Tensor Product Representations and Holographic Reduced Representations

Smolensky, 1990 & Plate, 1991

Tiwalayo Eisape, Joey Velez-Ginorio, Pedro Colon-Hernandez
{eisape, joeyv, pe2517}@mit.edu

Neuro-symbolic Models for NLP (6.884), October 2, 2020
Outline

1. Introductions (us + 3 others) (11:35 - 11:40)
3. Break out room (11:55 - 12:10 mins)
4. Discussion (12:10 - 12:20 mins)
5. [Early] Break (12:20 - 12:35)
6. TPR tutorial (12:35 - 12:50)
7. Discussion (12:50 - 12:55)
8. TPR Shortcomings; HRRs (12:55 - 1:10)
9. Discussion (1:10 - 1:25)
1. Introductions (us + 3 others) (11:35 - 11:40)
3. Break out room (11:55 - 12:10 mins)
4. Full Group Discussion (12:10 - 12:20 mins)
5. [Early] Break (12:20 - 12:35)
6. TPR tutorial (12:35 - 12:50)
7. Full Group Discussion (12:50 - 12:55)
8. TPR Shortcomings; HRRs (12:55 - 1:10)
9. Full Group Discussion (1:10 - 1:25)

Outline

1. Is variable binding necessary?
2. Do humans use a TPR-like mechanism?
3. Do current models approximate faithfulness?
4. Small group technical questions
Tensor Product Representations - why?

A one-sentence summary of the implications of this view for AI:

connectionist models may well offer an opportunity to escape the brittleness of symbolic AI systems ...

... This paper offers an example of what such a collaboration might look like.
Tensor Product Representations - why?

Jay is loved by Kay. Who loves Jay? Kay.
- Jay in role: subject of passive sentence
- Jay in role: object of wh-question
- Kay in role: object of passive by-phrase
- Kay in role: answer to wh-question

[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]
Tensor Product Representations - why?

Jay is loved by Kay. Who loves Jay? Kay.

- **Jay** in role: subject of passive sentence
- **Jay** in role: object of *wh*-question
- **Kay** in role: object of passive *by*-phrase
- **Kay** in role: answer to *wh*-question

[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]
Tensor Product Representations - why?

Jay is loved by Kay. Who loves Jay? Kay.

- Jay in role: subject of passive sentence
- Jay in role: object of wh-question
- Kay in role: object of passive by-phrase
- Kay in role: answer to wh-question

[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]
Tensor Product Representations - what?

Representing Structured Objects

(1) Decomposing the structures via roles | (2) representing variable/value bindings | (3) representing conjunctions

[Smolensky 1990, pg. 169]
Tensor Product Representations - what?
Tensor Product Representations - what?

1. Decomposing the structures via roles
Tensor Product Representations - what?

1. Decomposing the structures via roles
Tensor Product Representations - what?

(2) representing variable/value bindings
## Tensor Product Representations - what?

<table>
<thead>
<tr>
<th>Filler</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) representing variable/value bindings
Tensor Product Representations - what?

(Soulos et al. 2019)

(1) Decomposing the structures via roles
(2) Representing variable/value bindings
(3) Representing conjunctions
Tensor Products

‘Faithful’ 👶🏾 Tensor Product Representations
Tensor Products

‘Faithful’ 🧵 Tensor Product Representations
Faithfulness
Faithfulness
Faithfulness
Faithfulness
Faithfulness

New Role, New Representation!
Faithfulness

New Role, New Representation!
Faithfulness

Orthogonality
Theorem 3.3, Section 3.2

New Role, New Representation!

Linear Independence
Definition 2.8, Section 2.2.2
Tensor Products

‘Faithful’ 🤡 Tensor Product Representations

Graceful Decay 😈
Variable Binding

1. Is Variable Binding Necessary?
2. Do humans use a TPR-like mechanism?
3. Do current models approximate faithfulness?
4. Small group technical questions

Gary Marcus @GaryMarcus · Feb 6, 2018
I completely agreed, @tdietterich! Lots of cases where **variable binding** is absolutely necessary. No **binding**, no AGI.

Thomas G. Dietterich @tdietterich · Feb 6, 2018
Replying to @tdietterich @jahendler and @GaryMarcus
There are lots of cases where binding appears to be necessary. Ex 1: If you put X into your pocket and then walk to work, you will be able to take X out of your pocket at work. Ex 2: If I ask you query X and you know X, you will tell me X, for all X.
Break
TPR Tutorial
TPR Tutorial

(1) Symbolic Structure
TPR Tutorial

(1) Symbolic Structure
(2) Encoding w/ TPRs
TPR Tutorial

(1) Symbolic Structure
(2) Encoding w/ TPRs
(3) Representation Proofs
(1) Symbolic Structure: Give, the programming language
(1) Symbolic Structure : Give, the programming language

Syntax

\[ p ::= \text{(Give } p) \mid \square \]
(1) Symbolic Structure: Give, the programming language

Syntax

\[ p ::= \text{Give } p \mid \square \]

Examples

(Give \square)
(Give (Give \square))
(Give (Give (Give \square)))
(1) Symbolic Structure: Give, the programming language

**Syntax**

\[ p ::= \text{(Give } p \text{)} \mid \square \]

**Examples**

(Give \square)

(Give (Give \square))

(Give (Give (Give \square))}

**Semantics**

\[ \text{(Give } p \text{)} \rightarrow p \]

\[ p \rightarrow p' \]

\[ \text{(Give } p \text{)} \rightarrow \text{(Give } p' \text{)} \]
(1) **Symbolic Structure**: Give, the programming language

**Syntax**

\[ p ::= (\text{Give } p) | \square \]

**Semantics**

\[
\begin{align*}
\text{(Give } p) & \rightarrow p \\
p & \rightarrow p' \\
\text{(Give } p) & \rightarrow (\text{Give } p')
\end{align*}
\]

**Examples**

- \((\text{Give } \square) \rightarrow \square\)
- \((\text{Give } (\text{Give } \square)) \rightarrow \square\)
- \((\text{Give } (\text{Give } (\text{Give } \square))) \rightarrow \square\)
(2) Encoding w/ TPRs : Give, the programming language

A TPR is a mapping, $[p] : \text{Give}_4 \rightarrow \mathbb{R}^{4 \times 3}$, from a set of symbols to a vector space via filler/role decompositions. Here, $\text{Give}_4$ denotes the set of all Give programs up to length 4.
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \([p] : \text{Give}_4 \mapsto \mathbb{R}^{4 \times 3}\), from a set of symbols to a vector space via filler/role decompositions. Here, \(\text{Give}_4\) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ [], (\text{Give } []), (\text{Give (Give } [])), (\text{Give (Give (Give } [])))\}
\]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \( p : \text{Give}_4 \rightarrow \mathbb{R}^{4x3} \), from a set of symbols to a vector space via filler/role decompositions. Here, Give\(_4\) denotes the set of all Give programs up to length 4.

\[ \text{Give}_4 = \{ , (\text{Give } ) , (\text{Give (Give } )) , (\text{Give (Give (Give } )))\} \]

\[ \mathbb{R}^{4x3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 2 & 2 & 1 \\ 1 & 1 & 3 \end{bmatrix}, \ldots \} \]
A TPR is a mapping, \[ p \rightarrow \mathbb{R}^{4 \times 3} \], from a set of symbols to a vector space via filler/role decompositions. Here, \( \text{Give}_4 \) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ \square, (\text{Give} \ \square), (\text{Give} (\text{Give} \ \square)), (\text{Give} (\text{Give} (\text{Give} \ \square))) \}
\]

\[
\mathbb{R}^{4 \times 3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 3 \\ 2 & 0 & 0 \end{bmatrix}, \ldots \}
\]

Fillers

\[ f = \{ i_1, i_2, i_3, i_4 \} \]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \( \llbracket p \rrbracket : \text{Give}_4 \rightarrow \mathbb{R}^{4 \times 3} \), from a set of symbols to a vector space via filler/role decompositions. Here, \( \text{Give}_4 \) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ \Box, (\text{Give} \Box), (\text{Give} (\text{Give} \Box)), (\text{Give} (\text{Give} (\text{Give} \Box))) \}
\]

\[
\mathbb{R}^{4 \times 3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix}, \ldots \}
\]

Fillers
\[
f = \{ i_1, i_2, i_3, i_4 \}
\]

Roles
\[
r = \{ \text{Give}, \Box, \varepsilon \}
\]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \( \llbracket p \rrbracket : \text{Give}_4 \mapsto \mathbb{R}^{4 \times 3} \), from a set of symbols to a vector space via filler/role decompositions. Here, \( \text{Give}_4 \) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ \Box, (\text{Give } \Box), (\text{Give } (\text{Give } \Box)), (\text{Give } (\text{Give } (\text{Give } \Box)))\}
\]

\[
\mathbb{R}^{4 \times 3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 3 \\ 2 & 0 & 1 \end{bmatrix}, \ldots \}
\]

*Fillers*

\[ f = \{ i_1, i_2, i_3, i_4 \} \]

\[
(\text{Give } (\text{Give } (\text{Give } \Box))) = (i_1:\text{Give}) \land (i_2:\text{Give}) \land (i_3:\text{Give}) \land (i_4:\Box)
\]

*Roles*

\[ r = \{ \text{Give}, \Box, \varepsilon \} \]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \([p] : \text{Give}_4 \mapsto \mathbb{R}^{4x3}\), from a set of symbols to a vector space via filler/role decompositions. Here, \(\text{Give}_4\) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ \square, (\text{Give} \ \square), (\text{Give} \ (\text{Give} \ \square)), (\text{Give} \ (\text{Give} \ (\text{Give} \ \square)))\}
\]

\[
\mathbb{R}^{4x3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 1 & 3 \\ 2 & 0 & 1 \end{bmatrix}, \ldots \}
\]

- **Fillers**
  \[f = \{ i_1, i_2, i_3, i_4 \}\]
  
  \[
  (\text{Give} \ (\text{Give} \ (\text{Give} \ \square))) = (i_1 : \text{Give}) \land (i_2 : \text{Give}) \land (i_3 : \text{Give}) \land (i_4 : \square)
  \]

- **Roles**
  \[r = \{ \text{Give}, \square, \varepsilon \}\]
  
  \[
  (\text{Give} \ (\text{Give} \ \square)) = (i_1 : \varepsilon) \land (i_2 : \text{Give}) \land (i_3 : \text{Give}) \land (i_4 : \square)
  \]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \([p] : \text{Give}_4 \rightarrow \mathbb{R}^{4 \times 3}\), from a set of symbols to a vector space via filler/role decompositions. Here, \(\text{Give}_4\) denotes the set of all \(\text{Give}\) programs up to length 4.

\[
\text{Give}_4 = \{ \boxplus, (\text{Give } \boxplus), (\text{Give} (\text{Give } \boxplus)), (\text{Give} (\text{Give} (\text{Give } \boxplus)))\} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \}
\]

\[
\mathbb{R}^{4 \times 3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix} \}
\]

Fillers

\(f = \{ i_1, i_2, i_3, i_4 \}\)

Roles

\(r = \{ \text{Give}, \boxplus, \varepsilon \}\)

\((\text{Give} (\text{Give} (\text{Give} \boxplus)))) = (i_1: \text{Give}) \land (i_2: \text{Give}) \land (i_3: \text{Give}) \land (i_4: \boxplus)

\((\text{Give} (\text{Give} \boxplus)) = (i_1: \varepsilon) \land (i_2: \text{Give}) \land (i_3: \text{Give}) \land (i_4: \boxplus)

\((\text{Give} \boxplus) = (i_1: \varepsilon) \land (i_2: \varepsilon) \land (i_3: \text{Give}) \land (i_4: \boxplus)\)
Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, \( \llbracket p \rrbracket: \text{Give}_4 \rightarrow \mathbb{R}^{4x3} \), from a set of symbols to a vector space via filler/role decompositions. Here, \( \text{Give}_4 \) denotes the set of all Give programs up to length 4.

\[
\text{Give}_4 = \{ \quad , \quad , \quad , \quad \}
\]

\[
\mathbb{R}^{4x3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 2 & 1 \\ 1 & 1 & 3 \\ 1 & 0 \end{bmatrix}, \quad \ldots \}
\]

Fillers
\( f = \{ [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 1] \} \)

Roles
\( r = \{ \text{Give}, \quad , \quad , \quad \} \)

\[ \llbracket (\text{Give} (\text{Give} (\text{Give}))) \rrbracket = \llbracket (i_1: \text{Give}) \land (i_2: \text{Give}) \land (i_3: \text{Give}) \land (i_4: \quad ) \rrbracket \]
(2) Encoding w/ TPRs: Give, the programming language

A TPR is a mapping, $[p] : \text{Give}_4 \mapsto \mathbb{R}^{4\times3}$, from a set of symbols to a vector space via filler/role decompositions. Here, $\text{Give}_4$ denotes the set of all Give programs up to length 4.

$$\text{Give}_4 = \{ \Box, (\text{Give } \Box), (\text{Give } (\text{Give } \Box)), (\text{Give } (\text{Give } (\text{Give } \Box))) \}$$

$$\mathbb{R}^{4\times3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix}, \ldots \}$$

**Fillers**

$$f = \{ i_1, i_2, i_3, i_4 \}$$

$$[((\text{Give } (\text{Give } (\text{Give } \Box))))] = [\langle i_1 : \text{Give} \rangle \land \langle i_2 : \text{Give} \rangle \land \langle i_3 : \text{Give} \rangle \land \langle i_4 : \Box \rangle]$$

$$= (i_1 \otimes \text{Give}) + (i_2 \otimes \text{Give}) + (i_3 \otimes \text{Give}) + (i_4 \otimes \Box)$$

**Roles**

$$r = \{ \text{Give}, \Box, \varepsilon \}$$
(2) Encoding w/ TPRs : Give, the programming language

A TPR is a mapping, \[ p \rightarrow \mathbb{R}^{4 \times 3} \], from a set of symbols to a vector space via \text{filler/role} decompositions. Here, \( \text{Give}_4 \) denotes the set of all \text{Give} programs up to length 4.

\[
\text{Give}_4 = \{ \, \text{ }, (\text{Give } \text{ }) \, \text{, (Give (Give } \text{ )}), \text{ (Give (Give (Give } \text{ )))}) \, \}
\]

\[
\mathbb{R}^{4 \times 3} = \{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix}, \ldots \}
\]

\text{Fillers} \quad f = \{ \begin{array}{cccc}
[1] & [0] & [0] & [0] \\
[0] & [1] & [0] & [0] \\
[0] & [0] & [1] & [0] \\
[0] & [0] & [0] & [1]
\end{array} \}

\text{Roles} \quad r = \{ \text{Give} , \text{ }, \varepsilon \}

\[
\lbrack (\text{Give (Give (Give } \text{ )))} \rbrack = \lbrack (i_1 : \text{Give}) \wedge (i_2 : \text{Give}) \wedge (i_3 : \text{Give}) \wedge (i_4 : \text{ }) \rbrack
\]

\[
= (i_1 \otimes \text{Give} ) + (i_2 \otimes \text{Give} ) + (i_3 \otimes \text{Give} ) + (i_4 \otimes \text{ })
\]

\[
= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}
\]
Theorem. The following linear transformation is a representation of the instruction Give.

\[
\text{Give} : \sum_i f_i \otimes r_i \mapsto \sum_i f_i \otimes r_i
\]
(3) Representation Proofs: Give, the programming language

**Theorem.** The following linear transformation is a representation of the instruction Give.

\[
\text{Give} : \sum_i f_i \otimes r_i \mapsto \sum_i f_i \otimes r_i
\]

**Proof.**
Recall that w/ TPRs we encode Give programs as conjunctions of filler/role decompositions, i.e. \( \llbracket p \rrbracket = \sum_i f_i \otimes r_i \). Additionally, recall that: \((\text{Give } p) \rightarrow p\)

\[
\llbracket (\text{Give } p) \rrbracket = \llbracket p \rrbracket
= \sum_i f_i \otimes r_i
= \text{Give } \sum_i f_i \otimes r_i
= \text{Give } \llbracket p \rrbracket
\]

\(\square\)
Discussion

- How does this scale to larger programs in Give?
- What if our programming language was more complicated?
- Other thoughts...
Benefits & Shortcomings of Tensor Decomposition

+ No impositions on structure
+ Faithful
+ Variable binding
- Scaling up can be memory and compute demanding
  ○ Using ConceptNet as an example, ~4M nodes, ~40 relations might need to play around with pretty large tensors
Holographic Reduced Representations

- Use Circular Convolutions and Correlations to associate/disassociate vectors that represent structures
- Requires a reconstruction system to sort through the noise
- Circular Conv. and Circular Corr. can be manipulated to query structure
Representations with HRR

Sequences

\[ s_{abc} = a + a \otimes b + a \otimes b \otimes c \]
\[ s_{de} = d + d \otimes e \]
\[ s_{fgh} = f + f \otimes g + f \otimes g \otimes h \]

\[ S(abcdefgh) = s_{abc} + s_{abc} \otimes s_{de} + s_{abc} \otimes s_{de} \otimes s_{fgh} \]
Representations with HRR

Sequences

Variable Binding

\[ \tilde{t} = \tilde{x} \circ \tilde{a} + \tilde{y} \circ \tilde{b}. \]

Binding a to X and b to Y
Representations with HRR

Sequences

Running frame: Spot runs
\[ t_{\text{running}} = l_{\text{run}} + r_{\text{agent}} @ f_{\text{spot}} \]

Variable Binding

Seeing frame: Dick saw Spot run
\[ t_{\text{seeing}} = l_{\text{see}} + r_{\text{agent}} @ f_{\text{dick}} + r_{\text{object}} @ t_{\text{running}} \]
\[ = l_{\text{see}} + r_{\text{agent}} @ f_{\text{dick}} + r_{\text{object}} @ (l_{\text{run}} + r_{\text{agent}} @ f_{\text{spot}}) \]
Example: Filling a frame

Frame:

Job Application:
- Name
- Date

Filler
- September 1, 2020
## Example: Filling a frame

<table>
<thead>
<tr>
<th>Frame:</th>
<th>Job Application</th>
<th>0.35</th>
<th>0.28</th>
<th>0.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>-0.19</td>
<td>-0.14</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>-0.22</td>
<td>0.04</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>Filler:</td>
<td>September 1, 2020</td>
<td>0.06</td>
<td>0.05</td>
<td>-0.16</td>
</tr>
</tbody>
</table>
Example: Filling a frame

Binding Date & Filler

C[0]=0.1*0.05+-0.16*0.04+-0.22*0.06+=-0.0146

C: -0.0146
Example: Filling a frame

Binding Date & Filler

\[ C[1]=0.1\times-0.16+0.05\times-0.22+0.04\times0.06=-0.0246 \]

| C: | -0.0146 | -0.0246 |

<table>
<thead>
<tr>
<th>Date</th>
<th>0.06</th>
<th>0.05</th>
<th>-0.16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

September 1, 2020
Example: Filling a frame

Pairing Job Application + Date Field

\[ C[2] = 0.1 \times 0.06 + 0.05 \times 0.04 - 0.16 \times -0.22 = 0.0432 \]

| C:  | -0.0146 | -0.0246 | 0.0432 |

Date

-0.22 0.04 0.10

September 1, 2020
Example: Filling a frame

C: 

-0.0146  -0.0246  0.0432

C: {Date: September 1, 2020}
Example: Filling a frame

Add Name:
C: {Date: September 1, 2020} + Name
C':{Date: September 1, 2020, Name}
Example: Filling a frame

C': {Date: September 1, 2020, Name}
Example: Filling a frame

Add in Frame Label

C': {Date: September 1, 2020, Name}+ Job Application
C'':{Job Application: Date: September 1, 2020, Name}
Example: Filling a frame

C’': \{\text{Job Application: Date:September 1,2020, Name}\}

Keep in mind representations are stored in a distributed manner.
We used the “decoder” implicitly to clean the noise.
Our representations are the result of an FFT.
Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

January, 1, 2021
Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

January, 1, 2021
Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

Date: January 1, 2021

January, 1, 2021
Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

January, 1, 2021
Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

January, 1, 2021
Benefits of Holographic Representations

- Format for the two input vectors is not specified, only independently distributed
- **Space Efficiency:** you just need the 2 vectors rather than the whole Tensor, result is the same size as the input
- Can be calculated in $O(n \log n)$ with FFT
- HRRs could retain ambiguity while processing ambiguous input (New York as City and as Name)
- Easy analysis of capacity, scaling and generalization
Shortcomings of Holographic Representations

- Decoder/cleaner must store all the possible outputs. If it knows everything, then why not find a way to exploit it?
- Is the decoder static? How would you add some new domain?
- Elements of each vector must be independently distributed, but have meaningful features
- Hit until you decode the correct thing?
- Some operations to decode require additional machinery (recursive)
Encoding Methods?

TPR

HRR

OTHER
Encoding Methods?

- HRR
- TPR
- all other encodings of symbols in vector spaces
[END]