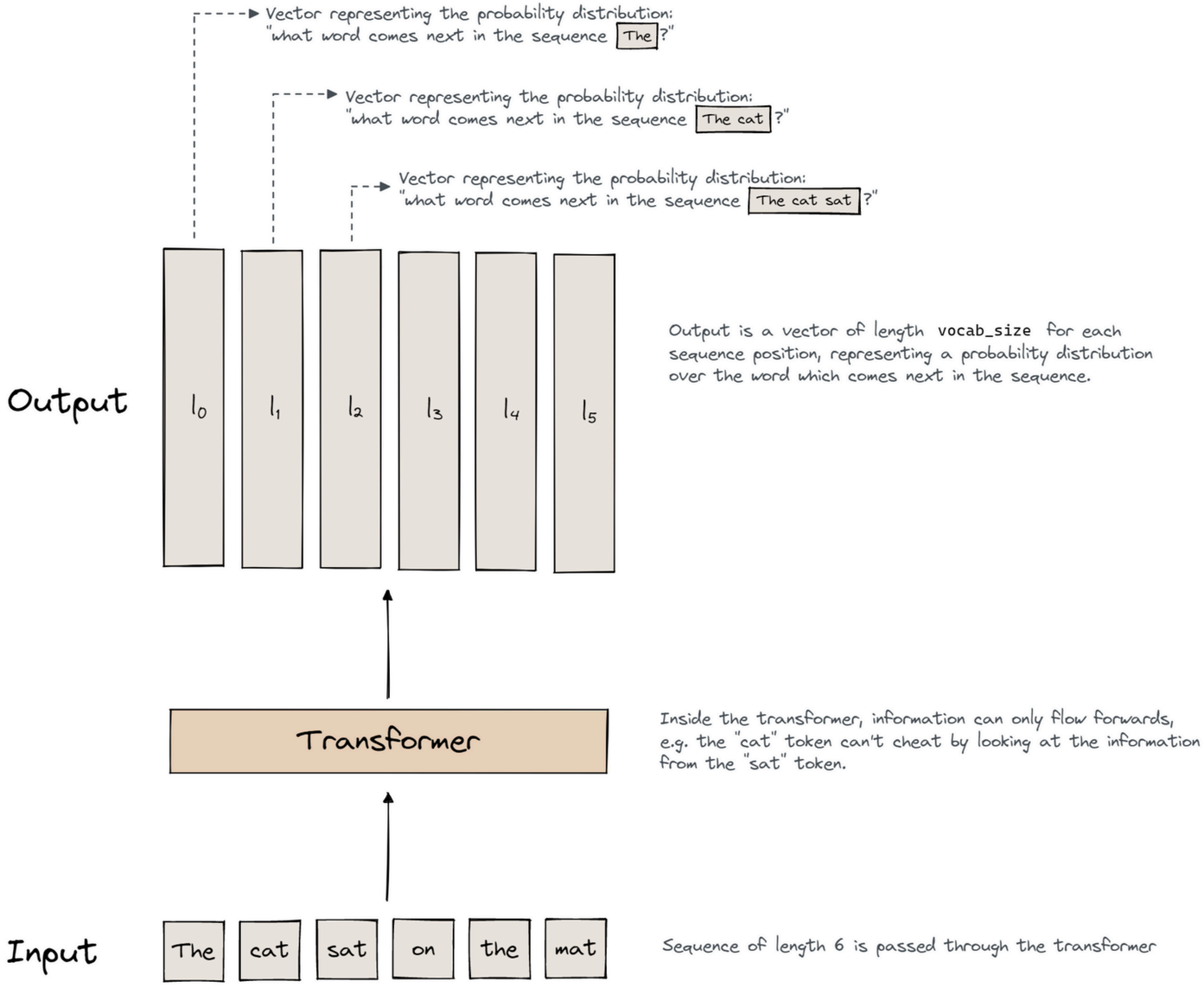
Mechanistic Interpretability

Part II: Circuits and Superposition

Ryan Chan

Transformers recap

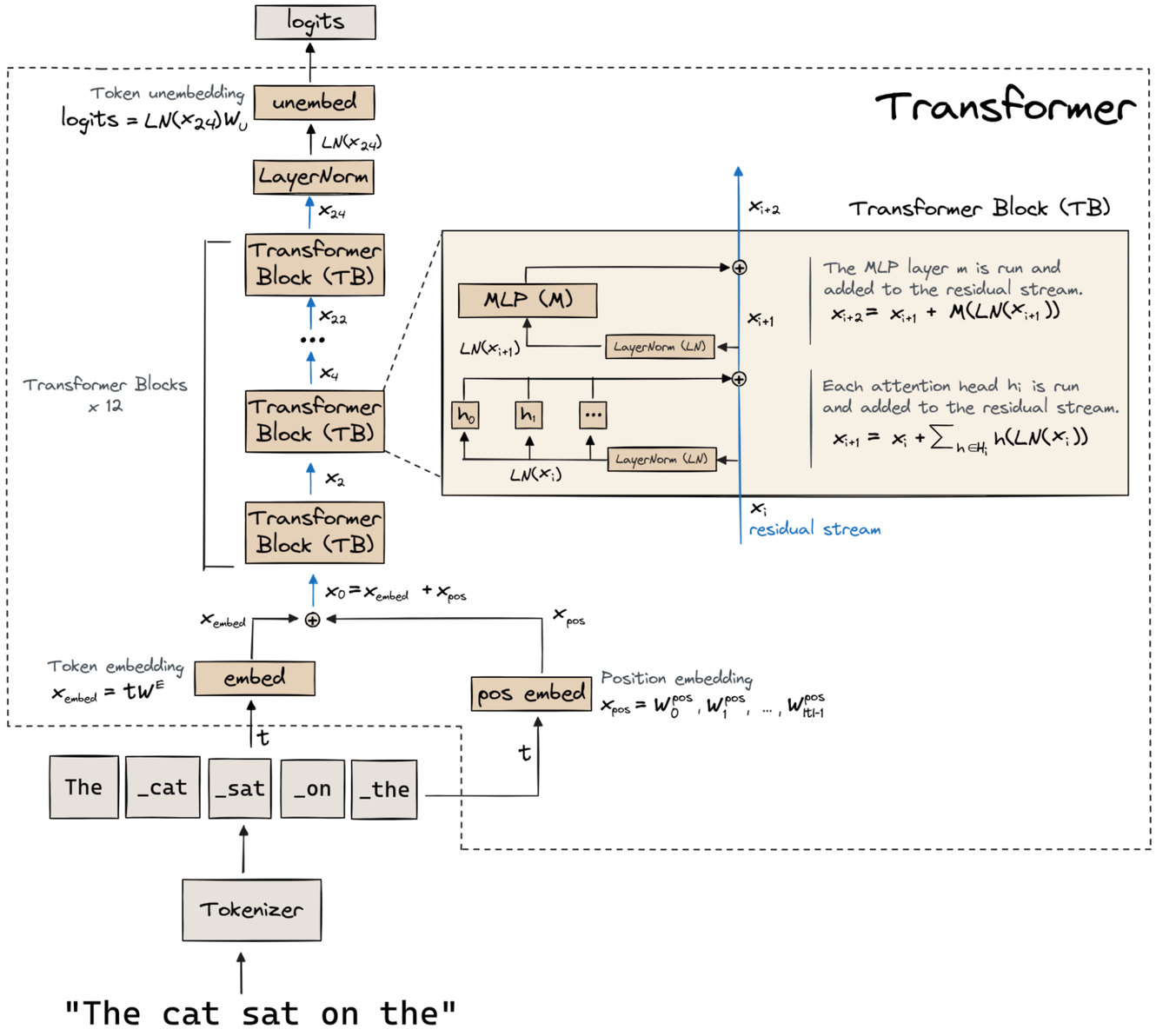
- In a transformer decoder, we give the model some piece of text, and train it to **predict the next token**
  - The transformer decoder is **autoregressive** - only predicts future words based on past data
- We input natural language (sequence of characters, strings, etc.) and convert that into smaller sub-units (*tokens*)
- In the output, we obtain a vector of probability distribution for the next token at each position



* Diagram's taken from Callum McDougall's ARENA Transformer Interpretability course

Transformers recap

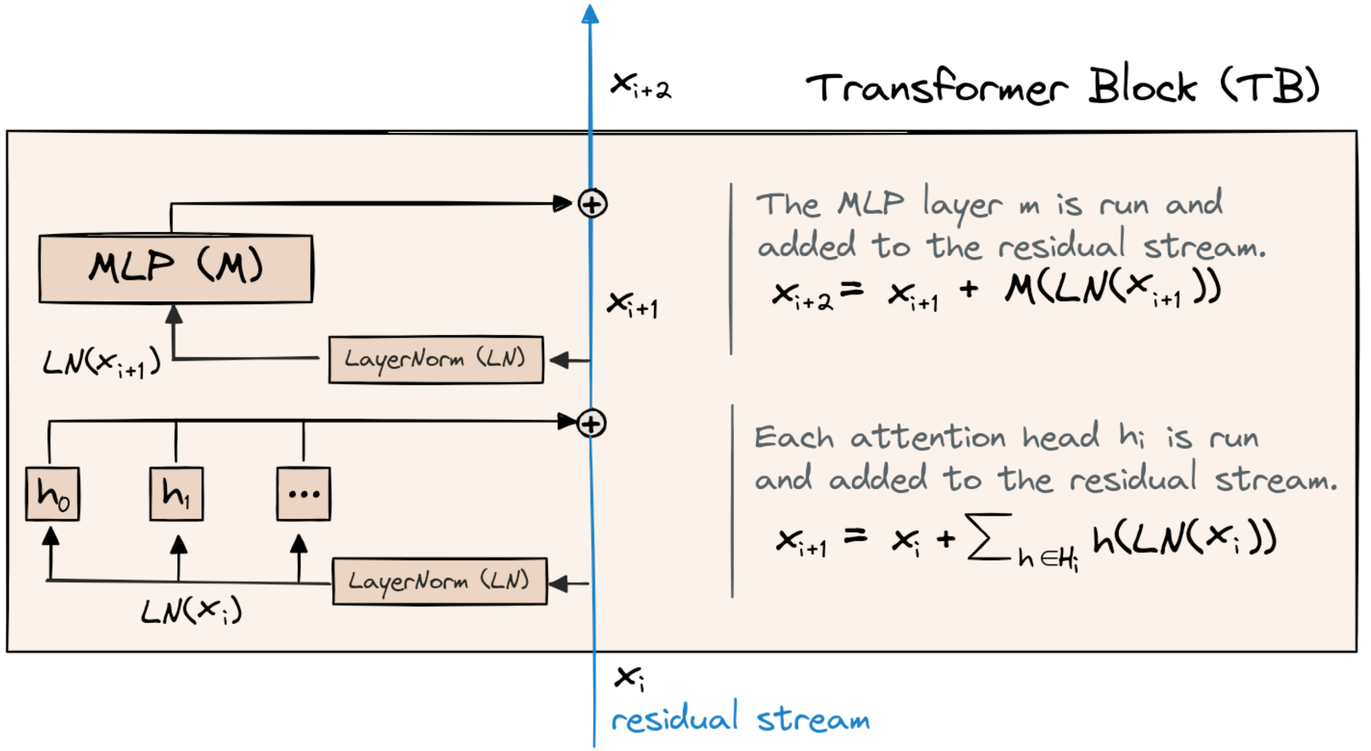
- After tokenization, we obtain embeddings for each of our tokens
  - Often position information is encoded by adding a position embedding
- Next, a series of Transformer blocks consisting of a combination of **multihead-attention** and **multi-layer perceptron (MLP) layers** are applied
  - Each of these computations are performed and added to the **residual stream**
  - There are also **layer norm** computations
- Lastly, we **unembed** and apply a linear map W_U from the final token representations to a vector of logits
  - Used to predict the next token in the sequence



* Diagram's taken from Callum McDougall's ARENA Transformer Interpretability course

Transformers recap

- We have a series of **transformer blocks**
  - **Attention** is applied first and moves information between the prior positions in the sequence and the current token
    - They are made up of n_heads heads each with their own attention pattern acting independently and additively
  - The **MLP** layers are standard neural networks with a single hidden layer and non-linear activation
    - The hidden dimension is typically $\text{d\_mlp} = 4 \cdot \text{d\_model}$
    - It is applied across positions independently
- Each of these add to the **residual stream** - the sum of all previous outputs of layers of the model
  - This has shape [batch, seq_len, d_model]
  - The initial value of the residual stream is $\mathbf{x}_0$ and $\mathbf{x}_i$ are later values after several attention and MLP layers have been applied and added



* Diagram's taken from Callum McDougall's ARENA Transformer Interpretability course

Attention

- Key role is to determine **where to move information** to and from and **what information to move**

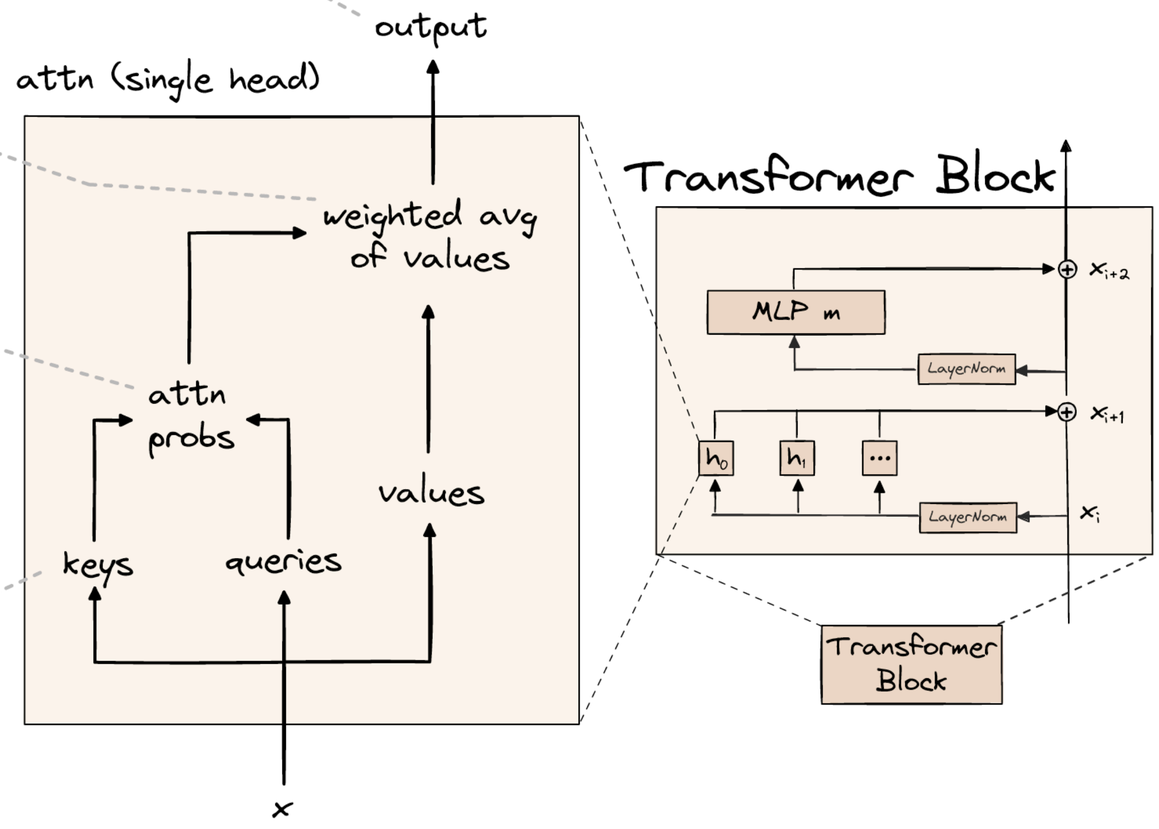
We apply a final linear transformation to our value vectors, mapping them up to the right size to be added to the residual stream.

For each destination position, we take a weighted average of value vectors from each source position, in accordance with how much attention destination pays to source.

Attention probabilities tell us how much each query position (destination) pays attention to each key position (source), i.e. this tells us where information moves to and from.

This is the point where we make sure information can't flow backwards; we set the attn probs to be zero if query comes before key.

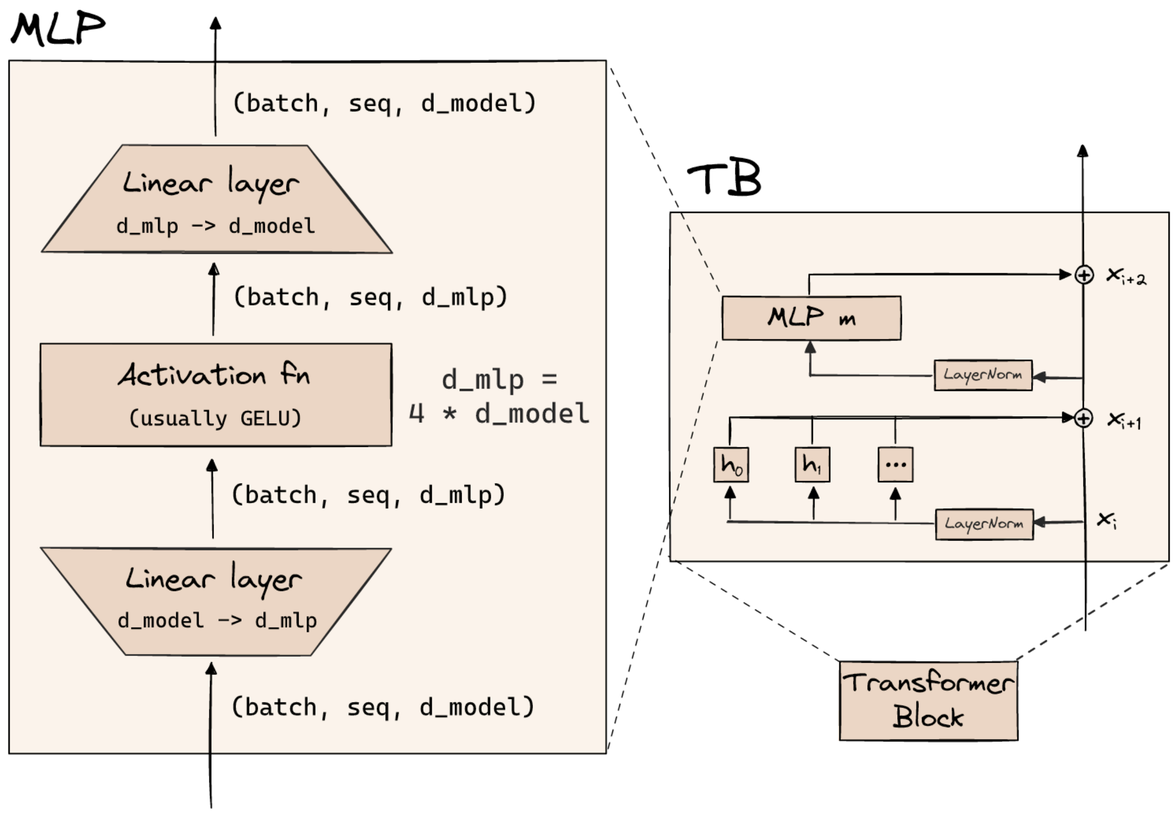
3 separate linear transformations give us keys, queries and values for each sequence position.



* Diagram's taken from Callum McDougall's ARENA Transformer Interpretability course

MLP

- Commonly thought as where knowledge gets stored in the transformer*
    - Attention mechanism is what moves information around between sequence positions, but the MLPs is where this information is processed



* Also watch 3Blue1Brown's "How might LLMs store facts" video for some intuition on this

** Diagram's taken from Callum McDougall's ARENA Transformer Interpretability course

What makes mechanistic interpretability on transformers difficult?

- The most fundamental challenge in reverse engineering neural networks is the **curse of dimensionality**
    - As models get larger, neural network activations live in a very high dimensional space and weights lie in an even higher dimensional space
- For mechanistic interpretability, the ultimately reduces to whether we can **decompose** the activation space into **independently understandable components** (i.e. features) and decompose the weights into **circuits** connecting the features
    - No current way to understand, search or enumerate such a space unless it can be decomposed into lower dimensional pieces that can be understood independently

What makes mechanistic interpretability on transformers difficult?

- In some cases, we can side step curse of dimensionality issues by rewriting neural networks in certain ways
  - e.g. in A Mathematical Framework for Transformer Circuits
- In other cases, we may be able to describe activations in terms of the model neurons
  - But often, neurons are **polysemantic**. We would need to solve this issue in order to decompose the activations into **monosemantic & interpretable features**
    - e.g. Toy Models of Superposition, Towards Monosemanticity…, Scaling Monosemanticity… papers from Anthropic are works towards tackling this issue

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

A Mathematical Framework for Transformer Circuits

- A **circuit** is a subset of a model's weights and non-linearities that map a set of earlier features to a set of later features
    - **Circuit analysis** is the process of identifying a subgraph of the model's computational graph that is human-understandable and responsible for completing a specific task (Wang et al. 2022*)
- Given the complexity and size of modern language models, in **A Mathematical Framework for Transformer Circuits**, Elhage et al. 2021 focus on analysing transformers with **two layers or less** which **only have attention blocks** (i.e. an **attention-only transformer**)
    - In contrast, Llama 3.1 405B has 126 layers with alternating attention and MLP blocks
- Purpose of the paper is to take a smaller model, try to understand them and try to carefully reason how gained insights could transfer to larger models

* Interpretability In The Wild: A Circuit for Indirect Object Identification in GPT-2 Small

Results summary

- Aimed to reverse engineer toy attention-only models and found:
  - Zero layer transformers (model with just embed and unembedding layers) model **bigram statistics**
    - *Bigrams* refer to predicting the next token based on what most frequently follows the current token, e.g. “Barack” → “Obama”
    - The bigram table can be accessed directly from the weights
  - One-layer attention-only transformers are an **ensemble of bigram and “skip-trigram” models**
    - *Skip-trigrams* are sequences/patterns of the form: A... B → C, e.g. “keep… in” → “mind”
    - The bigram and skip-trigram tables can be accessed directly from the weights
  - Two-layer attention-only transformers implement much more complex algorithms by **composing** outputs of attention heads
  - One- and two-layer attention transformers perform a very simple version of in-context learning

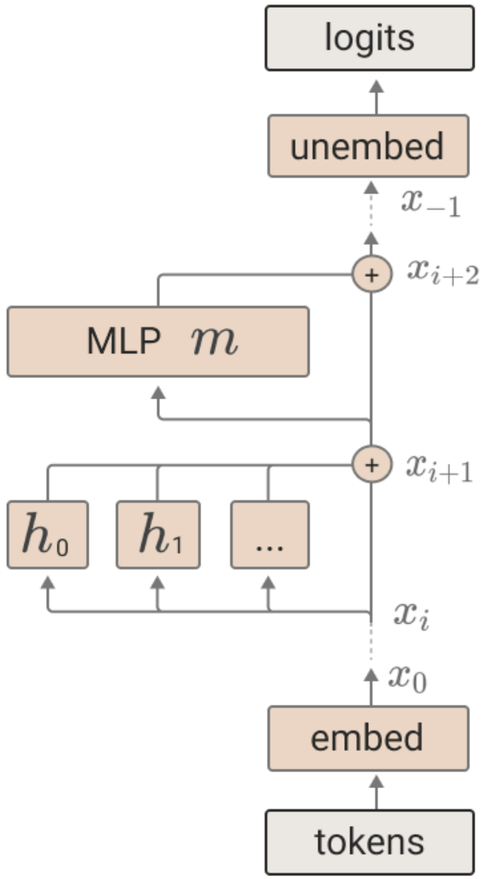
Methodology

- Start off by reframing the transformer into equivalent but non-standard ways which make it easier to reason about the model
- In modern deep learning, there's an emphasis on computational efficiency
  - The mathematical descriptions of models often mirror decisions in how one would implement the model efficiently
- There are many equivalent ways to represent the same computation
  - It's likely the most human interpretable representation and the most computationally efficient representation will be different

Why attention-only transformers?

- Attention presents new challenges not faced in previous mechanistic interpretability work
- MLP layers are very hard to understand - as of time of writing, hadn’t found much success in understanding them
- Argue that there are many circuits in transformers mediated primarily by attention heads which is the main focus of this study
    - Ultimately, we can only learn so much by focusing on attention as MLP layers make up 2/3rds of a standard transformer’s parameters

High level architecture



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.

$$x_0 = W_E t$$

Virtual weights and the residual stream

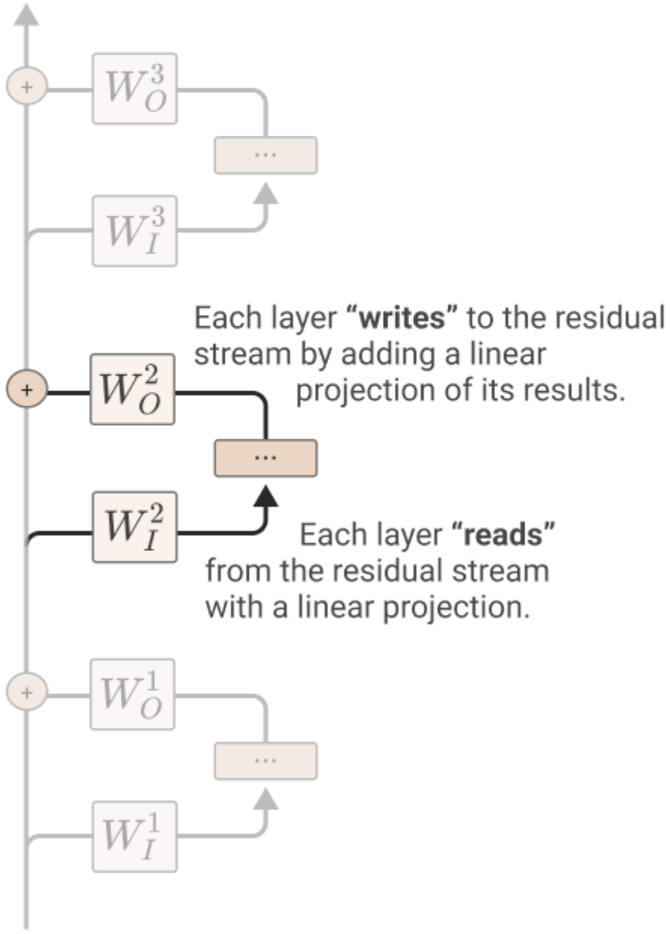
- Each layer in a transformer performs some computation and adds its results into the **residual stream**
  - This is the sum of the output of all previous layers and the original embedding
  - It serves as a **communication channel** - it does no processing itself and all the layers communicate through it
- The residual stream has a linear structure
  - Each layer first “reads” information from it using a linear transformation
  - They apply a linear transformation before adding back to “write” its output back into the stream

Virtual weights and the residual stream

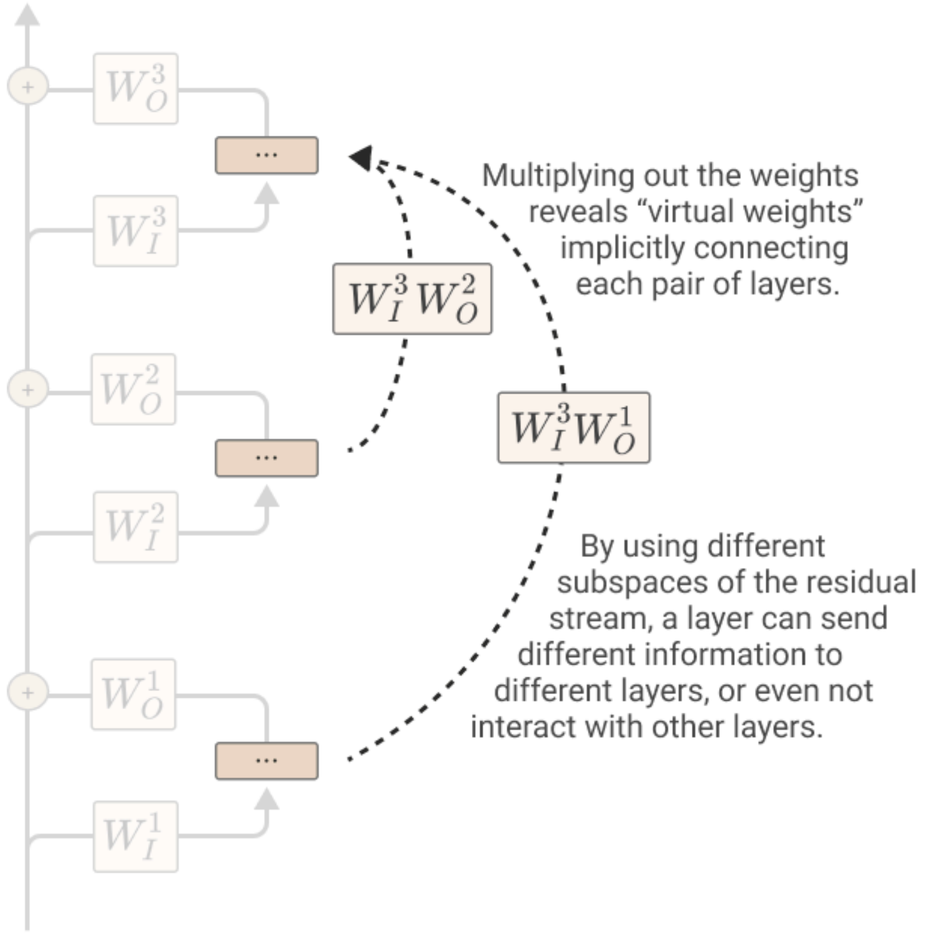
- We can think of implicit “virtual weights” which directly connect pairs of layers - even those separated by other layers by multiplying out their interactions through the residual stream
- Virtual weights are the product of output weights of one layer with input layers of another
  - They describe the extent to which a later layer reads information written by a previous layer

Virtual weights and the residual stream

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.

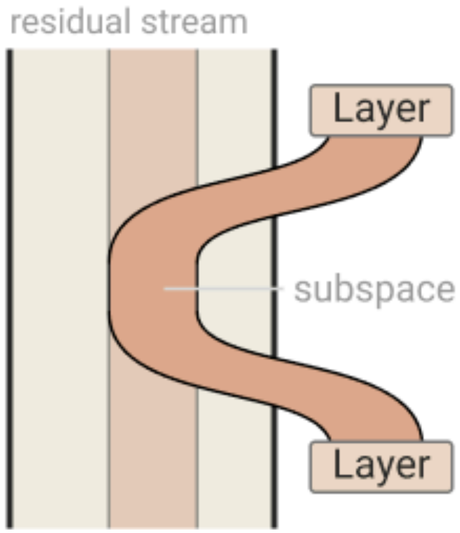


Residual stream memory management

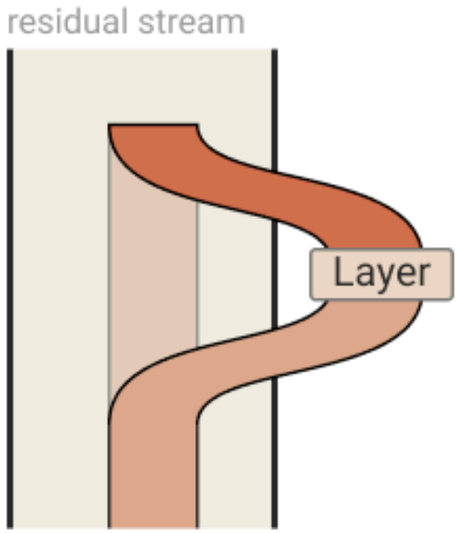
- The residual stream is a high dimensional space (e.g. in Llama 3.1 405B, model dimension is 16384)
- Layers could send different information to different layers by storing them in different subspaces
  - e.g. attention heads only operate on comparatively small subspace (often 64 or 128 - in Llama 3.1 405B, head size is 128)
- Once information is added, it remains in the residual stream unless another layer actively deletes it
- From this perspective, dimensions in the residual stream become something like “memory” or “bandwidth” that the model adds and writes to
- Hints towards MLP and attention heads perform some sort of “memory management”

Residual stream memory management

The residual stream is high dimensional, and can be divided into different subspaces.



Layers can interact by writing to and reading from the same or overlapping subspaces. If they write to and read from disjoint subspaces, they won't interact. Typically the spaces only partially overlap.



Layers can delete information from the residual stream by reading in a subspace and then writing the negative verison.

Some nice insights

- A common approach in this paper is to reframe computation in the transformer in more interpretable ways
- Two nice results:

  1. Attention heads are independent and additive

  2. Attention heads act to move information

Attention heads are independent

- Can think of attention layers as several completely independent attention heads $h \in H$ which operate completely in parallel and each add their output to the residual steam
- This isn't immediately obvious from Vaswani et al. 2017* where its described as concatenating the heads and multiplying by an output matrix
  - But it's equivalent to splitting the output matrix into equal size blocks for each head:

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

$$= \sum_{i=1}^{|H|} W_O^{h_i} r^{h_i} \in \mathbb{R}^{d_{model}}$$

- $W_O^H \in \mathbb{R}^{d_{model} \times |H| \cdot d_{head}}$
- $W_O^{h_i} \in \mathbb{R}^{d_{model} \times d_{head}}$
- $r^{h_i} \in \mathbb{R}^{d_{head}}$

Attention heads are independent

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

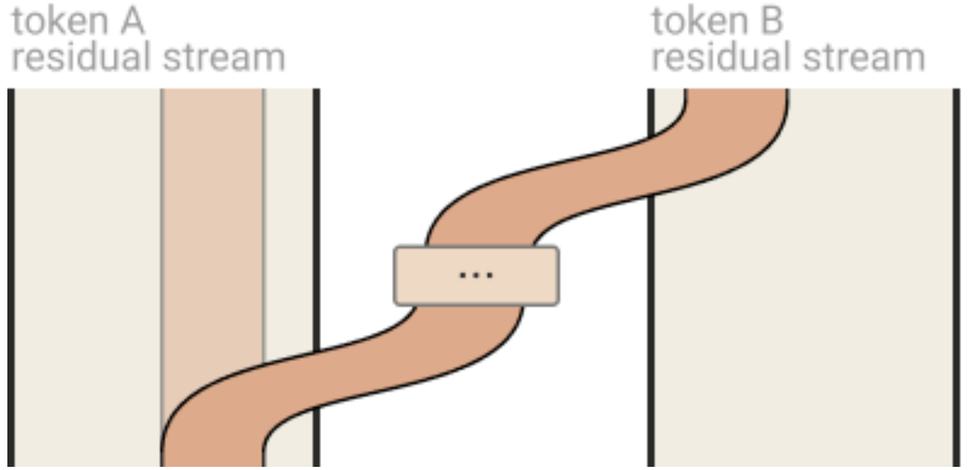
$$= \sum_{i=1}^{|H|} W_O^{h_i} r^{h_i} \in \mathbb{R}^{d_{model}}$$

- $W_O^H \in \mathbb{R}^{d_{model} \times |H| \cdot d_{head}}$
- $W_O^{h_i} \in \mathbb{R}^{d_{model} \times d_{head}}$
- $r^{h_i} \in \mathbb{R}^{d_{head}}$

- Given the first part of attention (computing output value vectors r^{h_i}) are in parallel, we can consider the whole attention mechanism as several completely independent processes with its own output matrix too
- We typically prefer the concatenated definition as it produces a larger and more compute efficient matrix multiply
  - But for interpretability, this representation helps

Attention heads as information movement

- The fundamental action of attention heads it to move information between token positions
  - They read information from the residual stream of one token and write it to the residual stream of another
  - Which tokens to move information from is completely separable to what information is “read” to be moved and how it is “written” to the destination
- Again, to see this, it’s useful to re-write attention



Attention heads as information movement

- Computing the output of an attention head is done in three steps:

1. For each token position $i \in \{1, \ldots, n\}$, compute the value vector in the residual stream: $\mathbf{v}_i = W_V \mathbf{x}_i$
2. Compute the "result vector" by linearly combing value vectors according to the attention pattern: $\mathbf{r}_i = \sum_j A_{ij} \mathbf{v}_j$
3. Compute the output vector of the head for each token: $\mathbf{h}(\mathbf{x})_i = W_0 \mathbf{r}_i$
    - Usually this is written as one matrix multiply, but we've seen this is equivalent to each head having it's own output matrix

Attention heads as information movement

- Noting that $\mathbf{x}$ is a 2D matrix (consisting of a sequence of vectors), we can re-write this using tensor products
- If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$, then $A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$
  - They have the property $(A \otimes B) \cdot (C \otimes D) = (AC) \otimes (BD)$

Attention heads as information movement

- Using tensor products, we rewrite the process of attention as:

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

- Using the property of tensor products, we have

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

$$A = \text{softmax}(Q^T K)$$
$$= \text{softmax}(\mathbf{x}^T W_Q^T W_K \mathbf{x})$$

- While this is mathematically equivalent, implementing attention this way is horribly inefficient

What does this mean?

- Attention heads move information from the residual stream of one token to another
- An attention head is applying two operations:
    1. A which governs which token's information is moved from and to
    2. $W_O W_V$ governs which information is read from the source token and how it's written to the destination token
- A is the only non-linear part of the equation using the softmax
- W_Q and W_K always operate together, and similarly W_O and W_V always operate together
    - They introduce $W_{OV} = W_O W_V$ and $W_{QK} = W_Q^T W_K$

$$h(x) = (A \otimes W_O W_V) \cdot x$$

A mixes across tokens while $W_O W_V$ acts on each vector independently.

$$\begin{aligned} A &= \text{softmax}(Q^T K) \\ &= \text{softmax}(\mathbf{x}^T W_Q^T W_K \mathbf{x}) \end{aligned}$$

Zero-layer transformer

- Model only has two steps:

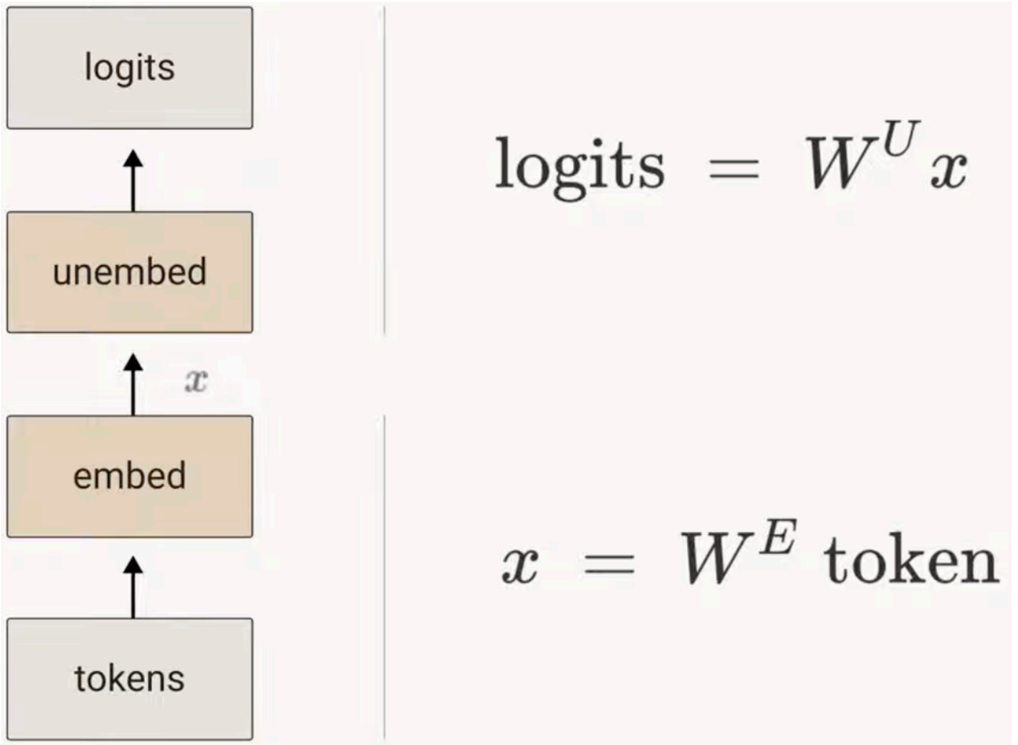
  1. Obtain embedding token

  2. Multiply by unembedding matrix to obtain logits

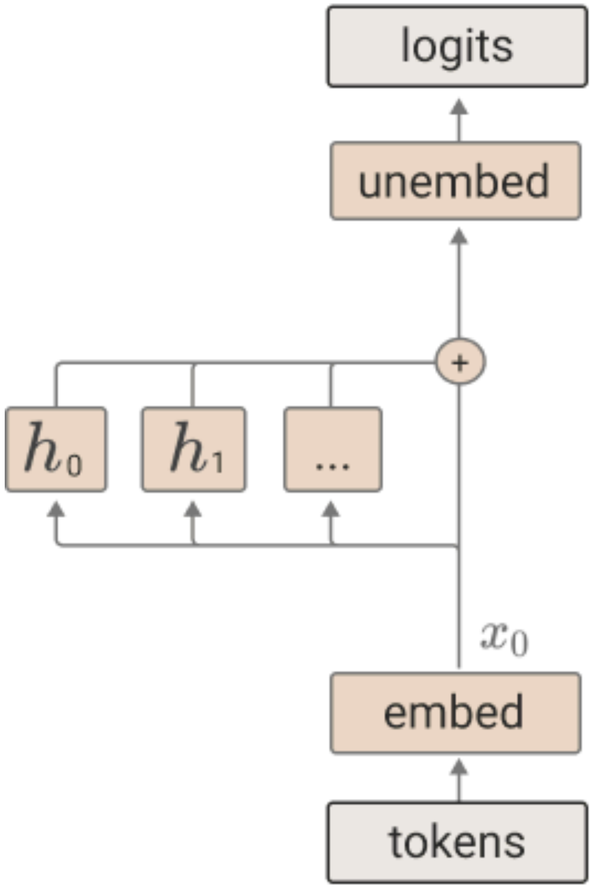
- Cannot move information between tokens, essentially we are predicting the next token from the present

  - Optimal behaviour of this model (logits $= W^U W^E$ token) is to approximate the bigram log-likelihood

- $W^U W^E$ represents the “direct path” where a token flows directly through the residual stream without going through any layers



One-layer attention-only transformer



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.

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

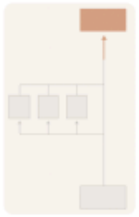
Token embedding.

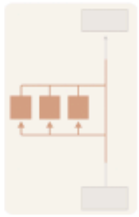
$$x_0 = W_E t$$

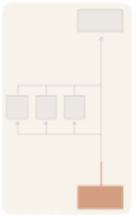
One-layer attention-only transformer

- By using tensor notation and the alternative representations of attention heads derived previously, we can represent the transformer as

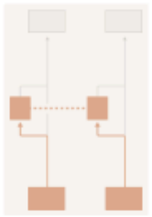
$$T = \underbrace{\mathrm{Id} \otimes W_U}_{\text{The token unembedding maps residual stream vectors to logits.}} \cdot \underbrace{\left(Id + \sum_{h \in H_1} A^h \otimes W_{OV}^h \right)}_{\text{The attention layer has multiple heads. The result of each is added into the residual stream.}} \cdot \underbrace{\mathrm{Id} \otimes W_E}_{\text{The token embedding maps tokens to residual stream vectors.}}$$







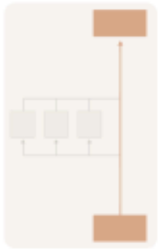
where $A^h = \underbrace{\mathrm{softmax}^*}_{\text{Softmax with autoregressive masking}} \left(\underbrace{t^T \cdot W_E^T W_{QK}^h W_E \cdot t}_{\text{Attention pattern logits are produced by multiplying pairs of tokens through different sides of } W_{QK}^h} \right)$

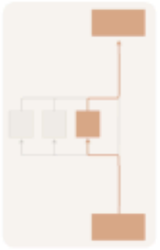


One-layer attention-only transformer

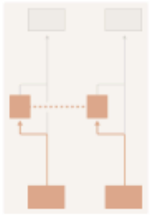
- Using the mixed product property of tensor products, we can simplify to

$$T = \underbrace{\mathrm{Id} \otimes W_U W_E}_{\text{“Direct path” term contributes to bigram statistics.}} + \underbrace{\sum_{h \in H} A^h \otimes (W_U W_{OV}^h W_E)}_{\text{The attention head terms describe the effects of attention heads in linking input tokens to logits. } A^h \text{ describes which tokens are attended to while } W_U W_{OV}^h W_E \text{ describes how each token changes the logits if attended to.}}$$





$$\text{where } A^h = \underbrace{\mathrm{softmax}^*}_{\text{Softmax with autoregressive masking}} \left(\underbrace{t^T \cdot W_E^T W_{QK}^h W_E \cdot t}_{\text{Attention pattern logits are produced by multiplying pairs of tokens through different sides of } W_{QK}^h.} \right)$$



What does this mean?

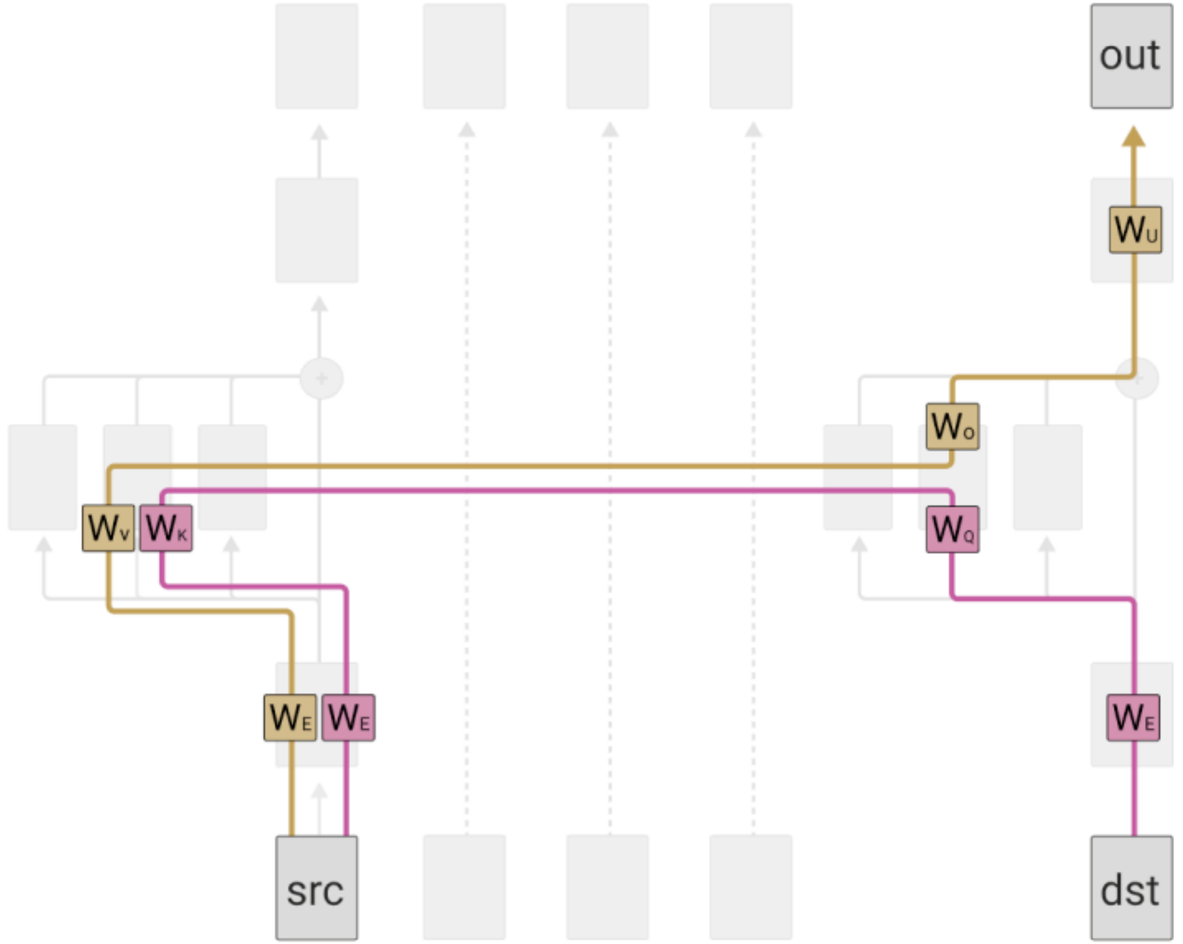
$$T = \underbrace{\mathrm{Id} \otimes W_U W_E}_{} + \sum_{h \in H} A^h \otimes (W_U W_{OV}^h W_E)$$

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

- The terms in this expression shows that there are two separable operations which are described by two $[n_{vocab}, n_{vocab}]$ matrices:
  1. $W_E^T W_{QK}^h W_E$ - this is the **query-key (QK) circuit**
     - Provides an attention score for each query and key token
     - Describes how much a given query “wants” to attend to a given key
  2. $W_U W_{OV}^h W_E$ - this is the **output-value (OV) circuit**
     - Describes how a given token will affect the output tokens if attended to

Two circuits in the one-layer transformer

- We can view this as paths through the model, starting and ending at tokens
  - Attention pattern is function of both source and destination token
- OV and QK circuits can be understood individually
  - QK circuit determines which *“source” token the present “destination” token should attend back* to and copy information from
  - OV circuit describes the resulting *effect on the “out” predictions* for the next token



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

$$W_U W_O W_V W_E$$

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

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

Consequences / interpretations

- The resulting matrices are enormous: for a vocabulary of ~50000 tokens, a single expanded OV matrix has ~2.5 billion entries
- To analyse behaviour, it's possible to read off OV and QK matrices and search for large entries - this revealed interesting behaviour in one-layer transformers
  - In particular, they seem to have functionality to model **skip-trigrams**: these are patterns of the form A... B → C (or [source] . . . [destination][out] and the [out] is modified), e.g. "keep… in" → "mind"
  - One-layer transformers seem to dedicate significant portions of its capacity to **copying**, which is a primitive form of in-context learning

Copying / in-context learning

- In this example, they fix a given source token and look at largest corresponding QK entries (destination) and largest corresponding OV entries (out token)
- Seems to do some version of copying (which is like a very simple version of in-context learning)
- Also found other interesting skip-trigrams that help to explain certain bugs
  - Representing skip-trigrams in this two-step way is like representing $f(a,b,c) = f_1(a,b)f_2(a,c)$ and they cannot capture three-way interactions flexibly which can lead to errors

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"

Primarily positional attention heads

- Found examples of attention heads which strongly prefer to attend to certain positions
  - Potentially highlights how attention heads can handle position
- In this example, the attention head seems to primarily attend to the present or previous token

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”

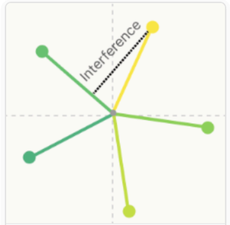
Two-layer transformers

- By using tensor notation and the alternative representations of attention heads derived previously, we can represent the transformer as

$$T = \underbrace{\mathrm{Id} \otimes W_U}_{\text{}} \cdot \underbrace{\left(Id + \sum_{h \in H_2} A^h \otimes W_{OV}^h\right)}_{\text{The second attention layer has multiple attention heads which add into the residual stream}} \cdot \underbrace{\left(Id + \sum_{h \in H_1} A^h \otimes W_{OV}^h\right)}_{\text{The first attention layer has multiple attention heads which add into the residual stream}} \cdot \underbrace{\mathrm{Id} \otimes W_E}_{\text{}}$$

$$= \underbrace{\mathrm{Id} \otimes W_U W_E}_{\text{“Direct path” term contributes to bigram statistics.}} + \underbrace{\sum_{h \in H_1 \cup H_2} A^h \otimes (W_U W_{OV}^h W_E)}_{\text{The individual attention head terms describe the effects of individual attention heads in linking input tokens to logits, similar to those we saw in the one layer model.}} + \underbrace{\sum_{h_2 \in H_2} \sum_{h_1 \in H_1} (A^{h_2} A^{h_1}) \otimes (W_U W_{OV}^{h_2} W_{OV}^{h_1} W_E)}_{\text{The virtual attention head terms correspond to V-composition of attention heads. They function a lot like individual attention heads, with their own attention patterns (the compositon of the heads patterns) and own OV matrix.}}$$

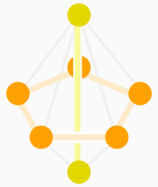
Toy Models of Superposition



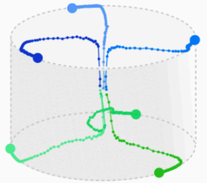
SECTION 1
Background & Motivation

SECTION 2
Demonstrating Superposition

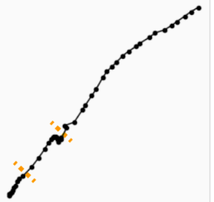
SECTION 3
Superposition as a Phase Change



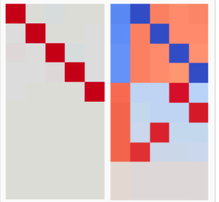
SECTION 4
The Geometry of Superposition



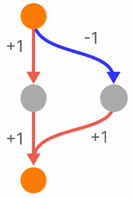
SECTION 5
Learning Dynamics



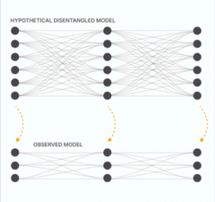
SECTION 6
Relationship to Adversarial Examples



SECTION 7
Superposition in a Privileged Basis



SECTION 8
Computation in Superposition



SECTION 9
The Strategic Picture

Discussion

Does this occur in real models?

Open Questions

SECTION 10
Discussion

Related Work

SECTION 11
Related Work

Comments & Replications

SECTION 12
Comments & Replications

AUTHORS

Nelson Elhage*, Tristan Hume*, Catherine Olsson*, Nicholas Schiefer*, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg*, Christopher Olah‡

AFFILIATIONS

Anthropic, Harvard

PUBLISHED

Sept 14, 2022

Toy Models of Superposition

- Mechanistic interpretability would be far easier if individual neurons in networks corresponded cleanly to interpretable features of the input
  - e.g. curve/colour detectors in image classifiers or a neuron firing on text related to “cycling”
- But this is often not the case that features correspond directly to neurons, especially in LLMs
- In particular, neurons are often **polysemantic**, responding and activating to mixtures of unrelated features and inputs
- Some questions:
  - Why is it that neurons sometimes align with features and sometimes don’t?
  - Why do some models and tasks have many of these clean neurons while they’re rare in others?

Superposition

- **Superposition** is the phenomenon where models represent *more features than they have dimensions*
  - When features of the input are sparse, superposition allows compression beyond what a linear model could do at the cost of “interference” that requires non-linear filtering
- “Sparsity” means that most of the features are inactive (zero or null) for any given data point while only a few are active
  - Many features seem to be sparse in the sense that they only rarely occur, e.g.
    - In vision, most patches do not contain a edge, curve, or dog
    - In language, most tokens don’t refer to Manchester United or to the Golden Gate Bridge
- In Toy Models of Superposition, they consider small ReLU models trained with synthetic data with sparse input features

Superposition as sparsity increases

- Consider a one layer ReLU network with **five input features** of varying importance with **two dimension embedding** and vary the sparsity of features
  - When features are **dense** (all features often active), model learns to only represent features only in an **orthogonal basis** of the most two important features and other features aren't represented - prioritises the two most important features
  - As **sparsity increases**, model finds ways to **represent all features using combinations** of its two-dimensional space
    - Leverages the fact that not all features need to be represented simultaneously

The Linear Representation Hypothesis

- We think of neural networks as having **features of the input** represented as **directions in activation space**

- This is the **linear representation hypothesis** which has two properties:

    1. **Decomposability**: Network representations can be described in terms of independently understandable features

        - If we hope to reverse engineer neural networks, we need a property like decomposability to tackle the curse of dimensionality

    2. **Linearity**: Features are represented by direction

        - Provides a way to access this decomposition

- If we believe the linear representation hypothesis, we hope to determine / identify which directions in the activation space correspond to independent features of the input

Features as directions

- Generally think of features as being represented by directions, e.g. in word embeddings: $\mathbf{w}(\text{"king"}) - \mathbf{w}(\text{"man"}) + \mathbf{w}(\text{"woman"}) \approx \mathbf{w}(\text{"queen"})$
- In a linear representation, each feature f_i has a corresponding representation direction W_i
  - The presence of multiple features $f_1, f_2, \ldots$ activating with values $x_{f_1}, x_{f_2}, \ldots$ is represented by $x_{f_1} W_1 + x_{f_2} W_2 + \ldots$
- A representation is **basis aligned** if all W_i are one-hot basis vectors
- Ideally, feature directions are **orthogonal** or **close-to** orthogonal

Superposition Hypothesis

- Sometimes features correspond directly to neurons and it's not clear why and when this happens, but hypothesise two driving forces

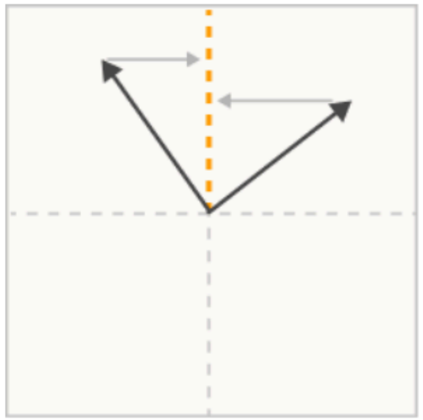
  1. Only some representations have a privileged basis which encourages features to align with basis directions (i.e. correspond to neurons)

  2. But linear representations can represent more features than dimensions using **superposition** and do this by using **almost-orthogonal** directions

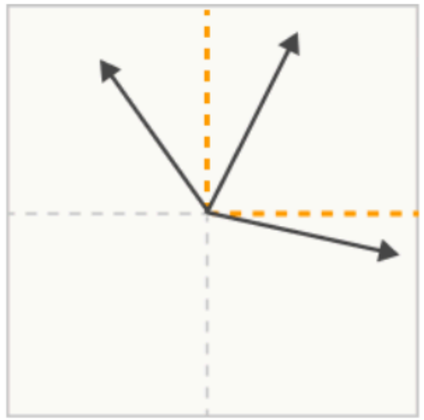
- Neurons are often polysemantic, responding to unrelated features

  - One explanation is the **superposition hypothesis** - roughly this is the idea that neural networks *want to represent more features than they have neurons*

Superposition Hypothesis



Polysemanticity is what we'd expect to observe if features were not aligned with a neuron, despite incentives to align with the privileged basis.



In the **superposition hypothesis**, features can't align with the basis because the model embeds more features than there are neurons. Polysemanticity is inevitable if this happens.

- Two mathematical results which suggests this could be possible

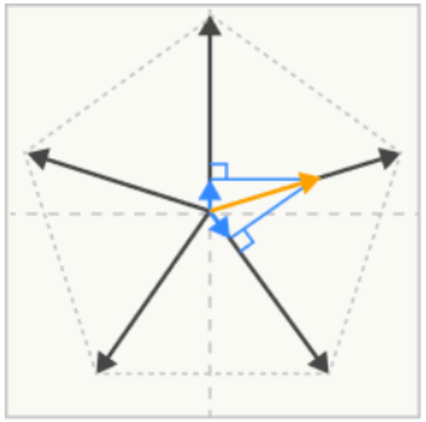
1. **Almost orthogonal vectors**: While it's not possible to have more than n orthogonal vectors in an n-dimensional space, it is possible to have $\exp(n)$ many "almost orthogonal" vectors (i.e. $< \epsilon$ dot product)*

2. **Compressed sensing**: In general, if one projects a vector into a lower-dimensional space, one cannot reconstruct the vector. But if the original vector is sparse, it is possible

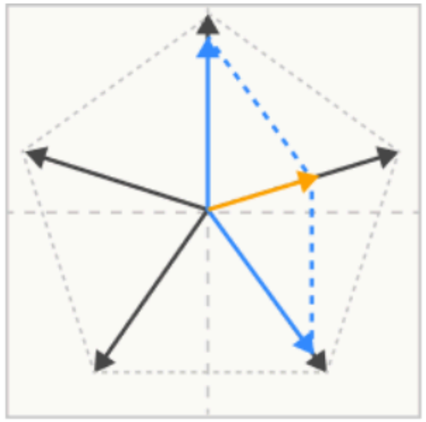
* This is the Johnson-Lidenstrauss lemma. Also see 3Blue1Brown's illustration and experiment of this

“Noisily simulating larger, sparse networks”

- Concretely, the superposition hypothesis is that features are represented as **almost-orthogonal** directions in the vector space of neuron outputs / activations
- This comes at the cost of some “noise” or “interference” - one feature activating looks like other features slightly activating
  - But for highly sparse features, this cost is outweighted by the benefit of being able to represent more features

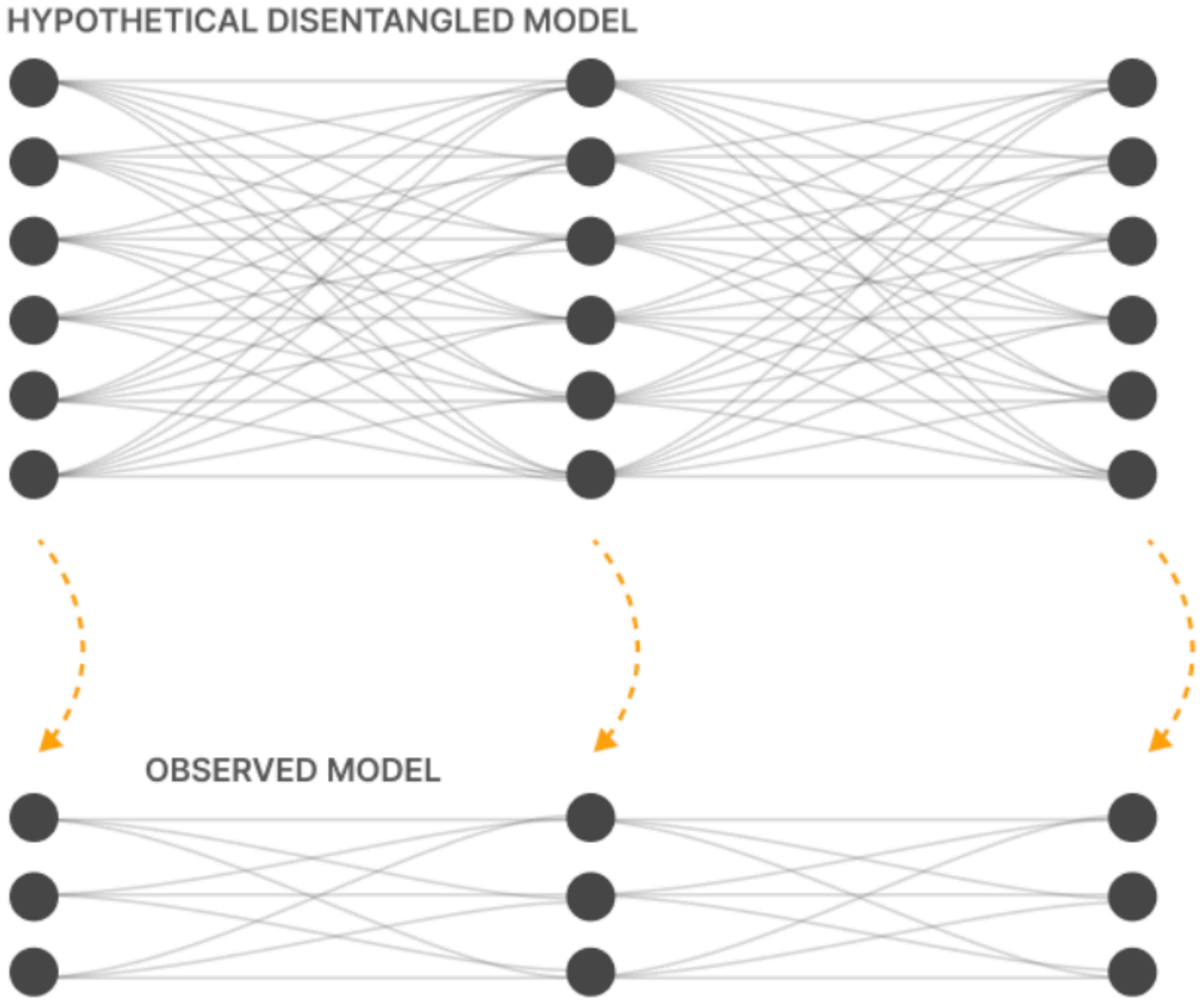


Even if only **one sparse feature** is active, using linear dot product projection on the superposition leads to **interference** which the model must tolerate or filter.



If the features aren’t as sparse as a superposition is expecting, **multiple present features** can additively interfere such that there are multiple possible nonlinear reconstructions of an **activation vector**.

“Noisily simulating larger, sparse networks”



Under the superposition hypothesis, the neural networks we observe are **simulations of larger networks** where every neuron is a disentangled feature.

These idealized neurons are **projected** on to the actual network as “almost orthogonal” vectors over the neurons.

The network we observe is a **low-dimensional projection** of the larger network. From the perspective of individual neurons, this presents as polysemanticity.

Key takeaways

- The approach to understanding models is based on the **linear representation hypothesis** and the **superposition hypothesis**:
  - Neural network activations are *decomposable* and can be decomposed into features
  - Features correspond to *directions*
  - Neural networks use the existence of *almost-orthogonal directions* in high-dimensional spaces to **represent more features than there are dimensions**
- In Toy Models of Superposition, they demonstrate superposition by analysing small ReLU networks trained on synthetic data with sparse input features

Three ways out

1. Create models without superposition

   - In future papers, they conclude that this is not feasible

2. Find an *overcomplete basis* that describes how features are represented in models with superposition

   - This is the focus of the two following Anthropic papers - using **sparse autoeconders** to decompose the activations of a model

3. Hybrid approaches which changes models which don't resolve superposition, but making it easier for a second stage of analysis to find an overcomplete basis that describes it