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

Introductions (us + 3 others) (11:35 - 11:40)

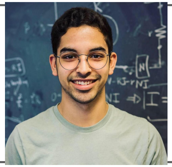
TPRs - why/what? (11:40 - 11:55)

Break out room (11:55 - 12:10 mins)

Discussion (12:10 - 12:20 mins)

[Early] Break (12:20 - 12:35)

TPR tutorial (12:35 - 12:50)



Discussion (12:50 - 12:55)

TPR Shortcomings; HRRs (12:55 - 1:10)

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

Introductions (us + 3 others) (11:35 - 11:40)

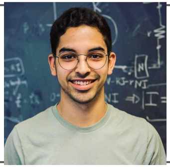
TPRs - why/what? (11:40 - 11:55)

Break out room (11:55 - 12:10 mins)

Full Group Discussion (12:10 - 12:20 mins)

[Early] Break (12:20 - 12:35)

TPR tutorial (12:35 - 12:50)



Full Group Discussion (12:50 - 12:55)

TPR Shortcomings; HRRs (12:55 - 1:10)

Full Group Discussion (1:10 - 1:25)

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**.

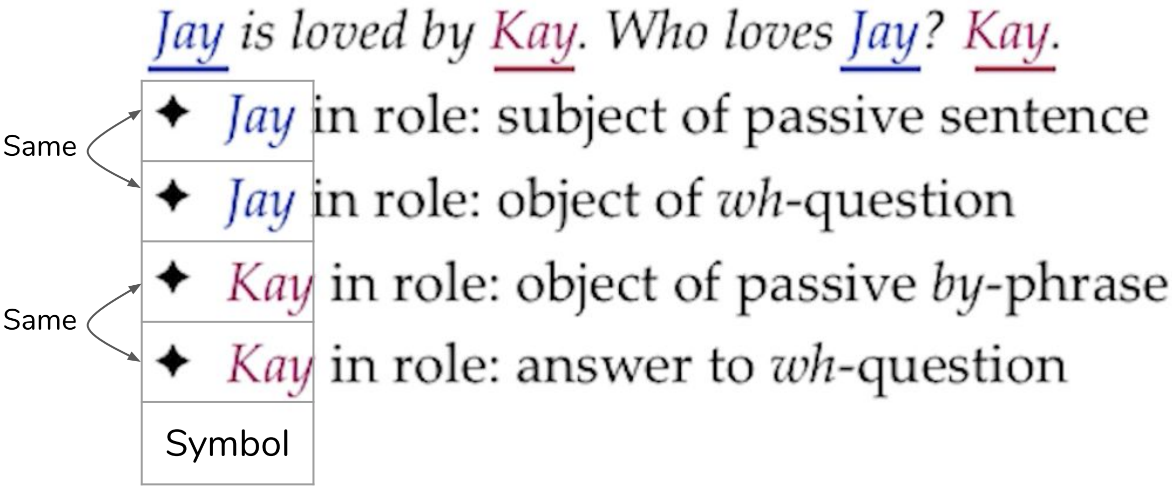
Tensor Product Representations - *why?*

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

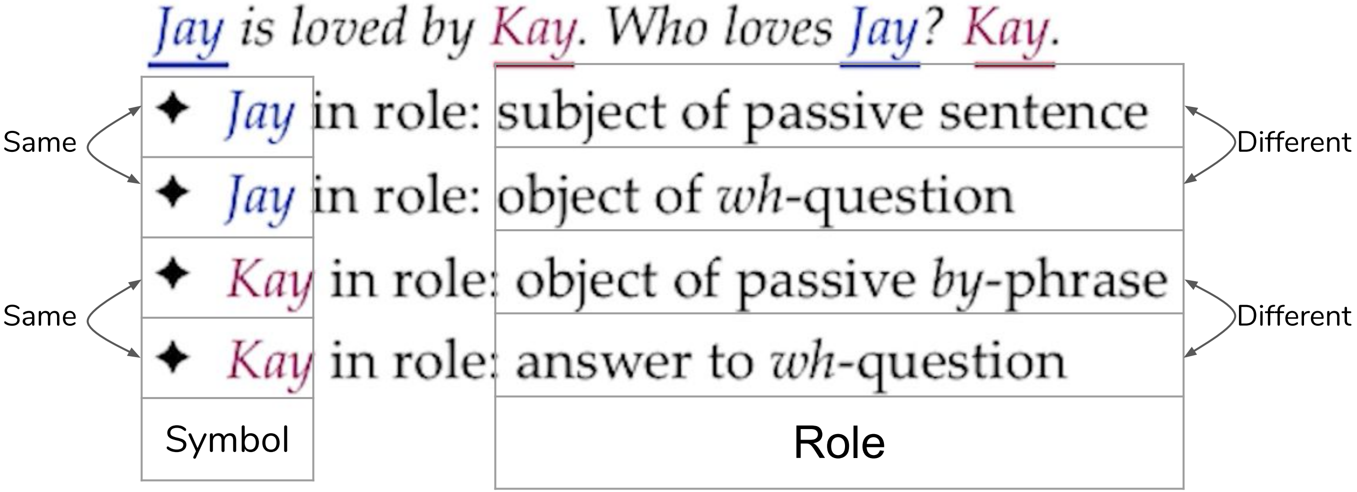
[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]

Tensor Product Representations - *why?*



[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]

Tensor Product Representations - *why?*



[Paul Smolensky HLAI Keynote (2019); Newell, A. (1980)]

Tensor Product Representations - *what*?

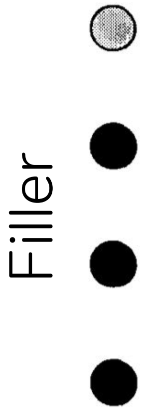
Representing Structured Objects

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

[Smolensky 1990, pg. 169]

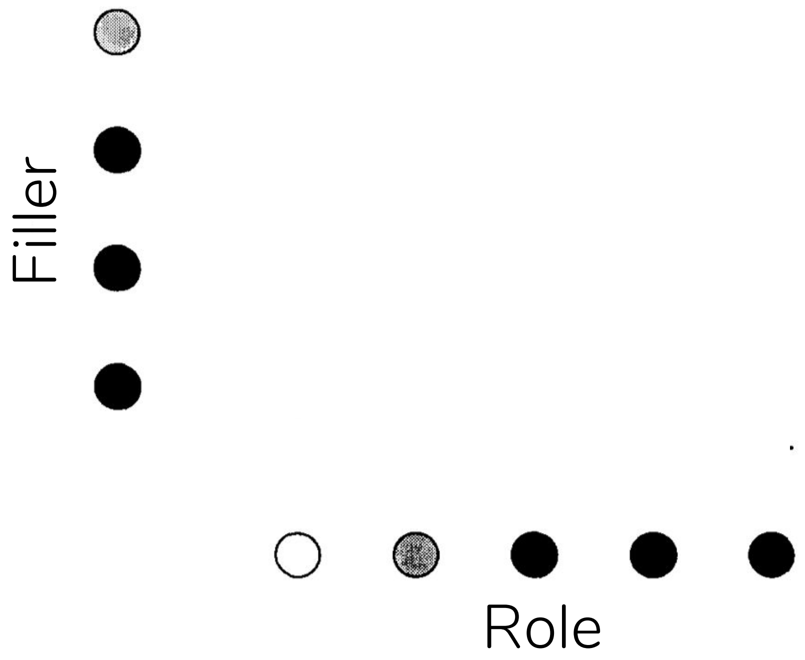
Tensor Product Representations - *what*?

Tensor Product Representations - *what*?



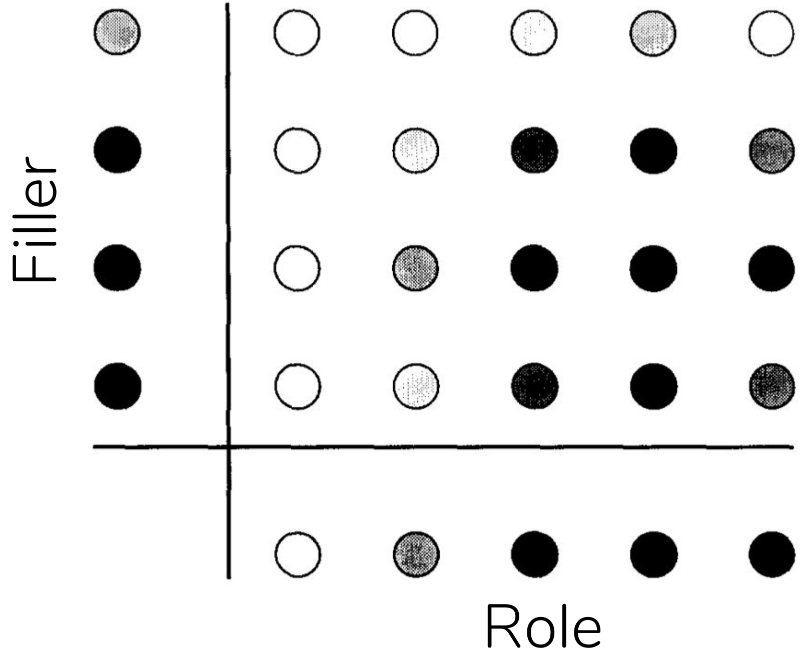
(1) Decomposing the structures via roles

Tensor Product Representations - *what?*



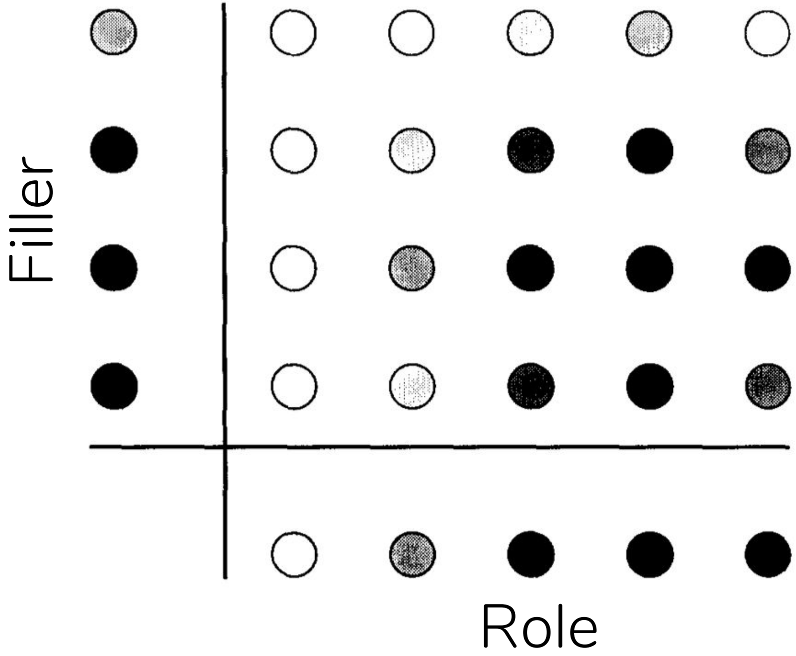
(1) Decomposing the structures via roles

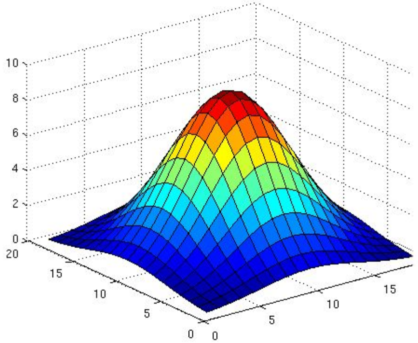
Tensor Product Representations - *what*?



(2) representing variable/value bindings

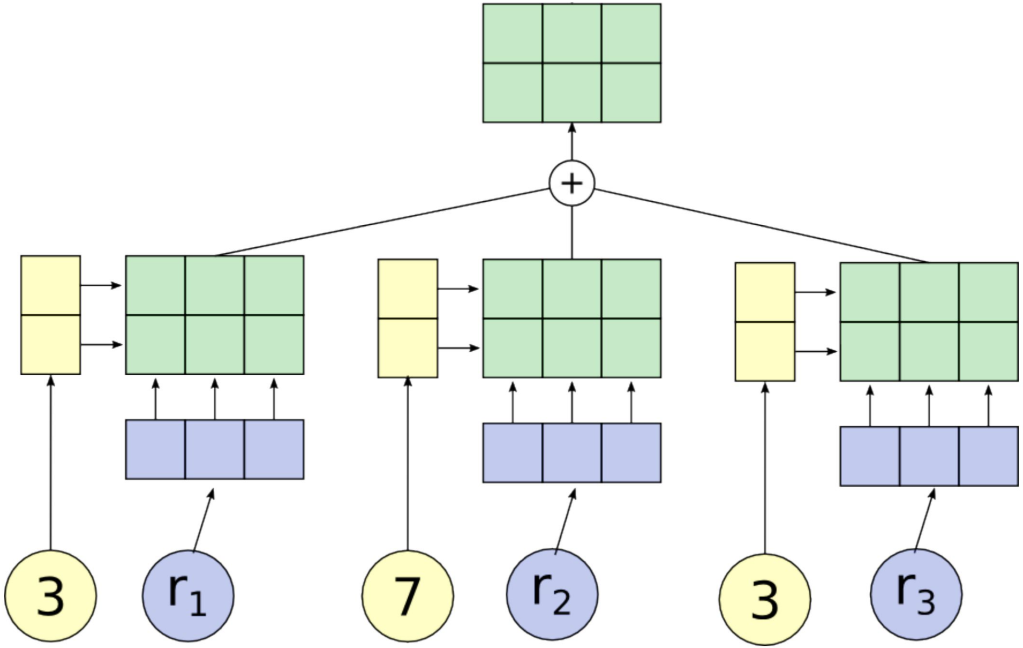
Tensor Product Representations - *what*?





(2) representing variable/value bindings

Tensor Product Representations - *what*?

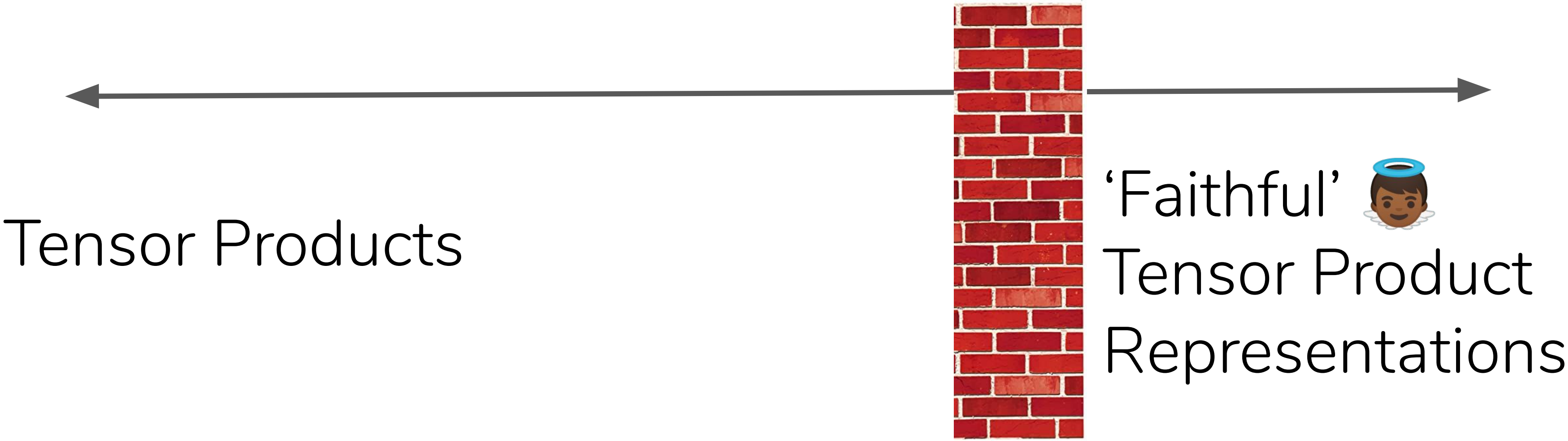


[Soulos et al. 2019]

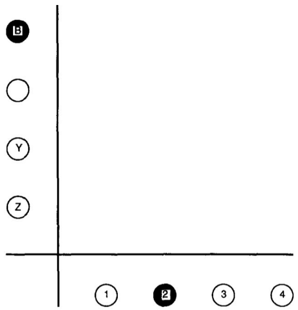
(3) representing conjunctions

Tensor Products

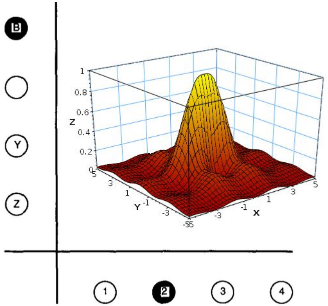
‘Faithful’ 👼
Tensor Product
Representations


Tensor Products
‘Faithful’ 👼🏾
Tensor Product
Representations

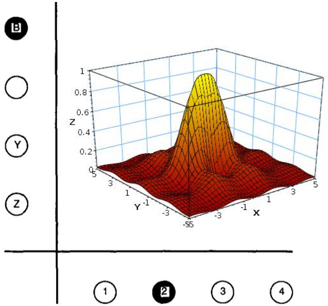
Faithfulness

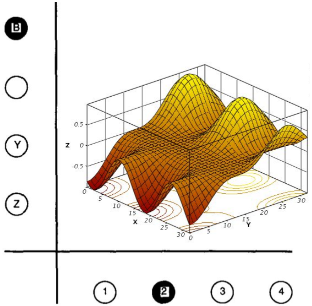


Faithfulness

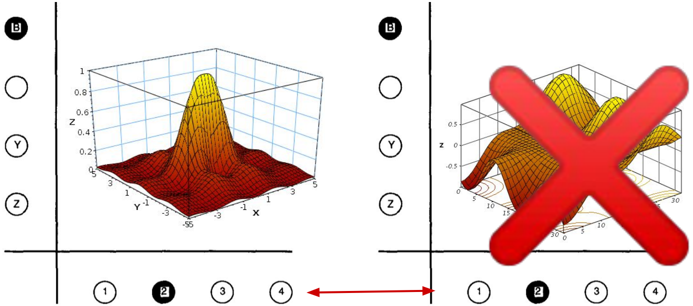


Faithfulness

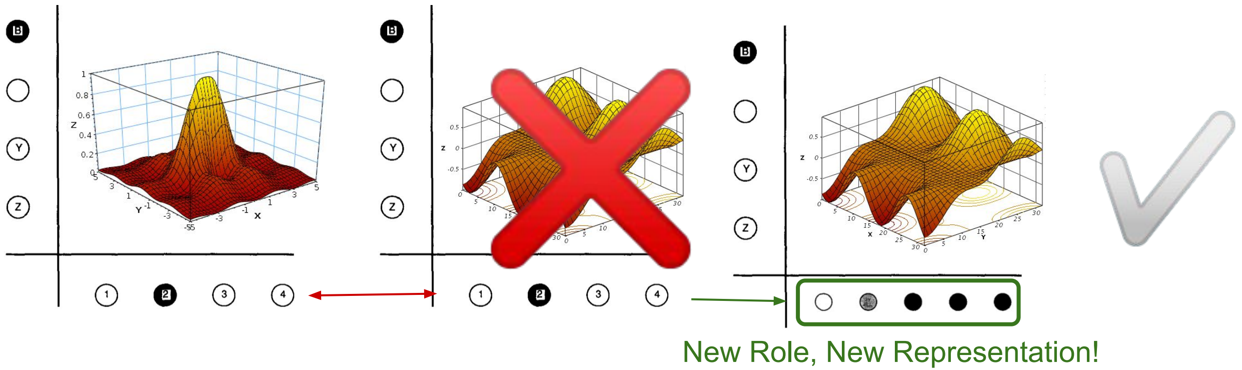




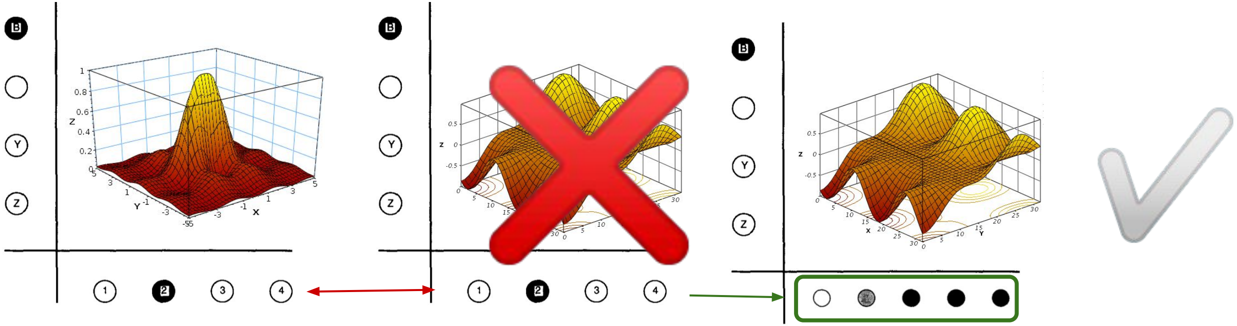
Faithfulness



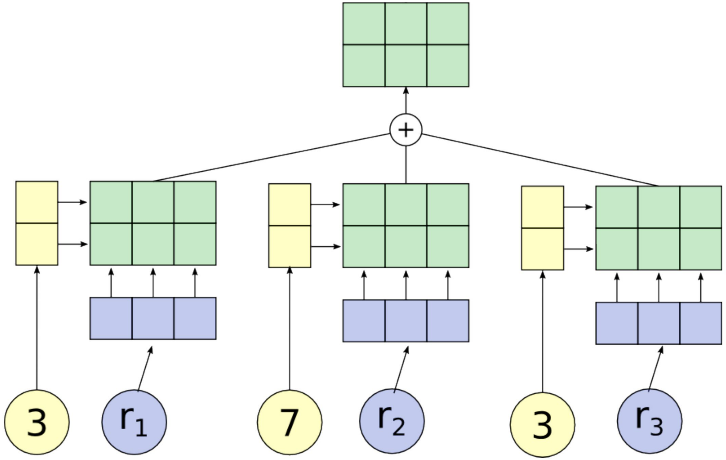
Faithfulness



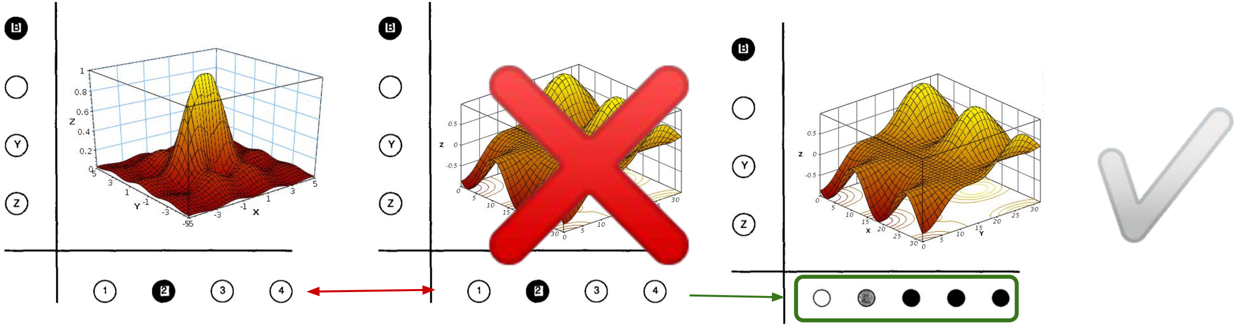
Faithfulness



New Role, New Representation!

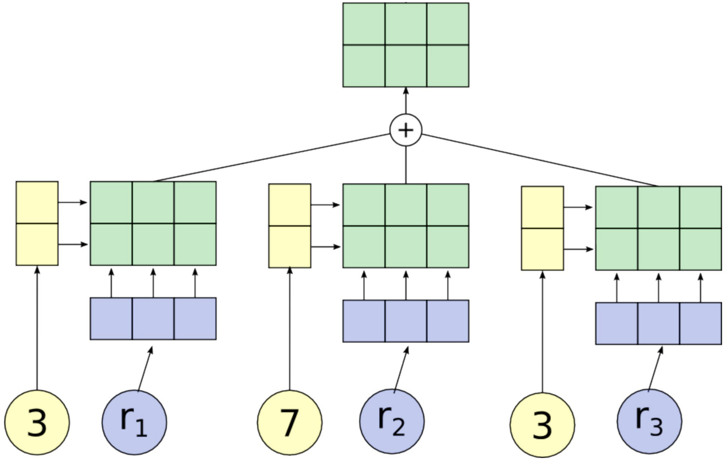


Faithfulness

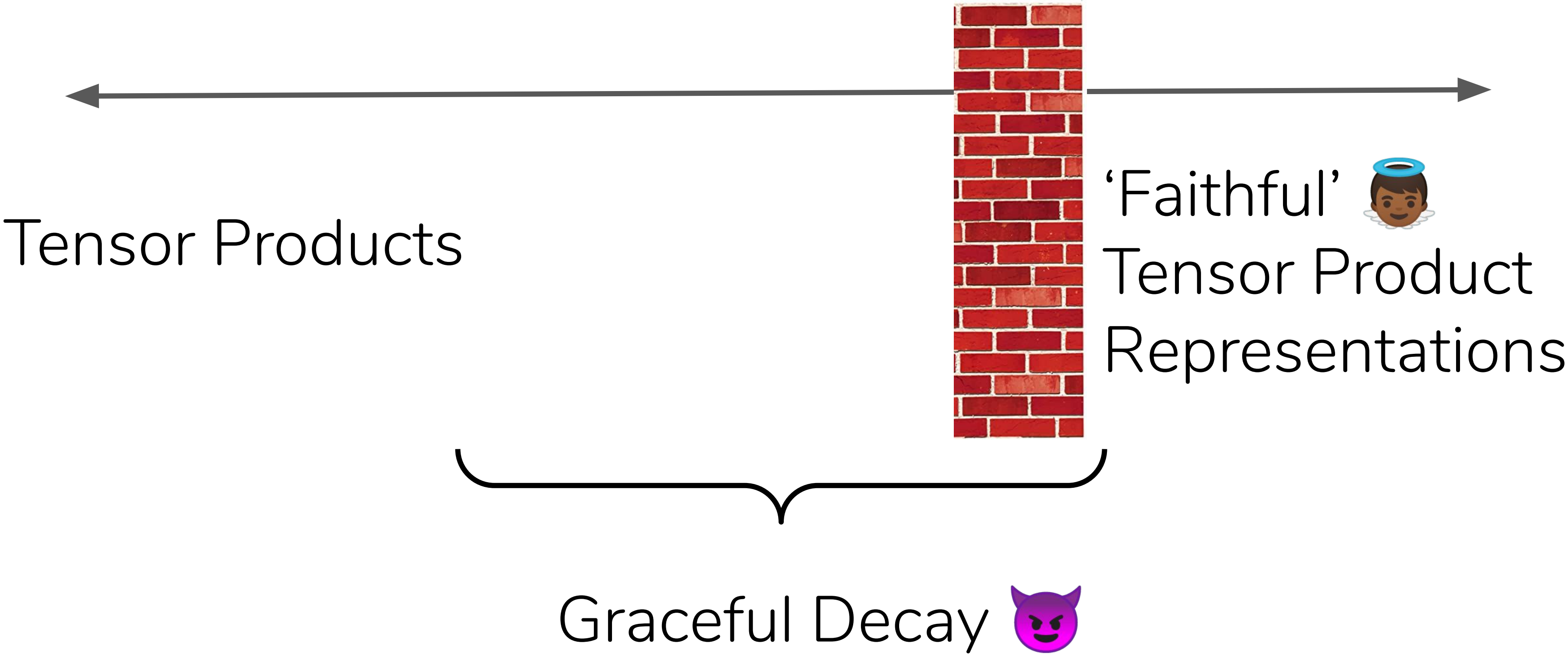


New Role, New Representation!

Orthogonality
Theorem 3.3, Section 3.2



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

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, forall X.

1 5 9

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

<u>(1) Symbolic Structure</u> : *Give*, the programming language

Syntax

p ::= (Give p) | $\square$

(1) Symbolic Structure : *Give*, the programming language

Syntax

p ::= (Give p) | ⬜

Examples

(Give ⬜)

(Give (Give ⬜))

(Give (Give (Give ⬜))

(1) Symbolic Structure : *Give*, the programming language

Syntax

p ::= (Give p) | □

Examples

(Give □)

(Give (Give □))

(Give (Give (Give □))

Semantics

$$\frac{}{(\text{Give } p) \longrightarrow p}$$

$$\frac{p \longrightarrow p'}{(\text{Give } p) \longrightarrow (\text{Give } p')}$$

(1) Symbolic Structure : *Give*, the programming language

Syntax

p ::= (Give p) | □

Examples

(Give □) → □

(Give (Give □)) → (Give □)

(Give (Give (Give □))) → (Give (Give □))

Semantics

$$\frac{}{(\text{Give } p) \longrightarrow p}$$

$$\frac{p \longrightarrow p'}{(\text{Give } p) \longrightarrow (\text{Give } p')}$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4\times3}$, 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.

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4\times3}$, 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.

$$Give_4 = \{ \square, (Give\ \square), (Give\ (Give\ \square)), (Give\ (Give\ (Give\ \square)))\}$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4x3}$, 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.

$$Give_4 = \{ \square, (Give\ \square), (Give\ (Give\ \square)), (Give\ (Give\ (Give\ \square)))\}$$

$$R^{4x3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix}, \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix}, \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix}, \ldots\ldots \right\}$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4x3}$, 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.

$$Give_4 = \{ \square, (Give\ \square), (Give\ (Give\ \square)), (Give\ (Give\ (Give\ \square)))\}$$

$$R^{4x3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix}, \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix}, \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix}, \ldots\ldots \right\}$$

Fillers

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

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4x3}$, 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.

$$Give_4 = \{ \square, (Give\ \square), (Give\ (Give\ \square)), (Give\ (Give\ (Give\ \square)))\}$$

$$R^{4x3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix}, \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix}, \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix}, \ldots\ldots \right\}$$

Fillers

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

Roles

$$r = \{ Give, \square, \varepsilon \}$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4x3}$, 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.

$Give_4$ = { □ , (Give □), (Give (Give □)), (Give (Give (Give □)))}

$$R^{4x3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix},\ \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix},\ \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix},\ \ldots.. \right\}$$

Fillers

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

Roles

r = { Give , □ , ε }

(Give (Give (Give □))) = (i_1:Give) $\wedge$ (i_2:Give) $\wedge$ (i_3:Give) $\wedge$ (i_4: □)

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : \text{Give}_4 \mapsto R^{4\times3}$, 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.

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

$$R^{4\times3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix} , \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix} , \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix} , \ldots\ldots \right\}$$

Fillers

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

Roles

$r = \{ \text{Give} , \square , \varepsilon \}$

$$(\text{Give } (\text{Give } (\text{Give } \square))) = (i_1\text{:Give}) \wedge (i_2\text{:Give}) \wedge (i_3\text{:Give}) \wedge (i_4\text{:} \square)$$

$$(\text{Give } (\text{Give } \square)) = (i_1\text{: } \varepsilon\ ) \wedge (i_2\text{:Give}) \wedge (i_3\text{:Give}) \wedge (i_4\text{:} \square)$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : \text{Give}_4 \mapsto R^{4\text{x}3}$, 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.

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

$$R^{4\text{x}3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix}\ ,\ \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix}\ ,\ \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix}\ ,\ .....\ \right\}$$

Fillers

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

Roles

$$r = \{ \text{Give} ,\ \square\ , \varepsilon \}$$

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

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

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

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : Give_4 \mapsto R^{4x3}$, 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.

$$Give_4 = \{ \square, (Give\ \square), (Give\ (Give\ \square)), (Give\ (Give\ (Give\ \square)))\}$$

$$R^{4x3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix},\ \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix},\ \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix},\ \ldots\ldots \right\}$$

Fillers

$$f = \left\{ \underset{\mathbf{i_1}}{\begin{matrix}[1]\\ [0]\\ [0]\\ [0]\end{matrix}},\ \underset{\mathbf{i_2}}{\begin{matrix}[0]\\ [1]\\ [0]\\ [0]\end{matrix}},\ \underset{\mathbf{i_3}}{\begin{matrix}[0]\\ [0]\\ [1]\\ [0]\end{matrix}},\ \underset{\mathbf{i_4}}{\begin{matrix}[0]\\ [0]\\ [0]\\ [1]\end{matrix}} \right\}$$

Roles

$$r = \left\{ \overset{[1\ 0\ 0]}{\mathbf{Give}},\ \overset{[0\ 1\ 0]}{\square},\ \overset{[0\ 0\ 1]}{\boldsymbol{\varepsilon}} \right\}$$

$$[\![(Give\ (Give\ (Give\ \square)))]\!] = [\![(i_1{:}Give) \wedge (i_2{:}Give) \wedge (i_3{:}Give) \wedge (i_4{:}\square)]\!]$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : \text{Give}_4 \mapsto R^{4\text{x}3}$, 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.

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

$$R^{4\text{x}3} = \left\{ \begin{matrix}[0\ 0\ 0]\\ [1\ 1\ 1]\\ [1\ 1\ 1]\\ [0\ 0\ 2]\end{matrix}\ ,\ \begin{matrix}[1\ 2\ 0]\\ [0\ 1\ 0]\\ [1\ 0\ 1]\\ [2\ 0\ 1]\end{matrix}\ ,\ \begin{matrix}[2\ 2\ 1]\\ [1\ 1\ 3]\\ [1\ 0\ 1]\\ [2\ 0\ 0]\end{matrix}\ ,\ \ldots..\ \right\}$$

Fillers

$$f = \left\{ \underset{\mathbf{i_1}}{\begin{matrix}[1]\\ [0]\\ [0]\\ [0]\end{matrix}},\ \underset{\mathbf{i_2}}{\begin{matrix}[0]\\ [1]\\ [0]\\ [0]\end{matrix}},\ \underset{\mathbf{i_3}}{\begin{matrix}[0]\\ [0]\\ [1]\\ [0]\end{matrix}},\ \underset{\mathbf{i_4}}{\begin{matrix}[0]\\ [0]\\ [0]\\ [1]\end{matrix}} \right\}$$

Roles

$$r = \{ \overset{[1\ 0\ 0]}{\mathbf{Give}},\ \overset{[0\ 1\ 0]}{\square},\ \overset{[0\ 0\ 1]}{\boldsymbol{\varepsilon}} \}$$

$$[\![(\text{Give}\ (\text{Give}\ (\text{Give}\ \square)))]\!] = [\![(i_1{:}\text{Give}) \wedge (i_2{:}\text{Give}) \wedge (i_3{:}\text{Give}) \wedge (i_4{:}\ \square)]\!]$$

$$= (\mathbf{i_1} \otimes \mathbf{Give}) + (\mathbf{i_2} \otimes \mathbf{Give}) + (\mathbf{i_3} \otimes \mathbf{Give}) + (\mathbf{i_4} \otimes \square)$$

(2) Encoding w/ TPRs : *Give*, the programming language

A TPR is a mapping, $[\![p]\!] : \text{Give}_4 \mapsto R^{4\text{x}3}$, 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.

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

$$R^{4\text{x}3} = \left\{ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix}, \ldots\ldots \right\}$$

Fillers

$$f = \left\{ \underset{\mathbf{i_1}}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}}, \underset{\mathbf{i_2}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}}, \underset{\mathbf{i_3}}{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}, \underset{\mathbf{i_4}}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \right\}$$

Roles

$$r = \left\{ \overset{[1\ 0\ 0]}{\textbf{Give}}, \overset{[0\ 1\ 0]}{\square}, \overset{[0\ 0\ 1]}{\boldsymbol{\varepsilon}} \right\}$$

$$[\![(\text{Give } (\text{Give } (\text{Give } \square)))]\!] = [\![(i_1{:}\text{Give}) \wedge (i_2{:}\text{Give}) \wedge (i_3{:}\text{Give}) \wedge (i_4{:}\square)]\!]$$

$$= (\mathbf{i_1} \otimes \textbf{Give}) + (\mathbf{i_2} \otimes \textbf{Give}) + (\mathbf{i_3} \otimes \textbf{Give}) + (\mathbf{i_4} \otimes \square)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

(3) Representation Proofs : *Give*, the programming language

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

$$\mathbf{Give} : \sum_i \mathbf{f}_i \otimes \mathbf{r}_i \mapsto \sum_i \mathbf{f}_i \otimes \mathbf{r}_i$$

(3) Representation Proofs : *Give*, the programming language

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

$$\mathbf{Give} : \textstyle\sum_i \mathbf{f}_i \otimes \mathbf{r}_i \mapsto \sum_i \mathbf{f}_i \otimes \mathbf{r}_i$$

Proof.
Recall that w/ TPRs we encode *Give* programs as conjunctions of filler/role decompositions, i.e. $[\![p]\!] = \sum_i \mathbf{f}_i \otimes \mathbf{r}_i$. Additionally, recall that: (Give p) $\rightarrow$ p

$$\begin{aligned} [\![(\text{Give } p)]\!] &= [\![p]\!] \\ &= \textstyle\sum_i \mathbf{f}_i \otimes \mathbf{r}_i \\ &= \mathbf{Give} \textstyle\sum_i \mathbf{f}_i \otimes \mathbf{r}_i \\ &= \mathbf{Give}\ [\![p]\!] \end{aligned}$$

$\square$

Discussion

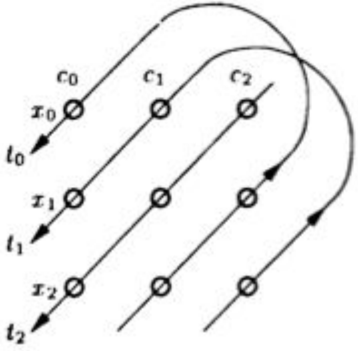
- How does this scale to larger programs in Give?
- What if our programming language was more complicated?
- Other thoughts…

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



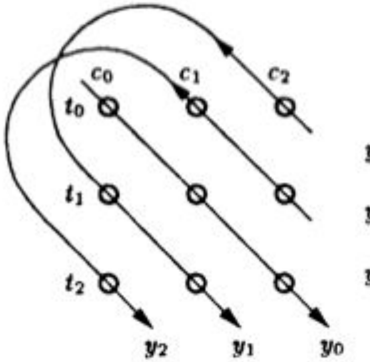
$$\mathbf{t} = \mathbf{c} \circledast \mathbf{x}$$

$$t_0 = c_0 x_0 + c_2 x_1 + c_1 x_2$$

$$t_1 = c_1 x_0 + c_0 x_1 + c_2 x_2$$

$$t_2 = c_2 x_0 + c_1 x_1 + c_0 x_2$$

Circular Convolution



$$\mathbf{y} = \mathbf{c} \bullet \mathbf{t}$$

$$y_0 = c_0 t_0 + c_1 t_1 + c_2 t_2$$

$$y_1 = c_2 t_0 + c_0 t_1 + c_1 t_2$$

$$y_2 = c_1 t_0 + c_2 t_1 + c_0 t_2$$

Circular Correlation (Inverse)

Representations with HRR

Sequences

$$\mathbf{s}_{abc} = \mathbf{a} + \mathbf{a} \circledast \mathbf{b} + \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}$$

$$\mathbf{s}_{de} = \mathbf{d} + \mathbf{d} \circledast \mathbf{e}$$

$$\mathbf{s}_{fgh} = \mathbf{f} + \mathbf{f} \circledast \mathbf{g} + \mathbf{f} \circledast \mathbf{g} \circledast \mathbf{h}$$

S(abcdefgh)= $\mathbf{s}_{abc} + \mathbf{s}_{abc} \circledast \mathbf{s}_{de} + \mathbf{s}_{abc} \circledast \mathbf{s}_{de} \circledast \mathbf{s}_{fgh}$

Representations with HRR

Sequences

Variable Binding

$$\tilde{\mathbf{t}} = \tilde{\mathbf{x}} \circledast \tilde{\mathbf{a}} + \tilde{\mathbf{y}} \circledast \tilde{\mathbf{b}}.$$

Binding a to X and b to Y

Representations with HRR

Sequences

Variable Binding

Frame-Slots

Running frame: Spot runs

$$\mathbf{t}_{running} = \mathbf{l}_{run} + \mathbf{r}_{agent} \circledast \mathbf{f}_{spot}$$

Seeing frame: Dick saw Spot run

$$\begin{aligned}\mathbf{t}_{seeing} &= \mathbf{l}_{see} + \mathbf{r}_{agent} \circledast \mathbf{f}_{dick} + \mathbf{r}_{object} \circledast \mathbf{t}_{running} \\ &= \mathbf{l}_{see} + \mathbf{r}_{agent} \circledast \mathbf{f}_{dick} \\ &\quad + \mathbf{r}_{object} \circledast (\mathbf{l}_{run} + \mathbf{r}_{agent} \circledast \mathbf{f}_{spot})\end{aligned}$$

Example: Filling a frame

Frame:

Job Application:

- Name
- Date

Filler

- September 1, 2020

Example: Filling a frame

Frame:	Job Application	0.35	0.28	0.11
	Name	0.19	-0.14	0.02
	Date	-0.22	0.04	0.10
Filler:	September 1, 2020	0.06	0.05	-0.16

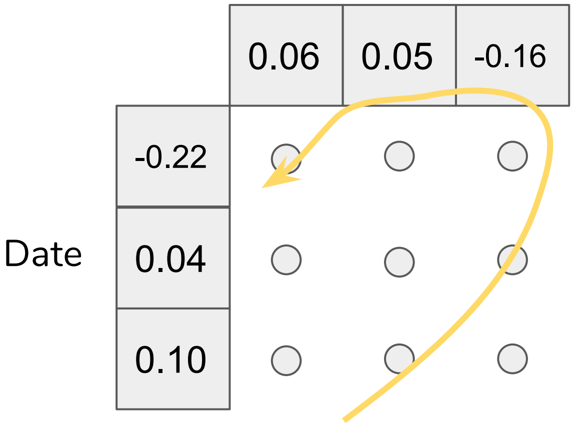
Example: Filling a frame

Binding Date & Filler

September 1, 2020

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

C: -0.0146



Example: Filling a frame

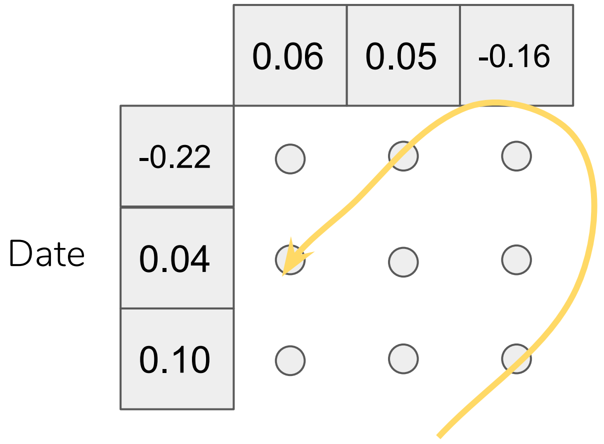
Binding Date & Filler

September 1, 2020

C[1]=0.1*-0.16+0.05*-0.22+0.04*0.06=-0.0246

C:

-0.0146	-0.0246

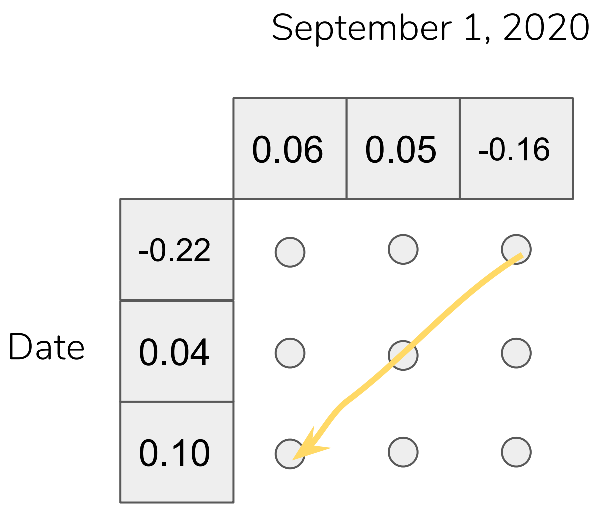


Example: Filling a frame

Pairing Job Application + Date Field

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

C:	-0.0146	-0.0246	0.0432

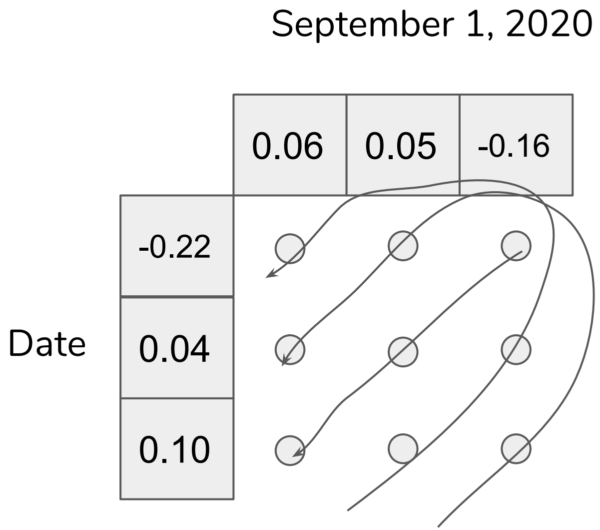


Example: Filling a frame

C:

-0.0146	-0.0246	0.0432

C: {Date:September 1,2020}



Example: Filling a frame

September 1, 2020

16

C:

C: {Jo

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

C’: {Jo

Add in Frame Label

C’: {Date: September 1, 2020, Name}+ Job Application

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

Example: Filling a frame

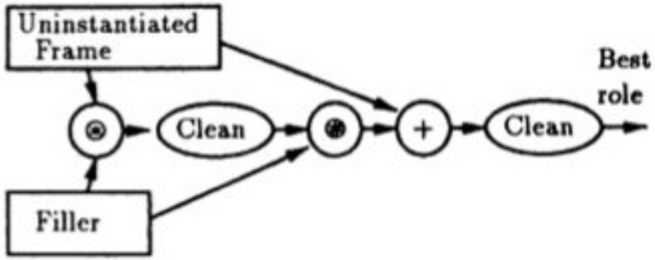
C'': {Job Application: Date:September 1,2020, Name}

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



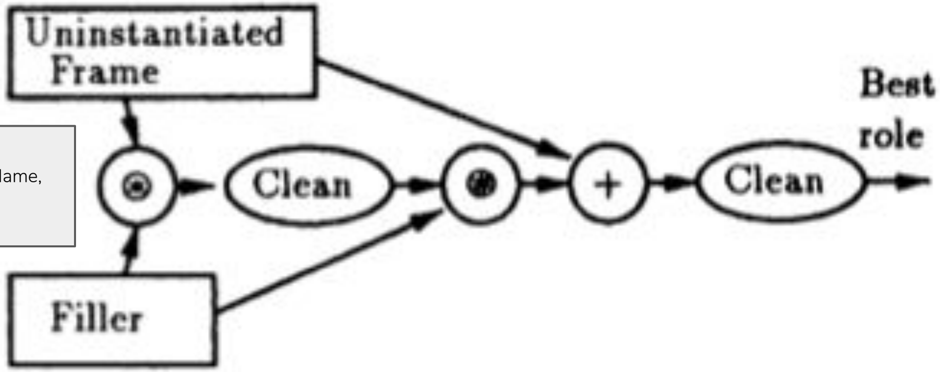
January, 1, 2021

Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

Job Application: Name, Date:January 1,

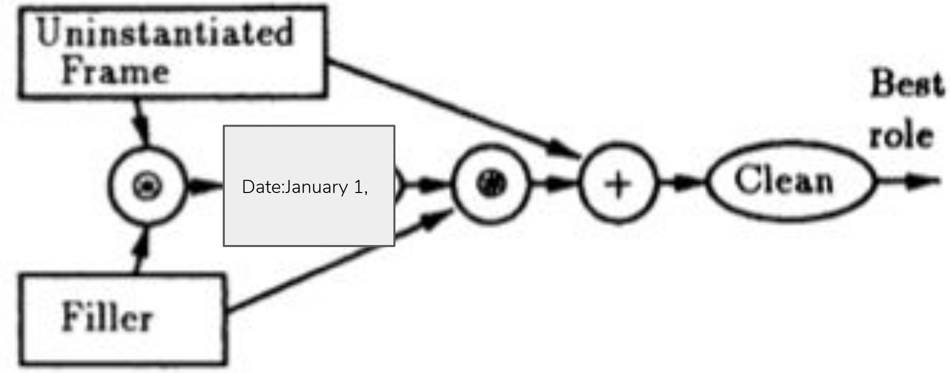


January, 1, 2021

Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

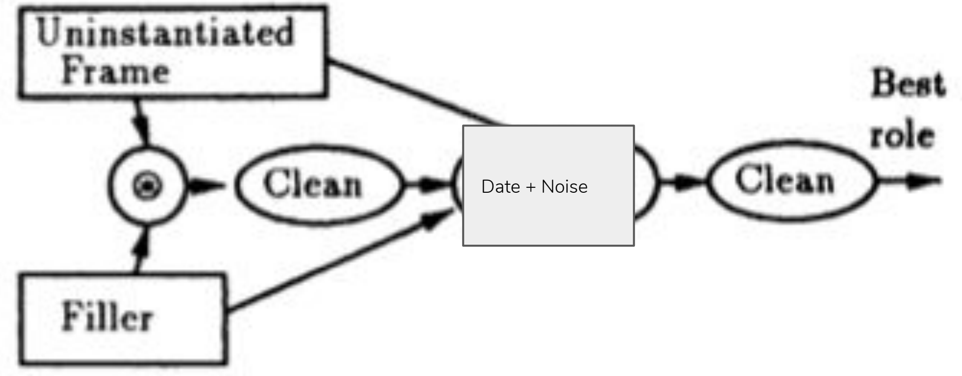


January, 1, 2021

Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date

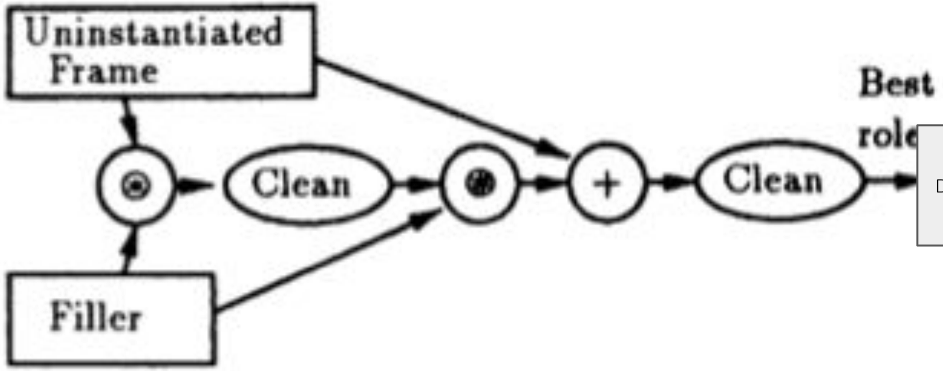


January, 1, 2021

Example Application for Holographic Representation

Best role finder:

Job Application: Name, Date



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)

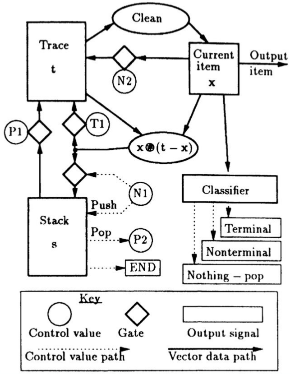
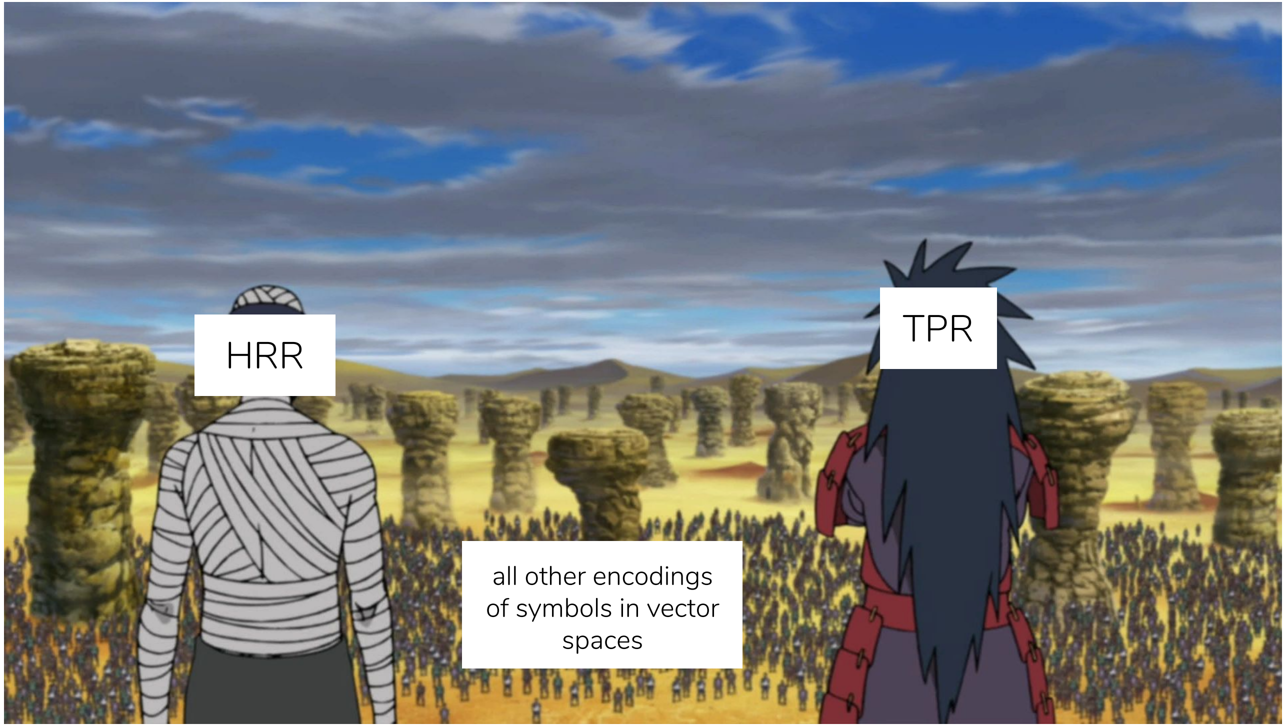


Figure 5: A chunked sequence readout machine.

Encoding Methods?

Encoding Methods?



[END]