A Mathematical Framework for Transformer Circuits

ANTHROPIC

2025. 03.18

JongGeun Lee, Jungwoo Yang, Haesung Pyun

Content

1. Introduction

1. Introduction – Transformer Circuits Thread

Transformer Circuits Thread

Articles

FEBRUARY 2025

Insights on Crosscoder Model Diffing – A preliminary note on using crosscoders to diff models.

JANUARY 2025

Circuits Updates — January 2025 – A collection of small updates: dictionary learning optimization techniques.

DECEMBER 2024

Stage-Wise Model Diffing – A preliminary note on model diffing through dictionary fine-tuning.

A Mathematical Framework for Transformer Circuits

AUTHORS

Nelson Elhage*†, Neel Nanda*, Catherine Olsson*, Tom Henighan†, Nicholas Joseph†, Ben Mann†, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, Chris Olah‡

AFFILIATION

Anthropic

PUBLISHED

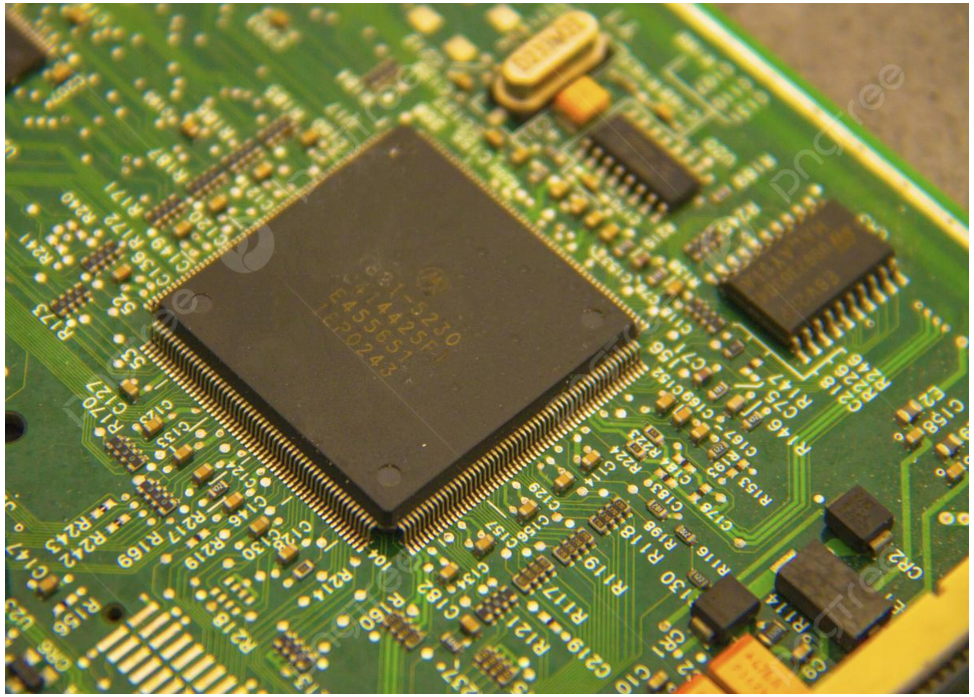
Dec 22, 2021

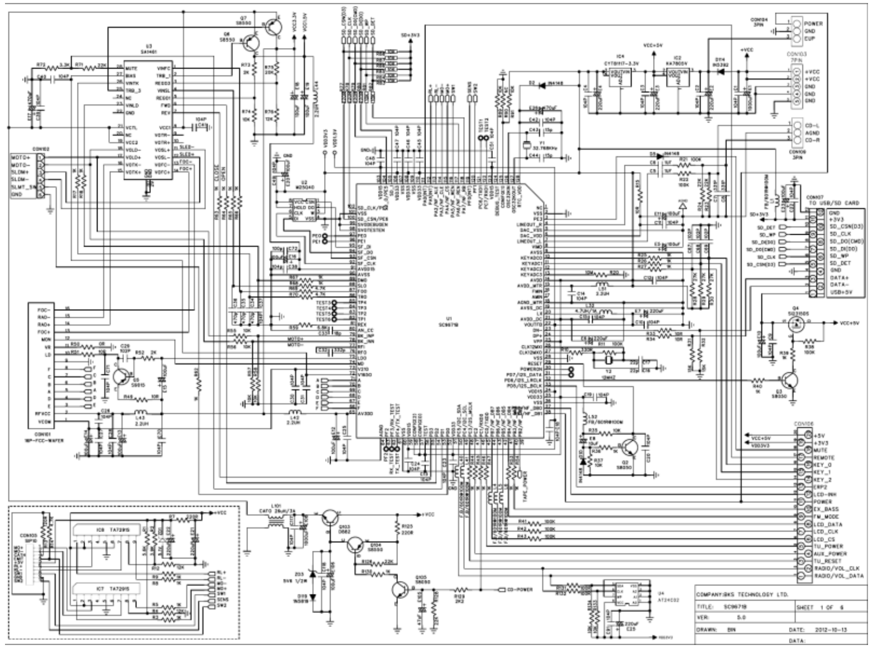
* Core Research Contributor; † Core Infrastructure Contributor; ‡ Correspondence to colah@anthropic.com; Author contributions statement below.

- A foundational article for understanding inner workings of transformer architecture.

- Antrophic, Transformer Circuits Thread
- Antrophic, A Mathematical Framework for Transformer Circuits

1. Introduction – What is Circuit?

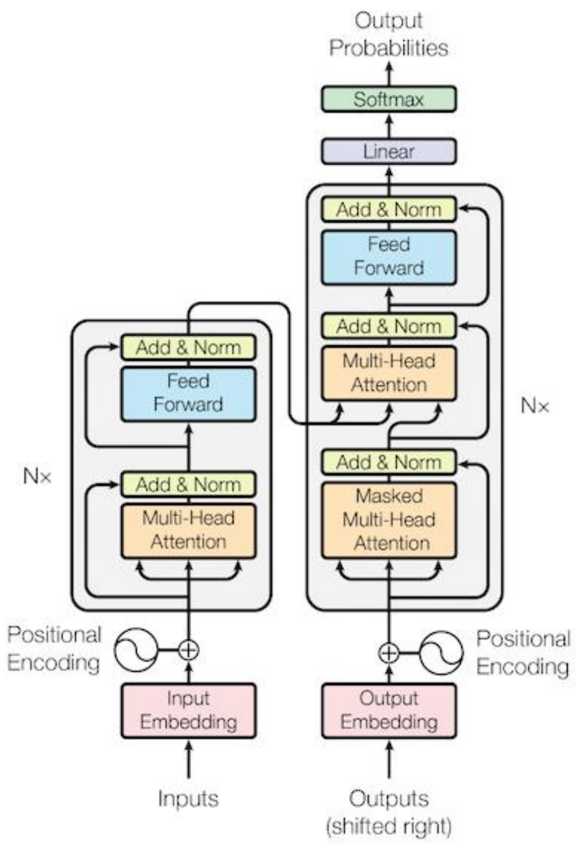




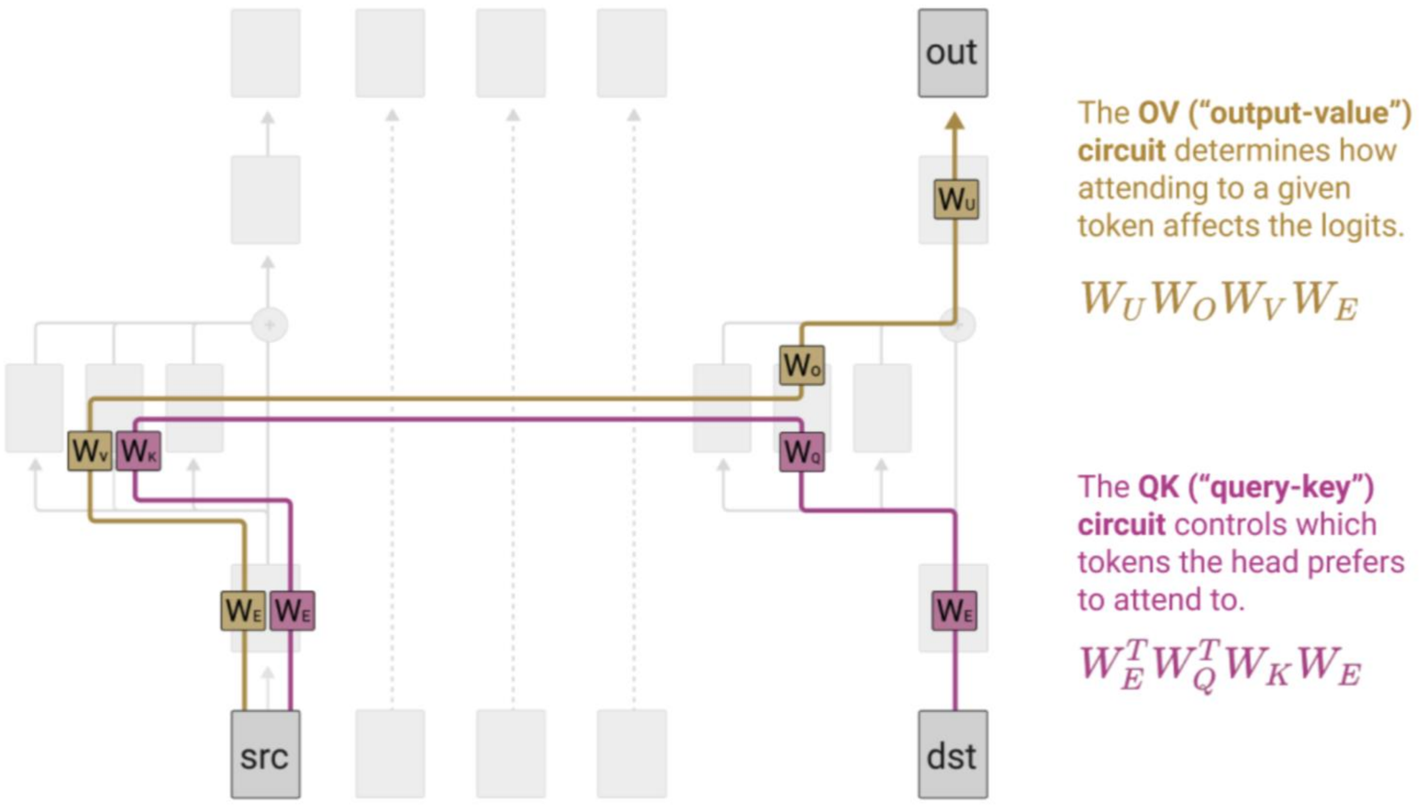
- It seems complex at first glance, but knowing each part's role makes it understandable.

1. Introduction – What is Circuit?

Transformer



Transformer Circuit



- This is an attempt to view the Transformer as a circuit and analyze how each component affects the logits.

1. Introduction – Model Simplifications

Toy Transformers

- At most two layers (0, 1, 2)
  - More than two layers will be talked in Week 4 (In-context Learning and Induction Heads)
- Attention-Only (No MLPs)
  - The authors emphasized the importance of the MLP layer, but excluded it due to difficulties in interpretation.
- Get rid of layer normalization
- No biases
  - The bias term was omitted, as it can be incorporated by extending the weight with an additional dimension.

- Antrophic, In-context Learning and Induction Heads.

1. Introduction – Reverse Engineering Results

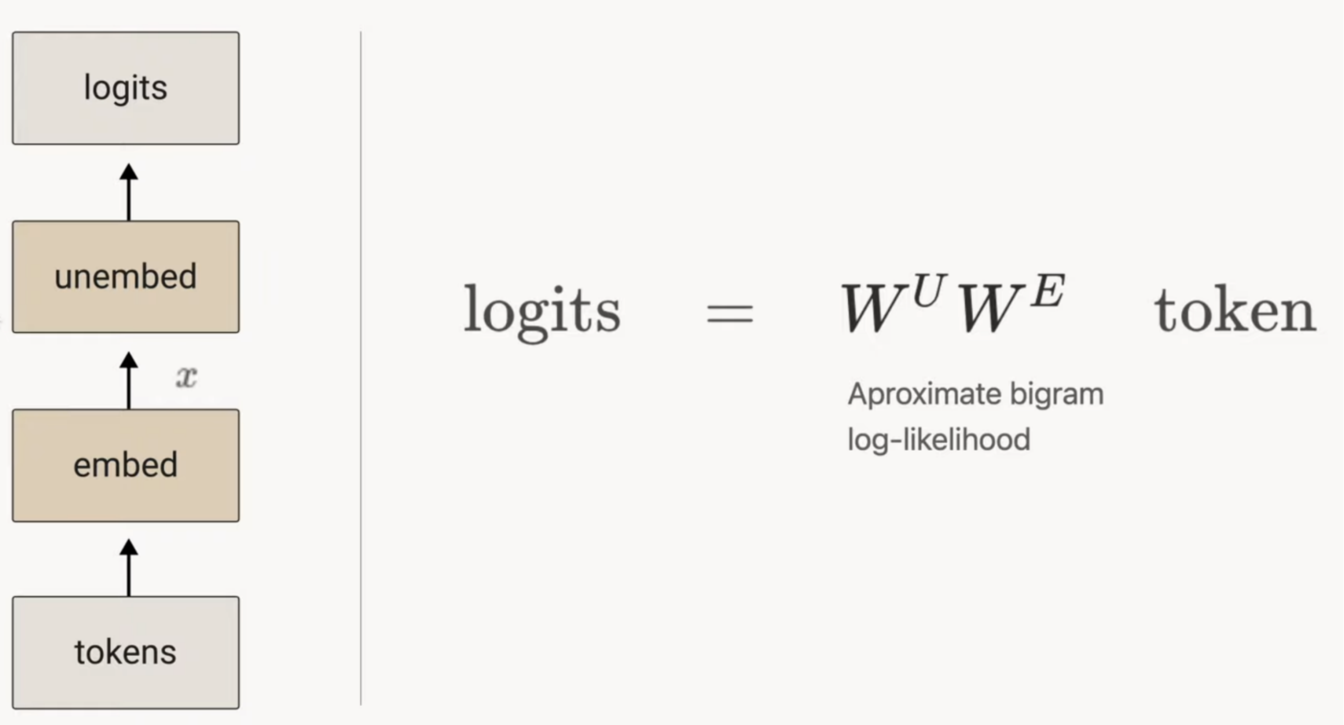
Reverse Engineering Results

- Zero layer transformers model **bigram statistics**.
- One layer attention-only transformers are an ensemble of bigram and "**skip-trigram**" (sequences of the form "A… B C") models.
- Two layer attention-only transformers can implement much more complex algorithms using **compositions of attention heads**.

1. Introduction – Reverse Engineering Results

Zero Layer Transformer

- Zero layer transformers model **bigram statistics**.



- Since there is no interaction between tokens, the task becomes a bi-gram problem—predicting the most appropriate next token based solely on the current one.

1. Introduction – Reverse Engineering Results

One Layer Transformer

- One layer attention-only transformers are an ensemble of bigram and "**skip-trigram**" (sequences of the form "A… B C") models.

```
Skip-trigram: [source]... [destination][out]
```

Some examples of large entries QK/OV circuit

Source Token	Destination Token	Out Token	Example Skip Tri-grams
" perfect"	" are", " looks", " is", " provides"	" perfect", " super", " absolute", " pure"	" perfect... are perfect", " perfect... looks super"
" large"	" contains", " using", " specify", " contain"	" large", " small", " very", " huge"	" large... using large", " large... contains small"
" two"	" One", "\n ", " has", "\r\n ", "One"	" two", " three", " four", " five", " one"	" two... One two", " two... has three"
"lambda"	" $\\", "}{\\", "+\\", "(\\", " ${\\"	"lambda", "sorted", " lambda", "operator"	"lambda... $\\lambda", "lambda... +\\lambda"
"nbsp"	"&", " \"&", "}&", ">&", "=&"	"nbsp", "01", "gt", "00012", "nbs", "quot"	"nbsp... ", "nbsp... > "
"Great"	"The", " The", " the", " contains", " /"	" Great", " great", " poor", " Every"	"Great... The Great", "Great... the great"

- A **skip-trigram** pattern refers to cases where a token (e.g., **C**) is predicted by attending not only to the previous token (**B**) but also to a **distant earlier token** (**A**).

1. Introduction – Reverse Engineering Results

One Layer Transformer

- One layer attention-only transformers are an ensemble of bigram and "**skip-trigram**" (sequences of the form "A… B C") models.

```
Skip-trigram: [source]... [destination][out]
```

- In natural language, long-range dependencies are common — a distant token (A) can influence later tokens (B, C).
- A one-layer transformer can predict the next token (C) by:
  - Referring to the immediately **previous token** (B → C), like a **bigram** model
  - Referring to a **distant but relevant token** (A → C), similar to a **skip-trigram** pattern

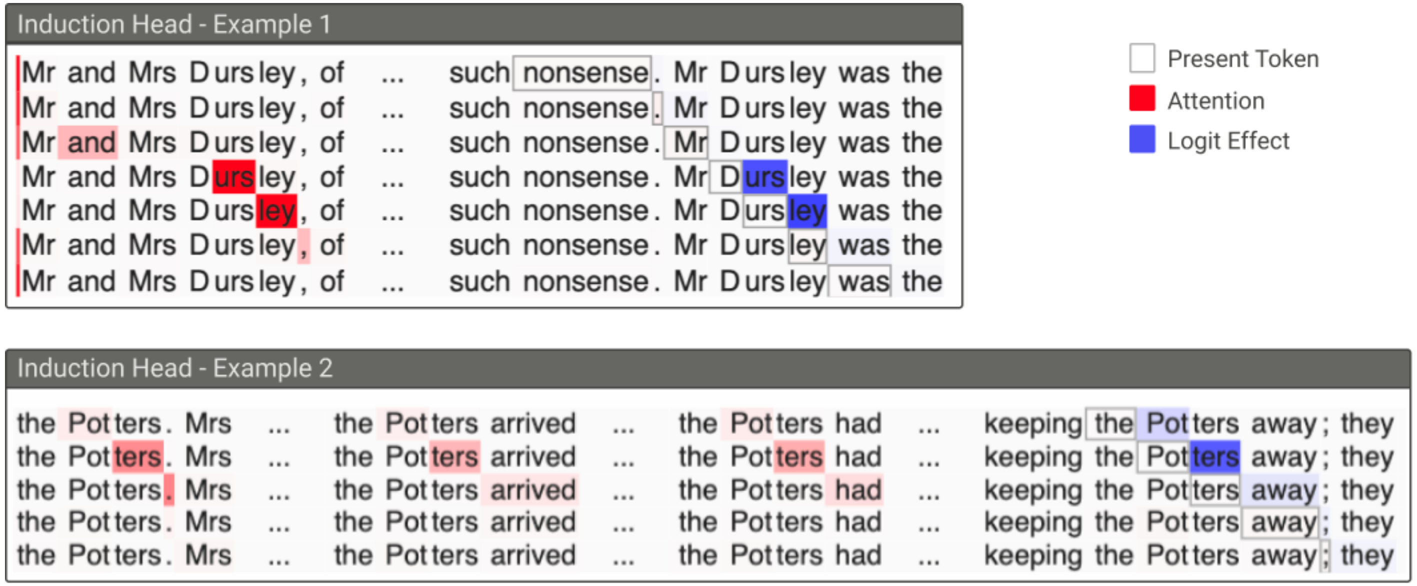
1. Introduction – Reverse Engineering Results

Two Layer Transformer

- Two layer attention-only transformers can implement much more complex algorithms using **compositions of attention heads**.

Induction Head Behavior

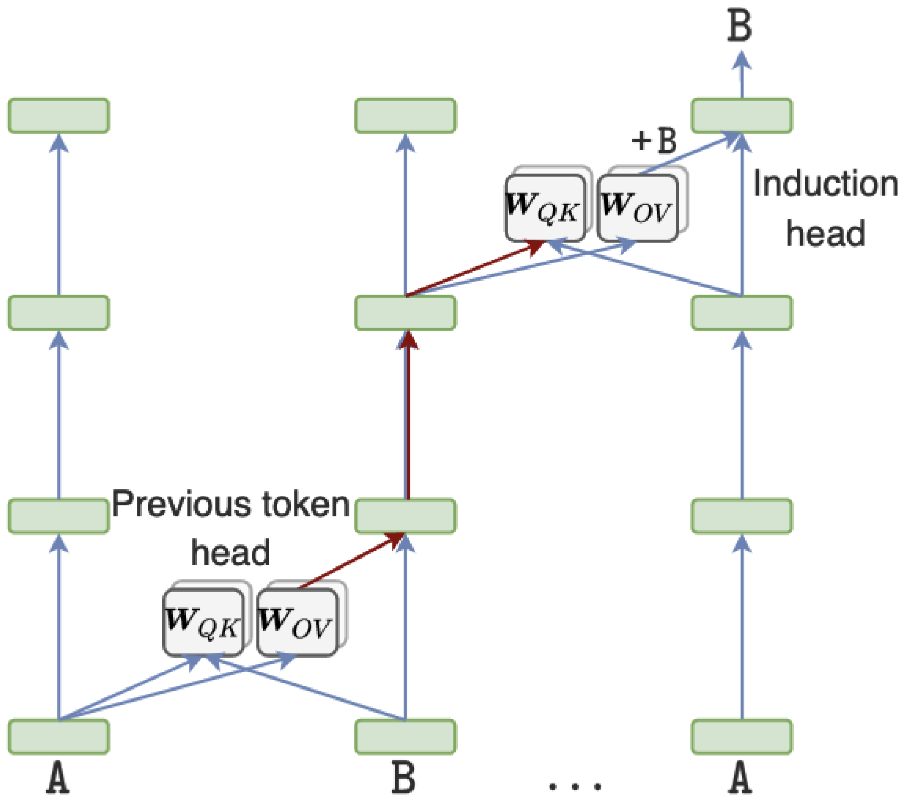
- Induction heads search for a previous occurrence of the current token, and if found, copy the token that followed it.



1. Introduction – Reverse Engineering Results

Two Layer Transformer

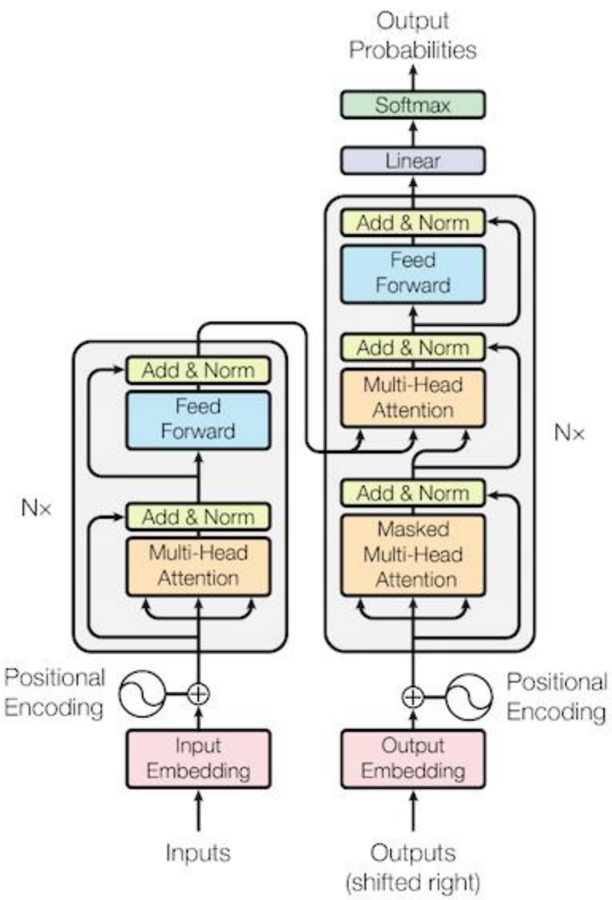
- Two layer attention-only transformers can implement much more complex algorithms using **compositions of attention heads**.



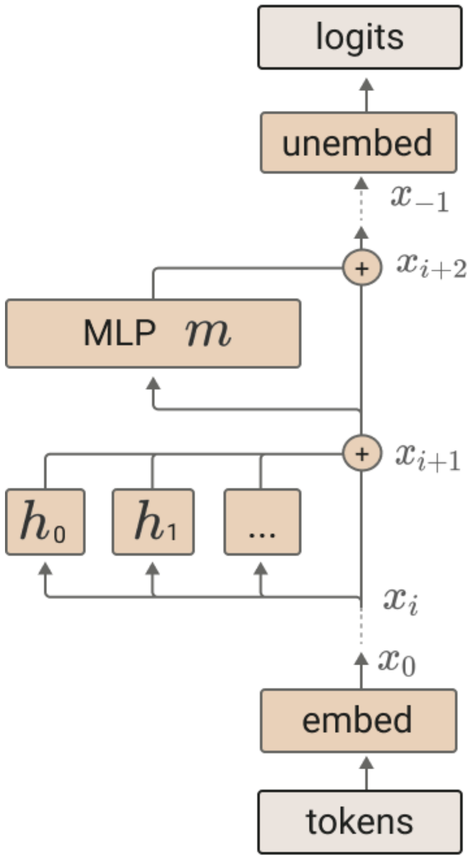
2. Transformer Overview

2. Transformer Overview – Residual Stream

Transformer (Vaswani et al., 2017)



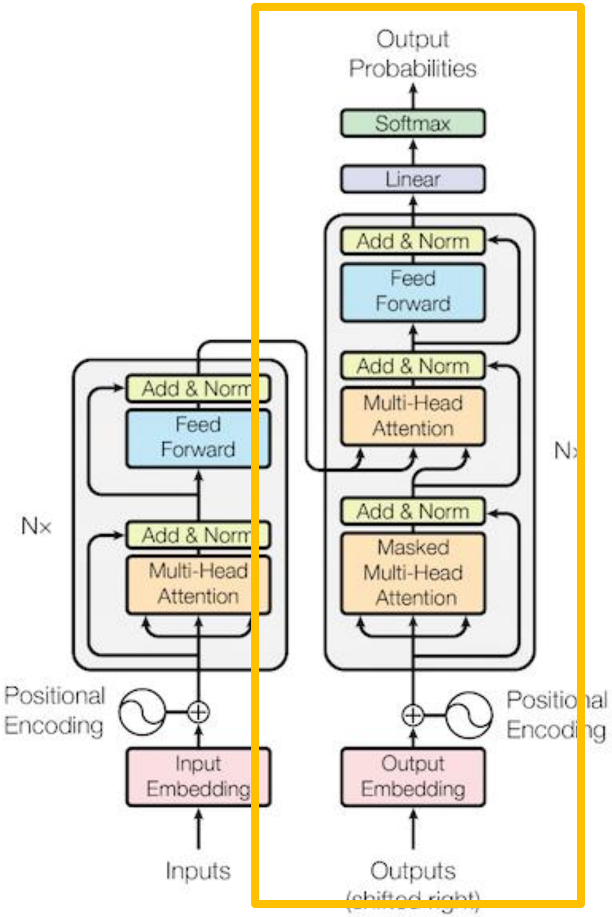
Transformer (Elhage et al., 2021)



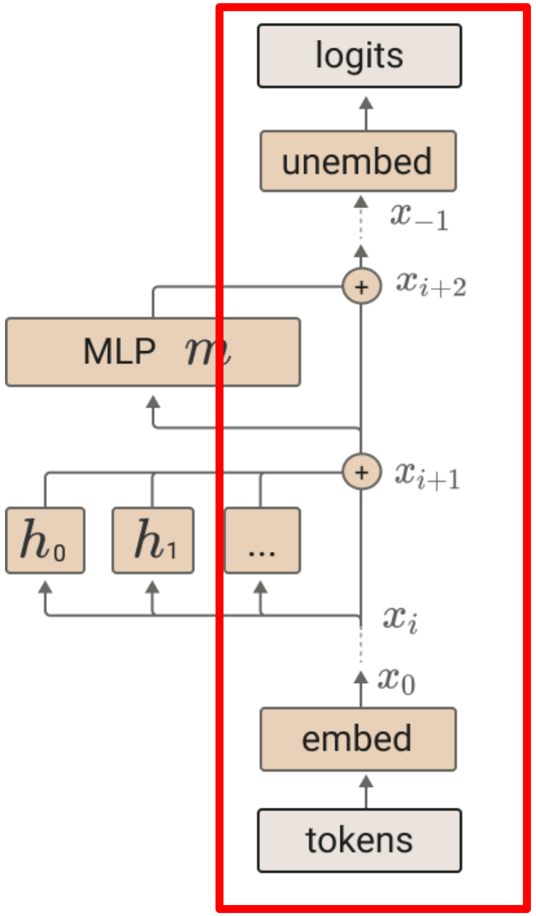
- This figure represents author's perspective on the Transformer architecture.

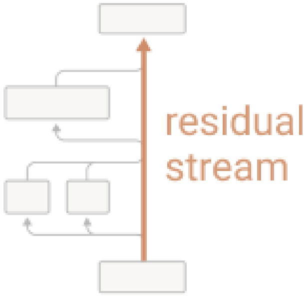
2. Transformer Overview – Residual Stream

Transformer (Vaswani et al., 2017)



Transformer (Elhage et al., 2021)





- Both the attention and MLP layers each "read" their input from the **residual stream.**
- And, then "write" their result to the **residual stream** by adding a linear projection back in.

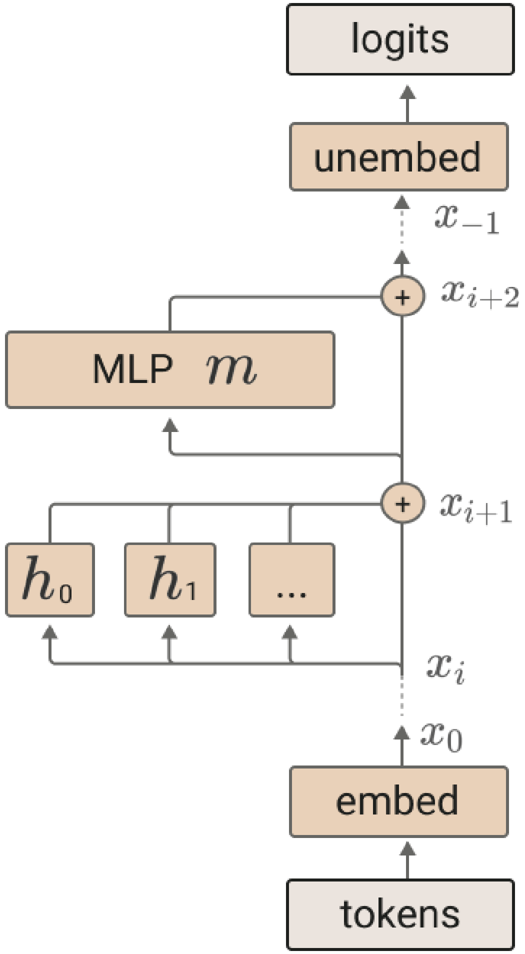
2. Transformer Overview – Residual Block

$W_U : V \times d_{model}$

$x_i : d_{model} \times 1$

$W_E : d_{model} \times V$

$t : V \times 1$



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

Each attention head, h, is run and added to the residual stream.

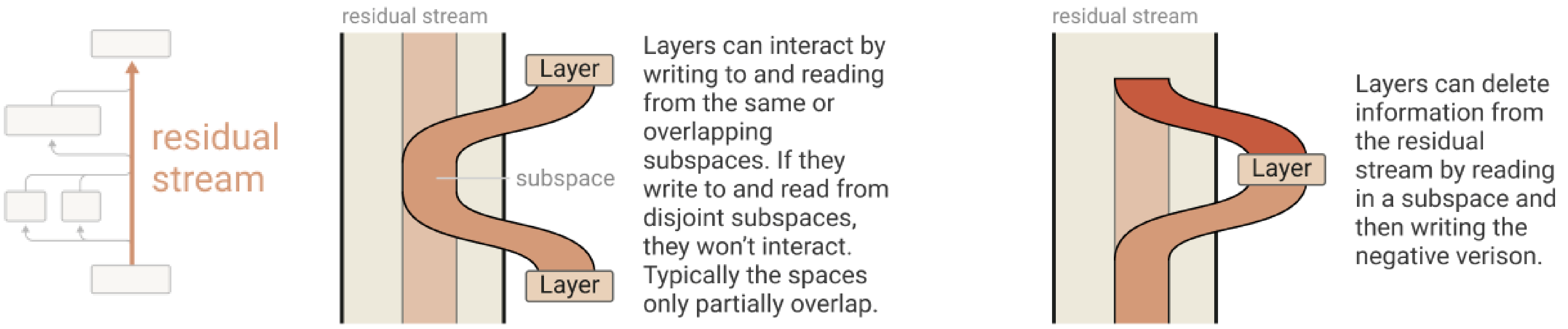
$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

One residual block

Token embedding.

$$x_0 = W_E t$$

2. Transformer Overview – Virtual Weights

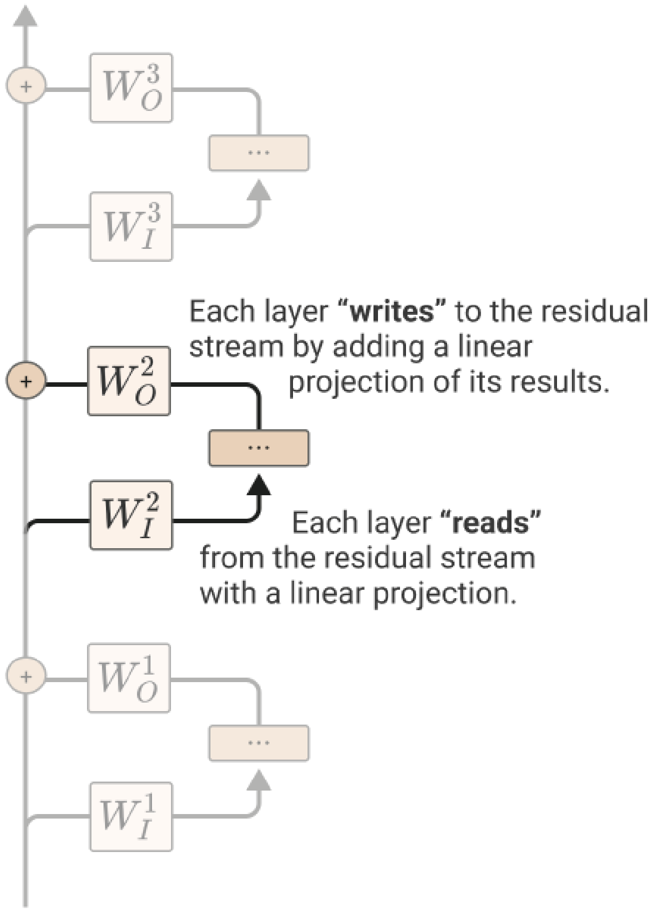


Residual stream:

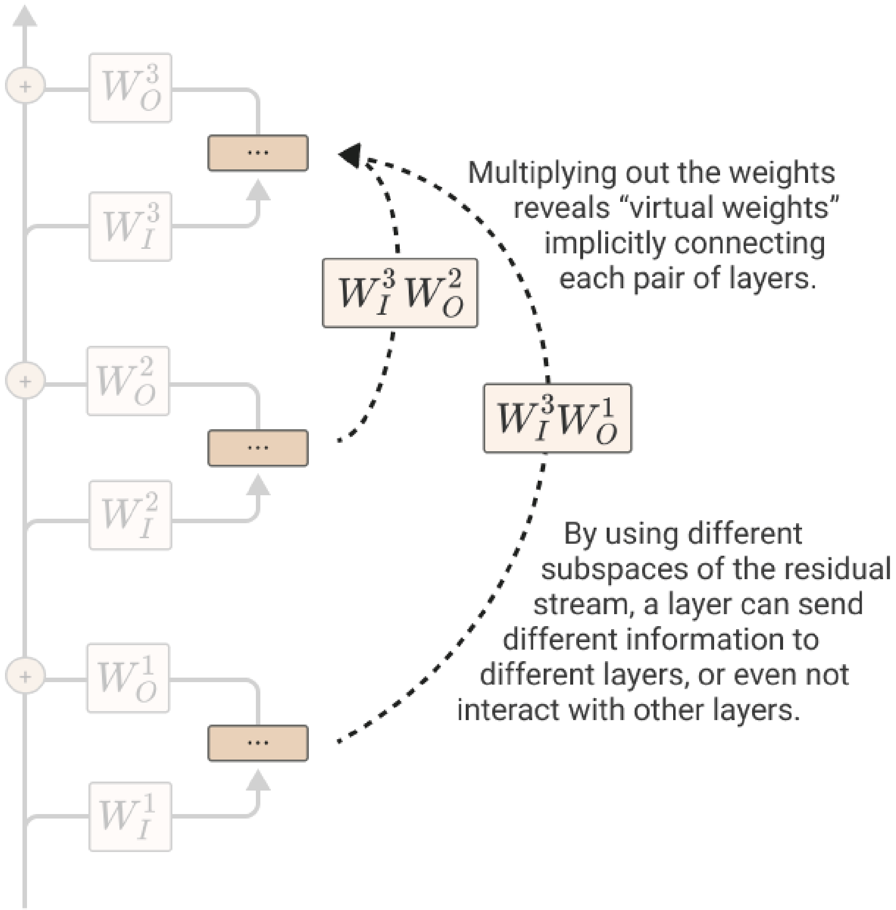
- is high-dimensional, (e.g., Llama 8B: 4096, 70B: 28,672) and can be divided into different **subspaces.**
- allows different components to move information efficiently by operating in **distinct or shared subspaces.**

2. Transformer Overview – Virtual Weights

The residual stream is modified by a sequence of MLP and attention layers “reading from” and “writing to” it with linear operations.

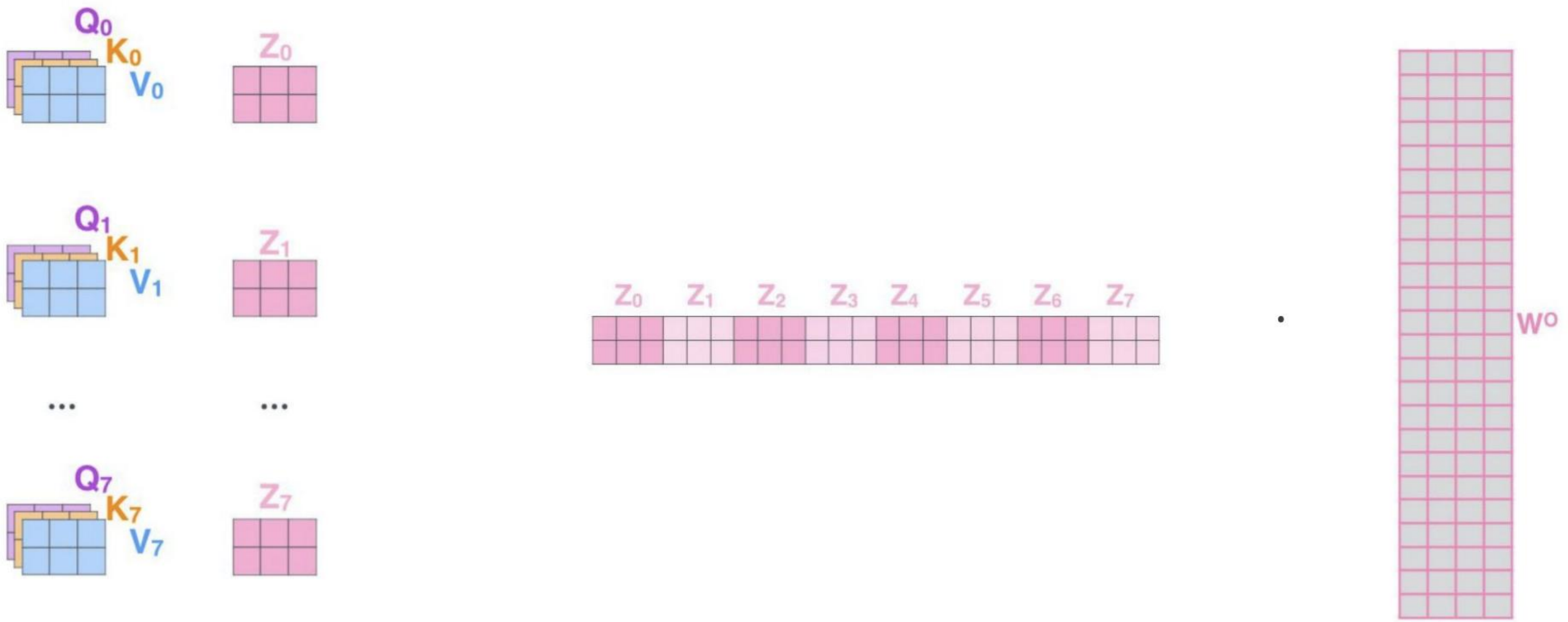


Because all these operations are linear, we can “multiply through” the residual stream.



2. Transformer Overview – Attention Heads are Independent and Additive

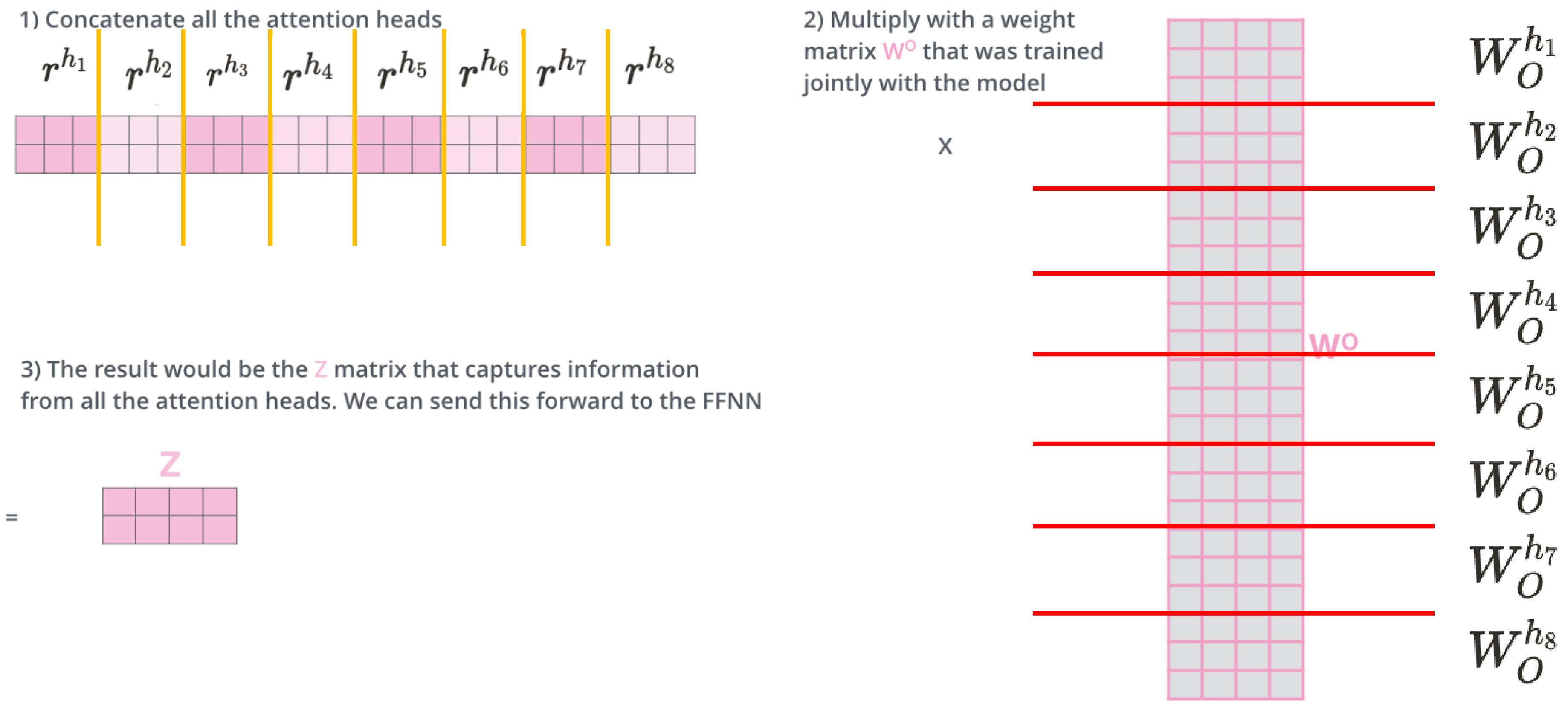
$$W_O^H \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \begin{bmatrix} W_O^{h_1}, \ W_O^{h_2}, \ \dots \end{bmatrix} \cdot \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \sum_i W_O^{h_i} r^{h_i}$$



- [Jay Alammar, The Illustrated Transformer.]

2. Transformer Overview – Attention Heads are Independent and Additive

$$W_O^H \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \left[W_O^{h_1},\ W_O^{h_2},\ \dots \right] \cdot \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \sum_i W_O^{h_i} r^{h_i}$$



- [Jay Alammar, The Illustrated Transformer.](#)

2. Transformer Overview – Attention Heads are Independent and Additive

$$W_O^H \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \begin{bmatrix} W_O^{h_1}, \ W_O^{h_2}, \ \dots \end{bmatrix} \cdot \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \dots \end{bmatrix} = \sum_i W_O^{h_i} r^{h_i}$$

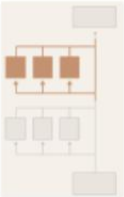
- **Specialized Attention Heads:**
  - **Previous Token Heads**
  - **Copying Heads**
  - **Induction Heads**
  - **…**

- While the idea that **attention heads are independent and additive** may seem unimportant at first, it provides a powerful lens:
- Transformer behavior emerges from **the composition of specialized heads**, each performing distinct and meaningful roles.

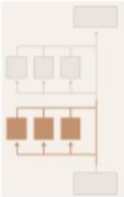
2. **Transformer Overview** – Tensor Product

- **Understanding the tensor product is essential** for analyzing how Transformers apply attention across positions and dimensions.

$$T = \mathrm{Id}\otimes W_U \cdot \left(Id + \sum_{h\in H_2} A^h \otimes W_{OV}^h\right) \cdot \left(Id + \sum_{h\in H_1} A^h \otimes W_{OV}^h\right) \cdot \mathrm{Id}\otimes W_E$$



The second **attention layer** has multiple attention heads which add into the residual stream

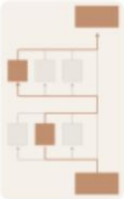


The first **attention layer** has multiple attention heads which add into the residual stream

$$= \mathrm{Id}\otimes W_U W_E + \sum_{h\,\in\, H_1\cup H_2} A^h \otimes (W_U W_{OV}^h W_E) + \sum_{h_2\,\in\, H_2}\ \sum_{h_1\,\in\, H_1} (A^{h_2} A^{h_1}) \otimes (W_U W_{OV}^{h_2} W_{OV}^{h_1} W_E)$$

"Direct path" term contributes to bigram statistics.

The **individual attention head** terms describe the effects of individual attention heads in linking input tokens to logits, similar to those we saw in the one layer model.



The **virtual attention head** terms correspond to V-composition of attention heads. They function a lot like individual attention heads, with their own attention patterns (the compositon of the heads patterns) and own OV matrix.

- Tensor Product multiplies **per position** or **across positions**.

2. Transformer Overview – Tensor Product

- A product like $Id \otimes W_V$ with **identity on the left**) represents multiplying each position in our context by a matrix. (**per position**)

$$\mathrm{Id}_3 \otimes W_V = \begin{bmatrix} 1 \cdot W_V & 0 & 0 \\ 0 & 1 \cdot W_V & 0 \\ 0 & 0 & 1 \cdot W_V \end{bmatrix}$$

$$(\mathrm{Id}_3 \otimes W_V)x = \begin{bmatrix} 1 \cdot W_V & 0 & 0 \\ 0 & 1 \cdot W_V & 0 \\ 0 & 0 & 1 \cdot W_V \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} W_V x_1 \\ W_V x_2 \\ W_V x_3 \end{bmatrix}$$

- A product like $A \otimes Id$ (with **identity on the right**) represents multiplying **across positions**.

$$A = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} \qquad V = \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix}$$

$$A \otimes I_3 = \begin{bmatrix} 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 \\ 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 \\ 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 \end{bmatrix}$$

2. Transformer Overview – Tensor Product

- A product like $A \otimes Id$ (with **identity on the right**) represents multiplying **across positions**.

$$A = \begin{bmatrix} \boxed{0.6} & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} \qquad V = \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix}$$

$$A \otimes I_3 = \begin{bmatrix} 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 \\ 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 \\ 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 \end{bmatrix}$$

2. Transformer Overview – Tensor Product

- A product like $A \otimes Id$ (with **identity on the right**) represents multiplying **across positions**.

$$A = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} \qquad V = \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix}$$

$$(A \otimes I_3)V = \begin{bmatrix} 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0.6 & 0 & 0 & 0.3 & 0 & 0 & 0.1 \\ 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & 0.5 & 0 & 0 & 0.3 \\ 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0.3 & 0 & 0 & 0.6 \end{bmatrix} \begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \\ V_{21} \\ V_{22} \\ V_{23} \\ V_{31} \\ V_{32} \\ V_{33} \end{bmatrix} = \begin{bmatrix} 0.6V_{11} + 0.3V_{21} + 0.1V_{31} \\ 0.6V_{12} + 0.3V_{22} + 0.1V_{32} \\ 0.6V_{13} + 0.3V_{23} + 0.1V_{33} \\ 0.2V_{11} + 0.5V_{21} + 0.3V_{31} \\ 0.2V_{12} + 0.5V_{22} + 0.3V_{32} \\ 0.2V_{13} + 0.5V_{23} + 0.3V_{33} \\ 0.1V_{11} + 0.3V_{21} + 0.6V_{31} \\ 0.1V_{12} + 0.3V_{22} + 0.6V_{32} \\ 0.1V_{13} + 0.3V_{23} + 0.6V_{33} \end{bmatrix}$$

2. Transformer Overview – Tensor Product

- **Mixed-product property**

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$

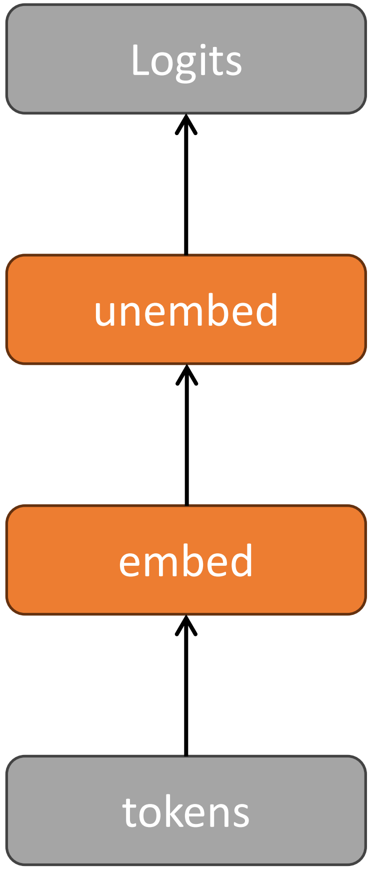
- **Attention Layer**

$$h(x) = \underbrace{(\mathrm{Id} \otimes W_O)}_{\text{Project result vectors out for each token } (h(x)_i = W_O r_i)} \cdot \underbrace{(A \otimes \mathrm{Id})}_{\text{Mix value vectors } \textit{across} \text{ tokens to compute result vectors } (r_i = \sum_j A_{i,j} v_j)} \cdot \underbrace{(\mathrm{Id} \otimes W_V)}_{\text{Compute value vector for each token } (v_i = W_V x_i)} \cdot x$$

$$h(x) = \underbrace{(A \otimes W_O W_V)}_{A \text{ mixes across tokens while } W_O W_V \text{ acts on each vector independently.}} \cdot x$$

3. Zero-Layer Transformers

3. Zero-Layer Transformers



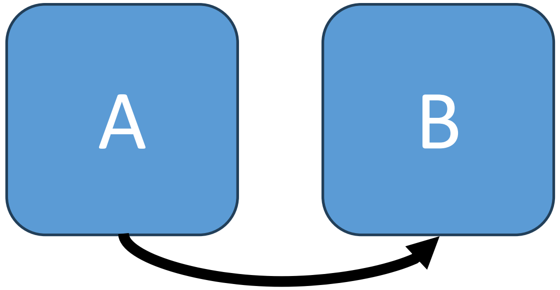
Logit

$\mathrm{Logits} = \mathrm{W_U x_0}$

$\mathrm{x_0} = \mathrm{W_E t}$

t

$\mathrm{Logits} = \mathrm{W_U W_E t}$



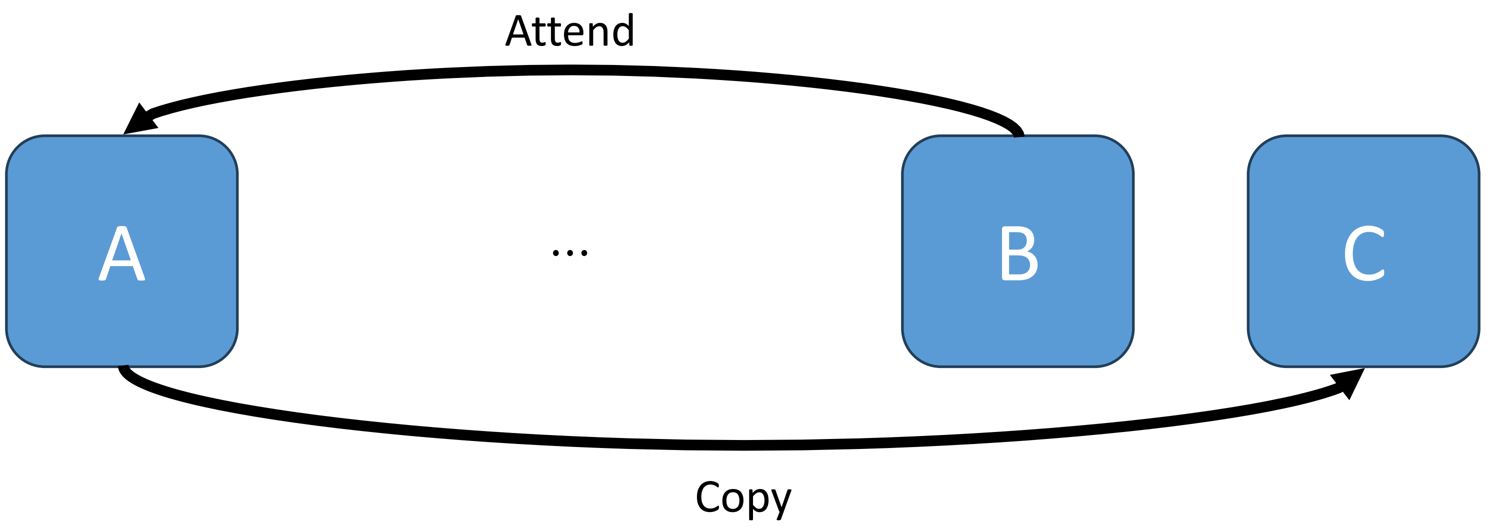
Bigram Model = $p(w_i|w_{i-1})$

- Model **cannot** move information from **other token**
- $W_U W_E$: Approximate **bigram log-likelihood**
- "**Direct path**": token embedding flows directly down the residual stream to the unembedding
- "Barack" is often followed by "Obama"

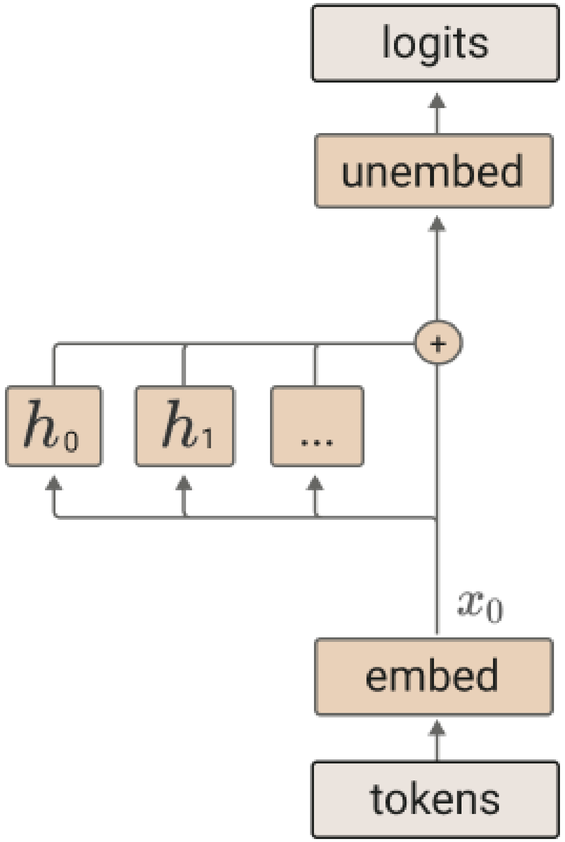
4. One-Layer Transformers

4. One-Layer Transformers – Overview

- Ensemble of a **bigram model** and several "**skip-trigram**" models
- Skip-trigram models: ("A…BC")
- Attend from the **present token**("B") to a **previous token** ("A") and copy information to **next tokens** ("C")



4. One-Layer Transformers – One-Layer



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_1$$

Each attention head, h, is run and added to the residual stream.

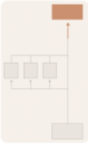
$$x_1 = x_0 + \sum_{h \in H} h(x_0)$$

Token embedding.

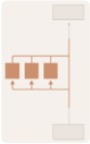
$$x_0 = W_E t$$

4. One-Layer Transformers – The Path Expansion Trick

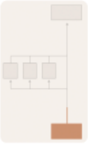
$$T = \mathrm{Id} \otimes W_U \cdot \left(Id + \sum_{h \in H_1} A^h \otimes W_{OV}^h\right) \cdot \mathrm{Id} \otimes W_E = \mathrm{Id} \otimes W_U W_E + \sum_{h \in H} A^h \otimes (W_U W_{OV}^h W_E)$$



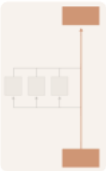
The token **unembedding** maps residual stream vectors to logits.



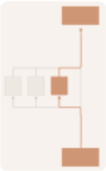
The **attention layer** has multiple heads. The result of each is added into the residual stream.



The token **embedding** maps tokens to residual stream vectors.



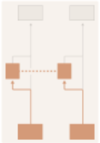
"Direct path" term contributes to bigram statistics.



The **attention head** terms describe the effects of attention heads in linking input tokens to logits. A^h describes which tokens are attended to while $W_U W_{OV}^h W_E$ describes how each token changes the logits if attended to.

where $A^h = \mathrm{softmax}^*\left(t^T \cdot W_E^T W_{QK}^h W_E \cdot t\right)$

Softmax with autoregressive masking



Attention pattern logits are produced by multiplying pairs of tokens through different sides of W_{QK}^h.

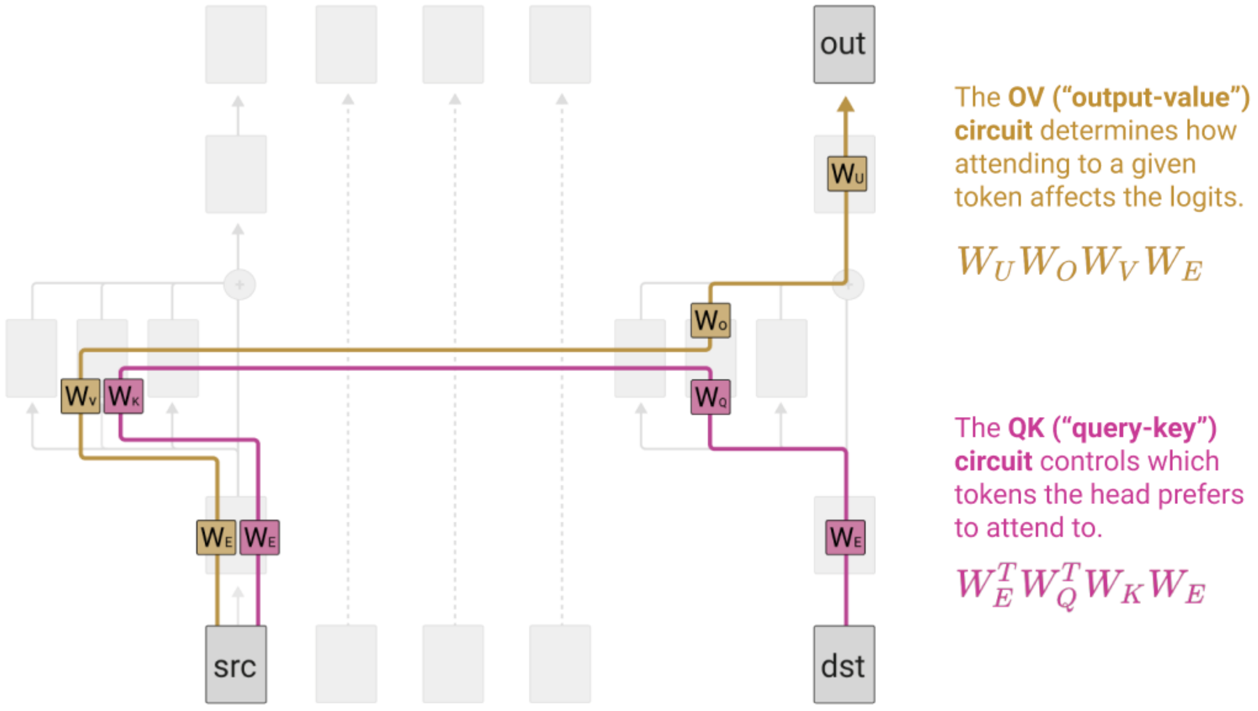
Zero-layer Transformer

- **Sum of end-to-end path**

- A product like $A \otimes W$ multiplies the vector at each position by W and across positions with A. It doesn't matter which order you do this in.
- The products obey the mixed-product property $(A \otimes B) \cdot (C \otimes D) = (AC) \otimes (BD)$.

4. One-Layer Transformers – Query-Key and Output-Value Circuits

$$A^h \otimes (W_U W_{OV}^h W_E) \text{ where } A^h = \text{softmax}(t^T \cdot W_E^T W_{QK}^h W_E \cdot t)$$



Two $[n_{vocab}, n_{vocab}]$ matrices

- $W_E^T W_{QK}^h W_E$ **("Query-Key(QK) circuit")** : how much a query token "wants" to **attend** to a key token
- $W_U W_{OV}^h W_E$ **("Output-Value(OV) circuit")** : how a given token will **affect** the output logit if attended to
- "**Frozen**" attention pattern
- Logits are **a linear function** of the tokens

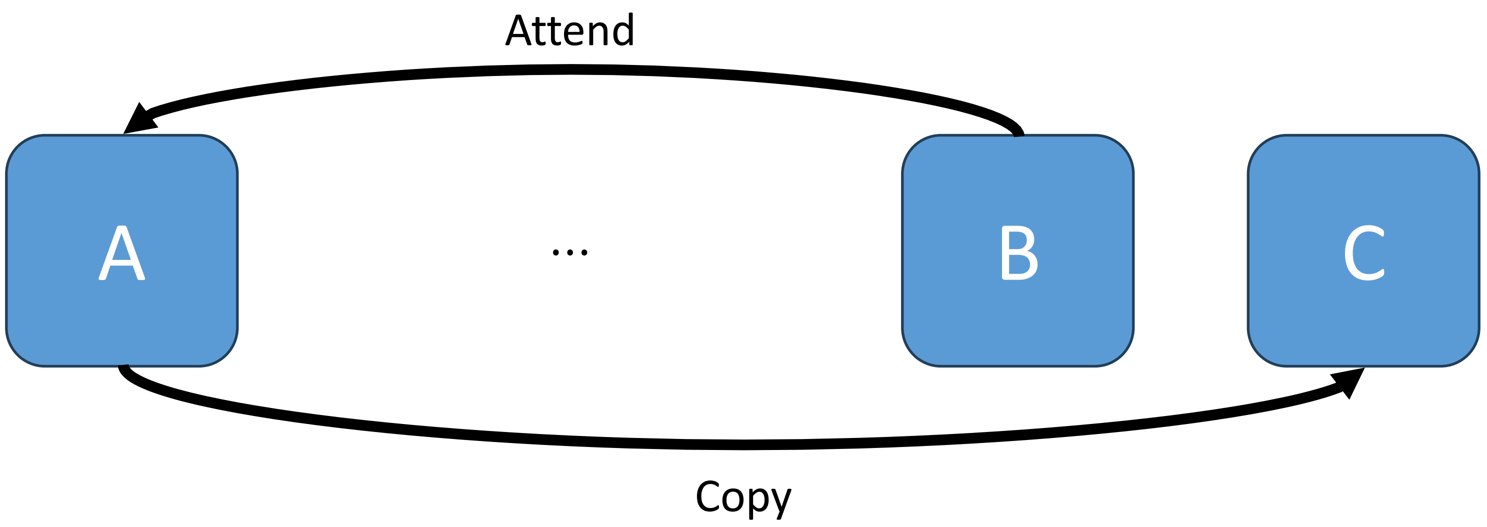
4. One-Layer Transformers – Skip-Trigrams

Skip-Trigram: [source]… [destination][out]

- Copying / primitive in-context learning
- Other interesting skip-trigrams
- Primarily positional attention heads
- Skip-trigram “Bugs”

4. One-Layer Transformers – Skip-Trigrams: Copying

- Dedicate an enormous fraction of their capacity to **copying**
- The OV circuit set things up so that tokens, if attended by the head, **increase the probability of that token and similiar tokens**



4. One-Layer Transformers – Skip-Trigrams: Copying

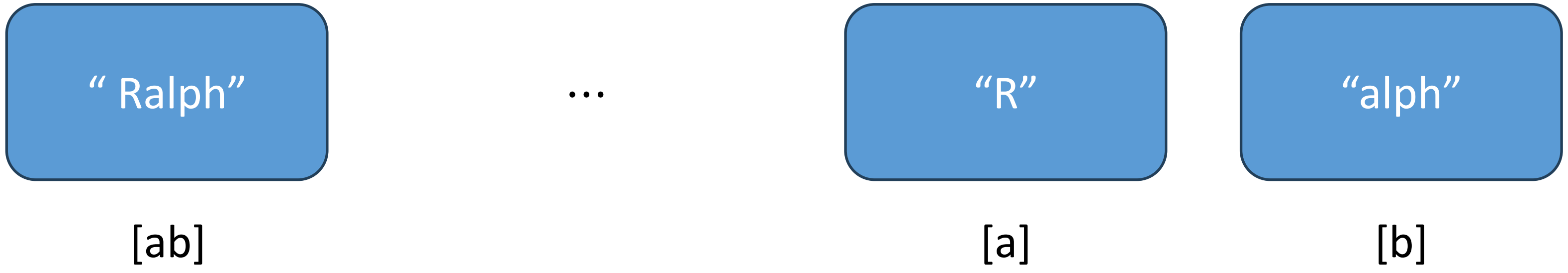
- **Fixed** source token
- Largest QK entries (destination)
- Largest OV entries (out)

Some examples of large entries QK/OV circuit

Source Token	Destination Token	Out Token	Example Skip Tri-grams	
“ perfect”	“ are”, “ looks”, “ is”, “ provides”	“ perfect”, “ super”, “ absolute”, “ pure”	“ perfect... are perfect”, “ perfect... looks super”	
“ large”	“ contains”, “ using”, “ specify”, “ contain”	“ large”, “ small”, “ very”, “ huge”	“ large... using large”, “ large... contains small”	
“ two”	“ One”, “\n ”, “ has”, “\r\n ”, “One”	“ two”, “ three”, “ four”, “ five”, “ one”	“ two... One two”, “ two... has three”	
“lambda”	“ $\\”, “}{\\”, “+\\”, “(\\”, “ ${\\”	“lambda”, “sorted”, “ lambda”, “operator”	“lambda... $\\lambda”, “lambda... +\\lambda”	LaTex
“nbsp”	“&”, “ \"&”, “}&”, “>&”, “=&”	“nbsp”, “01”, “gt”, “00012”, “nbs”, “quot”	“nbsp... ”, “nbsp... > ”	HTML
“Great”	“The”, “ The”, “ the”, “ contains”, “ /”	“ Great”, “ great”, “ poor”, “ Every”	“Great... The Great”, “Great... the great”	

4. One-Layer Transformers – Skip-Trigrams: Primitive In-context learning

- Tokenizers typically **merge spaces onto the start of words** (“ Ralph”)
- Less common words
  - common to map to a single token when a space is in front of them (“ Ralph” -> [“ Ralph”])
  - **split** when there isn’t a space (“Ralph” -> [“R”, “alph”])
- Can observe attention heads which handle **copying for words that split into two tokens without a space**
- When observing a fragmented token, then **attend back to complete tokens with a space** and then **predict the continuation**
- Kind of **mimicking** the induction heads ([A][B]…[A][B])



4. One-Layer Transformers – Skip-Trigrams: Primitive In-context learning

More examples of large entries QK/OV circuit

Source Token	Destination Token	Out Token	Example Skip Tri-grams
"indy"	" C", "C", " V", "V", " R", " c"	"indy", "obby", "INDY", "loyd"	"indy... Cindy", "indy... CINDY"
" Pike"	" P", "P", "V", "Sp", " V", "R"	"ike", "ikes", "ishing", "owler"	" Pike... Pike", " Pike... Spikes"
" Ralph"	"R", " R", " P", "P", "V", " r"	"alph", "ALPH", "obby", "erald"	" Ralph... Ralph", " Ralph... RALPH"
" Lloyd"	"L", " L", " P", "P", "R", " C"	"loyd", "alph", "\n ", "acman", ... "atherine"	" Lloyd... Lloyd", " Lloyd... Catherine"
" Pixmap"	" P", " Q", "P", " p", " U"	"ixmap", "Canvas", "Embed", "grade"	" Pixmap... Pixmap", " Pixmap... QCanvas"

4. One-Layer Transformers – Skip-Trigrams: Primitive In-context learning

Primitive In-Context Learning Patterns

[b]...[a]→[b]	[b]...[a]→[b']	[ab]...[a]→[b]	[ab]...[a]→[b']
[two]...[One]→[two]	[two]...[has]→[three]	[Ralph]...[R]→[alph]	[Ralph]...[R]→[ALPH]
[perfect]...[are]→[perfect]	[perfect]...[looks]→[super]	[Pike]...[P]→[ike]	[Pike]...[P]→[ikes]
[nbsp]...[&]→[nbsp]	[nbsp]...[&]→[gt]	[Pixmap]...[P]→[ixmap]	
[lambda]...[$\\]→[lambda]	[lambda]...[$\\]→[operator]	[Lloyd]...[L]→[loyd]	

- More interesting and powerful algorithm in two-layer transformer

4. One-Layer Transformers – Other interesting skip-trigrams

- **[Python]** Predicting that the python keywords `else`, `elif` and `except` are more likely after an indentation is reduced using skip-trigrams of the form: `\n\t\t\t … \n\t\t → else/elif/except` where the first part is indented N times, and the second part $N-1$, for various values of N, and where the whitespace can be tabs or spaces.
- **[Python]** Predicting that `open()` will have a file mode string argument: `open … "," → [rb / wb / r / w]` (for example `open("abc.txt","r")`)
- **[Python]** The first argument to a function is often `self`: `def … ( → self` (for example `def method_name(self):`)
- **[Python]** In Python 2, `super` is often used to call `.__init__()` after being invoked on `self`: `super … self → ).__` (for example `super(Parent, self).__init__()`)

4. One-Layer Transformers – Other interesting skip-trigrams

- **[Python]** increasing probability of method/variables/properties associated with a library: `upper … . → upper/lower/capitalize/isdigit`, `tf … . → dtype/shape/initializer`, `datetime… → date / time / strftime / isoformat`, `QtWidgets … . → QtCore / setGeometry / QtGui`, `pygame … . → display / rect / tick`
- **[Python]** common patterns `for... in [range/enumerate/sorted/zip/tqdm]`
- **[HTML]** `tbody` is often followed by `<td>` tags: `tbody … < → td`
- **[Many]** Matching of open and closing brackets/quotes/punctuation: `(** … X → **)`, `(' … X → ')`, `"% … X → %"`, `'</ … X → >'` (see 32 head model, head 0:27)

4. One-Layer Transformers – Other interesting skip-trigrams

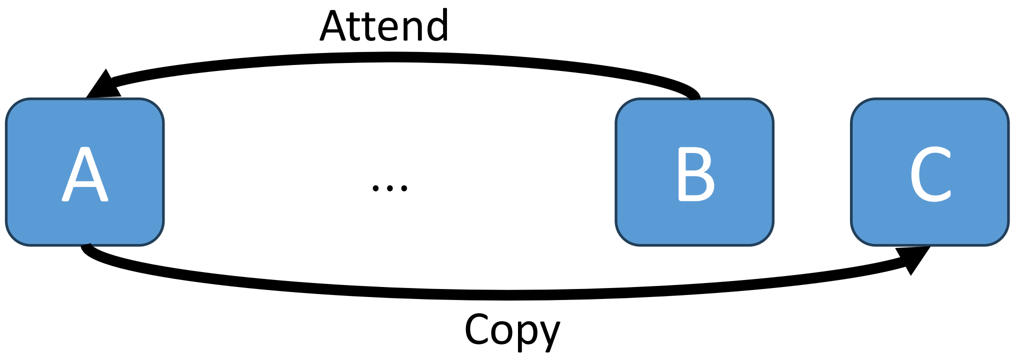
- **[LaTeX]** In LaTeX, every `\left` command must have a corresponding `\right` command; conversely `\right` can only happen after a `\left`. As a result, the model predicts that future LaTex commands are more likely to be `\right` after `\left`: `left … \ → right`
- **[English]** Common phrases and constructions (e.g. `keep … [in → mind / at → bay / under → wraps]`, `difficult … not → impossible`)
  - For a single head, here are some trigrams associated with the query `" and"`: `back and → forth`, `eat and → drink`, `trying and → failing`, `day and → night`, `far and → away`, `created and → maintained`, `forward and → backward`, `past and → present`, `happy and → satisfied`, `walking and → talking`, `sick and → tired`, … (see 12 head model, head 0:0)
- **[URLs]** Common URL schemes: `twitter … / → status`, `github … / → [issues / blob / pull / master]`, `gmail … . → com`, `http … / → [www / google / localhost / youtube / amazon]`, `http … : → [8080 / 8000]`, `www … . → [org / com / net]`

Zero-layer transformer (**Bigram log-likelihood**)

One-layer transformer (Skip-Trigram)

- Copying / primitive in-context learning
- Other interesting skip-trigrams
- Primarily positional attention heads
- Skip-trigram "Bugs"

Skip-Trigram: [source]… [destination][out]



4. One-Layer Transformers – Primarily positional attention heads

Some examples of large entries QK/OV Circuit for Primarily Positional Heads

Source Token	Destination Token	Out Token	Examples
" corresponding"	*Primarily Positional*	" to", "to", " for", "markup", " with"	" corresponding to", " correspoding with"
" coinc"	*Primarily Positional*	" with", " closely", "with", " con"	" coinc[ides] with", " coinc[ides] closely"
" couldn"	*Primarily Positional*	" resist", " compete", " stand", " identify"	" couldn['t] resist", " couldn['t] stand"
" shouldn"	*Primarily Positional*	" have", " be", " remain", " take"	" shouldn['t] have", " shouldn['t] be"

- Attends to the **present** token or the **previous** token
- W_{QK} works like the **rotational matrix**
- Can select for any relative positional offset by rotating the dimensions containig **sinusoidal** information

4. One-Layer Transformers – Skip-trigram "Bugs"

- Skip-trigram in a "**factored form**" split between the OV and QK matries
- Representing a function $\mathrm{f(a, b, c) = f_1(a, b)\ f_2(a, c)}$
- **Can't capture** the three way **interaction flexibly**
- head increases the probability of both keep … in mind and keep … at bay
    - must also increase the probability of keep … in bay and keep … at mind
- An early demontration of using interpretability to **understand model failures**

4. One-Layer Transformers – Skip-trigram "Bugs"

Limited Expressivity Can Create Bugs which Seem Strange from the Outside

Source Token	Destination Token	Out Token	"Correct" Skip Tri-grams	"Bug" Skip Tri-grams
" Pixmap"	" P", " Q", "P", " p", " U"	"ixmap", "Canvas", "Embed", "grade"	" Pixmap... Pixmap", " Pixmap... QCanvas"	" Pixmap... PCanvas"

Source Token	Destination Token	Out Token	"Correct" Skip Tri-grams	"Bug" Skip Tri-grams
" Lloyd"	"L", " L", " P", "P", "R", " C"	"loyd", "alph", "\n ", "acman", ... "atherine"	" Lloyd... Lloyd", " Lloyd... Catherine"	" Lloyd... Cloyd", " Lloyd... Latherine"

Source Token	Destination Token	Out Token	"Correct" Skip Tri-grams	"Bug" Skip Tri-grams
" keep"	" in", " at", " out", " under", " off"	" bay", ... " mind", ... " wraps"	" keep... in mind", " keep... at bay", " keep... under wraps"	" keep... in bay", " keep... at wraps", " keep... under mind"

- Early demonstration of using interpretability to understand **model failures**

4. One-Layer Transformers – OV/QK Matrices

Qualitative analysis -> Quantitative analysis

How do we automatically **detect copying heads**?

- OV and QK matrices are **extremely low rank**
  - 50,000 x 50,000, but **rank** is 64 or 128
- Reveals hints of much **simpler structure**
  - Like **cluster structure**
- Copying behavior is **widespread**

=> Eigendecomposition

4. One-Layer Transformers – OV Matrices Eigendecomposition

$$Mv_i = \lambda_i v_i$$

$$v_i \coloneqq \text{ eigenvector} \quad \lambda_i \coloneqq \text{ eigenvalue}$$

$$M = W_U W_{OV}^h W_E$$

$$v_i = a_1 t_1 + a_2 t_2 + \cdots + a_n t_n$$

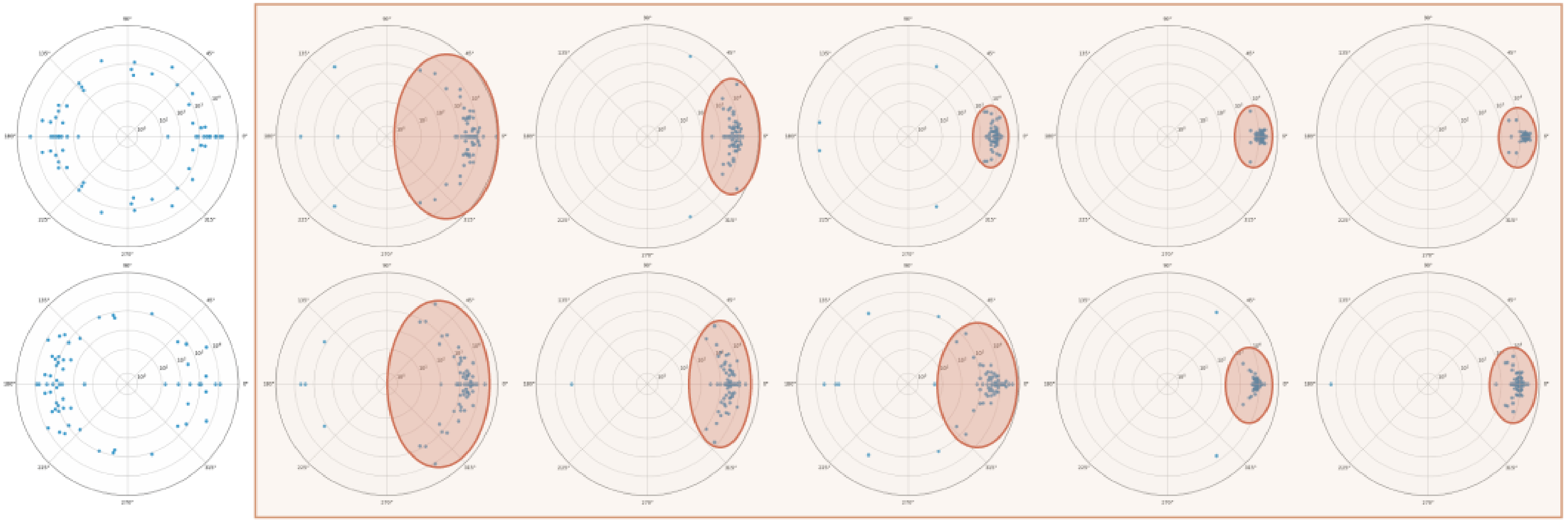
- If λ_i is a **postive** real number, there's **a linear combination of tokens** which **increases** **the linear combination of logits** of those same tokens
- A set of tokens mutually increase their own probability
  - Tokens with Plural words
  - Tokens starting with a given first letter
- Eigenvectors have both **positive** and **negative** entries => there are **two sets** of tokens
  - Increase probability in the same set
  - Decrease probability in those other set

Copying requires λ_i to be positive

4. One-Layer Transformers – OV Matrices Eigendecomposition

Eigenvalue analysis of **first layer** attention head OV circuits

10/12 of layer 1 heads have mostly positive OV eigenvalues, and appear to significantly perform copying

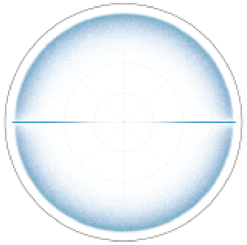


non-positive eigenvalues not copying heads?

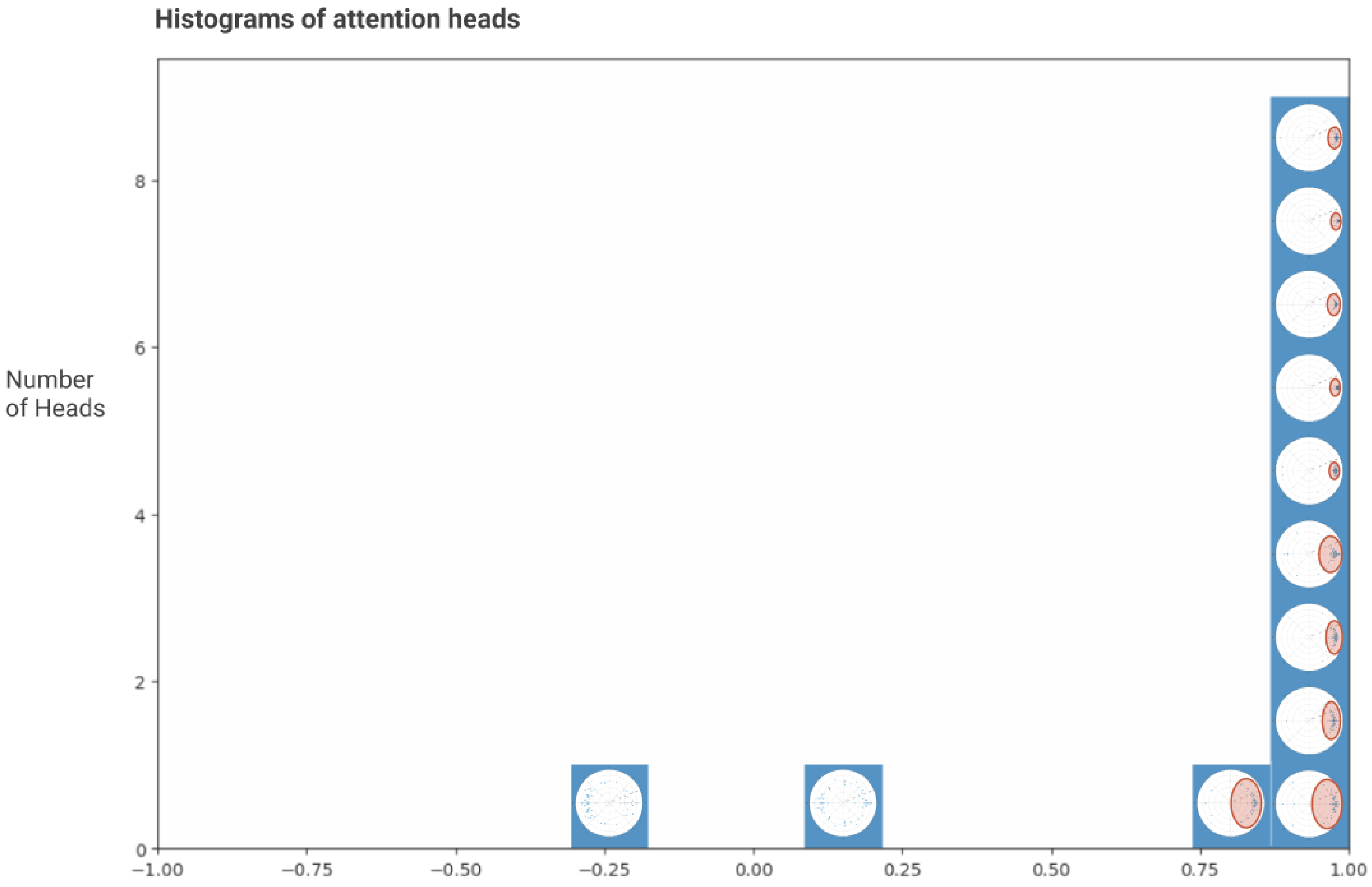
positive eigenvalues copying heads?

We use a **log scale** to represent magnitude, since it varies by many orders of magnitude.

Eigenvalue distribution for randomly initialized weights. Note that the mostly — and in some cases, entirely— positive eigenvalues we observe are very different from what we randomly expect.



4. One-Layer Transformers – OV Matrices Eigendecomposition



$$\frac{\sum_i \lambda_i}{\sum_i |\lambda_i|}$$

Histograms of attention heads by the fraction of their eigenvalues that are positive can be a useful summary.

4. One-Layer Transformers – OV Matrices Eigendecomposition

- Author is not fully sure of eigenvalue-based summary statistic
- **Positive eigenvalue** -> might not mean copying matrix
  - Map some token to decreasing the logits of that same token
  - "**Copying on average**"
- Detecting "Copying Matrices" by other ways
  - **Diagonal of OV matrix**: how each token affects its own probability
    - Positive-leaning
  - $\mathrm{Tr}(M) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i$

$$\begin{bmatrix} a_{11} & & \\ & \ddots & \\ & & a_{nn} \end{bmatrix}$$

5. Two-Layer Transformers

Contents of Two-Layer Transformers

1) Recap One-layer Transformer

2) What happens with a Two-layer Transformer

3) Overview of Induction Head

4) How Induction heads work

5) Composition

6) Term Importance Analysis

7) Virtual Attention Head

5. Two-Layer Transformers – 1) Recap One-layer Transformers

Some examples of large entries QK/OV circuit

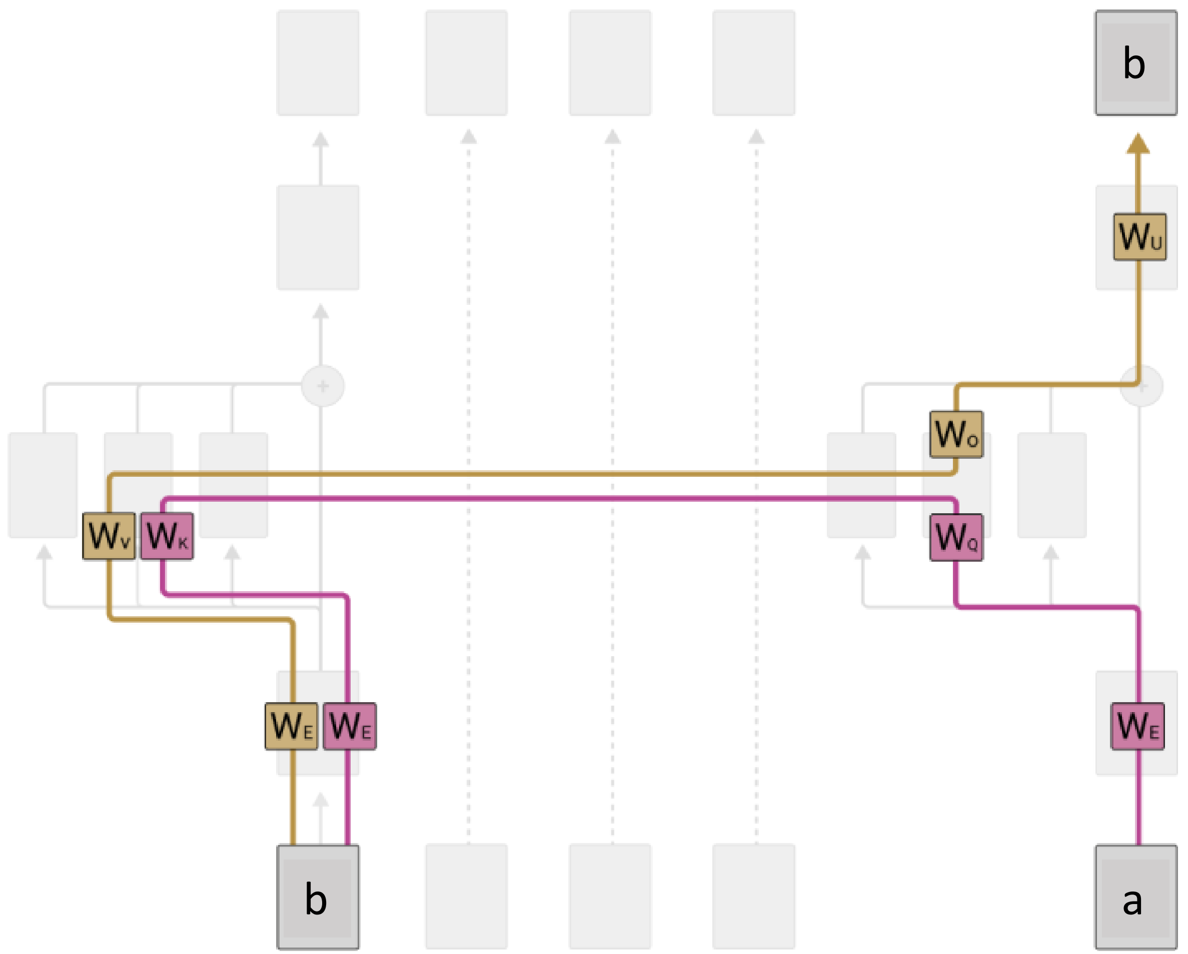
Source Token	Destination Token	Out Token	Example Skip Tri-grams
" perfect"	" are", " looks", " is", " provides"	" perfect", " super", " absolute", " pure"	" perfect... are perfect", " perfect... looks super"
" large"	" contains", " using", " specify", " contain"	" large", " small", " very", " huge"	" large... using large", " large... contains small"
" two"	" One", "\n ", " has", "\r\n ", "One"	" two", " three", " four", " five", " one"	" two... One two", " two... has three"
"lambda"	" $\\", "}{\\", "+\\", "(\\", " ${\\"	"lambda", "sorted", " lambda", "operator"	"lambda... $\\lambda", "lambda... +\\lambda"
"nbsp"	"&", " \"&", "}&", ">&", "=&"	"nbsp", "01", "gt", "00012", "nbs", "quot"	"nbsp... ", "nbsp... > "
"Great"	"The", " The", " the", " contains", " /"	" Great", " great", " poor", " Every"	"Great... The Great", "Great... the great"

Primitive In-Context Learning Patterns

[b]...[a]→[b]	[b]...[a]→[b']	[ab]...[a]→[b]	[ab]...[a]→[b']
[two]...[One]→[two]	[two]...[has]→[three]	[Ralph]...[R]→[alph]	[Ralph]...[R]→[ALPH]
[perfect]...[are]→[perfect]	[perfect]...[looks]→[super]	[Pike]...[P]→[ike]	[Pike]...[P]→[ikes]
[nbsp]...[&]→[nbsp]	[nbsp]...[&]→[gt]	[Pixmap]...[P]→[ixmap]	
[lambda]...[$\\]→[lambda]	[lambda]...[$\\]→[operator]	[Lloyd]...[L]→[loyd]	

Copying!

5. Two-Layer Transformers – 1) Recap One-layer Transformers



The **OV ("output-value") circuit** determines how attending to a given token affects the logits.

$$W_U W_O W_V W_E$$

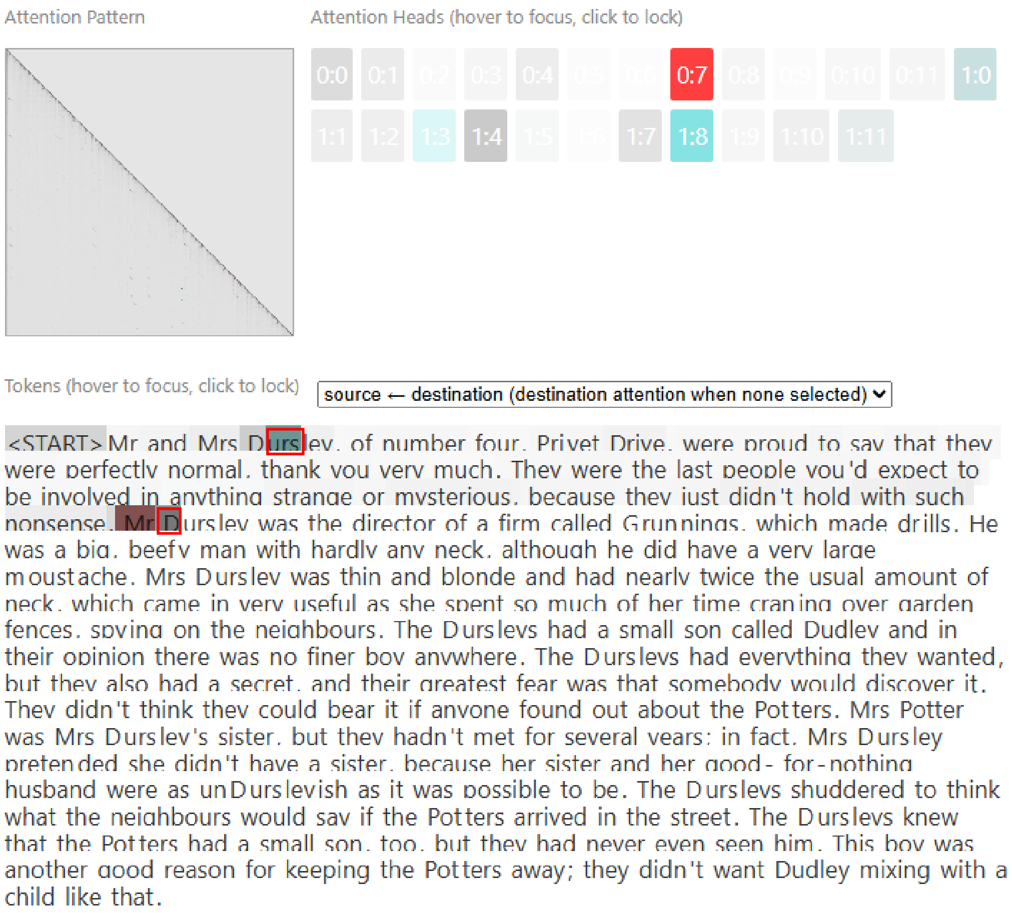
-> Increase the probability of the attended token and similar tokens to a lesser extent

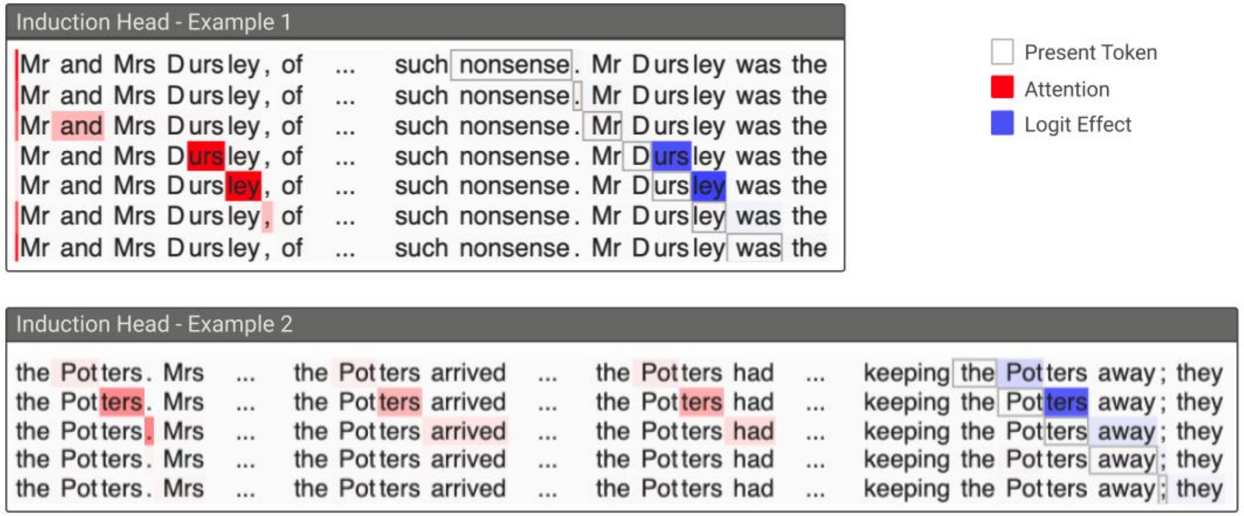
The **QK ("query-key") circuit** controls which tokens the head prefers to attend to.

$$W_E^T W_Q^T W_K W_E$$

-> Attends the token that could plausibly be the next token

5. Two-Layer Transformers – 2) What happens with a Two-layer Transformer



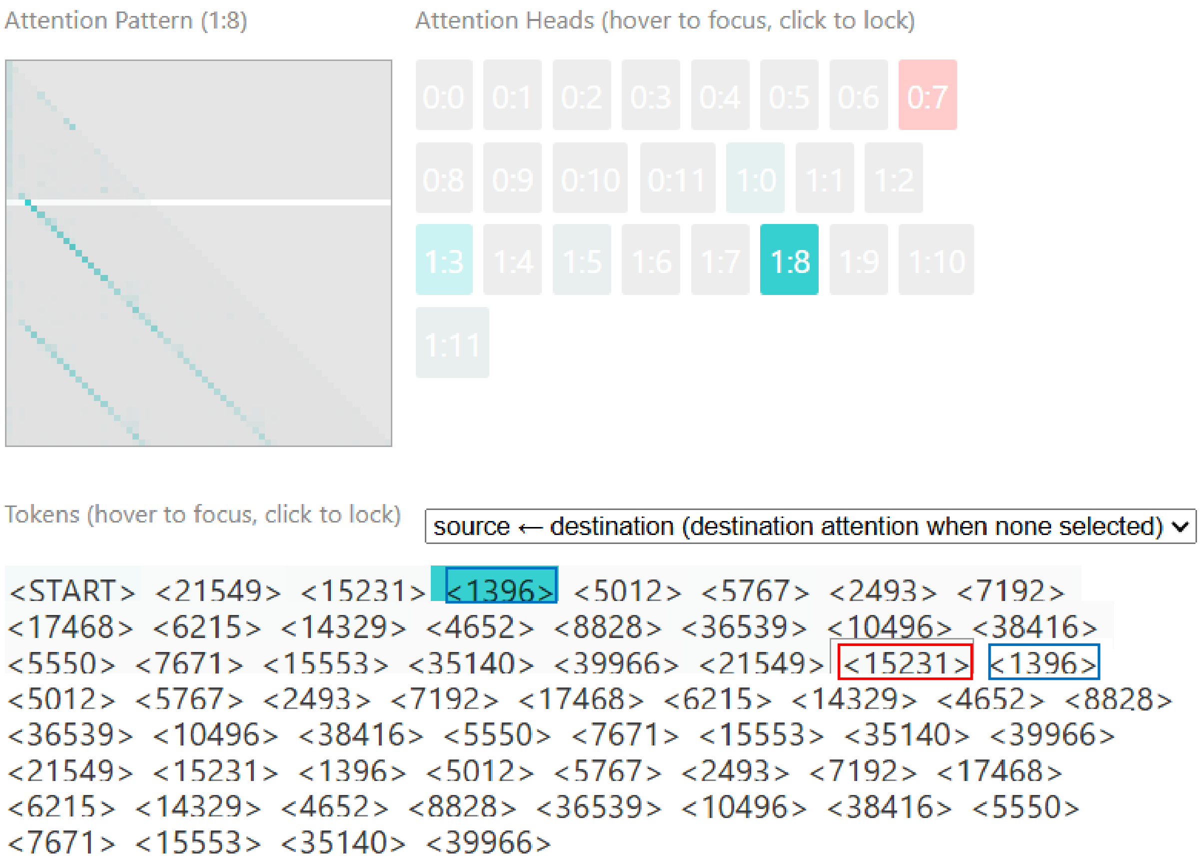


the attention patterns with attention weights scaled by $\left|\left|v_{src}^{h}\right|\right|$

-> how big a vector is moved from each position?

-> See how useful it consider each source token

5. Two-Layer Transformers – 2) What happens with a Two-layer Transformer



5. Two-Layer Transformers – 3) Overview of Induction head

What is the difference between copying and induction?

- **Copying**
    - Happens when tokens are plausible in terms of bigram-ish statistics
    - Looking for places it might be able to repeat a token

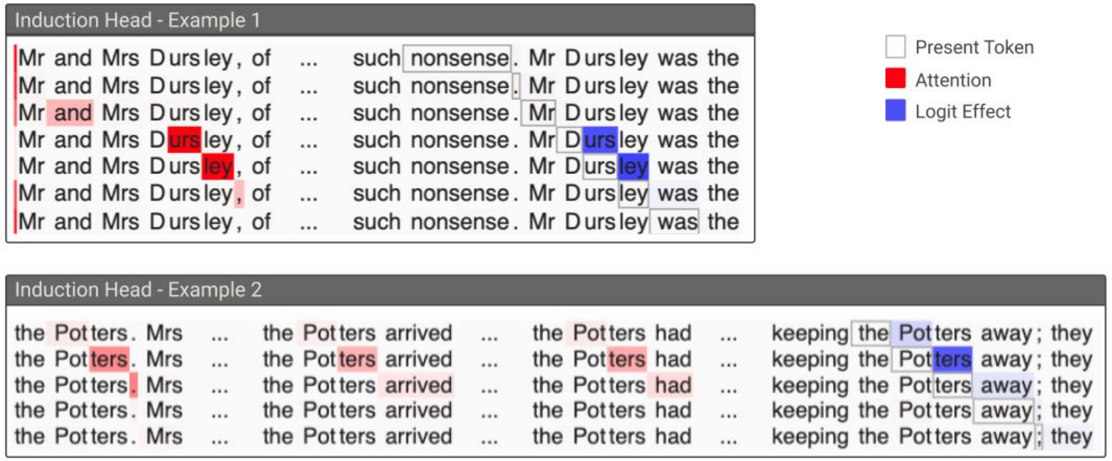
- **Induction**
    - Looks like an algorithm that doesn't depend on learned statistics about whether one token can plausibly follow another
    - Integrating the information about the context of the token by considering how the token was previously used and looks out for simliar cases

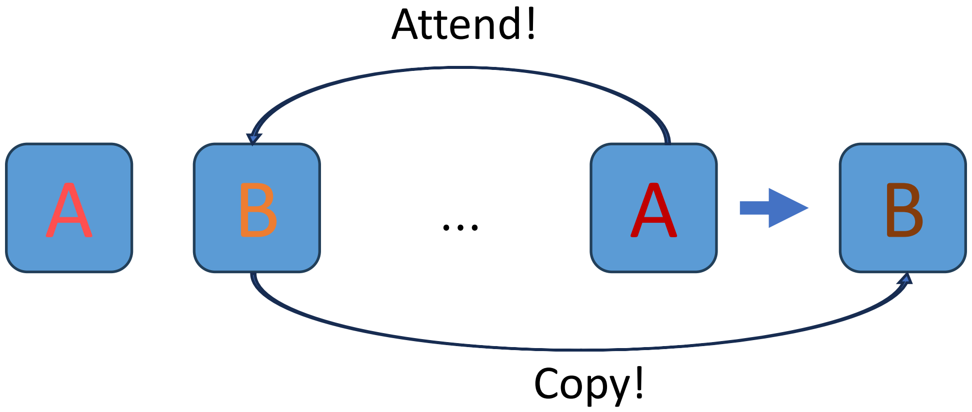
5. Two-Layer Transformers – 3) Overview of Induction head

Induction head

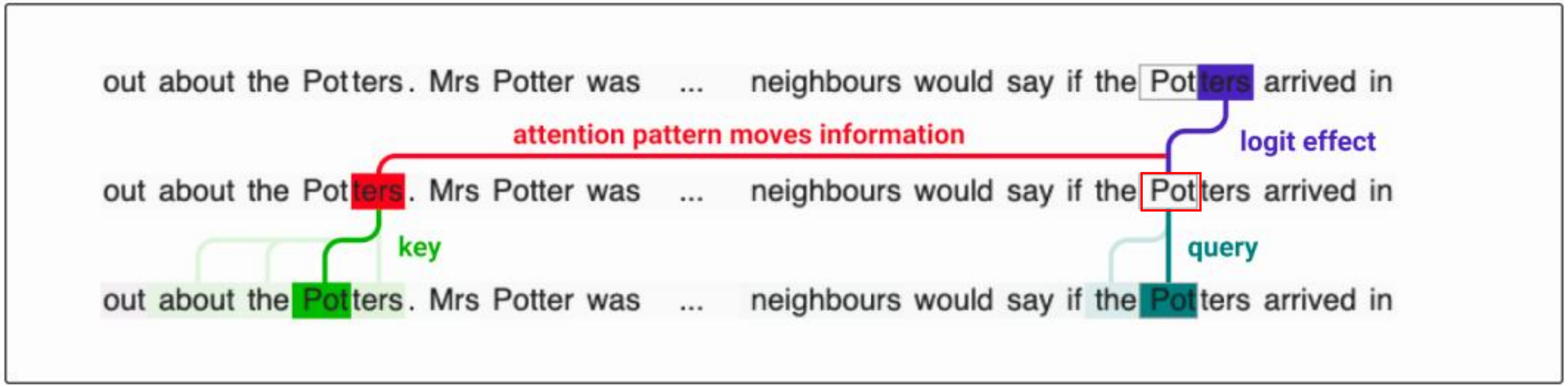
search previous examples of A\` in the context

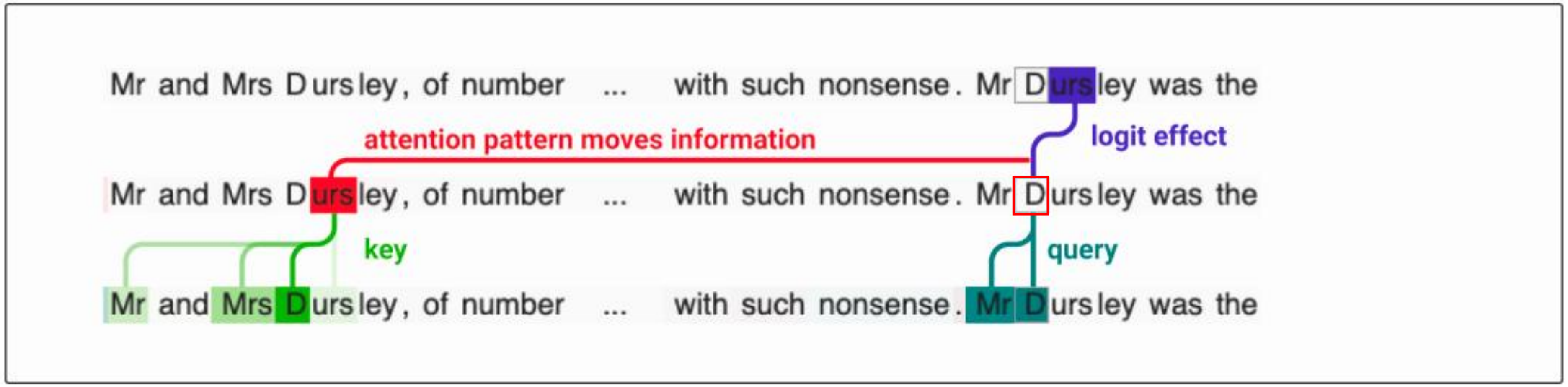
- if not find:
  - attend to the <*START*> token
- if find:
  - look at the next token B in previous case
  - copy the B to predict the next token B\`



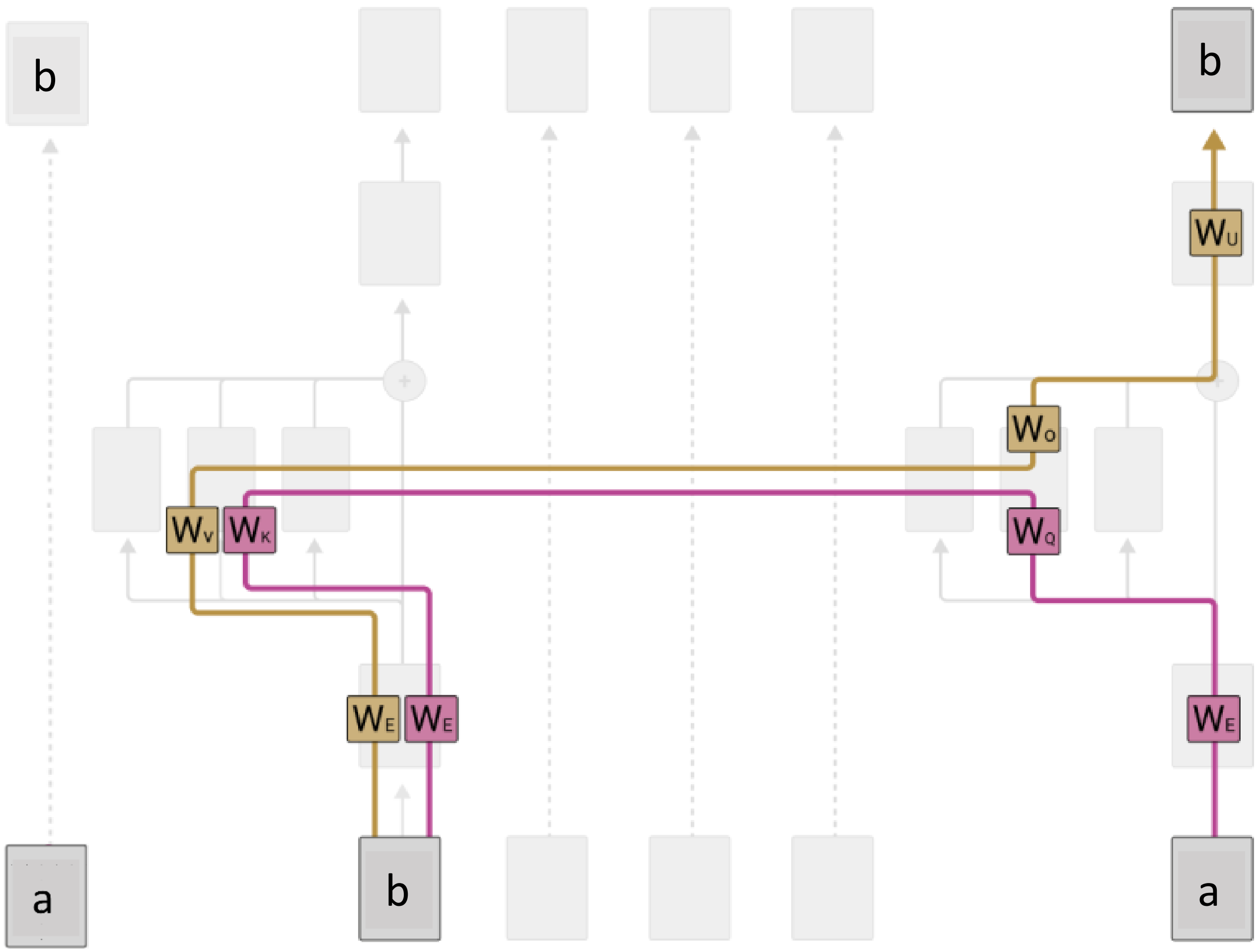


5. Two-Layer Transformers – 4) How Induction heads work





5. Two-Layer Transformers – 4) How Induction heads work



OV circuit

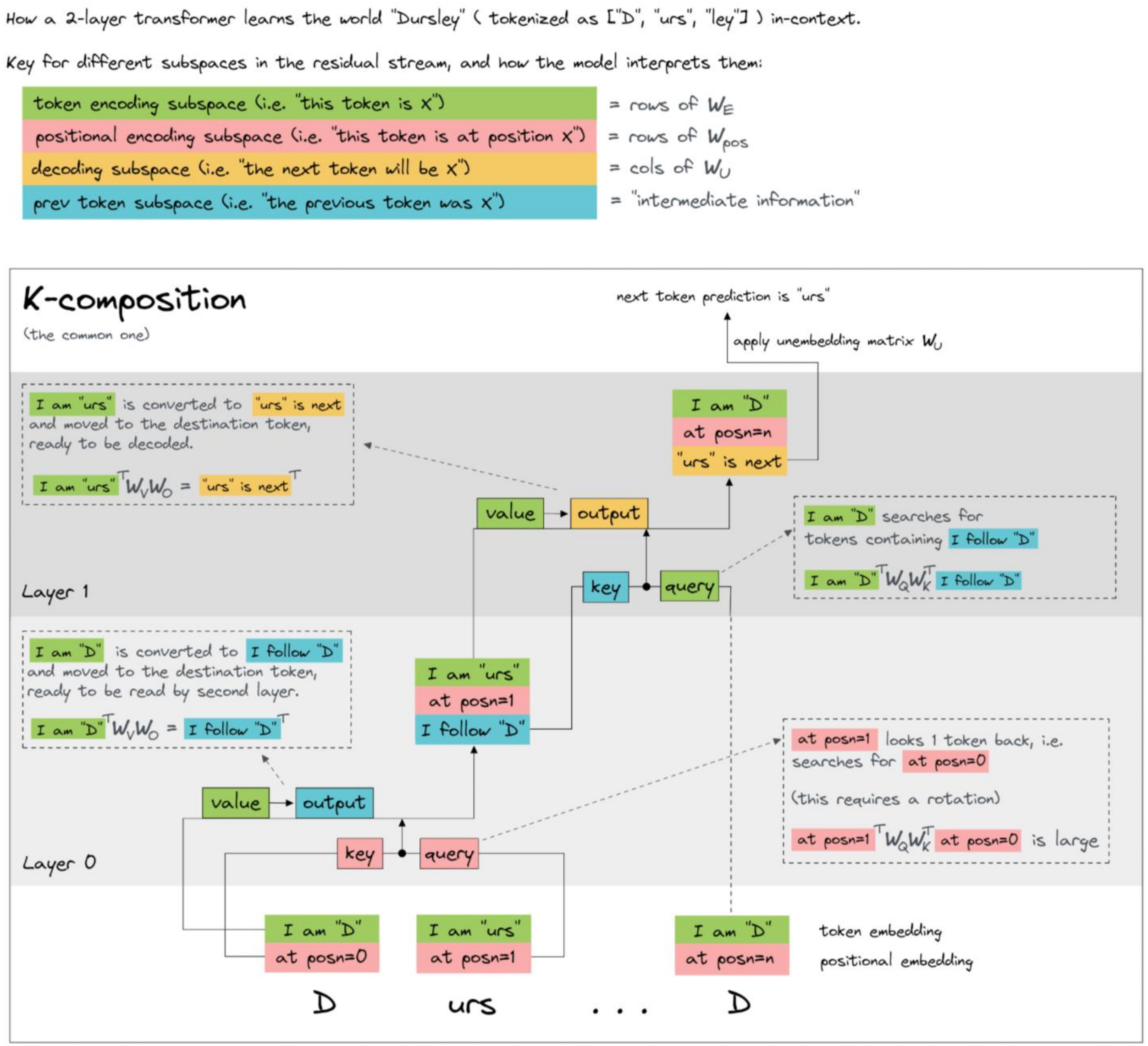
- Increase the probability of the attended token and similar tokens in to a lesser extent

QK circuit

~~- Attends the token that could plausibly be the next token~~

-> Find the same matching and attend to the next token of previous use case

5. Two-Layer Transformers – 4) How Induction heads work



5. Two-Layer Transformers – 4) How Induction heads work

Induction head

- As the depth increases, composition of attention heads appears
- Residual stream conveys sum of all the outputs of attention heads and the direct path
- Consequently, W_Q, W_K, W_V of 2nd layer reads in a subspace of this composed residual stream

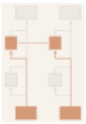
$$C_{QK}^{h\in H_2} = \left(\mathrm{Id}\otimes\mathrm{Id}\otimes W_E^T + \sum_{h_q\in H_1} A^{h_q}\otimes\mathrm{Id}\otimes(W_{OV}^{h_q}W_E)^T\right) \cdot \mathrm{Id}\otimes\mathrm{Id}\otimes W_{QK}^h \cdot \left(\mathrm{Id}\otimes\mathrm{Id}\otimes W_E + \sum_{h_k\in H_1}\mathrm{Id}\otimes A^{h_k}\otimes W_{OV}^{h_k}W_E\right)$$

The **"query side" residual stream** at the start of the second layer contains both the layer 1 direct path and layer 1 attention heads. All terms are of the form $\ldots\otimes\mathrm{Id}\otimes\ldots$ because they don't move key information.

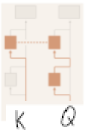
W_{QK} of the second layer head combines both sides into attention scores.

The **"key side" residual stream** at the start of the second layer contains both the layer 1 direct path and attention heads. All terms are of the form $\mathrm{Id}\otimes\ldots$ because they don't move query information.

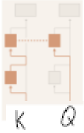
$$= \mathrm{Id}\otimes\mathrm{Id}\otimes(W_E^T W_{QK}^h W_E) + \sum_{h_q\in H_1} A^{h_q}\otimes\mathrm{Id}\otimes\left(W_E^T W_{OV}^{h_q\,T} W_{QK}^h W_E\right) + \sum_{h_k\in H_1}\mathrm{Id}\otimes A^{h_k}\otimes\left(W_E^T W_{QK}^h W_{OV}^{h_k} W_E\right) + \sum_{h_q\in H_1}\sum_{h_k\in H_1} A^{h_q}\otimes A^{h_k}\otimes\left(W_E^T W_{OV}^{h_q\,T} W_{QK}^h W_{OV}^{h_k} W_E\right)$$



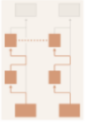
The no composition term. Both first layer follows the direct path on both the key and query side.



These terms correspond to pure **Q-composition**. A previous attention head is used to generate the query side, but the key side is the first layer direct path.



These terms correspond to pure **K-composition**. A previous attention head is used to generate part of the key, but the query side is the first layer direct path.



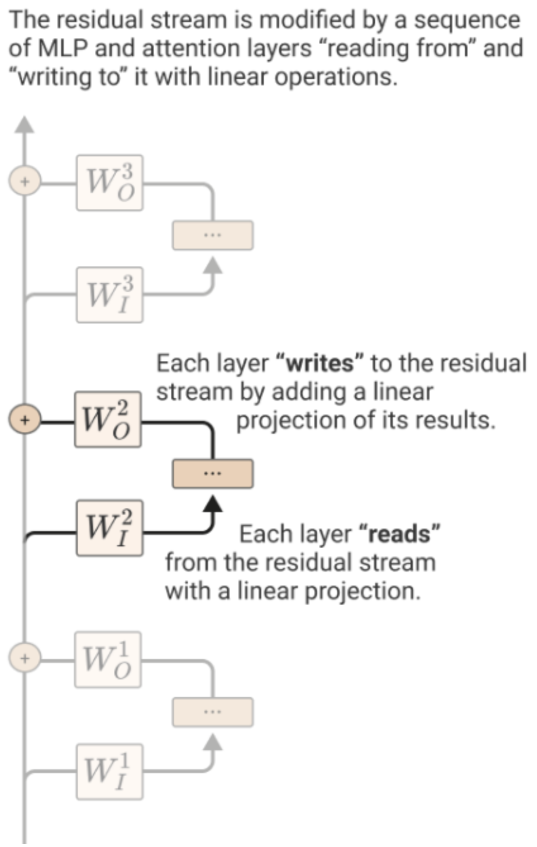
These terms are interactions between both **Q-composition** and **K-composition**. A previous attention head is used to generate the query and key sides.

5. Two-Layer Transformers – 5) Compositions

Composition

- Q-Composition
  - W_Q projects (reads in) a subspace affected by a previous head
    - Use the context to figure out what the right source token is
- K-Composition
  - W_K projects (reads in) a subspace affected by a previous head
    - Use the context and intelligence to figure out where to get the information from
- V-composition
  - W_V projects (reads in) a subspace affected by a previous head
    - Figure out the information that is more meaningful than just the token at that position

5. Two-Layer Transformers – Revisit Read and Write



Keep in mind that all outputs are summed in the residual stream!

Write (Embed)

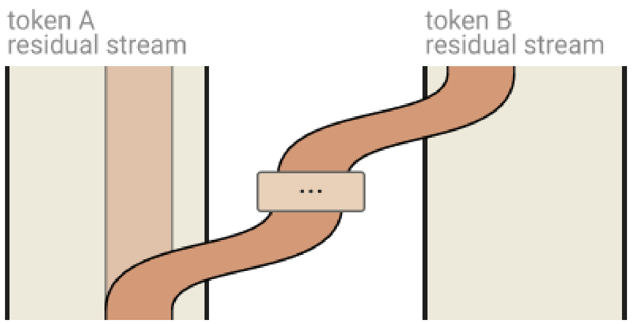
- Hidden dimension (small) -> residual dimension (big)
- The model chooses a set of directions in residual stream and write the information to those

Read (Project)

- Residual dimension (big) -> hidden dimension (small)
- The model focuses on meaningful directions
- By aligning directions with W_I , the model only reads in the information it really cares about in the sea of information

5. Two-Layer Transformers – Revisit Read and Write

$W_O W_V$ governs which information is read from the source token and how it is written to the destination



Attention heads copy information from the residual stream of one token to the residual stream of another. They typically write to a different subspace than they read from.

$$h(x) = \underbrace{(\mathrm{Id} \otimes W_O)}_{\text{Project result vectors out for each token } (h(x)_i = W_O r_i)} \cdot \underbrace{(A \otimes \mathrm{Id})}_{\text{Mix value vectors } across \text{ tokens to compute result vectors } (r_i = \sum_j A_{i,j} v_j)} \cdot \underbrace{(\mathrm{Id} \otimes W_V)}_{\text{Compute value vector for each token } (v_i = W_V x_i)} \cdot x$$

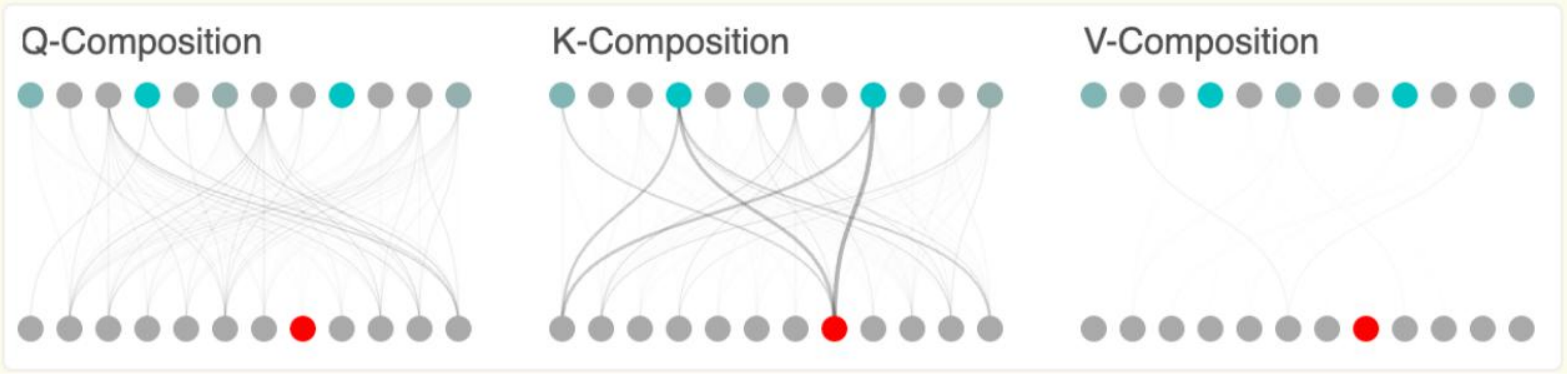
$$h(x) = \underbrace{(A \otimes W_O W_V)}_{A \text{ mixes across tokens while } W_O W_V \text{ acts on each vector independently.}} \cdot x$$

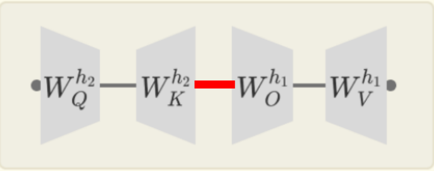
5. Two-Layer Transformers – 5) Compositions

Composition

- Q-Composition
  - W_Q projects (reads in) a subspace affected by a previous head
    - Use the context to figure out what the right source token is
- K-Composition
  - W_K projects (reads in) a subspace affected by a previous head
    - Use the context and intelligence to figure out where to get the information from
- V-composition
  - W_V projects (reads in) a subspace affected by a previous head
    - Figure out the information that is more meaningful than just the token at that position

5. Two-Layer Transformers – 5) Compositions





$||W_Q^{h_2 T} W_K^{h_2} W_O^{h_1} W_V^{h_1}||_F$

This is the correct measure of composition. The most important term is the product of $W_K^{h_2}$ and $W_O^{h_1}$ ( ), which detects that the key of h_2 is built from the output of h_1.

Q-composition - $\dfrac{\left\|W_{QK}^{h_2\,T} W_{OV}^{h_1}\right\|_F}{\left\|W_{QK}^{h_2\,T}\right\|_F \left\|W_{OV}^{h_1}\right\|_F}$ K-composition - $\dfrac{\left\|W_{QK}^{h_2} W_{OV}^{h_1}\right\|_F}{\left\|W_{QK}^{h_2}\right\|_F \left\|W_{OV}^{h_1}\right\|_F}$ V-composition - $\dfrac{\left\|W_{OV}^{h_2} W_{OV}^{h_1}\right\|_F}{\left\|W_{OV}^{h_2}\right\|_F \left\|W_{OV}^{h_1}\right\|_F}$

5. Two-Layer Transformers – 5) Composition

$W_O W_V$ governs which information is read from the source token and how it is written to the destination

-> W_{OV} governs the subspace of the residual stream which the attention head reads in and write to

Let's decompose W_{OV} with SVD

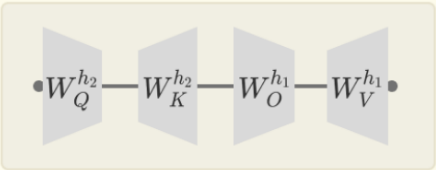
$$W_{OV} = U\Sigma V$$

Σ: only a subset of diagonal elements are non-zero

V: which subspace of the residual stream being attended to
project (align) information it really cares about

U: which subspace of the destination residual stream
embed (align) a chosen set of directions in residual stream

5. Two-Layer Transformers –5) Compositions



$||W_Q^{h_2 T} W_K^{h_2} W_O^{h_1} W_V^{h_1}||_F$

This is the correct measure of composition. The most important term is the product of $W_K^{h_2}$ and $W_O^{h_1}$ ( ), which detects that the key of h_2 is built from the output of h_1.

$W_{QK}, W_{OV} \in \mathbb{R}^{d_m \times d_m}$

$U, S, V \in \mathbb{R}^{d_m \times d_m}$

$W_{QK} = U_{QK} S_{QK} V_{QK}$

$W_{OV} = U_{OV} S_{OV} V_{OV}$

K-composition - $\dfrac{\left\|W_{QK}^{h_2} W_{OV}^{h_1}\right\|_F}{\left\|W_{QK}^{h_2}\right\|_F \left\|W_{OV}^{h_1}\right\|_F}$

Q-composition - $\dfrac{\left\|W_{QK}^{h_2\,T} W_{OV}^{h_1}\right\|_F}{\left\|W_{QK}^{h_2\,T}\right\|_F \left\|W_{OV}^{h_1}\right\|_F}$

$$\begin{aligned}
&\| W_{QK} W_{OV} \|_F \\
&= \| W_Q^T W_K W_O W_V \|_F \\
&= \| U_{QK} S_{QK} V_{QK} U_{OV} S_{OV} V_{OV} \|_F \\
&= \| S_{QK} V_{QK} U_{OV} S_{OV} \|_F
\end{aligned}$$

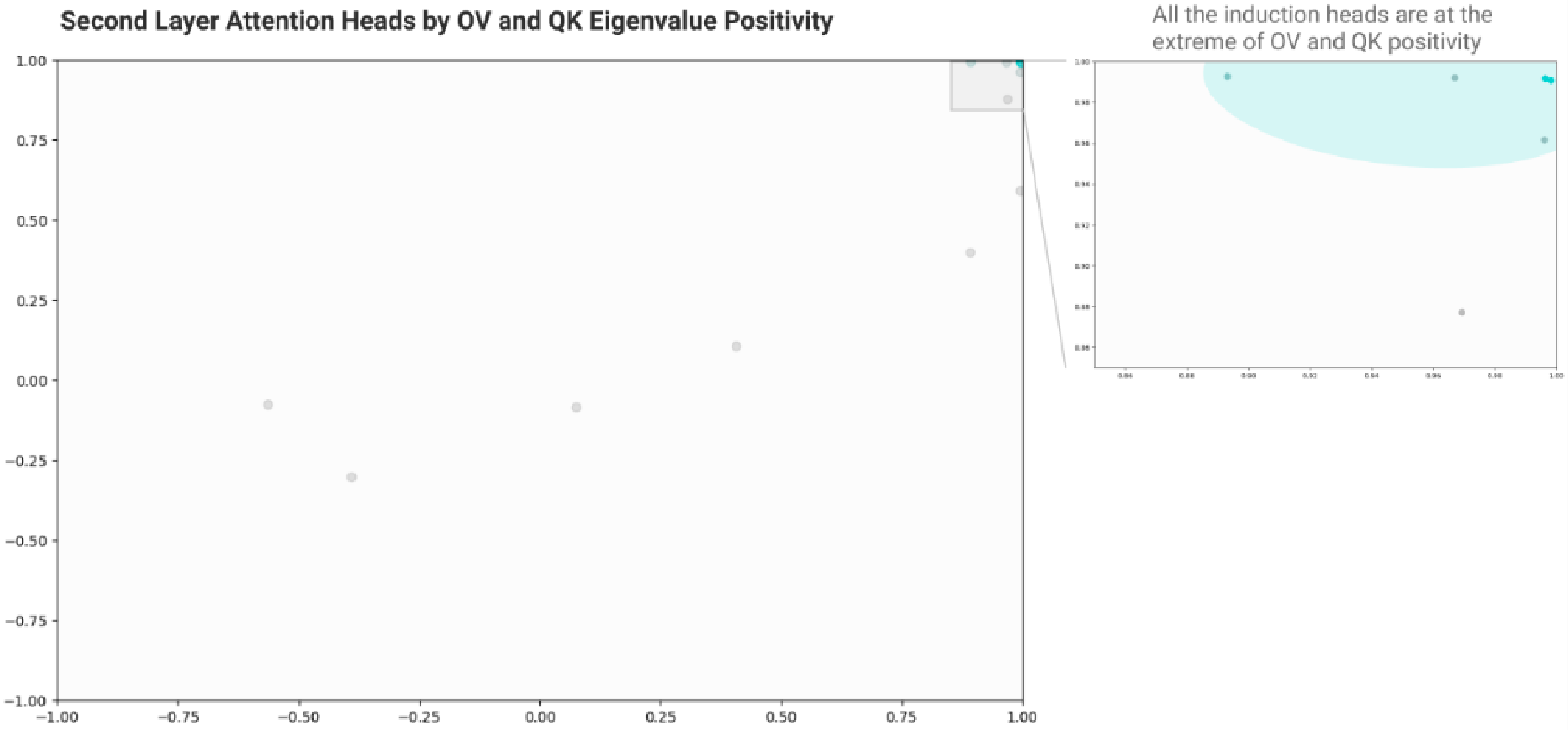
$$\begin{aligned}
&\| W_{QK}^T W_{OV} \|_F \\
&= \| W_K^T W_Q W_O W_V \|_F \\
&= \| V_{QK}^T S_{QK}^T U_{QK}^T U_{OV} S_{OV} V_{OV} \|_F \\
&= \| S_{QK}^T U_{QK}^T U_{OV} S_{OV} \|_F
\end{aligned}$$

5. Two-Layer Transformers – 4) How Induction heads work

Eigenvalue positivity of W in the QK term

$$\mathrm{Id} \otimes A^{h_{-1}} \otimes W$$

$$\left(\frac{\sum \lambda_i}{\sum |\lambda_i|} \text{ of } W_E^T W_{QK}^h W_{OV}^{h_{-1}} W_E \right)$$



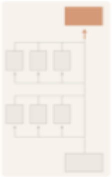
Eigenvalue positivity of the OV circuit

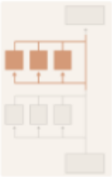
$$\left(\frac{\sum \lambda_i}{\sum |\lambda_i|} \text{ of } W_U W_{OV}^h W_E \right)$$

5. Two-Layer Transformers – Term Importance Analysis

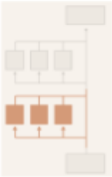
Path Expansion of Logits

$$T = \mathrm{Id} \otimes W_U \cdot \left(Id + \sum_{h \in H_2} A^h \otimes W_{OV}^h \right) \cdot \left(Id + \sum_{h \in H_1} A^h \otimes W_{OV}^h \right) \cdot \mathrm{Id} \otimes W_E$$

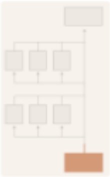




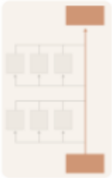
The second **attention layer** has multiple attention heads which add into the residual stream



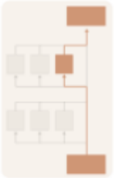
The first **attention layer** has multiple attention heads which add into the residual stream



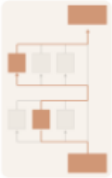
$$= \mathrm{Id} \otimes W_U W_E + \sum_{h \in H_1 \cup H_2} A^h \otimes (W_U W_{OV}^h W_E) + \sum_{h_2 \in H_2} \sum_{h_1 \in H_1} (A^{h_2} A^{h_1}) \otimes (W_U W_{OV}^{h_2} W_{OV}^{h_1} W_E)$$



"Direct path" term contributes to bigram statistics.

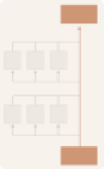
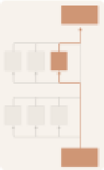
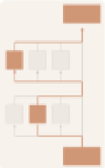


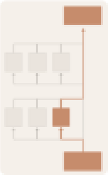
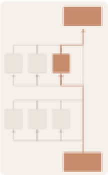
The **individual attention head** terms describe the effects of individual attention heads in linking input tokens to logits, similar to those we saw in the one layer model.



The **virtual attention head** terms correspond to V-composition of attention heads. They function a lot like individual attention heads, with their own attention patterns (the compositon of the heads patterns) and own OV matrix.

5. Two-Layer Transformers – Term Importance Analysis

Type	Example	Equation	Marginal Loss Reduction
direct path order 0		$W_U W_E$	**- 1.8 nats** relative to uniform predictions -1.8 nats/term (- 1.8 nats / 1 term)
individual attention head order 1		$A^h \otimes (W_U W_{OV}^h W_E)$	**- 5.2 nats** relative to only using direct path -0.2 nats/term (5.2 nats / 24 terms)
virtual attention head order 2		$(A^{h_2} A^{h_1}) \otimes (W_U W_{OV}^{h_2} W_{OV}^{h_1} W_E)$	**- 0.3 nats** relative to only using above -0.002 nats/term (0.3 nats / 144 terms)

Type	Example	Marginal Loss Reduction	Notes
Layer 1 Attention Heads		**- 0.05 nats** -0.004 nats/head relative to direct path + layer 2 **- 1.3 nats** -0.1 nats/head relative to direct path only	Relatively small effect, but keep in mind these heads also contribute to layer 2 QK circuits.
Layer 2 Attention Heads		**- 4.0 nats** -0.3 nats/head relative to direct path + layer 1 **- 5.2 nats** -0.4 nats/head relative to direct path only	**We'll focus on these.** Much larger effect. These heads are a lot more sophisticated than the layer 1 heads, since they can use layer 1 heads in their QK circuits.

5. Two-Layer Transformers – Virtual Attention Heads

What Are They?

- **V-Composition**
  - Formed by multiplying OV matrices of two attention
  - Creates a “virtual head” that behaves like with its own attention pattern

Why Are They Interesting?

- **Powerful Compositions**
  - Let models combine behaviors (e.g., attend to prior tokens, then further refine or shift focus).
- **Scalability**
  - Grow exponentially by composition
- **Small Functional Units**
  - Handle niche tasks without allocating a full, “large” head.

Key Insight

- May be crucial in deeper models, where they offer more **flexible and granular** attention patterns.

Thank you!