# Thinking Like Transformers

Gail Weiss, Yoav Goldberg, Eran Yahav

Presented by Breandan Considine

McGill University

*breandan.considine@mail.mcgill.ca*

July 20, 2021

# How does a transformer think, intuitively?

> Intuitively, transformers' computations are applied to their entire input in parallel, using attention to draw on and combine tokens from several positions at a time as they make their calculations (Vaswani et al., 2017; Bahdanau et al., 2015; Luong et al., 2015). The iterative process of a transformer is then not along the length of the input sequence but rather the depth of the computation: the number of layers it applies to its input as it works towards its final result.
>
> –Weiss, Goldberg & Yahav

# What purpose does the RASP language serve?

We find RASP a natural tool for conveying transformer solutions to [...] tasks for which a human can encode a solution: we do not expect any researcher to implement, e.g., a strong language model or machine-translation system in RASP...we focus on programs that convey concepts people can encode in "traditional" programming languages, and the way they relate to the expressive power of the transformer...

Considering computation problems and their implementation in RASP allows us to "think like a transformer" while abstracting away the technical details of a neural network in favor of symbolic programs.

–Weiss, Goldberg & Yahav

# RASP: Restricted Access Sequence Processing

A curious argument: Not saying that transformers are RASPs, but *for problems we can solve*, they often share the same compiled representation.

How do we do pack useful compute into matrix/vector arithmetic?

- Data types: $\{\mathbb{R}, \mathbb{N}, \mathbb{B}, \Sigma\}^n, \mathbb{B}^{n \times n}$
- Elementwise ops: $\{+, \times, \mathtt{pow}\} : \mathbb{N}^n \to \mathbb{N}^n$ (e.g. $\mathbf{x} + \mathbf{1}$)
- Predicates: $\{\mathbb{R}, \mathbb{N}, \Sigma\}^n \to \mathbb{B}^n$
- Lazily-evaluated functions: $\mathtt{indices}$: $\Sigma^n \to \mathbb{N}^n$

# Selection operator

Takes a key, query and predicate, and returns a selection matrix:

$$\texttt{select} : \Big( \underbrace{(* \rightarrow \mathbb{N}^n)}_{key} \times \underbrace{(* \rightarrow \mathbb{N}^n)}_{query} \times \underbrace{(\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B})}_{predicate} \Big) \rightarrow \mathbb{B}^{n \times n}$$

---

$$\texttt{select}\big( \underbrace{[0,1,2]}_{key}, \underbrace{\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}}_{query}, < \big) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 0 < 1 & 1 < 1 & 2 < 1 \\ 0 < 2 & 1 < 2 & 2 < 2 \\ 0 < 3 & 1 < 3 & 2 < 3 \end{bmatrix} \end{array}$$

$$= \begin{bmatrix} \mathbf{T} & F & F \\ \mathbf{T} & \mathbf{T} & F \\ \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix}$$

# Aggregate

Takes a selection matrix, a list, and averages the selected values:

$$\texttt{aggregate} : \big((* \to \mathbb{B}^{n \times n}) \times \underbrace{(* \to \mathbb{N}^n)}_{list}\big) \to \mathbb{R}^n$$

---

$$\texttt{aggregate(} \begin{bmatrix} \mathbf{T} & F & F \\ \mathbf{T} & \mathbf{T} & F \\ \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix}, \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} \texttt{)} = \begin{bmatrix} \frac{10}{1} & \frac{10+20}{2} & \frac{10+20+30}{3} \end{bmatrix}$$

$$= \begin{bmatrix} 10 & 15 & 20 \end{bmatrix}$$

# Dyck words and languages

**Definition**

A **Dyck-1 word** is a string containing the same number of ('s and )'s, where the number of )'s in every prefix is less than or equal to the number of ('s.

✓ `((())), (())(), ()(()), (()()), ()()()`

✗ `)))(((, ))(()(, )()))((, ))()((, )()()(, ((()(),...`

**Definition**

A **Dyck-n word** is a Dyck word with n bracket types.

✓ `()[]{}, ([]{}), [(){}], {()[]}, ([]){}, [()]{}, ...`

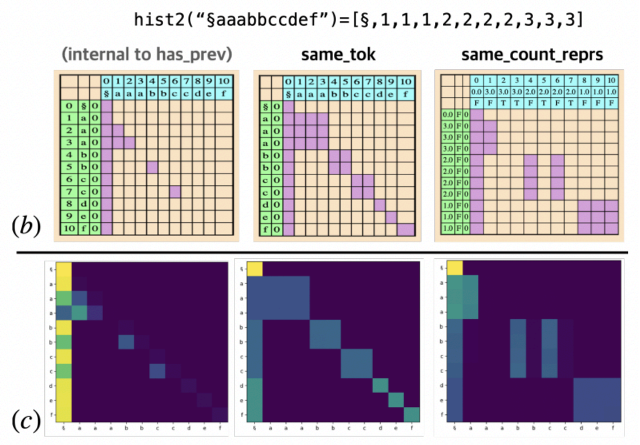
✗ $\underbrace{\texttt{([)]\{\}, [\{]\}(), [\{](\}), ([)\{]\}, \ldots}}_{\text{shuffle Dyck-3}}$ , `}{()]]`

# Experiments

Three sets of experiments:

- With attention supervision: assuming the solution could be learned, would it even work in the first place?
- Without attention supervision: does standard supervision (i.e. cross-entropy loss on the target without teacher forcing) recover the same (or similar) solution?
- How tight are the RASP-implied bounds for the minimum number of layers and maximum attention heads? Compile the RASP to a transformer: there exists a transformer which can solve the task! But can we recover it via learning?

# Experiment 1: Compiling RASP to transformers



# Experiment 2: Can it be learned / does it learn the same thing?

| Language | Layers | Heads | Test Acc. | Attn. Matches? |
|---|---|---|---|---|
| Reverse | 2 | 1 | 99.99% | ✓✗ |
| Hist BOS | 1 | 1 | 100% | ✓ |
| Hist no BOS | 1 | 2 | 99.97% | ✓✗ |
| Double Hist | 2 | 2 | 99.58% | ✓✗ |
| Sort | 2 | 1 | 99.96% | ✗ |
| Most Freq | 3 | 2 | 95.99% | ✗ |
| Dyck-1 PTF | 2 | 1 | 99.67% | ✓✗ |
| Dyck-2 PTF [8] | 3 | 1 | 99.85% | ✗ |

# Experiment 3: How tight are the L, H bounds?

| Language | RASP | Average test accuracy (%) with... | | | |
|---|---|---|---|---|---|
| | $L, H$ | $L, H$ | $H-1$ | $L-1$ | $L-1, 2H$ |
| Reverse | $2, 1$ | **99.9** | - | 23.1 | 41.2 |
| Hist | $1, 2$ | **99.9** | 91.9 | - | - |
| 2-Hist | $2, 2$ | **99.0** | 73.5 | 40.5 | 83.5 |
| Sort | $2, 1$ | 99.8 | - | 99.0 | **99.9** |
| Most Freq | $3, 2$ | **93.9** | 92.1 | 84.0 | 90.2 |
| Dyck-1 | $2, 1$ | **99.3** | - | 96.9 | 96.4 |
| Dyck-2 | $3, 1$ | **99.7** | - | 98.8 | 94.1 |

# Threats to validity

- How do we know whether RASP solutions are learnable? What do the L, H bounds really tell us?
- Do transformers really think this way? Or are we just selecting problems which we can force a transformer to reproduce or which are relatively "learnable"?
- Are there tasks for which the RASP-implied bounds do not predict learnability in practice?
- What other evidence could be shown to demonstrate transformers actually think this way?
- What can we say about uniqueness? Is there a way to canonicalize attention?

# Unanswered questions/Future work

- Is there a way to "scale up" to longer sequences? How would you "upsample" a RASP heatmap?
- Would it be possible to extract RASP source code from a pretrained transformer?
- What other evidence could be shown to demonstrate transformers actually think this way?
- What can we say about uniqueness? Is there a way to canonicalize attention?

# References


- Weiss et al., Thinking Like Transformers (2021)
- Bhattamishra et al., On the Ability and Limitations of Transformers to Recognize Formal Languages (2020)