A Theoretical View on Sparsely Activated Networks

Cenk Baykal [*1] **Nishanth Dikkala** [*2] **Rina Panigrahy** [*2] **Cyrus Rashtchian** [*1] **Xin Wang** [*1]

Abstract

Deep and wide neural networks successfully fit very complex functions today, but dense models are starting to be prohibitively expensive. To mitigate this, one promising research direction is networks that activate a sparse subgraph of the network. The subgraph is chosen by a data-dependent routing function, enforcing a fixed mapping of inputs to subnetworks (e.g., the Mixture of Experts (MoE) paradigm). However, there is little theoretical grounding for these sparsely activated models. As our first contribution, we present a formal model of such sparse networks that captures salient aspects of popular MoE architectures. Then, we show how to construct sparse networks that provably match the approximation power and total size of dense networks on Lipschitz functions. The sparse networks use exponentially fewer inference operations than dense networks, leading to a faster forward pass. This offers a theoretical insight into why sparse networks work well in practice. Finally, we present empirical findings that support our theory; compared to dense networks, sparse networks give a favorable trade-off between number of active units and approximation quality.

1. Introduction

Overparameterized networks yield performance gains as their sizes increase. This trend has been most prominent with large transformer-based language models (Brown et al., 2020; Devlin et al., 2018; Raffel et al., 2019). However, using large, dense networks makes training/inference expensive, and computing a forward pass may require trillions of floating point operations (FLOPs). It is an active area of research to improve the scalability and efficiency without decreasing the expressiveness/quality of the models.

One way to achieve this goal is to only activate part of the network at a time. For example, the Mixture of Experts (MoE) paradigm (Jordan & Jacobs, 1994; Shazeer et al., 2017) uses a two-step approach. First, each input is mapped to a certain subnetwork, known as an expert. Then, upon receiving this input, only this particular subnetwork performs inference, leading to a smaller number of operations compared to the total number of parameters across all experts. Switch Transformers (Fedus et al., 2021) successfully use a refined version of the MoE idea, where the input may be the embedding of a token or part of a hidden layer's output. Researchers have investigated many ways to perform the mapping, such as Scaling Transformers (Jaszczur et al., 2021) or using pseudo-random hash functions (Roller et al., 2021). In all cases, the computation of the mapping function, a.k.a. the 'routing' function, takes significantly less time than the computation across all experts.

The success of these approaches is surprising. A natural conjecture is that restricting to a subnetwork would *reduce* expressive power and quality. However, the guiding wisdom is that not all parameters of a network are required for the model to make its prediction for any given example. Our goal is to analyze the approximation power of sparse models.

Our Results. Our first contribution is a formal model of networks that have one or more sparsely activated layers with data-dependent sparsity. We show that our model captures popular architectures (e.g., Switch and Scaling Transformers) by simulating the sparse layers in these models.

We next prove that sparse models suffice to learn a fairly large class of functions. One of our main techniques is to use locality sensitive hashing (LSH) to determine the sparse activation pattern. LSH maps points in a topological space with a distance measure (like $\mathbb{R}^d$) to 'buckets' such that nearby points map to same bucket. The total number of buckets used is the size of the hash table. In Theorem 4.1, we show that LSH-based sparse models can approximate real-valued Lipschitz functions in $\mathbb{R}^d$. We assume that our inputs lie in a k-dimensional manifold within $\mathbb{R}^d$ ($k < d$). To get ϵ approximation error, we need an LSH table of size approximately $O((\sqrt{dk}/\epsilon)^k)$ but a forward pass only requires time $O(dk \log(1/\epsilon))$ as only one of the $O((\sqrt{dk}/\epsilon)^k)$ non-empty buckets are accessed for any given example.

[1]**AUTHORERR: Missing \icmlaffiliation.** [2]Google Research. Correspondence to: Nishanth Dikkala <nishanthd@google.com>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

In Theorem 4.2, we complement our upper bounds by proving a lower bound of $\Omega((2\sqrt{d}/\epsilon)^k)$ on the size needed for both dense and sparse models (when points live in k dimensions). This lower bound implies that a forward pass on a dense model takes time $\Omega(d(2/\sqrt{k}\epsilon)^k)$, which is exponentially worse than the time taken by the sparse model. Altogether, we show that for the general class of Lipschitz functions, sparsely activated layers are as expressive as dense layers but need to perform significantly fewer floating point operations (FLOPs) per example. We perform experiments in Section A that investigate the relative power of various models. By studying scaling behaviors as the model size grows, we demonstrate models with data-dependent sparse layers outperform dense models of the same size.

Related Work. Sparsely activated networks have had enormous empirical success (Artetxe et al., 2021; Du et al., 2021; Kim & Awadalla, 2020; Nie et al., 2021; Riquelme et al., 2021; Shazeer et al., 2017; Wang et al., 2021). The Switch Transformer (Fedus et al., 2021) is one of the first major, contemporary applications of the sparsely activated layers. Follow-up works such as Scaling Transformers (Jaszczur et al., 2021) and other hash functions (Roller et al., 2021) aim to improve the sparse layers. These papers build upon seminal MoE works (Jacobs, 1995; Jacobs et al., 1991; Jordan & Jacobs, 1994), and other uses of the MoE paradigm (Lepikhin et al., 2021; Shazeer et al., 2018). To the best of our knowledge, there is no systematic theoretical study of modern sparsely activated networks.

The above work on dynamic sparsity builds on previous *static* sparsity efforts, e.g., weight quantization (Li et al., 2020), dropout (Srivastava et al., 2014), and pruning (see the survey (Hoefler et al., 2021) and references). Static sparsity means that the subnetwork activation does not change in a data-dependent way. The focus is on generalization and compression, instead of achieving fast inference time with a huge number of parameters. Our work builds on locality sensitive hashing (LSH), a well-studied technique for approximate nearest neighbor search (see the survey (Andoni et al., 2018) or the book (Har-Peled, 2011) and references therein for LSH background). For uses of LSH in deep learning, sketch-based memory improves network capacity (Ghazi et al., 2019; Panigrahy et al., 2021). Other work uses LSH to improve training time or memory (Chen et al., 2020; 2015; Rae et al., 2016). Our work differs from the prior studies because we implement the LSH-based approach with sparsely activated networks, with the goal of reducing inference time and achieving low regression error.

2. Preliminaries

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a multivariate real-valued function that we want to learn with a neural network. We consider regression, and we aim to minimize the mean-squared error or the ℓ_∞ error. For a function f defined on a set Γ and an estimator $\hat{f}_n$, we define $\|f - \hat{f}_n\|_\infty = \sup_{x\in\Gamma} |f(x) - \hat{f}_n(x)|$. For some results, we approximate f on a subset $\mathcal{V} \subseteq \mathbb{R}^d$. For example, $\mathcal{V}$ may be the intersection of $[-1, 1]^d$ and a k-dimensional subspace. We also consider the case when f is L-Lipschitz, meaning that $|f(x)-f(x')| \leq L\cdot\|x-x'\|_2$ for all $x, x' \in \mathbb{R}^d$. We define $[n] = \{1, 2, \ldots, n\}$.

2.1. Data-Dependent Sparse Model

A dense neural network g with a fully connected final layer can be expressed as $g(x) = A \cdot \phi(x)$ where $A \in \mathbb{R}^{1\times t}$ is a matrix and $\phi : \mathbb{R}^d \to \mathbb{R}^t$ is a function (e.g., ϕ captures the representation learned up until the final layer). Here, t is the width of the final layer and d is the input dimensionality.

Our focus is on networks with a sparsely activated final layer which we call the *Data-Dependent Sparse Model (*DSM*)*. Formally, let t be the width, and let $s \leq t$ be a sparsity parameter. Then, we consider functions g of the form $g(x) = A^x \cdot \phi(x)$ where $A^x \in \mathbb{R}^{1\times t}$ and $\phi : \mathbb{R}^d \to \mathbb{R}^t$. The crux of the model is the final layer. The sparsity comes from letting $A^x = A \circ \mathsf{mask}(x)$, where $\mathsf{mask}(x) \in \{0, 1\}^{1\times t}$ is an s-sparse indicator vector, and "$\circ$" is the entry-wise product. The mask zeroes out certain positions, and A contains the learned weights but no longer depends on x. Intuitively, the mask is the "routing function" for the sparse activations. Under the above definitions, let $\mathsf{DSM}(d, s, t)$ be the set of functions $g = (A \circ \mathsf{mask}(x)) \cdot \phi(x)$. In what follows, we use A^x as shorthand for $A \circ \mathsf{mask}(x)$.

In the DSM model, we wish to understand the effect of sparsity on how well the network can approximate certain functions. Transformers may have multiple sparse layers. To capture this, we can compose DSM functions. For example, two sparse layers comes from $g(x) = A \circ \mathsf{mask}_2(x) \circ \phi_2(\mathsf{mask}_1(x) \circ \phi_1(x))$. We focus on a single sparse layer in what follows, which suffices for our main results.

2.2. Hash-based Routing

Prior sparse models compute hash functions of the input vector x to determine mask and the network activations (Roller et al., 2021). Our main theorem uses this strategy, considering LSH families. LSH has the property that nearby points are more likely to end up in the same hash bucket than far away points. We can use LSH to define a general class of efficient regression functions. In Section 3, we prove that the DSM model captures popular sparse architectures and the following LSH model.

LSH **Model.** We review a popular LSH family for the Euclidean distance ((Datar et al., 2004)). Let $h_1, \ldots, h_m : \mathbb{R}^d \to \{-1, 1\}$ be m distinct hash functions. Partition the space into 2^m buckets based on the m sign patterns

$z_x = (h_1(x), \ldots, h_m(x))$ over all $x \in \mathbb{R}^d$. Then, for each $z \in \{-1, 1\}^m$, we can specify a function $\hat{g}_z(x)$, where the goal is for g_z to approximate the target function f in the part of space associated with z (i.e., points x that have pattern z under $h_1, \ldots, h_m$). More generally, we can allow s sets of such hash functions $(h_1^i, \ldots, h_m^i)$, and s sets of these approximation functions $(\hat{g}^1_{z^1}, \ldots, \hat{g}^s_{z^s})$ for $i = 1, \ldots, s$. On input x, we compute the sign patterns $z^1, \ldots, z^s$ and output $g(x) = \sum_{i=1}^s \alpha_i \hat{g}^i_{z^i}(x)$. Further, we can restrict each $\hat{g}^i_z$ to be a degree $\Delta \geq 0$ polynomial. For a fixed LSH family of possible hash functions, we let $\mathsf{LSH}(d, s, m, \Delta)$ denote this class of functions. In many cases, we only need $\hat{g}^i_z$ to be a constant function, i.e., $\Delta = 0$, and we shorten this as $\mathsf{LSH}(d, s, m, 0) := \mathsf{LSH}(d, s, m)$.

Euclidean LSH Model (Datar et al., 2004). This is a popular LSH family for points in $\mathbb{R}^d$. In this case, each hash function outputs an integer (instead of ± 1 above). Each bucket in this model is defined by a set of hyperplane inequalities. There are two parameters (D, ϵ). We sample D random directions $a_1, \ldots, a_D \in \mathbb{R}^d$ where each coordinate of each a_i is an independent normal variable. In addition we sample $b_i \sim \mathrm{Unif}[0, \epsilon]$ independently for $i \in [D]$. For a point $x \in \mathbb{R}^d$, we compute an index into a bucket via a function $h_i : \mathbb{R}^d \to \mathbb{Z}$ defined as $h_i(x) = \left\lfloor \frac{a_i^\top x + b_i}{\epsilon} \right\rfloor$. Here, the index i ranges over $i \in [D]$, leading to a vector of D integers.

3. Simulating Models with DSM

We formally justify the $\mathsf{DSM}(d, s, t)$ model by simulating other models using it. We start with simple examples (interpolation and k nearest neighbor (k-NN) regression), then move on to transformers and the LSH model. For the simple examples, we only need one hidden layer, where $\phi(x) = \sigma(Bx)$ for a matrix $B \in \mathbb{R}^{t \times d}$ and non-linearity σ.

Interpolation. We show how to compute f at t points $x_1, \ldots, x_t$. When $s = 1$, we can set $A_i = f(x_i)/\langle b_i, x_i \rangle$, where b_i is the ith row of B. Further, we let $\mathsf{mask}(x_i)$ have a one in the ith position and zeroes elsewhere. Then, $g(x_i) = (A \circ \mathsf{mask}(x_i))Bx_i = f(x_i)$.

Sparse networks perform k-NN regression. We sketch how the $\mathsf{DSM}(d, k, n)$ model can simulate k-NN with $g(x) = A\sigma(Bx)$. Let the rows of B be a set of n unit vectors $b_1, \ldots, b_n \in \mathbb{R}^d$. For the target function f, let $A = \frac{1}{k}(f(b_1), \ldots, f(b_n))$. Define $\sigma(Bx)$ to have ones in the top k largest values in Bx and zeroes elsewhere. For a unit vector x, these k positions correspond to the k largest inner products $\langle x, b_i \rangle$. Since $\|x - b_i\|_2 = 2 - 2\langle x, b_i \rangle$, the non-zero positions in $\sigma(Bx)$ encode the k nearest neighbors of x in $\{b_1, \ldots, b_n\}$. Thus, $g(x)$ computes the average of f at these k points, which is exactly k-NN regression. Moreover, only k entries of A are used for any input, since $\sigma(Bx)$ is k-sparse; however, computing $\sigma(Bx)$ takes $O(nd)$ time. While there is no computational advantage from the sparsity in this case, the fact that DSM can simulate k-NN indicates the power of the model.

3.1. Simulating transformer models with DSM

Real-world networks have many hidden layers and multiple sparse layers. For concreteness, we describe how to simulate a sparsely activated final layer. As mentioned above, we can compose functions in the DSM model to simulate multiple sparse layers.

Switch Transformers (Fedus et al., 2021). The sparse activations in transformers depend on a *routing function* $R : \mathbb{R}^d \to \{1, \ldots, L\}$, where $R(x)$ specifies the subnetwork that is activated (for work on the best choice of R, see e.g., (Roller et al., 2021)). To put this under the $\mathsf{DSM}(d, s, t)$ model, consider a set of trainable matrices $A_1, \ldots, A_L \in \mathbb{R}^{1 \times s}$, where the total width is $t = s \cdot L$. On input x, we think of A^x as a $1 \times t$ matrix with s non-zero entries equal to $A_{R(x)}$. In other words, A is the concatenation of $A_1, \ldots, A_L$, and $\mathsf{mask}(x)$ is non-zero on the positions corresponding to $A_{R(x)}$.

Scaling Transformers (Jaszczur et al., 2021). The key difference between Switch and Scaling transformers is that the latter imposes a block structure on the sparsity pattern. Let t be the width, and let s be the number of blocks (each of size $t' = t/s$). Scaling Transformers use only one activation in each of the s blocks. In the $\mathsf{DSM}(d, s, t)$ model, we capture this with A^x as follows. Let $e_i \in \{0, 1\}^{t'}$ denote the standard basis vector (i.e., one-hot encoding of $i \in [t']$). The sparsity pattern is specified by indices $(i_1, \ldots, i_s)$. Then, $A^x = (\alpha_1 e_{i_1}, \ldots, \alpha_b e_{i_s})$ for scalars $\alpha_1, \ldots, \alpha_s \in \mathbb{R}$.

3.2. Simulating the LSH model using DSM

We explain how to simulate the LSH model using the DSM model. The key insight is to view A^x as depending on the LSH buckets that contain x, where we have s non-zero weights for the s buckets that contain each input. In the $\mathsf{LSH}(d, s, m, \Delta)$ model, there are s sets of m hash functions, leading to $s \cdot 2^m$ hash buckets. We use width $t = s \cdot 2^m$ for the $\mathsf{DSM}(d, s, t)$ network. The entries of A^x are in one-to-one mapping with the buckets, where only s entries will be non-zero depending on the s buckets that x hashes to, that is, the values $(h_1^i(x), \ldots, h_m^i(x)) \in \{-1, 1\}^m$ for $i = 1, 2, \ldots, s$.

We now determine the values of these s non-zero entries. We store a degree Δ polynomial $\hat{g}(x) : \mathbb{R}^d \to \mathbb{R}$ associated with each bucket. For our upper bounds, we only need degree $\Delta = 0$, but we mention the general case for completeness. If $\Delta = 0$, then $\hat{g}$ is simply a constant α depending on the bucket. An input x hashes to s buckets, associated

with s scalars $(\alpha_1, \ldots, \alpha_s)$. To form A^x, set s entries to the α_i values, with positions corresponding to the buckets. For degree $\Delta \geq 1$, we store coefficients of the polynomials $\hat{g}$, leading to more model parameters. Section 4 contains details on using LSH to approximate Lipschitz functions with sparse networks.

Computing and storing the LSH buckets. Determining the non-zero positions in A^x only requires $O(sm)$ hash computations, each taking $O(d)$ time with standard LSH families (e.g., hyperplane LSH). We often take m to be a large constant. Thus, the total number of operations to compute a forward pass in the network $O(smd) \approx O(sd)$. The variable m above determines the total number of distinct buckets we will have (2^m). For an n point dataset, $m = O(\log n)$ is a realistic setting in theory. Therefore, $2^m = \text{poly}(n)$ is often a reasonable size for the hash table. The hash function typically adds very few parameters. In summary, the LSH computation does not asymptotically increase the FLOPs for a forward pass in the network.

4. Data-Dependent Sparse Models are more Efficient than Dense Models

For a very general class of functions, LSH-based learners yield similar ℓ_∞ error as dense neural networks while making inference significantly more efficient. It is a common belief in the machine learning community that although many of the datasets we encounter can appear to live in high-dimensional spaces, there is a low-dimensional manifold on which the inputs lie. To model this, we assume in our theory that the inputs lie in a k-dimensional subspace (a linear manifold) of $\mathbb{R}^d$. Here $k \ll d$. Theorem 4.1 shows that the LSH model we propose can learn high-dimensional Lipschitz functions with a low ℓ_∞ error efficiently when the input comes from a uniform distribution on an unknown low-dimensional subspace. Theorem C.1 extends this result to when the input comes from an unknown manifold with a bounded curvature. We present Theorem 4.1 here and defer Theorem C.1 to Section C. All proofs are in the appendix.

Theorem 4.1. *For any $f : [-1, 1]^d \to \mathbb{R}$ that is L-Lipschitz, and for an input distribution $\mathcal{D}$ that is uniform on a k-dimensional subspace in $[-1, 1]^d$, an LSH-based learner can learn f to ϵ-uniform error using a hash table of size $O(L\sqrt{d}^k/\epsilon^k)$ with probability ≥ 0.8. The total time for a forward pass on a test sample is $O(dk \log(L\sqrt{d}/\epsilon))$.*

The key idea behind this theorem is to use LSH to produce a good routing function. The locality of the points hashed to an LSH bucket lets us control the approximation error. By using a large number of buckets, we can ensure their volume is small. Then, outputting a representative value suffices to locally approximate the target Lipschitz function (since its value changes slowly).

The above construction assumes knowledge of the dimensionality of the input subspace k. Fortunately, any upper bound on k would also suffice. The table size of the LSH model scales exponentially in k but not d. Thus, an LSH-based learner adapts to the dimensionality of the input subspace.

Theorem 4.1 shows that sparsely activated models (using LSH of a certain table size) are powerful enough to approximate and learn Lipschitz functions. Next, we show a complementary nearly matching lower bound on the width required by dense model to approximate the same class of functions. We use an information theoretic argument.

Theorem 4.2. *Consider the problem of learning L-Lipschitz functions on $[-1, 1]^d$ to ℓ_∞ error ϵ when the inputs are sampled from a uniform distribution over an unknown k-dimensional subspace of $\mathbb{R}^d \cap [-1, 1]^d$. A dense model of width w with a random bottom layer requires $w = \Omega\left(\frac{(\sqrt{d}L)^k}{(C\epsilon)^k}\right)$, for a sufficiently large constant C.*

Our approach for this theorem is to use a counting argument. We first bound the number of distinct functions, which is exponential in the number of parameters (measured in bits). We then construct a large family of target functions that are pairwise far apart from each other. Hence, if we learn the wrong function, we incur a large error. Our function class must be large enough to represent any possible target function, and this gives a lower bound on the size of the approximating network.

Theorem 4.2 shows a large gap in the time complexity of inference using a dense model and an LSH model. The inference times taken by a dense model vs. a sparse model differ exponentially in $1/\epsilon$.

$$\textbf{Sparse: } O\left(dk \log(1/\epsilon)\right) \quad \text{vs.} \quad \textbf{Dense: } \Omega\left(d\left(\frac{2\sqrt{d}}{C\epsilon}\right)^k\right)$$

Overall, the above theorems show that LSH-based sparsely activated networks can approximate Lipschitz functions on a k-dimensional subspace. The size and sample complexity match between sparse and dense models, but the sparse models are exponentially more efficient for inference.

References

Andoni, A., Indyk, P., and Razenshteyn, I. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pp. 3287–3318. World Scientific, 2018.

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., et al.

Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, B., Liu, Z., Peng, B., Xu, Z., Li, J. L., Dao, T., Song, Z., Shrivastava, A., and Re, C. Mongoose: A learnable lsh framework for efficient neural network training. In *International Conference on Learning Representations*, 2020.

Chen, W., Wilson, J., Tyree, S., Weinberger, K., and Chen, Y. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pp. 2285–2294. PMLR, 2015.

Chen, Z. and Dongarra, J. J. Condition numbers of gaussian random matrices. *SIAM Journal on Matrix Analysis and Applications*, 27(3):603–620, 2005.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*, 2021.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Ghazi, B., Panigrahy, R., and Wang, J. Recursive sketches for modular deep learning. In *International Conference on Machine Learning*, pp. 2211–2220. PMLR, 2019.

Har-Peled, S. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

Hinton, G. Lecture Notes, Toronto, Hinton, 2012, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

Jacobs, R. A. Methods for combining experts' probability assessments. *Neural computation*, 7(5):867–888, 1995.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jaszczur, S., Chowdhery, A., Mohiuddin, A., Kaiser, Ł., Gajewski, W., Michalewski, H., and Kanerva, J. Sparse is enough in scaling transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2): 181–214, 1994.

Kim, Y. J. and Awadalla, H. H. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*, 2020.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. E. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *arXiv preprint arXiv:2002.11794*, 2020.

Nie, X., Cao, S., Miao, X., Ma, L., Xue, J., Miao, Y., Yang, Z., Yang, Z., and Cui, B. Dense-to-sparse gate for mixture-of-experts. *arXiv preprint arXiv:2112.14397*, 2021.

Panigrahy, R., Wang, X., and Zaheer, M. Sketch based memory for neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 3169–3177. PMLR, 2021.

Rae, J. W., Hunt, J. J., Harley, T., Danihelka, I., Senior, A., Wayne, G., Graves, A., and Lillicrap, T. P. Scaling memory-augmented neural networks with sparse reads and writes. *arXiv preprint arXiv:1610.09027*, 2016.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N.

Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34, 2021.

Roller, S., Sukhbaatar, S., Szlam, A., and Weston, J. Hash layers for large sparse models. *arXiv preprint arXiv:2106.04426*, 2021.

Rudelson, M. and Vershynin, R. Smallest singular value of a random rectangular matrix. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 62(12):1707–1739, 2009.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR (Poster)*. OpenReview.net, 2017. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2017.html#ShazeerMMDLHD17.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., Sepassi, R., and Hechtman, B. Mesh-tensorflow: Deep learning for supercomputers. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/3a37abdeefe1dab1b30f7c5c7e581b93-Paper.pdf.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Valiant, G. and Valiant, P. Estimating the unseen: an n/log (n)-sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 685–694, 2011.

Wang, S., Sun, Y., Xiang, Y., Wu, Z., Ding, S., Gong, W., Feng, S., Shang, J., Zhao, Y., Pang, C., et al. Ernie 3.0 titan: Exploring larger-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2112.12731*, 2021.

A. Experiments

To empirically verify our theoretical findings, we align our experiments with our proposed models and compare dense models, data-dependent sparse models (DSM), LSH models, and sparse models with random hash based sparse layers. While DSM and LSH models are analyzed in the previous section, random hash based layers introduce sparsity with a random hash function, as an alternative of learnable routing modules and LSH in MoE models (Roller et al., 2021). Our goal is to show that the DSM and LSH models achieve small MSE while using much fewer activated units than dense models.

A.1. Experimental set-up

Dense, DSM and random hash sparse models contain a random-initialized, non-trainable bottom layer, (a Top-K layer for DSM and a random hash layer for random hash models to enforce sparsity), and a trainable top layer, with varying number of hidden units and sparsity levels. LSH models have non-trainable hyperplane coefficients for hashing and a trainable scalar in each bucket (the scalar determines the output of the network for points in that bucket). We compare dense models and three sparse models (DSM, LSH, and random).

We evaluate with synthetic data from two random, Lipschitz target functions that are commonly used basis functions for arbitrary continuous functions. These random functions allow us to empirically evaluate the construction from Theorem 4.1, while comparing different routing functions.

Random polynomial. $p(x)$ of degree d for $x \in \mathbb{R}^n$ with sum of coefficient absolute values < 1.

Random hypercube function. $f : [-1, 1]^n \to \mathbb{R}$ which interpolates the indicator functions at each corner with random $\{-1, 1\}$ value at each corner. Concretely, the function is defined as follows: for each corner point $y \in \{-1, 1\}^n$, its indicator function is $I_y(x) = \prod_{i=1}^{n} \frac{1+y_i x_i}{2}$. Sample random values $v_y \in \{-1, 1\}$ with probability $(0, 5, 0.5)$ independently for each $y \in \{-1, 1\}^n$, the random hypercube polynomial function is $f(x) = \sum_{y \in \{-1,1\}^n} v_y I_y(x)$.

Random function generation. For the random polynomial functions, we randomly generate coefficients of the monomials by sampling from a uniform distribution $\mathcal{U}([-1, 1])$ and scale the coefficients so that their absolute values sum up to 1.0 (this is to ensure the Lipschitz constant of the generated function is bounded by a constant independent of dimension and degree of the polynomial). For the random hypercube function, we sample values of the function at each corner independently from a uniform distribution on $-1, 1$, and interpolate using the indicator functions.

Train/Test dataset generation. For a given target function f (polynomial or hypercube), we sample independently from $\mathcal{U}([-1, 1]^n)$ (where n is the input dimension) to generate the input features x and compute target value $y = f(x)$. The train dataset contains 2^{16} (x, y) pairs and the test dataset contains 2^{14} (x, y) pairs.

Training setting. All the models in Section A are trained for 50 epochs using the RMSProp (Hinton) optimizer with a learning rate of 10^{-5}. For the one dimension example in Section 1, the model is trained for 200 epochs using the RMSProp optimizer with a learning rate of 5×10^{-6}.

Random hash sparse model. We discussed the design of DSM and LSH models in Section 2. Here we present the details of the random hash model, where the sparsity pattern is determined by a random hash of the input data (i.e. the same input data would always have the same sparsity pattern). The following code snippet shows the generation of a random mask that only depends on the input data using TensorFlow 2.x.

```
import tensorflow as tf

# seed: a fixed random seed
# inputs: the input tensor
# mask_dim: size of the masked tensor
# num_buckets: a large integer
# k: the dimension after masking

input_dim = inputs.shape[-1]
if input_dim != mask_dim:
  proj = tf.random.stateless_normal(
    shape=(input_dim, mask_dim),
```

```
    seed=seed)
  inputs = tf.einsum(
    '...i,io->...o', inputs, proj)
hs = tf.strings.to_hash_bucket_fast(
  tf.strings.as_string(inputs),
  num_buckets=num_buckets)
top_k_hash = tf.expand_dims(
  tf.nn.top_k(hs, k).values[..., -1],
  axis=-1)
mask = hs >= top_k_hash
```

A.2. Learning random polynomials under other parameter settings

We present experiment results for learning random polynomial target functions with low intrinsic dimensions. To be precise, the target polynomial is $p(Ax)$, where p is a polynomial of degree d with sum of coefficient absolute value < 1, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{k \times n}$ is a matrix with random orthogonal rows, and $n > k$. Note now the intrinsic dimension of the domain is k, while the inputs x has higher dimension n. In Figure 3, we compare the mean squared loss for dense models and DSMs for $n = 64$, $k = 8$, and $d = 4$. We observe similar behavior as Figure 1, where the input dimension is the same as the intrinsic dimension, validating our analysis in Section 3.

We will also present results on a real dataset, CIFAR-10, which corroborates our findings from the synthetic data experiments.

A.3. Results

MSE for random functions. Figures 1 and 2 show the scaling behavior of the DSM and LSH models for a random polynomial function and hypercube function. Sparsity helps in both DSM and LSH models, both achieving better quality than dense models using the same number of activated units. In Figure 4, we further compare the DSM and LSH models with the random hash sparse models, and we see random hash sparse models underperform dense models, suggesting data-dependent sparsity mechanisms (such as DSM and LSH) are effective ways to utilize sparsity in models.

FLOPs. To further qualtify the efficiency gain of sparse models, we compare the MSE at the same FLOPs for sparse/dense models in Table 1. The first column is the # FLOPs for the dense model; models in the 3rd and 4th columns use same # FLOPs but have more parameters (only 50% or 25% active). DSM uses only 18k FLOPs and gets smaller MSE than dense model with 73k FLOPs.

FLOPs	eval MSE (dense)	eval MSE (DSM 50% sparsity)	eval MSE (DSM 25% sparsity)
18432	0.01015	0.01014	**0.009655**
36864	0.01009	0.007438	**0.005054**
73728	0.01046	0.006115	**0.001799**

Table 1. FLOPs and evaluation Mean Squared Error (eval MSE).

CIFAR-10. We also compare the scaling behavior of DSM and dense models on CIFAR-10 (Krizhevsky et al., 2009). The baseline model is a CNN with 3 convolutional layers (followed by max-pooling), a dense layer with varying number of units, and a final dense layer that computes the logits (referred as CNN + dense). For the data-dependent sparse models, we use the same architecture, except we change the penultimate dense layer with a data-dependent sparse layer (referred as CNN + DSM). Both models are trained with ADAM optimizer for 50 epochs and evaluated on the test dataset for model accuracy with no data augmentation; see Figure 5 and Table 2 for the accuracy versus number of activated units. As with the synthetic datasets, DSMs outperform dense models at the same number of activated units.

A.4. Discussion

Our experimental results (e.g., Figures 1, 2, and 4) show that sparsely activated models can efficiently approximate both random polynomials and random hypercube functions. Intuitively, the DSM and LSH models employ the sparsity as a way to partition the space into nearby input points. Then, because the target function is Lipschitz, it is easy to provide to local approximation tailored to the specific sub-region of input space. On the other hand, the uniform random hash function performs poorly for these tasks precisely because it does not capture the local smoothness of the target function.

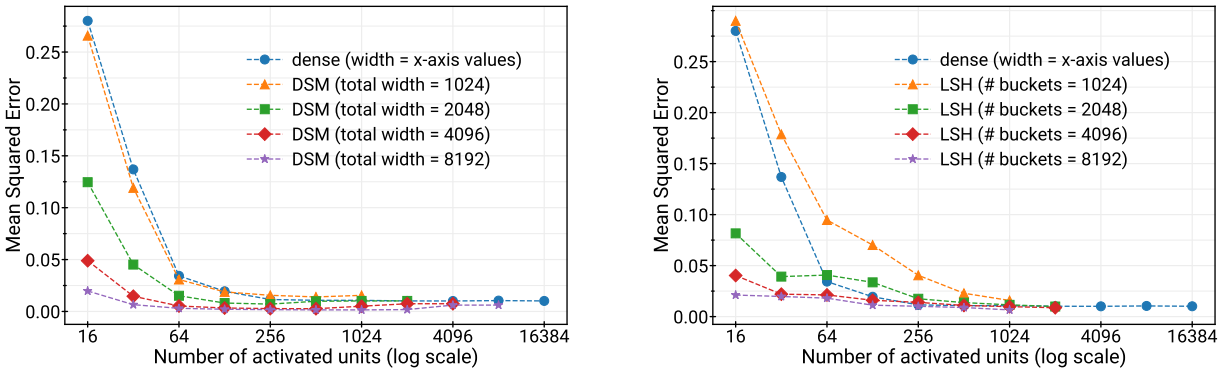


Figure 1. Scaling behavior of DSM and LSH models compared with dense models for a degree 4 random polynomial with input dimension 8: (a) DSM outperforms dense model at the same number of activated units and quality improves as total width increases; (b) LSH model outperforms dense model when number of buckets is large (≥ 2048) and quality improves as number of buckets increase.

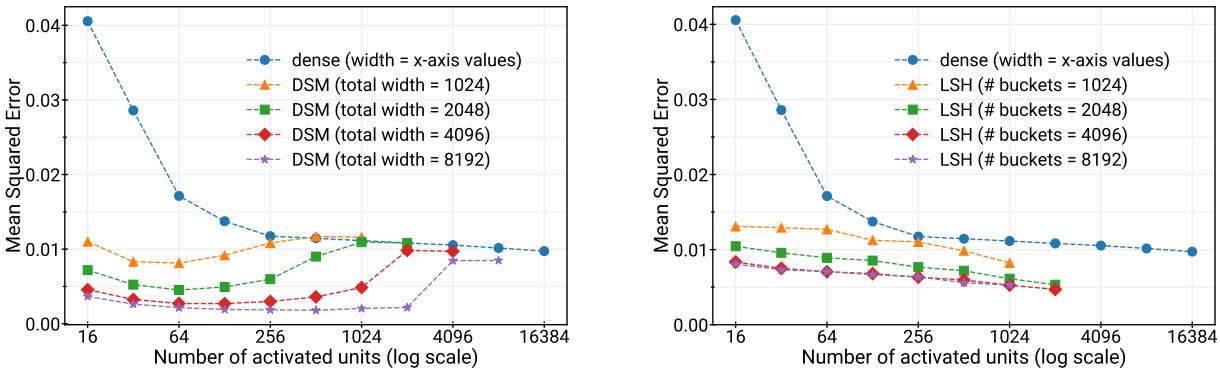


Figure 2. Scaling behavior of DSM and LSH models compared with dense models for a random hypercube function with input dimension 8. Both DSM and LSH models outperform corresponding dense models with the same number of activated units.

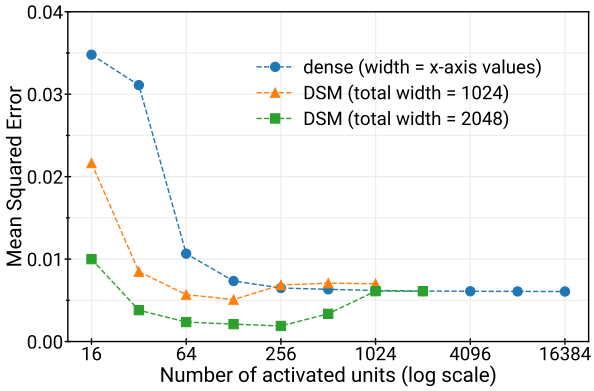
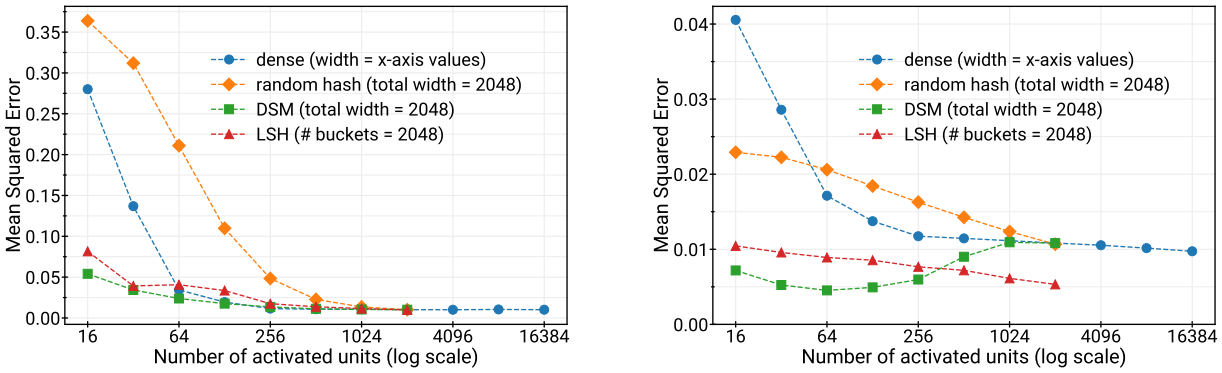


Figure 3. Scaling behavior of DSM compared with dense models for a random polynomial with low intrinsic dimensional domain. Similar to Figure 1, DSM outperforms dense models at the same number of activated units.



(a) random polynomial

(b) random hypercube function

Figure 4. Scaling behavior of dense, random hash, DSM, and LSH models. DSM and LSH models outperform dense models, while random hash models underperform dense models with the same number of activated units.

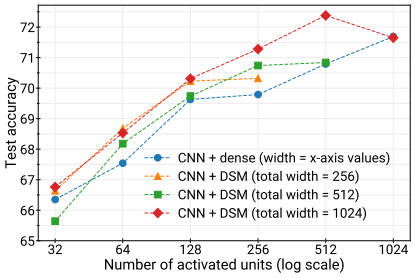


Figure 5. Scaling behavior of DSM compared with dense models on the CIFAR-10 dataset. Similar to Figure 4a, DSM outperforms dense models at the same number of activated units.

Model \ # activated units	256	512
Dense	69.79	70.79
DSM (50% sparse)	**70.74**	71.33
DSM (25% sparse)	69.8	**71.68**

Table 2. CIFAR-10 test accuracy for dense/DSM models with the same number of activated units. While not strictly monotonic, wider and sparser models outperform narrow and dense ones.

On CIFAR-10, we also see that the DSM model performs comparably or better than the dense network. In particular, Figure 5 shows that the "CNN + DSM" approach with total width 1024 improves upon or matches the dense model. In this case, the sparse activation allows the network to classify well while using only a fraction of the number of FLOPs. In Table 2, we see that DSM model outperforms the dense model when we control for the number of activated units in the comparison.

Limitations. Our experiments focus on small datasets and 2-layer networks as a way to align with our theoretical results. Prior work on sparsely activated networks has shown success for large-scale NLP and vision tasks. Our experiments complement previous results and justify the DSM and LSH models by showing their ability to approximate Lipschitz functions (consistent with our theorems). It would be good to further evaluate the DSM and LSH models for large-scale tasks, for example, by using them as inspiration for routing functions in MoE vision transformers, such as V-MoE (Riquelme et al., 2021). It would be interesting to evaluate on larger datasets, such as ImageNet as well. We experimented with a handful of hyperparameter settings, but a full search may lead to different relative behavior between the models (similarly, we only evaluated a few parameter settings for the random functions and the synthetic data generation, which is far from exhaustive).

B. Proofs of Main Upper and Lower Bounds

B.1. Proof of the Lipschitz Upper Bound

Proof of Theorem 4.1. We use the Euclidean-LSH construction of Lemma B.1 with parameter ϵ/L. In any sub-region of the k-dimensional subspace that has a small diameter, the Lipschitz nature of the function together with Lemma B.1 will imply that we can approximate it by just a constant and incur only ϵ error in ℓ_∞. In particular, given a point x_1 belonging to an LSH bucket, we can set $\hat{f}(x) = f(x_1)$ everywhere in that bucket. For any x_2 also mapping to the same bucket, from Lemma B.1, we have that $\|x_1 - x_2\|_2 \leq \epsilon/L$. Since f is L-Lipschitz,

$$|\hat{f}(x_2) - f(x_2)| = |f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|_2 \leq \epsilon. \tag{1}$$

Next we look at how many samples we need to obtain the guarantee $\|\hat{f} - f\|_\infty \leq \epsilon$. A rare scenario that we have to deal with for the sake of completeness is when there exist buckets of such small volume that no training data point has mapped to them and consequently we don't learn any values in those buckets. At test time, if we encounter this rare scenario of mapping to a bucket with no value learnt in it, we simply run an approximate nearest neighbor search among the train points. For our prediction, we use the bucket value associated with the bucket that the approximate nearest neighbor maps to. To control the error when doing such a procedure, we take enough samples to approximately form an $\epsilon/\Gamma L$ cover of Γ for a large enough constant Γ. The size of an $\epsilon/\Gamma L$ cover of Γ is $O((2\Gamma L\sqrt{d}/\epsilon)^k)$. This implies that, via a coupon collector argument, when the input distribution is uniform over the region Γ, $O(k(2\Gamma L\sqrt{d}/\epsilon)^k \log(2\Gamma L\sqrt{d}/\epsilon))$ samples will ensure that with very high probability, for every test point x there exists a train example x_i such that $\|x - x_i\|_2 \leq 2\epsilon/\Gamma L$. The test error is $|f(x) - \hat{f}(x_i)| \leq |f(x) - f(x_i)| + |f(x_i) - \hat{f}(x_i)| = O(\epsilon)$. Computing the exact nearest neighbor is a slow process. Instead we can compute the approximate nearest neighbor using LSH very quickly. We lose another $O(\epsilon)$ error due to this approximation. Choosing Γ appropriately we can make the final error bound exactly ϵ. This leads to our stated sample complexity bound.

This implies that $\|f - \hat{f}\|_\infty \leq \epsilon$. Hence using an Euclidean LSH with $O(k)$ hyperplanes we can learn an ϵ-approximation to f. The time to compute $\hat{f}(x)$ for a new example is the time required to compute the bucket id where it maps to. Since there are k hyperplanes and our input is d-dimensional, computing the projections of x on the k hyperplanes takes $O(dk)$ time. Then we need to perform a division by the width parameter ϵ/L, which would take time equal to the number of bits requires to represent L/ϵ. Hence the total time taken would be $O(dk \log(L/\epsilon))$. □

The above theorem uses a lemma about Euclidean LSH, which we present next.

Lemma B.1. *Consider a Euclidean* LSH *model in d dimensions with Ck hyperplanes and width parameter ϵ where C is a large enough constant. Consider a region Γ defined by the intersection of a k-dimensional subspace with $[-1, 1]^d$. We have that the* LSH *model defines a partitioning of Γ into buckets. Let c be a constant. Then, with probability ≥ 0.9,*

1. *Projecting any bucket of the LSH onto Γ corresponds to a sub-region with diameter $\leq \epsilon/c$.*
2. *At most $\left(\frac{2\sqrt{d}}{\epsilon}\right)^{O(k)}$ buckets have a non-empty intersection with Γ.*

Proof of Lemma B.1. Let $K = Ck$. Let the random hyperplanes chosen by the Euclidean LSH be $a_1, \ldots, a_K$. Let the width parameter used by the LSH be ϵ_{LSH}. The value of ϵ_{LSH} we choose will be determined later. Since the distribution of entries is spherically symmetric, the projection of the vectors onto the k-dimensional subspace will also form a Euclidean-LSH model. Henceforth in our analysis we can assume that all our inputs are projected onto the k-dimensional space Γ and that we are performing LSH in a k-dimensional space instead of a d-dimensional one. Let $A = [a_1, \ldots, a_K]^\top$ be the matrix whose columns are the vectors perpendicular to the hyperplanes chosen by the LSH. Note that $A \in \mathbb{R}^{K \times k}$. Then we have, from tail properties of the smallest singular value distribution of Gaussian random matrices (e.g. see (Chen & Dongarra, 2005; Rudelson & Vershynin, 2009)), for a large enough constant c,

$$\Pr[\sigma_{\min}(A) \geq c\sqrt{k}] \geq 9/10. \tag{2}$$

For two points $x_1, x_2 \in \Gamma$ to map to the same LSH bucket, $\|A(x_1 - x_2)\|_\infty \leq \epsilon_{LSH}$. This implies that $\|A(x_1 - x_2)\|_2 \leq \epsilon_{LSH}\sqrt{k}$, which together with (2) implies that $\|x_1 - x_2\|_2 \leq \epsilon_{LSH}/c$ with probability $\geq 9/10$. At the same time, since we $x_1, x_2 \in [-1, 1]^d$, the maximum distance along any direction is at most the length of any diagonal, which is $2\sqrt{d}$. Moreover, along any hyperplane direction sampled by the LSH, we grid using a width ϵ_{LSH}. Since the total number of hyperplanes is

Ck the maximum number of LSH buckets possible is $\left(\frac{2\sqrt{d}}{\epsilon_{LSH}}\right)^{O(k)}$. We set $\epsilon_{LSH} = c\epsilon$. Then, with high probability over the draw of the hyperplanes, the diameter of any bucket $\leq \epsilon_{LSH}/c = \epsilon$. The upper bound on the maximum number of LSH buckets required to cover the region Γ also follows. □

B.2. Proof of the Dense Lower Bound

Proof of Theorem 4.2. Assuming B bits per parameter, in our dense layer model we have 2^{Bw} distinct possible configurations. We show a lower bound on the width w by constructing a class of functions $\mathcal{F}$ defined on a k-dimensional subspace within $[-1, 1]^d$ such that three properties simultaneously hold:

1. each $f \in \mathcal{F}$ is L-Lipschitz,
2. the number of functions in $\mathcal{F}$ is at least $\Omega(2^{(2\sqrt{d}L/C\epsilon)^k})$
3. for $f_1 \neq f_2 \in \mathcal{F}$, we have $\|f_1 - f_2\|_\infty > \epsilon$.

These three properties together will imply that $w \geq \frac{1}{B}(2\sqrt{d}/C\epsilon)^k$ as otherwise by there would have to be two functions $f_1 \neq f_2 \in \mathcal{F}$ that are approximated simultaneously by the same dense network, which is impossible since $\|f_1 - f_2\|_\infty > \epsilon$. We construct $\mathcal{F}$ as follows. Given the d-dimensional cube $[-1, 1]^d$, we pick a subset of k diagonals of the cube such that they are linearly independent. We consider the k-dimensional region defined by the intersection of the subspace generated by these diagonals and the cube $[-1, 1]^d$. Denote the region we obtain by G. Let $e_1, \ldots, e_k$ form an orthonormal basis for the subspace G lies in. We grid G into k-dimensional cubes of side length $2\epsilon/L$ aligned along its bases $\{e_i\}_{i=1}^k$. For the center of every cube we pick a random assignment from $\{+\epsilon, -\epsilon\}$. Then we interpolate the function everywhere in G such that (i) it satisfies the assigned values at the centers of the cubes and (ii) its value decreases linearly to 0 with radial distance from the center. That is, given the set of cube centers V

$$f(x) = \sum_{v \in V} \max(0, f(v) - L \operatorname{sgn}(f(v))\|x - v\|_2)$$

To understand the Lipschitz properties of such an interpolation, note that the slope at any given point in G is either 0 or L, which bounds the Lipschitz constant by L. The total number of cubes that lie within G is at least $(\sqrt{d}L/C\epsilon)^k$ for some constant C and hence $\mathcal{F}$ contains a total of $(2)^{(\sqrt{d}L/C\epsilon)^k}$ functions. Moreover, given any $f_1, f_2 \in \mathcal{F}$ such that $f_1 \neq f_2$, there exists a cube center where their values differ by 2ϵ giving us the third desired property as well. Consequently, we get that to attain ϵ-uniform error successfully on $\mathcal{F}$ we need

$$2^{Bw} \geq 2^{(\sqrt{d}L/(C\epsilon))^k},$$

which implies that $w = \Omega((\sqrt{d}L)^k/(C\epsilon)^k)$. □

C. LSH Models Can Also Learn Lipschitz Functions on k-Manifolds

A k-dimensional manifold (referred to as a k-manifold) can loosely be thought of as a k-dimensional surface living in a higher dimensional space. For example the surface of a sphere in 3-dimensions is a 2-dimensional manifold. We consider k-manifolds in $\mathbb{R}^d$ that are homeomorphic to a k-dimensional subspace in $\mathbb{R}^d$. We assume that our k-dimensional manifold M_k is given by a transform $f : R^k \to R^k$ applied on k-dimensional subspace of $\mathbb{R}^d$ L_k. To control the amount of distortion that can occur when going from L_k to M_k, the Jacobian of f is assumed to have a constant condition number for all $x \in L_k$. We now state our main upper bound for manifolds, showing that LSH models can adapt and perform well even with non-linear manifolds of a bounded distortion from a linear subspace.

Theorem C.1. *For any $f : [-1, 1]^d \to \mathbb{R}$ that is 1-Lipschitz, and for an input distribution $\mathcal{D}$, which is uniform on a k-manifold in $[-1, 1]^d$, an* LSH *model can learn f to ϵ-uniform error with $O(k\sqrt{dk}^k \log(\sqrt{dk}/\epsilon)/\epsilon^k)$ samples using a hash table of size $O(\sqrt{dk}^k/\epsilon^k)$ with probability ≥ 0.8. The total time required for a forward pass on a new test sample is $O(dk \log(1/\epsilon))$.*

Proof. The main idea of the proof is to follow similar arguments from Theorem 4.1 on the subspace L_k and try to bound the amount of distortion the arguments face when mapped to the manifold M_k. Since we are no longer dealing with a subspace

(linear manifold), the argument that an LSH in d-dimensions can be viewed as an equivalent LSH in k-dimensions does not hold. We use Euclidean-LSH models with $O(d)$ hyperplanes. Furthermore, we will use multiple LSH models each defined using $O(d)$ hyperplanes. The main challenge in the proof is to show that the total number of buckets used in approximating f do not grow exponentially in d, which is a possibility now as we use $O(d)$ hyperplanes.

Lemma C.2. *For any $x \in \mathbb{R}^d$, a d-dimensional sphere of radius $O(\epsilon/d)$ centered at x is fully contained in the bucket where the center of the sphere maps to with probability ≥ 0.9.*

Proof. Along any hyperplane direction the gap between parallel hyperplanes is ϵ. Since any point is randomly shifted before being mapped to a bucket we get that with probability $1 - O(1/d)$, x is more than $\Omega(1/d)$ away from each of the two parallel hyperplanes on either side. So with probability $(1 - O(1/d))^{O(d)} = \Omega(1)$ the entire sphere is contained inside the LSH bucket x maps to. □

Lemma C.3. *Using $O(k \log d)$ Euclidean-LSH functions, we get that every $x \in L_k$, there exists a bucket in at least one of the $O(k \log d)$ buckets x gets mapped to such that the entire k-dimensional sphere of radius $O(\epsilon/d)$ centered at x is contained within the bucket.*

Proof. We use a covering number argument. The maximum volume of a k-dimensional subspace within $[-1, 1]^d$ is $(2\sqrt{d})^k$. We cover this entire volume using spheres of radius ϵ/d. The total number of spheres required to do this are $O((2d\sqrt{d})^k/\epsilon^k)$. We now do a union bound over all the sphere centers in our cover above. For a single sphere, the probability that it does not go intact into a bucket in any of the $O(k \log d)$ LSH functions is $d^{-\Omega(k)}$. By a union bound we can bound the probability that there exists a sphere center that does not go intact into a bucket to be $d^{-\Omega(k)}$. Hence the Lemma statement holds with exceedingly large probability of $1 - d^{\Omega(k)}$. □

Now, we only include buckets with volume at least $(\Omega(\epsilon/(d\sqrt{k})))^k$. We can do this procedure using approximate support estimation algorithms (Valiant & Valiant, 2011). This takes time and sample complexity $S/\log S$ where S is the size of the support. With constant probability all points in L_k are mapped to some such high volume bucket in at least one of the LSH functions. The total number of buckets with this minimum volume is at most $(O((d^2\sqrt{k})/\epsilon))^k$, which is also an upper bound on the sample complexity and running time of the support estimation procedure. Now, we lift all the above results when we go to M_k from L_k. Since the Jacobian of the manifold map f has a constant condition number, its determinant is at most $\exp(k)$; so the volume of any region in L_k changes by at most an $\exp(\pm O(k))$ multiplicative factor when it goes to M_k. So all volume arguments in the previous proofs hold with multiplicative factors $\exp(\pm O(k))$. This concludes our proof. □

D. Lower Bound for Analytic Functions

The functions described in the lower bound presented earlier are continuous but not differentiable everywhere as they are piecewise linear functions. In Theorem D.1 we show that we can make the lower bound stronger by providing a construction of L-Lipschitz analytic functions (which are differentiable everywhere).

Theorem D.1. *A dense model of width w with a random bottom layer requires*

$$w = \Omega\left(\frac{2^{k^2/2}(LC_1)^k}{(\sqrt{k}\pi\epsilon)^k}\right),$$

where C_1 is a large enough constant, to learn L-Lipschitz analytic functions on $[-1, 1]^d$ to ℓ_∞ error ϵ when the inputs are sampled uniformly over a unknown k-dimensional subspace of $\mathbb{R}^d \cap [-1, 1]^d$. Moreover, the number of samples required to learn the above class of functions is

$$\Omega\left(w \log w\right),$$

where $w = \Omega\left(\frac{2^{k^2/2}(LC_1)^k}{(\sqrt{k}\pi\epsilon)^k}\right)$.

Proof of Theorem D.1. We construct a family $\mathcal{F}$ of analytic functions that are L-Lipschitz described using the Fourier basis functions. Each $f \in \mathcal{F}$ will be of the form

$$f(x) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} \cdots \sum_{n_k=0}^{\infty} a_{n_1 n_2 \ldots n_k} \exp\left(i\pi n^\top x\right),$$

for $x \in [-1,1]^k$. We pick a small value of $0 < \epsilon_1 < 1$. We assume $1/\epsilon_1$ is an integer for convenience. If it is not, we can simply take $\lceil 1/\epsilon_1 \rceil$ instead. For a set of integers $(n_1, n_2, \ldots, n_k) \in [1/\epsilon_1]^k$, let $\eta_{n_1 n_2 \ldots n_k} \in \{\pm\}$. We use η_n as a shorthand when it is not ambiguous. The family $\mathcal{F}$ is defined as the set of functions f below

$$f(x) = \sum_{n_1,\ldots,n_k=0}^{1/\epsilon_1} \eta_n \epsilon_1^\alpha \left(\exp(i\pi n^\top x) + \exp(i\pi n^\top x)\right), \tag{3}$$

where each η_n is chosen to be either $\pm L/(C\sqrt{k}\pi)$ for a large enough constant C and α will be determined later. There are $(1/\epsilon_1)^k$ Fourier bases in each f and the coefficient of each is set to be $\pm L\epsilon_1^\alpha/(C\sqrt{k}\pi)$. Hence we have

$$|\mathcal{F}| = 2^{\left((1/\epsilon_1)^k\right)}. \tag{4}$$

Next we argue that a larger than 0.9 fraction of the functions in $\mathcal{F}$ are L-Lipschitz. We have,

$$\begin{aligned}
\nabla f(x) &= \sum_{n_1,\ldots,n_k=0}^{1/\epsilon_1} \eta_n \epsilon_1^\alpha i\pi(\exp(i\pi n^\top x) - \exp(i\pi n^\top x))n \\
&= \sum_{n_1,\ldots,n_k=0}^{1/\epsilon_1} -2\eta_n \pi \sin(\pi n^\top x)\epsilon_1^\alpha n \qquad (5) \\
\implies \mathbb{E}\left[\nabla f(x)\right] &= 0, \qquad (6)
\end{aligned}$$

where the last expectation is over the uniform measure over functions in $\mathcal{F}$. To get a bound on $\|\nabla f(x)\|_2$ we bound each $(\nabla f(x))_i$ with high probability. Each $(\nabla f(x))_i$ is a sum of $(1/\epsilon_1)^k$ independent random variables, namely η_n. We saw above that $\mathbb{E}[(\nabla f(x))_i] = 0$. To bound $|(\nabla f(x))_i|$ with high probability we will use McDiarmid's inequality. An upper bound on how much the value of $(\nabla f(x))_i$ can change when any one η_n flips in value is computed as $4\epsilon_1^{(\alpha-1)} L/C\sqrt{k}$. Then, an application of McDiarmid's concentration inequality gives us that,

$$\begin{aligned}
\Pr\left[|(\nabla f(x))_i| > t\right] &\le 2\exp\left(\frac{-t^2 k C^2 \epsilon_1^{(k+2-2\alpha)}}{16L^2}\right), \\
\implies |(\nabla f(x))_i| &\le \frac{L}{\sqrt{k}\epsilon_1^{(k+2-2\alpha)/2}} \qquad (7)
\end{aligned}$$

with probability ≥ 0.9 for a large enough constant C. This implies that

$$\|\nabla f(x)\|_2 \le \frac{L}{\epsilon_1^{(k+2-2\alpha)/2}} \tag{8}$$

with probability ≥ 0.9 for a randomly sampled $f \in \mathcal{F}$. Now, let η_f denote the vector of η_n values in sequence for any f. Using McDiarmid's (or Hoeffdings) concentration bound again, we also get that, with probability ≥ 0.9, the Hamming distance between η_{f_1} and η_{f_2} for two f randomly sampled from $\mathcal{F}$ is at least $c(1/\epsilon_1)^k$ for a small enough constant $c < 1$. This implies that for randomly sampled f_1, f_2,

$$\begin{aligned}
&f_1(x) - f_2(x) \\
&= \sum_{n_1,\ldots,n_k=0}^{1/\epsilon_1} 2\eta'_n \epsilon_1^\alpha \left(\exp(i\pi n^\top x) + \exp(i\pi n^\top x)\right), \qquad (9)
\end{aligned}$$

where η'_n is non-zero for at least $c(1/\epsilon_1)^k$ of the terms from the above argument about the Hamming distance. Parseval's identity then implies that

$$\begin{aligned}
&\frac{1}{2^k}\int_{-1}^{1}\dots\int_{-1}^{1}(f_1(x)-f_2(x))^2dx_1\dots dx_k \\
&\geq 4L^2\epsilon_1^{2\alpha}c\frac{1}{\epsilon_1^k C^2 k\pi^2} \\
&\implies \|f_1-f_2\|_\infty \geq \frac{2^{(k/2+1)}L\sqrt{c}\epsilon_1^{(\alpha-k/2)}}{C\sqrt{k}\pi}. \qquad (10)
\end{aligned}$$

Finally we note that by union bound, at least a 0.8 fraction of the functions in $\mathcal{F}$ satisfy both our Lipschitzness property (8) and (10) simultaneously. Setting $\alpha = k/2+1$ and $\epsilon_1 = \frac{C\sqrt{k}\pi\epsilon}{L\sqrt{c}2^{k/2}}$ we get that to achieve a strictly smaller error than 2ϵ in the $\|.\|_\infty$ sense, one requires a dense model with a width of

$$\Omega\left(2^{k^2/2}\left(\frac{LC_1}{\sqrt{k}\pi\epsilon}\right)^k\right).$$

□

E. Conclusion

We provided the first systematic theoretical treatment of modern sparsely activated networks. To do so, we introduced the DSM model, which captures the sparsity in Mixture of Experts models, such as Switch and Scaling Transformers. We showed that DSM can simulate popular architectures as well as LSH-based networks. Then, we exhibited new constructions of sparse networks. Our use of LSH to build these networks offers a theoretical grounding for sparse networks. We complemented our theory with experiments, showing that sparse networks can approximate various functions.

For future work, it would be interesting to implement LSH-based networks in transformers for language/vision tasks. A related question is to determine the best way to interpolate in each LSH bucket (e.g., a higher degree polynomial may work better). Another question is whether a dense model is more powerful than a sparse model with the same number of total trainable parameters. Theorem 4.1 only says that a sparse model with similar number of parameters as a dense model can more efficiently (fewer FLOPs) represent Lipschitz functions. This does not say *all* functions expressible by a dense model are also expressible by a sparse model. This is non-trivial question as A^x depends on the input (i.e., $\mathsf{DSM}(d,t,t)$ may be more expressive than the dense model with width t). We expect that dense networks can be trained to perform at least as well as sparse networks, *assuming the width is large enough*. The dense networks should optimize the weights in the last layer to approximate the function, but they may not favor certain neurons depending on the input.