COMPUTING with HIGH-DIMENSIONAL VECTORS

Pentti Kanerva

UC Berkeley,
Redwood Center for Theoretical Neuroscience
Stanford, CSLI
pkanerva@csli.stanfrod.edu

- Motivation and Background
- What is HD Computing?
- Example from Language
- HD Computing Architecture
- The Math that Makes HD Computing Work
- Contrast with Neural Nets/Deep Learning
- Summary

MOTIVATION AND BACKGROUND

Brains represent a constant *challenge* to our models of computing: von Neumann, AI, Neural Nets, Deep Learning

. Complex behavior

- Perception, learning
- Concepts, thought, language, ambiguity
- Flexibility, adaptivity

. Robustness

- Sensory signals are variable and noisy
- Neurons malfunction and die

. Energy efficiency

- 20 W

Brains provide *clues* to computing architecture

. Very large circuits

- 40 billion (4×10^{10}) neurons
- 240 trillion (2.4×10^{14}) synapses

Assuming 1 bit per synapse -> 30 Terabytes
= 30 million books
= 800 books per day for 100 years

. Large fan-ins and fan-outs
- Up to 200,000 per neuron
- 6,000 per neuron on average

. Activity is widely distributed, highly parallel

However, *reverse-engineering* the brain in the absence of an adequate theory of computing is next to impossible

The theory must explain

- Speed of learning
- Retention over a lifetime
- Generalization from examples
- Reasoning by analogy
- Tolerance for variability and noise in data
- ...

KEY OBSERVATIONS

Essential properties of mental functions and perception can be explained by the mathematical properties of high-dimensional spaces

. Distance between concepts in semantic space

- *Distant concepts* connected by *short links*

  man $\not\approx$ lake
  man ≈ fisherman ≈ fish ≈ lake
  man ≈ plumber ≈ water ≈ lake

. Recognizing faces: never the same twice

Dimensionality expansion rather than reduction

. Visual cortex, hippocampus, cerebellum

WHAT IS HIGH-DIMENSIONAL (HD) COMPUTING?

It is a system of computing that operates on high-dimensional **vectors**

. The algorithms are based on *operations on vectors*

Traditional computing operates on **bits** and **numbers**

. Built-in circuits for arithmetic and for Boolean logic

ROOTS in COGNITIVE SCIENCE

The idea of computing with high-dimensional vectors is **not new**

- 1950s - Von Neumann: *The Computer and the Brain*
- 1960s – Rosenblatt: *Perceptron*
- 1970s and '80s - Artificial Neural Nets/ Parallel Distributed Processing/Connectionism
- 1990s – Plate: *Holographic Reduced Representation*

What is **new**?

- *Nanotechnology* for building very large systems
  - In need of a compatible theory of computing

AN EXAMPLE OF HD ALGORITHM: Identify the Language

MOTIVATION: People can identify languages by how they sound, without knowing the language

We emulated it with identifying languages by how they look in print, without knowing any words

METHOD

. Compute a 10,000-dimensional **profile** vector for each language and for each test sentence

. Compare profiles and choose the closest one

DATA

- 21 European Union languages
- Transcribed in Latin alphabet
- "Trained" with a million bytes of text per language
- Tested with 1,000 sentences per language from an independent source

COMPUTING a PROFILE

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

10K *random*, equally probable +1s and -1s

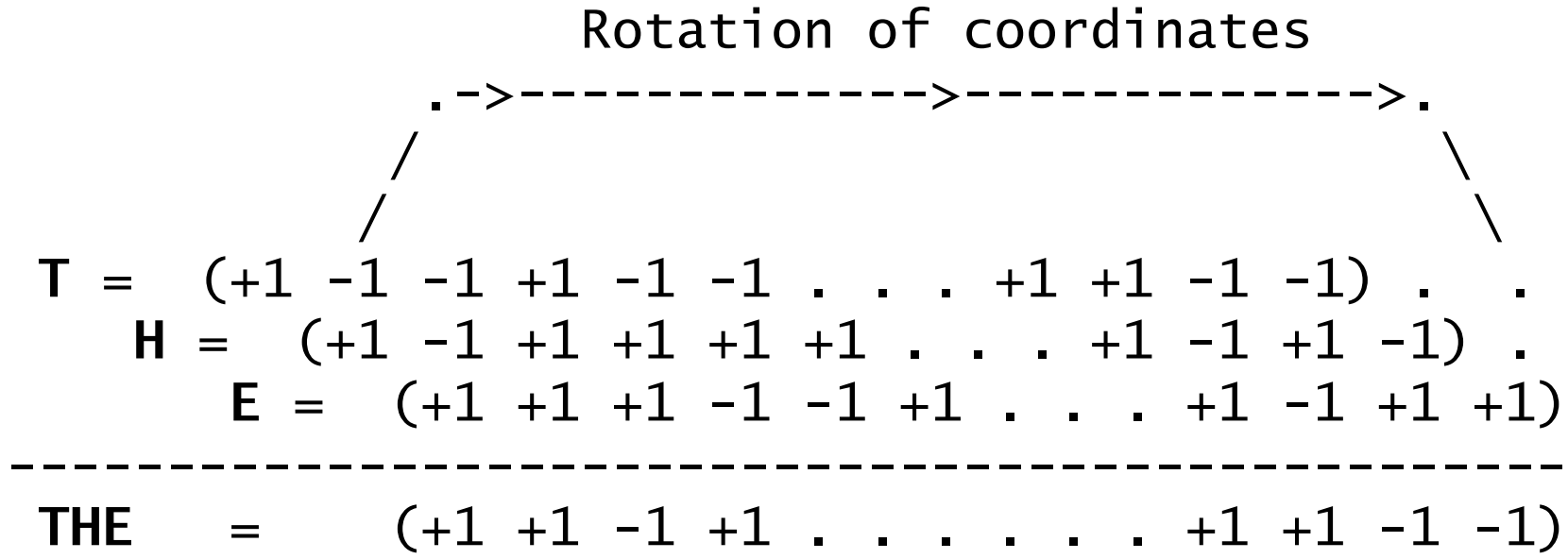
A = (-1 +1 -1 +1 +1 +1 -1 ... +1 +1 -1)
B = (+1 -1 +1 +1 +1 -1 +1 ... -1 -1 +1)
C = (+1 -1 +1 +1 -1 -1 +1 ... +1 -1 -1)
...
Z = (-1 -1 -1 -1 +1 +1 +1 ... -1 +1 -1)
= (+1 +1 +1 +1 -1 -1 +1 ... +1 +1 -1)

stands for the space

All languages use the *same* set of letter vectors

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

Example: "*the*" is encoded by the 10K-dimensional 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

Step 3. ACCUMULATE PROFILE VECTOR

Add all trigram vectors of a text into a 10,000-D Profile Vector. For example, the text segment

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

gives rise to the following trigram vectors, which are added into the profile for English

Eng += **THE** + **HE#** + **E#Q** + **#QU** + **QUI** + **UIC** + ...

NOTE: Profile is a HD vector that summarizes short letter sequences (trigrams) of the text; it's **histogram** of a kind

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 *language 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

Step 5. The profile for English, Eng, was queried for the letter most likely to follow "*th*". It is "*e*", with *space*, "*a*", "*i*", "*r*", and "*o*" the next-most likely, in that order

. Form query vector: **Q** = *rr***T** * *r***H**
. Query by using multiply: **X** = **Q***Eng
. Find closest letter vectors:

X ≈ E, #, A, I, R, O

Summary of Algorithm

- Start with **random** 10,000-D bipolar vectors for *letters*
- Compute 10,000-D vectors for *trigrams* with **permute** (rotate) and **multiply**
- **Add** all trigram vectors into a 10,000-D *profile* for the *language* or the *test sentence*
- Compare profiles with *cosine*

Speed

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

Simplicity and Scalability

It is *equally easy* to compute profiles from
 . all 531,441 possible 4-letter sequences, or
 . all 14,348,907 possible 5-letter sequences, or
 . all 387,420,489 possible 6-letter sequences, or
 . all ... or
from *combinations* thereof

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.
`

ARCHITECTURE FOR HIGH-DIMENSIONAL COMPUTING

Computing with HD vectors **resembles** traditional computing with bits and numbers

. Circuits (ALU) for operations on HD vectors

. Memory (RAM) for storing HD vectors

Main **differences** beyond high dimensionality

. Distributed (holographic) representation

  - Computing in **superposition**

. Beneficial use of **randomness**

Illustrated with binary vectors:

- Computing with **10,000-bit words**

Binary and bipolar are mathematically equivalent

- binary 0 <--> bopolar 1
- binary 1 <--> bipolar -1
- XOR <--> multiply
- majority <--> sign

Note, and not to confuse:

- Although XOR is *addition modulo 2*, it is the **multiplication** operator for binary vectors

10K-BIT ARITHMETIC (ALU)

OPERATIONS correspond to those with numbers

. "**ADD**" vectors

- Coordinatewise majority: **A** = [**B** + **C** + **D**]

. "**MULTIPLY**" vectors

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

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

. **COMPARE** vectors for similarity

- Hamming distance, cosine

10K-BIT WIDE MEMORY (high-D "RAM")

Neural-net associative memory (e.g., Sparse Distributed Memory, 1984)

. Addressed with 10,000-bit words

. Stores 10,000-bit words

. Addresses can be noisy

. Can be made arbitrarily large

  - for a lifetime of learning

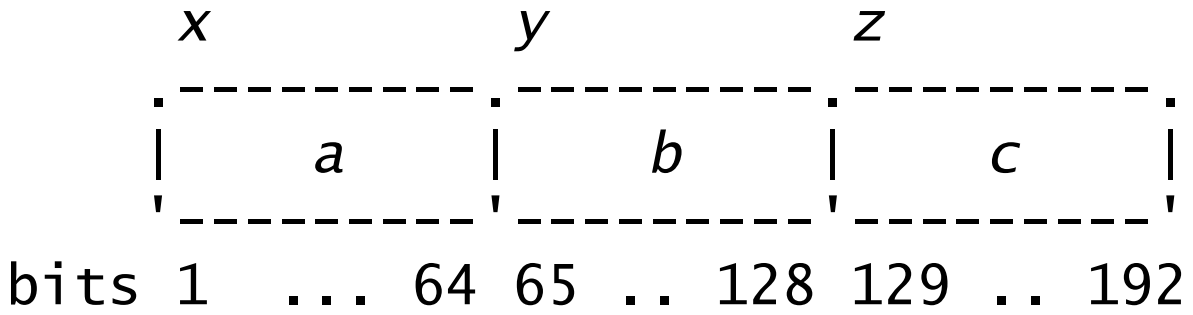
. Circuit resembling *cerebellum's*

  - David Marr (1969), James Albus (1971)

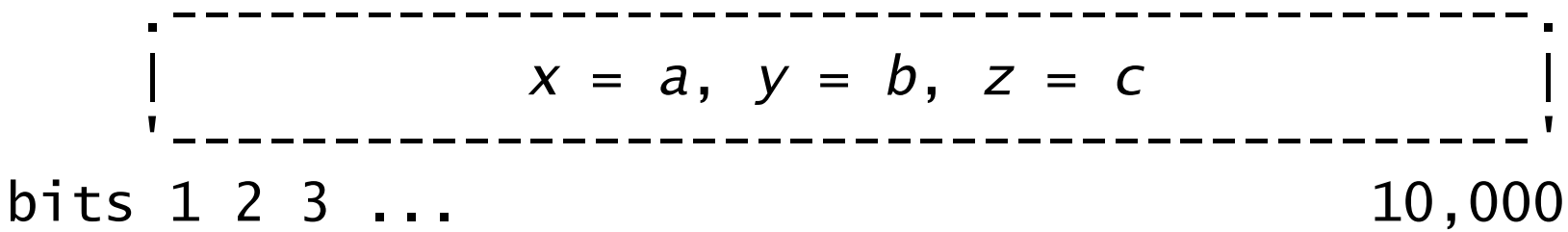
DISTRIBUTED (HOLOGRAPHIC) ENCODING OF DATA

Example: $h = \{x = a,\ y = b,\ z = c\}$

TRADITIONAL record with fields



DISTRIBUTED, SUPERPOSED, $N = 10{,}000$, *no fields*



ENCODING $h = \{x = a,\ y = b,\ z = c\}$

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

ENCODING $h = \{x = a,\ y = b,\ z = c\}$

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

ENCODING $h = \{x = a,\ y = b,\ z = c\}$

```
 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 $h = \{x = a,\ y = b,\ z = c\}$

```
 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 $h = \{x = a,\ y = b,\ z = c\}$

```
 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 $h = \{x = a,\ y = b,\ z = c\}$

```
 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
```

ENCODING $h = \{x = a,\ y = b,\ z = c\}$

```
 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
       Majority =  1 1 1 0 0 ... 1 0  = H
```

DECODING: What's the value of **X** in **H**?

DECODING: What's 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's the value of **X** in **H**?

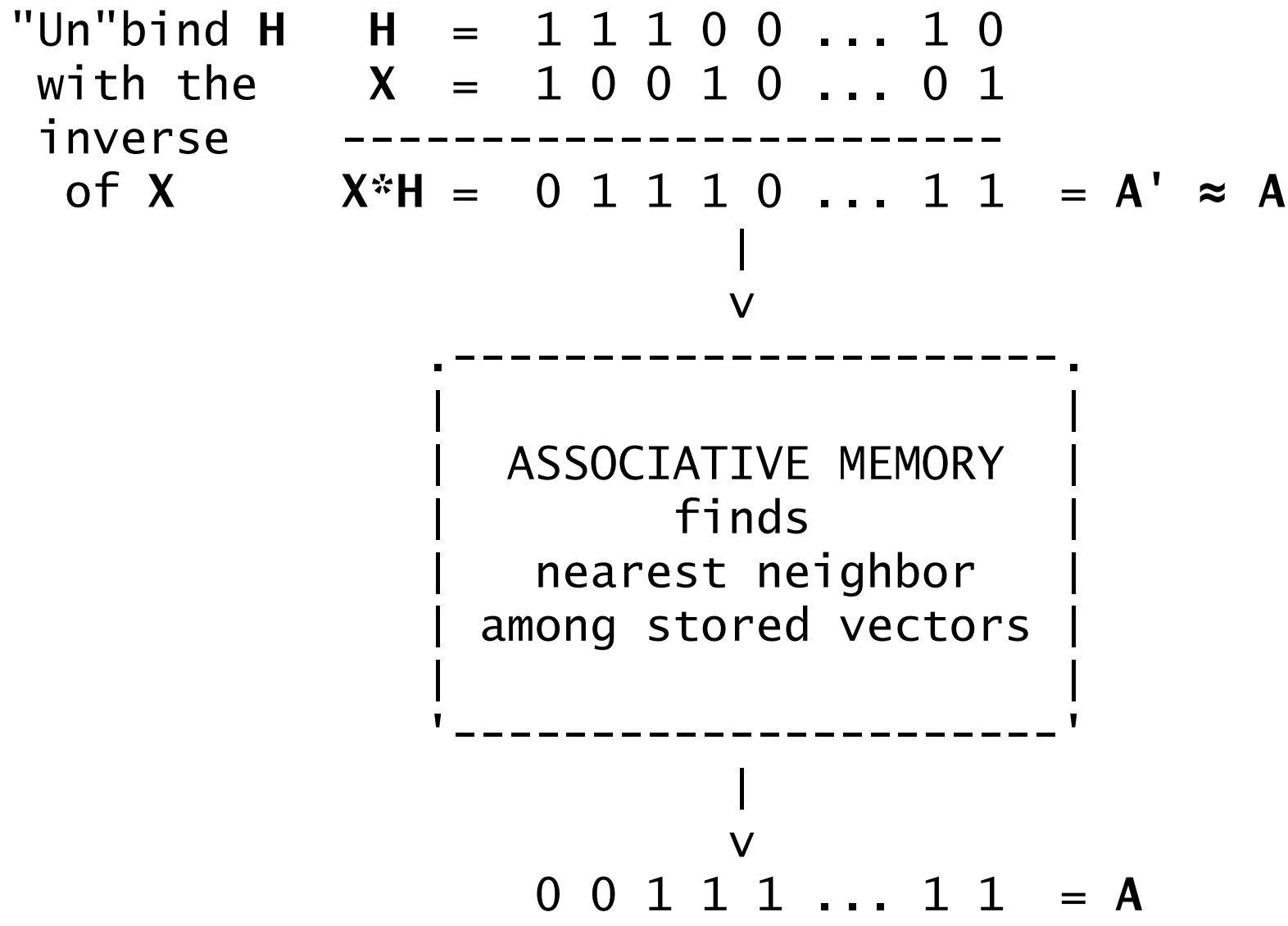
"Un"bind **H** with the inverse of **X**

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's the value of **X** in **H**?



In symbols:

Encoding of $h = \{x = a, y = b, z = c\}$
with majority rule

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

What's 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, approximately *orthogonal*

THE MATH THAT MAKES HD COMPUTING WORK

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

. Three operations on vectors produce a vector of the **same dimensionality**

  - Addition
  - Multiplication
  - Permutation

. 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

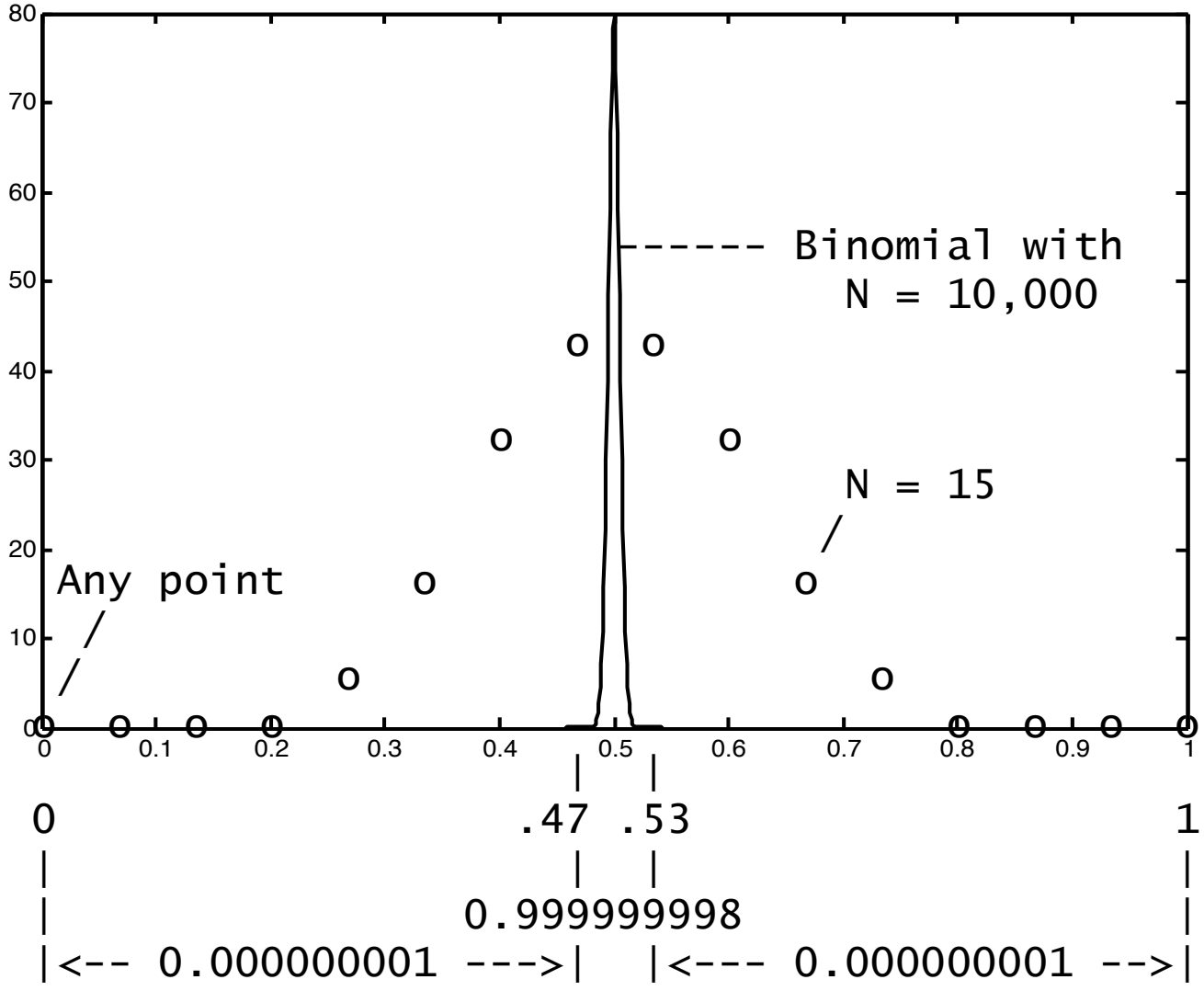
. Permutation adds richness to HD algebra

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--and could have been stored in the system's memory

FIGURE 1 (next page). Distribution of distances in 15-dimensional and 10,000-dimensional spaces. In 10,000-D most vectors/points are nearly orthogonal (d ≈ 0.5) to any given vector.


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

CONTRAST with NEURAL NETS/DEEP LEARNING

Major similarities

- Statistical learning from data
- Data can be noisy
- Both use high-dimensional vectors, although neural-net learning algorithms bog down when dimensionality gets very high (e.g., 10K)

Major differences

. HD computing is founded on rich and powerful **mathematical theory** rather than on *heuristics*

. New codewords are made from existing ones in a **single application** of the HD vector operations

. HD **memory** is a **separate** function
  - Sores data and retrieves the best match (nearest-neighbor search)

As a consequence, HD algorithms are

- transparent,

- incremental (on-line), and

- scalable

Contrasted with neural nets that train large matrices of weights in *multiple passes* over the training data

- As a consequence, encoding and data storage become *entangled* in hard-to-fathom black boxes

The neural-net/deep-learning paradigm suggests no particular role for a structure like the *cerebellum*, which contains over half the neurons in the brain

SUMMARY

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

  - Not a subset of linear algebra

. High dimensionality (10K) 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

FORECAST

. A new breed of computers will be built exploiting the mathematics of high-D spaces and properties of nanomaterials

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

- Vision: relational reasoning
  -- "What's below 2 and to the left of 1?"

- Language: analogical reasoning
  -- "What's the Dollar of Mexico?"

- Streaming data: multi-sensor integration

- Autonomous robots

- Open-ended evolving systems

WHAT NEXT?

We need to become fluent in high-dimensional computing

. Explore HD algorithms
  - Machine learning
  - Language
  - Big Data

. Today's computers are more than adequate also for many applications
  - Semantic vectors: Gavagai AB in Sweden
  - Classifying gestures from 64-channel EMG

. Explore nanomaterials

. Build into circuits and systems

T h a n k Y o u!

Supported by

. Japan 's MITI grant to Swedish Institute of Computer Science under Real World Computing (RWC) program

. 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

EARLY CONTRIBUTIONS, 1990-2010

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, Hyperdimensional Computing

Stanford EE Computer Systems Colloquium (EE380)
4:30 pm on Wednesday October 25, 2017 at B03 Gates

Computing with High-Dimensional Vector

Pentti Kanerva, UC Berkeley & Stanford

Abstract

Computing with high-dimensional vectors complements traditional computing and occupies the gap between symbolic AI and artificial neural nets. Traditional computing treats bits, numbers, and memory pointers as basic objects on which all else is built. I will consider the possibility of computing with high-dimensional vectors as basic objects, for example with 10,000-bit words, when no individual bit nor subset of bits has a meaning of its own--when any piece of information encoded into a vector is distributed over all components. Thus a traditional data record subdivided into fields is encoded as a high-dimensional vector with the fields superposed.

Computing power arises from the operations on the basic objects--from what is called their *algebra*. Operations on bits form Boolean algebra, and the addition and multiplication of numbers form an algebraic structure called a "*field*." Two operations on high-dimensional vectors correspond to the *addition* and *multiplication* of numbers. With *permutation* of coordinates as the third operation, we end up with a system of computing that in some ways is richer and more powerful than arithmetic, and also different from linear algebra. Computing of this kind was anticipated by von Neumann, described by

Plate, and has proven to be possible in high-dimensional spaces of different kinds.

The three operations, when applied to *orthogonal or nearly orthogonal* vectors, allow us to encode, decode and manipulate sets, sequences, lists, and arbitrary data structures. One reason for high dimensionality is that it provides a nearly endless supply of nearly orthogonal vectors. Making of them is simple because a randomly generated vector is approximately orthogonal to any vector encountered so far. The architecture includes a memory which, when cued with a high-dimensional vector, finds its nearest neighbors among the stored vectors. A neural-net associative memory is an example of such.

Circuits for computing in high-D are thousands of bits wide but the components need not be ultra-reliable nor fast. Thus the architecture is a good match to emerging nanotechnology, with applications in many areas of machine learning. I will demonstrate high-dimensional computing with a simple algorithm for identifying languages.