THE BRAIN'S CIRCUITS SUGGEST COMPUTING WITH HIGH-DIMENSIONAL VECTORS

Pentti Kanerva
pkanerva@berkeley.edu
Redwood Center for Theoretical Neuroscience

- Why high-D?
- Computing in high-D
- An example from language
- The math that makes it work
- Summary and a forecast

WHY HIGH-D?

COMPUTING RESOURCES INHERENT IN BRAINS

. Number of neurons: 40 billion (4×10^{10})

. Number of synapses: 240 trillion (2.4×10^{14})

Assuming 1 bit per synapse -> 30 Terabytes
= 30 million books
= 800 books per day for 100 years
= 76,000 bits/sec. for 100 years

. Fan-in/fan-out:
- Up to 200,000 per neuron
- 6,000 per neuron on average

PRINCIPLES OF ARCHITECTURE suggested by such large numbers

1. **LARGE CIRCUITS**

Computing with high-dimensional vectors, e.g., with

. *10,000-bit words, rather than with 64-bit:*

30-Terabyte brain

= 24 billion 10,000-bit words

2. **MASSIVE MEMORY**

Large amounts of information is learned fast and retained for a *lifetime*. This suggests learning that is minimally invasive, hence

. *Encoding of information and memory storage are **separate** processes*

- In artificial neural nets they are **entangled** which can lead to *catastrophic forgetting*

3. **POSITIVE ROLE OF RANDOMNESS**

The human genome is much too small to specify the brain's construction in minute detail:

3 billion base pairs
vs.
40 billion neurons and
240,000 billion synapses

This suggests a major role for randomness within an overall wiring plan, hence

. *Representation that can benefit from randomness*

4. RELIABLE COMPUTING WITH UNRELIABLE COMPONENTS, GRACEFUL DEGRADATION

Brains tolerate neuron death. This suggests distributing information maximally into the high-dimensional vectors, hence

. *Distributed representation*

- holographic/holistic

These principles can be realized in a fully **general** system of computing that

- resembles *traditional computing* with numbers but works also with components that can fail

- resembles *artificial neural nets* but avoids high-dimensional gradient descent/ backpropagation

COMPUTING WITH 10,000-BIT WORDS VS. 64-BIT

HOW do we compute WITH 64-BIT WORDS?

. Arithmetic/Logic Unit (ALU)
  - **Add** numbers
  - **Multiply** numbers
  - **Compare** numbers

. Random Access Memory (RAM)
  - Store numbers
  - Store pointers
  - Store program code (of numbers and pointers)

10,000-BIT ARITHMETIC (ALU)

OPERATIONS correspond to those with numbers:

. **Add** vectors

  - Thresholded sum: **A** = [**B** + **C** + **D**]

. "**Multiply**" vectors

  - Exclusive-Or, XOR: **M** = **A***B**

+ **Permute** (rotate) vector coordinates: **P** = *r***A**

. **Compare** vectors for similarity

  - Hamming distance, cosine

10,000-BIT WIDE MEMORY (high-D "RAM")

Neural-net associative memory (e.g., Sparse Distributed Memory)

. Content addressable (CAM)

. Addressed with 10,000-bit words

. Stores 10,000-bit words

. Addresses can be noisy

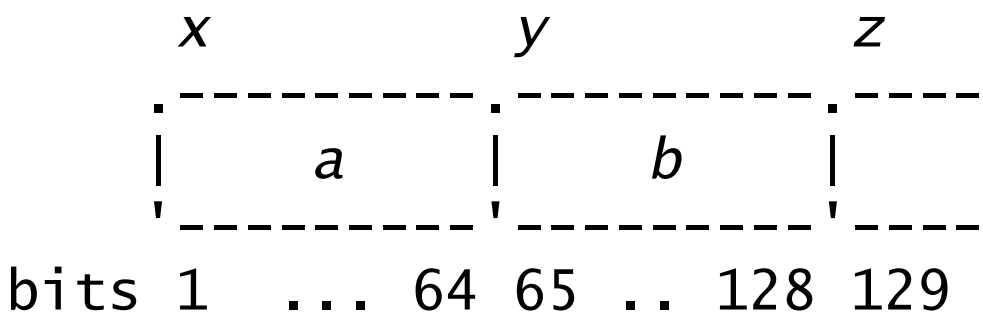
. Can be made arbitrarily large

- for lifetime of learning

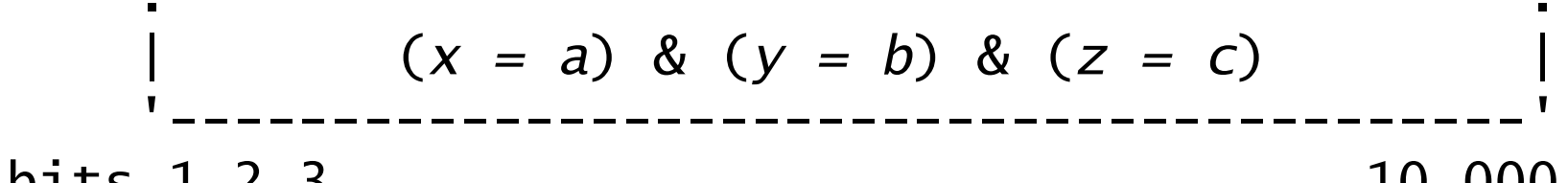
HOLOGRAPHIC/DISTRIBUTED ENCODING OF DATA,
an example

h = '(x = a) and (y = b) and (z = c)'

TRADITIONAL record with fields:



DISTRIBUTED, N = 10,000, no fields:



ENCODING h = '(x = a) and (y = b) and (z = c)'

The variables x, y, z and the values a, b, c are represented by random 10,000-bit vectors **X**, **Y**, **Z**, **A**, **B** and **C**.

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

X = 10010...01 **X** and **A** are bound with XOR
A = 00111...11

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

```
 X  = 10010...01     X and A are bound with XOR
 A  = 00111...11
-----------------
X*A = 10101...10 -> 1 0 1 0 1 ... 1 0   (x = a)
```

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

```
 X  = 10010...01     X and A are bound with XOR
 A  = 00111...11
-----------------
X*A = 10101...10 -> 1 0 1 0 1 ... 1 0   (x = a)

 Y  = 10001...10
 B  = 11111...00
-----------------
Y*B = 01110...10 -> 0 1 1 1 0 ... 1 0   (y = b)
```

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

```
 X  = 10010...01     X and A are bound with XOR
 A  = 00111...11
-----------------
X*A = 10101...10 -> 1 0 1 0 1 ... 1 0   (x = a)

 Y  = 10001...10
 B  = 11111...00
-----------------
Y*B = 01110...10 -> 0 1 1 1 0 ... 1 0   (y = b)

 Z  = 01101...01
 C  = 10001...01
-----------------
Z*C = 11100...00 -> 1 1 1 0 0 ... 0 0   (z = c)
```

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

```
 X   = 10010...01     X and A are bound with XOR
 A   = 00111...11
-----------------
X*A = 10101...10 -> 1 0 1 0 1 ... 1 0   (x = a)

 Y   = 10001...10
 B   = 11111...00
-----------------
Y*B = 01110...10 -> 0 1 1 1 0 ... 1 0   (y = b)

 Z   = 01101...01
 C   = 10001...01
-----------------
Z*C = 11100...00 -> 1 1 1 0 0 ... 0 0   (z = c)
                   --------------------------
                    2 2 3 1 1 ... 2 0   sum
```

ENCODING *h* = '(*x* = *a*) and (*y* = *b*) and (*z* = *c*)'

```
 X  = 10010...01     X and A are bound with XOR
 A  = 00111...11
-----------------
X*A = 10101...10 -> 1 0 1 0 1 ... 1 0   (x = a)

 Y  = 10001...10
 B  = 11111...00
-----------------
Y*B = 01110...10 -> 0 1 1 1 0 ... 1 0   (y = b)

 Z  = 01101...01
 C  = 10001...01
-----------------
Z*C = 11100...00 -> 1 1 1 0 0 ... 0 0   (z = c)
                   --------------------------
              Sum =  2 2 3 1 1 ... 2 0
Threshold at 3/2 =  1 1 1 0 0 ... 1 0  = H
```

DECODING: What is the value of **X** in **H**?

DECODING: What is the value of **X** in **H**?

$$\mathbf{H} = 1\ 1\ 1\ 0\ 0\ \ldots\ 1\ 0$$
$$\mathbf{X} = 1\ 0\ 0\ 1\ 0\ \ldots\ 0\ 1$$

DECODING: What is the value of **X** in **H**?

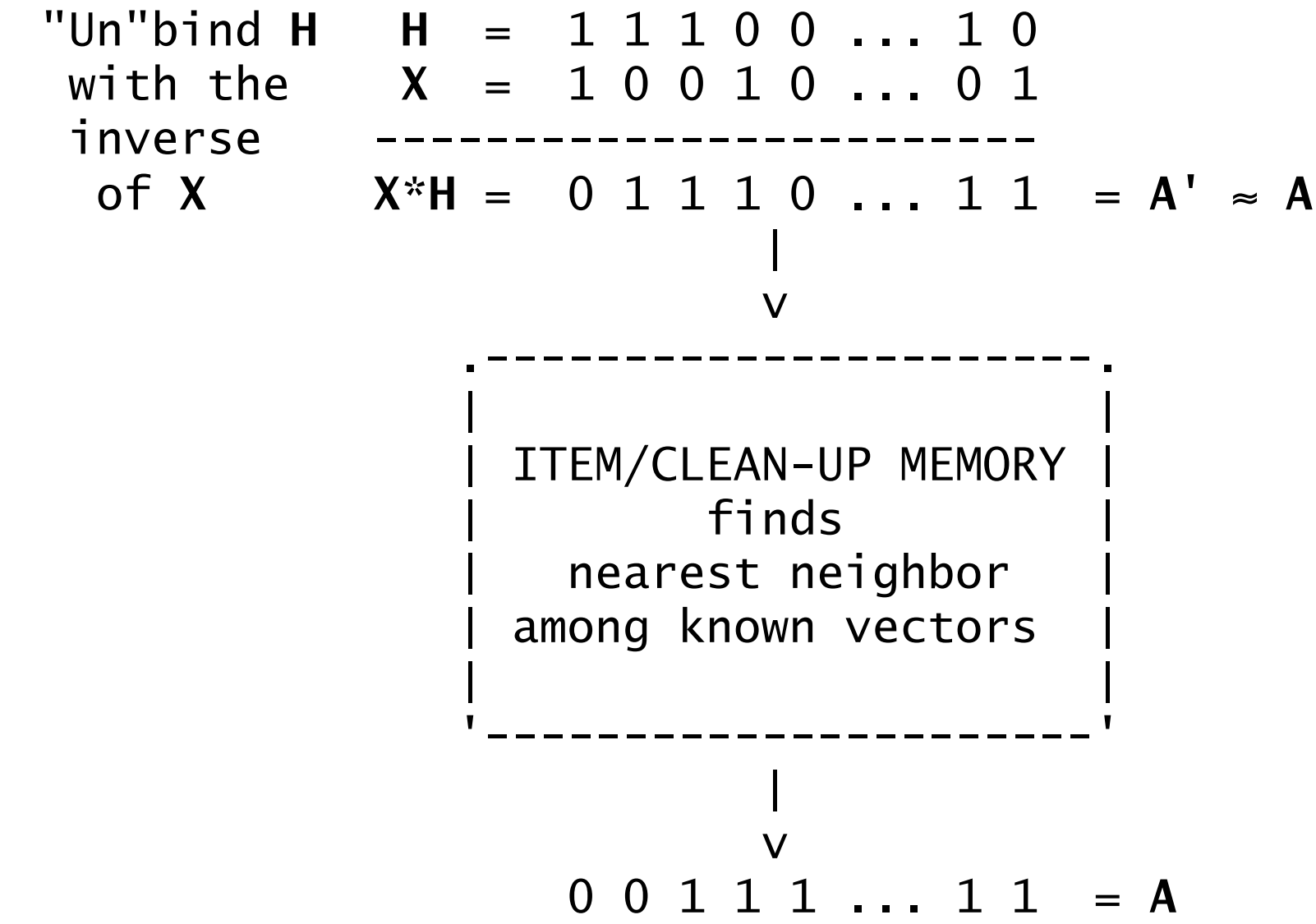
"Un"bind **H** with the inverse of **X**

H = 1 1 1 0 0 ... 1 0

X = 1 0 0 1 0 ... 0 1

X*H = 0 1 1 1 0 ... 1 1 = **A'** ≈ **A**

DECODING: What is the value of **X** in **H**?



In symbols:

H = [**X***A + **Y***B + **Z***C]

What is the value of **X** in **H**?

X*H = **X***[**X***A + **Y***B + **Z***C]
= [**X***X*A + **X***Y*B + **X***Z*C] (1)
= [**A** + noise + noise] (2)
≈ **A**

(1) because * *distributes* over +
(2) **X***X cancels out because XOR (*) is its own *inverse*
"noise" means: dissimilar to any known vector

HIGH-D ALGORITHM: An example from language

MOTIVATION: People can identify languages by how they sound

We emulated it with identifying languages by how they look in print

Method:

Compute a **profile** for each language and compare the profiles

. A profile is a HD vector that summarizes short letter sequences (*trigrams*) of the text. Profile is a kind of **histogram.**

COMPUTING THE PROFILE

Step 1. **ENCODE LETTERS** with 27 **seed** vectors

10,000 random equally probable 1s and -1s

A = (-1 +1 -1 +1 +1 +1 -1 ... +1 +1 -1)

B = (+1 -1 +1 +1 +1 -1 +1 ... -1 -1 +1)

C = (+1 -1 +1 +1 -1 -1 +1 ... +1 -1 -1)

...

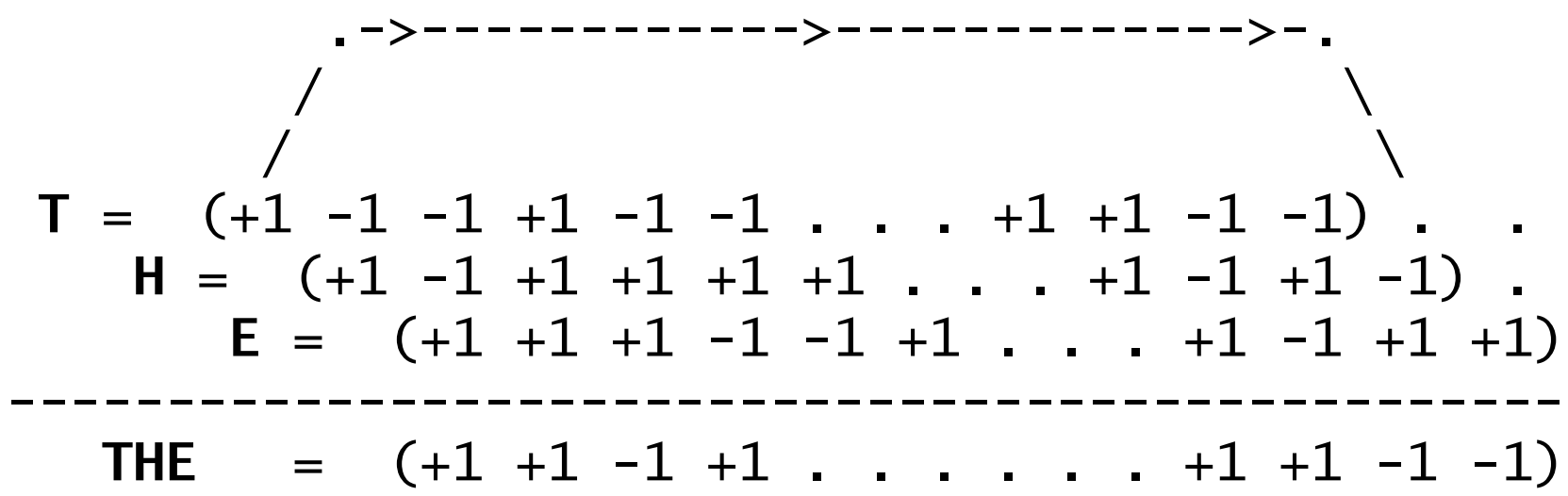
Z = (-1 -1 -1 -1 +1 +1 +1 ... -1 +1 -1)

= (+1 +1 +1 +1 -1 -1 +1 ... +1 +1 -1)

stands for the space

Step 2. **ENCODE TRIGRAMS** with rotate and multiply

Example: "the" is encoded by the 10,000-D vector **THE**



In symbols:

THE = *rr***T** * *r***H** * **E**

where

r is 1-position *rotate* (it's a permutation)
* is componentwise *multiply*

The trigram vector **THE** is approximately orthogonal to all the letter vectors **A**, **B**, **C**, ..., **Z** and to all the other (19,682) possible trigram vectors (and to all 531,441 possible tetragram vectors ...)

Step 3. **ACCUMULATE PROFILE** VECTOR

Add all the trigram vectors of a text into a single 10,000-D **Profile Vector.**

For example, the text segment

"*the quick brown fox jumped over* ..."

gives rise to the following trigram vectors:

#TH THE HE# E#Q #QU QUI UIC ICK CK# K#B ...

Note: The profile for a million bytes of text is a sum of a million 10,000-D trigram vectors

Step 4. **TEST** THE PROFILES of 21 EU languages:

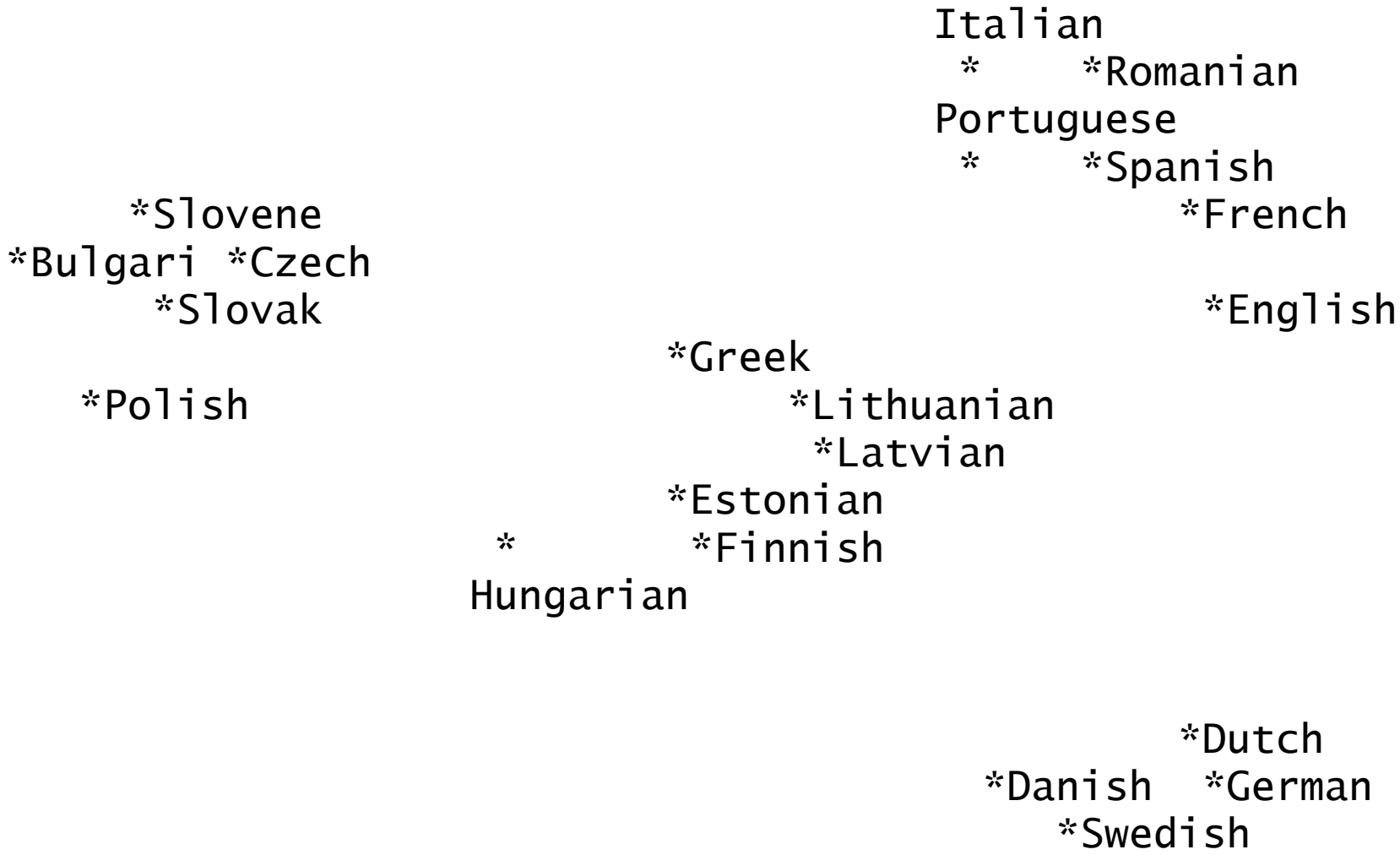
. Similarity between vectors/profiles: Cosine

cos(**X**, **X**) = 1

cos(**X**,-**X**) = -1

cos(**X**, **Y**) = 0 if **X** and **Y** are *orthogonal*

Step 4a. Projected onto a plane, the profiles cluster in known *families*



Step 4b. The language profiles were compared to the profiles of 21,000 test sentences (1,000 sentences from each language).

The best match agreed with the correct language 97.3% of the time.

The entire experiment ("training" and testing) took less than 8 minutes on a laptop computer

Reference:

Joshi, A., Halseth, J., and Kanerva, P. (2017). Language geometry using random indexing. In J. A. de Barros, B. Coecke & E. Pothos (eds.) *Quantum Interaction*, 10th International Conference, QI 2016, pp. 265-274. Springer.

THE MATH THAT MAKES IT WORK

- A **randomly** chosen vector is **dissimilar** to any vector seen so far--approximately *orthogonal*, "quasiorthogonal"

- *Addition* produces a vector that is **similar** and *multiplication* and *permutation* produce vectors that are **dissimilar** to the input vectors

- *Multiplication* is **invertible** and **distributes** over addition

- *Permutation* is **invertible** and **distributes** over both addition and multiplication

- *Multiplication* and *permutation* **preserve similarity**

COMPUTING WITH HD VECTORS IS BEST UNDERSTOOD
IN (modern/abstract) **ALGEBRA** TERMS;

. *Addition* and *multiplication* approximate an algebraic **field** *over the vector space*.

  - Note: The usefulness of arithmetic with numbers is based on the same idea: addition and multiplication form a field

. *Permutation* adds richness to HD algebra:

  - e.g., any *finite group* can be realized with permutations

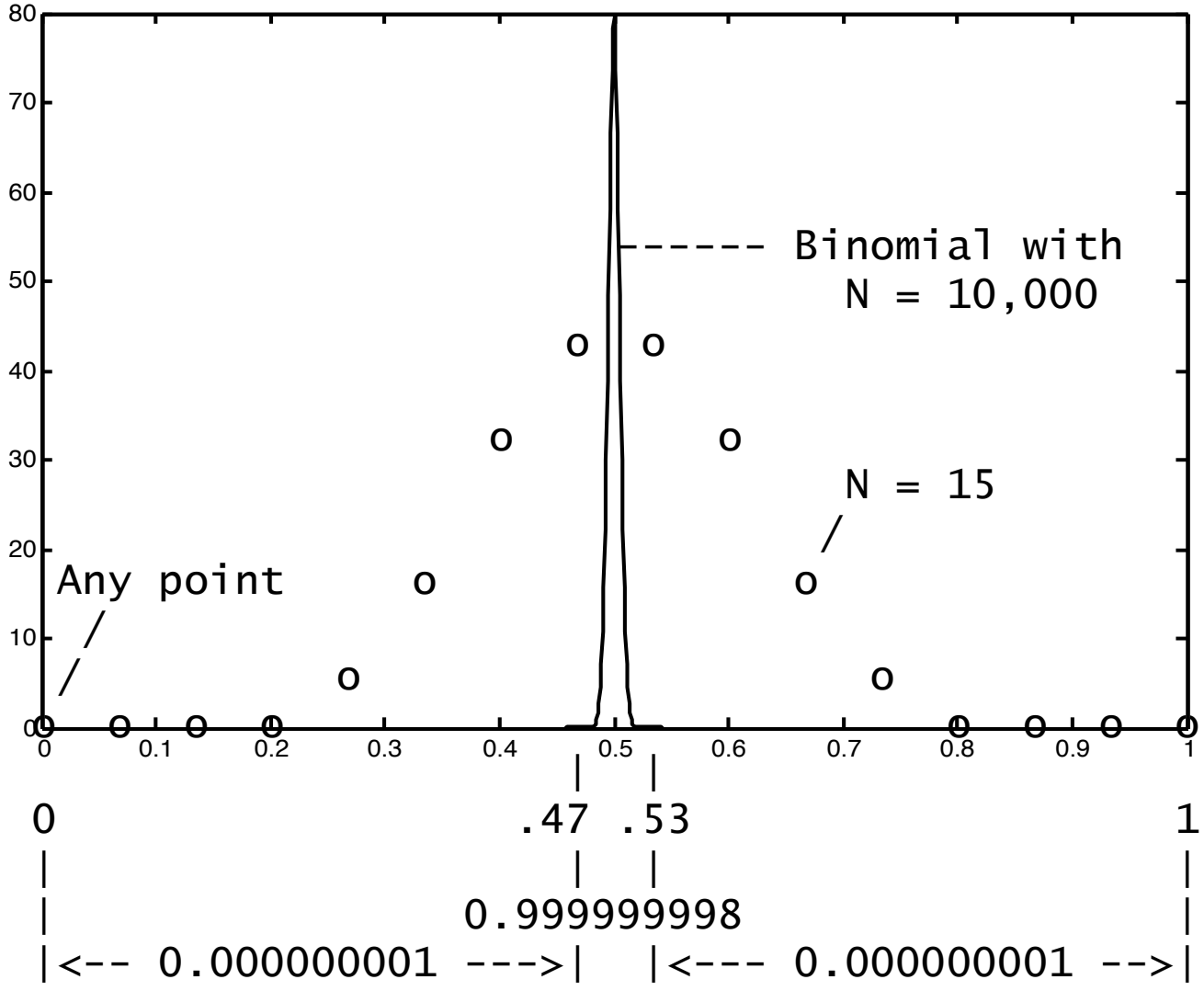
. The algebra makes HD computing **transparent**

  - Open the "black box"

HIGH DIMENSIONALITY IS THE KEY

- In 10,000 dimensions there are 10,000 mutually orthogonal vectors but billions of nearly orthogonal vectors

- A randomly chosen vector is nearly orthogonal to any that has been seen so far--that could have been stored in the system's memory


80
70
60
50
40
30
20
10
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
----- Binomial with
N = 10,000
N = 15
Any point
0
.47 .53
1
0.999999998
|<-- 0.000000001 --->|
|<--- 0.000000001 -->|
Similar,
correlated
Dissimilar,
uncorrelated
Opposite

SUMMARY

. HD Computing is based on the rich and often subtle mathematics of high-dimensional spaces

  - Not subsumed under liner algebra

. High dimensionality (10,000-D) is more important than the nature of the dimensions:

  - Multiply and add operators of the right kind exist also for real and complex vectors

. Holographic/holistic representation makes high-D computing robust and brainlike

EARLY CONTRIBUTIONS

Tony Plate: Holographic Reduced Representation (HRR)

Ross Gayler: Multiply-Add-Permute (MAP) architecture

Gayler & Levi: Vector-Symbolic Architecture (VSA)

Rachkovskij & Kussul: Context-Dependent Thinning

Dominic Widdows: Geometry and Meaning

Widdows & Cohen: Predication-based Semantic Indexing (PSI)

Chris Eliasmith: Semantic Pointer Architecture Unified Network (SPAUN)

Gallant & Okaywe: Matrix Binding of Additive Terms

My: Sparse Distributed Memory, Binary Spatter Code, Random Indexing, and Hyperdimensional Computing

FORECAST

. A new breed of computers will be built exploiting the geometry of high-D spaces and the algebra of operations on high-D vectors

. Simpler, faster, more powerful machine-learning algorithms become possible:

- Natural language
- Analogical reasoning
- Vision
- Multi-sensor integration
- Big Data
- Autonomous robots
- Open-ended evolving systems

. The theory of high-D computing is an excellent match to nanometer electronic technology

  - Computers commensurate with brains will become practical

  - This can happen in the next 20-25 years

. Meanwhile--next 10 years--we need to become fluent in computing with high-dimensional vectors

  - It is not particularly hard, only different from what we are used to

  - Present-day computers are more than adequate
    -- as the proving ground and
    -- for many applications

ACKNOWLEDGMENT of support

Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA

Intel Strategic Research Alliance program on Neuromorphic Architectures for Mainstream Computing

NSF 16-526: Energy-Efficient Computing: from Devices to Architectures (E2CDA). A Joint Initiative between NSF and SRC

T H E n d

T H A N K Y O U!

****
Salishan Conference on High-Speed Computing
Session 3: Neuro-Inspired Computing, April 26, 2017
****

The Brain's Circuits Suggest Computing with High-Dimensional Vectors

ABSTRACT

Traditional computing is deterministic. It is based on bits and assumes that the bits compute perfectly. However, (near)perfection at high speeds consumes large amounts of energy and limits the scaling-down of the underlying circuit elements. Contrast this with the brain's powers of perception and learning. They go far beyond what computers can do, are accomplished with little energy by slow and imprecise neurons wired together according to a general plan, with many details left to chance. Can we make sense of that kind of computing and build systems on the same principles?

Computing with high-dimensional vectors (hypervectors, ultrawide words) is based on the idea that the brain's powers are rooted in the mathematics of very-high-dimensional spaces--when the dimensionality is in the thousands to tens-of-thousands. The very size of the brain's circuits suggests very high dimensionalities.

Information encoded into a hypervector is distributed equally over all vector components--the vectors are not subdivided into fields. Operations on such vectors are the key to computing with them.

Traditional computers work with bits and numbers, and their most important operations are built into the hardware. These include circuits for the addition and multiplication of numbers. Similar operations are used with hypervectors: one corresponds to addition and another to multiplication. A third operation simply permutes the coordinates and a fourth computes the similarity of vectors.

It is possible to build very rich systems of computing based on these four operations. Their practical value follows from several factors: (1) simple operations yield vectors that are machine-learning friendly, (2) the operations are easily distributed and performed in parallel, (3) computation is extremely robust even with unreliable low-resolution components, and (4) the prerequisite circuits are a natural fit to next-generation nanotechnology. In the field of computing at large, high-dimensional computing combines symbolic computing and neural-net computing/deep learning, and it complements numeric computing. It has been demonstrated with semantic vectors, language identification, and EEG signal classification, among other things, and research into the theory and its application is ongoing,