Discovering the Compositional Structure of Vector Representations with Role Learning Networks

Paul Soulos
Tom McCoy
Tal Linzen
Paul Smolensky

1 Johns Hopkins University
2 Microsoft
tl;dr

- Our technique can uncover latent compositionality in vector representations
- Interpreting compositional structure sheds light on how these models function
- We understand the inner workings well enough to write down a symbolic algorithm to produce the neural encoding
- Our approximation allows us to directly manipulate the internal representations to produce desired behavior.
What's in a compositional representation?

Consider a sequence of digits [4, 2, 7, 9]

- A set of fillers (tokens)
- Example: \{4, 2, 7, 9\}
What's in a compositional representation?

Consider a sequence of digits [4, 2, 7, 9]

- A set of fillers (tokens)
  - Example: {4, 2, 7, 9}
- A set of roles (positions in the structure)
  - Example: Left-to-right {first, second, third, fourth}
What's in a compositional representation?

Consider a sequence of digits [4, 2, 7, 9]

- A set of fillers (tokens)
  - Example: {4, 2, 7, 9}
- A set of roles (positions in the structure)
  - Example: Left-to-right {first, second, third, fourth}
- A binding operation (placing a filler in a specific role filler:role)
  - Example: {4:first}
What's in a compositional representation?

Consider a sequence of digits [4, 2, 7, 9]

- A set of fillers (tokens)
  - Example: {4, 2, 7, 9}
- A set of roles (positions in the structure)
  - Example: Left-to-right {first, second, third, fourth}
- A binding operation (placing a filler in a specific role filler:role)
  - Example: {4: first}
- A composition operation (stitching all of the bound filler:roles together)
  - Example: {4: first, 2: second, 7: third, 9: fourth}
How can neural networks represent compositional structure?

{4: first, 2: second, 7: third, 9: fourth}
How can neural networks represent compositional structure?

Task: Autoencode

Tensor Product Representations (TPRs)

- A set of fillers (tokens)
- A set of roles (positions in the structure)
- A binding operation (placing a filler in a specific role filler:role)
- A composition operation (stitching all of the bound filler:roles together

Smolensky (1990)
How can neural networks represent compositional structure?

Tensor Product Representations (TPRs)

- A set of fillers (tokens)
  
  Every filler $f_i$ is vector

- A set of roles (positions in the structure)
  
  Every role $r_i$ is vector

- A binding operation (placing a filler in a specific role filler:role)
  
  Tensor product: $f_i \otimes r_i$

- A composition operation (stitching all of the bound filler:roles together)
  
  Sum: $\sum f_i \otimes r_i$

Smolensky (1990)
Tensor Product Encoder

McCoy, Linzen, Dunbar, and Smolensky (2019)
Dissecting Compositionality in Vector Representations (DISCOVER)

**Goal:** Discover implicit compositional structure in learned encodings $E$

**Target Network**

$E = \sum f_i \otimes r_i$
Dissecting Compositionality in Vector Representations (DISCOVER)

**Goal:** Discover implicit compositional structure in learned encodings $E$

**Approach:** Discover implicit compositional structure in the target network's learned encoding $E$ by approximating $E$ with $\hat{E}$

$$E = \sum f_i \otimes r_i$$

$$\hat{E} = \sum f_i \otimes r_i$$

Minimize $\text{MSE}(\hat{E}, E)$
Dissecting Compositionality in Vector Representations (DISCOVER)

**Goal:** Discover implicit compositional structure in learned encodings $E$

**Approach:** Discover implicit compositional structure in the target network's learned encoding $E$ by approximating $E$ with $\hat{E}$

**Evaluation:** Pass the compositional encoding to the non-compositional decoder. There is no fine-tuning.

$E \approx \sum f_i \otimes r_i$

$\hat{E} = \sum f_i \otimes r_i$

Minimize $\text{MSE}(\hat{E}, E)$
What structure is the target network learning?

Task: Autoencode

4,2,7,9 → [complex representation] → 4,2,7,9

{4:first, 2:second, 7:third, 9:fourth}

Left-to-right (LTR) seems intuitive for copying. We want a FIFO queue to maintain the order.
What structure is the target network learning?

Task: Autoencode

Left-to-right (LTR) seems intuitive for copying. We want a FIFO queue to maintain the order.

Task: Reversal

Right-to-left (RTL) seems intuitive for reversal. We want a LIFO stack to reverse the order.
Engineered Roles

McCoy, Linzen, Dunbar, and Smolensky (2019)
Engineered Roles

McCoy, Linzen, Dunbar, and Smolensky (2019)

Substitution accuracy

Copying

Reversal

Left-to-right
Right-to-left
Bidirectional

McCoy, Linzen, Dunbar, and Smolensky (2019)
Differentiable Role Assignment

Encoder

Decoder

4,2,7,9

{4:?, 2:?, 7:?, 9:?}

4,2,7,9
Differentiable Role Assignment

Encoder

Decoder

\{4:, 2:, 7:, 9:\}

\[\sum f_i \otimes r_i\]

\[\begin{array}{ccc}
  r_{11} & r_{1..} & r_{1n1} \\
  r_{1..} & r_{...} & r_{1n} \\
  r_{1d} & r_{..d} & r_{nd}
\end{array}\]

Role Matrix \( R \)
Differentiable Role Assignment

Encoder

Decoder

\{4?:, 2?:, 7?:, 9?:\}

\sum f_i \otimes r_i

Soft attention over the learned Role Matrix for role assignment

n roles of dimension d

Role Matrix R
Target network is a GRU seq2seq architecture

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td>JUMP</td>
</tr>
<tr>
<td>jump left</td>
<td>LTURN JUMP</td>
</tr>
<tr>
<td>jump thrice</td>
<td>JUMP JUMP JUMP</td>
</tr>
<tr>
<td>jump opposite left after walk around right</td>
<td>RTURN WALK RTURN WALK RTURN WALK RTURN WALK LTURN LTURN JUMP</td>
</tr>
</tbody>
</table>
## SCAN

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td>JUMP</td>
</tr>
<tr>
<td>jump left</td>
<td>LTURN JUMP</td>
</tr>
<tr>
<td>jump thrice</td>
<td>JUMP JUMP JUMP</td>
</tr>
<tr>
<td>jump opposite left after walk around right</td>
<td>RTURN WALK RTURN WALK RTURN WALK LTURN LTURN JUMP</td>
</tr>
</tbody>
</table>

Target network is a GRU seq2seq architecture

<table>
<thead>
<tr>
<th>Target</th>
<th><strong>Learned</strong></th>
<th>LTR</th>
<th>RTL</th>
<th>Bi</th>
<th>Tree</th>
<th>BOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.5%</td>
<td>94.8%</td>
<td>6.7%</td>
<td>7.0%</td>
<td>10.7%</td>
<td>4.3%</td>
<td>4.5%</td>
</tr>
</tbody>
</table>

Table 1: Substitution accuracy for various encoders

Lake and Baroni (2018)
Using manual analysis of the role predictions, we created a symbolic algorithm for assigning roles to fillers.
The algorithm matches 98.7% of the role learning network's predictions on the test set.
Using manual analysis of the role predictions, we created a symbolic algorithm for assigning roles to fillers.

The algorithm matches 98.7% of the role learning network's predictions on the test set.

Most roles are defined based on position in a subclause (e.g. last element of the first subclause)

Example roles:
- Role 30: Always assigned to and
- Role 17: Only appears in sequences that contain the word after

These two roles allow the decoder to understand the basic syntax of the command.
Differentiable API Design

- Consider SCAN as a coding assignment between a pair of students.
  - Let's call them “Encoder” and “Decoder”
Differentiable API Design

- Consider SCAN as a coding assignment between a pair of students.
- Let's call them “Encoder” and “Decoder”
- They split the assignment so that Encoder parses the input into a data structure, and Decoder produces the output from this data structure

```java
? encode(List<Input Tokens>)

List<Output Tokens> decode(?)
```
Differentiable API Design

- Consider SCAN as a coding assignment between a pair of students.
- Let's call them “Encoder” and “Decoder”
- They split the assignment so that Encoder parses the input into a data structure, and Decoder produces the output from this data structure

```java
encode(List<Input Tokens>)
List<Output Tokens> decode(?)
```

<isAnd, isAfter, subclauseOneAction, subclauseOneSecondWord...> encode(List<Input Tokens>)
List<Output Tokens> decode(<isAnd, isAfter, subclauseOneAction, subclauseOneSecondWord...>)
Constituent Surgery

$$\text{emb(jump twice)} - \text{TPR(jump)} + \text{TPR(run)} = \text{emb(run twice)}$$

JUMP JUMP $\rightarrow$ RUN RUN
Constituent Surgery

run:11 left:36 twice:8 after:43 jump:10 opposite:17 right:4 thrice:46 →
TR TR JUMP TR TR JUMP TR TR JUMP TL RUN TL RUN
– run:11 + look:11 →
TR TR JUMP TR TR JUMP TR TR JUMP TL LOOK TL LOOK
Constituent Surgery

run: 11 left: 36 twice: 8 after: 43 jump: 10 opposite: 17 right: 4 thrice: 46 →
TR TR JUMP TR TR JUMP TR TR JUMP TL RUN TL RUN
− run: 11 + look: 11 →
TR TR JUMP TR TR JUMP TR TR JUMP TL LOOK TL LOOK
− jump: 10 + walk: 10 →
TR TR WALK TR TR WALK TR TR WALK TL LOOK TL LOOK
Constituent Surgery

run:11 left:36 twice:8 after:43 jump:10 opposite:17 right:4 thrice:46 →
TR TR JUMP TR TR JUMP TR TR JUMP TL RUN TL RUN
– run:11 + look:11 →
TR TR JUMP TR TR JUMP TR TR JUMP TL LOOK TL LOOK
– jump:10 + walk:10 →
TR TR WALK TR TR WALK TR TR WALK TL LOOK TL LOOK
– left:36 + right:36 →
TR TR WALK TR TR WALK TR TR WALK TR LOOK TR LOOK
Constituent Surgery

run:11 left:36 twice:8 after:43 jump:10 opposite:17 right:4 thrice:46 →
TR TR JUMP TR TR JUMP TR TR JUMP TL RUN TL RUN
– run:11 + look:11 →
TR TR JUMP TR TR JUMP TR TR JUMP TL LOOK TL LOOK
– jump:10 + walk:10 →
TR TR WALK TR TR WALK TR TR WALK TL LOOK TL LOOK
– left:36 + right:36 →
TR TR WALK TR TR WALK TR TR WALK TR TR LOOK TR LOOK
– twice:8 + thrice:8 →
TR TR WALK TR TR WALK TR TR WALK TR TR LOOK TR LOOK TR LOOK
Constituent Surgery

run:11 left:36 twice:8 after:43 jump:10 opposite:17 right:4 thrice:46 →
TR TR JUMP TR TR JUMP TR TR JUMP TL RUN TL RUN
  − run:11 + look:11 →
TR TR JUMP TR TR JUMP TR TR JUMP TL LOOK TL LOOK
  − jump:10 + walk:10 →
TR TR WALK TR TR WALK TR TR WALK TL LOOK TL LOOK
  − left:36 + right:36 →
TR TR WALK TR TR WALK TR TR WALK TR TR WALK TR LOOK TR LOOK
  − twice:8 + thrice:8 →
TR TR WALK TR TR WALK TR TR WALK TR TR WALK TR LOOK TR LOOK TR LOOK
  − opposite:17 + around:17 →
TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR WALK TR LOOK TR LOOK TR LOOK
## Sentence Embedding Models

<table>
<thead>
<tr>
<th></th>
<th>Learned</th>
<th>LTR</th>
<th>RTL</th>
<th>Bi</th>
<th>Tree</th>
<th>BOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>InferSent</td>
<td>4.05e-4</td>
<td>8.21e-4</td>
<td>9.70e-4</td>
<td>9.16e-4</td>
<td>7.78e-4</td>
<td>4.34e-4</td>
</tr>
<tr>
<td>Skip-thought</td>
<td>9.30e-5</td>
<td>9.91e-5</td>
<td>1.78e-3</td>
<td>3.95e-4</td>
<td>9.64e-5</td>
<td>8.87e-5</td>
</tr>
<tr>
<td>SST</td>
<td>5.58e-3</td>
<td>8.35e-3</td>
<td>9.29e-3</td>
<td>8.55e-3</td>
<td>5.99e-3</td>
<td>9.38e-3</td>
</tr>
<tr>
<td>SPINN</td>
<td>.139</td>
<td>.184</td>
<td>.189</td>
<td>.181</td>
<td>.178</td>
<td>.176</td>
</tr>
</tbody>
</table>

Mean-squared error for learned and engineered role schemes.
Future Directions

- Train the Tensor Product Encoder end-to-end
- Tensor Product Decoder
- Does a compositional bias improve training?
  - Train faster, fewer parameters, better generalization
- Improving natural language models with a compositional bias
Thank you!

- Run the code yourself
  - https://github.com/psoulos/role-decomposition

- Want more details?
  - Come by the poster
  - Check out the paper:

Acknowledgements
This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1746891, and work partially supported by NSF INSPIRE grant BCS-1344269. All opinions are our own.