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‡

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

Transformer [1] language models are an emerging technology that is gaining increasingly broad real-world use, for example in systems like GPT-3 [2], LaMDA [3], Codex [4], Meena [5], Gopher [6], and similar models. However, as these models scale, their open-endedness and high capacity creates an increasing scope for unexpected and sometimes harmful behaviors. Even years after a large model is trained, both creators and users routinely discover model capabilities – including problematic behaviors – they were previously unaware of.

One avenue for addressing these issues is *mechanistic interpretability*, attempting to reverse engineer the detailed computations performed by transformers, similar to how a programmer might try to reverse engineer complicated binaries into human-readable source code. If this were possible, it could potentially provide a more systematic approach to explaining current safety problems, identifying new ones, and perhaps even anticipating the safety problems of powerful future models that have not yet been built. A previous project, the Distill *Circuits* thread [7], has attempted to reverse engineer vision models, but so far there hasn't been a comparable project for transformers or language models.

In this paper, we attempt to take initial, very preliminary steps towards reverse-engineering transformers. Given the incredible complexity and size of modern language models, we have found it most fruitful to start with the simplest possible models and work our way up from there. Our

aim is to discover simple algorithmic patterns, motifs, or frameworks that can subsequently be applied to larger and more complex models. Specifically, in this paper we will study *transformers with two layers or less which have only attention blocks* – this is in contrast to a large, modern transformer like GPT-3, which has 96 layers and alternates attention blocks with MLP blocks.

We find that by conceptualizing the operation of transformers in a new but mathematically equivalent way, we are able to make sense of these small models and gain significant understanding of how they operate internally. Of particular note, we find that specific attention heads that we term "induction heads" can explain in-context learning in these small models, and that these heads only develop in models with at least two attention layers. We also go through some examples of these heads operating in action on specific data.

We don't attempt to apply to our insights to larger models in this first paper, but in a forthcoming paper, we will show that both our mathematical framework for understanding transformers, and the concept of induction heads, continues to be at least partially relevant for much larger and more realistic models – though we remain a very long way from being able to fully reverse engineer such models.

Summary of Results

REVERSE ENGINEERING RESULTS

To explore the challenge of reverse engineering transformers, we reverse engineer several toy, attention-only models. In doing so we find:

- **Zero layer transformers model bigram statistics.** The bigram table can be accessed directly from the weights.

- **One layer attention-only transformers are an ensemble of**

bigram and "skip-trigram" (sequences of the form "A... B C") models. The bigram and skip-trigram tables can be accessed directly from the weights, without running the model. These skip-trigrams can be surprisingly expressive. This includes implementing a kind of very simple in-context learning.

- **Two layer attention-only transformers can implement much more complex algorithms using compositions of attention heads.** These compositional algorithms can also be detected directly from the weights. Notably, two layer models use attention head composition to create "induction heads", a very general in-context learning algorithm.[1]

- One layer and two layer attention-only transformers use very different algorithms to perform in-context learning. Two layer attention heads use qualitatively more sophisticated inference-time algorithms — in particular, a special type of attention head we call an induction head — to perform in-context-learning, forming an important transition point that will be relevant for larger models.

CONCEPTUAL TAKE-AWAYS

We've found that many subtle details of the transformer architecture require us to approach reverse engineering it in a pretty different way from how the InceptionV1 Circuits work [7]. We'll unpack each of these points in the sections below, but for now we briefly summarize. We'll also expand on a lot of the terminology we introduce here once we get to the appropriate sections. (To be clear, we don't intend to claim that any of these points are necessarily novel; many are implicitly or explicitly present in other papers.)

- **Attention heads can be understood as independent operations, each outputting a result which is added into the residual stream.** Attention heads are often described in an alternate "concatenate and multiply" formulation for computational efficiency, but this is mathematically equivalent.

- **Attention-only models can be written as a sum of interpretable end-to-end functions** mapping tokens to changes in logits. These functions correspond to "paths" through the model, and are linear if one freezes the attention patterns.

- **Transformers have an enormous amount of linear structure.** One can learn a lot simply by breaking apart sums and multiplying together chains of matrices.

- Attention heads can be understood as having two largely independent computations: a QK ("query-key") circuit which

computes the attention pattern, and an OV ("output-value") circuit which computes how each token affects the output if attended to.

- Key, query, and value vectors can be thought of as intermediate results in the computation of the low-rank matrices $W_Q^T W_K$ and $W_O W_V$. It can be useful to describe transformers without reference to them.

- Composition of attention heads greatly increases the expressivity of transformers. There are three different ways attention heads can compose, corresponding to keys, queries, and values. Key and query composition are very different from value composition.

- All components of a transformer (the token embedding, attention heads, MLP layers, and unembedding) communicate with each other by reading and writing to different subspaces of the residual stream. Rather than analyze the residual stream vectors, it can be helpful to decompose the residual stream into all these different communication channels, corresponding to paths through the model.

Transformer Overview

Before we attempt to reverse engineer transformers, it's helpful to briefly review the high-level structure of transformers and describe how we think about them.

In many cases, we've found it helpful to reframe transformers in equivalent, but non-standard ways. Mechanistic interpretability requires us to break models down into human-interpretable pieces. An important first step is finding the representation which makes it easiest to reason about the model. In modern deep learning, there is — for good reason! — a lot of emphasis on computational efficiency, and our mathematical descriptions of models often mirror decisions in how one would write efficient code to run the model. But when there are many equivalent

ways to represent the same computation, it is likely that the most human-interpretable representation and the most computationally efficient representation will be different.

Reviewing transformers will also let us align on terminology, which can sometimes vary. We'll also introduce some notation in the process, but since this notation is used across many sections, we provide a detailed description of all notation in the notation appendix as a concise reference for readers.

Model Simplifications

To demonstrate the ideas in this paper in their cleanest form, we focus on "toy transformers" with some simplifications.
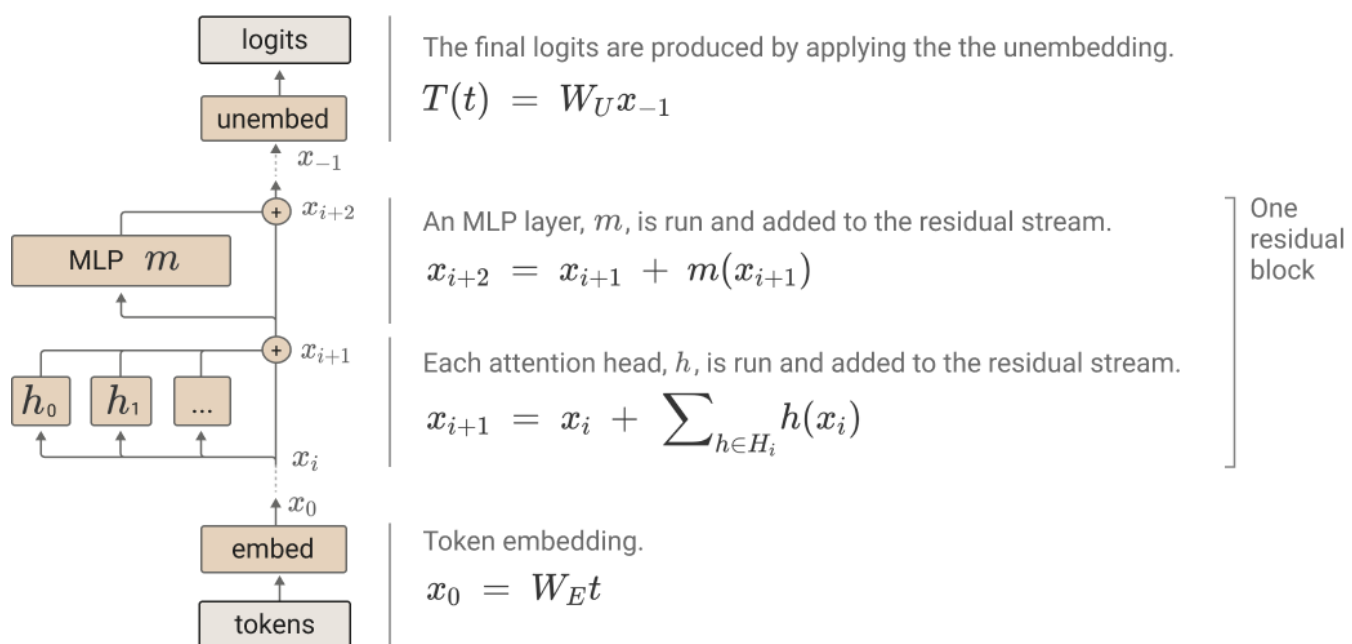
In most parts of this paper, we will make a very substantive change: we focus on "attention-only" transformers, which don't have MLP layers. This is a very dramatic simplification of the transformer architecture. We're partly motivated by the fact that circuits with attention heads present new challenges not faced by the Distill circuits work, and considering them in isolation allows us to give an especially elegant treatment of those issues. But we've also simply had much less success in understanding MLP layers so far; in normal transformers with both attention and MLP layers there are many circuits mediated primarily by attention heads which we can study, some of which seem very important, but the MLP portions have been much harder to get traction on. This is a major weakness of our work that we plan to focus on addressing in the future. Despite this, we will have some discussion of transformers with MLP layers in later sections.

We also make several changes that we consider to be more superficial and are mostly made for clarity and simplicity. We do not consider biases, but a model with biases can always be simulated without them by folding them into the weights and creating a dimension that is always one. Additionally, biases in attention-only transformers mostly multiply out to functionally be biases on the logits. We also ignore layer normalization. It adds a fair amount of complexity to consider explicitly, and up to a variable scaling, layer norm can be merged into adjacent weights. We also expect that, modulo some implementational annoyances, layer norm could be substituted for batch normalization (which can fully be folded into adjacent parameters).

High-Level Architecture

There are several variants of transformer language models. We focus on autoregressive, decoder-only transformer language models, such as GPT-3. (The original transformer paper had a special encoder-decoder structure to support translation, but many modern language models don't include this.)

A transformer starts with a token embedding, followed by a series of "residual blocks", and finally a token unembedding. Each residual block consists of an attention layer, followed by an MLP layer. Both the attention and MLP layers each "read" their input from the residual stream (by performing a linear projection), and then "write" their result to the residual stream by adding a linear projection back in. Each attention layer consists of multiple heads, which operate in parallel.

The final logits are produced by applying the 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.

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

Token embedding.

$$x_0 = W_E t$$

One residual block

Virtual Weights and the Residual Stream as a Communication Channel

One of the main features of the high level architecture of a transformer is that each layer adds its results into what we call the "residual stream."[2] The residual stream is simply the sum of the output of all the previous layers and the original embedding. We generally think of the residual stream as a communication channel, since it doesn't do any processing itself and all layers communicate through it.
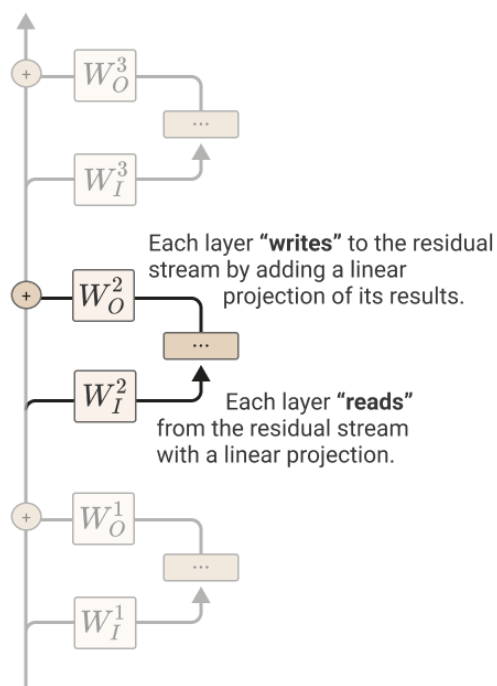
residual stream

The residual stream has a deeply linear structure.[3] Every layer performs an arbitrary linear transformation to "read in" information from the

residual stream at the start,[4] and performs another arbitrary linear transformation before adding to "write" its output back into the residual stream. This linear, additive structure of the residual stream has a lot of important implications. One basic consequence is that the residual stream doesn't have a "privileged basis"; we could rotate it by rotating all the matrices interacting with it, without changing model behavior.
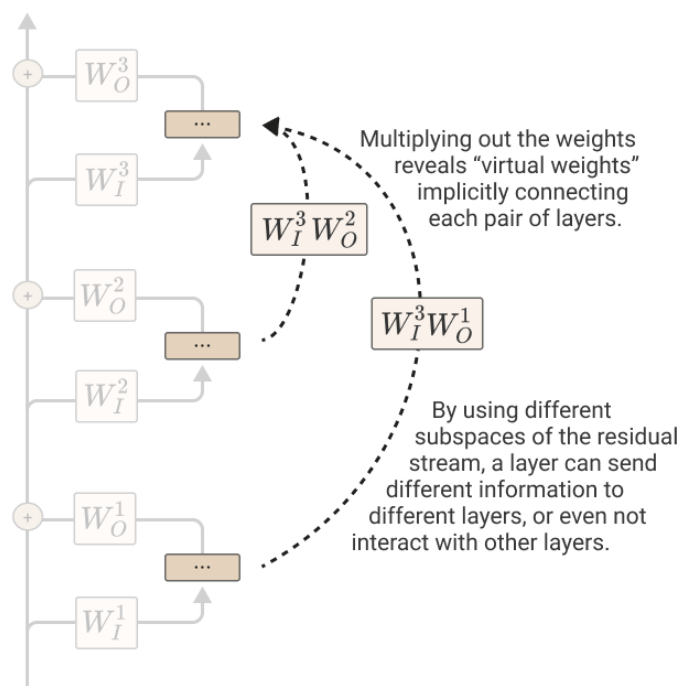
VIRTUAL WEIGHTS

An especially useful consequence of the residual stream being linear is that one can think of implicit "virtual weights" directly connecting any pair of layers (even those separated by many other layers), by multiplying out their interactions through the residual stream. These virtual weights are the product of the output weights of one layer with the input weights[5] of another (ie. $W_I^2 W_O^1$), and describe the extent to which a later layer reads in the information written by a previous layer.

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.

Each layer **"writes"** to the residual stream by adding a linear projection of its results.

Each layer **"reads"** from the residual stream with a linear projection.

Multiplying out the weights reveals "virtual weights" implicitly connecting each pair of layers.

$W_I^3 W_O^2$

$W_I^3 W_O^1$

By using different subspaces of the residual stream, a layer can send different information to different layers, or even not interact with other layers.

SUBSPACES AND RESIDUAL STREAM BANDWIDTH

The residual stream is a high-dimensional vector space. In small models, it may be hundreds of dimensions; in large models it can go into the tens of thousands. This means that layers can send different information to different layers by storing it in different subspaces. This is especially important in the case of attention heads, since every individual head operates on comparatively small subspaces (often 64 or 128

dimensions), and can very easily write to completely disjoint subspaces and not interact.

Once added, information persists in a subspace unless another layer actively deletes it. From this perspective, dimensions of the residual stream become something like "memory" or "bandwidth". The original token embeddings, as well as the unembeddings, mostly interact with a relatively small fraction of the dimensions.[6] This leaves most dimensions "free" for other layers to store information in.

It seems like we should expect residual stream bandwidth to be in very high demand! There are generally far more "computational dimensions" (such as neurons and attention head result dimensions) than the residual stream has dimensions to move information. Just a single MLP layer typically has four times more neurons than the residual stream has dimensions. So, for example, at layer 25 of a 50 layer transformer, the residual stream has 100 times more neurons as it has dimensions before it, trying to communicate with 100 times as many neurons as it has dimensions after it, somehow communicating in superposition! We call tensors like this "bottleneck activations" and expect them to be unusually challenging to interpret. (This is a major reason why we will try to pull apart the different streams of communication happening through the residual stream apart in terms of virtual weights, rather than studying it directly.)

Perhaps because of this high demand on residual stream bandwidth, we've seen hints that some MLP neurons and attention heads may perform a kind of "memory management" role, clearing residual stream dimensions set by other layers by reading in information and writing out the negative version.[7]

The residual stream is high dimensional, and can be divided into different subspaces.
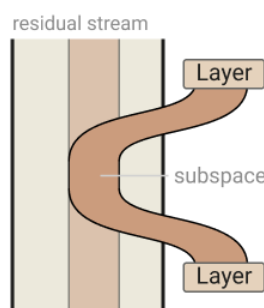
Layers can interact by writing to and reading from the same or overlapping subspaces. If they write to and read from disjoint subspaces, they won't interact. Typically the spaces only partially overlap.
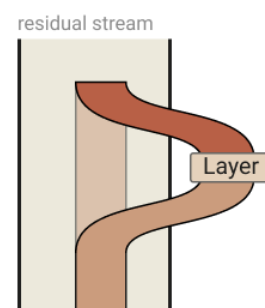
Layers can delete information from the residual stream by reading in a subspace and then writing the negative verison.

Attention Heads are Independent and Additive

As seen above, we think of transformer attention layers as several

completely independent attention heads $h \in H$ which operate completely in parallel and each add their output back into the residual stream. But this isn't how transformer layers are typically presented, and it may not be obvious they're equivalent.

In the original Vaswani *et al.* paper on transformers [1], the output of an attention layer is described by stacking the result vectors r^{h1}, r^{h2}, ..., and then multiplying by an output matrix W_O^H. Let's split W_O^H into equal size blocks for each head $[W_O^{h1}, W_O^{h2} ...]$. Then we observe that:

$$W_O^H \begin{bmatrix} r^{h1} \\ r^{h2} \\ \cdots \end{bmatrix} = \begin{bmatrix} W_O^{h1}, W_O^{h2}, \cdots \end{bmatrix} \cdot \begin{bmatrix} r^{h1} \\ r^{h2} \\ \cdots \end{bmatrix} = \sum_i W_O^{hi} r^{hi}$$

Revealing it to be equivalent to running heads independently, multiplying each by its own output matrix, and adding them into the residual stream. The concatenate definition is often preferred because it produces a larger and more compute efficient matrix multiply. But for understanding transformers theoretically, we prefer to think of them as independently additive.

Attention Heads as Information Movement

But if attention heads act independently, what do they do? The fundamental action of attention heads is moving information. They read information from the residual stream of one token, and write it to the residual stream of another token. The main observation to take away from this section is that which tokens to move information from is completely separable from what information is "read" to be moved and how it is "written" to the destination.
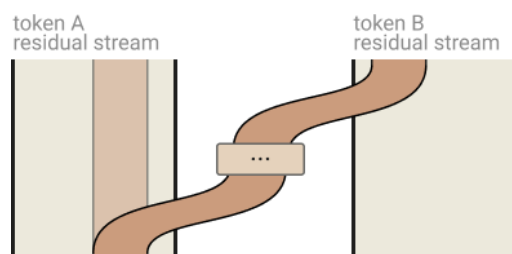
Attention heads copy information from the residual stream of one token to the residual stream of another. They typically write to a different subspace than they read from.

To see this, it's helpful to write attention in a non-standard way. Given an attention pattern, computing the output of an attention head is typically described in three steps:

1. Compute the value vector for each token from the residual stream ($v_i = W_V x_i$).

2. Compute the "result vector" by linearly combining value vectors according to the attention pattern ($r_i = \sum_j A_{i,j} v_j$).

3. Finally, compute the output vector of the head for each token ($h(x)_i = W_O r_i$).[8]

Each of these steps can be written as matrix multiply: why don't we collapse them into a single step? If you think of x as a 2d matrix (consisting of a vector for each token), we're multiplying it on different sides. W_V and W_O multiply the "vector per token" side, while A multiplies the "position" side. Tensors can offer us a much more natural language for describing this kind of map between matrices (if tensor product notation isn't familiar, we've included a short introduction in the notation appendix). One piece of motivation that may be helpful is to note that we want to express linear maps from matrices to matrices: $[n_{\text{context}}, d_{\text{model}}] \to [n_{\text{context}}, d_{\text{model}}]$. Mathematicians call such linear maps "(2,2)-tensors" (they map two input dimensions to two output dimensions). And so tensors are the natural language for expressing this transformation.

Using tensor products, we can describe the process of applying attention as:

$$h(x) \;=\; (\text{Id} \otimes W_O) \;\cdot\; (A \otimes \text{Id}) \;\cdot\; (\text{Id} \otimes W_V) \;\cdot\; x$$

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

Applying the mixed product property and collapsing identities yields:

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

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

What about the attention pattern? Typically, one computes the keys $k_i = W_K x_i$, computes the queries $q_i = W_Q x_i$ and then computes the attention pattern from the dot products of each key and query vector $A = \text{softmax}(q^T k)$. But we can do it all in one step without referring to keys and queries: $A = \text{softmax}(x^T W_Q^T W_K x)$.

It's worth noting that although this formulation is mathematically equivalent, actually implementing attention this way (ie. multiplying by $W_O W_V$ and $W_Q^T W_K$) would be horribly inefficient!

OBSERVATIONS ABOUT ATTENTION HEADS

A major benefit of rewriting attention heads in this form is that it surfaces a lot of structure which may have previously been harder to observe:

- Attention heads move information from the residual stream of one token to another.

    - A corollary of this is that the residual stream vector space — which is often interpreted as a "contextual word embedding" — will generally have linear subspaces corresponding to information copied from other tokens and not directly about the present token.

- An attention head is really applying two linear operations, A and $W_O W_V$, which operate on different dimensions and act independently.

    - A governs which tokens information is moved from and to.

    - $W_O W_V$ governs which information is read from the source token and how it is written to the destination token.[9]

- A is the only non-linear part of this equation (being computed from a softmax). This means that if we fix the attention pattern, attention heads perform a linear operation. This also means that, without fixing A, attention heads are "half-linear" in some sense, since the per-token linear operation is constant.

- W_Q and W_K always operate together. They're never independent. Similarly, W_O and W_V always operate together as well.

    - Although they're parameterized as separate matrices, $W_O W_V$ and $W_Q^T W_K$ can always be thought of as individual, low-rank matrices.

    - Keys, queries and value vectors are, in some sense, superficial. They're intermediary by-products of computing these low-rank matrices. One could easily reparametrize both factors of the low-rank matrices to create different vectors, but still function identically.

    - Because $W_O W_V$ and $W_Q W_K$ always operate together, we like to define variables representing these combined matrices, $W_{OV} = W_O W_V$ and $W_{QK} = W_Q^T W_K$.

- Products of attention heads behave much like attention heads themselves. By the distributive property,
$$(A^{h2} \otimes W_{OV}^{h2}) \cdot (A^{h1} \otimes W_{OV}^{h1}) = (A^{h2} A^{h1}) \otimes (W_{OV}^{h2} W_{OV}^{h1}).$$

The result of this product can be seen as functionally equivalent to an attention head, with an attention pattern which is the composition of the two heads $A^{h2} A^{h1}$ and an output-value matrix $W_{OV}^{h2} W_{OV}^{h1}$. We call these "virtual attention heads", discussed in more depth later.

Zero-Layer Transformers

Watch videos covering similar content to this section: 0 layer theory

Before moving on to more complex models, it's useful to briefly consider a "zero-layer" transformer. Such a model takes a token, embeds it, unembeds it to produce logits predicting the next token:

$$T \;=\; W_U W_E$$

Because the model cannot move information from other tokens, we are simply predicting the next token from the present token. This means that the optimal behavior of $W_U W_E$ is to approximate the bigram log-likelihood.[10]

This is relevant to transformers more generally. Terms of the form $W_U W_E$ will occur in the expanded form of equations for every transformer, corresponding to the "direct path" where a token embedding flows directly down the residual stream to the unembedding, without going through any layers. The only thing it can affect is the bigram log-likelihoods. Since other aspects of the model will predict parts of the bigram log-likelihood, it won't exactly represent bigram statistics in larger models, but it does represent a kind of "residual". In particular, the $W_U W_E$ term seems to often help represent bigram statistics which aren't described by more general grammatical rules, such as the fact that "Barack" is often followed by "Obama".[11]
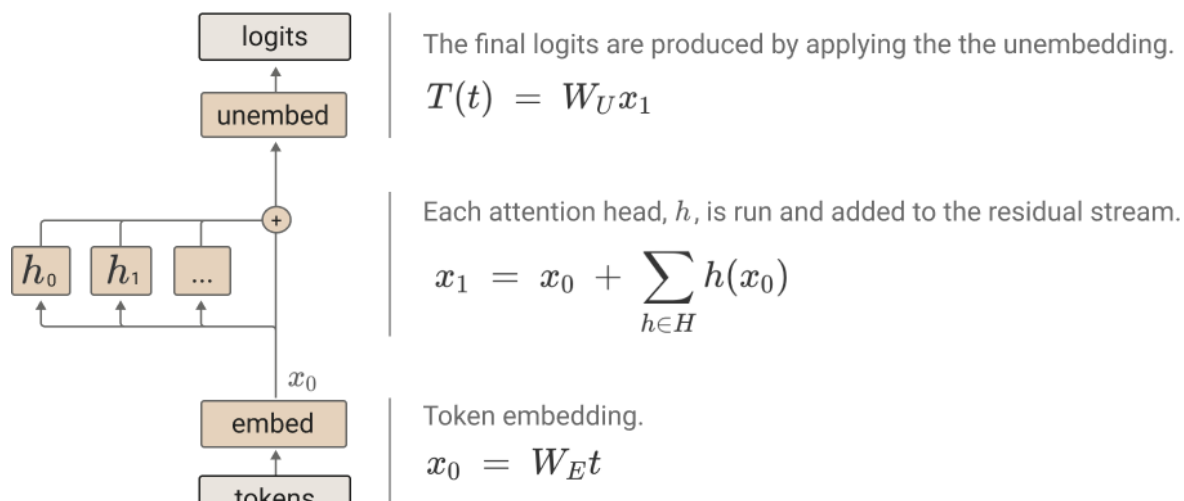
One-Layer Attention-Only Transformers

We claim that one-layer attention-only transformers can be understood as an ensemble of a bigram model and several "skip-trigram" models (affecting the probabilities of sequences "A... BC").[12] Intuitively, this is because each attention head can selectively attend from the present token ("B") to a previous token ("A") and copy information to adjust the probability of possible next tokens ("C").

The goal of this section is to rigorously show this correspondence, and demonstrate how to convert the raw weights of a transformer into interpretable tables of skip-trigram probability adjustments.
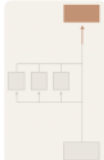
The Path Expansion Trick

Recall that a one-layer attention-only transformers consists of a token embedding, followed by an attention layer (which independently applies attention heads), and finally an unembedding:

The final logits are produced by applying the the unembedding.

$$T(t) = W_U x_1$$

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

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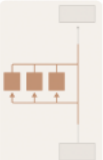
Token embedding.

$$x_0 = W_E t$$

Using tensor notation and the alternative representation of attention heads we previously derived, we can represent the transformer as a product of three terms.
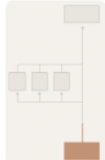
$$T \quad = \quad \underbrace{\mathrm{Id} \otimes W_U}_{} \quad \cdot \quad \underbrace{\left(\mathrm{Id} + \sum_{h \in H_1} A^h \otimes W_{OV}^h \right)}_{} \quad \cdot \quad \underbrace{\mathrm{Id} \otimes W_E}_{}$$

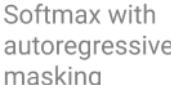The **token unembedding** maps residual stream vectors to logits.

The **attention layer** has multiple heads. The result of each is added into the residual stream.
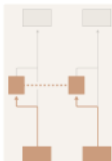
The **token embedding** maps tokens to residual stream vectors.

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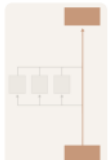
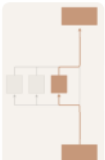Softmax with autoregressive masking

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

Our key trick is to simply expand the product. This transforms the product (where every term corresponds to a layer), into *a sum where every term corresponds to an end-to-end path*.

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

"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.

We claim each of these end-to-end path terms is tractable to understand, can be reasoned about independently, and additively combine to create model behavior.

The direct path term, $\mathrm{Id} \otimes W_U W_E$, also occurred when we looked at the zero-layer transformer. Because it doesn't move information between positions (that's what $\mathrm{Id} \otimes \ldots$ denotes!), the only thing it can contribute to is the bigram statistics, and it will fill in missing gaps that other terms

don't handle there.

The more interesting terms are the attention head terms.

Splitting Attention Head terms into Query-Key and Output-Value Circuits

For each attention head h we have a term $A^h \otimes (W_U W_{OV}^h W_E)$ where $A^h = \text{softmax}\left(t^T \cdot W_E^T W_{QK}^h W_E \cdot t\right)$. How can we map these terms to model behavior? And while we're at it, why do we get these particular products of matrices on our equations?

The key thing to notice is that these terms consist of two separable operations, which are at their heart two $[n_{\text{vocab}},\ n_{\text{vocab}}]$ matrices:

- $W_E^T W_{QK}^h W_E$ — We call this matrix the "query-key (QK) circuit." It provides the attention score for every query and key token. That is, each entry describes how much a given query token "wants" to attend to a given key token.

- $W_U W_{OV}^h W_E$ — We call this matrix the "Output-Value (OV) circuit." It describes how a given token will affect the output logits if attended to.

To intuitively understand these products, it can be helpful to think of them as paths through the model, starting and ending at tokens. The QK circuit is formed by tracing the computation of a query and key vector up to their attention head, where they dot product to create a bilinear form. The OV circuit is created by tracing the path computing a value vector and continuing it through up to the logits.
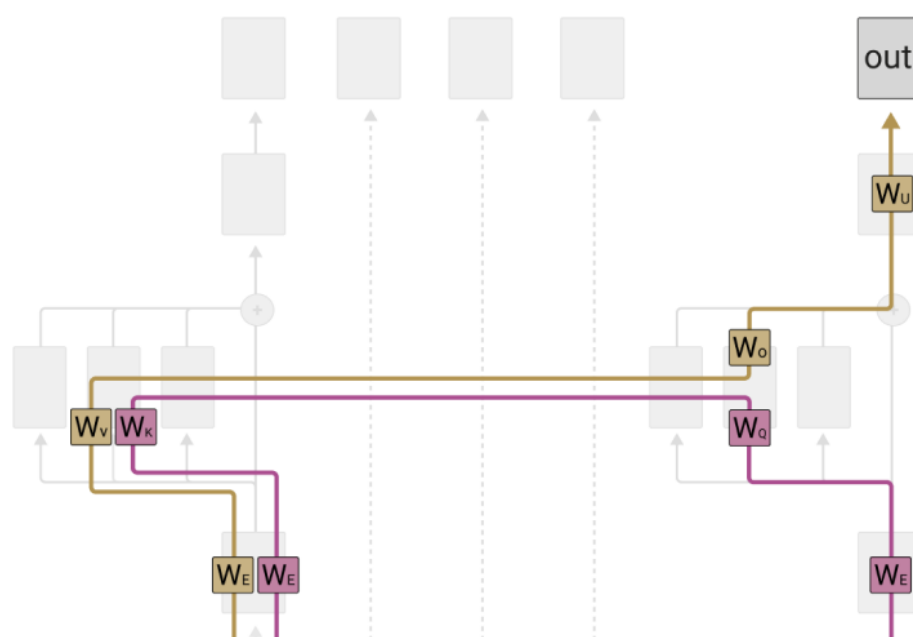
The **OV ("output-value") circuit** determines how attending to a given token affects the logits.

$$W_U W_O W_V W_E$$

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

$$W_E^T W_O^T W_K W_E$$

The attention pattern is a function of both the source and destination token[13] , but once a destination token has decided how much to attend to a source token, the effect on the output is solely a function of that source token. That is, if multiple destination tokens attend to the same source token the same amount, then the source token will have the same effect on the logits for the predicted output token.

OV AND QK INDEPENDENCE (THE FREEZING ATTENTION PATTERNS TRICK)

Thinking of the OV and QK circuits separately can be very useful, since they're both individually functions we can understand (linear or bilinear functions operating on matrices we understand).

But is it really principled to think about them independently? One thought experiment which might be helpful is to imagine running the model twice. The first time you collect the attention patterns of each head. This only depends on the QK circuit.[14] The second time, you replace the attention patterns with the "frozen" attention patterns you collected the first time. This gives you a function where the logits are a linear function of the tokens! We find this a very powerful way to think about transformers.

Interpretation as Skip-Trigrams

One of the core challenges of mechanistic interpretability is to make neural network parameters meaningful by contextualizing them (see discussion by Voss *et al*. in Visualizing Weights [12]). By multiplying out the OV and QK circuits, we've succeeded in doing this: the neural network parameters are now simple linear or bilinear functions on tokens. The QK circuit determines which "source" token the present "destination" token attends back to and copies information from, while the OV circuit describes what the resulting effect on the "out" predictions for the next token is. Together, the three tokens involved form a "skip-trigram" of the form `[source]... [destination][out]` , and the "out" is modified.

It's important to note that this doesn't mean that interpretation is trivial. For one thing, the resulting matrices are enormous (our vocabulary is ~50,000 tokens, so a single expanded OV matrix has ~2.5 billion entries):

we revealed the one-layer attention-only model to be a compressed Chinese room, and we're left with a giant pile of cards. There's also all the usual issues that come with understanding the weights of generalized linear models acting on correlated variables and fungibility between variables. For example, an attention head might have a weight of zero because another attention head will attend to the same token and perform the same role it would have. Finally, there's a technical issue where QK weights aren't comparable between different query vectors, and there isn't a clear right answer as to how to normalize them.

Despite this, we do have transformers in a form where all parameters are contextualized and understandable. And despite these subtleties, we can simply read off skip-trigrams from the joint OV and QK matrices. In particular, searching for large entries in these matrices reveals lots of interesting behavior.

In the following subsections, we give a curated tour of some interesting skip-trigrams and how they're embedded in the QK/OV circuits. But full, non-cherrypicked examples of the largest entries in several models are available by following the links:

- **Large QK/OV entries - 12 heads, d_head=64**

- **Large QK/OV entries - 32 heads, d_head=128**

COPYING / PRIMITIVE IN-CONTEXT LEARNING

One of the most striking things about looking at these matrices is that most attention heads in one layer models dedicate an enormous fraction of their capacity to copying. The OV circuit sets things up so that tokens, if attended to by the head, increase the probability of that token, and to a lesser extent, similar tokens. The QK circuit then only attends back to tokens which could plausibly be the next token. Thus, tokens are copied, but only to places where bigram-ish statistics make them seem plausible.

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"

Source Token	Destination Token	Out Token	Example Skip Tri-grams
"nbsp"	"&", "\"&", "}&", ">&", "=&"	"nbsp", "01", "gt", "00012", "nbs", "quot"	"nbsp... ", "nbsp... > "
"Great"	"The", " The", " the", " contains", " /"	" Great", " great", " poor", " Every"	"Great... The Great", "Great... the great"

In the above example, we fix a given source token and look at the largest corresponding QK entries (the destination token) and largest corresponding OV entries (the out token). The source token is selected to show interesting behavior, but the destination and out token are the top entries unless entries are explicitly skipped with an ellipsis; they are colored by the intensity of their value in the matrix.

Most of the examples are straightforward, but two deserve explanation: the fourth example (with skip-trigrams like `lambda… $\lambda`) appears to be the model learning LaTeX, while the fifth example (with the skip-trigram `nbsp… > `) appears to be the model learning HTML escape sequences.

Note that most of these examples are copying; this appears to be very common.

We also see more subtle kinds of copying. One particularly interesting one is related to how tokenization for transformers typically works. Tokenizers typically merge spaces onto the start of words. But occasionally a word will appear in a context where there isn't a space in front of it, such as at the start of a new paragraph or after a dialogue open quote. These cases are rare, and as such, the tokenization isn't optimized for them. So for less common words, it's quite common for them to map to a single token when a space is in front of them (`" Ralph" → [" Ralph"]`) but split when there isn't a space (`"Ralph" → ["R", "alph"]`).

It's quite common to see skip-trigram entries dealing with copying in this case. In fact, we sometimes observe attention heads which appear to partially specialize in handling copying for words that split into two tokens without a space. When these attention heads observe a fragmented token (e.g. `"R"`) they attend back to tokens which might be the complete word with a space (`" Ralph"`) and then predict the continuation (`"alph"`). (It's interesting to note that this could be thought of as a very special case where a one-layer model can kind of mimic the <u>induction heads</u> we'll see in two layer models.)

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", " P"	"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"

We can summarize this copying behavior into a few abstract patterns that we've observed:

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]	

All of these can be seen as a kind of very primitive in-context learning. The ability of transformers to adapt to their context is one of their most interesting properties, and this kind of simple copying is a very basic form of it. However, we'll see when we look at a two-layer transformer that a much more interesting and powerful algorithm for in-context learning is available to deeper transformers.

OTHER INTERESTING SKIP-TRIGRAMS

Of course, copying isn't the only behavior these attention heads encode.

Skip-trigrams seem trivial, but can actually produce more complex behavior than one might expect. Below are some particularly striking skip-trigram examples we found in looking through the largest entries in the expanded OV/QK matrices of our models.

- **[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` :

- **[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__()`)

- **[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)

- **[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]
```

One thing to note is that the learned skip-trigrams are often related to idiosyncrasies of one's tokenization. For example collapsing whitespace together allows individual tokens to reveal indentation. Not merging backslash into text tokens means that when the model is predicting LaTeX, there's a token after backslash that must be an escape sequence. And so on.

Many skip tri-grams can be difficult to interpret without specific knowledge (e.g. `Israel … K → nes` only makes sense if you know Israel's legislative body is called the "Knesset"). A useful tactic can be to try typing potential skip tri-grams into Google search (or similar tools) and look at autocompletions.

PRIMARILY POSITIONAL ATTENTION HEADS

Our treatment of attention heads hasn't discussed how attention heads handle position, largely because there are now several competing methods (*e.g.* [1, 13, 14]) and they would complicate our equations. (In the case of standard positional embeddings, the one-layer math works out to multiplying W_{QK} by the positional embeddings.)

In practice, the one-layer models tend to have a small number of attention heads that are primarily positional, strongly preferring certain relative positions. Below, we present one attention head which either attends to the present token or the previous token.[15]

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"

SKIP-TRIGRAM "BUGS"

One of the most interesting things about looking at the expanded QK and OV matrices of one layer transformers is that they can shed light on transformer behavior that seems incomprehensible from the outside.

Our one-layer models represent skip-trigrams in a "factored form" split between the OV and QK matrices. It's kind of like representing a function $f(a,b,c) = f_1(a,b)f_2(a,c)$. They can't really capture the three way interactions flexibly. For example, if a single head increases the probability of both `keep… in mind` and `keep… at bay`, it *must* also increase the probability of `keep… in bay` and `keep… at mind`. This is likely a good trade for the model on balance, but is also, in some sense, a bug. We frequently observe these in attention heads.

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… P**Canvas**"

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… **C**loyd", " Lloyd… **L**atherine"

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**"

Highlighted text denotes skip-trigram continuations that the model presumably ideally wouldn't increase the probability of. Note that `QCanvas` is a class involving pixmaps in the popular Qt library. `Lloyd... Catherine` likely refers to Catherine Lloyd Burns. These examples are slightly cherry-picked to be interesting, but very common if you look at the expanded weights for models linked above.

Even though these particular bugs seem in some sense trivial, we're excited about this result as an early demonstration of using interpretability to understand model failures. We have not further explored this phenomenon, but we'd be curious to do so in more detail. For instance, could we characterize how much performance (in points of loss or otherwise) these "bugs" are costing the model? Does this particular class continue to some extent in larger models (presumably partially, but not completely, masked by other effects)?

Summarizing OV/QK Matrices

We've turned the problem of understanding one-layer attention-only transformers into the problem of understanding their expanded OV and QK matrices. But as mentioned above, the expanded OV and QK matrices are enormous, with easily billions of entries. While searching for the largest entries is interesting, are there better ways to understand them? There are at least three reasons to expect there are:

- The OV and QK matrices are extremely low-rank. They are 50,000 x 50,000 matrices, but only rank d_{head} (64 or 128). In some sense, they're quite small even though they appear large in their expanded form.

- Looking at individual entries often reveals hints of much simpler structure. For example, we observe one head where names of people all have the top queries like `" by"` (e.g. `"Anne… by → Anne"`) while location names have top queries like `" from"` (e.g. `"Canada… from → Canada"`). This hints at something like cluster structure in the matrix.

- Copying behavior is widespread in OV matrices and arguably one of the most interesting behaviors. (We'll see in the next section that there's analogous QK matrix structure in two layer models that's used to search for similar tokens to a query.) It seems like it should be possible to formalize this.

We don't yet feel like we have a clear right answer, but we're optimistic that the right kind of matrix decomposition or dimensionality reduction could be highly informative. (See the technical details appendix for notes on how to efficiently work with these large matrices.)

DETECTING COPYING BEHAVIOR

The type of behavior we're most excited to detect in an automated way is copying. Since copying is fundamentally about mapping the same vector to itself (for example, having a token increase its own probability) it seems unusually amenable to being captured in some kind of summary statistic.

However, we've found it hard to pin down exactly what the right notion is; this is likely because there are lots of slightly different ways one could draw the boundaries of whether something is a "copying matrix" and we're not yet sure what the most useful one is. For example, we don't observe this in the models discussed in this paper, but in slightly larger models we often observe attention heads which "copy" some mixture of gender, plurality, and tense from nearby words, helping the model use the correct pronouns and conjugate verbs. The matrices for these attention heads aren't exactly copying individual tokens, but it seems like they are copying in some very meaningful sense. So copying is actually a more complex concept than it might first appear.

One natural approach might be to use eigenvectors and eigenvalues. Recall that v_i is an eigenvector of the matrix M with an eigenvalue λ_i if $Mv_i = \lambda_i v_i$. Let's consider what that means for an OV circuit $M = W_U W_{OV}^h W_E$ if λ_i is a positive real number. Then we're saying that there's a linear combination of tokens[16] which increases the linear combination of logits of those same tokens. Very roughly you could think

of this as a set of tokens (perhaps all tokens representing plural words for a very broad one, or all tokens starting with a given first letter, or all tokens representing different capitalizations and inclusions of space for a single word for a narrow one) which mutually increase their own probability. Of course, in general we expect the eigenvectors have both positive and negative entries, so it's more like there are two sets of tokens (e.g. tokens representing male and female words, or tokens representing singular and plural words) which increase the probability of other tokens in the same set and decrease those in others.

The eigendecomposition expresses the matrix as a set of such eigenvectors and eigenvalues. For a random matrix, we expect to have an equal number of positive and negative eigenvalues, and for many to be complex.[17] But copying requires positive eigenvalues, and indeed we observe that many attention heads have positive eigenvalues, apparently mirroring the copying structure:

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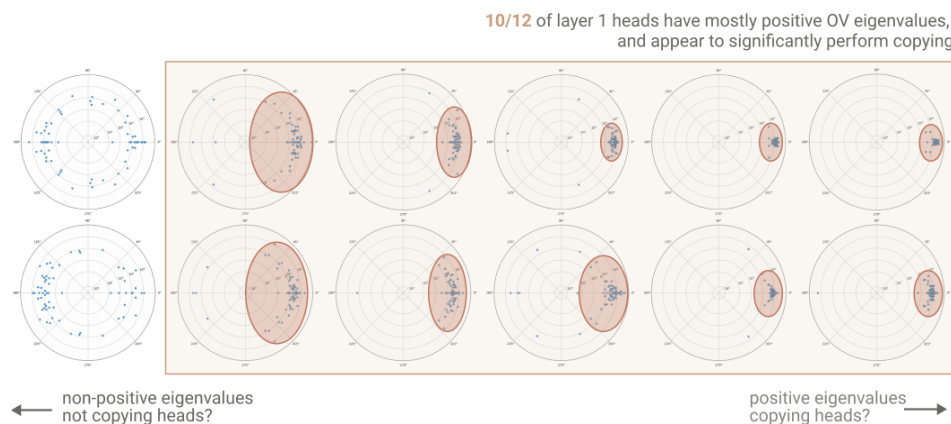
We use a **log scale** to represent magnitude, since it varies by many orders of magnitude.

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.

non-positive eigenvalues
not copying heads?
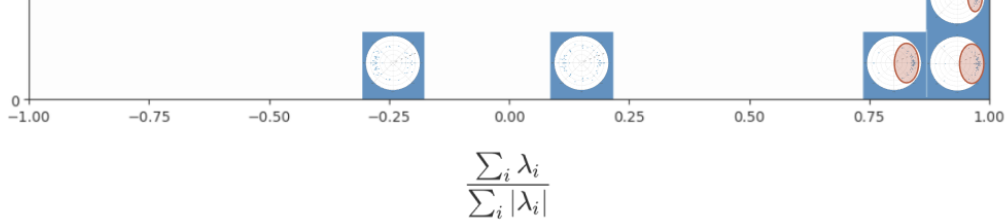
positive eigenvalues
copying heads?
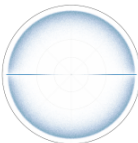
One can even collapse that down further and get a histogram of how many of the attention heads are copying (if one trusts the eigenvalues as a summary statistic):

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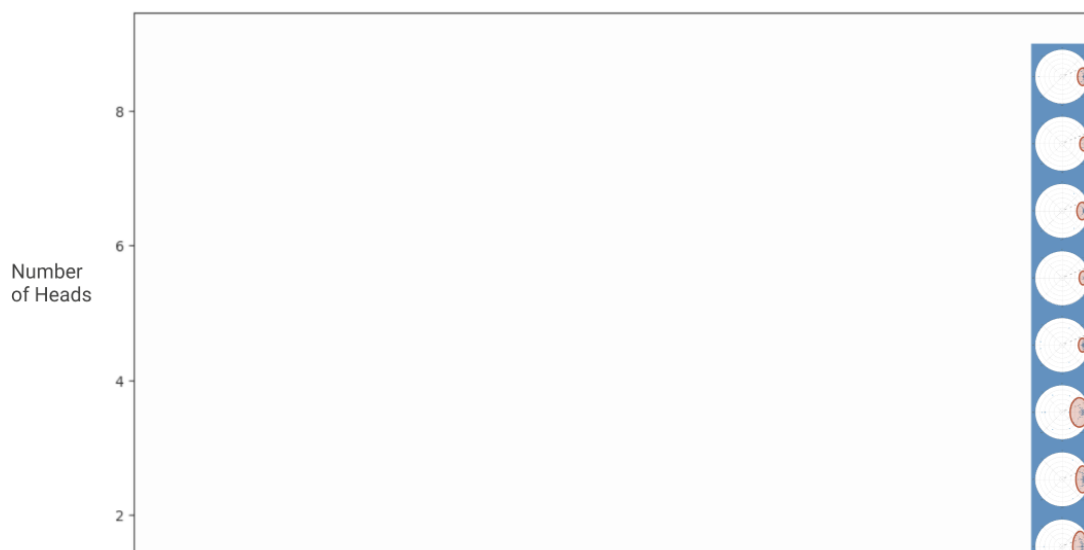
Histograms of attention heads by the fraction of their eigenvalues that are positive can be a useful summary.

Number of Heads

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

It appears that 10 out of 12 heads are significantly copying! (This agrees with qualitative inspection of the expanded weights.)

But while copying matrices must have positive eigenvalues, it isn't clear that all matrices with positive eigenvalues are things we necessarily want to consider to be copying. A matrix's eigenvectors aren't necessarily orthogonal, and this allows for pathological examples;[18] for example, there can be matrices with all positive eigenvalues that actually map some tokens to decreasing the logits of that same token. Positive eigenvalues still mean that the matrix is, in some sense, "copying on average", and they're still quite strong evidence of copying in that they seem improbable by default and empirically seem to align with copying. But they shouldn't be considered a dispositive proof that a matrix is copying in all senses one might reasonably mean.

One might try to formalize "copying matrices" in other ways. One possibility is to look at the diagonal of a matrix, which describes how each token affects its own probability. As expected, entries on the diagonal are very positive-leaning. We can also ask how often a random token increases its own probability more than any other token (or is one of the k-most increased tokens, to allow for tokens which are the same with a different capitalization or with a space). All of these seem to point in the direction of these attention heads being copying matrices, but it's not clear that any of them is a fully robust formalization of "the primary behavior of this matrix is copying". It's worth noting that all of these potential notions of copying are linked by the fact that the sum of the eigenvalues is equal to the trace is equal to the sum of the diagonal.

For the purposes of this paper, we'll continue to use the eigenvalue-based summary statistic. We don't think it's perfect, but it seems like quite strong evidence of copying, and empirically aligns with manual inspection and other definitions.

Do We "Fully Understand" One-Layer Models?

There's often skepticism that it's even possible or worth trying to truly reverse engineer neural networks. That being the case, it's tempting to

point at one-layer attention-only transformers and say "look, if we take the most simplified, toy version of a transformer, at least that minimal version can be fully understood."

But that claim really depends on what one means by fully understood. It seems to us that we now understand this simplified model in the same sense that one might look at the weights of a giant linear regression and understand it, or look at a large database and understand what it means to query it. That is a kind of understanding. There's no longer any algorithmic mystery. The contextualization problem of neural network parameters has been stripped away. But without further work on summarizing it, there's far too much there for one to hold the model in their head.

Given that regular one layer neural networks are just generalized linear models and can be interpreted as such, perhaps it isn't surprising that a single attention layer is mostly one as well.

Two-Layer Attention-Only Transformers

Videos covering similar content to this section: [2 layer theory](#), [2 layer term importance](#), [2 layer results](#)

Deep learning studies models that are *deep*, which is to say they have many layers. Empirically, such models are very powerful. Where does that power come from? One intuition might be that depth allows composition, which creates powerful expressiveness.

Composition of attention heads is the key difference between one-layer and two-layer attention-only transformers. Without composition, a two-layer model would simply have more attention heads to implement skip-trigrams with. But we'll see that in practice, two-layer models discover ways to exploit attention head composition to express a much more powerful mechanism for accomplishing in-context learning. In doing so, they become something much more like a computer program running an

algorithm, rather than look-up tables of skip-trigrams we saw in one-layer models.

Three Kinds of Composition

Recall that we think of the residual stream as a communication channel. Every attention head reads in subspaces of the residual stream determined by W_Q, W_K, and W_V, and then writes to some subspace determined by W_O. Since the attention head vectors are much smaller than the size of the residual stream (typical values of $d_{\text{head}}/d_{\text{model}}$ might vary from around $1/10$ to $1/100$), attention heads operate on small subspaces and can easily avoid significant interaction.

When attention heads do compose, there are three options:

- Q-Composition: W_Q reads in a subspace affected by a previous head.

- K-Composition: W_K reads in a subspace affected by a previous head.

- V-Composition: W_V reads in a subspace affected by a previous head.
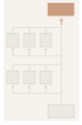
Q- and K-Composition are quite different from V-Composition. Q- and K-Composition both affect the attention pattern, allowing attention heads to express much more complex patterns. V-Composition, on the other hand, affects what information an attention head moves when it attends to a given position; the result is that V-composed heads really act more like a single unit and can be thought of as creating an additional "virtual attention heads". Composing movement of information with movement of information gives movement of information, whereas attention heads affecting attention patterns is not reducible in this way.

To really understand these three kinds of composition, we'll need to study the OV and QK circuits again.
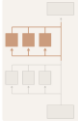
Path Expansion of Logits

The most basic question we can ask of a transformer is "how are the logits computed?" Following our approach to the one-layer model, we write out a product where every term is a layer in the model, and expand to create a sum where every term is an end-to-end path through the model.
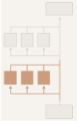
$$T \;=\; \mathrm{Id} \otimes W_U \;\cdot\; \left(\mathrm{Id} + \sum_{h \in H_2} A^h \otimes W_{OV}^h\right) \;\cdot\; \left(\mathrm{Id} + \sum_{h \in H_1} A^h \otimes W_{OV}^h\right) \;\cdot\; \mathrm{Id} \otimes W_E$$
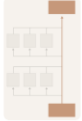
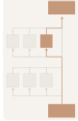The second **attention layer** has multiple attention heads which add into the residual stream

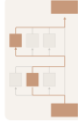The first **attention layer** has multiple attention heads which add into the residual stream

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

"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.

Two of these terms, the direct path term and individual head terms, are identical to the one-layer model. The final "virtual attention head" term corresponds to V-Composition. Virtual attention heads are conceptually very interesting, and we'll discuss them more later. However, in practice, we'll find that they tend to not play a significant role in small two-layer models.

Path Expansion of Attention Scores QK Circuit

Just looking at the logit expansion misses what is probably the most radically different property of a two-layer attention-only transformer: Q-composition and K-composition cause them to have much more expressive second layer attention patterns.

To see this, we need to look at the QK circuits computing the attention patterns. Recall that the attention pattern for a head h is $A^h = \mathrm{softmax}^*\left(t^T \cdot C_{QK}^h t\right)$, where C_{QK}^h is the "QK-circuit" mapping tokens to attention scores. For first layer attention heads, the QK-circuit is just the same matrix we saw in the one-layer model: $C_{QK}^{h \in H1} = W_E^T W_{QK}^h W_E$.

But for the second layer QK-circuit, both Q-composition and K-composition come into play, with previous layer attention heads potentially influencing the construction of the keys and queries. Ultimately, W_{QK} acts on the residual stream. In the case of the first layer this reduced to just acting on the token embeddings: $C_{QK}^{h \in H1} = x_0^T W_{QK}^h x_0 = W_E^T W_{QK}^h W_E$. But by the second layer, $C_{QK}^{h \in H2} = x_1^T W_{QK}^h x_1$ is acting on x_1, the residual stream after first layer attention heads. We can write this down as a product, with the first layer

both on the "key side" and "query side." Then we apply our path expansion trick to the product.

One complicating factor is that we have to write it as a 6-dimensional tensor, using two tensor products on matrices. This is because we're trying to express a multilinear function of the form $[n_{\text{context}},\ d_{\text{model}}] \times [n_{\text{context}},\ d_{\text{model}}] \rightarrow [n_{\text{context}},\ n_{\text{context}},]$. In the one-layer case, we could side step this by implicitly doing an outer product, but that no longer works. A natural way to express this is as a (4,2)-tensor (one with 4 input dimensions and 2 output dimensions). Each term will be of the form $A_q \otimes A_k \otimes W$ where $x(A_q \otimes A_k \otimes W)y = A_q^T x W y A_k$, meaning that A_q describes the movement of query-side information between tokens, A_k describes the movement of key-side information between tokens, and W describes how they product together to form an attention score.

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

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 $\ldots \otimes \text{Id} \otimes \ldots$ 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 $\text{Id} \otimes \ldots$ because they don't move query information.

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

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.

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

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.

Each of these terms corresponds to a way the model can implement more complex attention patterns. In the abstract, it can be hard to reason about them. But we'll return to them with a concrete case shortly, when we talk about induction heads.

Analyzing a Two-Layer Model

So far, we've developed a theoretical model for understanding two-layer attention-only models. We have an overall equation describing the logits

attention-only models. We have an overall equation describing the logits (the OV circuit), and then an equation describing how each attention head's attention pattern is computed (the QK circuit). But how do we understand them in practice? In this section, we'll reverse engineer a single two-layer model.

Recall that the key difference between a two-layer model and a one-layer model is Q-, K-, and V-composition. Without composition, the model is just a one-layer model with extra heads.

Small two-layer models seem to often (though not always) have a very simple structure of composition, where the only type of composition is K-composition between a single first layer head and some of the second layer heads.[19] The following diagram shows Q-, K-, and V-composition between first and second layer heads in the model we wish to analyze. We've colored the heads involved by our understanding of their behavior. The first layer head has a very simple attention pattern: it primarily attends to the previous token, and to a lesser extent the present token and the token two back. The second layer heads are what we call *induction heads*.
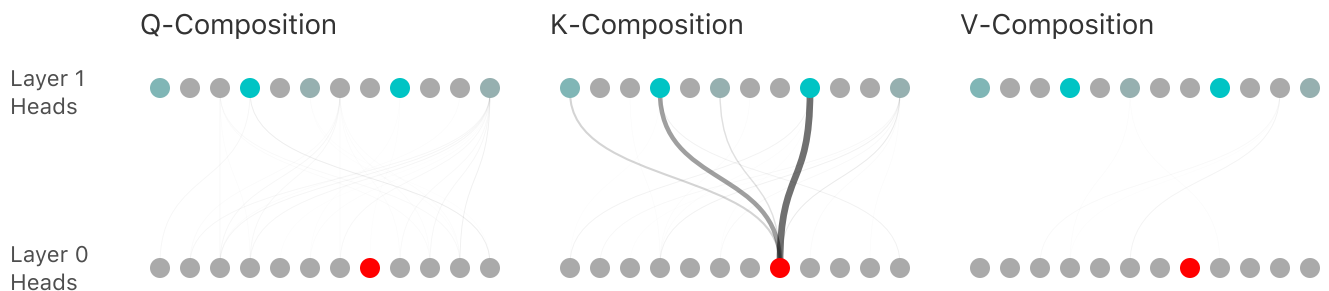

The above diagram shows Q-, K-, and V-Composition between attention heads in the first and second layer. That is, how much does the query, key or value vector of a second layer head read in information from a given first layer head? This is measured by looking at the Frobenius norm of the product of the relevant matrices, divided by the norms of the individual matrices. For Q-Composition, $\|W_{QK}^{h2\,T}W_{OV}^{h1}\|_F/(\|W_{QK}^{h2}\|_F\|W_{OV}^{h1}\|_F)$, for K-Composition $\|W_{QK}^{h2}W_{OV}^{h1}\|_F/(\|W_{QK}^{h2\,T}\|_F\|W_{OV}^{h1}\|_F)$, for V-Composition $\|W_{OV}^{h2}W_{OV}^{h1}\|_F/(\|W_{OV}^{h2}\|_F\|W_{OV}^{h1}\|_F)$. By default, we subtract off the empirical expected amount for random matrices of the same shapes (most attention heads have a much smaller composition than random matrices). In practice, for this model, there is only significant K-composition, and only with one layer 0 head.

One quick observation from this is that most attention heads are not involved in any substantive composition. We can think of them as, roughly, a larger collection of skip tri-grams. This two-layer model has a mystery for us to figure out, but it's a fairly narrowly scoped one. (We speculate this means that having a couple induction heads in some sense "outcompetes" a few potential skip-trigram heads, but no other type of composition did. That is, having more skip-trigram heads is a competitive use of second layer attention heads in a small model.)

In the next few sections, we'll develop a theory of what's going on, but before we do, we provide an opportunity to poke around at the attention heads using the interactive diagram below, which displays value-weighted attention patterns over the first paragraph of *Harry Potter and the Philosopher's Stone*. We've colored the attention heads involved in K-composition using the same scheme as above. (This makes it a bit hard to investigate the other heads; if you want to look at those, an interface for general exploration is available here).

We recommend isolating individual heads and both looking at the pattern and hovering over tokens. For induction heads, note especially the off-diagonal lines in the attention pattern, and the behavior on the tokens compositing Dursley and Potters.

Attention Pattern
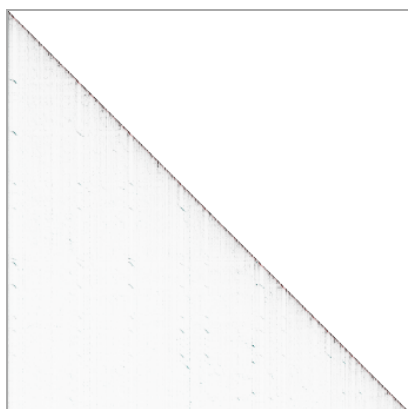
Attention Heads (hover to focus, click to lock)

0:0 0:1 0:2 0:3 0:4 0:5 0:6

0:7 0:8 0:9 0:10 0:11 1:0 1:1

1:2 1:3 1:4 1:5 1:6 1:7 1:8 1:9

1:10 1:11

Tokens (hover to focus, click to lock)   source ← destination (destination attention when none selected) ⌄

<START> Mr and Mrs Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Mr Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large moustache. Mrs Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbours. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs Potter was Mrs Dursley's sister, but they hadn't met for several years; in fact, Mrs Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbours would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that.

The above diagram shows the *value-weighted attention pattern* for various attention heads; that is, the attention patterns with attention weights scaled by the norm of the value vector at the source position $||v_{src}^{h}||$. You can think of the value-weighted attention pattern as showing "how big a vector is moved from each position." (This approach was also recently introduced by Kobayashi *et al.* [16].) This is especially useful because attention heads will sometimes use certain tokens as a kind of default or resting position when there isn't a token that matches what they're looking for; the value vector at these default positions will be small, and so the value weighted pattern is more informative.

The interface allows one to isolate attention heads, shows the overall attention pattern, and allows one to explore the attention for individual tokens. Attention heads involved in K-composition are colored using the same scheme as above. We suggest trying to isolate these heads.

If you look carefully, you'll notice that the aqua colored "induction heads" often attend back to previous instances of the token which *will* come next. We'll investigate this more in the next section. Of course, looking at attention patterns on a single piece of text — especially a well-known paragraph like this one — can't give us very high confidence as to how these heads behave in generality. We'll return to this later, once we have a stronger hypothesis of what's going on.

Induction Heads

In small two-layer attention-only transformers, composition seems to be primarily used for one purpose: the creation of what we call induction heads. We previously saw that the one-layer model dedicated a lot of its capacity to copying heads, as a crude way to implement in-context learning. Induction heads are a much more powerful mechanism for achieving in-context learning.

FUNCTION OF INDUCTION HEADS

If you played around with the attention patterns above, you may have already guessed what induction heads do. Induction heads search over the context for previous examples of the present token. If they don't find it, they attend to the first token (in our case, a special token placed at the start), and do nothing. But if they do find it, they then look at the *next* token and copy it. This allows them to repeat previous sequences of tokens, both exactly and approximately.

It's useful to compare induction heads to the types of in-context learning we observed in one layer models:

- One layer model copying head: `[b] … [a] → [b]`

    - And when rare quirks of tokenization allow:
        `[ab] … [a] → [b]`
- Two layer model induction head: `[a][b] … [a] → [b]`

The two-layer algorithm is more powerful. Rather than generically looking for places it might be able to repeat a token, it knows how the token was previously used and looks out for similar cases. This allows it to make much more confident predictions in those cases. It's also less

to make much more confident predictions in those cases. It's also less vulnerable to distributional shift, since it doesn't depend on learned statistics about whether one token can plausibly follow another. (We'll see later that induction heads can operate on repeated sequences of completely random tokens)

The following examples highlight a few cases where induction heads help predict tokens in the first paragraph of Harry Potter:
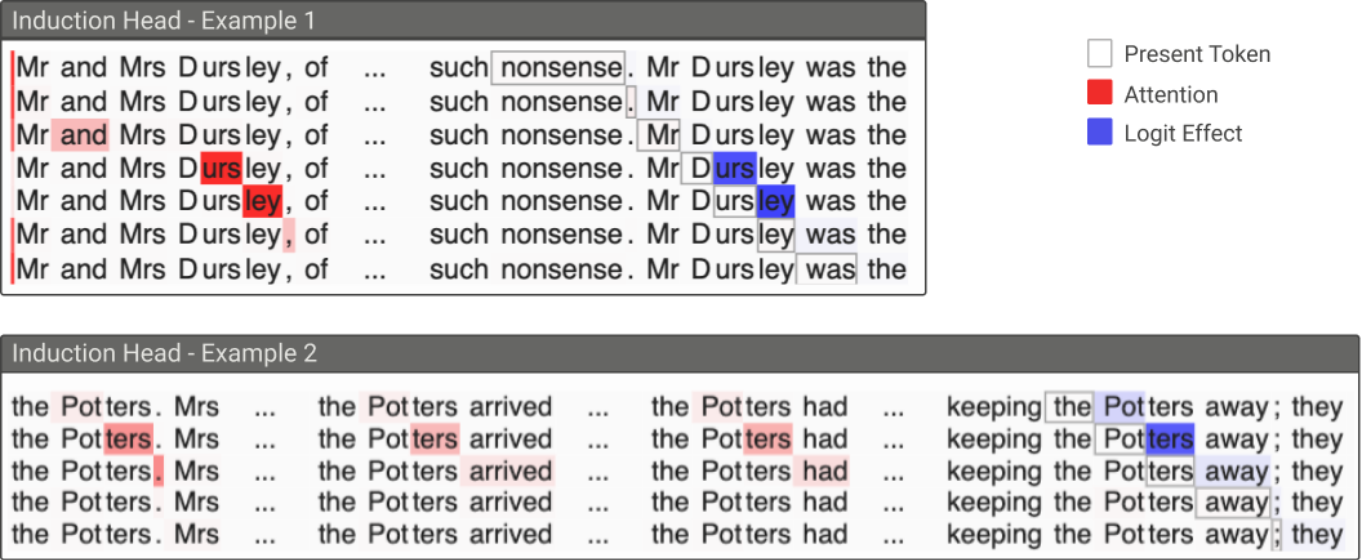

Raw attention pattern and logit effect for the induction head `1:8` on some segments of the first paragraph of *Harry Potter and the Philosopher's Stone*. The "logit effect" value shown is the effect of the result vector for the present token on the logit for the next token, $(W_U W_O^h r^h_{\text{pres\_tok}})_{\text{next\_tok}}$, which is equivalent to running the full OV circuit and inspecting the logit this head contributes to the next token.

Earlier, we promised to show induction heads on more tokens in order to better test our theory of them. We can now do this.

Given that we believe induction heads are attending to previous copies of the token and shifting forward, they should be able to do this on *totally random* repeated patterns. This is likely the hardest test one can give them, since they can't rely on normal statistics about which tokens typically come after other tokens. Since the tokens are uniformly sampled random tokens from our vocabulary, we represent the nth token in our vocabulary as `<n>`, with the exception of the special token `<START>`. (Notice that this is totally off distribution. Induction heads can operate on wildly different distributions as long as the more abstract property that repeated sequences are more like to reoccur holds true.)
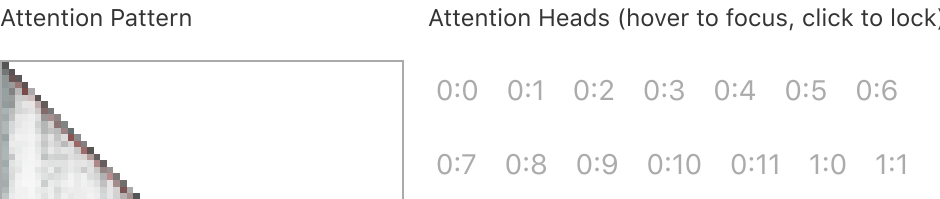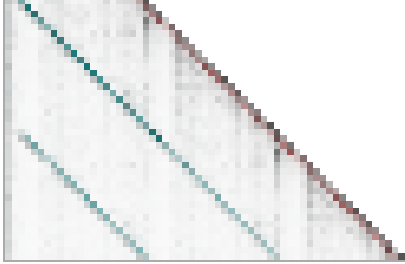
Attention Pattern   Attention Heads (hover to focus, click to lock)

0:0   0:1   0:2   0:3   0:4   0:5   0:6

0:7   0:8   0:9   0:10   0:11   1:0   1:1

Tokens (hover to focus, click to lock)   [source ← destination (destination attention when none selected) ⌄]

<START> <21549> <15231> <1396> <5012> <5767> <2493> <7192> <17468> <6215> <14329> <4652> <8828> <36539> <10496> <38416> <5550> <7671> <15553> <35140> <39966> <21549> <15231> <1396> <5012> <5767> <2493> <7192> <17468> <6215> <14329> <4652> <8828> <36539> <10496> <38416> <5550> <7671> <15553> <35140> <39966> <21549> <15231> <1396> <5012> <5767> <2493> <7192> <17468> <6215> <14329> <4652> <8828> <36539> <10496> <38416> <5550> <7671> <15553> <35140> <39966>

As in our previous attention pattern diagram, this diagram shows the value-weighted attention pattern for various heads, with each head involved in K-composition colored by our theory. Attention heads are shown acting on a random sequence of tokens, repeated three times. `<n>` denotes the nth token in our vocabulary.

This seems like pretty strong evidence that our hypothesis of induction heads is right. We now know what K-composition is used for in our two layer model. The question now is *how* K-composition accomplishes it.

HOW INDUCTION HEADS WORK

The central trick to induction heads is that the key is computed from tokens shifted one token back.[20] The query searches for "similar" key vectors, but because keys are shifted, finds the next token.

The following example, from a larger model with more sophisticated induction heads, is a useful illustration:

Mr and Mrs Dursley, of number ... with such nonsense. Mr Dursley was the
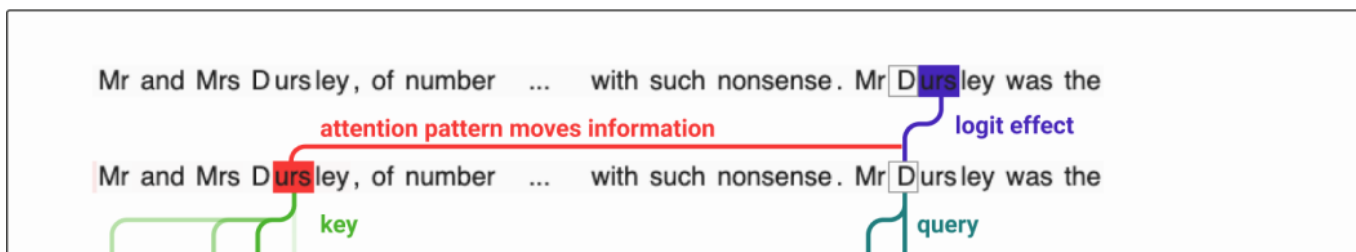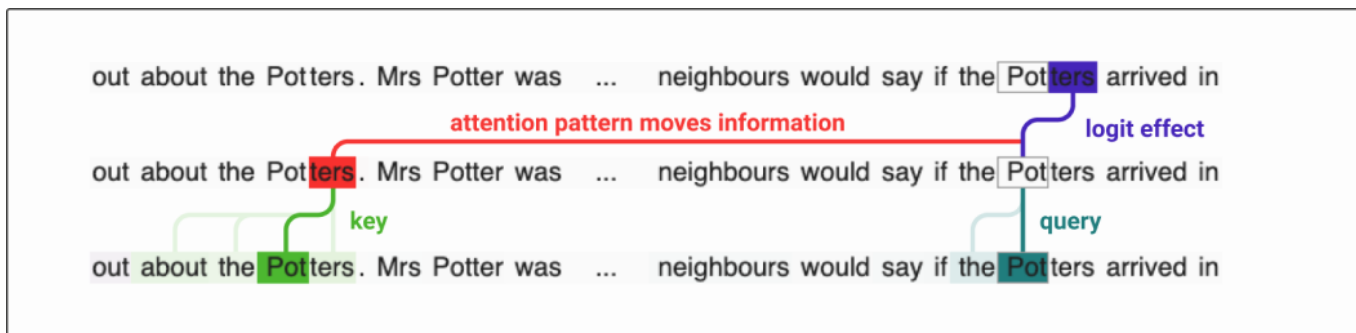
QK circuits can be expanded in terms of tokens instead of attention heads. Above, key and query intensity represent the amount each token increases the attention score. Logit effect is the OV circuit.

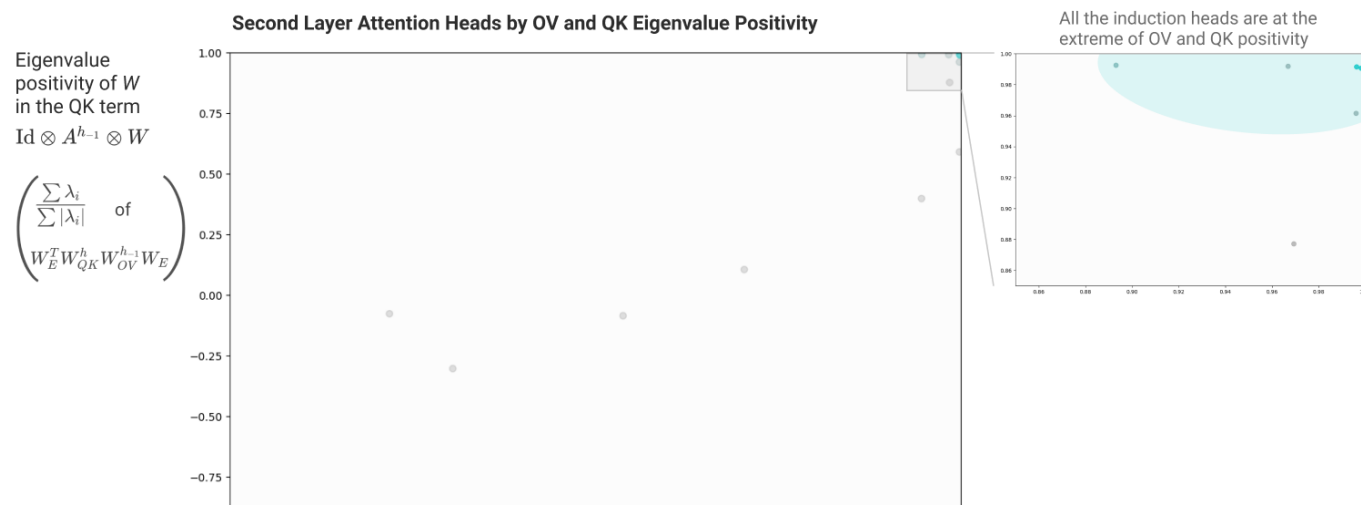The minimal way to create an induction head is to use K-composition with a previous token head to shift the key vector forward one token. This creates a term of the form $\mathrm{Id} \otimes A^{h-1} \otimes W$ in the QK-circuit (where A^{h-1} denotes an attention pattern attending to the previous token). If W matches cases where the tokens are the same — the QK version of a "copying matrix" — then this term will increase attention scores when the previous token before the source position is the same as the destination token. (Induction heads can be more complicated that this; for example, other 2-layer models will develop an attention head that attends a bit further than the previous token, presumably to create a $A^{h-1} \otimes A^{h-2} \otimes W$ term so some heads can match back further).

CHECKING THE MECHANISTIC THEORY

Our mechanistic theory suggestions that induction heads must do two things:

- Have a "copying" OV circuit matrix.

- Have a "same matching" QK circuit matrix associated with the $\mathrm{Id} \otimes A^{h-1} \otimes W$ term.

Although we're not confident that the eigenvalue summary statistic from the Detecting Copying section is the best possible summary statistic for detecting "copying" or "matching" matrices, we've chosen to use it as a working formalization. If we think of our attention heads as points in a 2D space of QK and OV eigenvalue positivity, all the induction heads turn out to be in an extreme right hand corner.

Second Layer Attention Heads by OV and QK Eigenvalue Positivity

Eigenvalue positivity of W in the QK term $\mathrm{Id} \otimes A^{h-1} \otimes W$

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

All the induction heads are at the extreme of OV and QK positivity
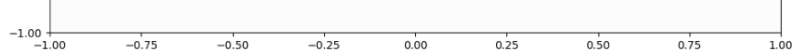
Eigenvalue
positivity of
the OV circuit $\left(\frac{\sum \lambda_i}{\sum |\lambda_i|} \text{ of } W_U W_{OV}^h W_E \right)$

One might wonder if this observation is circular. We originally looked at these attention heads because they had larger than random chance K-composition, and now we've come back to look, in part, at the K-composition term. But in this case, we're finding that the K-composition creates a matrix that is extremely biased towards positive eigenvalues — there wasn't any reason to suggest that a *large* K-composition would imply a *positive* K-composition. Nor any reason for all the OV circuits to be positive.

But that is exactly what we expect if the algorithm implemented is the described algorithm for implementing induction.

Term Importance Analysis

Earlier, we decided to ignore all the "virtual attention head" terms because we didn't observe any significant V-composition. While that seems like it's probably right, there's ways we could be mistaken. In particular, it could be the case that while every individual virtual attention head wasn't important, they matter in aggregate. This section will describe an approach to double checking this using ablations.

Ordinarily, when we ablate something in a neural network, we're ablating something that's explicitly represented in the activations. We can simply multiply it by zero and we're done. But in this case, we're trying to ablate an implicit term that only exists if you expand out the equations. We could do this by trying to run the version of a transformer described in our equations, but that would be horribly slow, and get exponentially worse as we considered deeper models.

But it turns out there's an algorithm which can determine the marginal effect of ablating the nth order terms (that is, the terms corresponding to paths through V-composition of n attention heads). The key trick is to run the model multiple times, replacing the present activations with activations from previous times you ran the model. This allows one to limit the depth of path, ablating all terms of order greater than that. Then by taking differences between the observed losses for each ablation, we can get the marginal effect of the nth order terms.
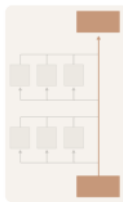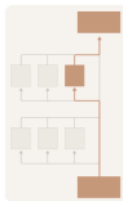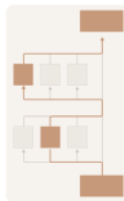
ALGORITHM FOR MEASURING MARGINAL LOSS REDUCTION OF NTH ORDER TERMS

> **Step 1:** Run model, save all attention patterns.
>
> **Step 2:** Run model, forcing all attention patterns to be the version you recorded, and instead of adding attention head outputs to residual stream, save the output, and then replace it with a zero tensor of the same shape. Record resulting loss.
>
> **Step n:** Run model, forcing all attention patterns to be the version you recorded, and instead of adding attention head outputs to residual stream, save the output, and replace it with the value you saved for this head last time. Record resulting loss.
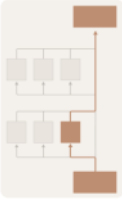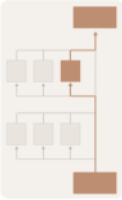
(Note that freezing the attention patterns to ground truth is what makes this ablation only target V-composition. Although this is in some ways the simplest algorithm, focused on the OV circuit, variants of this algorithm could also be used to isolate Q- or K-composition.)

As the V-composition results suggested, the second order "virtual attention head" terms have a pretty small marginal effect in this model. (Although they may very well be more important in other — especially larger — models.)

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)

We conclude that for understanding two-layer attention only models, we shouldn't prioritize understanding the second order "virtual attention heads" but instead focus on the direct path (which can only contribute to bigram statistics) and individual attention head terms. (We emphasize that this says nothing about Q- and K-Composition; higher order terms in the OV circuit not mattering only rules out V-Composition as important. Q- and K-Composition correspond to terms in the QK circuits of each head instead.)

We can further subdivide these individual attention head terms into those in layer 1 and layer 2:

Type	Example	Marginal Loss Reduction		Notes
Layer 1 Attention Heads		**- 0.05 nats** relative to direct path + layer 2	-0.004 nats/head	Relatively small effect, but keep in mind these heads also contribute to layer 2 QK circuits.
		- 1.3 nats relative to direct path only	-0.1 nats/head	
Layer 2 Attention Heads		**- 4.0 nats** relative to direct path + layer 1	-0.3 nats/head	**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 heards in their QK circuits.
		- 5.2 nats relative to direct path only	-0.4 nats/head	

This suggests we should focus on the second layer head terms.

Virtual Attention Heads

Although virtual heads turned out to be fairly unimportant for understanding the performance of the two layer model, we speculate that these may be much more important in larger and more complex transformers. We're also struck by them because they seem theoretically very elegant.

Recall that virtual attention heads were the terms of the form $(A^{h2}A^{h1}) \otimes (\ldots W_{OV}^{h2} W_{OV}^{h1} \ldots)$ in the path expansion of the logit equation, corresponding to the V-composition of two heads.

Where Q- and K-Composition affect the attention pattern, V-Composition creates these terms which really operate as a kind of independent unit which performs one head operation and then the other. This resulting object really is best thought of as the composition of the heads, $h_2 \circ h_1$. It has its own attention pattern, $A^{h2\circ h1} = A^{h2}A^{h1}$ and it's own OV matrix $W_{OV}^{h2\circ h1} = W_{OV}^{h2} W_{OV}^{h1}$. In deeper models, one could in principle have higher order virtual attention heads (e.g. $h_3 \circ h_2 \circ h_1$).

There are two things worth noting regarding virtual attention heads.

Firstly, this kind of composition seems quite powerful. We often see heads whose attention pattern attends to the previous token, but not heads who attend two tokens back - this may be because any useful

predictive power from the token two back is gained via virtual heads. Attention patterns can also implement more abstract things, such as attending to the start of the current clause, or the subject of the sentence - composition enables functions such as 'attend to the subject of the previous clause'.

Secondly, there are a *lot* of virtual attention heads. The number of normal heads grows linearly in the number of layers, while the number of virtual heads based on the composition of two heads grows quadratically, on three heads grows cubically, etc. This means the model may, in theory, have a lot more space to gain useful predictive power via the virtual attention heads. This is particularly important because normal attention heads are, in some sense, "large". The head has a single attention pattern determining which source tokens it attends to, and d_{head} dimensions to copy from the source to destination token. This makes it unwieldy to use for intuitively "small" tasks where not much information needs to be conveyed, eg attending to previous pronouns to determine whether the text is in first, second or third person, or attending to tense markers to detect whether the text is in past, present or future text.

Where Does This Leave Us?

Over the last few sections, we've made progress on understanding one-layer and two-layer attention-only transformers. But our ultimate goal is to understand transformers in general. Has this work actually brought us any closer? Do these special, limited cases actually shed light on the general problem? We'll explore this issue in follow-up work, but our general sense is that, yes, these methods can be used to understand *portions* of general transformers, including large language models.

One reason is that normal transformers contain some circuits which appear to be *primarily attentional*. Even in the presence of MLP layers, attention heads still operate on the residual stream and can still interact

directly with each other and with the embeddings. And in practice, we find instances of interpretable circuits involving only attention heads and the embeddings. Although we may not be able to understand the entire model, we're very well positioned to reverse engineer these portions.

In fact, we actually see some analogous attention heads and circuits in large models to those we analyzed in these toy models! In particular, we'll find that large models form many induction heads, and that the basic building block of their construction is K-composition with a previous token head, just as we saw here. This appears to be a central driver of in-context learning in language models of all sizes – a topic we'll discuss in our next paper.

That said, we can probably only understand a small portion of large language models this way. For one thing, MLP layers make up 2/3rds of a standard transformer's parameters. Clearly, there are large parts of a model's behaviors we won't understand without engaging with those parameters. And in fact the situation is likely worse: because many attention heads interact with the MLP layers, the fraction of parameters we can understand without considering MLP layers is even smaller than 1/3rd. More complete understanding will require progress on MLP layers. At a mechanistic level, their circuits actually have a very nice mathematical structure (see additional intuition). However, the clearest path forward would require individually interpretable neurons, which we've had limited success finding.

Ultimately, our goal in this initial paper is simply to establish a foothold for future efforts on this problem. Much future work remains to be done.

Related Work

CIRCUITS

The Distill *Circuits* thread [7] was a concerted effort to reverse engineer

The *Distill Circuits thread* [▸] was a concerted effort to reverse engineer the InceptionV1 model. Our work seeks to do something similar for large language models.

The Circuits approach needs to be significantly rethought in the context of language models. Attention heads are quite different from anything in conv nets and needed a new approach. The linear structure of the residual stream creates both new challenges (the lack of a privileged basis removes some options for studying it) but also creates opportunities (we can expand through it). Having circuits which are bilinear forms rather than just linear is also quite unusual (although it was touched on by Goh *et al.* [17] who investigated bilinear interactions between the image and language models).

We've noticed several interesting, high-level differences between the original circuits work on InceptionV1 and studying circuits in attention-only transformer language models:

- It's possible that circuit analysis of attention-only models scales very differently with model size. In an attention-only model, parameters are arranged in comparatively large, meaningful, largely linearly operating chunks corresponding to attention heads. This creates lots of opportunities to "roughly understand" quite large numbers of parameters. Even very large models only have a few thousand attention heads -- a scale where looking at every single one seems plausible. Of course, once one adds MLP layers, the majority of parameters are inside of them and this becomes a smaller relative win for understanding models.

- We've had a lot more success studying circuits in tiny attention-only transformers than we did with attempting to study circuits in small vision models. In small vision models, the problem is that neurons often aren't interpetable; nothing analogous seems to happen here, since we can reduce everything into end-to-end terms. However, perhaps when we study models with MLP layers (which can't be reduced into end-to-end terms) more closely, we'll also find that scale is needed to make neurons interpretable.

THE LOGIT LENS

Previous work by the LessWrong user Nostalgabrist on a method they call the "Logit Lens" [18] explores the same linear structure of the residual stream we heavily exploit. The logit lens approach notes that, since the residual stream is iteratively refined, one can apply the unembedding matrix to earlier stages of the residual stream (ie. essentially look at $W_U x_i$) to look at how model predictions evolve in some sense.

Our approach could be seen as making a similar observation, but

deciding that the residual stream isn't actually the fundamental object to study. Since it's the sum of linear projections of many attention heads and neurons, it's natural to simply multiply out the weights to look at how all the different parts contributing to the sum connect to the logits. One can then continue to exploit linear structure and try to push linearity as far back into the model as possible, which roughly brings one to our approach.

ATTENTION HEAD ANALYSIS

Our work follows the lead of several previous papers in exploring investigating transformer attention heads. Investigation of attention patterns likely began with visualizations by Llion Jones [19] and was quickly expanded on by others [20]. More recently, several papers have begun to seriously investigate correspondences between attention heads and grammatical structures [21, 22, 23].

The largest difference between these previous analyses of attention heads and our work really seems to be a matter of goals: we seek to provide an end-to-end mechanistic account, rather than empirically describe attention patterns. Of course, this preliminary paper is also different from this prior work in that we only studied very small toy models, and only really then to illustrate and support our theory. Finally, we focus on autoregressive transformers, rather than a denoising model like BERT [24].

Our investigations benefitted from these previous papers, and we have some miscellaneous thoughts on how our results relate to them:

- Like most of these papers (*e.g.* [21, 22]), we observe the existence of a previous token attention head in most models. Sometimes in small models we get an attention head that's more smeared out over the last two or three tokens instead.

- We mirror others findings that many attention heads appear to use punctuation or special tokens as a default (*e.g.* [20, 22]). Induction heads provide a concrete example of this. Similar to Kobayashi *et al.* [16] we find scaling attention patterns by the magnitude of value vectors to be very helpful for clarifying this.

- The toy models discussed in this paper do not exhibit the sophisticated grammatical attention heads described by some of this prior work. However, we do find more similar attention heads in larger models.

- Votia *et al.* [21] describe attention heads which preferentially attend to rare tokens; we wonder if those might be similar to the skip-trigram attention heads we describe.

- Several papers note the existence of attention heads which

attend to previous references to the present token. It occurs to us that what we call an "induction head" might appear like this in a bidirectional model trained on masked data rather than an autoregressive model (the mechanistic signature would be an $A^{hprev} \otimes A^{hprev} \otimes W$ in the QK expansion with large positive eigenvalues).

CRITICISM OF ATTENTION AS EXPLANATION

An important line of work critiques the naive interpretation of attention weights as describing how important a given token is in effecting the model's output (*see empirically* [25, 26]; *related conceptual discussion e.g.* [27, 28]; *but see* [29]).

Our framework might be thought of as offering — for the limited case of attention-only models — a typology of ways in which naive interpretation of attention patterns can be misleading, and a specific way in which they can be correct. When attention heads act in isolation, corresponding to the first order terms in our equations, they are indeed straightforwardly interpretable. (In fact, it's even better than that: as we saw for the one-layer model, we can also easily describe how these first order terms affect the logits!) However, there are three ways that attention heads can interact (Q-, K-, and V-Composition) to produce more complex behavior that doesn't map well to naive interpretation of the attention patterns (corresponding to an explosion of higher-order terms in the path expansion of the transformer). The question is how important these higher-order terms are, and we've observed cases where they seem very important!

Induction heads offer an object lesson in how naive interpretation of attention patterns can both be very informative and also misleading. On the one hand, the attention pattern of an induction head itself is very informative; in fact, for a non-trivial number of tokens, model behavior can be explained as "an induction head attended to this previous token and predicted it was going to reoccur." But the induction heads we found are totally reliant on a previous token head in the prior layer, through K-composition, in order to determine where to attend. Without understanding the K-composition effect, one would totally misunderstand the role of the previous token head, and also miss out on a deeper understanding of how the induction head decides where to attend. (This might also be a useful test case for thinking about gradient-based attribution methods; if an induction head confidently attends somewhere, its softmax will be saturated, leading to a small gradient on the key and thus low attribution to the corresponding token, obscuring the crucial role of the token preceding the one it attends to.)

BERTOLOGY GENERALLY

The research on attention heads mentioned above is often grouped in a larger body of work called "Bertology", which studies the internal representations of transformer language model representations, especially BERT [24]. Besides the analysis of attention heads, bertology research has several strands of inquiry, likely the largest is a line of work using probing methods to explore the linguistic properties at various stages of BERTs residual stream, referred to as embeddings in that literature. Unfortunately, we'd be unable to do justice to the full scope of Bertology here and instead refer readers to a fantastic review by Rogers *et al*. [30].

Our work in this paper primarily intersected with the attention head analysis aspects of Bertology for a few reasons, including our focus on attention-only models, our decision to avoid directly investigating the residual stream, and our focus on mechanistic over top-down probing approaches.

MATHEMATICAL FRAMEWORK

Our work leverages a number of mathematical observations about transformers to reverse engineer them. For the most part, these mathematical observations aren't in themselves novels. Many of them have been implicitly or explicitly noted by prior work. The most striking example of this is likely Dong *et al*. [31], who consider paths through a self-attention network in their analysis of the expressivity of transformers, deriving the same structure we find in our path expansion of logits. But there are many other examples. For instance, a recent paper by Shazeer *et al*. [32] include a "multi-way einsums" description of multi-headed attention, which might be seen as another expression of same tensor structure we've tried to highlight in attention heads. Even in cases where we aren't aware of papers observing mathematical structure we mention, we'd assume that they're known to some of the researchers deeply engaged in thinking about transformers. Instead, we think our contribution here is in leveraging this type of thinking to mechanistic interpretability of models.

OTHER INTERPRETABILITY DIRECTIONS

There are many additional approaches to neural network interpretability, including:

- Interpreting individual neurons (*in transformers* [33, 34]*; other LMs* [35, 36]*; vision* [37, 38, 39]*; but see* [40, 41])
- Influence functions ([42]*: but see* [43])

- Saliency maps (*e.g.* [44, 45, 46, 47, 48, 49, 50, 51]; *but see* [52, 53, 54])

- Feature visualization (*in LMs* [55, 56, 17]; *vision* [57, 58, 59, 44]; *tutorial* [60]; *but see* [61])

INTERPRETABILITY INTERFACES

It seems to us that interpretability research is deeply linked with visualizations and interactive interfaces supporting model exploration. Without visualizations one is forced to rely on summary statistics, and understanding something as complicated as a neural network in such a low-dimensional way is very limiting. The right interfaces can allow researchers to rapidly explore various kinds of high-dimensional structure: examine attention patterns, activations, model weights, and more. When used to ask the right questions, interfaces support both exploration and rigor.

Machine learning has a rich history of using visualizations and interactive interfaces to explore models (*e.g.* [62, 63, 64, 65, 66, 35]). This has continued in the context of Transformers (*e.g.* [67, 68, 69, 70]), especially with visualizations of attention (*e.g.* [20, 71, 72]).

RECENT ARCHITECTURAL CHANGES

Some recent proposed improvements to the transformer architecture have interesting interpretations from the perspective of our framework and findings:

- Primer [73] is a transformer architectured discovered through automated architecture search for more efficient transformer variants. The authors, So *et al.*, isolate two key changes, one of which is to perform a depthwise convolution over the last three spatial positions in computing keys, queries and value vectors. We observe that this change would make induction heads possible to express without K-composition.

- Talking Heads Attention [32] is a recent proposal which can be a bit tricky to understand. An alternative way to frame it is that it that, were regular transformer attention heads have $W_{OV}^h = W_O^h W_V^h$, talking heads attention effectively does $W_{OV}^h = \alpha_1^h W_O^1 W_V^1 + \alpha_2^h W_O^2 W_V^2 ...$, and the same thing for W_{QK}. This means that the OV and QK matrices of different attention heads can share components; if you believe that, say, multiple copying heads could share part of their OV matrix, this becomes natural.

Additional Resources

This article has a companion [problem set](#) of problems we found helpful in building our intuition for attention heads.

We've release one of the libraries we made for our research, [PySvelte](#) (which makes it convenient to use interactive visualizations from Python), and written an article describing another one called [Garcon](#) (our tooling for probing the internals of models, especially large ones).

Acknowledgments

Author Contributions

Theoretical Framework: The framework described in this paper developed as part of ongoing conversations and empirical investigation between Nelson Elhage, Catherine Olsson, Neel Nanda, and Chris Olah over the last year as they've tried to understand language models. It's impossible to disentangle all the contributions that contributed to its development, but some examples of the kind of contributions involved follow. Nelson Elhage discovered evidence of neurons performing memory management in the residual stream, informing the "residual stream as a communication channel" framing. Catherine Olsson carefully characterized dozens of neurons and attention heads (with other authors also helping out), greatly informing our overall thinking on how to think about different parts of the model. Neel Nanda greatly improved our thinking on how attention heads compose. Chris Olah developed the foundational ideas of the framework, discovered induction heads, and guided theoretical progress. These examples are only a small slice of the contributions each mentioned author made.

In addition to these core research contributors, other members of Anthropic contributed important insights to the framework. Jared Kaplan, Sam McCandlish,

contributed important insights to the framework. Jared Kaplan, Sam McCandlish, Andy Jones, and Dario Amodei contributed especially important insights.

Analysis of Toy Models: The analysis of toy models presented in this paper was done by Chris Olah and Neel Nanda. However, they were heavily based on previous experiments, including many by Nelson Elhage and Catherine Olsson. The special attention-only toy models used for this article were created by Nelson Elhage.

Writing: This article was drafted by Chris Olah. Neel Nanda made major pedagogical improvements to the draft. Dario Amodei contributed heavily to the high-level framing. Nelson Elhage did careful editing to improve clarity. Many members of Anthropic, including Nicholas Joseph, Zac Hatfield-Dodds and Jared Kaplan provided valuable feedback throughout the writing process. Others, including Deep Ganguli, Jack Clark, Ben Mann, Danny Hernandez, Liane Lovitt, and Tom Conerly significantly improved exposition by testing and giving feedback on approaches to explaining these ideas.

Infrastructure for Analyzing Models: The crucial piece of software infrastructure which made our analysis of models possible is a library called Garcon (described in another article) created by Nelson Elhage with help from Tom Brown, Sam McCandlish, and Chris Olah. Garcon makes it easy to access model activations and parameters, regardless of scale. Nelson and Catherine Olsson also built an internal website, based on top of Garcon, which allows users to see dataset examples as well as interactively edit text and see model activations and attention patterns. Although not presented here, this website was very important to the evolution of our thinking.

Visualizations: Having interactive visualizations to explore model activations was important in enabling us to explore large amounts of data. Chris Olah created PySvelte (see github) and many of our visualizations, with contributions form Nelson Elhage and Catherine Olsson. Conceptual illustrations of ideas in this article are by Chris Olah. Ben Mann set up infrastructure to allow convenient secure sharing of visualizations.

Model Training: All our interpretability research is enabled by having access to models to study, including large models. (Although the models used as examples in this particular paper are small, our framework was informed by experiments on large models.) Led by Tom Brown, Sam McCandlish, and Jared Kaplan, the majority of Anthropic's technical staff contributed to the development of our efficient distributed training infrastructure and the underlying machine learning. Core contributors include Nicholas Joseph, Tom Henighan, and Ben Mann. Nelson Elhage, Kamal Ndousse, Andy Jones, Zac Hatfield-Dodds, and Danny Hernandez also contributed to this infrastructure.

Cluster: Tom Henighan and Nova DasSarma, advised by Tom Brown and Sam McCandlish with contributions from many others at Anthropic, managed the research cluster our research depended on and maintained its stability. Nova provided important support when interpretability required unorthodox jobs to run on our cluster.

Other Contributions:

All staff at Anthropic provided valuable feedback throughout. Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish all provided essential high-level feedback.

Chris Olah led the project.

Additional Intuition and Observations

MLP Layers

This article has focused on attention-only transformers, without MLP layers. How can we extend this approach to understanding MLP models with MLP layers?

Recall that an MLP layer m computes it's activations a^m from the residual stream x_i by performing a matrix multiply and applying it's activation function. The activation vector is then projected back down and added into the residual stream:

$$x_{i+1} = x_i + W_O^m a^m$$

$$a^m = \mathrm{GeLU}(a_{\mathrm{pre}}^m)$$

$$a_{\mathrm{pre}}^m = W_I^m x_i$$
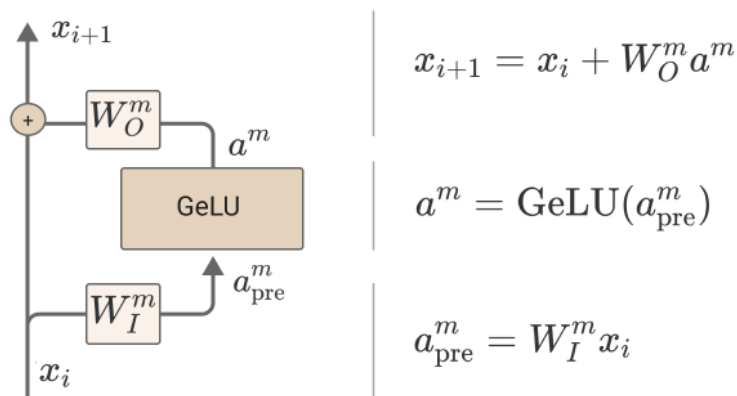
The presence of the GeLU activation means we can't linearize through MLP layers as we did attention layers. Instead, we likely need to take the approach of the Circuits project in reverse engineering vision models [7] : understanding what the neurons represent, how they're computed, and how they're used.

In theory, there's a lot of reason to be optimistic about understanding these neurons. They have an activation function which should encourage features to align with the basis dimensions. They're four times larger than the residual stream, and information doesn't need to flow through them, which are both factors one might expect to reduce polysemanticity. Unfortunately, things are much more challenging in practice. We've found the neurons much harder to develop hypotheses for, with the exception of neurons at ~5% depth through the model, which often respond to clusters of short phrases with similar meanings. We've focused this paper on understanding attention heads because we got a lot more early traction on attention heads.

Despite this, it's worth noting that there's a fairly clean story for how to mechanistically reason about neurons, which we can use when we find interpretable neurons.

Path Expansion of MLP Layer in One-Layer Model: For simplicity, let's consider a standard one layer transformer (ignoring layer norm and biases). The pre-activation values are a linear function of the residual stream. By applying our equation for attention heads, we get essentially the same equation we got for the one-layer transformer logits:

$$a_{\mathrm{pre}}^m = W_I^m \cdot \left(Id + \sum_{h \in H_1} A^h \otimes W_{OV}^h \right) \cdot W_E$$

$$a_{\text{pre}}^m = W_I^m W_E + \sum_{h \in H_1} A^h \otimes \left(W_I^m W_{OV}^h W_E \right)$$

This means we can study it with the same methods. $W_I^m W_E$ tells us how much different tokens encourage or inhibit this neuron through the residual stream, while $W_I^m W_{OV}^h W_E$ does the same for tokens if they're attended to by a given attention head. One might think of this as being similar to a neuron in a convolutional neural network, except where neuron weights in a conv net are indexed by relative position, these neuron weights are indexed by attention heads. The following section, Virtual Weights and Convolutional-Like Structure, will explore this connection in more detail.

What about the downstream effect of these neurons? $W_U W_O^m$ tells us how the activation of each neuron affects the logits. It's a simple linear function! (This is always true for the last MLP layer of a network, and gives one a lot of extra traction in understanding that layer.)

This approach becomes more difficult as a network gets deeper. One begins to need to reason about how one MLP layer affects another downstream of it. But the general circuits approach seems tractable, were it not for neurons being particularly tricky to understand, and there being enormous numbers of neurons.

Virtual Weights and Convolution-like Structure

When we apply path expansion to various terms in a transformer, we generally get virtual weights of the form:

$$y = \left(\text{Id} \otimes W_{\text{Id}} + \sum_h A^h \otimes W_h \right) x + \dots$$

(In the general case, h might include virtual attention heads.)

The main example we've seen in this paper is having y be logits for output tokens and x the one-hot encoded input tokens. But this can also arise in lots of other cases. For example, y might be MLP pre-activations and x might be the activations of a previous MLP layer.

Multiplying by $\left(\text{Id} \otimes W_{\text{Id}} + \sum_h A^h \otimes W_h \right)$ can be seen as a generalization of a convolution, with the weights $[W_{\text{Id}}, W_{h0}, W_{h1} \dots]$ and attention heads taking the place of relative position.

Exact Equivalence to Standard Convolution in Limited Cases: We claim that all convolutions can be expressed as one of the tensor products aboved, and thus (1) attention is a generalization of convolution; (2) specific configurations of attention correspond exactly to convolution.

Consider the convolution $W * x$. We claim that it can be rewritten as:

$$W * x = \sum_v A^v \otimes W_v$$

where v is the convolution offset, A^v is an attention pattern always attending to the relative position v (such as a previous token head), and W_v is the weight entry

the relative position v (such as a previous token head), and W_v is the weight entry corresponding to that offset.

This equivalence is explored in depth by Cordonnier *et al.* [74], who also empirically find that vision models often have <u>many 2D relative position heads</u>, similar to the previous token head we've observed.

Analogy to Convolution Generalizations: The correspondence to standard convolution breaks down for "dynamic" attention heads which attend based on some pattern other than fixed relative positions. However, it seems like there are many cases where they still are spiritually very convolution-like. For example, a "previous verb" head isn't a literal relative position (at least if we parameterize our input based on token index), but it seems morally like a type of convolution.

One way to resolve the above might be to consider graph convolutions. If we treat each attention pattern as a different set of weights on a graph, we could treat the tensor product as a sum of graph convolutions. (For an informal discussion of analogies between transformers and graph neural networks, readers may wish to look at an article by Joshi [75].) However, needing to consider multiple different weight graphs seems inelegant. There are other exotic kinds of convolutions based on abstract algebra (*see tutorial* [76]) and perhaps the right version could be found to find an exact correspondence.

But the fundamental thing is that when you see tensor products like the ones above, they can be thought of as something like a convolution with dynamic, soft, possibly semantic positions.

Why This is Useful: In reverse engineering convolutional neural networks, the Distill *Circuits* Thread [7] benefited greatly from the fact that transformer weights are organized into convolutions. For example, one can look at the weights between two neurons and see how different spatial positions affect things.

In the above example, we see the weights between two curve detectors; along the tangent of the curve, the other curve excites it. This would be much harder to understand if one had to look at 25 different weights; we benefit from organization.

We expect that similar organization may be helpful in understanding transformer weights, especially as we begin considering MLP layers.

Activation Properties

We often find it helpful to think different about various activations in transformers based on whether they have the following properties:

Privileged Basis vs Basis Free: A privileged basis occurs when some aspect of a model's architecture encourages neural network features to align with basis dimensions, for example because of a sparse activation function such as ReLU. In a transformer, the only vectors with privileged bases are tokens, attention

patterns and MLP activations. See the appendix table of variables for a full list of which activations have or don't have privileged activations.

Some types of interpretability only make sense for activations with a privileged basis. For example, it doesn't really make sense to look at the "neurons" (ie. basis dimensions) of activations like the residual stream, keys, queries or values, which don't have a privileged basis. This isn't to say that there aren't ways to study them; a lot of interesting work is done on word embeddings without assuming a privileged basis (*e.g.* [11]). But having a privileged basis does open up helpful approaches.

Bottleneck Activations: We say that an activation is a bottleneck activation if it is a lower-dimensional intermediate between two higher dimensional activations. For example, the residual stream is a bottleneck activation because it is the only way to pass information between MLP activations, which are typically four times larger than it. (Additionally, in addition to it being a bottleneck for the communication between adjacent MLP layers, it's also the only pathway by which arbitrary early MLP layers can communicate with arbitrary late MLP layers, so the stream may simultaneously be conveying different pieces of information between many different pairs of MLP layers, a much more extreme bottleneck than just 4x!) Similarly, a value vector is a bottleneck activation because it's much lower dimensional than the residual stream, and it's the only way to move information from the residual stream for one token position in the context to another (without the model dedicating an additional attention head to the same pattern, which it sometimes will). See above table for a list of which activations are or are not bottleneck activations.

Pointer Arithmetic with Positional Embeddings

Our models use a slightly unusual positional mechanism (similar to [14]) which doesn't put positional information into the residual stream. The popular rotary attention [13] approach to position has this same property. But it's worth noting that when models do have positional embeddings (as in [1]), several possibilities open up.

At their most basic level, positional embeddings are kind of like token addresses. Attention heads can use them to prefer to attend to tokens at certain relative positions, but they can also do much more:

- Transformers can do "pointer arithmetic" type operations on positional embeddings. We observed at least one case in GPT-2 where an induction head was implemented using this approach, instead of the one described above. First, an attention head attended to a similar token, returning it's positional embedding. Next, a query vector for another attention head was constructed with q-composition, rotating the positional embedding forward one token. This results in an induction head.

- We speculate that having an identifier for each token may be useful in some cases. For example, suppose you want all tokens in the same sentence to share an identifier. In a positional model, an attention head can attend to the previous period and copy its positional embedding into some subspace.

Identity Attention Heads?

It's worth noting that one could simplify all the math in this paper by introducing an "identity attention head," hId, such that $A^{h\text{Id}} = \text{Id}$ and $W_{OV}^{h\text{Id}} = \text{Id}$. This identity head would correspond to the residual stream, removing extra terms in the equations. In this paper, we chose not to do that and instead keep the residual stream as an explicit, differentiated term. The main reason for this is that it really does have different properties than attention heads. But in other cases, it can be useful to think about things the other way.

Notation

Variable Definitions

Main Model Activations and Parameters

Variable	Shape / Type	Description
$T(t)$	$[n_{\text{context}}, n_{\text{vocab}}]$	Transformer logits, for tokens t [activation, privileged basis]
t	$[n_{\text{context}}, n_{\text{vocab}}]$	One-hot encoded tokens [activation, privileged basis]
x^n	$[n_{\text{context}}, d_{\text{model}}]$	"Residual stream" or "embedding" vectors of model at layer n (one vector per context token) [activation, not privileged basis]
W_E	$[d_{\text{model}}, n_{\text{vocab}}]$	Token embedding [parameter]
W_P	$[d_{\text{model}}, n_{\text{context}}]$	Positional embedding [parameter]
W_U	$[n_{\text{vocab}}, d_{\text{model}}]$	Unembedding / softmax weights [parameter]

Superscripts and subscripts for variables may be elided when they aren't needed for clarity. For example, if we're only discussing a single attention head, one can write A instead of A^h.

Attention Heads Activations and Parameters

Variable	Shape / Type	Description
H_n	Set	Set of attention heads at layer n
$h(x)$	$[n_{\text{context}}, d_{\text{model}}]$	Output of attention head h [activation, not privileged basis]
A^h	$[n_{\text{context}}, n_{\text{context}}]$	Attention pattern of attention head h [activation, privileged basis]
q^h, k^h, v^h, r^h	$[n_{\text{context}}, d_{\text{head}}]$	Query, key, value and result vectors of attention head h (one vector per context token) [activation, not privileged basis]
W_Q^h, W_K^h, W_V^h	$[d_{\text{head}}, d_{\text{model}}]$	Query, key, and value weights of attention head h [parameter]
W_O^h	$[d_{\text{model}}, d_{\text{head}}]$	Output weights of attention head h [parameter]
W_{OV}^h	$[d_{\text{model}}, d_{\text{model}}]$	$W_{OV}^h = W_O^h W_V^h$ [parameter, low-rank]
W_{QK}^h	$[d_{\text{model}}, d_{\text{model}}]$	$W_{QK}^h = W_Q^{h\,T} W_K^h$ [parameter, low-rank]

MLP Layer Activations and Parameters

Variable	Shape / Type	Description
$m(x)$	$[n_{\text{context}}, d_{\text{model}}]$	Output of MLP layer m [activation, not privileged basis]
a^m	$[n_{\text{context}}, d_{\text{mlp}}]$	Activations of MLP layer m [activation, privileged basis]
W_I^m	$[d_{\text{mlp}}, d_{\text{model}}]$	Input weights for MLP layer m [parameter]
W_O^m	$[d_{\text{model}}, d_{\text{mlp}}]$	Output weights for MLP layer m [parameter]

Functions

Variable	Shape / Type	Description
$\text{GeLU}()$	Function	Gaussian Error Linear Units [77]
$\text{softmax}^*()$	Function	Softmax with autoregressive mask for attention distribution creation.

Tensor Product / Kronecker Product Notation

In machine learning, we often deal with matrices and tensors where we want to multiply "one side". In transformers specifically, our activations are often 2D arrays representing vectors at different context indices, and we often want to multiply "per position" or "across positions". Frequently, we want to do both! Tensor (or equivalently, Kronecker) products are a really clean way to denote this. We denote them with the $\otimes$ symbol.

Very pragmatically:

- A product like $\text{Id} \otimes W$ (with identity on the left) represents multiplying each position in our context by a matrix.

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

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

There are several completely equivalent ways to interpret these products. If the symbol is unfamiliar, you can pick whichever you feel most comfortable with:

- **Left-right multiplying:** Multiplying x by a tensor product $A \otimes W$ is equivalent to simultaneously left and right multiplying: $(A \otimes W)x = AxW^T$. When we add them, it is equivalent to adding the results of this multiplication: $(A_1 \otimes W_1 + A_2 \otimes W_2)x$ $= A_1 x W_1^T + A_2 x W_2^T$.

- **Kronecker product:** The operations we want to perform are linear

transformations on a flattened ("<u>vectorized</u>") version of the activation matrix x. But flattening gives us a huge vector, and we need to map our matrices to a much larger block matrix that performs operations like "multiply the elements which previously corresponded to a vector by this matrix". The correct operation to do this is the <u>Kronecker product</u>. So we can interpret $\otimes$ as a Kronecker product acting on the vectorization of x, and everything works out equivalently.

- **Tensor Product:** $A \otimes W$ can be interpreted as a <u>tensor product</u> turning the matrices A and W into a 4D tensor. In NumPy notation, it is equivalent to `A[:,:,None,None] * W[None, None, :, :]` (although one wouldn't computationally represent them in that form). More formally, A and W are "type $(1,1)$" tensors (matrices mapping vectors to vectors), and $A \otimes W$ is a "type $(2,2)$" tensor (which can map matrices to matrices).

Technical Details

Model Details

The models used as examples in this paper are zero, one, and two layer decoder-only, attention-only transformers [1]. For all models, $d_{\text{model}} = n_{\text{heads}} * d_{\text{head}}$, typically with $n_{\text{heads}} = 12$ and $d_{\text{head}} = 64$, but also with one explicitly noted example where $n_{\text{heads}} = 32$ and $d_{\text{head}} = 128$.

Models have a context size of 2048 tokens and use dense attention. (Dense attention was preferred over sparse attention for simplicity, but made a smaller context perferrable.) We use a positional mechanism similar to Press *et al.* [14], adding sinusoidal embeddings immediately before multiplying by W_Q and W_K to produce queries and keys. (This excludes <u>pointer-arithmetic based algorithms</u> without the distorted QK matrices like rotary.)

The training dataset is as described in Kaplan *et al.* [78].

Handling Layer Normalization

Our transformer models apply layer normalization every time they read off from the residual stream. In practice, this is done to encourage healthy activation and gradient scales, making transformers easier to train. However, it also means that the "read" operation of each layer is more complicated than the idealized version we've described above. In this section, we describe how we work around this issue.

Before considering layer normalization, let's consider the simpler case of batch normalization. Transformers typically use layer normalization over batch normalization due to subtleties around the use of batch normalization in autoregressive models, but it's an easy case to work out and would make our theory exactly align with the model. Batch normalization tracks a moving estimate of the mean and variance of its inputs, and uses that to independently normalize them, before rescaling with learned parameters. At inference time, a fixed estimate of the mean and variance is used. This means that it's simply a fixed linear transformation and bias, which can be multiplied into the adjacent learned linear transformation and bias. As such, batch normalization in models can typically be absorbed into other operations and ignored for the purposes of interpretability on the inference time model.

Layer normalization is a little more subtle. For each residual stream vector, we subtract off the average activation, normalize by variance, and then multiply by a learned set of diagonal weights and add a learned bias vector. It turns out that subtracting off the average activation is a fixed linear transformation, since it just zeros out a single dimension in the vector space. This means that everything except for normalizing by variance, layer normalization applies a fixed affine transformation. Normalizing by variance multiplies the vector by a scalar, and multiplying by a scalar commutes with all the other operations in a path. As a result, we can fold everything but normalization into adjacent parameters, and then think of the normalization scaling as a variable reweighting of the set of path terms going through that layer normalization. (The main downside of this is that it means that paths going through different sets of layers aren't easily comparable; for example, we can't trivially compare the importance of virtual attention heads composed of heads from different layers.)

A final observation is that layer normalization in the first attention layer of the model can alternatively be applied to every vector of the embedding matrix and then ignored. In some cases, this is more convenient (for example, we do this for skip-trigrams in our one-layer models).

In the future, it might be worth trying to train models with batch normalization to avoid this as a pain point, despite the complexities it presents.

Working with Low-Rank Matrices

In this paper, we find ourselves dealing with very large, but extremely low-rank matrices. Picking the right algorithm is often the difference between painfully slow operations on the GPU, and everything being instantaneous on your CPU.

First, we recommend keeping matrices in factored form whenever possible. When multiplying a chain of matrices, identify the "smallest" bottleneck point and represent the matrix as a product AB split at that bottleneck.

Eigenvalues: Exploit the fact that $\lambda_i(AB) = \lambda_i(BA)$ [79]. Calculating eigenvalues of a 64x64 matrix is greatly preferable to a 50,000x50,000 matrix.

SVD: The SVD can also be calculated efficiently, and a variant can be used to efficiently compute PCA:

- Compute the SVDs of A and B: $U_A S_A V_A = A$ and $U_B S_B V_B = B$.

- Define $C = S_A V_A U_B S_B$

- SVD C: $U_C S_C V_C = C$

- SVD of AB is $U_{AB} = U_A U_C$, $S_{AB} = S_C$, $V_{AB} = V_C V_B$.

Footnotes

1. We'll explore induction heads in much more detail in a forthcoming paper. [↵]

2. Constructing models with a residual stream traces back to early work by the Schmidhuber group, such as highway networks [8] and LSTMs [9], which have found significant modern success in the more recent residual network architecture [10]. In transformers, the residual stream vectors are often called the "embedding." We prefer the residual stream terminology, both because it emphasizes the residual nature (which we believe to be important) and also

because we believe the residual stream often dedicates subspaces to tokens other than the present token, breaking the intuitions the embedding terminology suggests. [↵]

3. It's worth noting that the completely linear residual stream is very unusual among neural network architectures: even ResNets [10], the most similar architecture in widespread use, have non-linear activation functions on their residual stream, or applied whenever the residual stream is accessed! [↵]

4. This ignores the layer normalization at the start of each layer, but up to a constant scalar, the layer normalization is a constant affine transformation and can be folded into the linear transformation. See discussion of how we handle layer normalization in the appendix. [↵]

5. Note that for attention layers, there are three different kinds of input weights: W_Q, W_K, and W_V. For simplicity and generality, we think of layers as just having input and output weights here. [↵]

6. We performed PCA analysis of token embeddings and unembeddings. For models with large d_{model}, the spectrum quickly decayed, with the embeddings/unembeddings being concentrated in a relatively small fraction of the overall dimensions. To get a sense for whether they occupied the same or different subspaces, we concatenated normalized embedding and embedding and applied PCA. This joint PCA process showed a combination of both "mixed" dimensions and dimensions used only by one; the existence of dimensions which are used by only one might be seen as a kind of upper bound on the extent to which they use the same subspace. [↵]

7. Some MLP neurons have very negative cosine similarity between their input and output weights, which may indicate deleting information from the residual stream. Similarly, some attention heads have large negative eigenvalues in their $W_O W_V$ matrix and primarily attend to the present token, potentially serving as a mechanism to delete information. It's worth noticing that while these may be generic mechanisms for "memory management" deletion of information, they may also be mechanisms for conditionally deleting information, operating only in some cases. [↵]

8. As discussed above, often multiplication by the output matrix is written as one matrix multiply applied to the concatenated results of all heads; however this version is equivalent. [↵]

9. What do we mean when we say that $W_{OV} = W_O W_V$ governs which subspace of the residual stream the attention head reads and writes to when it moves information? It can be helpful to consider the singular value decomposition $USV = W_{OV}$. Since $d_{head} < d_{model}$, W_{OV} is low-rank and only a subset of the diagonal entries in S are non-zero. The right singular vectors V describe which subspace of the residual stream being attended to is "read in" (somehow stored as a value vector), while the left singular vectors U describe what subspace of the destination residual stream they are written to. [↵]

10. This parallels an observation by Levy & Goldberg, 2014 that many early word embeddings can be seen as matrix factorizations of a log-likelihood matrix. [↵]

11. An interesting corollary of this is to note is that, though W_U is often referred to as the "un-embedding" matrix, we should *not* expect this to be the inverse of embedding with W_E. [↵]

12. Our use of the term "skip-trigram" to describe sequences of the form "A... BC"

is inspired by Mikolov *et al.* [11]'s use of the term "skip-gram" in their classic paper on word embeddings. [↵]

13. Technically, it is a function of all possible source tokens from the start to the destination token, as the softmax calculates the score for each via the QK circuit, exponentiates and then normalises [↵]

14. In models with more than one layer, we'll see that the QK circuit can be more complicated than $W_E^T W_{QK}^h W_E$. [↵]

15. How can a one layer model learn an attention head that attends to a relative position? For a position mechanism that explicitly encodes relative position like rotary [13] the answer is straightforward. However, we use a mechanism similar to [14] (and, for the purposes of this point, [1]) where each token index has a position embedding that affects keys and queries. Let's assume that the embeddings are either fixed to be sinusoidal, or the model learns to make them sinusoidal. Observe that, in such an embedding, translation is equivalent to multiplication by a rotation matrix. Then W_{QK} can select for any relative positional offset by appropriately rotating the dimensions containing sinusoidal information. [↵]

16. Before token embedding, we think of tokens as being one-hot vectors in a very high-dimensional space. Logits are also vectors. As a result, we can think about linear combinations of tokens in both spaces. [↵]

17. The most similar class of random matrix for which eigenvalues are well characterized is likely Ginibre matrices, which have Gaussian-distributed entries similar to our neural network matrices at initialization. Real valued Ginibre matrices are known to have positive-negative symmetric eigenvalues, with extra probability mass on the real numbers, and "repulsion" near them [15]. Of course, in practice we are dealing with products of matrices, but empirically the distribution of eigenvalues for the OV circuit with our randomly initialized weights appears to mirror the Ginibre distribution. [↵]

18. Non-orthogonal eigenvectors can have unintuitive properties. If one tries to express a matrix in terms of eigenvectors, one needs to multiply by the inverse of the eigenvector matrix, which can behave quite differently than naively projecting onto the eigenvectors in the non-orthogonal case. [↵]

19. There appears to be no significant V- or Q- composition in this particular model. [↵]

20. For models with position embeddings which are available in the residual stream (unlike rotary attention), a second algorithm for implementing induction heads is available; see our intuitions around position embeddings and pointer arithmetic algorithms in transformers. [↵]

References

1. Attention is all you need   [PDF]
   Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Advances in neural information processing systems, pp. 5998--6008.

2. Language models are few-shot learners   [PDF]
   Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. and others,, 2020. arXiv preprint arXiv:2005.14165.

3. LaMDA: our breakthrough conversation technology   [link]
   Collins, E. and Ghahramani, Z., 2021.

4. Evaluating large language models trained on code
   Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G. and others,, 2021. arXiv preprint arXiv:2107.03374.

5. Towards a human-like open-domain chatbot
   Adiwardana, D., Luong, M., So, D.R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y. and others,, 2020. arXiv preprint arXiv:2001.09977.

6. Scaling Language Models: Methods, Analysis & Insights from Training Gopher   [PDF]
   Rae, J.W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., Driessche, G.v.d., Hendricks, L.A., Rauh, M., Huang, P., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X.L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T., Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., d'Autume, C.d.M., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., Casas, D.d.L., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K. and Irving, G., 2021. Preprint.

7. Thread: Circuits
   Cammarata, N., Carter, S., Goh, G., Olah, C., Petrov, M., Schubert, L., Voss, C., Egan, B. and Lim, S.K., 2020. Distill. DOI: 10.23915/distill.00024

8. Highway networks   [PDF]
   Srivastava, R.K., Greff, K. and Schmidhuber, J., 2015. arXiv preprint arXiv:1505.00387.

9. Long short-term memory   [link]
   Hochreiter, S. and Schmidhuber, J., 1997. Neural computation, Vol 9(8), pp. 1735--1780. MIT Press.

10. Deep residual learning for image recognition   [PDF]
    He, K., Zhang, X., Ren, S. and Sun, J., 2016. Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770--778.

11. Linguistic regularities in continuous space word representations   [PDF]
    Mikolov, T., Yih, W. and Zweig, G., 2013. Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies, pp. 746--751.

12. Visualizing Weights   [link]
    Voss, C., Cammarata, N., Goh, G., Petrov, M., Schubert, L., Egan, B., Lim, S.K. and Olah, C., 2021. Distill. DOI: 10.23915/distill.00024.007

13. Roformer: Enhanced transformer with rotary position embedding
    Su, J., Lu, Y., Pan, S., Wen, B. and Liu, Y., 2021. arXiv preprint arXiv:2104.09864.

14. Shortformer: Better language modeling using shorter inputs

Press, O., Smith, N.A. and Lewis, M., 2020. arXiv preprint arXiv:2012.15832.

15. Real spectra of large real asymmetric random matrices   [PDF]
    Tarnowski, W., 2021. arXiv preprint arXiv:2104.02584.

16. Attention is not only a weight: Analyzing transformers with vector norms
    Kobayashi, G., Kuribayashi, T., Yokoi, S. and Inui, K., 2020. arXiv preprint
    arXiv:2004.10102.

17. Multimodal Neurons in Artificial Neural Networks
    Goh, G., Cammarata, N., Voss, C., Carter, S., Petrov, M., Schubert, L., Radford,
    A. and Olah, C., 2021. Distill. DOI: 10.23915/distill.00030

18. interpreting GPT: the logit len   [link]
    nostalgebraist,, 2020.

19. Tensor2tensor transformer visualization   [link]
    Jones, L., 2017.

20. A multiscale visualization of attention in the transformer model   [PDF]
    Vig, J., 2019. arXiv preprint arXiv:1906.05714.

21. Analyzing multi-head self-attention: Specialized heads do the heavy lifting,
    the rest can be pruned   [PDF]
    Voita, E., Talbot, D., Moiseev, F., Sennrich, R. and Titov, I., 2019. arXiv preprint
    arXiv:1905.09418.

22. What does bert look at? an analysis of bert's attention   [PDF]
    Clark, K., Khandelwal, U., Levy, O. and Manning, C.D., 2019. arXiv preprint
    arXiv:1906.04341.

23. Do attention heads in bert track syntactic dependencies?
    Htut, P.M., Phang, J., Bordia, S. and Bowman, S.R., 2019. arXiv preprint
    arXiv:1911.12246.

24. Bert: Pre-training of deep bidirectional transformers for language
    understanding
    Devlin, J., Chang, M., Lee, K. and Toutanova, K., 2018. arXiv preprint
    arXiv:1810.04805.

25. Attention is not explanation   [PDF]
    Jain, S. and Wallace, B.C., 2019. arXiv preprint arXiv:1902.10186.

26. Is attention interpretable?   [PDF]
    Serrano, S. and Smith, N.A., 2019. arXiv preprint arXiv:1906.03731.

27. On identifiability in transformers
    Brunner, G., Liu, Y., Pascual, D., Richter, O., Ciaramita, M. and Wattenhofer, R.,
    2019. arXiv preprint arXiv:1908.04211.

28. Quantifying attention flow in transformers
    Abnar, S. and Zuidema, W., 2020. arXiv preprint arXiv:2005.00928.

29. Attention is not not explanation   [PDF]
    Wiegreffe, S. and Pinter, Y., 2019. arXiv preprint arXiv:1908.04626.

30. A primer in bertology: What we know about how bert works
    Rogers, A., Kovaleva, O. and Rumshisky, A., 2020. Transactions of the
    Association for Computational Linguistics, Vol 8, pp. 842--866. MIT Press.

31. Attention is not all you need: Pure attention loses rank doubly exponentially
    with depth   [PDF]

Dong, Y., Cordonnier, J. and Loukas, A., 2021. arXiv preprint arXiv:2103.03404.

32. Talking-heads attention   [PDF]
Shazeer, N., Lan, Z., Cheng, Y., Ding, N. and Hou, L., 2020. arXiv preprint arXiv:2003.02436.

33. Knowledge neurons in pretrained transformers
Dai, D., Dong, L., Hao, Y., Sui, Z. and Wei, F., 2021. arXiv preprint arXiv:2104.08696.

34. Transformer feed-forward layers are key-value memories
Geva, M., Schuster, R., Berant, J. and Levy, O., 2020. arXiv preprint arXiv:2012.14913.

35. Visualizing and understanding recurrent networks   [PDF]
Karpathy, A., Johnson, J. and Fei-Fei, L., 2015. arXiv preprint arXiv:1506.02078.

36. Learning to generate reviews and discovering sentiment   [PDF]
Radford, A., Jozefowicz, R. and Sutskever, I., 2017. arXiv preprint arXiv:1704.01444.

37. Curve Detectors
Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M. and Olah, C., 2020. Distill. DOI: 10.23915/distill.00024.003

38. Object detectors emerge in deep scene cnns   [PDF]
Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. and Torralba, A., 2014. arXiv preprint arXiv:1412.6856.

39. Network Dissection: Quantifying Interpretability of Deep Visual Representations   [PDF]
Bau, D., Zhou, B., Khosla, A., Oliva, A. and Torralba, A., 2017. Computer Vision and Pattern Recognition.

40. On the importance of single directions for generalization   [PDF]
Morcos, A.S., Barrett, D.G., Rabinowitz, N.C. and Botvinick, M., 2018. arXiv preprint arXiv:1803.06959.

41. On Interpretability and Feature Representations: An Analysis of the Sentiment Neuron
Donnelly, J. and Roegiest, A., 2019. European Conference on Information Retrieval, pp. 795--802.

42. Understanding Black-box Predictions via Influence Functions   [PDF]
Koh, P.W. and Liang, P., 2017. International Conference on Machine Learning (ICML).

43. Influence functions in deep learning are fragile   [PDF]
Basu, S., Pope, P. and Feizi, S., 2020. arXiv preprint arXiv:2006.14651.

44. Deep inside convolutional networks: Visualising image classification models and saliency maps   [PDF]
Simonyan, K., Vedaldi, A. and Zisserman, A., 2013. arXiv preprint arXiv:1312.6034.

45. Visualizing and understanding convolutional networks   [PDF]
Zeiler, M.D. and Fergus, R., 2014. European conference on computer vision, pp. 818--833.

46. Striving for simplicity: The all convolutional net   [PDF]
Springenberg, J.T., Dosovitskiy, A., Brox, T. and Riedmiller, M., 2014. arXiv preprint arXiv:1412.6806.

47. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization   [PDF]
Selvaraju, R.R., Das, A., Vedantam, R., Cogswell, M., Parikh, D. and Batra, D., 2016. arXiv preprint arXiv:1610.02391.

48. Interpretable Explanations of Black Boxes by Meaningful Perturbation   [PDF]
Fong, R. and Vedaldi, A., 2017. arXiv preprint arXiv:1704.03296.

49. PatternNet and PatternLRP--Improving the interpretability of neural networks   [PDF]
Kindermans, P., Sch{"u}tt, K.T., Alber, M., M{"u}ller, K. and D{"a}hne, S., 2017. arXiv preprint arXiv:1705.05598. DOI: 10.1007/978-3-319-10590-1_53

50. Axiomatic attribution for deep networks   [PDF]
Sundararajan, M., Taly, A. and Yan, Q., 2017. arXiv preprint arXiv:1703.01365.

51. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation
Ancona, M., Oztireli, C. and Gross, M., 2019. International Conference on Machine Learning, pp. 272--281.

52. Sanity checks for saliency maps
Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M. and Kim, B., 2018. arXiv preprint arXiv:1810.03292.

53. The (un) reliability of saliency methods   [PDF]
Kindermans, P., Hooker, S., Adebayo, J., Alber, M., Schutt, K.T., Dahne, S., Erhan, D. and Kim, B., 2019. Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pp. 267--280. Springer.

54. Investigating sanity checks for saliency maps with image and text classification
Kokhlikyan, N., Miglani, V., Alsallakh, B., Martin, M. and Reblitz-Richardson, O., 2021. arXiv preprint arXiv:2106.07475.

55. Interpretable textual neuron representations for NLP
Poerner, N., Roth, B. and Sch{"u}tze, H., 2018. arXiv preprint arXiv:1809.07291.

56. What does BERT dream of?   [link]
Bauerle, A. and Wexler, J., 2020.

57. Visualizing higher-layer features of a deep network   [PDF]
Erhan, D., Bengio, Y., Courville, A. and Vincent, P., 2009. University of Montreal, Vol 1341, pp. 3.

58. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images   [PDF]
Nguyen, A., Yosinski, J. and Clune, J., 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 427--436. DOI: 10.1109/cvpr.2015.7298640

59. Inceptionism: Going deeper into neural networks   [HTML]
Mordvintsev, A., Olah, C. and Tyka, M., 2015. Google Research Blog.

60. Feature Visualization   [link]
Olah, C., Mordvintsev, A. and Schubert, L., 2017. Distill. DOI: 10.23915/distill.00007

61. How Well do Feature Visualizations Support Causal Understanding of CNN Activations?
Zimmermann, R., Borowski, J., Geirhos, R., Bethge, M., Wallis, T. and Brendel

Zimmermann, R., Borowski, J., Geirhos, R., Bethge, M., Wallis, T. and Brendel, W., 2021. Advances in Neural Information Processing Systems, Vol 34.

62. TensorFlow Playground   [link]
Smilkov, D., Carter, S., Sculley, D., Vi{'e}gas, F.B. and Wattenberg, M., 2017.

63. Activation Atlas   [link]
Carter, S., Armstrong, Z., Schubert, L., Johnson, I. and Olah, C., 2019. Distill.
DOI: 10.23915/distill.00015

64. The Building Blocks of Interpretability   [link]
Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. and Mordvintsev, A., 2018. Distill. DOI: 10.23915/distill.00010

65. Experiments in Handwriting with a Neural Network   [link]
Carter, S., Ha, D., Johnson, I. and Olah, C., 2016. Distill. DOI: 10.23915/distill.00004

66. Understanding neural networks through deep visualization
Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. and Lipson, H., 2015. arXiv preprint arXiv:1506.06579.

67. Interfaces for Explaining Transformer Language Models   [link]
Alammar, J., 2020.

68. The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models
Tenney, I., Wexler, J., Bastings, J., Bolukbasi, T., Coenen, A., Gehrmann, S., Jiang, E., Pushkarna, M., Radebaugh, C., Reif, E. and others,, 2020. arXiv preprint arXiv:2008.05122.

69. Visbert: Hidden-state visualizations for transformers
Aken, B.v., Winter, B., L{"o}ser, A. and Gers, F.A., 2020. Companion Proceedings of the Web Conference 2020, pp. 207--211.

70. What Have Language Models Learned?   [link]
Pearce, A., 2021.

71. exbert: A visual analysis tool to explore learned representations in transformers models
Hoover, B., Strobelt, H. and Gehrmann, S., 2019. arXiv preprint arXiv:1910.05276.

72. Dodrio: Exploring Transformer Models with Interactive Visualization
Wang, Z.J., Turko, R. and Chau, D.H., 2021. arXiv preprint arXiv:2103.14625.

73. Primer: Searching for efficient transformers for language modeling
So, D.R., Ma{'n}ke, W., Liu, H., Dai, Z., Shazeer, N. and Le, Q.V., 2021. arXiv preprint arXiv:2109.08668.

74. On the relationship between self-attention and convolutional layers
Cordonnier, J., Loukas, A. and Jaggi, M., 2019. arXiv preprint arXiv:1911.03584.

75. Transformers are Graph Neural Networks   [link]
Joshi, C., 2020. The Gradient.

76. Groups & Group Convolutions   [link]
Olah, C..

77. Gaussian error linear units (gelus)   [PDF]
Hendrycks, D. and Gimpel, K., 2016. arXiv preprint arXiv:1606.08415.

78. A General Language Assistant as a Laboratory for Alignment

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N. and others,, 2021. arXiv preprint arXiv:2112.00861.

79. **The low-rank eigenvalue problem**   [PDF]
Nakatsukasa, Y., 2019. arXiv preprint arXiv:1905.11490.