How Can We Be So Dense? The Robustness of Highly Sparse Representations

Subutai Ahmad [1] **Luiz Scheinkman** [1]

Abstract

Neural networks can be highly sensitive to noise and perturbations. In this paper we suggest that high dimensional sparse representations can lead to increased robustness to noise and interference. A key intuition we develop is that the ratio of the match volume around a sparse vector divided by the total representational space decreases exponentially with dimensionality, leading to highly robust matching with low interference from other patterns. We analyze efficient sparse networks containing both sparse weights and sparse activations. Simulations on MNIST, the Google Speech Command Dataset, and CIFAR-10 show that such networks demonstrate improved robustness to random noise compared to dense networks, while maintaining competitive accuracy. We propose that sparsity should be a core design constraint for creating highly robust networks.

1. Introduction

The literature on sparse representations dates back many decades, with neuroscience as a primary motivation (Kanerva, 1988; Olshausen & Field, 1997). More recent work has shown that hierarchical sparse representations can lead to natural looking hierarchical feature detectors (Lee et al., 2008; Chen et al., 2018), and that sparsity can sometimes lead to improved test set accuracies (Lee et al., 2009; Nair & Hinton, 2009; Srivastava et al., 2013; Rawlinson et al., 2018). Despite the above literature the majority of neural networks today rely on dense representations.

In this paper we propose that robustness to random perturbations is an inherent property of high dimensional sparse representations. We offer two main contributions. First, we analyze high dimensional sparse representations, and show that such representations are naturally more robust to noise and interference from random inputs. When matching sparse patterns, corrupted versions of a pattern are "close" to the original whereas random patterns are exponentially hard to match. Our second contribution is a simple sparse layer formulation that can be dropped into standard networks trained with backpropagation. We demonstrate improved robustness to noise while maintaining competitive accuracy in the standard zero noise scenario. At the end of the paper we discuss some possible areas for future work.

[1]Numenta, Redwood City, California, USA. Correspondence to: Subutai Ahmad <sahmad@numenta.com>.

Presented at the ICML 2019 Workshop on Uncertainty and Robustness in Deep Learning.

2. High Dimensional Sparse Representations

We develop properties of sparse representations as they relate to noise robustness and interference. In a typical neural network an input vector is matched against a weight vector using a dot product followed by a non-linearity such as $\tanh(\cdot)$ or $\text{ReLU}(\cdot)$. When comparing two sparse vectors via a dot product, the results are unaffected by the zero components of either vector. Ideally we would like the outputs of each layer to be invariant to random noise or perturbations. A key quantity therefore is the ratio of the matching volume around a candidate vector divided by the volume of the whole space. The larger the match volume around a vector, the more robust it is to noise. The smaller the ratio, the less likely it is that random inputs can affect the match.

We quantify the above ratio using binary vectors (following (Ahmad & Hawkins, 2016)) and show that the ratio decreases exponentially with increased dimensionality, while maintaining a large match volume. Let $\boldsymbol{x}$ be a binary vector of length n, and let $|\boldsymbol{x}|$ denote the number of non-zero entries. The dot product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ counts the overlap, or number of shared bits, between two such vectors. We would like to understand the probability of two vectors having significant overlap, i.e. overlap greater than some threshold θ.

We define the overlap set, $\Omega^n(\boldsymbol{x}_i, b, k)$, as the set of all vectors of size k that have exactly b bits of overlap with $\boldsymbol{x_i}$. The number of such vectors is given by:

$$|\Omega^n(\boldsymbol{x}_i, b, k)| = \binom{|\boldsymbol{x}_i|}{b}\binom{n - |\boldsymbol{x}_i|}{k - b} \tag{1}$$

If we select vectors from a uniform random distribution, the probability of significant overlap can be calculated as:

$$P(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta) = \frac{\sum_{b=\theta}^{|\boldsymbol{x}_i|} \mid \Omega^n(\boldsymbol{x}_i, b, |\boldsymbol{x}_j|) \mid}{\binom{n}{|\boldsymbol{x}_j|}} \quad (2)$$

where $\binom{n}{|\boldsymbol{x}_j|}$ is the set of all vectors (details in Supplementary section A.1).

We can generalize Eq. 2 to handle scalar representations. Binary and scalar vectors are similar in that the components containing zero do not affect the dot product, and thus the combinatorics in Eq. 2 are still applicable. Let $\boldsymbol{x}_w$ and $\boldsymbol{x}_i$ represent two sparse vectors such that $\|\boldsymbol{x}_w\|_0$ and $\|\boldsymbol{x}_i\|_0$ counts the number of non-zero entries in each. Let each non-zero component be independent and sampled from the distributions $P_{\theta_w}(x_w)$ and $P_{\theta_i}(x_i)$. The probability of a significant match is then:

$$P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta) = \frac{\sum_{b=\theta}^{\|\boldsymbol{x}_w\|_0} p_b \mid \Omega^n(\boldsymbol{x}_w, b, \|\boldsymbol{x}_i\|_0) \mid}{\binom{n}{\|\boldsymbol{x}_i\|_0}} \quad (3)$$

where p_b is the probability that the dot product is $>= \theta$ given that the overlap is exactly b components: $p_b = P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta \mid \|\boldsymbol{x}_w \cdot \boldsymbol{x}_i\|_0 = b)$

2.1. Impact of Dimensionality and Sparsity

Two key factors are the number of non-zero components, $|\boldsymbol{x}_i|$, and the dimensionality, n. Figure 1 provides an intuitive description of their impact in Eq. 2. Assume we have M prototype vectors, and we want to match noisy versions of these vectors. Around each prototype there is a set of matching vectors (small white circles in Figure 1A). If the threshold is very high, the set of matching vectors is small and there will be quite a bit of space between these sets. As you decrease θ matching is less strict and you can match noisier versions of each prototype. The cost is that the chance of matching the other vectors also increases because there is less free space in between (Figure 1B). It turns out that for sparse vectors, this cost is offset as you increase n. That is, as n increases, the denominator in Eq. 2 (and the corresponding free space) increases much faster than the numerator. For a fixed sparsity level, you can maintain highly tolerant matches without the cost of additional false positives simply by increasing the dimensionality.

Fig 2 illustrates these properties (details in A.3). The chart shows that for sparse vectors, match rates with random vectors drop rapidly as the underlying dimensionality increases. The probability of dense matches stays relatively high and unaffected by dimensionality, indicating that both sparseness and high dimensionality are key to robust matches (Ahmad & Hawkins, 2016).

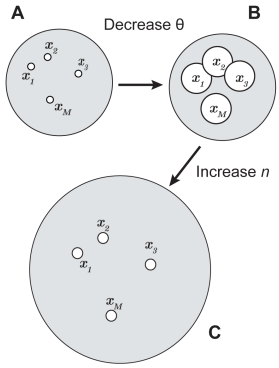

Figure 1. An illustration of the conceptual effect of decreasing the match threshold θ and increasing n, the dimensionality (see text.)

3. A Simple Differentiable Sparse Layer

We extend our previous work (Hawkins et al., 2011; Cui et al., 2017) and create a drop-in sparse layer for neural networks trained with back-propagation. Consider a network with L hidden layers. Let $\boldsymbol{y}^l$ denote the vector of outputs from layer l, respectively, with $\boldsymbol{y}^0$ as the input vector. $\boldsymbol{W}^l$ and $\boldsymbol{u}^l$ are the weights and biases for each layer. The feed forward outputs are then calculated as follows:

$$\hat{\boldsymbol{y}}^l = \boldsymbol{W}^l \cdot \boldsymbol{y}^{l-1} + \boldsymbol{u}^l$$
$$\boldsymbol{y}^l = f(\hat{\boldsymbol{y}}^l)$$

where f is an activation function such as $\text{ReLU}(\cdot)$.

We make two modifications to this basic formulation. First, we initialize the weights using a sparse random distribution, such that only a fraction of the weights contain non-zero values. The rest of the connections are treated as non-existent, i.e. the corresponding weights are zero throughout the life of the network. Second, only the top-k active units within each layer are maintained in $\boldsymbol{y}^l$, and the rest set to zero. This k-winners step can be thought of as a substitute for the ReLU function, with an adaptive threshold corresponding to the k'th largest activation (Makhzani & Frey, 2013).

One practical issue is that a small number of units can initially dominate and then, through learning, become active for a large percentage of patterns (also noted in (Makhzani & Frey, 2015; Cui et al., 2017)). Eq. 2 suggests we maximize the duty cycle of units in order to maximize the representational space. To address this we employ a boosting term which favors recently inactive units (see A.4.1 for details).

It is straightforward to extend the above to sparse convolutional layers (Section A.5). Overall this formulation leads to a simple efficient layer can be dropped into any standard network and trained using standard gradient descent. The implementation is closely related to the k-winner take all

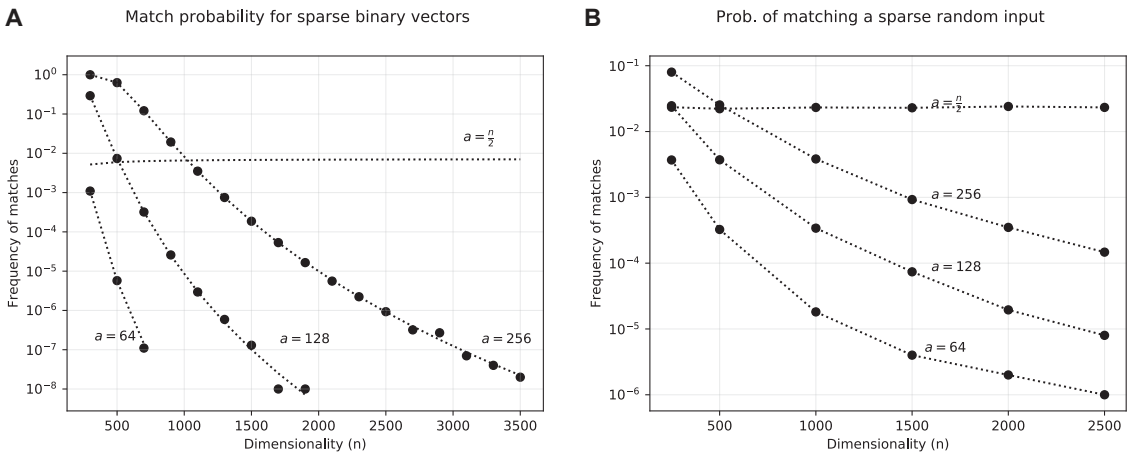

Figure 2. **A**: The probability of matches to random binary vectors (with a active bits) decreases exponentially with n. Dotted lines denote the theoretically predicted probabilities using Eq. 2. **B**: Sparse scalar vectors (with a non-zero components) behave similarly. The probability for a dense vector, $a = \frac{n}{2}$ stays relatively high, and does not decrease with dimensionality.

NETWORK	TEST SCORE	NOISE SCORE
DENSE CNN-1	99.14 ± 0.03	$74{,}569 \pm 3{,}200$
DENSE CNN-2	99.31 ± 0.06	$97{,}040 \pm 2{,}853$
SPARSE CNN-1	98.41 ± 0.08	$100{,}306 \pm 1{,}735$
SPARSE CNN-2	99.09 ± 0.05	$103{,}764 \pm 1{,}125$
DENSE CNN-2 SP3	99.13 ± 0.07	$100{,}318 \pm 2{,}762$
SPARSE CNN-2 D3	98.89 ± 0.13	$102{,}328 \pm 1{,}720$
SPARSE CNN-2 W1	98.2 ± 0.19	$100{,}322 \pm 2{,}082$
SPARSE CNN-2 DSW	98.92 ± 0.09	$70{,}566 \pm 2{,}857$

Table 1. Classification accuracies and total noise scores for MNIST. CNN-1 and CNN-2 = 1 or 2 convolutional layers.

networks in (Majani et al., 1989) and fixed sparsity networks in (Makhzani & Frey, 2015).

4. Simulation Results

4.1. MNIST

We tested both dense and sparse networks on MNIST, with added random noise (see Figure 3 (A), details in A.6). State of the art accuracies on MNIST sans augmentation are in the range $98.3 - 99\%$ respectively. Our accuracies (Table 1) are in the same range for both sparse and dense networks. We also computed an overall **noise score** which counted the total number of correct classifications across all noise levels. (Table 4 lists the key parameters.)

In Table 1, networks in the top section of the table (Dense CNN-1 and Dense CNN-2) are composed of standard dense convolutional and hidden layers. Networks in the middle section (Sparse CNN-1 and Sparse CNN-2) are composed of sparse convolutional and sparse hidden layers. Networks in the last section demonstrate the effects of ablation studies and contain a mixture of dense and sparse layers (see Section A.6). Overall the architectures with sparse layers performed significantly better on the noise score than the fully dense networks. Sparse CNN-2, the two layer completely sparse network, had the best noise score. The two fully dense networks performed substantially worse than the others on noise, even though their test accuracies were comparable. Figure 3 plots the accuracy of fully dense and sparse networks at different noise levels.

4.2. Google Speech Commands Dataset

In order to test sparsity on a domain other than vision, we used the Google Speech Commands dataset (GSC). GSC was made available in 2017 (Warden, 2017) and consists of 65,000 one-second long utterances of 30 keywords spoken by thousands of individuals. Standard convolutional nets using ten categories achieve accuracies in the range $91 - 92\%$ (Sainath & Parada, 2015; Tang & Lin, 2017), whereas ResNets (He et al., 2015a;b) lead to accuracies in the range of $95 - 96\%$ (Tang & Lin, 2017).

We trained dense and sparse versions of standard convolutional networks and achieved test set accuracies in the range of $96.5 - 97.2\%$ for 10 categories (see Supplementary Section A.7 for training details and parameters). Table 2 (left column) shows mean accuracy on the test set. Both dense and sparse networks had about the same test set accuracy.

We created noisy versions of the test set by blending in random noise. An overall noise score accumulates the correct classifications across all noise levels. Table 2 shows that sparse networks performed significantly better than the best dense network. Overall these results demonstrate that the robustness of sparse networks can apply across domains.

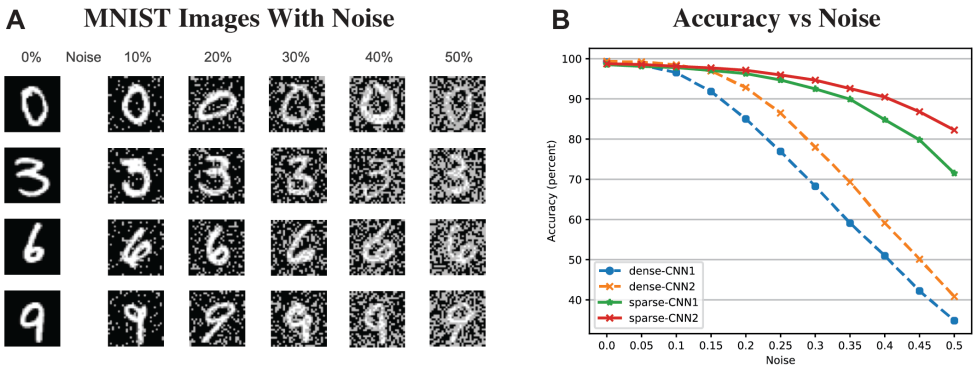

Figure 3. **A**. Example MNIST images with varying levels of noise. **B**. Classification accuracy as a function of noise level.

NETWORK	TEST SCORE	NOISE SCORE
DENSE CNN-2 (DR=0.0)	96.37± 0.37	8,730± 471
DENSE CNN-2 (DR=0.5)	95.69± 0.48	7,681± 368
SPARSE CNN-2	96.65± 0.21	11,233± 1013
SUPER-SPARSE CNN-2	96.57± 0.16	10,752± 942

Table 2. Scores for Google Speech Commands, averaged over 10 random seeds, ± one standard deviation. Dr = dropout levels.

4.3. CIFAR-10

There are a large number of different architectures that have been successfully applied to CIFAR-10, and many of these have scaled to larger datasets, such as ImageNet. Here we used CIFAR-10 to gauge the applicability of sparse representations to deeper networks, focusing on variants of DenseNet (Huang et al., 2016), VGG (Simonyan & Zisserman, 2014), and MobileNet (Howard, 2017).

Initial results on DenseNet and VGG are promising. Table 3 shows the results for various noise levels. For DenseNet, we applied our sparse convolutional layers to the transition blocks only, creating a "NotSoDenseNet". These had comparable accuracy with zero noise but demonstrated improved noise robustness as shown in Table 3. Our results on VGG networks were similar. Although the raw accuracy of VGG19-Sparse was slightly lower than its dense counterpart, its accuracy for even slightly noisy images were higher. With 10% of the pixels corrupted, accuracy of the sparse network was 12% higher than its dense counterpart.

We did not achieve improved results with MobileNet, possibly due to its factored convolutions. Level skipping is an interesting challenge. It is unclear how to introduce sparsity into ResNet architectures (He et al., 2015a) where level skipping connections have an additive effect on activity. We also did not apply sparsity to level skipping blocks in DenseNet. Applying sparsity to level skipping connections remains a topic for future study.

NOISE	DENSENET	NSDN	VGG19D	VGG19S
0.0%	92.80	93.09	93.24	92.10
2.5%	86.34	87.50	85.07	86.21
5.0%	77.19	79.10	75.88	79.00
7.5%	66.22	69.52	63.60	71.34
10.0%	55.10	61.13	52.41	64.18
12.5%	45.79	52.10	42.25	56.49
15.0%	38.67	45.25	35.25	50.86
17.5%	33.03	39.60	29.37	45.00

Table 3. Scores for CIFAR-10 as a function of noise. NSDN, VGG19D, and VGG19S stand for "NotSoDenseNet", VGG19-Dense and VGG19-Sparse, respectively.

5. Discussion

In this paper we proposed that sparse representations can be beneficial to the underlying robustness of networks. We developed intuitions and theory for the structure of vector matching in the context of binary sparse representations. We then constructed efficient neural network formulations of sparse networks and showed that this formulation seems to increase the overall robustness of the system to noisy inputs. A consistent finding in all our experiments was that raw test score was not a predictor of noise robustness, suggesting that focusing on pure test set accuracy alone is not sufficient for gauging performance under adverse conditions. Both dense and sparse networks showed high accuracies, but the sparse nets were significantly more robust.

We stress that the work here is still preliminary. In this paper we primarily focused on one perturbation type, random jitter to inputs, as a test of robustness. There are other types of perturbations that may be important to consider (Hendrycks & Dietterich, 2018). In addition there is still work remaining to demonstrate how to best apply sparsity to other network structures such as MobileNet, ResNet, and level skipping connections. Nevertheless, our results suggest that sparsity does not hurt, and that sparse networks should be given serious consideration in the overall design of robust systems.

Acknowledgements

We thank Jeff Hawkins, Ali Rahimi, Lucas Souza, and John Berkowitz for helpful discussions and comments.

References

Ahmad, S. and Hawkins, J. How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. *arXiv*, pp. arXiv:1601.00720 [q–bio.NC], jan 2016. URL https://arxiv.org/abs/1601.00720.

Chen, Y., Paiton, D., and Olshausen, B. The Sparse Manifold Transform. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10533–10544. Curran Associates, Inc., 2018.

Cui, Y., Ahmad, S., and Hawkins, J. The HTM Spatial Pooler a neocortical algorithm for online sparse distributed coding. *Frontiers in Computational Neuroscience*, 11:111, 2017. ISSN 1662-5188. doi: 10.3389/FNCOM.2017.00111. URL https://www.frontiersin.org/articles/10.3389/fncom.2017.00111/abstract.

Hawkins, J., Ahmad, S., and Dubinsky, D. Cortical Learning Algorithm and Hierarchical Temporal Memory, 2011. URL http://numenta.org/resources/HTM{_}CorticalLearningAlgorithms.pdf.

He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *http://arxiv.org/abs/1502.01852*, feb 2015a. URL http://arxiv.org/abs/1502.01852.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. *http://arxiv.org/abs/1512.03385*, dec 2015b. URL http://arxiv.org/abs/1512.03385.

Hendrycks, D. and Dietterich, T. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *http://arxiv.org/abs/1807.01697*, jul 2018. URL http://arxiv.org/abs/1807.01697.

Howard, M. W. Memory as perception of the past: Compressed time in mind and brain. *Trends in Cognitive Sciences*, 22(2):124–136, 2017. ISSN 13646613. doi: 10.1016/j.tics.2017.11.004. URL http://dx.doi.org/10.1016/j.tics.2017.11.004.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely Connected Convolutional Networks. *http://arxiv.org/abs/1608.06993*, aug 2016. URL http://arxiv.org/abs/1608.06993.

Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *http://arxiv.org/abs/1502.03167*, feb 2015. URL http://arxiv.org/abs/1502.03167.

Kanerva, P. *Sparse Distributed Memory*. The MIT Press, Cambridge, MA, 1988.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation Applied to Handwritten Zip Code Recognition, 1989. ISSN 0899-7667.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. ISSN 00189219. doi: 10.1109/5.726791.

Lee, H., Ekanadham, C., and Ng, A. Y. Sparse deep belief net model for visual area V2. *Advances In Neural Information Processing Systems*, 2008.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pp. 1–8, 2009. doi: 10.1145/1553374.1553453.

Majani, E., Erlanson, R., and Abu-Mostafa, Y. S. On the k-winners-take-all network. In *Advances in neural information processing systems*, pp. 634–642, 1989.

Makhzani, A. and Frey, B. k-Sparse Autoencoders. *http://arxiv.org/abs/1312.5663*, dec 2013. URL http://arxiv.org/abs/1312.5663.

Makhzani, A. and Frey, B. Winner-take-all autoencoders. *Advances in Neural Information Processing*, 2015. URL http://papers.nips.cc/paper/5783-winner-take-all-autoencoders.

Nair, V. and Hinton, G. E. 3D Object Recognition with Deep Belief Nets. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 22*, pp. 1339–1347. Curran Associates, Inc., 2009.

Olshausen, B. A. and Field, D. J. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37:3311–3325, 1997. ISSN 00426989. doi: 10.1016/S0042-6989(97)00169-7.

Rawlinson, D., Ahmed, A., and Kowadlo, G. Sparse Unsupervised Capsules Generalize Better. *http://arxiv.org/abs/1804.06094*, apr 2018. URL http://arxiv.org/abs/1804.06094.

Sainath, T. N. and Parada, C. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *http://arxiv.org/abs/1409.1556*, sep 2014. URL http://arxiv.org/abs/1409.1556.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. Compete to Compute. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2310–2318. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5059-compete-to-compute.pdf.

Tang, R. and Lin, J. Deep Residual Learning for Small-Footprint Keyword Spotting. *https://arxiv.org/abs/1710.10361*, oct 2017. URL https://arxiv.org/abs/1710.10361.

Tuguldur, E.-O. pytorch-speech-commands, 2018. URL https://github.com/tugstugi/pytorch-speech-commands.

Warden, P. Speech Commands: A public dataset for single-word speech recognition. *Dataset available*, 2017. URL http://download.tensorflow.org/data/speech{_}commands{_}v0.01.tar.gz.

A. Supplementary Material

These supplementary sections complement the main text, and provide additional detail as referenced above.

A.1. Mathematical Details

We define the overlap set, $\Omega^n(\boldsymbol{x}_i, b, k)$, as the set of all vectors of size k that have exactly b bits of overlap with $\boldsymbol{x_i}$. The number of such vectors is given by:

$$|\Omega^n(\boldsymbol{x}_i, b, k)| = \binom{|\boldsymbol{x}_i|}{b}\binom{n - |\boldsymbol{x}_i|}{k - b} \tag{4}$$

The left half of the above product counts all the ways we can select exactly b bits out of active bits in $|\boldsymbol{x}_i|$. The right half counts the number of ways we can select the remaining $k - b$ bits from the components of $\boldsymbol{x}_i$ that are zero. The product of these two quantities represents the number of all vectors with exactly b bits of overlap with $|\boldsymbol{x}_i|$. We can now count the number of vectors that match $\boldsymbol{x}_i$, i.e. where $\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta$ as:

$$\sum_{b=\theta}^{|\boldsymbol{x}_i|} \mid \Omega^n(\boldsymbol{x}_i, b, |\boldsymbol{x}_j|) \mid \tag{5}$$

If we select vectors from a uniform random distribution, the probability of significant overlap can be calculated as:

$$P(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta) = \frac{\sum_{b=\theta}^{|\boldsymbol{x}_i|} \mid \Omega^n(\boldsymbol{x}_i, b, |\boldsymbol{x}_j|) \mid}{\binom{n}{|\boldsymbol{x}_j|}} \tag{6}$$

where $\binom{n}{|\boldsymbol{x}_j|}$ is the set of all possible comparison vectors.

A.2. Impact of Dimensionality and Sparsity

As the underlying dimensionality n increases, the denominator in Eq. 6 (and the corresponding "free" space) increases much faster than the numerator. For a fixed sparsity level, you can maintain highly tolerant matches without the cost of additional false positives simply by increasing the dimensionality. Fig 2 **A** illustrates this trend for some example sparsities. In this figure we simulated matching with random vectors and plotted match rates with random vectors as a function of the number of active bits and the underlying dimensionality. In the simulation we repeatedly generated a random prototype vector with $|\boldsymbol{x}_i| = 24$ bits on and then attempted to match against random test vectors with a bits on. We matched using a threshold θ of 12 which meant that even vectors that were up to 50% different from $\boldsymbol{x}_i$ would match. We varied a and the dimensionality of the vectors, n. The dotted line shows the probabilities predicted by the equation, whereas the black dots are the values obtained through simulation.

The chart shows that for sparse binary vectors, match rates with random vectors drop rapidly as the underlying dimensionality increases. The horizontal line indicates the probability of matching $\boldsymbol{x}_i$ against dense vectors, with $a = n/2$. The probability of dense matches stays relatively high and unaffected by dimensionality, indicating that both sparseness and high dimensionality are key to robust matches. In (Ahmad & Hawkins, 2016) we develop additional properties, including the probability of false negatives.

A.3. Matching Sparse Scalar Vectors

Binary and scalar vectors are similar in that the components containing zero do not affect the dot product, and thus the combinatorics in Eq. 2 are still applicable. Eq. 1 represents

the set of scalar vectors where the number of non-zero multiplies in the dot product is exactly b, and Eq. 2 represents the probability that the number of non-zero multiplies is $>= \theta$. However, an additional factor is the distribution of scalar values. If components in one vector are extremely large relative to θ, the likelihood of a significant match will be high even with a single shared non-zero component.

We wanted to see if the exponential drop in random matches for binary vectors, demonstrated by Figure 2 (**A**) can be obtained using scalar vectors, and if so, the conditions under which they hold. Let $\boldsymbol{x}_w$ and $\boldsymbol{x}_i$ represent two sparse vectors such that $\|\boldsymbol{x}_w\|_0$ and $\|\boldsymbol{x}_i\|_0$ counts the number of non-zero entries in each. Let each non-zero component be independent and sampled from the distributions $P_{\theta_w}(x_w)$ and $P_{\theta_i}(x_i)$. The probability of a significant match is then:

$$P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta) = \frac{\sum_{b=\theta}^{\|\boldsymbol{x}_w\|_0} p_b \mid \Omega^n(\boldsymbol{x}_w, b, \|\boldsymbol{x}_i\|_0) \mid}{\binom{n}{\|\boldsymbol{x}_i\|_0}} \quad (7)$$

where p_b is the probability that the dot product is $>= \theta$ given that the overlap is exactly b components:

$$p_b = P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta \mid \|\boldsymbol{x}_w \cdot \boldsymbol{x}_i\|_0 = b) \quad (8)$$

There does not appear to be a closed form way to compute p_b for normal or uniform distributions so in Figure 2 (**B**) we show simulation results only. As before, we generated a large number of random vectors $\boldsymbol{x}_w$ and $\boldsymbol{x}_i$, and plotted the frequency of random matches. With $\|\boldsymbol{x}_w\|_0 = k$, we focus on simulations where the non-zero entries in $\boldsymbol{x}_w$ are uniform in $[-1/k, 1/k]$, and the non-zero entries in $\boldsymbol{x}_i$ are uniform in $S * [0, 2/k]$. We focus on this formulation because of the relationship to common network structures and weight initialization. $\boldsymbol{x}_w$ is a putative weight vector and $\boldsymbol{x}_i$ is an input vector to this layer from the previous layer (we assume unit activations are positive, the result of a ReLU-like nonlinearity). S controls the scale of $\boldsymbol{x}_i$ relative to $\boldsymbol{x}_w$.

Figure 2 (**B**) shows the behavior with $k = 32$ and $S = 1$. We varied the activity of the input vectors $\|\boldsymbol{x}_i\|_0 = a$ and the dimensionality of the vectors, n. We set $\theta = E[\boldsymbol{x}_w \cdot \boldsymbol{x}_w]/2.0$. The chart demonstrates that under these conditions we can achieve robust behavior similar to that of binary vectors. Figure 4 plots the effect of S on the match probabilities with a fixed $n = 1000$. As this chart shows, the error increases significantly as S increases. Taken together, these results show that the fundamental robustness properties of binary sparse vectors can also hold for sparse scalar vectors, as long as the overall scaling of vectors are in a similar range.

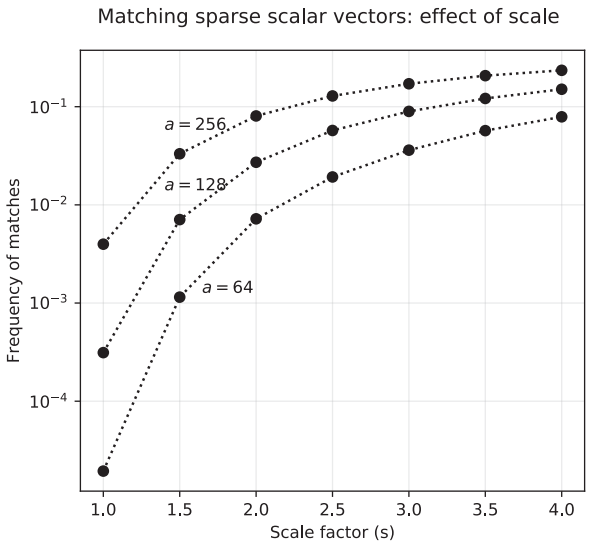

Figure 4. The impact of scale on vector matches with a fixed $n = 1000$. The larger the scaling discrepancy, the higher the probability of a false match.

A.3.1. Non-uniform Distribution of Vectors

Eq. 6 assumes the ideal case where vectors are chosen with a uniform random distribution. With a non-uniform distribution the error rates will be higher. The more non-uniform the distribution the worse the error rates. For example, if you mostly end up observing 10 inputs, your error rates will be bounded at around 10%. Thus, to optimize error rates, it is important to be as close to a uniform distribution as possible.

A.4. Sparse Layer Details

The network formalism for our sparse layer is an extension of our previous work on the HTM Spatial Pooler, a sparse coding algorithm that models sparse representations in the neocortex (Hawkins et al., 2011; Cui et al., 2017). Specifically, we formulated a version that is a drop-in layer for neural networks trained with back-propagation. The implementation is also closely related to the k-winner take all networks in (Majani et al., 1989) and fixed sparsity networks in (Makhzani & Frey, 2015).

The k-winners step can be thought of as a substitute for the ReLU function, with an adaptive threshold corresponding to the k'th largest activation (Makhzani & Frey, 2013). Similar to ReLU, the gradient of the layer is calculated as 1 above the threshold and 0 elsewhere. Non-zero weights are initialized using standard Kaiming initialization (He et al., 2015a). During inference we increase k by 50%, which led to slightly better accuracies. In all our simulations the last layer of each network is a standard linear output layer with log-softmax activation function.

Algorithm 1 k-winners layer

1: $\hat{\boldsymbol{y}}^l = \boldsymbol{w}^l \cdot \boldsymbol{y}^{(l-1)} + \boldsymbol{u}^l$
2: $b_i^l(t) = e^{\beta(\hat{a}^l - d_i^l(t))}$
3: topIndices$^l = topk(\boldsymbol{b}^l \odot \hat{\boldsymbol{y}}^l)$
4: $\boldsymbol{y}^l = 0$
5: $\boldsymbol{y}^l[\text{topIndices}^l] = \hat{\boldsymbol{y}}^l$
6: $d_i^l(t) = (1-\alpha)d_i^l(t-1) + \alpha \cdot [y_i^l(t) \in \text{topIndices}^l]$

A.4.1. BOOSTING

One practical issue with the above formulation is that it is possible for a small number of units to initially dominate and then, through learning, become active for a large percentage of patterns (this was also noted in (Makhzani & Frey, 2015; Cui et al., 2017)). Having a small number of active units negatively impacts the available representational volume. It is desirable for every unit to be equally active in order to maximize the robustness of the representation in Eq. 2. To address this we employ a boosting term (Hawkins et al., 2011; Cui et al., 2017) which favors units that have not been active recently. We compute a running average of each unit's duty cycle (i.e. how frequently it has been one of the top k units):

$$d_i^l(t) = (1-\alpha)d_i^l(t-1) + \alpha \cdot [i \in \text{topIndices}^l] \quad (9)$$

A boost coefficient b_i^l is then calculated for each unit based on the target duty cycle and the current average duty cycle:

$$b_i^l(t) = e^{\beta(\hat{a}^l - d_i^l(t))} \quad (10)$$

The target duty cycle $\hat{a}^l$ is a constant reflecting the percentage of units that are expected to be active, i.e. $\hat{a}^l = \frac{k}{|\boldsymbol{y}^l|}$. The boost factor, β, is a positive parameter that controls the strength of boosting. $\beta = 0$ implies no boosting ($b_i^l = 1$), and higher numbers lead to larger boost coefficients. In (Hawkins et al., 2011; Cui et al., 2017) we showed that Eq. 10 encourages each unit to have equal activation frequency and effectively maximizes the entropy of the layer.

The boost coefficients are used during the k-winners step to select which units remain active for this input. Through boosting, units which have not been active recently have a disproportionately higher impact and are more likely to win, whereas overly active units are de-emphasized. To determine the output of the layer, the non-boosted activity of each winning unit is kept and the remaining units are set to zero. The duty cycle is then updated. A pseudo-code description for the k-winners layer is in the Supplementary section (Algorithm 1). In our simulations we used $\beta = 1.0$ or 1.5 for all sparse simulations.

A.5. Sparse Convolutional Layers

We can apply the above algorithm to convolutional networks (CNNs) (LeCun et al., 1989). A canonical CNN layer uses a linear convolutional layer containing a number of filters, followed by a max-pooling (downsampling) layer, followed by ReLU. In order to implement sparse CNN layers, the k-winners layer is applied to the output of the max-pooling layer instead of ReLU (just as in our non-convolutional layers). However, since each filter in a CNN shares weights across the image, duty cycles are accumulated per filter. In our simulations dense and sparse CNN nets both have a hidden layer (which is dense or sparse, respectively) after the last convolutional layer, followed by a linear plus softmax layer. We used $5X5$ filters throughout with a stride of 1.

A.6. MNIST Training Details

For MNIST (LeCun et al., 1998) each network consisted of one or two convolutional layers, followed by a hidden layer, followed by a linear + softmax output layer. Sparse nets consisted of sparse convolutional layers followed by a sparse hidden layer.

Networks were trained using standard stochastic gradient descent to minimize cross entropy loss. We used starting learning rates in the range $0.01 - 0.04$, and the learning rate was decreased by a factor between 0.5 and 0.9 after each epoch. We also tried batch normalization (Ioffe & Szegedy, 2015) and found it did not help for MNIST (it did help significantly for Google Speech Commands results - see below). For sparse networks, we used a small mini-batch size (around 4), for the first epoch only, in order to let duty cycle calculations update frequently and settle. Hyperparameters such as the learning rate and network size were chosen using a validation set consisting of $10,000$ randomly chosen training samples. We then report final results on the test set using networks trained on the full training set.

For each test image we randomly set $\eta\%$ of the pixels to a constant value near white (the constant value was two standard deviations over the mean pixel intensity). Figure 3 (A) shows sample images for different noise levels. We generated 11 different noise levels with η ranging between 0 and 0.5 in increments of 0.05. We also computed an overall **noise score** which counted the total number of correct classifications across all noise levels. Results shown in Table 1 are averaged over 10 random seeds, $\pm$ one standard deviation.

Ablation studies: In order to judge the relative contributions of sparse layers we ran experiments where we replaced various sparse components with their dense counterparts, i.e. dense CNNs with sparse hidden layers, and vice versa. Dense CNN-2 SP3 contained two dense CNN layers fol-

lowed by the sparse third layer from Sparse CNN-2. Sparse CNN-2 D3 contained the same CNN layers as Sparse CNN-2 followed by the dense third layer from Dense CNN-2. Sparse CNN-2 W1 was identical to Sparse CNN-2 except that the weight sparsity was 1 (i.e. fully dense weights). Sparse CNN-2 DSW contained a third layer with dense outputs, but with a weight sparsity of 0.3%.

The results of these networks are shown in the bottom third of Table 1. From a noise robustness perspective, most of the variants (except for Sparse CNN-2 DSW) performed well, better than the best pure dense network. This supports the idea that sparsity in many forms may be helpful with robustness. It is interesting to note that the standard deviation of the noise score in these variants was also higher than that of the pure sparse networks. Overall the results with mixed networks were encouraging, and suggest a clear benefit to introducing sparsity at any level.

Impact of Dropout: The above results did not use dropout (Srivastava et al., 2014), which is generally thought to improve robustness. We found that dropout did occasionally improve the robustness of dense networks, but any improvements were modest and the dropout percentage had to be tuned carefully. For sparse nets dropout consistently reduced accuracies. Even with the optimal dropout percentage, the noise scores of dense networks were significantly lower than sparse nets.

A.7. Google Speech Commands Training Details

The GSC dataset contains predefined training, validation, and test sets. Reference convolutional nets using ten of the keyword categories (plus artificial "silence" and "unknown" categories created during training augmentation) achieve accuracies in the range $91 - 92\%$ (Sainath & Parada, 2015; Tang & Lin, 2017). In (Tang & Lin, 2017) they demonstrated improved accuracies in the range of $95 - 96\%$ using residual networks (ResNets (He et al., 2015a;b)).

A Kaggle competition using GSC (also limited to 10 categories) took place between November 2017 and early 2018[1]. For our simulations we use the preprocessing code provided by one of the top-10 contestants (Tuguldur, 2018) who achieved around $97 - 97.5\%$ accuracies using variants of ResNet and VGG (Simonyan & Zisserman, 2014) architectures. Following this implementation, audio samples in our simulations are converted to 32-band Mel spectograms before being fed to the network. During training we augment the data by randomly adjusting the amplitude, speed, and pitch of each training sample, and by randomly shifting and stretching samples in the frequency domain. No data augmentation is performed on the validation or test sets.

[1] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

We trained dense and sparse convolutional networks, with hyperparameters chosen based on the validation set. We were able to achieve reasonable accuracies using two convolutional layers, followed by a hidden layer and then a linear + softmax output layer. Our sparse networks had sparse convolutional layers as well as a sparse hidden layer. Unlike MNIST we found that batch normalization (Ioffe & Szegedy, 2015) accelerated learning significantly, and we used it for every layer.

We included a "Super-Sparse CNN-2" with a significantly sparser hidden layer. The hidden layer for this network had 10% weight sparsity, and a lower output sparsity (Table 4). This network had slightly lower noise score, but its score was still significantly higher than that of the dense networks.

Using the above setup we were able to achieve test set accuracies in the range of $96.5 - 97.2\%$ classifying the ten categories corresponding to the digits "zero" through "nine". Table 2 (left column) shows mean accuracy on the test set. Both dense and sparse networks had about the same accuracy. Dropout had a negative effect on the accuracy. Table 4 lists the key parameters in each network.

Generating Noisy Test Patterns: For each test audio sample $\boldsymbol{A}$ we generated a random white noise sample and blended them together:

$$\boldsymbol{A}^* = (1 - \eta)\boldsymbol{A} + \eta\text{whiteNoise}$$

We generated 11 different noise levels, with η ranging from 0 to 0.5 in increments of 0.05. Our overall noise score counted the total number of classifications across all noise levels.

A.8. Key Parameters

Table 4 lists the key sparsity related parameters used in the MNIST and GSC networks.

A.9. CIFAR-10 Training Details

For CIFAR-10 we focused on DenseNet (Huang et al., 2016), VGG (Simonyan & Zisserman, 2014), and MobileNet (Howard, 2017). We generated images with varying levels of noise by corrupting individual channels on randomly chosen pixels. For each test image we randomly choose $\eta\%$ of the pixels to corrupt. Within each pixel we randomly chose one of the three channels and set it to a constant value that is either plus or minus two standard deviations from the mean pixel intensity. η ranged between 0 and 0.175 in increments of 0.025.

Network implementations were based on https://github.com/bearpaw/pytorch-classification. For DenseNet we

NETWORK	L1 F	L1 SPARSITY	L2 F	L2 SPARSITY	L3 N	L3 SPARSITY	WT SPARSITY
MNIST							
DENSE CNN-1	30	100%			1000	100%	100%
DENSE CNN-2	30	100%	30	100%	1000	100%	100%
SPARSE CNN-1	30	9.3%			150	33.3%	30%
SPARSE CNN-2	32	8.7%	64	29.3 %	700	14.3%	30%
DENSE CNN-2 SP3	30	100%	30	100%	700	14.3%	30%
SPARSE CNN-2 D3	32	8.7%	64	29.3 %	1000	100%	100%
SPARSE CNN-2 W1	32	8.7%	64	29.3 %	700	14.3%	100%
SPARSE CNN-2 DSW	32	8.7%	64	29.3 %	1000	100%	30%
GSC							
DENSE CNN-2	64	100%	64	100%	1000	100%	100%
SPARSE CNN-2	64	9.5%	64	12.5%	1000	10%	40%
SUPER SPARSE CNN-2	64	9.5%	64	12.5%	1500	6.7%	10%

Table 4. Key parameters for each network. L1F and L2F denote the number of filters at the corresponding CNN layer. L1,2,3 sparsity indicates k/n, the percentage of outputs that were enforced to be non-zero. 100% indicates a special case where we defaulted to traditional ReLU activations. Wt sparsity indicates the percentage of weights that were non-zero. All parameters are available in the source code.

used a growth rate of 12. We used four blocks with sizes "[2, 4, 8, 4]". In addition, for our sparse NotSoDenseNet we substituted our k-winner layer in place of the ReLU used in the Transition blocks, with a sparsity level of 20%.

For VGG we used the standard VGG19 with BatchNorm in the "cifar" directory of the above repository. All learning parameters, including learning rate schedule, were identical to the values in the repository. For VGG19-Sparse we used a sparsity level of 25% for all layers. Weight sparsity was 50% for all layers except those in the first block, which were 100%.

A.10. Software

All code and experiments are available at `https://github.com/numenta/nupic.torch` and `https://github.com/numenta/nupic.research` as open source.