COMPUTING with HIGH-DIMENSIONAL VECTORS

ORIGINS OF THE IDEA

Cognitive Psychology, in reaction to Behaviorism

Cognitive Science: more interdisciplinary

Models of the mind increasingly influenced by computers and computing:

. The brain as a computer
. The computer as an electronic brain
. Artificial Intelligence (AI)

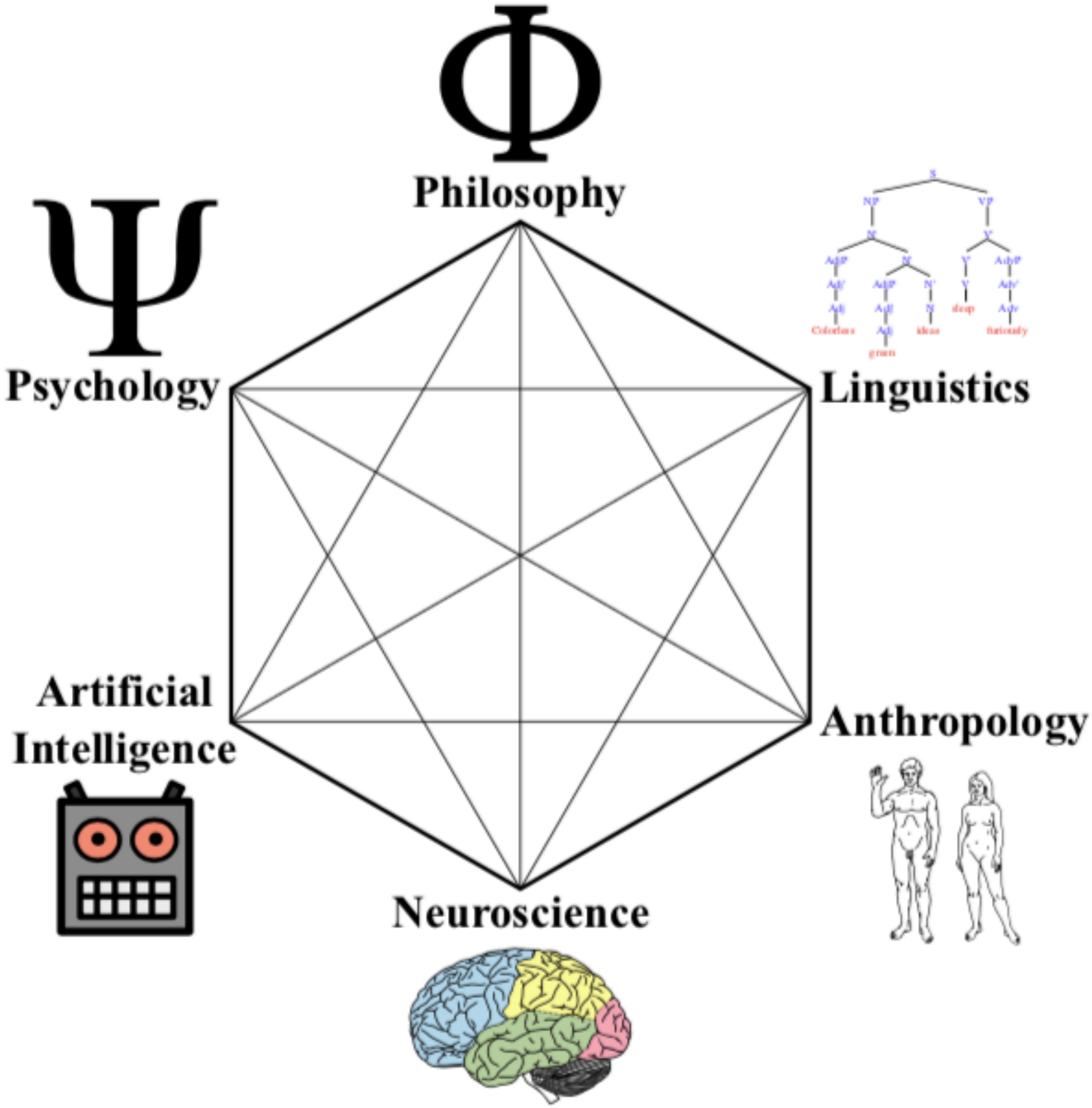
The mind is poorly modeled by conventional computers

Recommended reading:

Wikipedia and
Stanford Encyclopedia of Philosophy articles on

. Cognitive Psychology and

. Cognitive Science



Cognitive Science (from Wikipedia)

`

WHAT DO WE MEAN BY "COMPUTING"?

It's about **math**

. calculating
. arithmetic
. numbers

It's about keeping records and organizing **data**

. memory pointers

It's about **communication**

. world-wide web

It’s about **monitoring** and **control**

. robotics

Math has *co-evolved* with **physics** and **engineering**

Standard math serves their needs

Arithmetic with numbers serves standard math

Computing with numbers serves standard math

Traditional computing is optimized for computing with numbers

Traditional (von Neumann) computing architecture

```
           Central Processing Unit (CPU)
             .-----------------------.
             | Program control unit  |
   Input ->  |-----------------------| -> Output
             | Arithmetic/Logic Unit |
             |   (ALU)    + *        |
             '-----------------------'
                     |     ^     von Neumann
                     v     |     bottleneck
  .---------------------------------------------.
  | # # # # # # # # # # # # # # # # # # # # # # |
  | # # # # # # # # # # # # # # # # # # # # # # |
  | # # #                                 # # # |
  | # # #    Random Access Memory (RAM)   # # # |
  | # # #                                 # # # |
  | # # # # # # # # # # # # # # # # # # # # # # |
  | # # # # # # # # # # # # # # # # # # # # # # |
  '---------------------------------------------'
```

Arithmetic operations on numbers

. *addition*: 1 + 2 = 3
. *multiplication*: 2 * 3 = 6

. multiplication *distributes* over addition:

```
2 * (3 + 4) = 2 * 7 = 14

2 * (3 + 4) = (2 * 3) + (2 * 4)
            = 6 + 8 = 14
```

TRADITIONAL MATH AND COMPUTING WITH NUMBERS ARE A POOR MATCH TO WHAT BRAINS DO

What computers are good at, but brains are not

- Raw speed
- Fast and accurate arithmetic
- Following instructions literally

In contrast, brains do amazing things with minimal energy

- *Recognize* people and things
- *Learn* from example and reason by *analogy*
- Learn to use *language* and reason by *logic*
- Brains control our *interaction* with the world

CLAIM:

COMPUTING WITH VECTORS HAS ITS OWN MATH

THAT IS A BETTER MATCH TO WHAT BRAINS DO

HIGH-DIMENSIONAL REPRESENTATION IS COUNTERINTUITIVE AND SUBTLE (e.g., 10,000-bit vecors)

Nearly all pairs of vectors are **dissimilar**

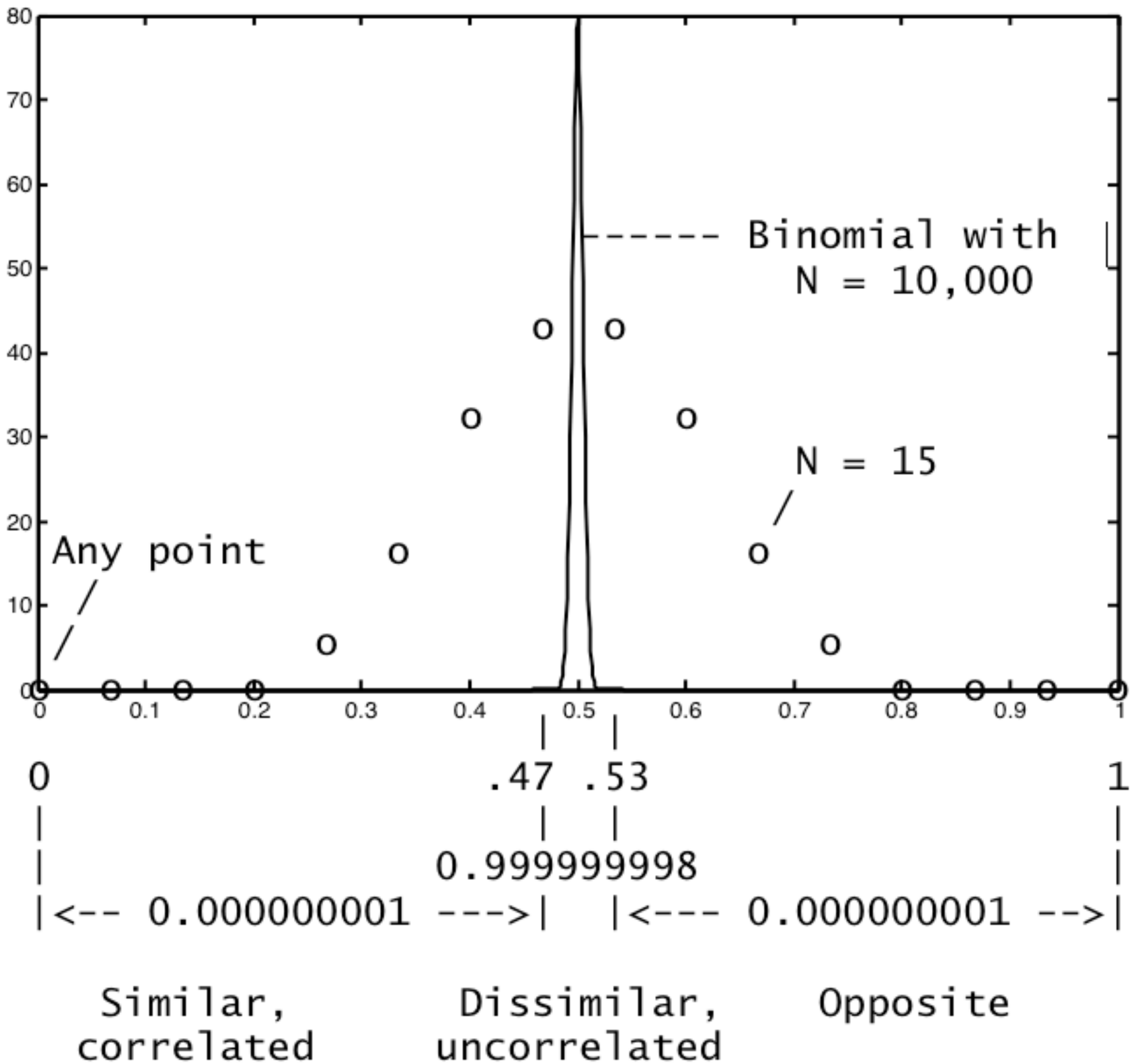
. pairs of random vectors are *approximately orthogonal*
  -- makes representation noise-tolerant, **robust**

Distant concepts have *similar neighbors*

man ≉ lake
man ≈ fisherman ≈ fish ≈ lake
man ≈ plumber ≈ water ≈ lake
plumber ≉ fish

Small cues bring forth complete memories:
"The name starts with T; oh yes, Stephan"

Can explain the *tip-of-the-tongue phenomenon*



Binomial distribution, N = 15 and N = 10,000

BUT HOW DO YOU COMPUTE WITH THE VECTORS?

Three simple operations make up a surprisingly powerful system of computing

- Fundamentally different from traditional neural nets/deep learning

- Different also from linear algebra

- The organization of computing, however, is *traditional*

von Neumann-like architecture for high-D vectors

```
         Central Processing Unit (CPU)
           .-----------------------.
           | Program control unit  |
  Input -> |-----------------------| -> Output
          /: Arithmetic/Logic Unit :\
        /  :   (ALU)   + * permute :  \
      /    '- - - - - - - - - - - -'    \
    /                                     \
  /                                         \
. - - - - - - - - - - - - - - - - - - - - - - .
| vec vec vec vec vec vec vec vec vec vec vec |
| vec vec vec vec vec vec vec vec vec vec vec |
| vec vec                             vec vec |
| vec vec      Associative Memory     vec vec |
| vec vec                             vec vec |
| vec vec vec vec vec vec vec vec vec vec vec |
| vec vec vec vec vec vec vec vec vec vec vec |
'---------------------------------------------'
```

HIGH-DIMENSIONAL MATH

Common to high-D vectors of many kind
. NOT a special case

Components

1. Three **operations** on vectors
   . *Addition*: **A** + **B**
   . *Multiplication*: **A*B**
   . *Permutation*: *r*(**A**)

2. Measure of **similarity** (distance-based)

3. Associative **memory**

Explained here with 10,000-dimensional vectors of +1s and -1s, called "**bipolar**"

. Bipolar is essentially the same as binary

RANDOM SEED VECTORS

```
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)
  ...
        <--------- 10,000 wide --------->
```

THREE OPERATIONS on VECTORS

1. **Addition** (+) is ordinary vector addition, possibly followed by normalization

```
   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)
 -------------------------------------------
 A+B+C = (+1 -1 +1 +3 +1 -1 +1 ... +1 -3 -1)
```

2. **Multiplication** (*) happens *coordinatewise*

```
   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)
 -------------------------------------------
 A*B*C = (-1 +1 -1 +1 -1 +1 -1 ... -1 -1 +1)
```

3. **Permutation** of Coordinates, "shuffle"

We use rotation (*r*) as an example of permutation

```
        ................>..................
        :                                 :
        :                                 :
  A = (-1 +1 -1 +1 +1 +1 -1 ... +1 -1 -1) :
           :  :  :  :  :  :      :  :  :  :
  r(A) = (+1 -1 +1 +1 +1 -1 ... +1 -1 -1 -1)
```

Vectors are compared for *similarity* with **dot product** (.) (or cosine or Pierson correlation)

A.A = 10,000, maximally similar, same
A.X = 0, maximally dissimilar, orthogonal

In high-dimensional spaces, almost all pairs of vectors are dissimilar, approximately orthogonal:

A.B ≈ 0 (small relative to 10,000)

One aim of high-dimensional computing is to represent similarity of meaning in similarity of HD vectors.

EXAMPLE 1. Data record with 3 fields

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

TRADITIONAL

```
      x          y          z
    .---------.---------.---------.
    |    a    |    b    |    c    |
    '---------'---------'---------'
bits 1  ... 64 65 .. 128 129 .. 192
```

HOLOGRAPHIC, SUPERPOSED D = 10,000, no fields

```
    .-----------------------------------------.
    |          x = a, y = b, z = c            |
    '-----------------------------------------'
     1 2 3 ...                          10,000
```

Representing $h = \{x = a,\ y = b,\ z = c\}$ **as a single vector**

Step 1. The variables *x*, *y*, *z* and the values *a*, *b*, *c* are represented by *random* 10K **seed vectors** of +1s and -1s:

X, Y, Z, A, B, C

Step 2. **Bind**: Variables are bound to values with coordinatewise *multiplication*

x = *a* becomes **X*****A**
y = *b* becomes **Y*****B**
z = *c* becomes **Z*****C**

Step 3. **Release**: What is the value of **X** in **X*****A**?

Multiply **X*****A** with the *inverse* of **X**

$$\mathbf{X}*(\mathbf{X}*\mathbf{A}) = (\mathbf{X}*\mathbf{X})*\mathbf{A} = \mathbf{A}$$

NOTE: Vectors of +/-1s are their own inverses

Step 4. **Superpose**: Variable-value pairs are superposed with coordinatewise *addition*

h = {*x* = *a*, *y* = *b*, *z* = *c*} becomes
H = **X*****A** + **Y*****B** + **Z*****C**

Step 5. **Release:** What is the value of **X** in **H**?

Multiply with (the inverse of) **X**

$$
\begin{aligned}
\mathbf{X}*\mathbf{H} &= \mathbf{X}*(\mathbf{X}*\mathbf{A} + \mathbf{Y}*\mathbf{B} + \mathbf{Z}*\mathbf{C}) \\
&= \mathbf{X}*\mathbf{X}*\mathbf{A} + \mathbf{X}*\mathbf{Y}*\mathbf{B} + \mathbf{X}*\mathbf{Z}*\mathbf{C} \\
&= \mathbf{A} + \text{noise} + \text{noise} \\
&= \mathbf{A}' \\
&\approx \mathbf{A}
\end{aligned}
$$

Step 6. **Clean-up**: *Associative memory*

. Find nearest neighbor among stored vectors

A' -> **A** with high probability

NOTE. This example demonstrates

. distributivity: *multiplication distributes over addition* (as it does in math with numbers)

. decoding with the *inverse* operation

SUMMARY of the ALGORITHM:

encoding $h = \{x = a,\ y = b,\ z = c\}$ as **H**

```
  X = -1+1+1-1...+1-1
  A = +1+1-1-1...-1-1
---------------------
X*A = -1+1-1+1...-1+1 -> -1 +1 -1 +1 ... -1 +1   {x = a}

  Y = -1+1+1+1...-1+1
  B = -1-1-1+1...-1+1
---------------------
Y*B = +1-1-1+1...+1+1 -> +1 -1 -1 +1 ... +1 +1   {y = b}

  Z = +1-1-1+1...+1-1
  C = -1+1+1+1...+1-1
---------------------
Z*C = -1-1-1+1...+1+1 -> -1 -1 -1 +1 ... +1 +1   {z = c}
                         ---------------------
                   Sum = -1 -1 -3 +1 ... +1 +3

                  Sign = -1 -1 -1 +1 ... +1 +1  = H
```

SUMMARY of the ALGORITHM: decoding **H** for value of *x*

```
          Sign = -1 -1 -1 +1 ... +1 +1  = H
  Inverse of X = -1 +1 +1 -1 ... +1 -1  = X
                 ---------------------
 Release: X*H = +1 -1 -1 -1 ... +1 -1  = A' ≈ A
                         |
                         v
             .-------------------------.
             |   ASSOCIATIVE MEMORY    |
             | finds nearest neighbor |
             |  among stored vectors  |
             '-------------------------'
                         |
                         v
                 +1 +1 -1 -1 ... -1 -1  = A
```

ENCODING SEQUENCES WITH PERMUTATION r

Text as an example

The 10K seed vectors **A**, **B**, **C**, ... represent letters of the alphabet a, b, c, ...

The sequence 'ab' can be encoded as follows

Start with the one-letter "sequence" for 'a':

A

and extend it to 'ab' with permute and multiply:

$$\mathbf{AB} = r(\mathbf{A})*\mathbf{B}$$

We can further extend it to 'abc' with permute and multiply, and so on ...

$$\mathbf{ABC} = r(\mathbf{AB}) * \mathbf{C}$$
$$= r(r(\mathbf{A}) * \mathbf{B}) * \mathbf{C}$$
$$= r(r(\mathbf{A})) * r(\mathbf{B}) * \mathbf{C}$$

$$\mathbf{ABCD} = r(r(r(\mathbf{A}))) * r(r(\mathbf{B})) * r(\mathbf{C}) * \mathbf{D}$$

...

NOTE. This encoding demonstrates the *distributivity of permutat*ion over multiplication; permutations distributes also over addition.

This encoding has been used for *N*-grams in experiments on language-identification

EXAMPLE 2: 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

. Demonstrated online

COMPUTING a PROFILE

Step 1. Choose 27 random **seed vectors** represent the LETERS

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 **permute and multiply**

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

```
                  Rotation of coordinates
        ...................>....................
        :  :                                :  :
        :  :                                :  :
 T =  (+1 -1 -1 +1 -1 -1 . . . +1 +1 -1 -1) +  -
    H =  (+1 -1 +1 +1 +1 +1 . . . +1 -1 +1 -1) +
       E =  (+1 +1 +1 -1 -1 +1 . . . +1 -1 +1 +1)
------------------------------------------------
 THE   =    (+1 +1 -1 +1 . . . . . . +1 +1 -1 -1)
```

In symbols:

THE = *r*(*r*(**T**)) * *r*(**H**) * **E**

where

r is 1-position rotate (it's a permutation)
* is coordinatewise multiplication

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. For example, **HET.THE** ≈ 0

Step 3. Accumulate PROFILE VECTORS

Add all trigram vectors of a text into a 10K 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

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

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

Step 4. Test the profiles of 21 EU languages

. Similarity between vectors/profiles: Cosine

$$\cos(\mathbf{X},\mathbf{X}) = 1$$
$$\cos(\mathbf{X},\mathbf{Y}) = 0 \text{ if } \mathbf{X} \text{ and } \mathbf{Y} \text{ are orthogonal}$$

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

Italian
* *Romanian
Portuguese
* *Spanish
*Slovene *French
*Bulgari *Czech
*Slovak *English
*Greek
*Polish *Lithuanian
*Latvian
*Estonian
* *Finnish
Hungarian

*Dutch
*Danish *German
*Swedish

Step 4b. The language profiles were compared to the profiles of 21,000 test sentences (1,000 per language). The best match agreed with the correct language 97.3% of the time

The experiment was done in one pass and took less than 8 minutes on a laptop computer

Step 5. The 10K profile for English, **Engl**, was QUERIED for the letter most likely to follow "*th*". It was "*e*", with *space*, "*a*", "*i*", "*r*", and "*o*" the next-most-likely, in that order

. Form a query vector: $\mathbf{q} = r(r(\mathbf{T})) * r(\mathbf{H})$
. Query with multiply: $\mathbf{x} = \mathbf{q} * \mathbf{Engl}$
. Find closest letter vectors:

$$\mathbf{x} \approx \mathbf{E}, \#, \mathbf{A}, \mathbf{I}, \mathbf{R}, \mathbf{O}$$

`

SUMMARY of the ALGORITHM

. Start with random 10,000-D 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 the cosine

Note the **simplicity** and **scaling** of the algorithm, in contrast to computing the exact histogram for each test sentence and comparing it to the histograms of the languages.

With 27 letters there are 19,683 possible trigrams to keep track of, for tetragrams the number is 531,441 and for pentagrams it is 14,348,907. The language experiment was done also with tetragrams (they performed the best) and pentagrams with no added complexity (the runtime and memory use were the same).

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.

SUMMARY

High-dimensional vectors have a math of their own

It is subtle and counterintuitive

It can be understood in terms that are familiar to us from math with numbers:
- addition
- multiplication
- inverse
- distributivity

Permutations give it added power

Computing with high-dimensional vectors has grown out of attempts to model cognition (perception, memory, learning, language, concepts, the mind)

THEnd

(but really the beginning: Neurosc 299, 1 Sep 2021)