# Representing structured relational data in Euclidean vector spaces

Tony Plate

**tplate@acm.org**

**http://www.d-reps.org**

**October 2004**



# Overview

- A general method for representing structured relational data in finite dimensional Euclidean vector spaces (“flat” vectors)
- Interesting properties: similarity, transformations, etc.
- Learning
- Relationships to structural kernel methods
- Potential large-scale benchmark problems

## Motivation

- Intuition is that connectionist-style models *feel* more right than symbolic models
  - flat vector representations can capture gradations of meaning
  - have techniques for learning
    - learning the flat vector representations
    - learning to perform tasks using those representations
- Fodor & Pylyshyn had some valid points; claims:
  - compositionality is important
    - recursion, role-filler bindings
  - no good connectionist representation for compositional structure
  - any connectionist representation will be just implementation details
- Questions I tried to answer
  - how can compositional structure be represented in flat vectors?
  - is this anything more than implementation details in a symbolic system?



## How to represent structured relational data in Euclidean vector spaces

- Vector space representations typically have:
  - vector addition (superposition)
  - scalar multiplication
  - distance function
  - normalization (sometimes)
- To represent structure, also need:
  - vector multiplication (for binding)

## Binding to represent structure

- Consider representing relational structure, e.g., (bit fido john)
- Superposition (I.e., **bite+john+fido**) not suitable -- looses binding info
- With a binding operation (as well as superposition) can use role filler bindings:
  **agent * fido + object * john**
- Can also represent sequences, e.g., abc:
  **p1*a+p2*b+p3*c** or **a+a*b+a*b*c**



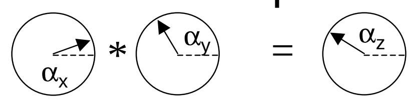
## Properties of binding operation

- **a*b** must not be similar to **a** or **b** (in contrast to superposition)
- nice if **a*b** is similar to **a*b'** to extent that b is similar to **b'**
- want inverse so that $\mathbf{a}^{-1}\mathbf{*}(\mathbf{a*b})\ \mathbf{=a}$ (or approx)
- can have arbitrary numbers of roles in relations
- if **a*b** is a vector of same dimension as **a** and **b** can have recursive relations:
  (believe mary (bit fido john))

# Implementing binding (multiplication)

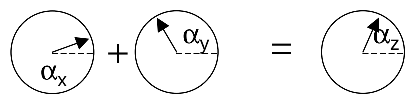
- Holographic Reduced Representations in frequency space
- Rotor values ($x_i$ is a complex value)
- Useful normalization is unit magnitude $|x_i|=1$
- Vector multiplication is elementwise multiplication -- addition of phase angles

  **z = x * y** if $z_i = x_i * y_i$




$$\alpha_z = \alpha_x + \alpha_y$$

- Superposition is elementwise addition -- corresponds to midpoint of phase angles






# Summary of operations

- Vectors: elements are phase angles
- Superposition: averaging of phase angles
- Binding: addition of phase angles
- Normalization: all elements have unit magnitude
- Similarity: sum of cosines of elementwise angle differences
- Represent relational structure by superposition of role-filler bindings
- Decode structures using inverse of binding

# Similar systems

- Binary Spatter Codes (Pentti Kanerva 1996)
- Multiplicative binding (Ross Gayler 1998)
- APNNs & Context dependent thinning (Dmitri Rachkovskij & Ernst Kussul 2001)



# Interesting properties

- Similarity -- design representation to have desired similarity properties
- Fast (linear time) methods for:
  - Similarity (dot-product)
  - Structure transformations
  - Identification of corresponding entities

## Designing similarity

- "Natural" similarity over relational structure:
  - similar entities contributes to similarity
  - similar structure contributes to similarity
  - similar entities in similar roles contributes to similarity
- Get this with role-filler binding representations, if we add in fillers:

  **bite + fido + john + agent*fido + object*john**
- Can use this scheme recursively
- These representations for hierarchical structures can model human performance on analog recall
  - similarity is sensitive to both contents and structure
  - see Plate 2003



## Fast operations

- Similarity – dot product similarity reflects structural and surface similarity
- Can do structure transformations with learning or with straightforward vector algebra

  $(\mathrm{rel}_1\ x\ y) \rightarrow (\mathrm{rel}_2\ y\ x)$

  $\mathbf{agt_1{*}x{+}obj_1{*}y} \rightarrow \mathbf{agt_2{*}y{+}obj_2{*}x}$

  transforming vector is ($\mathbf{agt_1^{-1}{*}obj_2{+}obj_1^{-1}{*}agt_2}$)
- Identification of corresponding entities

  E.g., x is involved in structure A, what is in the corresponding position in structure B?

  $(x^{-1}{*}A)^{-1}{*}B$

# Learning

- Theoretically easy to incorporate fixed HRR operations in neural networks
- Learning representations of tokens from data (cf sequence learning Plate 2003)
- Learning structural properties of data
- Learning transformations (cf Jane Neumann's work, Chris Eliasmith)



# Convolution kernels

- Introduced by David Haussler 1999
- Provide a similarity measure for structured objects
- Examples:
  - String kernel – similarity of two strings is proportional to number of common substrings
  - Tree kernel – similarity of two trees is proportional to number of common subtrees
- Similarity measure can be expressed as dot-product in very high-dimensional space

## Convolution kernels (continued)

- This similarity measure can be computed in polynomial time (dynamic programming)
- Using Support Vector Machines, can find linear classifiers in the high-dimensional space (without ever having to explicitly construct vectors in that space)
- Interesting large scale applications: document classification, parsing, gene analysis



## Relationship between HRRs & convolution kernels

- HRRs can approximate a convolution kernel
- Consider an all-substring kernels (substrings are ordered but non-contiguous)
- Two strings: abc & adc
- All-substring convolution kernel contains 11 features:
  - abc, adc, ab, ac, bc, ad, dc, a, b, c, d
- abc: `10111001110`
- adc: `01010111011` (overlap is 3: ac, a, c)

# HRR similarity $\approx$ convolution kernel

- With HRRs, use uncorrelated high-d vectors for a, b, c, d (with Euclidean length 1)
- Represent strings as superposition of bindings of all substrings (use a non-commutative form of convolution so that $a*b \neq b*a$)

$$\begin{aligned}
abc &= a*b*c + a*b + a*c + b*c + a + b + c \\
adc &= a*d*c + a*d + a*c + d*c + a + d + c \\
abc \cdot adc &= (a*b*c + \ldots + b + c) \cdot (a*d*c + a*d + \ldots + c) \\
&\approx a*c \cdot a*c + a \cdot a + c \cdot c \\
&\approx 3
\end{aligned}$$

(terms like $a*c \cdot a*d$ & $a*c \cdot a$ & $c \cdot a$ are all approximately zero because of initial choice of vectors and randomizing property of convolution)



# Comparison of HRR similarity and convolution kernels

- Convolution kernel:
  - each combinatorial feature in a single numeric element in a very high-d vector (discrete similarity of features)
  - vectors in high-d space usually not explicitly computed
- HRR similarity:
  - use wide pattern to represent each combinatorial feature
  - should use relatively few combinatorial features
  - computing dot-product similarity very fast
  - continuous similarity comes for free:
    if a is similar to a', then a*b will be similar to a'*b
  - possible to use neural-net learning to learn representations of base vectors (by back propagating through convolution)
  - although HRR similarity only approximates the convolution kernel, it is still a valid kernel function for SVM because it is a dot product

# Large scale applications

- Working on real applications has a number of advantages:
  - focuses attention on important aspects of techniques
  - allows meaningful comparison among very different approaches
  - helps to promote good approaches
- Lots of data is now available!
  - Language (textual) data
  - Biological data (genetic and gene expression)



# Available data sets and applications

- Text classification
  - Reuters 21578: publically available, widely used
  - TREC data sets: yearly conferences since early 90’s, very large data sets, lots of experiments, not free
- Word sense disambiguation
  - “Senseval” project: publically available data, 3 conferences
- Parsing, e.g., using Penn Treebank and annotations
- Part-of-speech (POS) tagging: tons of data, many good systems (not perfect though)
- Predicate argument classification (e.g., PropBank project, 1m words)
- Note that SVM techniques have been applied to all the above
- Another possible application: help system for an open-source software project, e.g., statistical system R: hundreds of add-on packages, thousands of functions

# Example

- Word sequence kernels (Cancedda, Gaussier, Croutte & Renders, 2003) (also see character string kernels Lodhi, Saunders, Shawe-Taylor, Christianini & Watkins, 2002)
- Applied to subset of Reuters 21578 (newswire documents with categories)
- Performance measure was recall & precision & overall measures on text classification
- Kernels were words & word substrings, with gaps
- Examples of questions addressed:
  - when using pairs of words as features, does order matter?
    - using order impairs performance
  - do higher order features (word pairs, triples) help?
    - help precision, hurt recall, hurt overall



# END!

## Connections: Real HRRs≡Phase HRRs

- HRRs with real values (n elements in vector)
  - normalization: normalize whole vector to have Euclidean length of one
  - element values normally distributed with mean 0 and variance 1/n
  - superposition: elementwise addition
  - binding: circular convolution (each element of **z** is the sum of n pairs of elements of **x** and **y**)
  - similarity: dot-product
  - If no normalization, phase HRRs (freq. domain) are equivalent to real HRRs (spatial domain) (via FFT)



## Connections: Binary spatter code { Quantized Phase HRRs

- Kanerva's Binary spatter code (1996)
  - binary vector elements, 50% density
  - superposition: majority function (could involve several arguments)
  - binding: exclusive-OR
  - similarity: number of matching elements
  - equivalent to phase HRRs quantized to two values: +1 and –1

## Other ways of implementing binding

- Tensor products (Smolensky 1990)
  - binding is outer-product
- RAAMs (Pollack 1990)
  - roles are weight matrices
  - fillers are activation vectors
  - binding is matrix-vector multiplication
- Random Sigma-Pi networks (Plate 1994, 1998)
  - element of z is sum of n randomly selected pairs of x-y elements



## Domain independent procedures for feature construction

- LSA (Latent Semantic Analysis) (Landauer, Deerswester, Dumais & colleagues)
- Constructs vector reps for words such that similar words are represented by similar vectors
- Based on principle component analysis of raw document frequency vectors: e.g. 8 documents
  - tiger: `0 0 1 0 0 2 0 0`: occurred once in 3rd document and twice in 6th document
  - lion: `0 1 1 0 0 1 0 0`: occurred once in 2nd, 3rd & 6th document

## LSA continued

- Obtain more compact vector reps for words in terms of the principal components of raw feature vectors

documents

words

1001020…
0010021…
0101210…

- Ways to generalize:
  - Elements of raw representation vectors are contexts in which objects may appear
  - Use other context matrices, e.g., co-occurrence matrix



## Summary

- Can represent relational structure (& nested structure) in Euclidean vector space
- Representations for composite objects are constructed compositionally
- Can capture similarity of composite objects – essential for learning
- A wide range of representational techniques
- Domain independent techniques for feature construction are available