the previous item, this is exactly the statement $\|\sigma\|_\infty \leq \|\sigma\|_2 \leq \sqrt{r} \|\sigma\|_\infty$ for the vector $\sigma = (\sigma_1, \dots, \sigma_r) \in \mathbb{R}^r$, which is proved as in Problem 1.1. The first inequality is tight iff $r = 1$, i.e., $A = \sigma_1 u_1 v_1^T$ has rank one. The second is tight iff all singular values are equal, $\sigma_1 = \dots = \sigma_r$ (e.g., A is a multiple of an orthogonal matrix, or of an orthogonal projection).

- ★ Fix $k < r$. What is $\min_{B \in \mathbb{R}^{n \times d}: \text{rank}(B) \leq k} \|A - B\|_F$, and which B achieves it? What about the same question for the spectral norm $\|A - B\|_2$?

This is the Eckart–Young(–Mirsky) theorem: the best rank- k approximation, in both norms, is the truncated SVD $A_k = \sum_{i=1}^k \sigma_i u_i v_i^T$, and the minima are $\|A - A_k\|_F = \sqrt{\sum_{i>k} \sigma_i^2}$ and $\|A - A_k\|_2 = \sigma_{k+1}$ (both by the first item, since $A - A_k = \sum_{i>k} \sigma_i u_i v_i^T$ is an SVD).

Here is the proof for the spectral norm. Let B have rank $\leq k$. Then $\ker(B)$ has dimension $\geq d - k$, while $\text{span}(v_1, \dots, v_{k+1})$ has dimension $k + 1$, so the two subspaces of $\mathbb{R}^d$ intersect in a non-zero vector; let x be such a vector with $\|x\|_2 = 1$. Then $\|(A - B)x\|_2^2 = \|Ax\|_2^2 = \sum_{i \leq k+1} \sigma_i^2 \langle v_i, x \rangle^2 \geq \sigma_{k+1}^2 \sum_{i \leq k+1} \langle v_i, x \rangle^2 = \sigma_{k+1}^2$, hence $\|A - B\|_2 \geq \sigma_{k+1}$. The Frobenius case takes a bit more work (see any linear algebra textbook). Low-rank approximation will appear repeatedly in the class (randomized linear algebra, compression of the KV-cache, low-rank fine-tuning).

2.3 Linear systems and least squares

Suppose A is a n -by- n real matrix and $b \in \mathbb{R}^n$. For the unknown vector $x \in \mathbb{R}^n$, describe what are the possible sets of solutions for the following:

- $Ax = b$;

The solution set here is either an affine subspace of $\mathbb{R}^n$ or empty. If $Ax = b$ has at least one solution, $v : Av = b$, (i.e. is consistent), then the solution set is $v + \ker(A)$, an affine subspace of the same dimension as $\ker(A)$ (if $x \in \ker(A)$, then $A(x + v) = Ax + Av = 0 + b$; conversely, if $Ax' = b$ then $x' - v \in \ker(A)$). Otherwise, the system is inconsistent (i.e. $\text{rank}(A) < \text{rank}([A|b])$) and the solution set is simply the empty set.

- $\star \operatorname{argmin}_{x \in \mathbb{R}^n} \|Ax - b\|_2$;

The set $\{Ax : x \in \mathbb{R}^n\} = \text{col}(A)$ of achievable vectors is a linear subspace of $\mathbb{R}^n$. The point of a subspace closest to b is unique: it is the orthogonal projection b^* of b onto $\text{col}(A)$. Hence the set of minimizers is $\{x : Ax = b^*\}$, which, by the previous item, is an affine subspace of dimension $\dim \ker(A)$; in contrast to the previous item, it is never empty (it is a single point iff A is invertible). When $b \in \text{col}(A)$, it coincides with the solution set of $Ax = b$.

- $\star$ let $f(x) = \|Ax - b\|_2^2$. Compute the gradient $\nabla f(x)$. What equation must a minimizer x^* of f satisfy, and how does this relate to the previous item?

Expanding, $f(x) = (Ax - b)^T(Ax - b) = x^T A^T A x - 2b^T A x + b^T b$, so $\nabla f(x) = 2A^T A x - 2A^T b = 2A^T(Ax - b)$. A minimizer must satisfy $\nabla f(x^*) = 0$, i.e., the *normal equations* $A^T A x^* = A^T b$. Since f is convex (its Hessian $2A^T A$ is positive semidefinite), this condition is also sufficient, so the set of minimizers is the solution set of the linear system $A^T A x = A^T b$. This system is always consistent (it is the condition $Ax - b \perp \text{col}(A)$, i.e., $Ax = b^*$), which matches the previous item: an affine subspace of dimension $\dim \ker(A^T A) = \dim \ker(A)$.

To give a better idea of what's asked, consider the " $Ax = b$ " case when $n = 1$ (i.e., just a simple equation). Here's the answer for this case:

The possible sets of solutions are: 1) there may not be a solution (empty set), 2) may be exactly one solution, or 3) any $x \in \mathbb{R}$ is a solution.

The above 3 cases correspond to when 1) $A = 0, b \neq 0$, 2) $A \neq 0$, 3) $A = b = 0$.

3 Probability

3.1 Mean estimation

Let p be a probability distribution over $[0, 1]$. The goal is to estimate the mean μ of the distribution p , up to an additive error ϵ , where $\epsilon \in (0, 1)$ is given in advance. How many samples suffice to output some $\hat{\mu}$ such that $\Pr[|\hat{\mu} - \mu| \leq \epsilon] \geq 0.9$? More generally, how many samples suffice for $\Pr[|\hat{\mu} - \mu| \leq \epsilon] \geq 1 - \delta$, for a given $\delta \in (0, 1)$?

Assuming, of course, that samples are independent, our algorithm will simply be to take the empirical mean of sufficiently many samples. Let X_i be the random variable corresponding to the i th sample.

Then clearly, $\mu = \mathbb{E}[X_i]$ for all i . Let $\hat{\mu} = \frac{\sum_{i=1}^k X_i}{k}$, so that $\mathbb{E}[\hat{\mu}] = \mu$ and, by independence, $\text{Var}[\hat{\mu}] = \text{Var}[X_1]/k \leq 1/k$ (as $X_1 \in [0, 1]$; in fact $\text{Var}[X_1] \leq 1/4$). By Chebyshev's inequality, we have

$$\Pr[|\mu - \hat{\mu}| \geq \epsilon] \leq \frac{\text{Var}[\hat{\mu}]}{\epsilon^2} \leq k^{-1} \cdot \epsilon^{-2}.$$

Hence $k = 10/\epsilon^2$ samples suffice, and, in general, $k = 1/(\delta\epsilon^2)$ samples suffice for failure probability δ . (The dependence on δ can be improved to $O(\log(1/\delta))$, using either a stronger concentration bound — Chernoff/Hoeffding — or a “median trick”; we will see this in the class.)

3.2 Random projections

Let $x, y \in \mathbb{R}^d$ be fixed vectors, and let $g = (g_1, \dots, g_d)$ be a random vector whose coordinates are i.i.d. standard Gaussians, $g_i \sim N(0, 1)$ (recall that $\mathbb{E}[g_i] = 0$, $\mathbb{E}[g_i^2] = 1$, and $\mathbb{E}[g_i^4] = 3$).

- Compute $\mathbb{E}[\langle g, x \rangle]$ and $\mathbb{E}[\langle g, x \rangle^2]$. More generally, compute $\mathbb{E}[\langle g, x \rangle \cdot \langle g, y \rangle]$.

By linearity of expectation, $\mathbb{E}[\langle g, x \rangle] = \sum_i x_i \mathbb{E}[g_i] = 0$. Next, $\mathbb{E}[\langle g, x \rangle \langle g, y \rangle] = \sum_{i,j} x_i y_j \mathbb{E}[g_i g_j]$; for $i \neq j$, independence gives $\mathbb{E}[g_i g_j] = \mathbb{E}[g_i] \mathbb{E}[g_j] = 0$, while $\mathbb{E}[g_i^2] = 1$. Hence $\mathbb{E}[\langle g, x \rangle \langle g, y \rangle] = \sum_i x_i y_i = \langle x, y \rangle$, and in particular $\mathbb{E}[\langle g, x \rangle^2] = \|x\|_2^2$. (Note that this used only that the coordinates have mean 0, variance 1, and are pairwise independent; e.g., random signs $g_i \in \{-1, +1\}$ work too.)

- ★ Take k independent copies $g^{(1)}, \dots, g^{(k)}$ of g , and output $Z = \frac{1}{k} \sum_{j=1}^k \langle g^{(j)}, x \rangle^2$ as an estimate of $\|x\|_2^2$. How large should k be so that $\Pr[(1 - \epsilon)\|x\|_2^2 \leq Z \leq (1 + \epsilon)\|x\|_2^2] \geq 0.9$?

Z is the empirical mean of k independent copies of $Y = \langle g, x \rangle^2$, which has $\mathbb{E}[Y] = \|x\|_2^2$, so this is a mean estimation problem as in 3.1 and we need the variance of Y . We compute $\mathbb{E}[Y^2] = \mathbb{E}[\langle g, x \rangle^4] = \sum_{i,j,k,l} x_i x_j x_k x_l \mathbb{E}[g_i g_j g_k g_l]$. By independence and $\mathbb{E}[g_i] = 0$, the expectation vanishes unless the indices pair up: either all four are equal, contributing $\sum_i x_i^4 \mathbb{E}[g_i^4] = 3 \sum_i x_i^4$, or they form two distinct pairs, which can happen in 3 ways ($i = j \neq k = l$, $i = k \neq j = l$, $i = l \neq j = k$), contributing $3 \sum_{i \neq j} x_i^2 x_j^2$. Together, $\mathbb{E}[Y^2] = 3 \left(\sum_i x_i^2 \right)^2 = 3\|x\|_2^4$, so $\text{Var}[Y] = 2\|x\|_2^4$. (Alternatively: $\langle g, x \rangle$ is a sum of independent Gaussians, hence itself Gaussian, $N(0, \|x\|_2^2)$, whose 4th moment is $3\|x\|_2^4$.)

Hence $\text{Var}[Z] = 2\|x\|_2^4/k$, and Chebyshev gives $\Pr[|Z - \|x\|_2^2| \geq \epsilon\|x\|_2^2] \leq \frac{2\|x\|_2^4/k}{\epsilon^2\|x\|_2^4} = \frac{2}{k\epsilon^2}$, which is ≤ 0.1 for $k = 20/\epsilon^2$. Note that k does not depend on the dimension d : the random projection $x \mapsto \frac{1}{\sqrt{k}}(\langle g^{(1)}, x \rangle, \dots, \langle g^{(k)}, x \rangle)$ reduces the dimension from d to $O(1/\epsilon^2)$ while preserving the norm up to $1 \pm \epsilon$. This is the Johnson–Lindenstrauss lemma (modulo a better tail bound), to be covered in the class.

3.3 Hashing

A function $h : U \rightarrow [m]$, where U is a discrete set (e.g., integers of $O(\log n)$ bits), and $m \in \mathbb{N}$, is called a hash function with m buckets. (One can see this as assigning one of the m buckets to each possible item from U .) Most common hash function is a random hash function: each $h(i)$, $i \in U$, is chosen iid at random from $[m]$.

Below, consider a set $S \subset U$ of size n .

- Let h be a random hash function with $m = n$. What is the expected size of a bucket, i.e., $|h^{-1}(i) \cap S|$ for any $i \in [m]$?

For $x \in S$, let X_x denote the indicator random variable for $h(x) = i$. Because h is random, $\mathbb{E}[X_x] = \Pr_h[h(x) = i] = 1/m$ for all $x \in S$. Moreover, the size of the i th bucket is simply the sum over all X_x 's. Then, via linearity of expectation, the expected load of i is $\mathbb{E}[\sum_{x \in S} X_x] = \sum_{x \in S} \mathbb{E}[X_x] = n/m = 1$.

- A collision is an event where distinct $x, y \in S$ satisfy $h(x) = h(y)$. What is the expected number of collisions among elements of S under h ? (To give an example, if one has precisely three distinct items $x, y, z \in S$ falling into the same bucket, they generate 3 collisions: (x, y) , (y, z) , and (z, x)).

For $x \neq y \in S$, let $Z_{x,y}$ denote the indicator random variable for the event that $h(x) = h(y)$. Then $\mathbb{E}[Z_{x,y}] = \Pr_h[h(x) = h(y)] = 1/m$ — fix any $h(x)$, then because $y \neq x$ and h is a random function there is a $1/m$ probability h maps y to $h(x)$. By linearity of expectation, we can count collisions by simply summing $Z_{x,y}$ over distinct pairs. We have that the expected number of collisions is

$$\mathbb{E}\left[\sum_{\{x,y\} \subseteq S: x \neq y} Z_{x,y}\right] = \sum_{\{x,y\} \subseteq S: x \neq y} \mathbb{E}[Z_{x,y}] = \binom{n}{2}/m = \frac{n(n-1)}{2m}.$$

When $n = m$, this is $\Theta(n)$ collisions.

A family $\mathcal{H}$ of hash functions h is k -wise independent if for any distinct $a_1, \dots, a_k \in U$ and (not necessarily distinct) $q_1, \dots, q_k \in [m]$, we have that

$$\Pr_{h \in \mathcal{H}}[h(a_1) = q_1 \wedge \dots \wedge h(a_k) = q_k] = 1/m^k.$$

Note that the aforementioned random hash function is $|U|$ -wise independent (or, to be more precise, the family $\mathcal{H}$ of *all* functions $h : U \rightarrow [m]$ is $|U|$ -wise independent).

- ★ How do the estimates from above change if our hash function h is chosen from some 2-wise independent hash family $\mathcal{H}_2$?

Nothing changes. Note that in both of the above we are simply using linearity of expectation to reduce computation of the big expectation to the expectation of our indicator variables. So, it is simply a matter of showing that the expectations of X_x and $Z_{x,y}$ are unchanged if h is chosen from a pairwise independent family (instead of uniformly at random). Indeed, summing over the possible values of $h(y)$ for some $y \neq x$, $\mathbb{E}[X_x] = \sum_{j=1}^m \Pr[h(x) = i \wedge h(y) = j] = m \cdot \frac{1}{m^2} = 1/m$, and $\mathbb{E}[Z_{x,y}] = \sum_{i=1}^m \Pr_h[h(x) = h(y) = i] = m \cdot \frac{1}{m^2} = 1/m$.

4 Algorithms

4.1 Matrix products

Let $Q, K, V \in \mathbb{R}^{n \times d}$ be three matrices, with $d \ll n$ (think $n = 10^5$ and $d = 100$). Assume that multiplying an a -by- b matrix with a b -by- c matrix takes $O(abc)$ time (the schoolbook algorithm).

- What is the runtime of computing the n -by- d matrix $(QK^T)V$ by first forming QK^T ? What is the runtime of computing $Q(K^TV)$ instead? (Why are the two results equal?) Which order should one use?

Forming QK^T (an n -by- d times a d -by- n matrix) takes $O(n \cdot d \cdot n) = O(n^2d)$ time, and then multiplying the n -by- n result by the n -by- d matrix V takes another $O(n^2d)$: total $O(n^2d)$ (and $\Theta(n^2)$ memory for the intermediate matrix). In the other order, K^TV (d -by- n times n -by- d) takes $O(nd^2)$ and produces a d -by- d matrix; then $Q(K^TV)$ takes $O(n \cdot d \cdot d) = O(nd^2)$: total $O(nd^2)$, with only $O(d^2)$ extra memory. The two results are equal because matrix multiplication is associative. One should use the second order: it is faster by a factor of n/d (a factor of 1000 in the example), and, more importantly, its runtime is *linear* in n .

- ★ Now suppose we need to compute $\text{softmax}(QK^T)V$, where softmax is applied to each row of the n -by- n matrix $M = QK^T$: row i of M , $(M_{i1}, \dots, M_{in})$, is replaced by $(e^{M_{i1}}, \dots, e^{M_{in}}) / \sum_{j=1}^n e^{M_{ij}}$. Does the faster order from the previous item still apply? Why or why not? (This is the attention mechanism of transformers; computing it fast is a topic of the class.)

No. The trick relied on associativity, i.e., on QK^T being used *linearly*. The softmax is a non-linear function applied to the entries of $M = QK^T$: the resulting n -by- n matrix $\text{softmax}(M)$ is generically of full rank n , so it is no longer a product of an n -by- d and a d -by- n matrix, and there is nothing to re-associate. The straightforward algorithm therefore forms all n^2 entries of M , in time $\Theta(n^2d)$: quadratic in the sequence length n , which is the central computational bottleneck of transformers. In the class we will see that (under standard complexity assumptions) this is unavoidable in general for exact computation, and how to approximate the attention matrix in time near-linear in n (e.g., by approximating the kernel $e^{\langle q, k \rangle}$ by a low-rank map to recover associativity, or by using nearest neighbor search / sampling to find the few large entries of each row). Dropping the exponential altogether, to regain associativity, is called “linear attention”.

4.2 Sampling from a discrete distribution

You are given n non-negative weights $w_1, \dots, w_n$ (e.g., $w_i = e^{s_i}$ for some scores s_i , as in the last layer of a language model), with $W = \sum_{i=1}^n w_i > 0$. The task is to draw a random index $I \in [n]$ with $\Pr[I = i] = w_i/W$, repeatedly (many times, from the same weights). Assume that you can draw a uniformly random real number in $[0, 1]$ in $O(1)$ time, and that arithmetic on real numbers takes $O(1)$ time.

- Give an algorithm that, after $O(n)$ preprocessing, draws each sample in $O(\log n)$ time. Prove that it outputs I with the correct distribution, and analyze its runtime.

Preprocessing: compute the prefix sums $S_0 = 0$ and $S_i = S_{i-1} + w_i$ for $i = 1, \dots, n$ (so $S_n = W$), in $O(n)$ time. *Sampling:* draw U uniform in $[0, 1]$, set $u = U \cdot W$, and output the smallest index i with $S_i > u$, found by binary search over the array S .

Correctness. The half-open intervals $[S_{i-1}, S_i)$, for $i = 1, \dots, n$, partition $[0, W)$, and u is uniform on $[0, W)$. Since S is non-decreasing, the set $\{i : S_i > u\}$ is a suffix of $[n]$, so its smallest element is well defined and is found correctly by binary search; moreover it is the unique i with $S_{i-1} \leq u < S_i$. Hence $\Pr[I = i] = \Pr[u \in [S_{i-1}, S_i)) = (S_i - S_{i-1})/W = w_i/W$, as required. (If $w_i = 0$, the interval is empty and i is never output, as it should be.)

Runtime. $O(n)$ preprocessing, and $O(\log n)$ per sample for the binary search, plus $O(1)$ for drawing U .

- ★ Suppose that between samples the weights change: an update sets $w_i \leftarrow w'_i$ for a single index i (and, correspondingly, W changes). Can you support both updates and samples in $O(\log n)$ time each?

Yes. Updating a prefix-sum array is too slow (up to n entries change), so instead store the weights at the leaves of a balanced binary tree (say a complete binary tree with n leaves, in left-to-right order, of depth $\lceil \log_2 n \rceil$), and at each internal node store the sum of the weights in its subtree; the root stores W . *Update:* change the leaf i and recompute the sums along the path from the leaf to the root: $O(\log n)$. *Sample:* draw u uniform in $[0, W)$ and walk down from the root: at a node whose left child has sum L , if $u < L$ go left, otherwise set $u \leftarrow u - L$ and go right; output the leaf reached. This is the same binary search as before, performed on the implicit prefix sums, so its correctness follows from the previous item (formally: by induction, when at a node with subtree weights $w_a, \dots, w_b$, the current u equals the original u minus S_{a-1} , and the leaf reached is the one with $S_{i-1} \leq u < S_i$). Each sample takes $O(\log n)$. (A Fenwick tree achieves the same with $O(n)$ memory and simpler code. For static weights, the *alias method* even achieves $O(1)$ per sample after $O(n)$ preprocessing.)

4.3 NP-hardness

Let P be an optimization problem (say, a minimization problem) that is NP-hard, and assume $P \neq \text{NP}$. For each of the following statements, say whether it follows from these assumptions, and briefly why or why not.

1. There is no polynomial-time algorithm that solves P exactly on all inputs.

Follows. This is essentially the definition: P being NP-hard means that every problem in NP reduces to P in polynomial time, so a polynomial-time algorithm for P would give a polynomial-time algorithm for every problem in NP, i.e., $P = \text{NP}$.

2. There is no polynomial-time algorithm that solves P exactly on some restricted class of inputs (e.g., inputs with a special structure).

Does not follow. NP-hardness is a worst-case statement over all inputs; a restricted class of inputs can be easy. For example, Vertex Cover is NP-hard, but is solvable in polynomial time on bipartite graphs (and trivially on trees); SAT is NP-hard, but 2-SAT is in P. (Trivially, any finite class of inputs is “easy” too.)

3. There is no polynomial-time algorithm that, on every input, outputs a solution whose cost is at most twice the optimum.

Does not follow. NP-hardness of computing the exact optimum says nothing by itself about approximating it. Many NP-hard problems admit polynomial-time constant-factor approximations (Vertex Cover has a simple 2-approximation, and Knapsack has a $(1 + \epsilon)$ -approximation for every fixed $\epsilon > 0$), while for others even a coarse approximation is NP-hard (e.g., approximating Max Clique within a factor $n^{1-\epsilon}$). Which case one is in requires a separate theorem (a “hardness of approximation” result); such results will appear in the class.

4. Every algorithm that solves P exactly on all inputs takes time $2^{\Omega(n)}$ on some inputs of size n .

Does not follow. $P \neq NP$ rules out only polynomial-time algorithms; it does not rule out, say, an $n^{O(\log n)}$ -time (quasi-polynomial) or $2^{O(\sqrt{n})}$ -time algorithm for P . Exponential lower bounds require stronger hypotheses (such as the Exponential Time Hypothesis). Note also that the direction “NP-hard problems can be solved in exponential time” is usually true and easy (brute force); the statement is about the impossibility of anything faster, which is the non-trivial direction.

(Several problems in the LLM pipeline are NP-hard — for example, choosing an optimal vocabulary for a tokenizer — and are handled in practice by heuristics or approximation algorithms. The distinctions above will be a recurring theme in the class.)

4.4 ★ Alice tells Bob

Suppose Alice holds a number n and wants to communicate it to Bob *approximately* using minimal possible communication. In particular, Bob should be able to output some n' satisfying $n \leq n' \leq 2n$. How many bits does Alice need to send to Bob to achieve this goal? (Answer up to $O(\cdot)$ is enough.)

The answer is $O(\log \log n)$ bits suffice as shown by the following protocol.

- **Alice:** On input n , send the index of the most significant bit of n , $\text{msb}(n)$ (i.e. the unique number m such that $2^m \leq n < 2^{m+1}$). (Note that the length of the binary representation of n is at most $\log n$. Therefore, the largest index of this representation can be represented with $\log \log n$ bits.)
- **Bob:** Upon receiving a binary representation of a number m , output 2^{m+1} . (Indeed $n \leq 2^{m+1} \leq 2n$.)

(This is also optimal for deterministic protocols: the inputs $1, 4, 16, \dots, 4^j, \dots$ (for $j \leq \log_4 n$) pairwise require different outputs from Bob — an output valid for input 4^j lies in $[4^j, 2 \cdot 4^j]$, and these intervals are disjoint — and hence different messages from Alice; so Alice needs at least $\log_2(\log_4 n) = \Omega(\log \log n)$ bits.)

Does the answer change if it’s a “dialogue” — Alice and Bob sequentially communicate to each other a number of bits (think of it as if Bob can “ask questions”) — and the “total communication” is the total number of bits exchanged ?

Interaction does not help in this case and the complexity remains the same because Alice can simply simulate a virtual Bob in her head (because Bob has no input, so Alice knows everything about him). In particular, suppose there was a cheaper (deterministic) interactive protocol. Then, we can compile this into a non-interactive protocol by having Alice simulate the entire interaction and send the transcript to Bob. Bob then plays the transcript for himself and outputs in accordance with the interactive protocol.