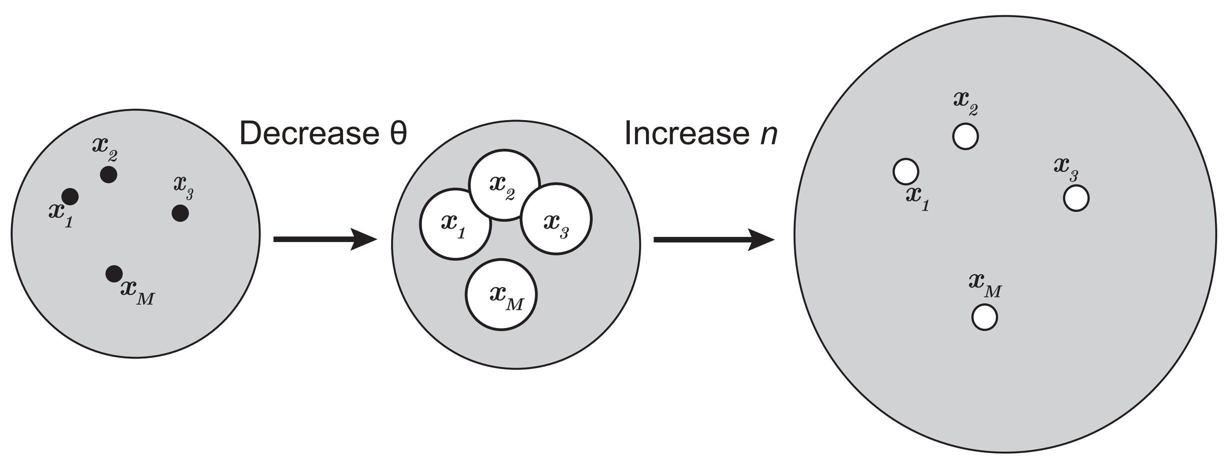
How Can We Be So Dense?
The Robustness of Highly Sparse Representations

Subutai Ahmad and Luiz Scheinkman
sahmad@numenta.com, lscheinkman@numenta.com

Key takeaways

1. Sparse representations are inherently robust.
Consider a match between two *n*-dim vectors via dot product:

$$P(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta)$$



For sparse vectors, false matches decrease exponentially as you increase n. See this panel for details.

2. We create a simple differentiable sparse layer that exploits these properties. This formulation can be dropped in to almost any network. See panel below for details.

3. Tests show that sparse networks have the same accuracy as their dense counterparts, but are consistently more robust to random noise. Tested with MNIST, Google Speech Commands, and CIFAR-10 on a variety of network architectures.

We propose that sparsity should be a key design principle for robustness.

Combinatorics of sparse representations

We want each layer to be invariant when matching corrupted inputs. When comparing two sparse vectors via a dot product, the results are unaffected by the zero components of either vector. A key measure is the ratio of all matching vectors divided by the volume of the whole space. The larger the number of matching vectors, the more robust it is to noise. The smaller the ratio, the less likely it is that other inputs can lead to false positives.

Binary sparse representations: Let $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ be binary vectors of length n.

The number of vectors of length *k* that exactly match *b* components of $\boldsymbol{x}_i$ is:

Number of ways to select exactly *b* non-zero components of $\boldsymbol{x}_i$

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

Number of ways to select remaining bits

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

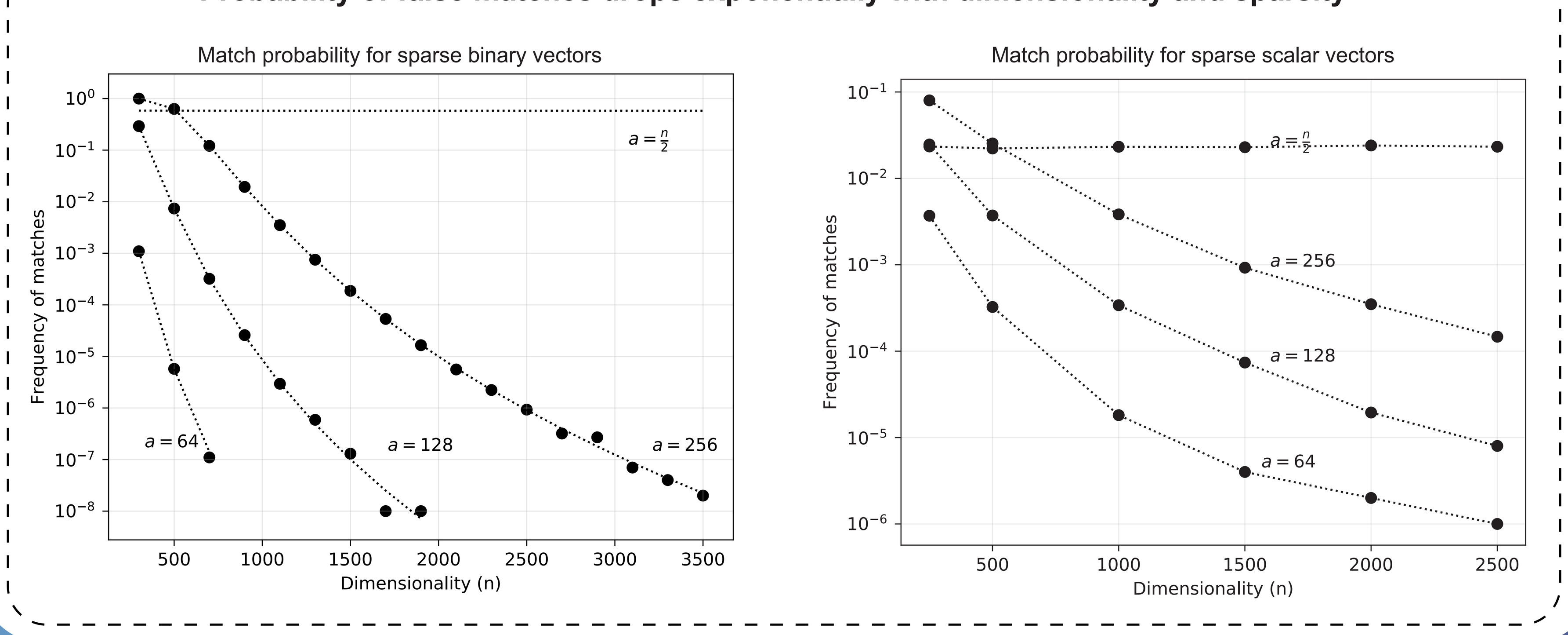
Fraction denotes all possible matching vectors divided by the number of possible vectors.

Scalar sparse representations:

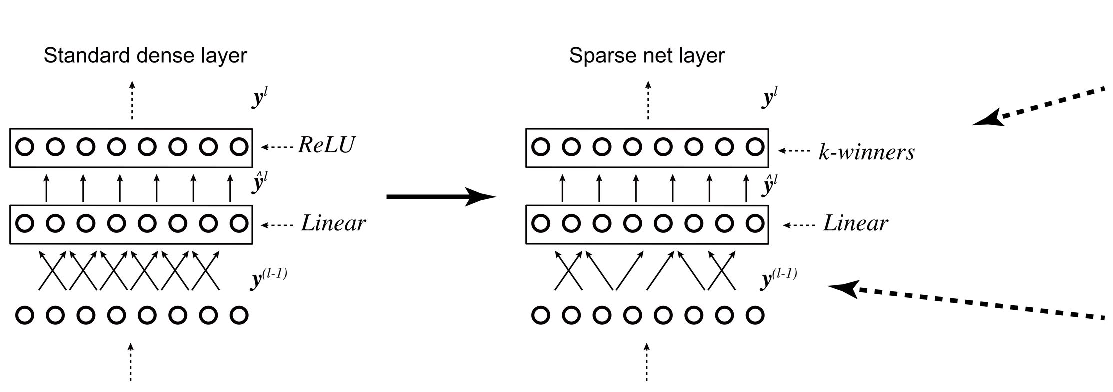
The combinatorics apply to scalar vectors, when the magnitudes of all components are similar.

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

Probability of false matches drops exponentially with dimensionality and sparsity



A simple differentiable sparse layer



Outputs of top-k units are maintained, the rest are set to 0. Just like ReLU, but with dynamic threshold.

Weight matrix is sparse. Most weights are zero, and maintained as zero throughout.

Algorithm details

- A few units can initially dominate. We use an exponential boosting term that favors units with low activation frequency This helps maximize the overall entropy of the layer.

- Line 6 computes the "duty cycle", or average activation frequency.

- Easy extension to sparse convolutional layers (need to accumulate duty cycle for each filter since kernel weights are shared)

- Activation sparsity for our layers are between 10% and 30%

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

Related work:
(Majani et al., 1989)
(Hawkins, Ahmad, & Dubinsky, 2011)
(Makhzani & Frey, 2015)

Simple to use in PyTorch

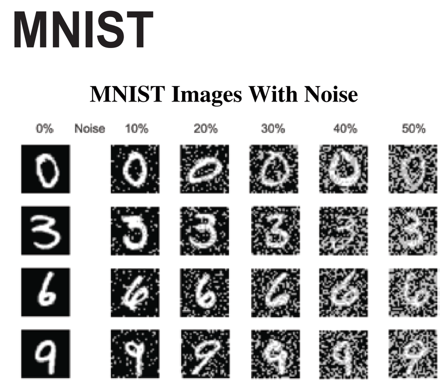
```python
from nupic.torch.modules import KWinners2d, KWinners, SparseWeights, SparseWeights2d, Flatten

sparseCNN = nn.Sequential(
    # Sparse CNN layer
    SparseWeights2d(
        nn.Conv2d(in_channels=IN_CHANNELS, out_channels=OUT_CHANNELS, kernel_size=KERNEL_SIZE),
        WEIGHT_SPARSITY),
    KWinners2d(channels=OUT_CHANNELS, percent_on=PERCENT_ON, boostStrength=BOOST_STRENGTH),

    # MaxPool layer
    nn.MaxPool2d(kernel_size=2),

    # Flatten before passing to linear layer
    Flatten(),

    # Sparse Linear layer
    SparseWeights(nn.Linear(CNN_OUTPUT_LEN, HIDDEN_SIZE), WEIGHT_SPARSITY),
    KWinners(n=HIDDEN_SIZE, percent_on=PERCENT_ON, boostStrength=BOOST_STRENGTH),

    # Output layer
    nn.Linear(HIDDEN_SIZE, OUTPUT_SIZE),
).to(device)
```

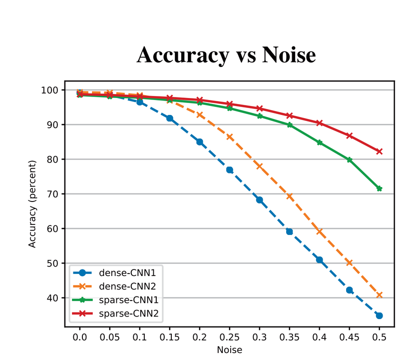
Try it out!
github.com/numenta/nupic.torch

Results

Sparse networks consistently show improved robustness to random noise.

MNIST





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	$70{,}566 \pm 2{,}857$

Google Speech Commands

- Dataset of spoken one word commands
- 65,000 utterances, thousands of individuals
- SOA is around 95 - 97.5% for 10 categories
- Noisy audio samples:

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

11 different noise levels, 0.0 to 0.5

Network	Test Score	Noise Score
Dense CNN-2 (Dr=0.0)	96.37 ± 0.37	$8{,}730 \pm 471$
Dense CNN-2 (Dr=0.5)	95.69 ± 0.48	$7{,}681 \pm 368$
Sparse CNN-2	96.65 ± 0.21	$11{,}233 \pm 1013$
Super-sparse CNN-2	96.57 ± 0.16	$10{,}752 \pm 942$

CIFAR-10

- NotSoDenseNet - sparse version of DenseNet Contains sparse transition and linear layers (No sparsity in level-skipping layers.)

- VGG19-Sparse - standard VGG19 w/ batchNorm, but contains sparse CNN layers

Noise	DenseNet	NotSoDenseNet	VGG19-Dense	VGG19-Sparse
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

Future work

- Test with other noise types and network architectures.

- Additional benchmarks (CIFAR-100, etc.)

Network details/parameters:
github.com/numenta/nupic.research/tree/master/projects/whydense