The Median is Easier than it Looks: Approximation with a Constant-Depth, Linear-Width ReLU Network

Abhigyan Dutta ABHIGYAND@PURDUE.EDU
Purdue University

Itay Safran SAFRANI@BGU.AC.IL
Stein Faculty of Computer and Information Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Paul Valiant PVALIANT@PURDUE.EDU
Purdue University

Editors: Steve Hanneke and Tor Lattimore

Abstract

We study the approximation of the median of d inputs using ReLU neural networks. We present depth-width tradeoffs under several settings, culminating in a constant-depth, linear-width construction that achieves exponentially small approximation error with respect to the uniform distribution over the unit hypercube. By further establishing a general reduction from the maximum to the median, our results break a barrier suggested by prior work on the maximum function, which indicated that linear width should require depth growing at least as $\log \log d$ to achieve comparable accuracy. Our construction relies on a multi-stage procedure that iteratively eliminates non-central elements while preserving a candidate set around the median. We overcome obstacles that do not arise for the maximum to yield approximation results that are strictly stronger than those previously known for the maximum itself.

Keywords: Deep learning theory, Neural network approximation, ReLU neural network, Depth separations, Maximum, Median, Lower bounds, Upper bounds

1. Introduction

Neural networks have become one of the most prominent tools in machine learning in recent years. Their success is often attributed, among other factors, to their expressive power (Cybenko, 1989; Hornik et al., 1989; Leshno et al., 1993). A popular line of theoretical work, aimed at understanding the role of depth, studies depth-width tradeoffs in approximation capabilities (Telgarsky, 2016; Eldan and Shamir, 2016; Arora et al., 2016; Yarotsky, 2017; Liang and Srikant, 2017; Safran and Shamir, 2017; Safran et al., 2019; Venturi et al., 2021; Safran et al., 2024b,a), showing that increased depth can lead to significantly smaller overall network size. Despite the variety of settings in which depth-width separation results have been established, the assumptions underlying these results and the target functions used are often tailored to technically facilitate the analysis, as handling more natural settings is typically much harder. Motivated by this, recent work has shifted focus to more naturally occurring target functions. Among these, the maximum function (Mukherjee and Basu, 2017; Hertrich et al., 2021; Matoba et al., 2022; Haase et al., 2023; Safran et al., 2024b; Bakaev et al., 2025a; Averkov, 2025; Grillo et al., 2025; Safran, 2026) has received significant attention in recent years, as it plays a pivotal role in many areas of machine learning. While it has long been known that the maximum of d inputs can be computed exactly by a neural network of depth logarithmic in d (Arora et al., 2016; Bakaev et al., 2025b), it remains an open question whether this

function can be realized by a shallower ReLU network without imposing any restrictions on the approximating architecture (Hertrich et al., 2021; Haase et al., 2023; Bakaev et al., 2025a; Averkov, 2025; Grillo et al., 2025).

Despite the recent focus on the precise computation of the maximum function, it is arguably more interesting from a practical perspective to study whether a given target function can be approximated (in an L_2 sense, with respect to some underlying input distribution) rather than computed exactly. This approach is better aligned with machine learning applications, since achieving good generalization typically only requires a small approximation error. Moreover, practical learning algorithms, such as gradient descent, only find approximate local minima of the objective rather than converge to its exact value. The recent work of Safran et al. (2024b) studies the depth-width trade-offs in *approximating* the maximum function with respect to continuous distributions. One of their contributions is a construction that requires only linear width and depth $\mathcal{O}(\log\log d)$ to approximate the maximum, but necessitates super-linear width for shallower architectures, suggesting a potential barrier to achieving a similar approximation with linear width at smaller depth. This non-constant depth requirement is further strengthened by the recent result in Safran (2026), where it is shown that *exact* computation of the maximum indeed requires super-linear width for constant depth.

Apart from the maximum function, there is a vast literature studying more general continuous piecewise linear functions (CPWL) (Arora et al., 2016; He et al., 2020; Hertrich et al., 2021; Chen et al., 2022). One such prototypical example is the *median* function, whose computation is a more general and challenging problem than the maximum, since, unlike the maximum, the median of medians of a partition of the input is not necessarily the median of the original input—a property commonly leveraged to construct efficient approximations of the maximum. Moreover, as pointed out earlier, computing the maximum using a neural network with size linear in d is relatively straightforward, but there are currently no known constructions that achieve the same size for the median function. Furthermore, for sufficiently large d, there is no known construction of depth $c\log_2(d)$, even with c in the thousands, that computes the median with polynomial size. While it is a classic result that algorithms exist for computing the median in linear time (Blum et al., 1973), it is not clear how such an algorithm could be implemented using a small-size neural network. Moreover, existing Boolean circuit lower bounds for the majority function imply[1] super-polynomial Boolean circuit size lower bounds for approximating the median with constant depth (O'Donnell and Wimmer, 2007). This naturally raises the following question: can a linear-sized neural network approximate the median function?

In this paper, we study how well ReLU networks can approximate various CPWL functions with respect to the uniform distribution over the unit hypercube. Specifically, we focus on approximating the rank-k element of a d-dimensional input, which includes both the maximum and the median as special cases. We prove various depth-width trade-offs for approximating the rank-k function with respect to the uniform distribution on the unit hypercube, showing that increasing depth reduces the width required and culminating in a perhaps surprising linear-width, *constant*-depth construction. Our proof is technically involved and requires several intermediate steps, some of which may be of independent interest. In particular, we show that it is possible to repeatedly estimate a window around the rank-k element while zeroing out all elements outside this window. After four such iterations, the set of non-zero elements, which contains the true rank-k element with overwhelming probability, becomes small enough to allow a novel implementation of a hashing trick to extract

1. This can be shown by a simple reduction: when all inputs are binary, the median of the input coincides with the majority function.

the rank-k element. As a special case with $k = 1$, this breaks the barrier suggested by prior work (Safran et al., 2024b; Safran, 2026). Additionally, we establish the first linear-sized approximation for the median. This result demonstrates that relaxing the precision requirement to a practical error threshold enables significantly more compact representations.

The rest of this paper is structured as follows: After presenting our main contributions in further detail below, we turn to review additional related work in the literature. In Section 2, we introduce our notation and framework. In Section 3 we present our upper bound constructions, and Section 4 specifies complementary lower bounds. Finally, in Section 5 we summarize this paper and detail potential future work directions.

Our contributions

- We provide a ReLU network construction with depth 46 and width linear in d that approximates the median function with respect to the uniform distribution on the unit hypercube (Theorem 4). Specifically, this construction achieves an accuracy that is exponentially small in d for sufficiently large weights.

- We prove that for any k, there exists a depth-3 ReLU network with width $\mathcal{O}(d^2)$, that approximates the rank-k element to arbitrary accuracy with respect to the uniform distribution over the unit hypercube, provided the network weights are sufficiently large (Theorem 2).

- We show that a mild increase in depth can result in a noticeable improvement in the width requirement for computing any rank. For any k, there exists a ReLU network with depth-5 and width roughly $\mathcal{O}(d^{5/3})$ that approximates the rank-k element of the input to exponentially small target accuracy, provided that the weights are sufficiently large (Theorem 3).

- We provide a reduction from the exact computation of any rank-k function to the exact computation of the maximum. By reducing to the exact lower bound in Safran (2026), we derive an exact computation lower bound for computing the median (Theorem 7). By combining this with our approximation result in Theorem 4, we demonstrate a separation between the exact and approximate computation regimes. This formalizes the gap between these two settings, showing that obtaining an approximation is a significantly less stringent requirement than computing the function exactly.

- Finally, we introduce a general reduction scheme from the median function to the maximum, enabling approximation lower bounds for the maximum to directly imply corresponding bounds for the median (Theorem 8). Specifically, by applying this reduction to existing results in Safran et al. (2024b), we derive lower bounds for approximating the median at depths 2 and 3 (Corollaries 11 and 13, respectively). Furthermore, we obtain a general lower bound for arbitrary depth, establishing that the required width must be at least linear in d (Theorem 14).

Additional related work

L_2 approximation of the maximum. The work most closely related to ours is Safran et al. (2024b), which establishes lower and upper bounds for approximating the maximum function with respect to the uniform distribution on the unit hypercube. Their constructions rely on the idempotent

property of the maximum where the maximum of maxima of subsets equals the global maximum. Since this property fails to hold for the median, we introduce a more intricate multi-stage construction that yields an efficient approximation for the median and consequently for the maximum. Notably, our approach improves upon the upper bounds in Safran et al. (2024b) within the regimes we consider. While our results are restricted to an accuracy threshold that is exponentially small in the input dimension, we emphasize that this setting encompasses the error regimes typically encountered in practical applications and captures the most relevant approximation scenarios.

Exact computation of CPWL functions. The exact computation of CPWL functions has a rich literature. Based on the fact that any CPWL function can be expressed as a nested sum of minima and maxima of affine transformations (Ovchinnikov, 2000; Wang and Sun, 2005), Arora et al. (2016) show that depth $\lceil \log d \rceil + 1$ suffices to compute any CPWL function on $\mathbb{R}^d$. A subsequent line of work has improved the required network complexity, either by reducing the depth (Bakaev et al., 2025b) or by decreasing the width based on the geometric properties of the target function (He et al., 2020; Hertrich et al., 2021; Chen et al., 2022). Notably, these upper bounds depend on the number of convex regions in the function's polyhedral decomposition. While this quantity is exactly d for the maximum, for the median it is a combinatorial quantity that grows exponentially with d, rendering such architectures prohibitively large.

A related line of work on sorting networks (Knuth, 1998) provides constructions that imply the exact computation of the median with ReLU networks. Using depth that scales as $\mathcal{O}(\log d)$ with a large constant c or as $\mathcal{O}(\log^2 d)$, there exist networks with linear width that compute the median (Batcher, 1968; Ajtai et al., 1983). However, because these constructions require either significantly more depth than the maximum or rely on prohibitively large constants, they remain far less efficient than current architectures for the maximum.

In light of the above, to achieve efficient median computation with constant depth and linear width, it is natural to relax the approximation requirement. We show that even demanding exponentially small accuracy results in a dramatic improvement over the exact computation bounds discussed above, bypassing the logarithmic depth barriers inherent to the exact regime.

2. Preliminaries and notation

Notations. We use bold-faced letters to denote vectors: $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$. We use the shorthand $[n] \coloneqq \{1, 2, \ldots, n\}$. We use $\max(\mathbf{x})$ and $\mathrm{med}(\mathbf{x})$ to denote the *maximum* and *median* of the entries in the vector $\mathbf{x}$, respectively. Given a set S, we denote by $\mathcal{U}(S)$ the uniform distribution over S. Given a vector $\mathbf{x} \in \mathbb{R}^d$, we denote its rank-$k$ element using $\mathcal{R}_k(\mathbf{x})$—namely, the k^{th} entry when $\mathbf{x}$ is sorted in ascending order. All logarithms have base e unless otherwise stated.

Neural networks. Throughout, we use the notation $[z]_+ = \max\{0, z\}$ for the ReLU activation function. In our work we consider fully connected, feed-forward neural networks; i.e., each neuron in the network computes some non-linear activation function σ. When σ is restricted to a certain class of activation functions or specifically a ReLU, we explicitly mention this in the text. A depth-h Neural network consists of h hidden layers of neurons, followed by the output neuron which computes an affine transformation of its inputs. Each hidden layer computes an affine transformation of its inputs and then applies its activation function separately on each coordinate before propagating the output forward to the next hidden layer. The *depth* of a neural network is the number of hidden layers in it plus one. The *width* of a network is defined as the number of neurons in the largest

hidden layer. Finally, the *size* of a neural network is defined as the overall number of neurons across all layers.

Approximation error. We focus solely on a regression setting, where a neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$ computes a real-valued function of its input. We use the square loss throughout, and the approximation error is measured with respect to an underlying distribution $\mathcal{D}$, supported in $\mathbb{R}^d$. While our results are presented mainly for $\mathcal{D} = \mathcal{U}\left([0,1]^d\right)$, we define our approximation scheme in a more general manner, as our results can be extended to hold for a broader family of continuous distributions. Formally, given a neural network $\mathcal{N}$, a target function $f : \mathbb{R}^d \to \mathbb{R}$ to be approximated, and a data distribution $\mathcal{D}$, our approximation error is the expected square loss given by

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}}\left[\left(\mathcal{N}(\mathbf{x}) - f(\mathbf{x})\right)^2\right].$$

3. Neural network approximation for the median function

In this section, we present our upper bounds for the approximation of the median function. As previously discussed, we establish tradeoffs between width and depth, showing that increasing depth allows a reduction in the required width.

Remark 1 (Input distribution) *We emphasize that our constructions are effective for a significantly broader family of continuous distributions. Given that our analysis primarily requires input permutation invariance and sufficiently distinct values, extending these upper bounds to hold for any i.i.d. and bounded continuous distribution is not too difficult: the neural networks do not need to know the input distribution, only bounds on its range and density. However, to maintain clarity within an already technically demanding analysis, we focus on the uniform distribution to provide a more accessible exposition of the core mechanisms.*

3.1. Depth 3 and width $\mathcal{O}(d^2)$ median computation

We start with the simplest positive approximation result, which allows the approximation of the median using depth 3 and width $\mathcal{O}(d^2)$ when the data distribution is uniform over the unit hypercube. This result not only provides us with a constructive method for approximating the median using depth 3, it also gives a flavor of the arguments we use in our main result which build upon it, as this architecture is pivotal in building up more sophisticated constructions to extract the median.

Theorem 2 *For any dimension d and any target accuracy $\epsilon > 0$, there exists a ReLU neural network $\mathcal{N}$ of depth 3 and width $\mathcal{O}\left(d^2\right)$, and magnitude of weights bounded by $\frac{12d^4}{\epsilon}$, such that*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] \leq \epsilon.$$

The above construction, whose proof is available in Appendix B, can be seen as a generalization of Safran et al. (2024b, Theorem 3.3). It not only allows the extraction of the median element, but any rank $k \in [d]$ element, which includes the maximum as a special case, while also using the same network architecture. The main intuition behind the construction is that we can use the first hidden layer to compute pairwise indicators, comparing all pairs of inputs (hence the quadratic width requirement). Thereafter, the second hidden layer aggregates these results, identifies the

entry whose indicators sum to the desired rank, and then outputs it. We remark that more generally, this implies a depth-3 width-$\mathcal{O}(d^2)$ construction with d outputs that sorts all inputs, which is a fundamental building block in the constructions that we present next.

3.2. Depth 5 and width roughly $\mathcal{O}(d^{5/3})$ median computation

The following result demonstrates that even a small increase in depth from 3 to 5 can result in a meaningful reduction in the required width for approximating the median. Specifically, the reduction in width is from a quadratic to $\mathcal{O}(d^{5/3+\gamma})$, where $\gamma > 0$ is a confidence parameter that can be made arbitrarily small, at the cost of an (exponentially negligible) additional approximation error. More formally, our result is the following.

Theorem 3 *For any dimension d, any target accuracy $\epsilon > 0$, there exists a ReLU neural network $\mathcal{N}$ of depth 5 and width $\mathcal{O}(d^{5/3+\gamma})$, and with magnitude of weights bounded by $\frac{12d^6}{\epsilon}$, such that*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] \leq \epsilon + \exp\left(-\Omega\left(d^{2\gamma}\right)\right).$$

The proof of the theorem, which appears in Appendix C, relies on a probabilistic argument. Our architecture utilizes the first two hidden layers to partition the input into batches and compute their central elements using the construction devised in the previous subsection. Because a sufficiently large sample of central elements is exponentially likely to capture the global median, and because the resulting candidate set is significantly smaller than d, the next step employs two additional hidden layers to compare all candidates against all inputs. This identifies the true median while maintaining a width strictly smaller than quadratic. The parameter γ controls the confidence of this construction's success, which leads to the inevitable exponentially small error floor.

We remark that our construction carries a width requirement with an exponent of at least $5/3$, representing a fundamental limitation of the current approach that cannot be further improved. In contrast, the analogous result for the maximum in Safran et al. (2024b, Theorem 3.4) requires an exponent of only $4/3$. This suggests that while the median exhibits depth-width tradeoffs similar to those of the maximum, its required width appears to scale more steeply. This naturally raises the question of whether the median can be approximated by a depth-$\mathcal{O}(\log\log d)$, linear-width network, as is possible for the maximum function. In the next subsection, we show, perhaps surprisingly, that not only is linear width achievable, but it can even be attained using *constant* depth.

3.3. Depth $\mathcal{O}(1)$ and width $\mathcal{O}(d)$ median computation

The following is our main result in this paper.

Theorem 4 *For any dimension d and any target accuracy $\epsilon > 0$, there exists a ReLU neural network $\mathcal{N}$ of depth 46 and width $\mathcal{O}(d)$, with magnitude of weights bounded by $\mathcal{O}\left(\frac{d^2}{\epsilon}\right)$, such that*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] \leq \epsilon + \exp\left(-d^{\Omega(1)}\right).$$

The proof of the above theorem, which appears in Appendix D, is technically involved and requires several intermediate constructions. Given the inherent complexity of the proof, we have

prioritized clarity of exposition over obtaining a tighter depth analysis, and therefore we did not attempt to optimize the depth, which can likely be further improved to a constant smaller than 46.

At a high level, the proof relies on an iterative procedure that partitions the current elements into batches and propagates only their central elements, effectively zeroing out inputs unlikely to be the median. This process continues until no more than d^α (for a sufficiently small $\alpha > 0$) of the original inputs remain non-zero. While sorting these remaining elements using the construction from Theorem 2 would ideally yield the median, a significant implementation hurdle remains: ReLU networks cannot selectively sort only non-zero values. To circumvent this, we employ a novel hashing scheme that maps the non-zero elements into a reduced-dimension vector where they are guaranteed to be distinct with high probability. By subsequently sorting this hashed vector, we can extract the true median.

3.3.1. PROOF SKETCH OF THEOREM 4—SPECIALIZED FOR THE MAXIMUM

In this subsection, we provide a more detailed yet accessible proof sketch of our main result. To illustrate our primary construction and implementation techniques, we present the argument for the simpler special case of extracting the maximum. Focusing on the maximum allows us to bypass some of the technical intricacies unique to the median, while still demonstrating the most significant and novel elements of the proof. This approach highlights how our method overcomes existing depth barriers that previously suggested a $\log \log d$ requirement for such architectures. See Figure 1 for a visual illustration of the stages of the construction.

For simplicity of presentation, it will be more convenient to describe an architecture as an algorithm, deferring implementation considerations with ReLU networks. We point out, however, that all neural network constructions used and implemented in our proofs are rigorously analyzed in Appendix G. Additionally, we make the following remark regarding our terminology and usage of randomness within our neural network constructions.

Remark 5 (Random selection within ReLU neural networks) *The algorithms in this section are described as though they have access to randomness—despite neural networks being deterministic objects. Our ultimate neural network constructions will "derandomize" the ideas of this section, sometimes by leveraging randomness coming from the input distribution of the neural network, and sometimes via explicit derandomization. Deriving our final linear-width high-probability bounds for the median involves subtle probabilistic analysis of the distribution of intermediate network data. However, for clarity of exposition, this section will describe the main (randomized) algorithms ideas without reference to these analytical hurdles.*

Sparsification step. The first step in our construction is the sparsification process which is described in Algorithm 1, where a significant portion of the lower-rank inputs are zeroed out.

Unlike our construction for the median which requires four iterations of the sparsification loop, the maximum only requires two. In the first iteration, we randomly select[2] a subset of the inputs of size $\alpha_1 = d^{0.5}$ and extract its maximum using two hidden layers with width linear in d. We then utilize an additional hidden layer with linear width to zero out all elements smaller than this approximate maximum. In expectation, this would zero out all but $d^{0.5}$ elements. Next, we seek

2. We emphasize that while ReLU networks are inherently deterministic, the required randomness is induced by our distributional assumptions. Consequently, the term 'random selection' refers to a deterministic architecture choosing an arbitrary subset of the input. See Remark 5 for further discussion.

Algorithm 1: Sparsification step for finding the maximum

Input: Vector: $\mathbf{x}_1 \in (0,1)^d$ with unique entries
Output: Modified Vector $\mathbf{x}_3$
`sparsifyingMax(`$\mathbf{x}_1$`)`:

1. For $i \in \{1,2\}$ define the parameters $\alpha_1 = d^{0.5}, \alpha_2 = d^{0.4}$.
2. For $i \in \{1,2\}$ do:
   (a) From $\mathbf{x}_i$ pick a random sample of α_i non-zero entries, denoted S_i.
   (b) Define $m_i = \max(S_i)$.
   (c) Create a new copy of $\mathbf{x}_i$ represented as $\mathbf{x}_{i+1}$, where in $\mathbf{x}_{i+1}$ all entries with values $< m_i$ are changed to 0.

to extract α_2 non-zero elements from the remaining entries. However, unlike the first iteration, we now have a sparsified vector, and sampling a non-zero subset from it is not straightforward. To overcome this obstacle, we require designing a new neural network, which we call the *non-zero element shortlisting* neural network (see Definition 47). It would be ideal to have $\alpha_2 = d^{0.5}$, since then we would hope to extract all the remaining non-zero elements and immediately find the overall maximum; however, the *non-zero element shortlisting* architecture would require superlinear width. Instead, we can extract $\alpha_2 = d^{0.4}$ elements in linear width, with high probability over a random permutation of the input, by chopping the input into small blocks, relying on concentration bounds on the number of non-zero elements in each block, to repeatedly run *non-zero element shortlisting* neural network on these blocks.

The second sparsification iteration leaves us with a sparsified vector of d entries, roughly $d^{0.1}$ of which are non-zero. In what follows, this extreme sparsity will enable us to explicitly extract these non-zero entries using a different approach.

Hashing step. In this final stage, we begin with a sparse vector $\mathbf{x}'$ containing approximately $d^{0.1}$ non-zero entries. If we could retain these non-zero entries but in a vector of size $\leq d^{0.5}$, then we could use a quadratic-width maximum architecture to finish the algorithm; so our goal here is to "hash down" the d input locations into $\leq d^{0.5}$ locations, so as not to produce any collisions between non-zero entries of $\mathbf{x}'$. One naive attempt is to partition the vector $\mathbf{x}'$ into blocks of size $d^{0.5}$ and sum the values within each batch to produce a lower-dimensional vector of size $d^{0.5}$, since in a randomly ordered vector, the $d^{0.1}$ non-zero locations will typically be spread out. If each block contains at most one non-zero coordinate, this reduced vector preserves the maximum. However, the probability that some block contains multiple non-zero values is inverse polynomial, which is not small enough for our aims.

To guarantee success, we implement a universal hashing scheme that maps the indices of $\mathbf{x}'$ to discrete bins. Because a single hash function may still fail, our architecture implements every element of a universal family of hash functions in parallel. We prove that for any sparse $\mathbf{x}'$, there exists at least one function in this family that maps each non-zero entry to a unique bin. We then employ a counting and masking procedure to identify this successful instance: the network counts the non-zero entries in $\mathbf{x}'$, and compares this to the non-zero counts of each hashed block. A match indicates

Algorithm 2: Hashing step for finding the maximum

Input: Sparse vector $\mathbf{x}'$
Output: $\max(\mathbf{x}')$
`sparseToMaximum`$(\mathbf{x}', p)$:

1. Use all hash functions $h : \mathbb{R}^d \to \mathbb{R}^p$ in $\mathcal{H}$ to map $\mathbf{x}'$.
2. Identify a collision-free $h \in \mathcal{H}$ and set $\mathbf{x}'' = h(\mathbf{x}')$.
3. Return $\max(\mathbf{x}'')$ by direct comparisons.

a collision-free hash, and the corresponding low-dimensional output $\mathbf{x}''$ is propagated. Finally, we extract the maximum from $\mathbf{x}''$ using a brute-force comparison that now requires only linear width due to the significantly reduced dimension. This hashing process is outlined in Algorithm 2.

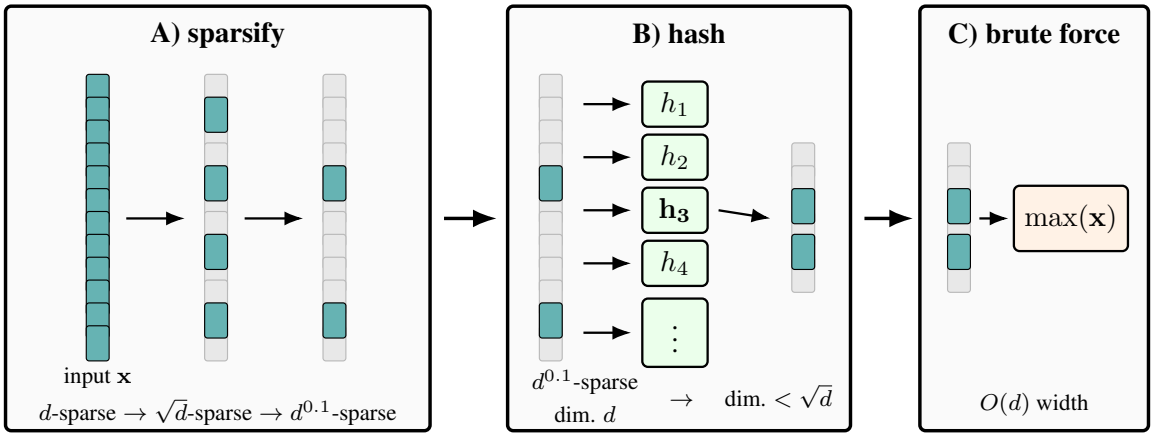

Figure 1: A visual illustration of the major steps in our constant depth and linear width construction. A) We sparsify the input until there are only $d^{0.1}$ non-zero entries remaining; B) we then lower the dimension of vector to be at most $\sqrt{d}$, while maintaining the maximum of the input; C) finally, we brute-force compute the maximum of the low-dimensional vector using width linear in d.

4. Lower bounds for computing the median

Having established upper bounds in the previous section, we now turn to the study of lower bounds. While this section, as with the rest of the paper, is primarily concerned with approximation results for the median function, we begin by considering the simpler case of exact computation. Specifically, we present a reduction from a recent exact computation result in Safran (2026). This approach allows us to illustrate the core intuition behind our reduction technique while bypassing the technical complexities inherent in the approximation setting, which we will delve into later in this section. Formally, we build upon the following result, stated here for completeness.

Theorem 6 (Safran (2026, Theorem 1.1)) *Suppose that $3 \leq k \leq \log_2 \log_2 d$. Let $\mathcal{N}$ be a depth-k ReLU network such that $\mathcal{N}(\mathbf{x}) = \max(\mathbf{x})$ for all $\mathbf{x} \in [0,1]^d$. Then, $\mathcal{N}$ has width at least*

$$\frac{1}{10} d^{1+\frac{1}{2^{k-2}-1}}.$$

With the above, we are able to establish the following depth hierarchy lower bound for the r-rank function:

Theorem 7 *Suppose that $r \in [d]$ and $3 \leq k \leq \log_2 \log_2 d$. Let $\mathcal{N}$ be a depth-k ReLU network such that $\mathcal{N}(\mathbf{x}) = \mathcal{R}_r(\mathbf{x})$ for all $\mathbf{x} \in [0,1]^d$. Then, $\mathcal{N}$ has width at least*

$$\frac{1}{40} d^{1+\frac{1}{2^{k-2}-1}}.$$

The above is proven via a direct reduction from the maximum function to the r-rank function, the details of which are provided in Appendix E.1. To provide intuition for this reduction, we focus on the median for simplicity. Suppose we are given a neural network architecture capable of computing the median for any input dimension d. For a given d-dimensional input, we can employ a $(2d-1)$-dimensional version of this architecture and pad the input with $d-1$ coordinates fixed to a value of 1. Since these auxiliary coordinates are at least as large as the original d inputs, the median of the resulting $(2d-1)$-dimensional vector coincides with the maximum of the original d inputs. Furthermore, since fixing input coordinates to constants yields a valid ReLU network of the same architecture (by appropriately modifying the bias terms in the first hidden layer), it follows that there exists a network of this type that computes the maximum precisely. Given that the exponent in Theorem 6 is at most 2 and $r \leq d-1$, this reduction results in a constant multiplicative blow-up of at most 4 in the network width.

The previous section suggested that computing the median is inherently more difficult than computing the maximum, which is reflected in the additional insights required to compute the median in our upper bounds. Theorem 7 formalizes this by demonstrating that the maximum can be reduced to the median (or any rank r), thereby rigorously establishing that the median is at least as difficult to compute as the maximum. Because the median represents the most challenging rank to compute, the remainder of our analysis focuses exclusively on the median function. We emphasize, however, that our results generalize to any rank r in a straightforward manner.

Despite the simplicity of the exact reduction, our primary objective is to characterize the complexity of approximate computation. To this end, we introduce a general reduction scheme that establishes an analogous result for L_2 approximation with respect to the uniform distribution on the unit hypercube, $[0,1]^d$. This setting is particularly motivated by existing lower bounds for the maximum function in the L_2 regime; by reducing from these results, we can derive corresponding approximation lower bounds for the median. Our reduction is as follows:

Theorem 8 *Let σ be any measurable activation function, and suppose that for all $d \geq 2$, there exists a depth-k, width-$w(d)$ σ-neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$ with weights bounded by $M(d)$, that satisfies*

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}([0,1]^d)} \left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x}) \right)^2 \right] \leq \varepsilon.$$

Then, there exists a depth-k, width-$w(2d)$ σ-neural network $\mathcal{N}'$ with weights bounded by $2M(2d)$ such that

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}([0,1]^d)} \left[\left(\mathcal{N}'(\mathbf{x}) - \max(\mathbf{x}) \right)^2 \right] \leq 8\sqrt{\pi d}\varepsilon.$$

The proof of the above theorem, deferred to Appendix E.2, builds upon the technical framework established in the proof of Theorem 6. The primary challenge in extending the reduction to the L_2 setting is that simply padding the d-dimensional input with $d-1$ ones restricts the resulting $(2d-1)$-dimensional inputs to a set of measure zero in $(2d-1)$-dimensional space, rendering the standard reduction unusable. To circumvent this, we instead sample the $d-1$ auxiliary coordinates from a uniform distribution supported in $[1,2]$ and permute the coordinates. This construction ensures that the median of the augmented input coincides with the maximum of the original input, while guaranteeing that the support of the resulting distribution has a non-negligible measure in $[0,2]^{2d-1}$. This measure is bounded away from zero because the probability of drawing approximately half of the auxiliary values from the lower half of their support is proportional to the mode of a binomial distribution, which accounts for the $\sqrt{d}$ factor appearing in our accuracy bound.

To utilize the reduction described above, we leverage existing lower bound results for the maximum function. First, however, we must adopt the same assumptions regarding the activation function as those employed in the prior literature:

Assumption 9 (Polynomially-bounded activation) *The activation function σ is Lebesgue measurable and satisfies*

$$|\sigma(x)| \leq C_\sigma \left(1 + |x|^{\alpha_\sigma}\right),$$

for all $x \in \mathbb{R}$ and for some constants $C_\sigma, \alpha_\sigma > 0$.

The following result, established in Safran et al. (2024b) and restated here in a slightly modified manner to better suit our context, shows that achieving an arbitrarily accurate approximation of the maximum function with respect to the uniform distribution on $[0,1]^d$, using a depth-2 ReLU network, requires the width to scale with the target accuracy ϵ.

Theorem 10 (Safran et al. (2024b, Theorem 4.2)) *For all natural $n \geq 1$, suppose that σ satisfies Assumption 9. Then, there exist constants $c_1, c_2 > 0$ which depend solely on σ such that for all dimensions $d \geq c_1$, a σ-neural network $\mathcal{N}$ of depth 2 and width at most n and with weights bounded by $\mathcal{O}(\exp(\mathcal{O}(d)))$ must satisfy*

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}\left([0,1]^d\right)} \left[\left(\mathcal{N}(\mathbf{x}) - \max(\mathbf{x})\right)^2\right] > \Omega\left(n^{-c_2}\right).$$

By utilizing the above result and our Theorem 8, absorbing constants inside the asymptotic notation, the following is an immediate corollary:

Corollary 11 *For all natural $n \geq 1$, suppose that σ satisfies Assumption 9. Then there exist constants $c_3, c_4 > 0$ which depend solely on σ such that for all dimensions $d \geq c_3$, a σ-neural network $\mathcal{N}$ of depth 2 and width at most n and with weights bounded by $\mathcal{O}(\exp(\mathcal{O}(d)))$ must satisfy*

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}\left([0,1]^d\right)} \left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] > \Omega\left(n^{-c_4}\right).$$

Next, we extend our analysis to higher depths by providing a reduction from a known depth-3 lower bound for the maximum function. Specifically, we leverage the following result from Safran et al. (2024b):

Theorem 12 (Safran et al. (2024b, Theorem 4.3)) *Suppose that $\mathcal{N}$ is a depth-3 ReLU network of width at most $\frac{d^2}{5}$ and with weights bounded by $\exp(\mathcal{O}(d))$. Then, there exist absolute constants $c_1, c_2 > 0$ such that for all $d \geq c_1$,*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \max(\mathbf{x})\right)^2\right] > \Omega\left(d^{-c_2}\right).$$

With the above and Theorem 8, we are able to show the following result via a reduction.

Corollary 13 *Suppose that $\mathcal{N}$ is a depth-3 ReLU network of width at most $\frac{d^2}{20}$ and with weights bounded by $\exp(\mathcal{O}(d))$. Then, there exist absolute constants $c_3, c_4 > 0$ such that for all $d \geq c_3$,*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] > \Omega\left(d^{-c_4}\right).$$

Lastly, the following result is an extension of the lower bound derived in Safran et al. (2024b, Theorem 4.4) for neural networks of arbitrary depth.

Theorem 14 *Let $d \geq 2$, and suppose that $\mathcal{N}$ is a neural network employing any activation function and having first hidden layer width of at most $d - 1$. Then,*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x})\right)^2\right] \geq \Omega\left(d^{-5.5}\right).$$

The above theorem shows that width of at least d is required for approximating the median. Since a direct application of the reduction technique from Theorem 8 incurs a constant factor loss of 0.5, we instead adapt the approach of Safran et al. (2024b) to establish a width d lower bound for the median. See Appendix E.3 for the full proof.

5. Summary and future work

In this work, we have established several lower and upper bounds for the approximation of the median function using neural networks. Our analysis highlights that the median, which is inherently more complex than the maximum, requires novel techniques for efficient approximation, culminating in a constant-depth, linear-width construction for the uniform distribution over the unit hypercube. Together with our exact computation lower bound in Theorem 7, these results establish the first separation between the exact and approximate computation regimes. Specifically, we demonstrate that relaxing the accuracy requirement to even an exponentially small error threshold allows for a dramatic reduction in the depth required to achieve a high-accuracy approximation using a linearly-sized network.

Furthermore, while our constant-depth, linear-width construction cannot achieve arbitrarily high accuracy via weight-scaling alone as the primary upper bound in Safran et al. (2024b) does, it offers a superior architecture for error thresholds exceeding the exponentially small regime. This highlights that for the levels of approximation most commonly encountered in practical settings, the architectural requirements for the median (and, by extension, for the maximum via Theorem 8) may be significantly more modest than previously established.

There are several natural directions for future research. First, while our lower bounds establish that the median is at least as difficult to approximate as the maximum, our upper bounds strongly suggest that it is strictly more difficult. The latter required significantly more sophisticated insights and techniques to implement. It would be of great interest to explore whether this intuition can be formally verified by deriving lower bounds that do not rely on a reduction from the maximum, potentially yielding strictly stronger complexity results that are specific for the median function.

Second, given that scaling our weights beyond an exponential magnitude yields no additional benefit, it remains to be seen whether alternative constant-depth, linear-width constructions exist that can achieve arbitrarily high accuracy solely through weight-scaling, analogous to the primary result in Safran et al. (2024b). Conversely, establishing a provable barrier to such accuracy in the linear-width regime would further delineate the fundamental trade-offs between parameter magnitude and network architecture.

Finally, while Corollary 13 establishes that a linear-width approximation of the median to exponentially small accuracy is impossible at depth 3, Theorem 4 demonstrates that such an approximation is achievable at depth 46. Since we did not attempt to optimize the constant 46, it remains an open and intriguing question to determine the precise minimal depth required to achieve this level of accuracy within the linear-width regime.

Acknowledgments

Itay Safran is supported by Israel Science Foundation Grant No. 1753/25. Abhigyan Dutta and Paul Valiant are partially supported by NSF awards CCF-2228814 and CCF-2127806 and ONR Award N00014-24-1-2695.

References

Miklós Ajtai, János Komlós, and Endre Szemerédi. An $o(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83)*, STOC '83, pages 1–9, New York, NY, USA, 1983. Association for Computing Machinery. ISBN 0897910990. doi: 10.1145/800061.808726. URL https://doi.org/10.1145/800061.808726.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

Gennadiy Averkov. On the expressiveness of rational relu neural networks with bounded depth, 2025. URL https://arxiv.org/abs/2502.06283.

Egor Bakaev, Florestan Brunck, Christoph Hertrich, Daniel Reichman, and Amir Yehudayoff. On the depth of monotone relu neural networks and icnns. *arXiv preprint arXiv:2505.06169*, 2025a.

Egor Bakaev, Florestan Brunck, Christoph Hertrich, Jack Stade, and Amir Yehudayoff. Better neural network expressivity: Subdividing the simplex, 2025b. URL https://arxiv.org/abs/2505.14338.

Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, pages 307–314, 1968.

Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

Kuan-Lin Chen, Harinath Garudadri, and Bhaskar D Rao. Improved bounds on neural complexity for representing piecewise linear functions. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 7167–7180. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/2f4b6febe0b70805c3be75e5d6a66918-Paper-Conference.pdf.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

Moritz Grillo, Christoph Hertrich, and Georg Loho. Depth-bounds for neural networks via the braid arrangement. *arXiv preprint arXiv:2502.09324*, 2025.

Christian Haase, Christoph Hertrich, and Georg Loho. Lower bounds on the depth of integral relu neural networks via lattice polytopes. *arXiv preprint arXiv:2302.12553*, 2023.

Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38(3):502–527, 2020.

Christoph Hertrich, Amitabh Basu, Marco Di Summa, and Martin Skutella. Towards lower bounds on the depth of relu neural networks. *Advances in Neural Information Processing Systems*, 34: 3336–3348, 2021.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Donald E Knuth. *The art of computer programming: Sorting and searching, volume 3*. Addison-Wesley Professional, 1998.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Shiyu Liang and R. Srikant. Why deep neural networks for function approximation? In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Kyle Matoba, Nikolaos Dimitriadis, and François Fleuret. The theoretical expressiveness of max-pooling. *arXiv preprint arXiv:2203.01016*, 2022.

Anirbit Mukherjee and Amitabh Basu. Lower bounds over boolean inputs for deep neural networks with relu gates. *arXiv preprint arXiv:1711.03073*, 2017.

Sergei Ovchinnikov. Max-min representation of piecewise linear functions. *arXiv preprint math/0009026*, 2000.

Ryan O'Donnell and Karl Wimmer. Approximation by dnf: examples and counterexamples. In *International Colloquium on Automata, Languages, and Programming*, pages 195–206. Springer, 2007.

Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1): 26–29, 1955.

Itay Safran. A depth hierarchy for computing the maximum in relu networks via extremal graph theory. *arXiv preprint arXiv:2601.01417*, 2026.

Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *International Conference on Machine Learning*, pages 2979–2987. PMLR, 2017.

Itay Safran, Ronen Eldan, and Ohad Shamir. Depth separations in neural networks: what is actually being separated? In *Conference on Learning Theory*, pages 2664–2666. PMLR, 2019.

Itay Safran, Daniel Reichman, and Paul Valiant. Depth separations in neural networks: Separating the dimension from the accuracy. *arXiv preprint arXiv:2402.07248*, 2024a.

Itay Safran, Daniel Reichman, and Paul Valiant. *How Many Neurons Does it Take to Approximate the Maximum?*, pages 3156–3183. Symposium on Discrete Algorithms (SODA), 2024b. doi: 10.1137/1.9781611977912.113. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.113.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on Learning Theory*, pages 1517–1539. PMLR, 2016.

Luca Venturi, Samy Jelassi, Tristan Ozuch, and Joan Bruna. Depth separation beyond radial functions. *arXiv preprint arXiv:2102.01621*, 2021. Version 4, 22 Sep 2021.

David Wajc. Negative association. Lecture notes, 2017. URL https://www.cs.cmu.edu/~dwajc/notes/. Available online.

Shuning Wang and Xusheng Sun. Generalization of hinging hyperplanes. *IEEE Transactions on Information Theory*, 51(12):4425–4431, 2005.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Map of the Appendix

Appendix A. Appendix-specific notations

Recall our previously defined notations: We use bold-faced letters to denote vectors: $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$. For a positive integer n, we use the shorthand $[n] \coloneqq \{1, 2, \ldots, n\}$. We define by $\mathbb{Z}^{>0}$ the set of strictly positive integers. The indicator function is denoted by $\mathbb{1}\{\cdot\}$. For a vector $\mathbf{x}$, we define $\max(\mathbf{x})$ and $\mathrm{med}(\mathbf{x})$ to be the *maximum* and *median* of the entries in the vector $\mathbf{x}$, respectively. Given a set S, we denote by $\mathcal{U}(S)$ the uniform distribution over this set S. Given a vector $\mathbf{x} \in \mathbb{R}^d$, we denote its rank-$k$ element using $\mathcal{R}_k(\mathbf{x})$—namely, the k^{th} entry when $\mathbf{x}$ is sorted in ascending order. All logarithms have base e unless otherwise stated.

We call a vector $\mathbf{x} \in \mathbb{R}^d$, (d, ε) sparse if $\mathbf{x}$ has at most d^ε non-zero entries, i.e., $\sum_{i=1}^d \mathbb{1}\{x_i \neq 0\} \leq d^\varepsilon$. We occasionally treat a vector $\mathbf{x}$ as an array, and hence define the corresponding notations $\mathbf{x}[i : j] \coloneqq (x_i, x_{i+1}, \ldots x_j)$ and $\mathbf{x}[i] = x_i$. For $p \in [1, \infty)$, the ℓ_p norm of $\mathbf{x}$ is defined as $\|x\|_p \coloneqq \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$, where the specific case $p = \infty$ is defined as $\|\mathbf{x}\|_\infty \coloneqq \max_{i \in [d]} |x_i|$. Finally, we denote by $\mathbf{x}^{\neq 0}$ the set of non-zero entries among $\mathbf{x}$. We extensively use the following notion of separatedness and boundedness in Section G;

Definition 15 (δ separation and boundedness of vectors) *Given a vector $\mathbf{x}$, suppose that its non-zero entries satisfy the following:*

- $x_i \in [\delta, 1 - \delta]$ *and,*
- $\forall i \neq j,\ |x_i - x_j| \geq \delta$.

We denote the set of all $\mathbf{x}$ satisfying the above conditions by $\mathcal{S}_\delta^d$.

Appendix B. Depth 3, width $\mathcal{O}(d^2)$ median computation

We first show in Proposition 16 how to correctly sort inputs of size d using a quadratic width "all pairs" approach, provided the input is δ-separated and bounded. We then show that for inputs from the uniform hypercube, the input will be δ-separated with high probability for inverse-polynomial δ, leading to Theorem 2, bounding the expected squared error of this neural network on random input.

The below proposition shows how a neural network to *sort* its input, and we point out that we can trivially use this sorting network to return the median, by extracting the $d/2^{\text{th}}$ output.

Proposition 16 *For any dimension $d > 0$ and $\delta > 0$, there exists a ReLU neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}^d$ of width $4d^2$, using 2 hidden layers, and magnitudes of weights $\leq \frac{1}{\delta}$, that sorts any entirely non-zero input $\mathbf{x} \in \mathcal{S}_\delta^d$. For general input $\mathbf{x} \in \mathbb{R}^d$, the return values are bounded by $d\,\|\mathbf{x}\|_\infty$.*

Proof We compute the answer using the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$) by plugging in $d' = d$ and $\mathbf{r} = (1, 2, \ldots, d)$ and $p = d$. By Lemma 46, since $\mathbf{x} \in \mathcal{S}_\delta^d$ and is entirely non-zero, we have that the neural network outputs the elements of ranks $\mathbf{r} = (1, 2, \ldots, d)$ in ascending order. From Lemma 46, the network uses 2 hidden layers, has width $4d^2$, and the magnitudes of weights are bounded by $\frac{1}{\delta}$.

For all other cases, the outputs are upper bounded by $d\,\|\mathbf{x}\|_\infty$ from Lemma 46. ■

Algorithm 3: Probabilistic algorithm for median approximation (Superlinear Width).

Input: Vector: $\mathbf{x} \in (0,1)^d$ with unique and uniformly randomly permuted entries.
Output: $\mathrm{med}(\mathbf{x})$ with probability $\geq 1 - \exp\left(-\Omega(d^{2\gamma})\right)$.
`depth5MedianComputation(`$\mathbf{x}, \gamma$`):`

1. Partition the d elements into blocks of size $\lceil d^{2/3} \rceil$ and let q be the number of such blocks, where the last block might have smaller size. Denote the i^{th} block as $\mathbf{x}_i$, for $i \in [q]$. For $i \in [q-1]$, create a new vector $\mathbf{y}_i$ containing the elements of $\mathbf{x}_i$ with ranks belonging to the set $\left\{ \left\lfloor \frac{d^{2/3}}{2} - d^{\frac{1}{3}+\gamma} \right\rfloor, \ldots, \left\lceil \frac{d^{2/3}}{2} + d^{\frac{1}{3}+\gamma} \right\rceil \right\}$; and let $\mathbf{y}_q = \mathbf{x}_q$, effectively selecting the entirety of the last block, for simplicity.
2. Consider the entries of the concatenation $(\mathbf{y}_1, \ldots, \mathbf{y}_q)$ and compare each with all the entries of $\mathbf{x}$, counting the number of entries it is larger than; then output the median—namely, the element that wins $d/2 - 1$ comparisons. If no such entry exists, consider the algorithm to have failed.

B.1. Proof of Theorem 2

Proof We use Proposition 16 to show that, when the input $\mathbf{x}$ is δ-separated and appropriately bounded, the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$) will accurately compute the median. We combine this with an analysis of the expected squared error in the rare cases that the randomly chosen $\mathbf{x}$ violates the assumptions of Proposition 16.

From Lemma 26, plugging in $d' = d, \delta = \frac{\epsilon}{12d^4}$ we have $\mathbb{P}\left[\exists i, x_i = 0 \text{ or } \mathbf{x} \notin \mathcal{S}_\delta^d\right] \leq \frac{\epsilon}{4d^2}$. Denoting the event $\left[\exists i, x_i = 0 \text{ or } \mathbf{x} \notin \mathcal{S}_\delta^d\right]$ as F we can use Proposition 16 to conclude,

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x}))^2\right] &= \mathbb{P}\left[F\right] \cdot \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x}))^2 \,\middle|\, F\right] \\ &+ \mathbb{P}\left[\overline{F}\right] \cdot \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[(\mathcal{N}(\mathbf{x}) - \mathrm{med}(\mathbf{x}))^2 \,\middle|\, \overline{F}\right] \leq (d\,\|\mathbf{x}\|_\infty + 1)^2 \cdot \frac{\epsilon}{4d^2} \leq \epsilon\end{aligned}$$

Also, from Proposition 16 we have that the width of the neural network used is $\mathcal{O}(d^2)$, with 2 hidden layers with magnitude of weights bounded by $1/\delta = 12d^4/\epsilon$ concluding the proof. ■

Appendix C. Depth 5, width roughly $\mathcal{O}(d^{5/3})$ median computation

In this section, we present the depth 5 construction for extracting the median from an input of size d, and show its output is correct with high probability. For a concrete statement on the width and depth requirement along with the probability of correctness, refer to Theorem 3.

C.1. Construction and auxiliary lemmas

Definition 17 *We call an execution of Algorithm 3 "successful" if one of the shortlisted blocks $\mathbf{y}_i$, for $i \in [q]$, contains the median of $\mathbf{x}$.*

Proposition 18 *Algorithm 3, when given a uniform random permutation of a set $S \subset (0,1)$ of size d represented in the algorithm by vector $\mathbf{x} \in [0,1]^d$, is "successful" in the sense of Definition 17 with probability at least $1 - \exp\left(-\Omega\left(d^{2\gamma}\right)\right)$, and subsequently outputs the median $\mathrm{med}(\mathbf{x})$ when it is "successful".*

Proof Recall from Algorithm 3 we partition $\mathbf{x}$ into blocks $\mathbf{x}_i$ of size $\leq \lceil d^{2/3} \rceil$ for $i \in [q]$, where $q \leq \lceil d^{1/3} \rceil$. Since the entries of $\mathbf{x}$ are uniformly randomly permuted, the entries of any particular block $\mathbf{x}_i$ are a uniformly random subset of size $\leq \lceil d^{2/3} \rceil$ of the d entries. Defining $\mathbf{L} = \lfloor d^{2/3}/2 - d^{1/3+\gamma} \rfloor, \mathbf{R} = \lceil d^{2/3}/2 + d^{1/3+\gamma} \rceil$, for each block $\mathbf{x}_i$ for $i \in [q-1]$ we define $e_i^- = \mathcal{R}_{\mathbf{L}}(\mathbf{x}_i), e_i^+ = \mathcal{R}_{\mathbf{R}}(\mathbf{x}_i)$. For a particular block $\mathbf{x}_i$ we show that with high probability the number of elements less than or equal to $\mathrm{med}(\mathbf{x})$ in $\mathbf{x}_i$ is in the range $[\mathbf{L}, \mathbf{R}]$; this implies that $e_i^- \leq \mathrm{med}(\mathbf{x}) \leq e_i^+$.

We bound the number of elements $\leq \mathrm{med}(\mathbf{x})$ in $\mathbf{x}_i$ via part 1 of Lemma 27: $\mathbf{x}_i$ is a random subset of the elements of $\mathbf{x}$ of, called S in the context of Lemma 27 and which has size $n = \lceil d^{2/3} \rceil$; and let T denote those elements of $\mathbf{x}$ that are $\leq \mathrm{med}(\mathbf{x})$, which has size $k = d/2$; let $\epsilon = d^{\frac{1}{3}+\gamma} - 1$. Part 1 of Lemma 27 says that the probability that $|S \cap T|$ has distance $\geq \epsilon$ from its expectation $\frac{(d/2)\lceil d^{2/3} \rceil}{d}$ is at most $2e^{-2\epsilon^2/\lceil d^{2/3} \rceil}$. Thus, except with probability $\exp(-\Omega(d^{2\gamma}))$ we have that the number of elements less than or equal to $\mathrm{med}(\mathbf{x})$ in $\mathbf{x}_i$ is in the range $\frac{(d/2)\lceil d^{2/3} \rceil}{d} \pm \epsilon$, which is a subset of the range $[\mathbf{L}, \mathbf{R}]$, as desired.

Taking a union bound over all $q-1$ blocks for which we throw out elements, we conclude that, except with $(q-1)\exp(-\Omega(d^{2\gamma})) = \exp(-\Omega(d^{2\gamma}))$ probability, we will not throw out the median from any block. Thus since the median lies in *some* block $\mathbf{x}_i$, it must also lie in some block $\mathbf{y}_i$ for $i \in [q]$, except with probability $\exp(-\Omega(d^{2\gamma}))$.

Thus since Step 2 explicitly tests whether each element of the blocks $\mathbf{y}_i$ for $i \in [q]$ is the overall median, the algorithm will find and return the median as desired.

■

Proposition 19 *For any dimension $d > 0$ and any $\delta > 0$, if for some entirely non-zero vector $\mathbf{x} \in \mathcal{S}_\delta^d$ Algorithm 3 is "successful" in the sense of Definition 17, then the ReLU neural network outlined by Algorithm 4 successfully implements Algorithm 3 and thereby returns $\mathrm{med}(\mathbf{x})$, else, for all input $\mathbf{x} \in [0,1]^d$ it outputs a value within the interval $[-d^2, d^2]$. Moreover, this neural network has width $\mathcal{O}\left(d^{5/3+\gamma}\right)$ with 4 hidden layers, and with the magnitude of weights upper bounded by $1/\delta$.*

Proof We prove the correctness of the neural network presented in Algorithm 4 in implementing Algorithm 3 assuming the "success" of Algorithm 3 (Definition 17), on an entirely non-zero input $\mathbf{x} \in \mathcal{S}_\delta^d$. Specifically, will show how to implement steps 1 and 2 of Algorithm 3 using a ReLU neural network of width $\mathcal{O}\left(d^{5/3+\gamma}\right)$ and depth 5 in the corresponding steps (1,2) of Algorithm 4.

- **Step 1:** Informally, in this step we partition $\mathbf{x}$ into $q \leq \frac{d}{\lceil d^{2/3} \rceil} \in \mathcal{O}(d^{1/3})$ blocks and select entries with ranks (with respect to the block) in the set $\left\{ \left\lfloor \frac{d^{2/3}}{2} - d^{\frac{1}{3}+\gamma} \right\rfloor, \ldots, \left\lceil \frac{d^{2/3}}{2} + d^{\frac{1}{3}+\gamma} \right\rceil \right\}$ from each block. Step (1) is implements this step using the *rank selection* neural network. Subsequently we let $\left| \left\{ \left\lfloor \frac{d^{2/3}}{2} - d^{\frac{1}{3}+\gamma} \right\rfloor, \ldots, \left\lceil \frac{d^{2/3}}{2} + d^{\frac{1}{3}+\gamma} \right\rceil \right\} \right| = p' \in \mathcal{O}(d^{1/3+\gamma})$. Since

Algorithm 4: Given $\mathbf{x}$ compute the median of the entries (Neural network construction of Algorithm 3).

Input: Entirely non-zero vector $\mathbf{x} \in \mathcal{S}_\delta^d$, with uniformly randomly permuted entries.
Output: $\text{med}(\mathbf{x})$ or some value $\leq d^2$.
`depth5MedianComputation(`$\mathbf{x}, \gamma$`):`

1. Partition the d elements into blocks of size $\lceil d^{2/3} \rceil$ and let q be the number of such blocks, where the last block might have smaller size. Denote the i^{th} block as $\mathbf{x}_i$, for $i \in [q]$. For $i \in [q-1]$, create a new vector $\mathbf{y}_i$ containing the elements of $\mathbf{x}_i$ with ranks belonging to the set $\left\{ \left\lfloor \frac{d^{2/3}}{2} - d^{\frac{1}{3}+\gamma} \right\rfloor, \ldots, \left\lceil \frac{d^{2/3}}{2} + d^{\frac{1}{3}+\gamma} \right\rceil \right\}$; and let $\mathbf{y}_q = \mathbf{x}_q$, effectively selecting the entirety of the last block, for simplicity.
    - We do this in parallel for each of the q blocks using *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$).
2. Consider the entries of the concatenation $(\mathbf{y}_1, \ldots, \mathbf{y}_q)$ and compare each with all the entries of $\mathbf{x}$, counting the number of entries it is larger than; then output the median—namely, the element that wins $d/2 - 1$ comparisons. If no such entry exists, consider the algorithm to have failed.
    - We do this in parallel for each $\mathbf{y}_i, i \in [q]$ using *modified comparison* neural network (Definition 35, $\mathcal{N}_\delta^{C}$) and the *indicator function product* neural network (Definition 43, $\mathcal{N}^{IFP}$).

$\mathbf{x} \in \mathcal{S}_\delta^d$ and is entirely non-zero it follows that $\forall i \in [q-1],\ \mathbf{x}_i \in \mathcal{S}_\delta^{\lceil d^{2/3} \rceil}$ and is entirely non-zero, and using Lemma 46 (with $d' = \lceil d^{2/3} \rceil, p = p'$) we conclude that *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$) correctly computes this step for each block i. The q'th block is propagated unmodified.

We implement this step for each block $i \in [q-1]$ in parallel and again using Lemma 46 with $d' = \lceil d^{2/3} \rceil, p = p'$ we conclude that the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$) uses 2 hidden layers and width of $\mathcal{O}\left(q \cdot d^{4/3}\right) \in \mathcal{O}\left(d^{5/3}\right)$. Also, from Lemma 46 we have that the magnitude of weights used by this layer is $1/\delta$.

- **Step 2** Intuitively, in Step (2) we aim to compare all the shortlisted entries, denoted by $\mathbf{y}_i, \forall i \in [q]$ with the entries of $\mathbf{x}$ and return the element which is larger than $d/2 - 1$ entries of $\mathbf{x}$. For an entry y in $\mathbf{y} = (\mathbf{y}_1, \cdots, \mathbf{y}_q)$, since y is also a non-zero entry of $\mathbf{x} \in \mathcal{S}_\delta^d$ we can compute using the *comparison* neural network (Definition 35, $\mathcal{N}_\delta^{C}$) to count the number of entries in $\mathbf{x}$ it is larger than. Formally, we can compute $\sum_j \mathbb{1}\{y > x_j\}$ correctly, by using the neural network defined as $\sum_j \mathcal{N}_\delta^{C}(y, x_j)$, requiring a width of $2d$ and 1 hidden layer. We do this in parallel for the all the entries of $\mathbf{y}$, and the number of such entries is $\mathcal{O}(p'q) \in \mathcal{O}\left(d^{2/3+\gamma}\right)$. Hence, by Fact 36 this comparison step requires a width of $\mathcal{O}\left(d \cdot d^{2/3+\gamma}\right) \in \mathcal{O}\left(d^{5/3+\gamma}\right)$ and 1 hidden layer. Finally we use the *indicator function prod-*

uct neural network (Definition 43) to shortlist the element which wins $d/2-1$ comparisons by computing $\sum_k \mathcal{N}^{IFP}\left(y_k, \sum_j \mathcal{N}^C_\delta(y_k, x_j) - (d/2-1)\right)$. Namely, for each k the expression $\sum_j \mathcal{N}^C_\delta(y_k, x_j)$ counts the number of elements of $\mathbf{x}$ that are smaller than y_k; we then compare this number to $d/2-1$ and the $\mathcal{N}^{IFP}$ network outputs the single y_i for which the count matches, by Lemma 44. Counting the width, depth, and weights used, we see that this step uses width $\mathcal{O}(d^{5/3+\gamma})$, 2 hidden layers, and the magnitudes of weights are bounded by $1/\delta$.

From our assumption that Algorithm 3 is "successful" we have that there exists a block $\mathbf{y}_i$ that contains $\text{med}(\mathbf{x})$. Combining this with Step 2, where the overall ranks (with respect to $\mathbf{x}$) of every entry in $(\mathbf{y}_1, \cdots, \mathbf{y}_q)$ are correctly computed, and the element which wins $d/2-1$ comparisons is returned: we have that the neural network construction in Algorithm 4 correctly computes the median of $\mathbf{x}$. Thus, overall the construction requires 4 hidden layers, width of $\mathcal{O}(d^{5/3+\gamma})$ and magnitudes of weights bounded by $1/\delta$.

Otherwise, for arbitrary input $\mathbf{x} \in [0,1]^d$, we explicitly bound the magnitude of the outputs. From Lemma 46, $\|\mathbf{x}'\|_\infty \leq d\,\|\mathbf{x}\|_\infty$ i.e., the output of the first two layers of the neural network construction is bounded by $d\,\|\mathbf{x}\|_\infty$. Using Lemma 44 on this output, we have that the final output is bounded by $d\,\|\mathbf{x}'\|_\infty \leq d^2\,\|\mathbf{x}\|_\infty \leq d^2$, since $\mathbf{x} \in [0,1]^d$. ■

C.2. Proof of Theorem 3

Proof We use Proposition 19 to show that, when the input $\mathbf{x}$ is δ-separated and appropriately bounded, the neural network outlined in Algorithm 3 will accurately compute the median with probability at least $1-\exp(-\Omega(d^{2\gamma}))$ (where recall increasing γ will increase the width of our neural network), and always returns values in $[-d^2, d^2]$. We combine this with the bounds on the probability that a randomly chosen input will fail the input requirements of Proposition 19: from Lemma 26, plugging in $d' = d, \delta := \frac{\epsilon}{12d^6}$ we have that $\mathbb{P}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[\mathbf{x} \notin \mathcal{S}^d_{\epsilon/12d^6}\right] \leq \epsilon/4d^4$. Thus, from the union bound on these two failure modes,

$$\mathbb{P}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[\mathcal{N}(\mathbf{x}) \neq \text{med}(\mathbf{x})\right] \leq \frac{\epsilon}{4d^4} + \exp\left(-d^{\Omega(1)}\right).$$

In the cases that $\mathcal{N}(\mathbf{x}) \neq \text{med}(\mathbf{x})$, since the true median has a range of $[0,1]$ and our algorithm's returned answer is in the range $[-d^2, d^2]$ we have

$$\left(\mathcal{N}(\mathbf{x}) - \text{med}(\mathbf{x})\right)^2 \leq 4d^4$$

. Thus the mean squared error can be bounded as,

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x}) - \text{med}(\mathbf{x})\right)^2\right] \leq 4d^4 \cdot \left(\frac{\epsilon}{4d^4} + \exp\left(-\Omega\left(d^{2\gamma}\right)\right)\right) \leq \epsilon + \exp\left(-\Omega\left(d^{2\gamma}\right)\right)$$

concluding the proof of the first part of the theorem.

The neural network construction outlined in Proposition 19 has 4 hidden layers, width of $\mathcal{O}(d^{5/3+\gamma})$ and magnitudes of weights bounded by $1/\delta = 12d^6/\epsilon$ concluding the proof. ■

Appendix D. Depth 46, width $\mathcal{O}(d)$ median computation

In this appendix, we present the proof of Theorem 4. This appendix section is organized as follows:

In the initial sparsification step, we present our construction as an algorithm (Algorithm 5) and analyze the probabilistic properties of its data stream in Proposition 21. Informally, this proposition demonstrates that we can sparsify the input vector with high probability while preserving the essential probabilistic properties of the input distribution required for the proof to succeed. Subsequently, we show that for δ-separated and bounded inputs, this algorithm can be implemented via the neural network outlined in Algorithm 6 (Proposition 22), achieving constant depth and linear width. This implementation employs multiple sub-architectures, each serving a distinct role in the median extraction pipeline. These components are rigorously defined and analyzed in Appendix G.

In the second and final step, we utilize a deterministic hashing construction to reduce the dimensionality of the sparsified vector to $\mathcal{O}(\sqrt{d})$. This reduction allows us to compute the *median* via brute-force comparisons, a process that requires quadratic width relative to the reduced dimension, yet remains linear with respect to the original dimension d. We outline this step in Algorithm 8 (Proposition 24) and conclude this appendix section with the proof of Theorem 4.

D.1. Sparsification step

The goal of this step is to obtain a sparse vector $\mathbf{x}'$ from a vector with nonzero entries $\mathbf{x}$ that still maintains the median with high probability, with respect to the randomness induced by the input distribution (see step 1 in Algorithm 5). Subsequently, we show that when the entries of $\mathbf{x}$ are bounded and δ-separated, the desired steps can be implemented using some neural network architecture (see implementation details in Proposition 22).

We start with a vector $\mathbf{x}$, and sample a random subset of it. Using the median of this random subset as a reference, we zero out all the entries of $\mathbf{x}$ whose values do not fall within a certain radius around this reference median. This results in a significant reduction with high probability in the number of non-zero entries, while keeping the median. Crucially, we show that we can repeat this process by repeatedly taking a subset of the remaining non-zero entries in a deterministic manner that succeeds with high probability with respect to the initial randomness of input $\mathbf{x}$. For some specific choice of parameters, we show that with high probability, we can reach our desired sparsity within a constant number of iterations of the above scheme, where the median is preserved at each iteration.

This procedure is described in Algorithm 5. To analyze this algorithm, we will focus on the inner "for" loop (by using mathematical induction), and show that, with probability at least $1 - \exp\left(-d^{\Omega(1)}\right)$ over the uniformly distributed entries of its input, the algorithm is "successful" in the following sense.

Definition 20 *We call an execution of Algorithm 5 "successful" if at the end of each iteration i, we have the following three properties:*

1. $\mathbf{x}_i^{\neq 0}$ *consists of* ***contiguous*** *elements from the sorted version of* $\mathbf{x}_1$.
2. *The overall median,* $\mathrm{med}(\mathbf{x}_1)$*, is one of the non-zero entries of* $\mathbf{x}_{i+1}$.
3. *The number of non-zero entries in* $\mathbf{x}_{i+1}$ *has the bounds* $y_{i+1}3^{-i} \leq |\mathbf{x}_{i+1}^{\neq 0}| \leq y_{i+1}6^i$.

Algorithm 5: Probabilistic algorithm for median approximation (Linear Width)

Input: Vector: $\mathbf{x}_1 \in (0,1)^d$ with unique and uniformly randomly permuted entries
Output: Modified Vector $\mathbf{x}_5$
`probabilisticAlgorithm(`$\mathbf{x}_1$`):`

1. For $i \in \{1,\dots,4\}$ define the triples of exponents $(y_i', z_i', w_i') = (1, 0.5, 0.26), (0.76, 0.5, 0.26), (0.52, 0.5, 0.26), (0.28, 0.26, 0.14)$, and also let $y_5' = 0.16$; then define the parameters $z_i = \lceil d^{z_i'} \rceil, w_i = \lfloor d^{w_i'} \rfloor, y_i = d^{y_i'}$.
2. For $i \in \{1,\dots,4\}$ do:
   (a) From $\mathbf{x}_i$ we pick z_i non-zero entries as follows: if $i = 1$ then take the first z_i entries. Otherwise for $i > 1$, divide $\mathbf{x}_i$ into blocks of size $\lceil 3^i d^{1.01}/y_i \rceil$, and from the first $\lceil z_i/d^{0.01} \rceil$ such blocks, choose the first $\lceil d^{0.01} \rceil$ non-zero elements if possible (and FAIL otherwise), but discard elements after z_i total elements have been chosen. Denote this sample set of size z_i by S_i.
   (b) Define r_i to be the rank in $\mathbf{x}_i^{\neq 0}$ of the overall median, $\mathrm{med}(\mathbf{x}_1)$ (see Lemma 50 for how we compute r_i with a neural network, without knowing the median). Let $\mathbf{L}_i = \frac{r_i}{|\mathbf{x}_i^{\neq 0}|} z_i - w_i$, and $\mathbf{R}_i = \frac{r_i}{|\mathbf{x}_i^{\neq 0}|} z_i + w_i$. Let $e_i^- = \mathcal{R}_{\lfloor \mathbf{L}_i \rfloor}(S_i)$ if $\mathbf{L}_i \geq 1$ and $e_i^- = 0$ otherwise; and let $e_i^+ = \mathcal{R}_{\lceil \mathbf{R}_i \rceil}(S_i)$ if $\mathbf{R}_i \leq z_i$ and $e_i^+ = 1$ otherwise.
   (c) Create a new copy of $\mathbf{x}_i$ represented as $\mathbf{x}_{i+1}$, where in $\mathbf{x}_{i+1}$ all entries with values $> e_i^+$ or values $< e_i^-$ are changed to 0.

Proposition 21 *Algorithm 5, for any set $S \subset (0,1)$ of size d, represented in the algorithm by a vector $\mathbf{x}_1 \in (0,1)^d$, is "successful" in the sense of Definition 20 with probability at least $1 - \exp\left(-d^{\Omega(1)}\right)$ over random permutations of $\mathbf{x}_1$.*

If the final loop iteration is "successful", this means that the algorithm outputs a vector $\mathbf{x}_5$ with at most $y_5 \cdot 6^4$ non-zero entries, and where the overall median $\mathrm{med}(\mathbf{x}_1)$ is one of them; but we instead show the stronger property that *all* loop iterations $i \in \{1,\dots,4\}$ are successful with high probability. We use this stronger property below to show how to implement Algorithm 5 with a neural network, in Proposition 22.

Proof The proof is by induction on the loop variable i, from 1 to 4. However, since we have a probabilistic input, we will show that the induction step holds with high probability, and then use a union bound at the end, adding up the failure probabilities of the 4 steps.

The algorithm performs a few major steps which each give correct results with probability $\geq 1 - \exp\left(-d^{\Omega(1)}\right)$.

Informally, iteration i considers the vector of non-zero entries $\mathbf{x}_i^{\neq 0}$ and looks for an element of some desired rank r_i among these; it does this by taking a random sample S_i of size $z_i \ll |\mathbf{x}_i^{\neq 0}|$, and then looking for the element of proportionate rank $r_i \frac{z_i}{|\mathbf{x}_i^{\neq 0}|}$ in S_i, throwing out all elements significantly smaller or larger then this. We keep track of how many elements bigger than the median

we throw out, so that we exactly know the desired rank we seek among the remaining elements. And then we repeat this process at the next loop iteration, on a smaller input. (Technically, we zero-out entries instead of throwing them out, since zeroing out elements is a natural neural network operation. Think of 0 entries as being "invisible" to the algorithm.)

We start by stating our induction hypothesis, which is identical to the notion of "success" from Definition 20, but with the addition of one more property, property 0 below that follows easily from the structure of the algorithm, and which we prove first. We will prove the following induction hypothesis for $i \in \{1, \ldots, 5\}$.

Induction hypothesis

1. $\mathbf{x}_i^{\neq 0}$ is the intersection of $\mathbf{x}_1$ with some real interval, which we denote I_i.
2. The overall median, $\mathrm{med}\,(\mathbf{x}_1)$ is one of the non-zero entries of $\mathbf{x}_i$, and we let r_i denote its rank among the non-zero entries of $\mathbf{x}_i$.
3. The number of non-zero entries in $\mathbf{x}_i$ has the bounds $3^{-(i-1)} y_i \leq |\mathbf{x}_i^{\neq 0}| \leq 6^{i-1} y_i$.

Base case The base case, $i = 1$, of the induction hypothesis trivially holds: 1) $\mathbf{x}_1$ is trivially the intersection of $\mathbf{x}_1$ with $[0, 1]$; 2) the median of $\mathbf{x}_1$ is trivially in $\mathbf{x}_1$; and 3) since we defined $y_1 = d$, we trivially have that $3^{-0} y_1 \leq |\mathbf{x}_1^{\neq 0}| \leq 6^0 y_1$.

Induction step

- We analyze loop i of the algorithm, assuming the induction hypothesis, and show that, at the end of loop i, the $i + 1$ version of the induction hypothesis will hold with high probability.
- We first prove property 0 of the induction hypothesis. By the induction hypothesis, the non-zero elements of $\mathbf{x}_i$ are the intersection of $\mathbf{x}_1$ with some real interval I_i. Recall that the non-zero elements of $\mathbf{x}_{i+1}$ are defined, in step (2c), to be exactly the non-zero elements of $\mathbf{x}_i$ that lie in the real interval $[e_i^-, e_i^+]$. Thus the non-zero elements of $\mathbf{x}_{i+1}$ are exactly the elements of $\mathbf{x}_1$ that lie in the intersection of these real intervals, $[e_i^-, e_i^+] \cap I_i$, which is itself a real interval, which we denote I_{i+1}, proving this property of the induction.
- As a direct consequence of induction hypothesis 1 of the induction hypothesis, we point out that $\mathbf{x}_i$ must consist of a subset of $\mathbf{x}_1$ of *contiguous ranks* in $\mathbf{x}_1$, with these ranks comprises some interval of integers $\{r_i^-, \ldots, r_i^+\}$. We use this property crucially in the proof below.
- The induction hypothesis states that $\mathbf{x}_i$ contains the median, and also that the number of non-zero elements in $\mathbf{x}_i$ is in a certain interval. We reexpress both conditions in terms of r_i^-, r_i^+: we have that $\frac{d}{2} \in \left[r_i^-, r_i^+\right]$ and that $r_i^+ - r_i^- + 1 \in \left[3^{-(i-1)} y_i, 6^{i-1} y_i\right]$.
- For each of the $\leq d^2$ potential values of the ranks $r_i^-, r_i^+ \in \{1, \ldots, d\}$, we will separately show that the induction step succeeds with high probability (with respect to the uniformly random permutation of $\mathbf{x}_1$). From now on, **fix** a particular choice of ranks r_i^-, r_i^+ that satisfies the two conditions $\frac{d}{2} \in \left[r_i^-, r_i^+\right]$ and that $r_i^+ - r_i^- + 1 \in \left[3^{-(i-1)} y_i, 6^{i-1} y_i\right]$. We use the generic probability fact that for two events A, B we have $\mathbb{P}[A, B] \leq \mathbb{P}[A|B]$. Specifically, the probability that A) the induction step fails, and B) the $\mathbf{x}_i$ used in iteration i consists of those elements of $\mathbf{x}_1$ with ranks in the interval $[r_i^-, r_i^+]$, is at most the probability of the induction

step failing *given* that we start the algorithm in iteration i, setting $\mathbf{x}_i$ to be those elements of $\mathbf{x}_1$ with ranks in $[r_i^-, r_i^+]$.

- We first analyze step (2a) to show that the algorithm does not FAIL in this step (except with probability $\exp(-d^{\Omega(1)})$, with respect to a random permutation of the input $\mathbf{x}_1$). For $i = 1$ the claim is trivially true as the input $\mathbf{x}_1$ is entirely non-zero by assumption and the algorithm explicitly takes S_1 to be the first z_i elements.

  Otherwise, for $i > 1$, we have from induction hypothesis 3 that the number of non-zero entries in $\mathbf{x}_i$ is $\geq 3^{-(i-1)}y_i$. From our analysis, recall that $\mathbf{x}_i$ consists of those elements of $\mathbf{x}_1$ with ranks in $[r_i^-, r_i^+]$, and probabilities are always taken with respect to random permutations of $\mathbf{x}_1$. Thus the location of non-zero entries of $\mathbf{x}_i^{\neq 0}$ will be uniformly random. Thus, in each block of size $\lceil 3^i d^{1.01}/y_i \rceil$ the expected number of non-zero entries is thus at least $3 \cdot d^{0.01}$, of which we aim to choose the first $\lceil d^{0.01} \rceil$, if they exist. We analyze this existence probability via part 2 of Lemma 27: we succeed if the number of non-zero entries is within a factor of 2 of its expectation (for large enough d); and part 2 of Lemma 27 says the probability of this failing is exponentially small in the expectation itself, namely $\exp(-d^{\Omega(1)})$ as desired.

- Next we show that S_i is a uniformly random subset (of size z_i) of the set of entries of $\mathbf{x}_1$ with ranks in $[r_i^-, r_i^+]$. Recall that, by the set up our analysis, we fix ranks r_i^-, r_i^+, permute $\mathbf{x}_1$ uniformly at random, and let $\mathbf{x}_i$ set to 0 those elements of $\mathbf{x}_1$ whose ranks are not in the interval $[r_i^-, r_i^+]$. In step (2a) we choose S_i to be a portion of $\mathbf{x}_i$, attempting to choose non-zero entries from certain blocks. Importantly, considering different permutations of $\mathbf{x}_1$, the choice of locations of $\mathbf{x}_i$ that are chosen for S_i depends *only* on whether those locations in $\mathbf{x}_i$ are non-zero, which thus depends only on whether those locations in $\mathbf{x}_1$ have ranks in $[r_i^-, r_i^+]$. Namely, step (2a) chooses S_i in a way that is *unaffected* by permuting the elements of rank $[r_i^-, r_i^+]$ in $\mathbf{x}_1$. Thus all subsets of $\mathbf{x}_i^{\neq 0}$ of a given size z_i are *equally* likely to be chosen as S_i. Thus, conditioned on the algorithm not FAILing (as analyzed in the previous paragraph), we conclude that S_i must be a uniformly random subset of size z_i of the elements of rank $[r_i^-, r_i^+]$ in $\mathbf{x}_1$.

- We now analyze steps (2b), (2c) of the algorithm, using the conclusions from above that: the non-zero entries of $\mathbf{x}_i$ consist only of elements of ranks $[r_i^-, r_i^+]$, and S_i is a uniformly random subset of these elements of size z_i.

- In step (2c) we zero out those elements of $\mathbf{x}_i$ that are smaller than e_i^- or larger than e_i^+; we want to show that we do *not* zero out the median, $\mathrm{med}(\mathbf{x}_1)$.

  Recall that the rank of the median in $\mathbf{x}_i^{\neq 0}$ was defined to be r_i. We thus apply part 1 of Lemma 27 with $S = S_i$ of size $n = z_i$, and letting T equal the set of elements $\leq med$, which has size $k = |T| = r_i$, and letting $\epsilon = w_i$. We conclude that with probability $\geq 1 - 2e^{-2w_i^2/z_i}$, the rank of the median in S_i is strictly between $\mathbf{L}_i = \frac{r_i z_i}{|\mathbf{x}_i^{\neq 0}|} - w_i$ and $\mathbf{R}_i = \frac{r_i z_i}{|\mathbf{x}_i^{\neq 0}|} + w_i$.

  Thus the median is at least the element of rank $\lfloor \mathbf{L}_i \rfloor$ in S_i (if there exists an element of that rank, and otherwise the median is at least 0); this is exactly the condition that $\mathrm{med} \geq e_i^-$ defined in step (2c). In the other direction, the median is thus at most the element of rank $\lceil \mathbf{R}_i \rceil$ in S_i (if there exists an element of that rank, and otherwise $\mathrm{med} \leq 1$), which means that $\mathrm{med} \leq e_i^+$. Thus, overall, we have shown that with probability $\geq 1 - 2e^{-2w_i^2/z_i}$, this step

will not throw out the median. We have chosen w_i, z_i so that $w_i^2/z_i = d^{\Omega(1)}$, leading to the desired exponentially small failure probability for part 2 of the induction hypothesis.

- We now prove part 2 of the induction step. We aim to apply Lemma 28 to bound $|\mathbf{x}_{i+1}^{\neq 0}|$. In terms of Lemma 28, the universe $U = \mathbf{x}_i^{\neq 0}$, the random subsample of this is $S = S_i$, and the algorithm selects a real interval $I = I_{i+1}$. We first claim that $|S \cap I| \in [w_i, 3w_i]$ (for large enough d). By definition, $\mathbf{x}_{i+1}^{\neq 0} = S \cap I$ contains all elements of S whose ranks are between $\left\lfloor \frac{r_i z_i}{|\mathbf{x}_i^{\neq 0}|} - w_i \right\rfloor$ and $\left\lceil \frac{r_i z_i}{|\mathbf{x}_i^{\neq 0}|} + w_i \right\rceil$; and the number of such ranks is clearly at most $2w_i + 3$, and at least w_i—since $\frac{r_i z_i}{|\mathbf{x}_i^{\neq 0}|} \in (0, z_i]$, so the center of the rank interval, when rounded up, is a valid rank of S. For large enough d, the interval $[w_i, 2w_i + 3]$ is trivially contained in $[w_i, 3w_i]$.

  We thus invoke Lemma 28 to conclude that with except with probability $\exp(-d^{\Omega(1)})$ we have $|\mathbf{x}_{i+1}^{\neq 0}| \in \left(\frac{1}{2} w_i \frac{|\mathbf{x}_i^{\neq 0}|}{z_i}, 2 \cdot 3 \cdot w_i \frac{|\mathbf{x}_i^{\neq 0}|}{z_i}\right)$. Since by construction of y_i, y_{i+1} we have $\frac{w_i y_i}{z_i} \in [\frac{2}{3} y_{i+1}, y_{i+1}]$ for large enough d, and using part 2 of the induction hypothesis $3^{-(i-1)} y_i \leq |\mathbf{x}_i^{\neq 0}| \leq 6^{i-1} y_i$, we conclude that $|\mathbf{x}_{i+1}^{\neq 0}| \in [3^{-i} y_{i+1}, 6^i y_{i+1}]$, proving the induction step with the desired high probability.

- In conclusion, for each of the $\leq d^2$ choices of ranks r_i^-, r_i^+, we have shown that the induction step fails with probability $\exp\left(-d^{\Omega(1)}\right)$. We additionally take the union over all 4 iterations of the induction, absorbing $4d^2$ into the asymptotic notation, to yield our desired failure probability of $\exp\left(-d^{\Omega(1)}\right)$.

Thus the guarantees in the proposition, as given by hypotheses 1, 2, 3, hold with high probability. ■

Proposition 22 *For any $d > 0$ and $\delta > 0$ if for some entirely non-zero vector $\mathbf{x}_1 \in \mathcal{S}_\delta^d$ Algorithm 5 is "successful" in the sense of Definition 20, then the ReLU neural network outlined by Algorithm 6 faithfully implements Algorithm 5 on input $\mathbf{x}_1$ and thereby returns a sparse vector with at most $6^4 d^{0.16}$ non-zero entries with the true median, $\mathrm{med}(\mathbf{x}_1)$, as one of its non-zero entries. Moreover, this neural network has width $\mathcal{O}(d)$ and depth 34 with magnitude of weights bounded by $\mathcal{O}(\max(d^{1.5}, 1/\delta))$.*

Proof We will show that for each loop iteration i, and for each step **2(a)** through **2(c)** of Algorithm 5 in loop i, we can faithfully implement this step using a ReLU neural network of width $\mathcal{O}(d)$ and depth $\mathcal{O}(1)$ in the corresponding steps (2a,b,c) of Algorithm 6. Our analysis relies on the conditions that $\mathbf{x}_i \in \mathcal{S}_\delta^d$ (treated as an induction hypothesis), and the "success" conditions of Definition 20, which are assumed in this proposition. As a base case, the input $\mathbf{x}_1 \in \mathcal{S}_\delta^d$ by assumption.

- **Parameter Properties:** The choice of the above parameters will satisfy the following inequalities for $i \in \{1, 2, 3, 4\}$:

  1. $1.01 + z_i' - y_i' < 1$
  2. $2z_i' \leq 1$

- **Step 2(a):** Informally, in this step we partition $\mathbf{x}$ in blocks of size $\Theta\left(d^{1.01-y_i'}\right)$ and from $\Theta\left(d^{z_i'-0.01}\right)$ such blocks try to shortlist $\Theta\left(d^{0.01}\right)$ non-zero entries. Step (2a) is used to implement this step, except when $i=1$ when this does not require any neural network since all entries are non-zero and we simply extract the first z_1 entries. Otherwise, by the induction hypothesis for $i>1$, we have $\mathbf{x}_i \in \mathcal{S}_\delta^d$. Using Lemma 48 and the fact $\mathbf{x}_i \in \mathcal{S}_\delta^d$ we see that Step (2a) which uses the *non-zero element shortlisting* (Definition 47, $\mathcal{N}_\delta^{NZES}$) correctly implements Step 2(a) (of Algorithm 5). The shortlisting operations are done on blocks of size $\lceil 3^i d^{1.01}/y_i \rceil$ in parallel $\lceil z_i/d^{0.01} \rceil$ times, where in each instance, $\leq \lceil d^{0.01} \rceil$ entries from the block are shortlisted. (As described in Algorithm 5, we choose the number of entries $\leq \lceil d^{0.01} \rceil$ to shortlist from each block so that z_i total non-zero entries are chosen.)

  The width required for shortlisting non-zero entries from each block can be found via Lemma 48 by plugging in $d' = \lceil 3^i d^{1.01}/y_i \rceil, p \leq \lceil d^{0.01} \rceil$, yielding width $\mathcal{O}(d^{1.01-y_i'+0.01})$. Since we repeat this in parallel for each of the $\lceil z_i/d^{0.01} \rceil$ blocks, the total width required is $\mathcal{O}(d^{1.01-y_i'+0.01} \cdot d^{z_i'-0.01}) \in \mathcal{O}(d^{1.01+z_i'-y_i'}) \in \mathcal{O}(d)$ width (see Parameter Properties). Also from Lemma 48 this requires 3 hidden layers and magnitude of weights bounded by $2/\delta$. We denote this vector of non-zero entries as S_i.

- **Step 2(b):** Informally, in this step we compute the rank of $\text{med}(\mathbf{x}_1)$ among the non-zero entries in the sparse vector $\mathbf{x}_i$ and then compute the endpoints of new interval around $\text{med}(\mathbf{x}_1)$, i.e., e_i^-, e_i^+ using S_i. Step (2b) is used to implement this step. For $i=1$ we have $r_1 = d/2, \left|\mathbf{x}_i^{\neq 0}\right| = d$, $z_1 = \lceil d^{0.5} \rceil$ and $w_1 = \lceil d^{0.26} \rceil$ (see Parameter Properties) and hence we can pre-compute $r_i' = r_i z_i/d$ and subsequently $\lfloor \mathbf{L}_1 \rfloor = \lfloor r_1' - w_1 \rfloor, \lceil \mathbf{R}_1 \rceil = \lceil r_1' + w_1 \rceil$ without needing any neural network and we can skip to computing e_i^+, e_i^-. For iteration $i>1$, we are processing $\mathbf{x}_i^{\neq 0}$, which is a block of contiguous entries from $\mathbf{x}_1$, i.e., the intersection of $\mathbf{x}_1$ with some real interval, and $\mathbf{x}_i$ contains the overall median. We can thus use Lemma 50 by plugging in $d' = d'' = d, \mathbf{x} = \mathbf{x}_1, \mathbf{y} = \mathbf{x}_i, e = e_{i-1}^+, r = d/2$ to conclude that (2b) correctly computes the rank $r_i \in \mathbb{Z}^{>0}$ of the median in the new universe $\mathbf{x}_i^{\neq 0}$ using the *rank computing* (Definition 49, $\mathcal{N}_\delta^{RC}$) neural network i.e., $\mathcal{R}_{r_i}\left(\mathbf{x}_i^{\neq 0}\right) = \text{med}(\mathbf{x}_1)$. Further, from the same Lemma 50 we have that this step will require 1 hidden layer, a width of $\mathcal{O}(d)$ and magnitude of weights bounded by $1/\delta$.

  Next we note $r_i \in [d]$, (from the correct computation of r_i in the previous step) $\left|\mathbf{x}_i^{\neq 0}\right| \in [d]$ (for sufficiently large d by the assumption of "success" of Algorithm 5), $z_i \in [d]$ is a pre-determined constant and $\mathbf{x}_i \in \mathcal{S}_\delta^d$. We apply Lemma 52 with $d' = d, \mathbf{x} = \mathbf{x}_i, r = r_i, b = z_i$ to conclude that the *rank scaling* neural network (Definition 51, $\mathcal{N}_{\delta,z_i}^{RSC}$) in (2b) correctly scales the new rank and produces $r_i' = r_i z_i / \left|\mathbf{x}_i^{\neq 0}\right|$. In the previous step we used *rank computing* (Definition 49, $\mathcal{N}_\delta^{RC}$) with $\mathbf{y} = \mathbf{x}_i$ and hence the quantity $\left|\mathbf{x}_i^{\neq 0}\right|$ has already been computed by one of its layers. Thus, using Lemma 52 with pre-computed $\left|\mathbf{x}_i^{\neq 0}\right|$ we have that this step requires 1 hidden layer, width of $\mathcal{O}(d)$ and magnitude of weights bounded by $\mathcal{O}(\max(dz_i, 1/\delta)) \in \mathcal{O}(\max(d^{1.5}, 1/\delta))$ (see Parameter Properties).

Next we compute $\lfloor \mathbf{L}_i \rfloor = \lfloor r'_i - w_i \rfloor, \lceil \mathbf{R}_i \rceil = \lceil r'_i + w_i \rceil$ using the *ceiling* neural network (53, $\mathcal{N}_d^{CEI}$) on r'_i in parallel (where we use the ceiling network to also compute the floor, using the identity $\lfloor x \rfloor = -\lceil -x \rceil$) and then adding w_i (since w_i is an integer we do not need to include it in the input to $\mathcal{N}_d^{CEI}$). In the previous step we correctly computed $r'_i = r_i z_i / \left|\mathbf{x}_i^{\neq 0}\right|$ where $r_i, z_i, \left|\mathbf{x}_i^{\neq 0}\right|, w_i \in [d]$ and $r_i z_i / \left|\mathbf{x}_i^{\neq 0}\right| \in [-d, d]$ (follows from Parameter Properties and "success" Definition 20) and hence by Lemma 54 the quantities $\lfloor \mathbf{L}_i \rfloor = -\mathcal{N}^{CEI}(-r'_i) - w_i, \lceil \mathbf{R}_i \rceil = \mathcal{N}^{CEI}(r'_i) + w_i$ are computed correctly. Further by Lemma 54 this requires 1 hidden layer, $\mathcal{O}(d)$ width and magnitude of weights bounded by $\mathcal{O}(d)$.

We then compute $\max(\lfloor \mathbf{L}_i \rfloor + 1, 1)$ and $\min(\lceil \mathbf{R}_i \rceil + 1, z_i + 2) = -\max(-\lceil \mathbf{R}_i \rceil - 1, -z_i - 2)$ using the *maximum* neural network (Definition 34, $\mathcal{N}^{MAX}$). This requires 1 hidden layer, $\mathcal{O}(1)$ width and $\mathcal{O}(1)$ weights.

Finally recall that at the end of Step 2(b) we let e_i^- be the element with rank $\mathbf{L}_i$ from S_i if $\mathbf{L}_i \geq 1$ else we let with $e_i^- = 0$. And similarly we let e_i^+ be the element with rank $\mathbf{R}_i$ from S_i if $\mathbf{R}_i \leq z_i$ else we let with $e_i^+ = 1$. It is easy to see that since $S_i \subseteq \mathbf{x}_1 \in \mathcal{S}_\delta^d$ (from the "success" of Algorithm 5) the above step is equivalent to selecting e_i^-, e_i^+ as the elements with ranks $\max(\lfloor \mathbf{L}_i \rfloor + 1, 1), \min(\lceil \mathbf{R}_i \rceil + 1, z_i + 2)$ from $S_i \cup \{0, 1\} \in \mathbb{R}^{z_i + 2}$. We can thus use the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$), letting $d' = z_i + 2, \mathbf{r} = \{\max(\lfloor \mathbf{L}_i \rfloor + 1, 1), \min(\lceil \mathbf{R}_i \rceil + 1, z_i + 2)\}$ and $\mathbf{x} = S_i \cup \{0, 1\}$. From Lemma 46 we conclude that this correctly computes e_i^-, e_i^+. Further, from Lemma 46 this step will require 2 hidden layers, a width of $\mathcal{O}(d^{2z_i}) \in \mathcal{O}(d)$ (see Parameter Properties) and magnitude of weights bounded by $1/\delta$.

- **Step 2(c):** Informally, in this step we zero out entries not lying in the interval $[e_i^-, e_i^+]$. Noting from the earlier step that e_i^-, e_i^+ are elements of $\mathbf{x}_i^{\neq 0} \cup \{0, 1\}$ and $\mathbf{x}_i \in \mathcal{S}_\delta^d$ we can use Lemma 42 with $d' = d, \ell = e_i^-, u = e_i^+$ and $\mathbf{x} = \mathbf{x}_i$ to conclude that the *filtering* neural network (Definition 41, $\mathcal{N}_\delta^F$) used in (2c) correctly filters the entries of $\mathbf{x}_i$, i.e., only the entries of $\mathbf{x}_i$ lying in the range $\left[e_i^-, e_i^+\right]$ are kept unchanged while the rest of the entries are zeroed. This new vector is denoted as $\mathbf{x}_{i+1}$. Thus, Step 2(c) from Algorithm 5 is correctly implemented by (2c). Also from Lemma 42 we have that this step requires 1 hidden layer, a width of $\mathcal{O}(d)$ and magnitude of weights bounded by $1/\delta$.

  Finally, we point out, as promised at the beginning of the proof, that the vector $\mathbf{x}_{i+1}$ output by this step will lie in $\mathcal{S}_\delta^d$: we zero out certain entries from $\mathbf{x}_i$, and from the definition of $\mathcal{S}_\delta^d$, zeroing out entries maintains membership in $\mathcal{S}_\delta^d$; and thus since $\mathbf{x}_i$ was in $\mathcal{S}_\delta^d$ (either from the analysis of the previous iteration for $i > 1$, or because $i = 1$ and $\mathbf{x}_1 \in \mathcal{S}_\delta^d$ by assumption), we conclude our induction step that $\mathbf{x}_{i+1} \in \mathcal{S}_\delta^d$.

From the above neural network implementation of Algorithm 5 we see that each step requires a width of $\mathcal{O}(d)$ and hence the entire neural network implementation requires $\mathcal{O}(d)$ width. For the number of hidden layers required, we first note for the special case of $i = 1$ we skipped directly to the rank selection step in Step (2b). Thus, for $i = 1$ only 3 hidden layers are required. For the subsequent iterations ($i > 1$) we see from the above implementation that 10 hidden layers are required and since we iterate 3 times the total number of hidden layers required is 33. Also, each step of the implementation has magnitude of weights bounded by $\mathcal{O}(\max(d^{1.5}, 1/\delta))$. Finally we

conclude that a "successful" implementation of Algorithm 5 implies by Definition 20 that $\mathbf{x}_5$ will have at most $6^4 d^{0.16}$ non-zero entries with $\mathrm{med}(\mathbf{x}_1)$ as one of its non-zero entries.

■

Algorithm 6: Neural Network Implementation of Algorithm 5

Input: Entirely non-zero vector $\mathbf{x}_1 \in \mathcal{S}^d_\delta$, with uniformly randomly permuted entries.
Output: Vector $\mathbf{x}' \in \mathbb{R}^d$.
`sparsifyingFunction(`$\mathbf{x}_1$`):`

1. For $i \in \{1, \ldots, 4\}$ define the triples of exponents $(y'_i, z'_i, w'_i) = (1, 0.5, 0.26), (0.76, 0.5, 0.26), (0.52, 0.5, 0.26), (0.28, 0.26, 0.14)$, and also let $y'_5 = 0.16$; then define the parameters $z_i = \lceil d^{z'_i} \rceil, w_i = \lfloor d^{w'_i} \rfloor, y_i = d^{y'_i}$.

2. For $i \in \{1, 2, 3, 4\}$ do:

   (a) From $\mathbf{x}_i$ we pick z_i non-zero entries as follows: if $i = 1$ then take the first z_i entries. Otherwise for $i > 1$, divide $\mathbf{x}_i$ into blocks of size $\lceil 3^i d^{1.01}/y_i \rceil$, and from the first $\lceil z_i/d^{0.01} \rceil$ such blocks, choose the first $\lceil d^{0.01} \rceil$ non-zero elements if possible (and FAIL otherwise), but discard elements after z_i total elements have been chosen. Denote this sample set of size z_i by S_i.

   - We implement this with our *non-zero element shortlisting* (Definition 47, $\mathcal{N}^{NZES}_\delta$) where we do the shortlisting operation from each block in parallel.

   (b) Define r_i to be the rank in $\mathbf{x}_i^{\neq 0}$ of the overall median, $\mathrm{med}(\mathbf{x}_1)$. Compute $\mathbf{L}_i = \frac{r_i}{|\mathbf{x}_i^{\neq 0}|} z_i - w_i$, and $\mathbf{R}_i = \frac{r_i}{|\mathbf{x}_i^{\neq 0}|} z_i + w_i$. Finally compute $\lfloor \mathbf{L}_i \rfloor$ and $\lceil \mathbf{R}_i \rceil$.

   - First we compute r_i using the *rank computing* neural network (Definition 49, $\mathcal{N}^{RC}_\delta$) using the first element of S_i as a non-zero entry of $\mathbf{x}_i$.
   - After that we scale the rank using the *rank scaling* neural network (Definition 51, $\mathcal{N}^{RSC}_{\delta, z_i}$) to compute $\frac{r_i}{|\mathbf{x}_i^{\neq 0}|} z_i$.
   - Finally we compute $\lfloor \mathbf{L}_i \rfloor$ and $\lceil \mathbf{R}_i \rceil$ using the *floor* neural network (Definition 53, $\mathcal{N}^{CEI}_d$) and then compute $\mathbf{L}'_i = \max(\lfloor \mathbf{L}_i \rfloor + 1, 1)$ and $\mathbf{R}'_i = \min(\lceil \mathbf{R}_i \rceil + 1, z_i + 2)$ using the *maximum* neural network(Definition 34, $\mathcal{N}^{MAX}$).
   - For $i = 1$, the above steps are redundant and we use we use pre-computed values as $r_i = d/2, |\mathbf{x}_i^{\neq 0}| = d$.
   - Then we extract $e_i^+ = \mathcal{R}_{\mathbf{R}'_i}(S'_i)$ and $e_i^- = \mathcal{R}_{\mathbf{L}'_i}(S'_i)$ where $S'_i = S_i \cup \{0, 1\}$. We do the above operation using the *rank selection* neural network (Definition 45, $\mathcal{N}^{RS}_\delta$).

   (c) Create a new copy of $\mathbf{x}_i$ represented as $\mathbf{x}_{i+1}$, where in $\mathbf{x}_{i+1}$ all entries with values $< e_i^-$ or values $> e_i^+$ are changed to 0.

   - We do this using our *filtering* neural network (Definition 41, $\mathcal{N}^F_\delta$).

3. Return $\mathbf{x}' \leftarrow \mathbf{x}_5$ which contains at most $6^4 d^{0.16}$ non-zero entries with high probability one of which is $\mathrm{med}(\mathbf{x}_1)$.

Algorithm 7: Given any (d, ε) sparse vector $\mathbf{x}' \in \mathcal{S}_\delta^d$ return $\mathbf{x}'' \in \mathbb{R}^{d^\varepsilon}$ containing all non-zero entries of $\mathbf{x}'$

Input: Vector: $\mathbf{x}' \in \mathcal{S}_\delta^d$, $\mathbf{x}'$ is (d, ε) sparse
Output: Vector $\mathbf{x}'' \in \mathcal{S}_\delta^{d^\varepsilon}$
`hashingFunction`$(\mathbf{x}', \varepsilon, \gamma)$:

1. Letting p be the smallest prime larger than $d^{2\varepsilon+\gamma}$, use each of the p hash functions from the hash function family $\mathcal{H}_{d,2\varepsilon+\gamma}$ (Definition 31) to map $\mathbf{x}'$ to vectors of size p, creating a new vector $\mathbf{y} \in \mathbb{R}^{p^2}$.
   - We do this using p copies of the *hashing* neural network (Definition 55, $\mathcal{N}_\delta^H$).
2. Identify and extract the output of the first of the p hash functions that caused zero collisions, calling this output $\mathbf{y}' \in \mathbb{R}^p$.
   - We do this by using the *block extraction* neural network (Definition 57, $\mathcal{N}_{\delta,p}^{BE}$)
3. Select the d^ε largest entries from $\mathbf{y}'$ and denote this vector as $\mathbf{x}''$.
   (a) We do this by using the *rank selection* neural network (Definition 45 $\mathcal{N}_\delta^{RS}$).
4. Return $\mathbf{x}''$

D.2. Hashing step

Recall that our ReLU neural network outlined by Algorithm 6 will return a very sparse vector $\mathbf{x}'$, with d entries but at most $6^4 d^{0.16}$ non-zero entries, one of which is the true median (with high probability, as guaranteed by Proposition 22). We next explain how to "hash" these non-zero entries, deterministically and without collisions, into a shorter vector with at most $\sqrt{d}$ entries; given this, it is straightforward to conclude our algorithm with a brute-force quadratic-width sorting neural network (Definition 45, $\mathcal{N}_\delta^{RS}$) to return the median.

Hashing is typically viewed as a two-stage process where first a hash function h is randomly selected from a hash function family $\mathcal{H}$, and then h is applied to the input vector $\mathbf{x}'$. However, in our neural network setting we do not have access to randomness. Instead, we explicitly define (and precompute) a small hash function family $\mathcal{H}$ and deterministically evaluate $h(\mathbf{x}')$ for *all* $h \in \mathcal{H}$, showing mathematically that at least one such h must have 0 collisions, and algorithmically identifying and using this h in our neural network.

Proposition 23 *The algorithm* `hashingFunction`$(\mathbf{x}', \varepsilon, \gamma)$ *(Algorithm 7), when given any* (d, ε) *sparse vector* $\mathbf{x}' \in \mathcal{S}_\delta^d$ *as input, where* $d \geq \frac{1}{(2\varepsilon+\gamma)^{1/\gamma}}$*, returns* $\mathbf{x}'' \in \mathbb{R}^{d^\varepsilon}$ *containing all the non-zero entries of* $\mathbf{x}'$ *and is padded with* 0*'s if there are less than* d^ε *non-zero entries in* $\mathbf{x}'$*. Moreover, the function can be implemented by a neural network with width* $\mathcal{O}\left(d^{\max(4\varepsilon+2\gamma,1)}\right)$*, and depth* 7 *with magnitude of weights bounded by* $1/\delta$*.*

Proof Intuitively, the function uses hash functions h in the hash family $\mathcal{H}_{d,2\varepsilon+\gamma'}$ (Definition 31) to hash the entries of a (d,ε) sparse $\mathbf{x}'$ to $\mathcal{O}\left(d^{2\varepsilon+\gamma}\right)$ locations, looking for a hash function that produces zero collisions on the non-zero entries of the given input $\mathbf{x}'$. By Lemma 33 such a hash function exists in $\mathcal{H}_{d,2\varepsilon+\gamma'}$ provided $d \geq \frac{1}{(2\varepsilon+\gamma)^{1/\gamma}}$. We then use the results of this hash function to extract the non-zero entries of $\mathbf{x}'$. The neural networks used to do these operations are elaborated below. Let p be the smallest prime number greater than $d^{2\varepsilon+\gamma}$; because there is a prime in any positive integer interval $[\ell, 2\ell]$ (Bertrand's Postulate), we have that $p \in \mathcal{O}(d^{2\epsilon+\gamma})$.

- **Step 1:** Since the dimension d of the problem is fixed, and ε, γ are pre-determined parameters, we can thus implement each of the p hash functions $h : \mathbb{R}^d \to \mathbb{R}^p$ in $\mathcal{H}_{d,2\varepsilon+\gamma}$ (Definition 31) using a pre-determined *hashing* neural network (Definition 55, $\mathcal{N}_h^H$) as proven in Lemma 56. Thus we implement all p hash functions in $H_{d,2\varepsilon+\gamma}$ using 1 hidden layer and width of $p^2 \in \mathcal{O}\left(d^{4\varepsilon+2\gamma}\right)$ with magnitudes of weights in $\{0,1\}$.

  Further, by our assumption, the input $\mathbf{x}'$ is (d,ε) sparse and thus Lemma 33 guarantees that at least one of the hash functions in $\mathcal{H}_{d,2\varepsilon+\gamma}$ hashes all the non-zero elements of $\mathbf{x}'$ to *distinct* locations in $[p]$.

  To prepare for the next step, in parallel with this we should count the number of non-zero entries s in the input: let $s = \mathcal{N}_\delta^{NZC}(\mathbf{x}')$, which by Lemma 38 uses 1 hidden layer (in parallel with the previous construction, for no extra depth), width $2d'$, and weights of magnitude $\frac{1}{\delta}$.

- **Step 2:** We search for and extract the result of the hash function that produced zero collisions using the *block extraction* neural network $\mathcal{N}_{\delta,p}^{BE}(\mathbf{y}, s)$, which by Lemma 58 will operate correctly and use 3 hidden layers, width p^2, and have magnitudes of weights bounded by $\frac{1}{\delta}$. This will return a vector $\mathbf{y}' \in \mathbb{R}^p$ containing the $s \leq d^\varepsilon$ non-zero entries of our original $\mathbf{x}'$.

- **Step 3:** We then select the non-zero entries of $\mathbf{y}'$ with the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$). Namely, since we want to return the s non-zero entries from the nonnegative vector $\mathbf{y}' \in \mathbb{R}^p$, we simply ask for the elements of ranks $p-s+1, \ldots, p$, letting $\mathbf{x}'' = \mathcal{N}_\delta^{RS}(\mathbf{y}', (p-s+1, \ldots, p))$, which by Lemma 46 correctly returns the answer using 2 hidden layers, a width of $\mathcal{O}\left(d^{4\varepsilon+2\gamma}\right)$ and magnitude of weights bounded by $1/\delta$.

Thus, the neural network describe here correctly implements Algorithm 7 using a total of 6 hidden layers, a width of $\mathcal{O}(\max\left(d, d^{4\varepsilon+2\gamma}\right))$, and has magnitudes of weights bounded by $\mathcal{O}(\frac{1}{\delta})$. ■

Proposition 24 *For any dimension $d > 0$ and any $\delta > 0$ the function* `computeMedian(x)` *(Algorithm 8) where $\mathbf{x} \in \mathcal{S}_\delta^d$ and is entirely non-zero, with uniformly randomly permuted entries, returns* $\mathrm{med}(\mathbf{x})$ *with probability at least* $1 - \exp\left(-d^{\Omega(1)}\right)$. *In all other cases, the value returned by* `computeMedian(x)` *lies in* $[0,1]$. *Moreover, the function can be implemented by a ReLU neural network with width $\mathcal{O}(d)$ and depth 45 hidden layers with magnitude of weights bounded by* $\mathcal{O}(\max\left(d^{1.5}, 1/\delta\right))$.

Proof At a high level, the neural network `computeMedian()`, as outlined by Algorithm 8, first sparsifies the input $\mathbf{x}$ using the neural network `sparsifyingFunction()` (outlined by Algorithm 6) producing $\mathbf{x}'$. Following this, it reduces the dimensionality of $\mathbf{x}'$ using `hashingFunction()`

Algorithm 8: Given $\mathbf{x}$ compute its median

Input: Entirely non-zero vector $\mathbf{x} \in \mathcal{S}_\delta^d$, with uniformly randomly permuted entries.
Output: $\text{med}(\mathbf{x})$ with probability $1 - \exp\left(-d^{\Omega(1)}\right)$ or a value $\in [0, 1]$
`computeMedian(x)`:

1. $\mathbf{x}' \leftarrow$ `sparsifyingFunction`$(\mathbf{x})$, (Algorithm 6)
   - Requires 33 hidden layers, width of $\mathcal{O}(d)$ and magnitude of weights bounded by $\mathcal{O}\left(\max(1/\delta), d^{1.5}\right)$ to implement using a neural network.
   - Returns $(d, 0.16 + 4\log_d 6)$ sparse vector $\mathbf{x}' \in \mathcal{S}_\delta^d$ with probability at least $1 - \exp\left(-\left(d^{\Omega(1)}\right)\right)$. Also the entries of $\mathbf{x}'$ come from a contiguous block of entries in $\mathbf{x}$'s sorted order.
2. $\mathbf{x}'' \leftarrow$ `hashingFunction`$(\mathbf{x}')$ (Algorithm 7)
   - Requires 6 hidden layers, width of $\mathcal{O}(d)$ and magnitude of weights bounded by $1/\delta$ to implement using a neural network.
   - Returns $\mathbf{x}'' \in \mathbb{R}^{6^4 d^{0.16}}$ which contains all the non-zero entries of $\mathbf{x}'$ if $\mathbf{x}'$ is a $(d, 0.16 + 4\log_d 6)$ sparse vector and $d \geq \frac{1}{(2\varepsilon+\gamma)^{1/\gamma}}$ (we choose $\varepsilon = 0.16 + 4\log_d 6$ and choose $\gamma = 0.18$).
3. Compute relative rank r'' of $\mathcal{R}_{d/2}(\mathbf{x})$ in $\mathbf{x}''$.
   - We do this using our *rank selection* (Definition 45, $\mathcal{N}_\delta^{RS}$) and *rank computing* neural network (Definition 49, $\mathcal{N}_\delta^{RC}$) requiring 3 hidden layers, width of $\mathcal{O}(d)$ and magnitude of weights bounded by $1/\delta$.
4. Output $\min\left(1, \max\left(0, \mathcal{R}_{r''}(\mathbf{x}'')\right)\right)$
   - We do this by using the *rank selection* neural network (Definition 45 $\mathcal{N}_\delta^{RS}$), letting $m \leftarrow \mathcal{R}_{r''}(\mathbf{x}'')$, followed by computing and returning $[m]_+ - [m-1]_+$ requiring 3 hidden layers.

(as outlined by Algorithm 7) while preserving the non-zero elements, outputting $\mathbf{x}''$. Finally, it determines the rank r'' of the overall median relative to $\mathbf{x}''$ and then computes the median from these entries by a brute force all-pairs algorithm, provided $\text{med}(\mathbf{x})$ is indeed in this subset. We trim the output to the interval $[0, 1]$ so that even the rare cases where the neural network fails do not contribute much to the mean squared error.

Formally, since entries of $\mathbf{x}$ are uniformly randomly permuted and $\mathbf{x}$ is entirely non-zero we have by Proposition 21 that Algorithm 5 is "successful" as stated in Definition 20 with probability at least $1 - \exp\left(-d^{\Omega(1)}\right)$. Further, using the fact $\mathbf{x} \in \mathcal{S}_\delta^d$ and is entirely non-zero, we have from Proposition 22 that Algorithm 5 is correctly implemented by the neural network described

in Algorithm 6, which has 33 hidden layers, width $\mathcal{O}(d)$, and magnitudes of weights bounded by $\mathcal{O}\left(\max(1/\delta), d^{1.5}\right)$.

In the case of "success" (Definition 20), the number of non-zero entries in $\mathbf{x}'$ returned by Algorithm 6 is at most $6^4 d^{0.16}$ with $\text{med}(\mathbf{x})$ being one of them, and these non-zero entries form a contiguous block from the sorted version of $\mathbf{x}$ (Proposition 21). We use Algorithm 7, with parameters $\varepsilon = 0.16 + 4\log_d 6, \gamma = 0.18$, to hash the non-zero locations of $\mathbf{x}'$ and extract its non-zero entries $\mathbf{x}'^{\neq 0}$. We denote the extracted vector as $\mathbf{x}'' \in \mathbb{R}^{6^4 d^{0.16}}$, which contains all the non-zero entries of $\mathbf{x}'$ and is padded with 0's when there are less than $6^4 d^{0.16}$ non-zero entries. From Proposition 23 we have that Algorithm 7 can be implemented by a neural network of width $\mathcal{O}(d^{\max(4\varepsilon+2\gamma,1)}) \in \mathcal{O}(d)$—by our choice of ϵ, γ—and 6 hidden layers, with magnitudes of weights bounded by $1/\delta$.

Use the *rank selection* neural network $\mathcal{N}_\delta^{RS}$ to select the maximum element e of $\mathbf{x}''$, using 2 hidden layers, width $\mathcal{O}(d^{0.32})$, and magnitudes of weights bounded by $\frac{1}{\delta}$, by Lemma 46.

Recall from the proof of Proposition 22 that the non-zero entries of $\mathbf{x}''$ comprise a contiguous block of elements in $\mathbf{x}$ when $\mathbf{x}$ is sorted; also letting $d' = 6^4 d^{0.16}$, we have that $\mathbf{x}'' \in \mathcal{S}_\delta^{d'}$ since $\mathbf{x}''^{\neq 0} \subseteq \mathbf{x}$. Thus the *rank computing* neural network (Definition 49, $\mathcal{N}_\delta^{RC}$) accurately computes the rank r' of $\text{med}(\mathbf{x})$ in $\mathbf{x}''$ (using e as an auxiliary input) using 1 hidden layer, a width of $\mathcal{O}(d)$ and magnitudes of weights bounded by $1/\delta$.

We then compute $\mathcal{R}_{r''}(\mathbf{x}'')$ using the *rank selection* neural network (Definition 45, $\mathcal{N}_\delta^{RS}$). Using Lemma 46 and plugging in $d' = 6^4 d^{0.16}$ we have that this operation uses 2 hidden layers, a width of $\mathcal{O}(d)$ and magnitudes of weights bounded by $1/\delta$.

It is easy to check that $[x]_+ - [x-1]_+ = \min(1, \max(0, x)) = x$ when $x \in [0,1]$. Since for our input, the median will always be in $[0,1]$, this final trimming step will never modify the returned median m in the cases that m is accurate; but in all other cases the output is in the interval $[0,1]$ concluding the first claim of the proposition.

This final step clearly requires 1 hidden layer, $\mathcal{O}(1)$ width and $\mathcal{O}(1)$ magnitude of weights.

Finally, we put everything together to see that implementing `computeMedian()` requires a 45 hidden layers, width of $\mathcal{O}(d)$ and magnitude of weights bounded by $\mathcal{O}(d^{1.5}, 1/\delta)$ concluding the proof.

■

D.3. Proof of Theorem 4

Proof We use Proposition 24 to show that, when the input $\mathbf{x}$ is δ-separated and appropriately bounded, the neural network outlined in Algorithm 8 will accurately compute the median with probability at least $1 - \exp(-d^{\Omega(1)})$, and always returns values in $[0,1]$. We combine this with the bounds on the probability that a randomly chosen input will fail the input requirements of Proposition 24: from Lemma 26, plugging in $d' = d, \delta := \frac{\epsilon}{3d^2}$ we have that $\mathbb{P}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[\mathbf{x} \notin \mathcal{S}^d_{\epsilon/3d^2}\right] \leq \epsilon$. Thus, from the union bound on these two failure modes,

$$\mathbb{P}_{\mathbf{x}\sim\mathcal{U}([0,1]^d)}\left[\mathcal{N}(\mathbf{x}) \neq \text{med}(\mathbf{x})\right] \leq \epsilon + \exp\left(-d^{\Omega(1)}\right).$$

In the cases that $\mathcal{N}(\mathbf{x}) \neq \text{med}(\mathbf{x})$, since both the true median and our algorithm's returned answer are in the range $[0,1]$, our error is at most 1, and thus our squared error is also at most

1. Thus the mean squared error is at most the probability of failure, which we bounded above as $\epsilon + \exp\left(-d^{\Omega(1)}\right)$, concluding the proof of the first part of the theorem.

The neural network construction outlined in Proposition 24 has 45 hidden layers, width of $\mathcal{O}(d)$ and magnitudes of weights bounded by $\mathcal{O}(d^{1.5}, 1/\delta) \in \mathcal{O}(d^2/\epsilon)$ concluding the proof. ■

Appendix E. Lower bounds proofs

E.1. Proof of Theorem 7

Let $r \leq d-1$, and let $\mathcal{N}$ be as in the theorem statement, where the input dimension is $d+r-1$. Namely, it is a depth-k, width-n ReLU network satisfying

$$\mathcal{N}(\mathbf{x}) = \mathcal{R}_r(\mathbf{x})$$

for all $\mathbf{x} \in [0,1]^{d+r-1}$. Suppose by contradiction that

$$n < \frac{1}{40}(d+r-1)^{1+\frac{1}{2^{k-2}-1}}.$$

Given an input $\mathbf{x}' \in [0,1]^d$, we concatenate it with $r-1$ ones to obtain the vector $\mathbf{x}''$, and feed this input to $\mathcal{N}$. Since the original d inputs are not greater than the $r-1$ added inputs, the rank-r element is necessarily the maximum of the original d inputs, and since all $d+r-1$ coordinates are in the interval $[0,1]$, we have by our assumption on $\mathcal{N}$ that

$$\mathcal{N}(\mathbf{x}'') = \max(\mathbf{x}')$$

for all $\mathbf{x}' \in [0,1]^d$. Note that $r = d-1$ entails that $\mathcal{N}$ computes the median, so for $r \in [d-1]$ we can compute any rank between the median and the maximum. To compute lower ranks, one can simply pad with zeros instead of ones, in which case taking $r \in [d-1]$ computes all the ranks between the minimum and the median, hence the assumption that $r \leq d-1$ does not impose limits on the rank that we wish to compute.

We now construct a neural network $\mathcal{N}'$ that receives $\mathbf{x}'$ as input, rather than the $(d+r-1)$-dimensional $\mathbf{x}''$. This is easily achieved by substituting all the $r-1$ ones and modifying the bias term of the neurons in the first hidden layer, which does not change the architecture, and therefore of $\mathcal{N}'$ has the same width and depth as $\mathcal{N}$.

We compute

$$n < \frac{1}{40}(d+r-1)^{1+\frac{1}{2^{k-2}-1}} \leq \frac{1}{40}2^{1+\frac{1}{2^{k-2}-1}}d^{1+\frac{1}{2^{k-2}-1}} \leq \frac{1}{10}d^{1+\frac{1}{2^{k-2}-1}},$$

where the second inequality holds since $r \leq d-1$, and the last inequality holds since $k \geq 3$ which entails that the exponent is at most 2. But this contradicts Theorem 6, so it must hold that

$$n \geq \frac{1}{40}(d+r-1)^{1+\frac{1}{2^{k-2}-1}},$$

where a change of variables $d+r-1 \mapsto d$ concludes the proof of the theorem.

E.2. Proof of Theorem 8

Proof Let $\mathcal{N} : \mathbb{R}^{2d-1} \to \mathbb{R}$ be as in the theorem statement, where the input dimension is $2d-1$. Namely, it holds that

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^{2d-1}\right)}\left[\left(\mathcal{N}(\mathbf{x})-\mathrm{med}(\mathbf{x})\right)^2\right]\leq\varepsilon,$$

for a depth-k, width-$w(2d-1)\leq w(2d)$ σ-neural network $\mathcal{N}$ with weights bounded by $M(2d-1)\leq M(2d)$. Suppose that $\mathbf{x}_0\sim\mathcal{U}\left([0,1]^d\right)$, and consider the process of concatenating the input $\mathbf{x}_0$ with $\mathbf{x}_1\sim\mathcal{U}\left(\left[1,2-\frac{1}{2d-1}\right]^{d-1}\right)$ added coordinates, applying a uniformly sampled permutation on $[2d-1]$ on the resulting vector, and finally scaling it to the unit interval by multiplying it by $\frac{2d-1}{4d-3}$, to receive the outcome denoted by $\mathbf{x}'\in\mathbb{R}^{2d-1}$. Let A denote the resulting set of possible outcomes of this process, and note that applying this process to the uniform distribution over $[0,1]^d$ results in a uniform distribution over A.

We proceed by first lower bounding the probability that $\mathbf{x}\sim\mathcal{U}\left([0,1]^{2d-1}\right)$ will satisfy $\mathbf{x}\in A$. This is exactly the probability of successfully drawing a coordinate from $\left[0,\frac{d}{2d-1}\right]$ in precisely d draws out of $2d-1$, which is captured by the following binomial probability

$$\mathbb{P}\left[\mathbf{x}\in A\right]=\binom{2d-1}{d}\left(\frac{d}{2d-1}\right)^d\left(\frac{d-1}{2d-1}\right)^{d-1}=\frac{(2d-1)!}{d!(d-1)!}\left(\frac{d}{2d-1}\right)^d\left(\frac{d-1}{2d-1}\right)^{d-1}.$$

Using standard Stirling bounds (Lemma 25), we bound the above probability by lower bounding $(2d-1)!$ and upper bounding $d!$ and $(d-1)!$, yielding

$$\begin{aligned}\mathbb{P}\left[\mathbf{x}\in A\right]&\geq\frac{\sqrt{2\pi(2d-1)}\exp(-(2d-1))}{\sqrt{2\pi d}\exp(-d)\exp\left(\frac{1}{12d}\right)\sqrt{2\pi(d-1)}\exp(-(d-1))\exp\left(\frac{1}{12(d-1)}\right)}\\&\geq\frac{\sqrt{2d-2}}{\sqrt{2\pi d(d-1)}\exp\left(\frac{1}{12d}+\frac{1}{12(d-1)}\right)}\geq\frac{1}{\sqrt{\pi d}\exp\left(\frac{1}{8}\right)}\geq\frac{1}{2\sqrt{\pi d}},\end{aligned}\tag{1}$$

where the d^d, $(d-1)^{d-1}$ and $(2d-1)^{2d-1}$ terms cancel in the first inequality, the penultimate inequality holds for all $d\geq 2$, and the last inequality follows since $\exp(-1/8)\leq 0.5$.

With the above, we now turn to bound the approximation error as follows

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}'\sim\mathcal{U}(A)}\left[\left(\mathcal{N}(\mathbf{x}')-\max(\mathbf{x}_0)\right)^2\right]&=\mathop{\mathbb{E}}_{\mathbf{x}'\sim\mathcal{U}(A)}\left[\left(\mathcal{N}(\mathbf{x}')-\mathrm{med}(\mathbf{x}')\right)^2\right]\\&=\int_{\mathbf{x}'\in A}\left(\mathcal{N}(\mathbf{x}')-\mathrm{med}(\mathbf{x}')\right)^2\left(Pr[\mathbf{x}'\in A]\right)^{-1}d\mathbf{x}'\\&\overset{\text{Equation (1)}}{\leq}2\sqrt{\pi d}\int_{\mathbf{x}''\in[0,1]^{2d-1}}\left(\mathcal{N}(\mathbf{x}'')-\mathrm{med}(\mathbf{x}'')\right)^2\mathbb{1}\left\{\mathbf{x}''\in A\right\}d\mathbf{x}\\&\leq 2\sqrt{\pi d}\int_{\mathbf{x}''\in[0,1]^{2d-1}}\left(\mathcal{N}(\mathbf{x}'')-\mathrm{med}(\mathbf{x}'')\right)^2d\mathbf{x}\\&=2\sqrt{\pi d}\mathop{\mathbb{E}}_{\mathbf{x}''\sim\mathcal{U}([0,1]^{2d-1})}\left[\left(\mathcal{N}(\mathbf{x}'')-\mathrm{med}(\mathbf{x}'')\right)^2\right]\leq 2\sqrt{\pi d}\varepsilon.\end{aligned}\tag{2}$$

Now, assume by contradiction that

$$\mathop{\mathbb{E}}_{\mathbf{x}_0\sim\mathcal{U}(A)}\left[\left(\mathcal{N}(\tau(\mathbf{x}_0,\mathbf{x}_1))-\max(\mathbf{x}_0)\right)^2\right]>2\sqrt{\pi d}\varepsilon$$

for all $\mathbf{x}_1$ and permutations τ. Then by the law of total expectation, this implies that

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}'\sim\mathcal{U}(A)}\left[\left(\mathcal{N}(\mathbf{x}')-\max(\mathbf{x}_0)\right)^2\right]&=\mathop{\mathbb{E}}_{\mathbf{x}_1}\left[\mathop{\mathbb{E}}_{\mathbf{x}_0}\left[\left(\mathcal{N}(\tau(\mathbf{x}_0,\mathbf{x}_1))-\max(\mathbf{x}_0)\right)^2\right]|\mathbf{X}_1=\mathbf{x}_1,\tau\right]\\&>\mathop{\mathbb{E}}_{\mathbf{x}_1}\left[2\sqrt{\pi d}\varepsilon\right]=2\sqrt{\pi d}\varepsilon,\end{aligned}$$

contradicting Equation (2). We thus have that there exists some $\mathbf{x}_1$ and permutation τ on $[2d-1]$ such that

$$\mathop{\mathbb{E}}_{\mathbf{x}_0\sim\mathcal{U}\left(\left[0,\frac{2d-1}{4d-3}\right]^d\right)}\left[\left(\mathcal{N}(\tau(\mathbf{x}_0,\mathbf{x}_1))-\max(\mathbf{x}_0)\right)^2\right]\leq 2\sqrt{\pi d}\varepsilon.$$

Since we can substitute $\mathbf{x}_1$ in the first hidden layer of $\mathcal{N}$ and simulate the permutation τ by composing it with the weights of the first hidden layer, it follows that there exists a σ-neural network $\mathcal{N}''$ such that

$$\mathop{\mathbb{E}}_{\mathbf{x}_0\sim\mathcal{U}\left(\left[0,\frac{2d-1}{4d-3}\right]^d\right)}\left[\left(\mathcal{N}''(\mathbf{x})-\max(\mathbf{x})\right)^2\right]\leq 2\sqrt{\pi d}\varepsilon.$$

Lastly, we rescale the hidden layer weights of $\mathcal{N}''$ by 0.5 and its output neuron by 2 to obtain a neural network $\mathcal{N}'(\mathbf{x})=2\mathcal{N}''(0.5\mathbf{x})$ whose domain is multiplied by 2 and satisfies

$$\mathop{\mathbb{E}}_{\mathbf{x}_0\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}'(\mathbf{x})-\max(\mathbf{x})\right)^2\right]\leq 8\sqrt{\pi d}\varepsilon,$$

where the accuracy is rescaled according to Safran et al. (2019, Theorem 9). We note that $\mathcal{N}'$ maintains the same depth and width of $\mathcal{N}$, and has its weights multiplied by at most 2.

■

E.3. Proof of Theorem 14

Proof Assume d is even w.l.o.g. and consider the matrix of first hidden layer weights $W\in\mathbb{R}^{k\times d}$. Since $k\leq d-1$ by our assumption, we have $\dim(\ker(W))\geq 1$ (for the case d is odd using a slightly different choice of parameters in the below arguments will ensure we obtain the same result asymptotically). Let $\mathbf{v}=(v_1,\ldots,v_d)\in\ker(W)$ such that $\|\mathbf{v}\|_2=1$ and assume w.l.o.g. $\max(\mathbf{v})=v_1$. Consider the triangular matrix P below

$$P\coloneqq\frac{1}{2}\begin{pmatrix}\frac{1}{d}v_1 & 0 & 0 & \cdots & 0 & 0\\ \frac{1}{d}v_2 & 1-\frac{2}{d} & 0 & \cdots & 0 & 0\\ \frac{1}{d}v_3 & 0 & 1-\frac{2}{d} & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \frac{1}{d}v_{d-1} & 0 & 0 & \cdots & 1-\frac{2}{d} & 0\\ \frac{1}{d}v_d & 0 & 0 & \cdots & 0 & 1-\frac{2}{d}\end{pmatrix},$$

and a vector $\mathbf{b}_i$ whose first coordinate is $\frac{1}{2}\left(1-\frac{1}{d}\right)$, while $\frac{d}{2}-1$ entries have a value of $\frac{1}{2d}$ and $\frac{d}{2}$ entries have a value of $\frac{1}{2}+\frac{1}{2d}$ where i is used to index the set of all possible $\binom{d-1}{d/2}$ choices for entries with value $\frac{1}{2}+\frac{1}{2d}$. For ease of exposition, w.l.o.g. we consider the case where the first $\frac{d}{2}-1$ entries after the first coordinate is $\frac{1}{2d}$ and call this vector $\mathbf{b}_1$.

$$\mathbf{b}_1 \coloneqq \frac{1}{2}\begin{pmatrix} 1-\frac{1}{d} \\ \frac{1}{d} \\ \frac{1}{d} \\ \vdots \\ 1+\frac{1}{d} \\ 1+\frac{1}{d} \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\end{matrix}}\right\} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\end{matrix}}\right\} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\end{matrix}}\right\} \end{matrix} \begin{matrix} 1 \text{ time} \\ \\ \frac{d}{2}-1 \text{ times} \\ \\ \\ \frac{d}{2} \text{ times} \end{matrix}.$$

Now consider the set $\mathcal{P}_1 = \{P\mathbf{x}+\mathbf{b}_1 : \mathbf{x} \in [0,1]^d\}$. It is easy to verify that $\mathcal{P}_1 \subset [0,1]^d$ for $d \geq 2$. We also have,

$$\text{med}(\mathbf{p}) = p_1, \quad \forall \mathbf{p} \in \mathcal{P}_1. \tag{3}$$

The above follows by noting that every $\mathbf{p} \in \mathcal{P}_1$ can be written as $P\mathbf{x}+\mathbf{b}_1$ for some $\mathbf{x} \in [0,1]^d$ by definition. Hence, for some $\mathbf{x} \in [0,1]^d$ and p_i, $2 \leq i \leq d/2$ we have

$$p_i = \frac{1}{2d}v_i x_1 + \frac{1}{2}\left(1-\frac{2}{d}\right)x_i + \frac{1}{2d} \leq \frac{1}{2d}v_1 x_1 + \frac{1}{2}\left(1-\frac{1}{d}\right) = p_1$$

since $v_i \leq v_1, x_i \in [0,1]$ and $d \geq 2$ in our setup. Noting that $x_i \in [0,1]$, $1-\frac{2}{d} \geq 0$ and $\|\mathbf{v}\|_2 = 1$ implying $|v_j - v_k| \leq \sqrt{2}$, for the case p_i, $i > d/2$ we have,

$$p_i - p_1 = \frac{1}{2d}(v_i - v_1)x_1 + \frac{1}{2}\left(1-\frac{2}{d}\right)x_i + \frac{1}{d} \geq \frac{1}{2d}(v_i - v_1)x_1 + \frac{1}{d} \geq -\frac{\sqrt{2}}{2d} + \frac{1}{d} > 0.$$

It easy to check for the different choices of $\mathbf{b}_i, \forall i \in \left[\binom{d-1}{d/2}\right]$'s the corresponding set $\mathcal{P}_i$ has the p_1 as the median, with the only change being the indices of the entries greater than p_1. Also by a standard Stirling bound (Lemma 25) we have,

$$\begin{aligned}\binom{d-1}{d/2} \geq \frac{c}{\sqrt{d}}\frac{(d-1)^{d-1}}{\left(\frac{d}{2}\right)^{\frac{d}{2}}\left(\frac{d}{2}-1\right)^{\frac{d}{2}-1}} \geq \frac{c}{\sqrt{d}}\frac{(d-1)^{d-1}}{\left(\frac{d}{2}\right)^{\frac{d}{2}}\left(\frac{d}{2}\right)^{\frac{d}{2}-1}} \geq \frac{c}{\sqrt{d}}2^{d-1}\left(1-\frac{1}{d}\right)^{d-1} \\ \geq \frac{c}{10\sqrt{d}}2^{d-1}, d \geq 2,\end{aligned} \tag{4}$$

where c is a universal constant. Thus, we have at least $\frac{c}{10\sqrt{d}}2^{d-1}$ choices of $\mathbf{b}_i$.

We also have if $i \neq j$ then $\mathcal{P}_i \cap \mathcal{P}_j = \phi$. This follows by noting that if $i \neq j$ then there exists a coordinate k such that $(\mathbf{b}_i)_k \neq (\mathbf{b}_j)_k, k > 1$ and from the definition of $\mathcal{P}_i$ it is easy to verify that for $\mathbf{p} \in \mathcal{P}_i$ either $p_k \leq p_1$ or $p_k > p_1$ depending on whether $(\mathbf{b}_i)_k = \frac{1}{2d}$ or $\frac{1}{2}+\frac{1}{2d}$ respectively. Thus, it follows that if $i \neq j$ then $\mathcal{P}_i \cap \mathcal{P}_j = \phi$ as there exists a coordinate p_k whose range of values in $\mathcal{P}_i$ and in $\mathcal{P}_j$ are completely disjoint. Also since $\mathcal{P}_i \subseteq [0,1]^d$ we have $\cup_i \mathcal{P}_i \subseteq [0,1]^d$.

Putting all of the above together we get,

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{p}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{p})-\operatorname{med}(\mathbf{p})\right)^2\right] &\overset{\cup_i\mathcal{P}_i\subseteq[0,1]^d}{\geq} \int_{\cup_i\mathcal{P}_i}\left(\mathcal{N}(\mathbf{p})-\operatorname{med}(\mathbf{p})\right)^2 d\mathbf{p}\\ &= \sum_i\int_{\mathcal{P}_i}\left(\mathcal{N}(\mathbf{p})-\operatorname{med}(\mathbf{p})\right)^2 d\mathbf{p},\end{aligned} \tag{5}$$

where the last equality is due to the fact that $\mathcal{P}_i$'s are disjoint.

For ease of exposition, w.l.o.g. we focus on the set $\mathcal{P}_1$. We evaluate the integral inside the sum and using the change of variables $\mathbf{p}=P\mathbf{x}+\mathbf{b}_1$, $d\mathbf{p}=|\det(P)|\,d\mathbf{x}$, we have

$$\int_{\mathcal{P}_1}\left(\mathcal{N}(\mathbf{p})-\operatorname{med}(\mathbf{p})\right)^2 d\mathbf{p}=\int_{[0,1]^d}\left(\mathcal{N}(P\mathbf{x}+\mathbf{b}_1)-\operatorname{med}(P\mathbf{x}+\mathbf{b}_1)\right)^2|\det(P)|d\mathbf{x} \tag{6}$$

Letting $\mathbf{e}_i$ denote the standard unit vector with coordinate $e_i=1$, we get from $P\mathbf{x}=\frac{1}{2d}\mathbf{v}x_1+\sum_{i=2}^d\frac{1}{2}\left(1-\frac{2}{d}\right)x_i\mathbf{e}_i$ and $\mathbf{v}\in\ker(W)$ that we can write $\mathcal{N}(P\mathbf{x}+\mathbf{b}_1)=f(x_2,\ldots,x_d)$ for some function $f:\mathbb{R}^{d-1}\to\mathbb{R}$. Since P is triangular, we have $|\det(P)|=\frac{1}{2^d}\frac{1}{d}\left(1-\frac{2}{d}\right)^{d-1}v_1\geq\frac{1}{2^d}\frac{1}{10d}v_1$ for $d\geq 2$. Moreover, since $\|\mathbf{v}\|_\infty=v_1$ and $\|\mathbf{v}\|_2=1$, we have that $v_1\geq d^{-0.5}$ and we can further lower bound the above to obtain $|\det(P)|\geq\frac{1}{2^d}\frac{1}{10d^{1.5}}$. Plugging in $|\det(P)|$ and Equation (3) in Equation (6), we obtain

$$\begin{aligned}&\int_{[0,1]^d}\left(\mathcal{N}(P\mathbf{x}+\mathbf{b}_1)-\operatorname{med}(P\mathbf{x}+\mathbf{b}_1)\right)^2|\det(P)|d\mathbf{x}\\ &\geq\frac{1}{2^d}\frac{1}{10d^{1.5}}\int_{[0,1]^d}\left(\mathcal{N}(P\mathbf{x}+\mathbf{b}_1)-\operatorname{med}(P\mathbf{x}+\mathbf{b}_1)\right)^2 d\mathbf{x}\\ &\geq\frac{1}{2^d}\frac{1}{10d^{1.5}}\int_{[0,1]^d}\left(f(x_2,\ldots,x_d)-\left(\frac{1}{2d}v_1x_1+\frac{1}{2}\left(1-\frac{1}{d}\right)\right)\right)^2 d\mathbf{x}\\ &=\frac{1}{2^d}\frac{1}{10d^{1.5}}\int_{x_d}\cdots\int_{x_2}\int_{x_1}\left(f(x_2,\ldots,x_d)-\left(\frac{1}{2d}v_1x_1+\frac{1}{2}\left(1-\frac{1}{d}\right)\right)\right)^2 dx_1dx_2\ldots dx_d.\end{aligned} \tag{7}$$

It is easy to verify that the optimal constant approximation for the linear function $\frac{1}{2d}v_1x_1+\frac{1}{2}\left(1-\frac{2}{d}\right)$ is $\frac{1}{4d}v_1+\frac{1}{2}\left(1-\frac{2}{d}\right)$, in which case the optimal L_2 approximation error is,

$$\int_0^1\left(\frac{1}{4d}v_1+\frac{1}{2}\left(1-\frac{1}{d}\right)-\left(\frac{1}{2d}v_1x_1+\frac{1}{2}\left(1-\frac{1}{d}\right)\right)\right)^2 dx=\frac{v_1^2}{4d^2}\int_0^1\left(\frac{1}{2}-x\right)^2 dx=\frac{v_1^2}{48d^2}.$$

Plugging the above back in Equation (7) and using the fact that $v_1\geq d^{-0.5}$ again, we obtain,

$$\int_{\mathcal{P}_1}\left(\mathcal{N}(\mathbf{p})-\operatorname{med}(\mathbf{p})\right)^2 d\mathbf{p}\geq\frac{1}{2^d}\frac{1}{10d^{1.5}}\int_{x_d}\cdots\int_{x_2}\frac{v_1^2}{48d^2}dx_2\ldots dx_d\geq\frac{1}{2^d}\frac{1}{480d^{4.5}}. \tag{8}$$

It is easy to see that for any choice of the set $\mathcal{P}_i$ defined by $(P,\mathbf{b}_i)$, the lower bound on the approximation error is the same as given by Equation (8). Thus, plugging this error in Equation (5), we get

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^d\right)}\left[\left(\mathcal{N}(\mathbf{x})-\mathrm{med}(\mathbf{x})\right)^2\right] \geq \sum_i \int_{\mathcal{P}_i} \left(\mathcal{N}(\mathbf{p})-\mathrm{med}(\mathbf{p})\right)^2 d\mathbf{p}$$
$$\geq \frac{c}{10\sqrt{d}} 2^{d-1} \cdot \frac{1}{2^d} \frac{1}{480 d^{4.5}} = \frac{c}{9600 d^{5.5}},$$

where the second inequality follows from the lower bound on the total number of choices of i (Equation (4) and Equation (8)), concluding the proof.

■

Appendix F. Technical auxiliary lemmas

F.1. Probabilistic lemmas

Lemma 25 (Stirling's Approximation (Robbins, 1955)) *The following lower and upper bounds for $n!$ apply for all $n \geq 1$,*

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$

Recall that $\mathcal{S}_\delta^{d'}$ is the set of vectors in $\mathbb{R}^{d'}$ whose entries are δ-separated (or 0). We show that randomly chosen vectors will be δ-separated with high probability, for inverse-polynomial δ.

Lemma 26 *For any dimension d', we have for any $\delta > 0$,*

$$\mathop{\mathbb{P}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^{d'}\right)}\left[\mathbf{x} \in \mathcal{S}_\delta^{d'} \text{ and } \forall i,\ x_i \neq 0\right] \geq 1 - 3{d'}^2\delta.$$

Proof Sampling $\mathbf{x} \sim \mathcal{U}([0,1]^{d'})$ is equivalent to independently sampling $x_i \sim \mathcal{U}([0,1]),\ \forall i$. Since we want $\mathbf{x}$ to be entirely non-zero and we want $\mathbf{x} \in \mathcal{S}_\delta^{d'}$, we instead show that $\forall i,\ x_i \in [\delta, 1-\delta]$, and that for all indices $i < j$, we have $|x_i - x_j| \geq \delta$.

We have $\mathbb{P}_{x_i\sim\mathcal{U}([0,1])}\left[x_i \in [0,\delta) \cup (1-\delta, 1]\right] = 2\delta$. And thus, taking a union bound over all i,

$$\mathbb{P}\left[\mathbf{x} \notin [\delta, 1-\delta]^{d'}\right] \leq 2d'\delta. \tag{9}$$

Also, for a given pair of entries $i \neq j$ we have

$$\mathbb{P}\left[|x_i - x_j| < \delta\right] = \int_{[0,1]} \mathbb{P}\left[|x_i - x_j| < \delta \middle| x_i\right] dx_i \leq 2\delta.$$

Hence, taking a union bound over all such pairs of entries, we get

$$\mathbb{P}\left[\exists\ (i,j)\ s.t.\ |x_i - x_j| < \delta\right] \leq {d'}^2\delta \tag{10}$$

Thus, combining the results of Equation 9 and Equation 10, we conclude

$$\mathop{\mathbb{P}}_{\mathbf{x}\sim\mathcal{U}\left([0,1]^{d'}\right)}\left[\mathbf{x} \notin \mathcal{S}_\delta^{d'} \text{ or } \exists i,\ x_i = 0\right] \leq 2d'\delta + {d'}^2\delta \leq 3{d'}^2\delta,$$

as desired.

■

The below lemma states standard Chernoff bounds for the process of choosing a subset without replacement, often referred to as the *hypergeometric distribution*. These bounds are used in the probabilistic analysis of the subsampling process in Algorithm 5.

Lemma 27 *Given a universe U of d' elements, with T a subset of size k. Let S be a random subset of U of size n (chosen without replacement). Then we have the following bounds on the probability that the random variable $r = |S \cap T|$ is far from its expectation $\frac{kn}{d'}$:*

1. *For any $\varepsilon \geq 0$,* $\mathbb{P}\left[\left|r - \frac{kn}{d'}\right| \geq \varepsilon\right] \leq 2\exp\left(-2\frac{\varepsilon^2}{n}\right)$ *(Additive Chernoff)*
2. $\mathbb{P}\left[r \notin \left(\frac{1}{2}\frac{kn}{d'}, 2\frac{kn}{d'}\right)\right] \leq 2\exp\left(-\frac{1}{8}\frac{kn}{d'}\right)$ *(Multiplicative Chernoff for $\delta = \frac{1}{2}, 1$)*
3. *Generalizing the part 2 upper bound, for any $\delta \geq 0$,* $\mathbb{P}\left[r \geq \frac{kn}{d'}(1+\delta)\right] \leq \exp\left(-\frac{\delta^2 kn/d'}{2+\delta}\right)$

Proof The probability that a given entry (of the random permutation) is in S equals $\frac{n}{d'}$. If all the entries were independent, then we are bounding the probability that the sum of k samples from a $\frac{n}{d'}$ biased coin is more than ε from $k\frac{n}{d'}$. Additive and multiplicative Chernoff bounds yield the respective stated bounds *if* entries are independent.

Finally, we point out that a Chernoff bound for the *independent* case implies *identical* Chernoff bound for the "negatively associated" case, which includes sampling without replacement. (See (Wajc, 2017)).

■

While the previous lemma gives concentration bounds for subsampling without replacement, we next use this to give concentration bounds a sort of "inverse" of this process.

Consider the following game. Alice's favorite positive integer is r. Alice finds d' balls arbitrarily arranged at locations $\mathbf{x}$ on the real line; and then a random subset S of n of these balls is colored red. Alice now (arbitrarily) chooses a real interval I such the number of red balls in I is her favorite number, $|S \cap I| = r$. What can we say about the overall number of balls in I, regardless of how Alice chooses I? Intuitively, since $\frac{n}{d'}$ fraction of the balls are red, this number $|\mathbf{x} \cap I|$ should be roughly $r\frac{d'}{n}$, and we show this is true in Lemma 28.

Lemma 28 *Let $\mathbf{x}$ be a set of d' distinct real numbers, and let $[L, R]$ be an interval of integers, and let $n \leq d'$ be a nonnegative integer. Consider a probabilistic process P that uniformly randomly chooses a subset $S \subseteq \mathbf{x}$ of size n, and then arbitrarily selects a real interval $I = [I_\ell, I_r]$—possibly in terms of S—such that $|S \cap I| \in [L, R]$. Then we have*

$$\mathbb{P}_{(S,I)\sim P}\left[|\mathbf{x} \cap I| \in \left(\frac{1}{2}L\frac{d'}{n}, 2R\frac{d'}{n}\right)\right] \geq 1 - 3d'\exp\left(-\frac{L}{6}\right) \tag{11}$$

Proof Index $\mathbf{x}$ in sorted order, from x_1 up to $x_{d'}$. We point out that for any interval I, the set $\mathbf{x} \cap I$ consists of some (possibly empty) contiguous subset of $\mathbf{x}$, namely $\{x_i, \ldots, x_j\}$.

Suppose for the sake of contradiction that Equation 11 is violated.

One way Equation 11 could be violated is if $|x \cap I|$ is $\leq \frac{1}{2}L\frac{d'}{n}$. In this case, the set $\mathbf{x} \cap I$ must be a subset of some contiguous set $\{x_i, \ldots, x_j\}$ of size $k := \lfloor \frac{1}{2}L\frac{d'}{n} \rfloor$. For each fixed pair i, j such that $j - i + 1 = k$ we separately apply part 3 of Lemma 27, letting $T = \{x_i, \ldots, x_j\}$, and letting $1 + \delta := \frac{d'}{n} \cdot \frac{L}{\lfloor \frac{1}{2}L\frac{d'}{n} \rfloor}$, where we point out that $\delta \geq 1$. This leads to a probability bound of

$$\exp\left(-\frac{\delta^2 \lfloor \frac{1}{2}L\frac{d'}{n} \rfloor n/d'}{2+\delta}\right) = \exp\left(-L\frac{\delta^2}{(2+\delta)(1+\delta)}\right) \leq \exp\left(-\frac{L}{6}\right)$$

This bound applies for each choice of indices i, j with $j - i + 1 = \lfloor \frac{1}{2}L\frac{d'}{n} \rfloor$, so we thus take a union bound over the at most d' such choices of indices, to cover all cases.

On the other side, the other way Equation 11 could be violated is if $|\mathbf{x} \cap I|$ is $\geq 2R\frac{d'}{n}$. In this case, the set $\mathbf{x} \cap I$ must contain some contiguous set $\{x_i, \ldots, x_j\}$ of size $j - i + 1 = \lceil 2R\frac{d'}{n} \rceil$, which we denote as k when we invoke Lemma 27. For each pair i, j satisfying this condition, we consider the lower bound side of part 2 of Lemma 27: we bound the probability that a random subset $S \subset x$ of size n has intersection with $\{x_i, \ldots, x_j\}$ of size $\leq \frac{1}{2}\frac{kn}{d'}$ (which, since $\frac{1}{2}\frac{kn}{d'} \geq R$ also bounds the probability that this intersection has size $\leq R$) by $2\exp(-\frac{1}{8}\frac{kn}{d'})$, which is thus $\leq 2\exp(-\frac{R}{4})$. Since $R \geq L$, this probability is trivially at most $2\exp(-\frac{L}{6})$. This bound applies for each choice of indices i, j with $j - i + 1 = \lceil 2R\frac{d'}{n} \rceil$, so we thus take a union bound over the at most d' such choices of indices, to cover all cases.

■

F.2. Hash function construction

In this subsection, we explain how to adapt standard hash function techniques to construct a hash function family that will enable a collision-free hash of any (d', ε) sparse input vector. Given a hash function $h : [d'] \to [p]$, we apply it to a sparse vector $\mathbf{x} \in \mathbb{R}^{d'}$ to map it to a smaller dimensional vector $\mathbf{y} \in \mathbb{R}^p$ by applying h to each input coordinate, and summing the results that map to the same coordinate: the j^{th} coordinate of the output will equal $y_j = \sum_{i:h(i)=j} x_i$. Thus if the locations of non-zero entries of $\mathbf{x}$ get hashed to distinct locations by h, then these entries will be preserved in our smaller-dimensional vector $\mathbf{y}$.

The size of the hash family $\mathcal{H}$ and its output dimension p are a function of d', ε and a tunable parameter γ that decides the regime of d' for which the aforementioned success holds. We start with a few preliminaries before proceeding with the actual construction and its properties.

Definition 29 *We call a set $\mathcal{H}$ of hash functions mapping from $U \to M$ a* δ-nearly universal hash function family *if for all $x \neq y \in U$,*

$$\mathop{\mathbb{P}}_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \delta.$$

Definition 30 (Vector representation of indices) *Given positive integers d', n, p' such that $n \geq \log_p d'$, we can represent any number $i \in [d']$ as an n-digit number in base p, considered as a vector with n entries. We denote the base-p representation of i as $\text{base}_{[p]}(i)$, and, using standard indexing notation, the j^{th} digit of this base-p representation is $\text{base}_{[p]}(i)_j$.*

Next we construct the hash function family which will help us create *small* hash function families with the desired properties as stated at the start of the section.

Definition 31 (Vectorized Hashing) *Given any $d' \in \mathbb{Z}^{>0}$ and $\varepsilon' \in (0,1)$, we let p be the smallest prime number greater than $d'^{\varepsilon'}$. We define a hashing scheme mapping indices $[d'] \to [p]$ using the base p vector representation $\text{base}_p(i)$ of indices $i \in [d']$ using $n = \lceil \frac{1}{\varepsilon'} \rceil$ digits (Definition 30). We define the hash function h_a for each $a \in [p]$ as,*

$$h_a(i) = a + \sum_{j=1}^{n} a^j \cdot \text{base}_p(i)_j \mod p.$$

Define $\mathcal{H}_{d',\varepsilon'}$ to consist of all such h_a, namely, $\mathcal{H}_{d',\varepsilon'} \coloneqq \{h_a : a \in [p]\}$.

In the following lemma we prove that the family of hash functions constructed in Definition 31 is a δ-nearly universal hash function family (Definition 29) for some δ depending on the input and the output dimensions of the hash functions.

Lemma 32 *For any $d' \in \mathbb{Z}$ and $\varepsilon' > 0$, the hash function family $\mathcal{H}_{d',\varepsilon'} : [d'] \to [p]$ defined in Definition 31 is a δ-*nearly universal hash function family *(Definition 29) with $\delta = \frac{\lceil \frac{1}{\varepsilon'} \rceil}{d'^{\varepsilon'}}$ where p is defined to be the the smallest prime number larger than $d'^{\varepsilon'}$.*

Proof Picking $h_a \in \mathcal{H}_{d',\varepsilon'}$ uniformly randomly, we can write the probability of collision between any $i, i' \in [d']$ with $i \neq i'$ as,

$$\begin{aligned}\mathop{\mathbb{P}}_{a \sim \mathcal{U}([p])} \left[h_a(i) = h_a(i')\right] &= \mathop{\mathbb{P}}_{a \sim \mathcal{U}([p])} \left[a + \sum_{j=1}^{n} a^j \text{base}_p(i)_j \mod p = a + \sum_{j=1}^{n} a^j \text{base}_p(i')_j \mod p\right] \\ &= \mathop{\mathbb{P}}_{a \sim \mathcal{U}([p])} \left[\sum_{j=1}^{n} a^j \left(\text{base}_p(i)_j - \text{base}_p(i')_j\right) \mod p = 0\right]. \end{aligned} \tag{12}$$

By our underlying construction (Definition 31) and by our choice of p as the smallest prime number greater than $d'^{\varepsilon'}$ and $n = \lceil \frac{1}{\varepsilon'} \rceil$, we have $p^n \geq d'^{\varepsilon'}$, implying that n-digit base-p representation is unique for each $i \in [d']$; in other words $i \neq i' \implies \text{base}_{p,n}(i) \neq \text{base}_{p,n}(i')$. Thus, the expression $\sum_{j=1}^{n} a^j \left(\text{base}_p(i)_j - \text{base}_p(i')_j\right) \mod p$ is a non-zero polynomial mod p (in the variable a) of degree at most n, and hence has at most $n = \lceil \frac{1}{\varepsilon'} \rceil$ roots. Namely, at most n out of the p hash functions in $\mathcal{H}_{d',\varepsilon'}$ make i collide with i', giving us a bound on Equation 12 of

$$\mathop{\mathbb{P}}_{a \sim \mathcal{U}([p])} \left[h_a(i) = h_a(i')\right] \leq \frac{n}{p} \leq \frac{\lceil \frac{1}{\varepsilon'} \rceil}{d'^{\varepsilon'}},$$

where the final inequality is by noting from the definition of p that $p \geq d'^{\varepsilon'}$, concluding the proof. ■

In our construction we will want to hash a (d', ε) sparse vector $\mathbf{x}$ to a lower dimension in a way such that no collisions occur between any of its non-zero entries. Since we cannot induce randomness in a neural network we "derandomize" this by hashing $\mathbf{x}$ with *all* the hash functions h from a small yet well-behaved hash function family $\mathcal{H}$. In the following lemma we show that the family of Definition 31 will in fact allow us to hash (d', ε) sparse vectors without collisions.

Lemma 33 *For any $d' \in \mathbb{Z}^{>0}$ given $\varepsilon, \gamma \in (0,1)$, if $d' \geq \left(\frac{1}{2}\left\lceil \frac{1}{2\varepsilon+\gamma} \right\rceil\right)^{1/\gamma}$ then let p to be the smallest prime number larger than ${d'}^{2\varepsilon+\gamma}$, and consider the hash function family $\mathcal{H}_{d',2\varepsilon+\gamma}$ (Definition 31), where each $h \in \mathcal{H}_{d',2\varepsilon+\gamma}$ maps the index set $[d']$ to the set $[p]$. Then for any (d', ε) sparse vector $\mathbf{x}'$, there exists $h_\star \in \mathcal{H}_{d',2\varepsilon+\gamma}$ that maps the non-zero entries in $\mathbf{x}'$ to distinct locations in $[p]$, i.e., $h_\star$ hashes $\mathbf{x}' \in \mathbb{R}^d$ to $\mathbf{x}'' \in \mathbb{R}^p$ without any collisions between non-zero entries of $\mathbf{x}'$. Formally,*

$$\forall\ (d', \varepsilon) \text{ sparse } \mathbf{x}', \quad \exists h_\star \in \mathcal{H}_{d',2\varepsilon+\gamma} \ \ s.t \ \ \forall\ i \neq j,\ s.t.\ x'_i \neq 0 \neq x'_j \text{ we have } h_\star(i) \neq h_\star(j).$$

Proof At a high level, the proof relies on the fact $\mathcal{H}_{d',2\varepsilon+\gamma}$ is a δ-nearly universal hash function family with $\delta = \frac{\left\lceil \frac{1}{2\varepsilon+\gamma} \right\rceil}{{d'}^{2\varepsilon+\gamma}}$ (Lemma 32), which can be used to show that, for any (d, ε) sparse vector in $\mathcal{H}_{d',2\varepsilon+\gamma}$, the probability that no collision-free $h_\star$ exists is < 1. We use the shorthand $\mathcal{H}$ as a substitute of $\mathcal{H}_{d',2\varepsilon+\gamma}$ for convenience.

Formally, given any (d', ε) sparse vector $\mathbf{x}$, if we uniformly randomly sample a hash function h from $\mathcal{H}$ then the probability that any two non-zero entries get hashed to the same location) is bounded as

$$\begin{aligned}\mathop{\mathbb{P}}_{h\sim\mathcal{U}(\mathcal{H})}[\text{Failure}] = \mathop{\mathbb{P}}_{h\sim\mathcal{U}(\mathcal{H})}[\exists x_i \neq 0 \neq x_j; h(i) = h(j)] &\leq \sum_{\substack{i,j \\ i\neq j \\ x_i\neq 0\neq x_j}} \mathop{\mathbb{P}}_{h\sim\mathcal{U}(\mathcal{H})}[h(i) = h(j)] \\ &\leq \binom{{d'}^{\varepsilon}}{2}\delta < \frac{{d'}^{2\varepsilon}}{2}\frac{\left\lceil \frac{1}{2\varepsilon+\gamma} \right\rceil}{{d'}^{2\varepsilon+\gamma}} = \left\lceil \frac{1}{2\varepsilon+\gamma} \right\rceil \frac{1}{2{d'}^{\gamma}} \leq 1,\end{aligned}$$

where the second inequality is from the fact that $\mathcal{H}$ is δ-almost universal, taking a union bound over all $\leq \binom{{d'}^{\varepsilon}}{2}$ pairs of $\{i, j\}$ with non-zero entries, which is strictly less than $\frac{{d'}^{2\varepsilon}}{2}$, as used in the third inequality; and the last inequality is from the assumption $d' \geq \left(\frac{1}{2}\left\lceil \frac{1}{2\varepsilon+\gamma} \right\rceil\right)^{1/\gamma}$. Since we have bounded the probability over $h \in \mathcal{H}$ of *any* collision as strictly less than 1, we conclude that there exists $h_\star \in \mathcal{H}$ which has *no collisions* on the non-zero entries of the given $\mathbf{x}'$, as desired. ■

Appendix G. Neural network constructions and properties

Recall Definition 15. In this section we describe all the components we assemble to produce the neural networks of our main results. We explicitly show how to implement various natural primitives with shallow neural networks, along with a few more technical primitives needed for the specifics of our algorithms. The particular "circuitry" required by these neural networks is sometimes slightly intricate, but overall none of these constructions should be surprising, and this section can be skipped on a first read.

We state each neural network in a definition, and describe its properties immediately afterwards in a lemma.

Sometimes our neural networks may make use of data from previous layers, which could be thought of as using a "skip connection" where the result of a previous layer is sent along a wire that "skips" some intermediate layer(s). However, "skip connections" may lead to subtleties in the definition of width and therefore we do *not* use them in this paper. Instead, we can emulate a skip connection by the *identity transform*, where an input x is preserved across a ReLU layer via adding 2 neurons to the width, using the identity $x = [x]_+ - [-x]_+$.

G.1. Maximum neural network

Definition 34 *We define our* maximum *neural network* $\mathcal{N}^{MAX} : \mathbb{R}^2 \to \mathbb{R}$ *as*

$$\mathcal{N}^{MAX}(x_1, x_2) = [x_2]_+ - [-x_2]_+ + [x_1 - x_2]_+ .$$

G.2. Comparison neural network

We next define a neural network that tests whether $x_1 > x_2$; this network is parameterized by a tolerance δ, where the network will return the correct 0 or 1 answer *unless* x_1, x_2 are within δ of each other.

Definition 35 *We define our* comparison *neural network* $\mathcal{N}^C_\delta : \mathbb{R}^2 \to \mathbb{R}$ *as,*

$$\mathcal{N}^C_\delta(x_1, x_2) = \left[\frac{1}{\delta}(x_1 - x_2)\right]_+ - \left[\frac{1}{\delta}(x_1 - x_2 - \delta)\right]_+$$

Fact 36 *Given* $\delta > 0$*, for all inputs* $x_1, x_2 \in \mathbb{R}$ *we have*

$$\mathcal{N}^C_\delta(x_1, x_2) = \begin{cases} 1, & \text{If } x_1 > x_2 \text{ and } |x_1 - x_2| \geq \delta, \\ 0, & \text{If } x_1 \leq x_2, \\ \frac{1}{\delta}(x_1 - x_2) & \text{otherwise .} \end{cases}$$

Additionally, this neural network has 1 hidden layer, employs 2 hidden neurons, and the magnitude of weights is upper bounded by $\frac{1}{\delta}$*. Lastly, the output is always in the interval* $[0, 1]$.

G.3. Non-zero counter neural network

One important primitive in our algorithm is counting the number of non-zero entries in a non-negative vector, which we implement by repeated application of the comparison neural network $\mathcal{N}^C_\delta$ of Definition 35. Parameterized by a tolerance $\delta > 0$, we define the *non-zero counter* neural network $\mathcal{N}^{NZC}_\delta$ to output the number of non-zero entries in a vector $\mathbf{x} \in \mathbb{R}^{d'}$, provided that $\forall i, x_i \in \{0\} \cup [\delta, \infty)$.

Definition 37 *We define our* non-zero counter *neural network* $\mathcal{N}^{NZC}_\delta : \mathbb{R}^{d'} \to \mathbb{R}$ *as,*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{d'} : \quad \mathcal{N}^{NZC}_\delta(\mathbf{x}) = \sum_{i=1}^{d'} \mathcal{N}^C_\delta(x_i, 0) .$$

Lemma 38 *The* non-zero counter *neural network* $\mathcal{N}_\delta^{NZC}$ *(Definition 37) when given an input* $\mathbf{x} \in \mathbb{R}^{d'}$ *such that* $\forall i, x_i \in \{0\} \cup [\delta, \infty)$ *outputs the number of non-zero entries:*

$$\mathcal{N}_\delta^{NZC}(\mathbf{x}) = \sum_{i=1}^{d'} \mathbb{1}\{x_i > 0\}.$$

The non-zero counter *neural network requires* 1 *hidden layer, a width of* $2d'$*, and the magnitudes of the weights are bounded by* $\frac{1}{\delta}$.

Proof The output is evident from the use of *comparison* neural network (Definition 35, $\mathcal{N}_\delta^C$) and the assumption $\forall i, x_i \in \{0\} \cup [\delta, \infty)$. It is clear from Definition 37 and Definition 35 that $\mathcal{N}_\delta^{NZC}$ only requires 1 hidden layer and each $i \in [d']$ uses 2 neurons. Also from the use of *comparison* neural network (Definition 35, $\mathcal{N}_\delta^C$) we get that the magnitude of weights required is $\frac{1}{\delta}$. ■

G.4. Masking neural network

We will often want to create a binary "mask", recording which entries in a vector $\mathbf{x} \in \mathbb{R}^{d'}$ lie in a given interval $[\ell, u]$. We make use of $2d'$ copies of our *comparison* neural network to construct this *masking* neural network, which will be correct as long as each input x_i is *either* inside the interval $[\ell, u]$ or δ-far from the interval $[\ell, u]$.

Definition 39 *We define our* masking *neural network* $\mathcal{N}_\delta^M : \mathbb{R}^{d'} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{d'}$ *to have* i*'th output defined as,*

$$\textit{Given } \mathbf{x} \in \mathbb{R}^{d'}, u > \ell, i \in [d']: \quad \mathcal{N}_\delta^M(\mathbf{x}, u, \ell)[i] \coloneqq \mathcal{N}_\delta^C(x_i, \ell - \delta) - \mathcal{N}_\delta^C(x_i, u)$$

Lemma 40 *The* masking *neural network* $\mathcal{N}_\delta^M$ *(Definition 39) when given an input* $\mathbf{x} \in \mathbb{R}^{d'}$ *such that* $\forall i \in [d']$ *we have* $x_i \notin (\ell - \delta, \ell) \cup (u, u + \delta)$*, will output a vector* $\mathbf{x}' \in \mathbb{R}^{d'}$ *where,*

$$x_i' = \mathbb{1}\{\ell \le x_i \le u\}.$$

The masking *neural network requires* 1 *hidden layer, a width of* $4d'$*, and the magnitudes of the weights are bounded by* $\frac{1}{\delta}$.

Proof This network consists of $2d'$ copies of the *comparison* neural network $\mathcal{N}_\delta^C$ of Definition 35, each copy of which has 1 hidden layer, width 2, and magnitude of weights bounded by $\frac{1}{\delta}$ by Fact 36, which yields our depth, width, and weight bounds. Correctness for each i follows from the correctness of $\mathcal{N}_\delta^C$, described in Fact 36. ■

G.5. Filtering neural network

We next define a neural network that is intuitively similar to the *masking* network, except for $x_i \in [\ell, u]$ we will return x_i itself instead of 1 (and 0 if x_i is δ-far from the interval). We call this a *filtering* neural network—it filters out those x_i not in the interval $[\ell, u]$ and leaves $x_i \in [\ell, u]$ unchanged.

While the *filtering* neural network is analogous to the *masking* neural network, we will implement it different, explicitly defining a piecewise-linear function of x_i with 4 breakpoints. The input requirements for this network will be analogous to those for the *masking* network except with the additional requirement that all inputs $x_i \in [0, 1]$.

Definition 41 *We define our* filtering *neural network* $\mathcal{N}_\delta^F : \mathbb{R}^{d'} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{d'}$ *to have* i*'th output defined as,*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{d'}, u > \ell, i \in [d'] :$$
$$\mathcal{N}_\delta^F(\mathbf{x}, u, \ell)[i] \coloneqq \left[\frac{1}{\delta}(x_i - \ell) + x_i\right]_+ - \left[\frac{1}{\delta}(x_i - \ell)\right]_+ - \left[\frac{1}{\delta}(x_i - u)\right]_+ + \left[\frac{1}{\delta}(x_i - u) - x_i\right]_+.$$

Lemma 42 *The* filtering *neural networks* $\mathcal{N}_\delta^F$ *(Definition 41) for* $\delta \in (0, 1]$*, when given an input* $\mathbf{x} \in [0, 1]^{d'}$ *and numbers* $0 \le \ell < u \le 1$ *such that* $\forall i \in [d']$ *we have* $x_i \notin (\ell - \delta, \ell) \cup (u, u + \delta)$*, will output a vector* $\mathbf{x}' \in \mathbb{R}^{d'}$ *where,*

$$x_i' = x_i \cdot \mathbb{1}\{\ell \le x_i \le u\}.$$

The filtering *neural network requires* 1 *hidden layer, a width of* $4d'$*, and the magnitudes of the weights are bounded by* $\frac{1}{\delta} + 1$.

Proof To show correctness, we point out that $\mathcal{N}_\delta^F(\mathbf{x}, u, \ell)[i]$ is defined to be a piecewise-linear function of x_i with 4 breakpoints at $\frac{\ell}{1+\delta}, \ell, u, \frac{u}{1-\delta}$. To the left of the first breakpoint, the function equals 0; between the first two breakpoints the function has slope $\frac{1}{\delta} + 1$ so attains value ℓ at the second breakpoint, $x_i = \ell$; when $x_i \in [\ell, u]$ the function has slope 1 and thus will equal x_i; to the right of the third breakpoint u, the function will have slope $\frac{1}{\delta} - 1$, and will thus be 0 for x_i to the right of the last breakpoint. Our neural network will thus be correct for $x_i \in [\ell, u]$, and also for x_i beyond the two outermost breakpoints.

To show correctness for $x \le \ell - \delta$, we have from above that the function is correct to the left of the first breakpoint, namely, for $x \le \frac{\ell}{1+\delta}$, and we conclude from the fact that $\ell - \delta \le \frac{\ell}{1+\delta}$, which we verify by multiplying through by $1 + \delta$, moving all terms to the left, and dividing by δ to get the equivalent easily checked inequality that $\ell - 1 - \delta \le 0$, since $\ell < 1$ and $\delta > 0$.

Finally, to show correctness for $x \ge u + \delta$ though under the guarantee that $x \le 1$, we have from above that the function is correct to the right of the last breakpoint, namely, for $x \ge \frac{u}{1-\delta}$. Thus the only cases where the function could be wrong are those for which $u + \delta < \frac{u}{1-\delta}$; solving for u shows that this can only be true when $u > 1 - \delta$; but since we only need to show correctness for $x \ge u + \delta$, this means we only need to show correctness when $x > 1$, which is trivially true since these values of x are out of range of our assumptions, thus showing the lemma.

This network uses 1 hidden layer, a width of $4d'$, and the magnitudes of the weights are bounded by $\frac{1}{\delta} + 1$. ■

G.6. Indicator function product neural network

We introduce a simple neural network here that, given an input $x \in [0,1]$ and $s \in \mathbb{Z}$ will return $x \cdot \mathbb{1}\{s = 0\}$.

Definition 43 *We define our* indicator function product *neural network* $\mathcal{N}^{IFP} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ *as,*

$$\textit{Given } x, s \in \mathbb{R}, \quad \mathcal{N}^{IFP}(x, s) = [x + s]_+ - [x + s - 1]_+ - [s]_+ + [s - 1]_+.$$

Lemma 44 *The* indicator function product *neural network* $\mathcal{N}^{IFP}$ *(Definition 43), when given input* $x \in [0,1]$ *and integer* $s \in \mathbb{Z}$ *outputs*

$$\mathcal{N}^{IFP}(x, s) = x \cdot \mathbb{1}\{s = 0\}.$$

The indicator function product *neural network* $\mathcal{N}^{IFP}$ *has 1 hidden layer, width 4, and magnitude of weights 1.*

Further, for any $x, s \in \mathbb{R}$*, the output of the neural network has magnitude at most* $|x|$*.*

Proof For convenience, we define the function $\text{trim}_{[0,1]}(y)$ to "trim" the real number y to the range $[0,1]$, defined equivalently as $\text{trim}_{[0,1]}(y) := \max(0, \min(1, y))$. It is straightforward to check that trimming can be implemented as the difference of two ReLU units: for a real number y, we have $[y]_+ - [y-1]_+ = \text{trim}_{[0,1]}(y)$. We use this relation twice in the equation defining $\mathcal{N}^{IFP}(x, s)$ to see that $\mathcal{N}^{IFP}(x, s)$ equals

$$\text{trim}_{[0,1]}(x + s) - \text{trim}_{[0,1]}(s) \tag{13}$$

Recall that we assume $x \in [0,1]$. When $s = 0$, Equation 13 equals $\text{trim}_{[0,1]}(x) = x$; when $s < 0$, since $s \in \mathbb{Z}$, we have $s \leq -1$, and thus both terms of Equation 13 get trimmed to 0; and analogously, when $s > 0$, both terms of Equation 13 get trimmed to 1, and thus yield a difference of 0. Thus in all cases, Equation 13 equals $x \cdot \mathbb{1}\{s = 0\}$, yielding correctness.

This neural network clearly has 1 hidden layer, 4 neurons, and weights of magnitude 1.

Finally, to bound the magnitude for arbitrary $x, s \in \mathbb{R}$, we point out that Equation 13 equals 0 when $x = 0$, and, when considered as a function of x, has Lipschitz constant 1 since the $\text{trim}()$ function has Lipschitz constant 1. Thus $\mathcal{N}^{IFP}(x, s)$ has magnitude at most $|x|$, as claimed. ■

G.7. Rank selection neural network

We now define the *rank selection* neural network, which implicitly sorts its input using an all-pairs quadratic-width approach, and returns elements of the desired ranks. This neural network is the central component of the depth 3, quadratic width median finding result in Theorem 2. While for the subsequent sub-quadratic median finding neural networks we cannot use this network directly on the entire input, this network is a crucial component once we have used other techniques to reduce the size of the input.

The idea behind this neural network is to compare all pairs of elements and use the number of comparisons "won" by each entry to determine if that entry has the designated rank, and should hence be returned.

Definition 45 *We define our* rank selection *neural network* $\mathcal{N}_\delta^{RS} : \mathbb{R}^{d'} \times \mathbb{R}^p \to \mathbb{R}^p$ *as,*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{d'}, \mathbf{r} \in \mathbb{R}^p, \quad \text{and letting,} \quad y_k = 1 + \sum_{j \in [d']} \mathcal{N}_\delta^C(x_k, x_j), \quad k \in [d'],$$

$$\mathcal{N}_\delta^{RS}(\mathbf{x}, \mathbf{r})[i] = \sum_{k \in [d']} \mathcal{N}^{IFP}(x_k, r_i - y_k),$$

where $\mathcal{N}_\delta^{RS}(\mathbf{x}, \mathbf{r})[i]$ *is the* i'*th coordinate of the output.*

Lemma 46 *The* rank selection *neural network* $\mathcal{N}_\delta^{RS}$ *(Definition 45), when given an input* $\mathbf{x} \in [0,1]^{d'}$ *where for all* $j \neq k$ *we have either* $|x_j - x_k| \geq \delta$ *or* $x_j = x_k = 0$*, and given a vector of ranks* $\mathbf{r} \in \{1, \ldots, d'\}^p$*, outputs a vector* $\mathbf{x}' \in \mathbb{R}^p$ *with*

$$x'_i = \mathcal{R}_{r_i}(\mathbf{x}), \quad \forall i \in [p].$$

Further, given arbitrary input $\mathbf{x} \in \mathbb{R}^{d'}$ *we always have the bound*

$$|x'_i| \leq d' \|\mathbf{x}\|_\infty.$$

Moreover, the rank selection *neural network* $\mathcal{N}_\delta^{RS}$ *requires* 2 *hidden layers, a width of at most* $\max(2{d'}^2 + 2d' + 2p, 4pd')$ *and the magnitudes of weights are bounded by* $\frac{1}{\delta}$.

Proof We first show the correctness property.

The neural network first computes y_k, comparing x_k with every x_j and returning 1 plus the number of strictly smaller elements, yielding that y_k will be the rank of x_k in $\mathbf{x}$. We implement this with the *comparison* neural network $\mathcal{N}_\delta^C$, which will return correct answers by Fact 36 because all pairs x_j, x_k are either identical or δ-separated. (We point out that our input $\mathbf{x}$ may have repeated zeros, and for all such elements we will compute $y_k = 1$ because there are no strictly smaller elements.)

The i^{th} element of the return vector is computed in the next line via a d'-way sum with respect to k over $\mathcal{N}^{IFP}(x_k, r_i - y_k)$, which we analyze with Lemma 44, since $x_k \in [0,1]$ and $y_k \in \mathbb{Z}$. By Lemma 44 the neural network thus returns, as its i^{th} entry, $\sum_{k \in [d']} x_k \cdot \mathbb{1}\{y_k = r_i\}$. Namely, given as input a desired rank r_i, we return the sum of all entries x_k whose rank (previously stored as y_k) equals r_i. Since all entries except 0 are unique, the neural network will correctly return the element of rank r_i.

Next, we bound the return value of the neural network for arbitrary $\mathbf{x} \in \mathbb{R}^{d'}$: by Lemma 44, the expression $\mathcal{N}^{IFP}(x_k, r_i - y_k)$ has magnitude at most $|x_k|$. Summing this over $k \in [d']$ immediately gives our desired universal bound $|x'_i| \leq d' \|\mathbf{x}\|_\infty$.

For the depth of the network, we use two hidden layers, one corresponding to $\mathcal{N}_\delta^C$ and the other for $\mathcal{N}^{IFP}$. For the width in the first hidden layer, we compute $\mathcal{N}_\delta^C(x_k, x_j)$ for all pairs $j, k \in [d']$ and since $\mathcal{N}_\delta^C(x_i, x_j)$ requires 2 neurons (Fact 36) this contributes width $2{d'}^2$ in the first layer. In the second layer we use inputs $\mathbf{x}, \mathbf{r}$, and thus need to add 2 extra neurons in the first layer to compute the identity function for each value that we want to reuse later, contributing the remaining $2d' + 2p$ to the width of the first layer. In the second hidden layer we use we use $p \cdot d'$ copies of $\mathcal{N}^{IFP}$, thus requiring $4pd'$ neurons (by Lemma 44); taking the max of the neurons in the two layers gives us the stated width bound. The upper bound on the magnitude of the weights follows from the *comparison* neural network (Definition 35, $\mathcal{N}_\delta^C$) that requires weights of magnitude $\frac{1}{\delta}$ (Fact 36), since $\mathcal{N}^{IFP}$ only has weights of magnitude 1 by Lemma 44. ■

G.8. Non-zero element shortlisting neural network

We next define another key primitive that we exclusively use in our linear width construction. The *non-zero element shortlisting* neural network, parameterized by a return size p, when given a vector $\mathbf{x}$ and an interval $[\ell, u]$ will try to return p non-zero entries of $\mathbf{x}$ that lie in the interval $[\ell, u]$. This neural network uses the *indicator function product* neural network (Definition 43) in a related manner to the previous construction of the *rank selection* neural network, even though the end result is rather different.

Definition 47 *Given $\mathbf{x} \in \mathbb{R}^{d'}$ and $p \in \mathbb{Z}$ we define our* non-zero element shortlisting *neural network $\mathcal{N}^{NZES}_{\delta,p} : \mathbb{R}^{d'} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^p$ as,*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{d'}, \ell < u, \text{ let } \quad \ell' = \mathcal{N}^{MAX}(\ell, \delta/2) \text{ (Definition 34)},$$
$$\mathbf{y} = \mathcal{N}^{M}_{\delta/2}(\mathbf{x}, u, \ell') \text{ (Definition 39)}, \ \mathbf{f} = \mathcal{N}^{F}_{\delta}(\mathbf{x}, u, \ell) \text{ (Definition 41)}$$
$$\mathcal{N}^{NZES}_{\delta,p}(\mathbf{x}, u, \ell)[i] = \sum_{j \in [d']} \mathcal{N}^{IFP}\left(f_j, i - \sum_{j'=1}^{j} y_{j'}\right), i \in [p] \text{ (Definition 43)}.$$

where $\mathcal{N}^{NZES}_{\delta,p}(\mathbf{x}, u, \ell)[i]$ is the i'th coordinate of the output.

Lemma 48 *The* non-zero element shortlisting *neural network $n^{NZES}_{\delta,p}$ (Definition 47) takes as input $\mathbf{x} \in \mathcal{S}^{d'}_{\delta}$, and numbers $\ell < u$ such that $\ell, u \in \mathbf{x} \cup \{0, 1\}$. Letting p' denote the number of non-zero entries of $\mathbf{x}$ lying in the interval $[\ell, u]$, the neural network returns $\mathbf{x}' \in \mathbb{R}^p$ with*

$$x'_i = \begin{cases} x_j & \text{(such that } x_j \text{ is the } i'\text{th non-zero entry that lies in the interval } [\ell, u]), \quad i \in [\min(p, p')] \\ 0 & \text{if } i > \min(p, p'). \end{cases}$$

Moreover the non-zero element shortlisting *neural network requires 3 hidden layers, a width of $\mathcal{O}(pd')$ and the magnitudes of weights are bounded by $\frac{2}{\delta}$.*

Proof The proof is similar to the proof of Lemma 46. We first show the correctness property.

By definition, we have $\ell' = \max(\ell, \delta/2)$. The *masking* neural network $\mathcal{N}^{M}_{\delta/2}(\mathbf{x}, u, \ell')$ will return a $\{0, 1\}$ vector $\mathbf{y}$ recording for each $i \in [d']$ whether x_i is a non-zero value in $[\ell, u]$: by Lemma 40, the output will be correct as long as the boundaries ℓ, u are either equal to or $\delta/2$-separated from each x_i; this will be satisfied since the non-zero values of $\mathbf{x}$ lie in the range $[\delta, 1-\delta]$ by definition of $\mathcal{S}^{d'}_{\delta}$, and ℓ' is rounded up to $\delta/2$ by the max function in the case $\ell = 0$. The *filtering* neural network $\mathcal{N}^{F}_{\delta}(\mathbf{x}, u, \ell)$ will correctly filter its input, returning $f_i = x_i \mathbb{1}\{x_i \in [\ell, u]\}$ by Lemma 42, since the boundaries ℓ, u are either equal to or δ-separated from each x_i.

Since $\mathbf{f}$ contains entries in $[0, 1]$ and $\mathbf{y}$ contains integer entries, we apply Lemma 44 to conclude that

$$\mathcal{N}^{NZES}_{\delta,p}(\mathbf{x}, u, \ell)[i] = \sum_{j \in [d']} f_j \cdot \mathbb{1}\{i = \sum_{j'=1}^{j} y_{j'}\}$$

Namely, defining for the purposes of analysis a vector $\mathbf{z}$ whose j^{th} entry equals $\sum_{j'=1}^{j} y_{j'}$, we have that z_j counts how many of the first j entries of $\mathbf{x}$ are non-zero entries in the range $[\ell, u]$; the

indicator function will evaluate whether this matches i and return f_j in this case. Since (as shown above) $f_j = x_j \mathbb{1}\{x_j \in [\ell, u]\}$, we conclude that $\mathcal{N}_{\delta,p}^{NZES}(\mathbf{x}, u, \ell)[i]$ will return the i^{th} non-zero entry of $\mathbf{x}$ lying in the range $[\ell, u]$, if such an entry exists, concluding the correctness proof.

The bounds on the width, depth, and weights result from corresponding bounds on the components $\mathcal{N}^{MAX}(\ell, \delta/2)$, $\mathcal{N}_{\delta/2}^{M}(\mathbf{x}, u, \ell')$, $\mathcal{N}_{\delta}^{F}(\mathbf{x}, u, \ell)$, and $\mathcal{N}^{IFP}(f_j, i - \sum_{j'=1}^{j} y_{j'})$ from Lemmas 40, 42, 44 respectively.

■

G.9. Rank computing neural network

Recall that the idea of Algorithm 5 is to repeatedly "filter" $\mathbf{x}$ by zeroing out elements not lying in an interval $[\ell, u]$, in such a way that $\text{med}(\mathbf{x})$ is preserved, while returning a much sparser vector $\mathbf{y}$. A crucial component of this is determining the rank r of $\text{med}(\mathbf{x})$ in (the non-zero portion of) the new vector $\mathbf{y}$, which we do in the *rank computing* neural network. We compute this rank r taking an element $e \in \mathbf{y}^{\neq 0}$ and noticing that, since $\mathbf{y}^{\neq 0}$ is a contiguous portion of $\mathbf{x}$, the difference in ranks of $e, \text{med}(\mathbf{x})$ in $\mathbf{x}$ equals the difference of their ranks in $\mathbf{y}^{\neq 0}$, and since we know the rank of $\text{med}(\mathbf{x})$ in $\mathbf{x}$ equals $d'/2$, we can recover r by computing the rank of e in both vectors, and solving for r.

Definition 49 *Given $\mathbf{x} \in \mathbb{R}^{d'}, \mathbf{y} \in \mathbb{R}^{d''}$, and an element $e \in \mathbb{R}$ we define our* rank-computing *neural network $\mathcal{N}_{\delta}^{RC} : \mathbb{R}^{d'} \times \mathbb{R}^{d''} \times \mathbb{R} \to \mathbb{R}$ as,*

$$\begin{aligned} &\textit{Given } \mathbf{x} \in \mathbb{R}^{d'}, \mathbf{y} \in \mathbb{R}^{d''}, e \in \mathbb{R}, \\ \mathcal{N}_{\delta}^{RC}(\mathbf{x}, \mathbf{y}, e) &= d'/2 - d'' + \mathcal{N}_{\delta}^{NZC}(\mathbf{y}) + \sum_{j \in [d'']} \mathcal{N}_{\delta}^{C}(e, y_j) - \sum_{j \in [d']} \mathcal{N}_{\delta}^{C}(e, x_j). \end{aligned}$$

Lemma 50 *The* rank-computing *neural network $\mathcal{N}_{\delta}^{RC}$ (Definition 49) when given an entirely non-zero input $\mathbf{x} \in \mathcal{S}_{\delta}^{d'}$ and another input $\mathbf{y} \in \mathcal{S}_{\delta}^{d''}$, whose non-zero entries form a contiguous block in a sorted version of $\mathbf{x}$ containing $\text{med}(\mathbf{x})$ along with a third input e that is any non-zero entry of $\mathbf{y}$, the network outputs a rank r such that*

$$\mathcal{R}_r\left(\mathbf{y}^{\neq 0}\right) = \text{med}(\mathbf{x}).$$

Moreover the rank-computing *requires 1 hidden layer, a width of $\mathcal{O}(\max(d', d''))$ and the magnitudes of weights are bounded by $\frac{1}{\delta}$.*

Proof Recalling that input e is any non-zero entry of $\mathbf{y}$, let $r_{e,\mathbf{x}}$ denote its rank among the entries of $\mathbf{x}$ and let $r_{e,\mathbf{y}}$ denote the rank of e among the entries of $\mathbf{y}^{\neq 0}$. Further, let r be the rank of $\text{med}(\mathbf{x})$ among the non-zero entries of $\mathbf{y}$, where the rank of $\text{med}(\mathbf{x})$ in $\mathbf{x}$ equals $d'/2$ by definition of the median. Since both e and $\text{med}(\mathbf{x})$ lie in the contiguous block $\mathbf{y}^{\neq 0}$, when $\mathbf{x}$ is sorted, we have that the difference of ranks of e and $\text{med}(\mathbf{x})$ in $\mathbf{x}$ equals the difference of ranks of these elements in $\mathbf{y}^{\neq 0}$, yielding

$$r_{e,\mathbf{x}} - d'/2 = r_{e,\mathbf{y}} - r. \tag{14}$$

We use this expression to show that our neural network $\mathcal{N}_\delta^{RC}$ correctly computes r. Since $\mathbf{x} \in \mathcal{S}_\delta^{d'}$ and non-zero, all the elements of $\mathbf{x}$ are δ-separated, and thus we compute the rank of e in $\mathbf{x}$ as $r_{e,\mathbf{x}} = 1 + \sum_{j\in[d']} \mathcal{N}_\delta^C(e, x_j)$, by applying Fact 36 to correctly count 1 plus the number of x_j that are smaller than e. Analogously, the rank of e in $\mathbf{y}$ (including the zero entries for the moment) equals $1 + \sum_{j\in[d'']} \mathcal{N}_\delta^C(e, y_j)$; and the number of zero entries is found by taking d'' minus the result from the *non-zero counting* neural network (Definition 37) $\mathcal{N}_\delta^{NZC}(\mathbf{y})$. Subtracting yields that the rank of e in $\mathbf{y}^{\neq 0}$ equals $r_{e,\mathbf{y}} = 1 + \sum_{j\in[d'']} \mathcal{N}_\delta^C(e, y_j) - \left(d'' - \mathcal{N}_\delta^{NZC}(\mathbf{y})\right)$. Solving for r in Equation 14, we find that r is exactly the expression computed by our overall neural network.

Since our neural network applies $\mathcal{N}_\delta^{NZC}$ once, to an input of size d'', and, in parallel applies $\mathcal{N}_\delta^C$ $d' + d''$ times, we have (from Fact 36 and Lemma 38) that this network has 1 hidden layer, a width of $\mathcal{O}(\max(d', d''))$ and the magnitudes of weights bounded by $\frac{1}{\delta}$. ■

G.10. Rank scaling neural network

In our construction, one of the intermediate steps is to compute an analog rank r', of a particular rank r among a small random subset of entries of the non-zero entries. To do this we need to multiply the rank with the size of this subset and divide it by the number of non-zero entries which is non-trivial as the number of non-zero entries in our construction is a random quantity and we cannot pre-compute it. The *rank scaling* neural network helps us do this by simply computing all possible values r' can take and zeroing out all but the one that is the true value. Given $b \in [d']$, a rank r of a coordinate among the non-zero entries of $\mathbf{x} \in \mathcal{S}_\delta^{d'}$ and assuming $|\mathbf{x}^{\neq 0}| \geq 1$, this outputs r' defined as,

$$r' = \frac{rb}{|\mathbf{x}^{\neq 0}|}.$$

Definition 51 *Given* $\mathbf{x} \in \mathbb{R}^{d'}$ *and* $r \in \mathbb{R}$ *we define our* rank scaling *neural network* $\mathcal{N}_{\delta,b}^{RSC} : \mathbb{R}^{d'} \times \mathbb{R} \to \mathbb{R}$ *as,*

$$\begin{aligned} &\textit{Given } \mathbf{x} \in \mathbb{R}^{d'}, r \in \mathbb{R}, \textit{ let} && z = \mathcal{N}_\delta^{NZC}(\mathbf{x}), \\ &r' = \sum_{k=1}^{d'} \frac{d'b}{k} \cdot \mathcal{N}^{IFP}\left(\frac{r}{d'}, z - k\right) && \textit{(Definition 43).} \end{aligned}$$

Lemma 52 *The* rank scaling *neural network* $\mathcal{N}_{\delta,b}^{RSC}$ *(Definition 51) when given* $\mathbf{x} \in \mathcal{S}_\delta^{d'}$ *with* $|\mathbf{x}^{\neq 0}| \geq 1$*, and an input* $r \in [0, d']$*, outputs* r' *where*

$$r' = \frac{rb}{|\mathbf{x}^{\neq 0}|}.$$

Moreover rank scaling *neural network requires* 2 *hidden layers, a width of* $4d'$ *and magnitude of weights bounded by* $\max\left(d'b, \frac{1}{\delta}\right)$.

Proof Since $\mathbf{x} \in \mathcal{S}_\delta^{d'}$ we have by Lemma 38 that $z = \mathcal{N}_\delta^{NZC}(\mathbf{x})$ correctly computes the number of non-zero entries in $\mathbf{x}$, namely $|\mathbf{x}^{\neq 0}|$. In the next line, we apply $\mathcal{N}^{IFP}$, where since $\frac{r}{d'} \in [0, 1]$ and both z, k are integers, we have by Lemma 44 that the output value satisfies $r' = \sum_{k=1}^{d'} \frac{d'b}{k} \frac{r}{d'} \cdot \mathbb{1}\{k =$

$z\}$. Thus the sum has a non-zero contribution only for the term where $k = |\mathbf{x}^{\neq 0}|$, and thus the neural network outputs $r' = \frac{rb}{|\mathbf{x}^{\neq 0}|}$, as desired.

The number of hidden layers required by $\mathcal{N}_\delta^{NZC}$ is 1, from Lemma 38, and 1 more for $\mathcal{N}^{IFP}$ from Lemma 44. The width of the first layer is $2d'$ from Lemma 38, and the second layer has width $4d'$ from Lemma 44. The magnitude of weights required by $\mathcal{N}_\delta^{NZC}$ is $1/\delta$, and $\mathcal{N}^{IFP}$ has weights bounded by 1 by Lemma 44, but we scale the output by $\frac{d'b}{k} \leq d'b$, to give an overall bound of $\max(d'b, 1/\delta)$. ■

G.11. Ceiling neural network

In each step of our construction, we compute ranks scaled by certain quantities to obtain an analogous rank r' of the *median* in a smaller subset. Since ranks can only be integers we convert this to an integer through the *ceiling* ($\lceil . \rceil$) or *floor* ($\lfloor . \rfloor$) operations using the *ceiling* neural network (note that $\lfloor x \rfloor = -\lceil -x \rceil$). We are able to do this by exploiting the finite structure of the set of values r' can take. The *ceiling* neural network when input $\frac{a}{b}$ where $b \in [d']$ and $a \in \mathbb{Z}$ such that $-d' \leq \frac{a}{b} \leq d'$ computes $\lceil \frac{a}{b} \rceil$.

Definition 53 *We define our* ceiling *neural network* $\mathcal{N}_{d'}^{CEI} : \mathbb{R} \to \mathbb{R}$ *as,*

Given $x \in \mathbb{R}$,

$$\mathcal{N}_{d'}^{CEI}(x) = -d' + \sum_{i \in \{-d', \ldots, d'\}} d' \left([x - i]_+ - \left[x - i - \frac{1}{d'} \right]_+ \right).$$

Lemma 54 *The* ceiling *neural network* $\mathcal{N}_{d'}^{CEI}$ *(Definition 53), when given input* $x \in [-d', d']$ *that is also rational number* $x = \frac{a}{b}$ *with denominator* $b \in \{1, \ldots, d'\}$ *will output the ceiling of* x*,*

$$\mathcal{N}_{d'}^{CEI}(x) = \lceil x \rceil .$$

Moreover, the ceiling *neural network requires* 1 *hidden layer, a width of at most* $4(d'+1)$ *and magnitude of weights* d'*.*

Proof We analyze the expression inside the sum for rational x. In the case $x \leq i$, then we have $d' \left([x - i]_+ - \left[x - i - \frac{1}{d'} \right]_+ \right) = 0$. On the other hand, if $x > i$, for $i \in \mathbb{Z}$ and x a rational number with denominator at most d', then $x > i + \frac{1}{d'}$, yielding $d' \left([x - i]_+ - \left[x - i - \frac{1}{d'} \right]_+ \right) = 1$. Thus in general, this expression in the sum equals the indicator function $\mathbb{1}\{x > i\}$. Thus our neural network computes $\mathcal{N}_{d'}^{CEI}(x) = -d' + \sum_{i \in \{-d', \ldots, d'\}} \mathbb{1}\{x > i\}$, which for $x \in [-d', d']$ must return exactly $\lceil x \rceil$, as desired.

The number of hidden layers, the width, and the magnitudes of weights are clear from the definition. ■

G.12. Hashing neural network

In our construction, in one of the steps we reduce the dimensionality of our problem by hashing the locations in a sparse vector to a smaller-dimensional vector, according to hash functions h belonging to some family $\mathcal{H}$. Each such hash function is an explicit linear transformation with $0, 1$ coefficients: for a fixed hash function $h : [d'] \to [p']$, and given an input $\mathbf{x} \in \mathbb{R}^{d'}$, we output $\mathbf{y} \in \mathbb{R}^{p'}$ where $y_i = \sum_{j:\, h(j)=i} x_j$. This linear transform uses 0 ReLU layers in a neural network; however, we choose to include an artificial ReLU layer in our construction to separate this linear transform from any subsequent processing that occurs. This helps us to treat this neural network as an independent primitive in our construction, simplifying the explanation of our construction.

Definition 55 *Given a hash function $h : [d'] \to [p']$, we define our* hashing *neural network $\mathcal{N}_h^H : \mathbb{R}^{d'} \to \mathbb{R}^{p'}$ as,*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{d'},$$
$$\mathcal{N}_h^H(\mathbf{x})[i] = \left[\sum_{j:\, h(j)=i} x_j \right]_+ .$$

Lemma 56 *The hashing neural network $\mathcal{N}_h^H$ (Definition 55) when given an non-negative input $\mathbf{x} \in \mathbb{R}^{d'}$ outputs $\mathbf{y} \in \mathbb{R}^{p'}$ where each coordinate y_i is the sum of entries of $\mathbf{x}$ hashed to position i. Moreover, hashing neural network require one hidden layer, a width of p' and weights in $\{0, 1\}$.*

Proof The first statement is true by definition and noting that the entries of $\mathbf{x}$ are positive, while the second statement is straightforward from the definition. ■

G.13. Block extraction neural network

Recall that Algorithm 7 takes as input a sparse vector with s non-zero entries, and aims to return its non-zero entries by first trying a few hash functions to hash the locations to a smaller domain, and then identifying a hash function that leads to no collisions, so that we can extract the non-zero entries with small width. In this section we describe the neural network that looks through the results of applying q different hash functions, each mapping to a set of size p, and identifies the results of the first hash function that has led to zero collisions. Explicitly, we describe a neural network that takes as input q blocks of size p, and returns the first of these blocks that has exactly s non-zero entries; we call this the *block extraction* neural network. This network essentially consists of three layers of indicator functions, assembled via appropriate linear transforms to compute the desired output.

Definition 57 *We define our* block extraction *neural network $\mathcal{N}_{\delta,p}^{BE} : \mathbb{R}^{pq} \times \mathbb{R} \to \mathbb{R}^p$ by*

$$\text{Given } \mathbf{x} \in \mathbb{R}^{pq}, s \in \mathbb{R},$$
$$c_i = \mathcal{N}_\delta^{NZC}(x_{p(i-1)+1}, \ldots, x_{p \cdot i}),\ \forall i \in [q],$$
$$m_i = \mathcal{N}_\delta^{C}(c_i, s-1) - \mathcal{N}_\delta^{C}(c_i, s),\ \forall i \in [q],$$
$$\mathcal{N}_{\delta,p}^{BE}(\mathbf{x}, s)[j] = \sum_{i=1}^{q} \mathcal{N}^{IFP}\left(x_{j+(i-1)p}, m_i - 1 - \sum_{i'=1}^{i-1} m_{i'} \right), \forall j \in [p]$$

where $\mathcal{N}^{BE}_{\delta,p}(\mathbf{x}, s)[j]$ is the j'th coordinate of the output.

Lemma 58 *The* block extraction *neural network $\mathcal{N}^{BE}_{\delta,p} : \mathbb{R}^{pq} \times \mathbb{R} \to \mathbb{R}^p$, when given an input $\mathbf{x} \in \mathbb{R}^{pq}$ all of whose entries are in the set $\{0\} \cup [\delta, 1]$, and given an input $s \in \mathbb{Z}$, will consider $\mathbf{x}$ as being divided into q blocks of size p, and will return a copy of the first block that contains exactly s non-zero entries (returning zeros if no such block exists).*

The block extraction *neural network $\mathcal{N}^{BE}_{\delta,p}$ requires 3 hidden layers, has width $\mathcal{O}(pq)$, and the magnitudes of the weights are bounded by $\frac{1}{\delta}$.*

Proof We first apply the *non-zero counter* neural network $\mathcal{N}^{NZC}_{\delta}$ (Definition 37) to each of the q blocks of the input, correctly storing in c_i the number of non-zero entries in block i, by Lemma 38. Next, for each c_i we compute in m_i the indicator value of whether $c_i = s$, which we compute by two applications of the comparison neural network $\mathcal{N}^{C}_{d}$ (defined in Definition 35 and shown correct in Fact 36).

The final output step is the most intricate. Recall that the *indicator function product* neural network $\mathcal{N}^{IFP}$, on input a real number $y \in [0, 1]$ and an integer r, returns $y \cdot \mathbb{1}\{r = 0\}$ (see Definition 43 and Lemma 44). Thus the j'th output of our neural network equals $\sum_{i=1}^{q} x_{j+(i-1)p} \cdot \mathbb{1}\left\{m_i = 1 + \sum_{i'=1}^{i-1} m_{i'}\right\}$. We analyze the indicator function: since m_i each m_i is either 0 or 1, the indicator function condition "$m_i = 1 + \sum_{i'=1}^{i-1} m_{i'}$" will be true only when m_i is 1, and when i is the *first* such index (so that $\sum_{i'=1}^{i-1} m_{i'}$ will be 0). Thus the j'th output of our neural network looks at the j'th element of each of the q different blocks $i \in [p]$, and outputs it only for the *first* block with exactly s non-zero entries, as desired.

The bounds on the width, depth, and weights result from corresponding bounds on the components $\mathcal{N}^{NZC}_{\delta}$, $\mathcal{N}^{C}_{\delta}$, and $\mathcal{N}^{IFP}$ from Fact 36 and Lemmas 38 and 44 respectively. ■