Finding Feature Vectors in Transformers with Observable Propagation

Jacob Dunefsky
Department of Computer Science
Yale University
New Haven, CT 06520, USA
`jacob.dunefsky@yale.edu`

Abstract

In this work, we introduce a method, called "observable propagation", for finding linear features in Transformer-based language models corresponding to given "observables", linear functionals representing a property of the model's output that we want to measure. We can then compare linear features corresponding to different observables, in order to determine features that represent more complex concepts. We can also use these feature vectors' norms to get a rough estimate of the degree to which various attention heads will contribute to the model's output under the observable *without performing a forward pass*. We also define *coupling coefficients*, that predict the degree to which one feature's output is coupled to another without performing a forward pass. We demonstrate that the resulting features do not always correspond with features obtained via performing SVD on weight matrices, indicating that the method presented here provides novel utility in finding feature vectors that others might miss. We extend our method to nonlinearities in Transformers, providing mathematical results on how these feature vectors interact with LayerNorm layers, and explaining how feature vectors for MLP layers can be found. Finally, we present two case studies in which we demonstrate our method: one involving gendered pronoun prediction, and another involving gendered occupational bias. Preliminary results from the latter suggest that observable propagation can be used to mitigate gender bias in large language models.

1 Introduction

When a large language model predicts that the next token in a sentence is far more likely to be "him" than "her", what is causing it to make this decision? The field of mechanistic interpretability aims to answer questions such as these by investigating how to decompose the computation carried out by a model into human-understandable pieces. The hope is that by doing so, we can better predict models' outputs on real-world data distributions, understand where the model's actual behavior might differ from its intended behavior, determine modifications that can bring the model in greater alignment with our own goals, and build trust in models that are to be deployed in high-risk settings (or understand when models should not be trusted) (Olah et al., 2018).

One important notion in mechanistic interpretability is that of "features". A feature can be thought of as a simple function of the embeddings at a particular layer of the model, the value of which function is important for the model's computation at that layer. For instance, in the textual domain, features used by a language model at some layer might reflect whether a token is an adverb, whether the language of the token is French, and so on.

Possibly the most sought-after type of feature is a "linear feature", which is defined by a linear functional on the embedding space. Intuitively, a linear feature corresponds to a vector in embedding space, such that the model makes use, in its computation, of the extent to which the embedding lines up with that vector. (That is: the model makes use of the dot product of the embedding with that vector.) Linear features are in some sense the holy grails of features: they have all of the nice mathematical properties associated with linear functions, and are especially easy for humans understand. Olah (2022) goes as far as to argue that finding linear features is "[a]rguably [...] the central task" of mechanistic interpretability. Of course, modern-day neural networks are highly nonlinear beasts–but nevertheless, the hope is that there is still enough linear structure in these neural networks so as to allow linear features to illuminate a great portion of the models' computation.

Our primary contribution in this work is a paradigm for both finding feature vectors in large language models corresponding to given tasks and analyzing these features in order to understand how they affect various different tasks. We engage in a thorough exploration of how our approach handles nonlinearities in Transformers, including proving a theorem that demonstrates that LayerNorm do not change the direction of linear feature vectors (Theorem 1). We also introduce a metric of feature vector similarity called the "coupling coefficient", and prove that this metric measures the expected contribution of one feature vector to another task (Theorem 2). We then demonstrate in a case study that our methods can be used to find linear feature vectors in the language model GPT-Neo-1.3B (Black et al., 2021) that are responsible for gendered pronoun prediction (Section 6). We also provide preliminary results regarding the potential use of our method as a step towards mitigating gender bias in large language models (Section 7).

2 Related work

Recently, the field of mechanistic interpretability has been moving at a very quick pace, producing large amounts of research, much of which is made public via non-traditional means (e.g. blog posts) so that the larger community can benefit from it as quickly as possible. As such, rather than provide a detailed survey of mechanistic interpretability work, we will present a selection of research particularly relevant to our goal of finding feature vectors in Transformers.

To a large extent, mechanistic interpretability research in Transformers was inaugurated by Elhage et al. (2021), which provided a mathematical formulation for analyzing the internal computations of Transformers. Much of the vocabulary and concepts used in this article (e.g. "circuits", "OV/QK matrices") come from this paper. Additionally, the authors analyze the weight matrices of Transformers in order to attempt to determine which heads are "induction heads". This is somewhat similar to the general spirit of observable propagation, which involves analyzing weight matrices to find items of interest – although in our case, we seek feature vectors rather than attention heads.

There is a good amount of research involving finding feature vectors by performing supervised training of probes: small linear models that aim to find directions in embedding space that correspond to certain labels. Gurnee et al. (2023) find "neurons" (i.e. basis vectors in embedding space) somewhat corresponding to concepts such as "French text" and "code written in the Go programming language". Li et al. (2023) use probing to find directions in embedding space correlated with truthfulness, and use these feature directions to perturb the internal activations of the model in order to elicit more truthful answers from it. Note that observable propagation finds feature vectors analytically, without requiring any training on large datasets. (Note that for greater accuracy when dealing with nonlinearities, extremely small amounts of data (i.e. two examples) can be used with observable propagation in order to take gradients, as described in Section 4.3.)

One of the key ideas of our method is the simple fact that, for a linear model that maps $x \mapsto Wx$, then if we want to measure the dot product between the output Wx and some vector n, then this is equivalent to measuring the dot product between the input x and the vector $W^T n$. To our knowledge, using this idea to construct the feature vector $W^T n$ that directly corresponds to the "observable" n is novel. It is worth noting, however, that prior work does look at choosing among a pre-selected matrix of vectors by taking the dot product of a vector representing a set of tokens with each vector in the matrix. Miller & Neo (2023), for instance, attempt to find neurons – that is, column vectors in MLP weight matrices – that are responsible for predicting the token `" an"`, by looking at which neurons have the highest dot product with the embedding for `" an"`. Additionally, Millidge & Black (2023) investigate the singular value decompositions of weight matrices, and seek to find singular vectors corresponding to a set of tokens by looking at which singular vectors have the highest cosine similarities with the sum of the tokens' embeddings. However, we believe that our method improves on this prior work in a few ways. Most importantly is that our method is not restricted to finding the feature vectors most relevant to a given "observable" from a relatively small set. In Section 6.8, we demonstrate that our method finds feature vectors that do not have high cosine similarities with weight matrices' singular vectors. This shows the utility of being able to construct feature vectors, rather than merely choosing among some. In addition, our conception of an “observable” is that it is any linear functional on the logits, rather than simply a positive sum of tokens’ embeddings as was considered in this prior work. This gives observable propagation greater expressive power by allowing it to consider the whole space of linear functionals on logits.

One important part of our work is the extension of our method to address nonlinearities in Transformers. The most directly relevant work here was carried out by Nanda et al. (2023), who sought to use local linear approximations of Transformers in order to speed up interpretability investigations utilizing activation patching methods. In his blog post explaining this work, Neel Nanda discusses the use of logit differences as a useful metric for activation patching experiments (the generalization of which – i.e. linear functionals on the logits – we treat as concrete objects called “observables”); he discusses the idea of recursively taking gradients throughout a model with respect to embeddings; and he provides intuition regarding the gradients of LayerNorm. We build upon this work in many ways. Most importantly, we treat observables and feature vectors as concrete objects worthy of investigation, whereas Nanda et al. (2023) are primarily interested in determining the importance of different components in the model more than trying to find feature vectors. Additionally, we add heavily to their initial analysis of LayerNorm gradients. We rigorously prove in Theorem 1 that in high dimensions, LayerNorm should not affect the direction of feature vectors, according with their intuition; but in contrast, we demonstrate that treating LayerNorm gradients as constant is unprincipled, due to the effect that the norm of the input embeddings has on the gradient (see Section 4.3.3).

In Section 7, we present preliminary results regarding an application of our method to addressing gender bias in a language model by modifying its activations. One recent work that addresses this issue in this manner is that of Belrose et al. (2023), which present an analytical formula for transforming embeddings in order to provably prevent any linear classifier from detecting a protected attribute, such as gender. While their method is more robust than our preliminary result, it is worth noting that their method is very data-intensive and involves a computationally-complex training step. We hope that the ideas presented here might begin to pave the way for efficient and robust debiasing methods.

3 Our paradigm: observables

Often, in mechanistic interpretability, we care about interpreting the model’s computation on a specific task. In particular, the model’s behavior on a task can frequently be expressed as the difference between the logits of two tokens. For instance, Mathwin et al. (2023) attempt to interpret the model’s understanding of gendered pronouns, and as such, measure the difference between the logits for the tokens `" he"` and `" she"`. Similarly, Miller & Neo (2023) attempt to understand what causes the model to predict the indefinite article `" an"` instead of `" a"`, and thus measure the difference between the logits for those two tokens. This has been identified as a general pattern of taking “logit differences” that appears in mechanistic interpretability work (Nanda, 2022).

The first insight that we introduce is that each of these logit differences corresponds to a *linear functional on the logits*. That is, if the logits are represented by y, then each logit difference can be represented by $n^T y$ for some vector n. For instance, if e_{token} is the one-hot vector with a one in the position corresponding to the token *token*, then the logit difference between “ an” and “ a” corresponds to the linear functional $(e_{\text{“ an”}} - e_{\text{“ a”}})^T$. If we then broaden our view to consider linear functionals in general, then we can construct linear functionals that correspond to more complex concepts. For instance, a linear functional like $((e_{\texttt{" her"}} + e_{\texttt{" she"}} + e_{\texttt{"woman"}}) - (e_{\texttt{" him"}} + e_{\texttt{" he"}} + e_{\texttt{"man"}}))^T$ might represent a more general concept of female grammatical gender[1].

We thus define an **observable** to be a linear functional on the logits of a language model.[2] In doing so, we no longer consider logit differences as merely a part of the process of performing

[1]Note that the tokens `"woman"` and `"man"` correspond to the suffixes “-woman” and “-man”, as in “businesswoman” versus “businessman”.

[2]This name is intended to suggest the notion of “observables” from quantum mechanics; in that domain, observables are Hermitian operators corresponding to certain properties of a quantum system that one wants to measure. Although there are some differences between that setting and the LLM setting (for instance, a linear functional is not a Hermitian operator), the similarities between the two cases provide valuable intuition: in both cases, an observable is a linear map that corresponds to a type of measurement of a probabilistic system.

an interpretability experiment; rather, we consider the broader class of linear functionals as being objects amenable to study in their own right. As we will see, concretizing observables thus will enable us to find sets of feature vectors corresponding to different observables and determine the effect that different feature vectors have on different observables.

4 Observable propagation: from observables to feature vectors

In this section, we present our method, which we call "observable propagation", for finding feature vectors directly corresponding to a given observable. We start by explaining the method for simple cases, and then build up to a general understanding. A description of the full method is given in Section 4.4.

4.1 Single attention heads

Once one starts thinking in terms of observables, it becomes easier to find linear features corresponding to those observables. First, let us consider a linear model $f(x) = Wx$. Given an observable n, we can compute the measurement associated with n as $n^T f(x)$, which is just $n^T Wx$. But now, notice that $n^T Wx = (W^T n)^T x$. In other words, $W^T n$ is a feature vector in the domain, such that the dot product of the input x with the feature vector $W^T n$ directly gives the output measurement $n^T f(x)$. Because we have propagated the observable backward into the input domain, we call this method "observable propagation".

This gives us a blueprint for finding feature vectors for a given observable whenever we have linear structure in our model. And indeed, Transformers' attention layers display a large amount of linear structure. If we follow the formulation given in Elhage et al. (2021), then each attention layer can be decomposed as

$$x_j^{l+1} = x_j^l + \sum_{\text{attention head } h} \sum_{\text{token index } i} \text{score}_h(x_i^l, x_j^l) W_{l,h}^{OV} x_i^l \tag{1}$$

where x_j^l is the residual stream for token j at layer l, $\text{score}_{l,h}(x_i^l, x_j^l)$ is the attention score at layer l associated with attention head h for tokens x_i^l and x_j^l, and $W_{l,h}^{OV}$ is the combined output-value weight matrix for attention head h at layer l.

In order to go from the output x_j^L of our final attention layer L to our logits y_j, we multiply everything by an unembedding matrix W^U: $y_j = W^U x_j^L$.

Now, as Elhage et al. (2021) note, if we consider attention scores to be fixed, then the contribution of an attention layer to the residual stream is just a weighted sum of linear terms for each token. This means that given an observable n, we can decompose the final attention layer as

$$\begin{aligned} n^T y_j &= n^T W^U x_j^L + n^T \sum_{\text{attention head } h} \sum_{\text{token index } i} \text{score}_h(x_i^L, x_j^L) W^U W_{L,h}^{OV} x_i^L \\ &= n^T W^U x_j^L + \sum_{\text{attention head } h} \sum_{\text{token index } i} \text{score}_h(x_i^L, x_j^L) n^T W^U W_{L,h}^{OV} x_i^L \\ &= n^T W^U x_j^L + \sum_{\text{attention head } h} \sum_{\text{token index } i} \text{score}_h(x_i^L, x_j^L) ((W^U W_{L,h}^{OV})^T n)^T x_i^L \end{aligned}$$

Thus, for each attention head h, the amount that token embedding x_i^L contributes to the final measurement for observable n is proportional to $((W^U W_{L,h}^{OV})^T n)^T x_i^L$. This suggests that for an observable n, each attention head h in the final layer L has a feature vector given by $(W^U W_{L,h}^{OV})^T n$. But now, note that this works for previous-layer attention heads as well, because those attention heads' outputs remain in the residual stream x_j^L, and we still have the $n^T W^U x_j$ term in our sum.

As such, for any attention head h in any layer l, there is a feature vector corresponding to that attention head given by $(W^U W_{l,h}^{OV})^T n$.

4.2 Virtual attention heads

So far, these feature vectors correspond to the direct contribution that each attention head has to the output. But an earlier-layer attention head's output can then be used as the input to a later-layer attention head. In the formulation of Elhage et al. (2021), this is called "OV composition". The idea is as follows. Let l, l' be two layers in the transformer with $l > l'$ (that is, l corresponds to a later layer). Let h be an attention head index; let i be a token index. Then the term $W^{OV}_{l,h} x^l_i$ in Equation 1 can be decomposed further as

$$\begin{aligned} W^{OV}_{l,h} x^l_j &= W^{OV}_{l,h} \left(\cdots + \sum_{\text{attention head } h} \sum_{\text{token index } i'} \text{score}_h(x^{l'}_i, x^{l'}_{i'}) W^{OV}_{l',h} x^{l'}_{i'} \right) \\ &= W^{OV}_{l,h} \left(\cdots + \sum_{\text{attention head } h'} \sum_{\text{token index } i'} \text{score}_{h'}(x^{l'}_{i'}, x^{l'}_j) W^{OV}_{l',h'} x^{l'}_{i'} \right) \\ &= \cdots + \sum_{\text{attention head } h'} \sum_{\text{token index } i'} \text{score}_{h'}(x^{l'}_{i'}, x^{l'}_j) W^{OV}_{l,h} W^{OV}_{l',h'} x^{l'}_{i'} \end{aligned}$$

where the ellipses correspond to terms from layers below l other than l'. Thus, for attention heads h, h' in layers l, l' respectively and tokens i, i', we can look at the term $W^{OV}_{l,h} W^{OV}_{l',h'} x^{l'}_{i'}$. This corresponds to the path starting at token i' in layer l', which is then passed as the input to attention head h'; the output of this attention head for that token is then used as the input to attention head h in layer l. This is precisely a "virtual attention head". By the same reasoning as in Section 4.1, the feature vector associated with this virtual attention head is given by $(W^U W^{OV}_{l,h} W^{OV}_{l',h'})^T n$. Note that this process can be repeated *ad infinitum*, to account for any number of virtual attention heads.

4.3 Introducing nonlinearities

Unfortunately for us interested in mechanistic interpretability, there are many nonlinearities in Transformers. They come in three different forms: attention scores, LayerNorm, and MLP sublayers.

We can address the nonlinearity in attention scores by treating them as fixed, and by focusing on the information that each attention head outputs rather than the attention scores that weight these outputs. As for LayerNorm and MLP sublayers, we can deal with them by approximating them as linear functions using their first-order Taylor approximations.

4.3.1 Attention scores

The attention layers of Transformers combine information across tokens. They can be decomposed into two parts: the part that determines from which tokens information is taken, and the part that determines what information is taken from each token. Referring to Equation 1, in each term in the sum, the $\text{score}_h(x^l_i, x^l_j)$ factor determines from where information is taken, and the $W^{OV}_{l,h} x^l_i$ factor determines what information is taken. Note that the primary nonlinearity in attention layers comes from the computation of the attention scores, and their multiplication with the $W^{OV}_{l,h} x^l_i$ terms.

Elhage et al. (2021) use the term "QK circuit" to refer to the computations that determine from where information is taken. They use the term "OV circuit" to refer to the computations that determine what information is output from each token. Now, the authors note that if the attention scores treated as fixed, then all of the terms in the OV circuit are linear. Thus, we can deal with the fact that attention scores are nonlinear by restricting our analysis to the OV circuit.

Of course, in doing so, we are very much ignoring a lot of the computation that goes on in Transformers. For instance, the phenomenon of "induction heads" seems to be heavily reliant on QK circuits Elhage et al. (2021). That said, if we are trying to understand how the computation of a Transformer affects its output with respect to a given observable, then analyzing OV circuits in isolation tells us *what sort of information, at each layer, corresponds to our observable.* The QK circuits might determine where that information is read from, but it is analyzing the OV circuits that tells us what we should expect to find there.

4.3.2 Taylor approximations

Given a nonlinear function $f : \mathbb{R}^d \to \mathbb{R}$, then given some x_0, we can approximate $f(x)$ as $f(x) \approx f(x_0) + \nabla f(x_0) \cdot (x - x_0)$. As such, $f(x)$ is equal to $\nabla f(x_0) \cdot x$, plus some constant. Therefore, for observable n, if we set $f(x) = n^T g(x)$ for some nonlinear g, then we can consider $\nabla f(x_0)$ to be the feature vector associated with n^T for g. In other words, if g is a LayerNorm or an MLP, then by taking the gradient of $f(x) = n^T g(x)$ at some suitable point x_0, then this gives us our feature vector for that LayerNorm or MLP.

Additionally, we can use the multivariate chain rule to compose these feature vectors for nonlinear layers with other layers in the model. Specifically, if we have an observable n, and we want to find the feature vector at layer l' for the computational path that passes through a nonlinear function g at layer $l > l'$, then we can do the following:

1. First, find the feature vector for g at layer l for observable n: $\nabla f(x_0)$, where $f(x) = n^T g(x)$ and x_0 is the point at which we take the gradient.
2. Then, *treat the feature vector* $\nabla f(x_0)$ *as the new observable*, and find the feature vector at layer l' for this "observable".

Note that this is the more general, nonlinear equivalent of the procedure for finding feature vectors associated with virtual attention heads as detailed in Section 4.2.

4.3.3 LayerNorm

LayerNorms are per-token operations that are executed before every attention sublayer and before every MLP sublayer. There is also a LayerNorm before the final unembedding matrix that produces the logits. The LayerNorm function can be defined as follows:

$$\text{LayerNorm}(x) = \frac{Px}{\|Px\|}$$

where P is the orthogonal projection onto the hyperplane orthogonal to $\vec{1}$, the vector of all ones.

Because LayerNorms are ubiquitous in Transformers, whenever we want to compute the feature vector associated with a virtual attention head, we have to take into account all of the LayerNorms that lie on the path corresponding to this virtual attention head. As such, if the LayerNorm is highly nonlinear, then this would spell doom for the goal of using this approach to find feature vectors.

Nanda et al. (2023) provide intuition for why we should expect that in high-dimensional spaces, LayerNorm is approximately linear. This largely held empirically in our experiments when we used a linear approximation of LayerNorm with inputs from the same prompt template. However, in experiments involving inputs from different prompt templates, we found that taking the gradient of LayerNorm at an input from one of the prompt templates yielded a somewhat inaccurate approximation for inputs from other prompt templates; see Appendix A for more details on these experiments.

Even though the LayerNorm gradients are not perfectly constant, we did find something interesting: if n is an observable, then the gradient of $n^T \text{LayerNorm}(x)$ was always almost an exact scalar multiple of n. In other words, it seemed that *LayerNorms had almost no impact on the direction of feature vectors*.

In particular, this can be formalized as the following statement:

Theorem 1. *Define* $f(x; n) = n \cdot \text{LayerNorm}(x)$. *Define*

$$\theta(x; n) = \arccos\left(\frac{n \cdot \nabla_x f(x; n)}{\|n\| \, \|\nabla_x f(x; n)\|}\right)$$

– that is, $\theta(x; n)$ *is the angle between* n *and* $\nabla_x f(x; n)$. *Then if* x *and* n *are i.i.d.* $\mathcal{N}(0, I)$ *in* $\mathbb{R}^d$, *and* $d \geq 8$ *then*

$$\mathbb{E}\left[\theta(x; n)\right] < 2 \arccos\left(\sqrt{1 - \frac{1}{d-1}}\right)$$

Note that this bound is very loose – the actual expected angle between an observable n and the observable corresponding to n post-LayerNorm will always be lower then the given bound. A proof of this statement can be found in Appendix B.

There is an additional slight caveat regarding LayerNorms: after every LayerNorm in a Transformer, the output is multiplied by a fixed scalar constant equal to $\sqrt{d}$ (where d is the embedding diension), multiplied by a learned diagonal matrix, and added with a learned vector. Thus, the actual operation can be represented as $\sqrt{d}W\,\mathrm{LayerNorm}(x) + b$, where W is the learned matrix and b is the learned vector. Now, b does not affect the gradient. Additionally, empirically, most of the entries in W tend to be very close to one another (see Appendix A.2), which suggests that we can approximate W as a scalar, meaning that W primarily scales the gradient, rather than changing its direction.

Thus, since the direction of LayerNorm gradients is nearly constant for a given observable, then the reason why LayerNorm gradients are not constant is because their magnitudes are not constant. In particular, the gradient of $\mathrm{LayerNorm}(x)$ can be shown to be $\frac{1}{\|Px\|}P\left(I - \frac{(Px)(Px)^T}{\|Px\|^2}\right)n$ (see Appendix B). P and $\left(I - \frac{(Px)(Px)^T}{\|Px\|^2}\right)$ are both orthogonal projections that leave $\|n\|$ relatively untouched, so the remaining suspect is the $\frac{1}{\|Px\|}$ factor; by Lemma 1 in Appendix B, we have that $\frac{1}{\|Px\|} \approx \frac{1}{\|x\|}$. Indeed, experimentally, we found that the norms of the embeddings vary greatly depending on the layer at which the norms are being investigated. (See Appendix **??**.) Thus, if we have a good estimate of $\|x\|$ for a given set of input prompts at a given layer, then we can account for this term by dividing the gradient by $1/\|x\|$.

The takeaway here: if we want to analyze the cosine similarity between feature vectors, then *we can do so without worrying about LayerNorms*. If we want to measure properties of feature vectors related to their norms, then we can obtain a very good approximation by multiplying the feature vectors by $\frac{\sqrt{d}}{\|x\|}W$, where d is the embedding dimension, W is the scaling matrix of the LayerNorm, and $\|x\|$ is the norm of a token embedding at that layer.

4.3.4 MLPs

Finding feature vectors for MLPs is a relatively straightforward application of the first-order Taylor approximation. However, there is a fear that if one takes the gradient at the wrong point, then the local gradient will not reflect well the larger-scale behavior of the MLP. For example, the output of the MLP with respect to a given observable might be *saturated* at a certain point: the gradient at this point might be very small, and might even point in a direction inconsistent with the MLP's gradient in the unsaturated regime.

To alleviate this, we use the following method. Define $g(x) = n^T\,\mathrm{MLP}(x)$, where n is a given observable. If this observable n represents the logit difference between two tokens, then we should be able to find an input on which this difference is very negative, along with an input on which this difference is very positive. For example, if n represents the logit difference between the token `" her"` and the token `" him"`, then an input containing a male name should make this difference very negative, and an input containing a female name should cause this difference to be very positive.

Thus, we have two points x_- and x_+ such that $g(x_-) < 0$ and $g(x_+) > 0$. Since MLPs are continuous, there therefore must be some point x^* at which $g(x^*) = 0$: a point that lies on the decision boundary of the MLP. It stands to reason that the gradient at this decision boundary is more likely to capture the larger-scale behavior of the MLP and is less likely to be saturated, when compared to the gradient at more "extreme" points like x_- and x_+. This tends to be borne out empirically; refer to Figure 4.

4.4 Summary and general form of observable propagation

In this section, we present observable propagation, our method for finding feature vectors corresponding to any given observable. In particular, we look at the feature vectors in the "OV circuit" of the Transformer; the OV circuit determines what information is read from and written to each token during the attention layers. The general form of observable propagation can be implemented as follows.

Consider a computational path $\mathcal{P}$ in the model through sublayers $l_1 < l_2 < \cdots < l_k$. Denote the function implemented at sublayer l_k by f_k. Then for a given observable n, the feature vector corresponding to sublayer l in $\mathcal{P}$ can be computed recursively as follows:

1. Find y_k, the feature vector corresponding to l_k. If f_k is an attention head with OV matrix W_k, then y_k is given by $y_k = W_k^T n$. If f_k is an, then y_k can be approximated as $\nabla(n^T f_k(x))|_{x=x_0}$ for some suitable value of x_0. In particular, if we have a point x_- such that $n^T f_k(x_-) < 0$, and if we have a point x_+ such that $n^T f_k(x_+) > 0$, then a good choice of x_0 would be a point on the line between x_- and x_+ such that $n^T f_k(x_0) = 0$.
2. Now, before every sublayer, there is a corresponding LayerNorm operation defined by $\sqrt{d}W\,\mathrm{LayerNorm}(x) + b$, where b is some vector, d is the model dimensionality, and W is some diagonal matrix. For greatest accuracy, one can treat this LayerNorm operation as a distinct nonlinear sublayer, and find a feature vector corresponding to the LayerNorm by taking the gradient. But if one has an estimation of the norm of inputs $\|x\|$, then one can approximate the gradient as $\frac{\sqrt{d}}{\|x\|}Wy_k$. If one only cares about the directions of the feature vectors and not their magnitudes, then the LayerNorms can be ignored entirely.
3. Once y_k is found, then y_{k-1} can be found by repeating the above process, using y_k as the observable rather than n.

Note that this process essentially corresponds to taking the gradient of the computational subgraph corresponding to path $\mathcal{P}$. But if one wanted to use an autograd library to directly compute the gradient, one would have to modify the model's forward pass code to keep values outside of the path $\mathcal{P}$ fixed. In contrast, this method of computing the gradient is easy to implement, without forcing the user to hook into the model code.

5 Data-free analysis of feature vectors

Once we have used this to obtain a given set of feature vectors, we can then perform some preliminary analyses on them, using solely the vectors themselves. This can give us insights into the behavior of the model without having to run forward passes of the model on data.

5.1 Feature vector norms

One technique that can be used to assess the relative importance of model components is looking at the norms of the feature vectors associated with those components. To see why, recall that if y is a feature vector associated with observable n for a model component that implements function f, then for an input x, we have $n \cdot f(x) = y \cdot x$. Now, we have the following fact:

Fact. *Let $x \sim \mathcal{N}(0, I)$. Then for a given y, we have $\mathbb{E}[(x \cdot y)^2] = \|y\|^2$.*

Proof. First, note that $(x \cdot y)^2 = \|x\|^2 \|y\|^2 \cos(\theta)^2$, where θ is the angle between x and y. Now, because the norm of x is independent of its direction, we have $\mathbb{E}\left[(x \cdot y)^2\right] = \|y\|^2 \,\mathbb{E}\left[\|x\|^2\right] \mathbb{E}\left[\cos(\theta)\right]$. If the dimension of the vector space is d, then $\mathbb{E}\left[\|x\|^2\right] = d$ (using the mean of a chi-squared distribution). Additionally, because the expected squared dot product between a given vector and a uniformly-distributed unit vector is $1/d$, we have $\mathbb{E}\left[\cos(\theta)^2\right] = 1/d$. Therefore, $\mathbb{E}\left[(x \cdot y)^2\right] = \|y\|^2$. $\square$

As such, if we have no prior knowledge regarding the distribution of inputs to this component of the model, we can expect the square of the output of the component to be equal to the squared norm of the feature vector. Thus, components with larger feature vectors should have larger outputs; this was borne out empirically in experiments (see Section 6.5).

5.1.1 Dealing with LayerNorms

If one wants to perform a preliminary analysis of feature vectors without running the model on any data, then looking at the feature vectors without taking into account LayerNorms can provide a fast and reasonable "ansatz" of which components will be the most important. However, for the reasons discussed in Section 4.3.3, in order to obtain feature vectors with accurate magnitudes, knowledge about the norm of the input embeddings at each layer is required: if $\widetilde{\|x\|}$ is an estimate of the norm of the input embeddings to the LayerNorm, then the feature vectors should be multiplied by $\sqrt{d}W/\widetilde{\|x\|}$ for each LayerNorm.

However, for the purposes of predicting which components will have the highest dot products, it is important to note: *the final feature vector in the computational path being analyzed should only be multiplied by $\sqrt{d}W$, not $\sqrt{d}W/\widetilde{\|x\|}$.* To see why, let $(\sqrt{d}W/\widetilde{\|x\|})y$ be the final feature vector, after the final LayerNorm factor is applied. Then for any given x, we have that $(\sqrt{d}W/\widetilde{\|x\|})y \cdot x$ is proportional to $(\sqrt{d}W/\widetilde{\|x\|})\ \|y\|\ \|x\|$, which is approximately $\sqrt{d}W\ \|y\|$.

A convenient consequence of this is that when analyzing paths in the computational subgraph of length one – that is, paths that do not contain any compositionality such as virtual attention heads – then ignoring the estimated norm of embeddings still provides a very accurate idea of the relative importance of attention heads. This is because the only time that a $(\sqrt{d}W/\widetilde{\|x\|})$ term appears with the factor of $1/\widetilde{\|x\|}$ included is for the final LayerNorm before the logits output. As such, since this factor is not dependent on the layer of the component being analyzed, it can be ignored.

5.1.2 Dealing with attention heads

Also, there is a small caveat with this method when applied to attention heads. Recall from Section 4.3.1 that we are only examining what information the attention heads read and write between tokens, and not where the attention heads choose to look for that information. If an attention head has a large feature vector, but tends to attend to tokens with low values for that feature, then that the attention head would contribute less to the overall output than the norm of its feature vector would suggest; this did happen occasionally in experiments. Nevertheless, in our experiments, looking at feature vector norms yielded a superset of relevant attention heads in experiments; as such, we found this process to be a useful step in determining which attention heads to further analyze.

5.2 Coupling coefficients

An important question that we might want to ask about observables is the following: to what extent should we expect inputs that yield high outputs under observable n_1 to also yield high outputs for another observable n_2? If the outputs under n_1 are highly correlated with the outputs under n_2, then this suggests that the model uses the same underlying features for both observables, and treats both observables very similarly.

As an example of why we might care about this question, consider the problem of gender bias detection in language models. We might ask ourselves: if the model predicts a high logit difference between tokens `" actress"` and `" actor"`, then is the model also likely to predict a high logit difference between tokens `" nurse"` and `" programmer"`? If so, then this would indicate that the model views the profession of nursing as a female profession when compared to the profession of programming, just as the word `" actress"` denotes a specifically female profession when compared to the word `" actor"`.

Having motivated this problem, let us translate it into the language of feature vectors. If n_1 and n_2 are observables with feature vectors y_1 and y_2 for a function f, then for inputs x, we have $n_1 \cdot f(x) = y_1 \cdot x$ and $n_2 \cdot f(x) = y_2 \cdot x$. Now, if we constrain our input x to have norm c, and constrain $x \cdot y_1 = k$, then what is the expected value of $x \cdot y_2$? And what are the maximum/minimum values of $x \cdot y_2$? We present the following theorem to answer both questions:

Theorem 2. *Let $y_1, y_2 \in \mathbb{R}^d$. Let x be uniformly distributed on the hypersphere defined by the constraints $\|x\| = s$ and $x \cdot y_1 = k$. Then we have*

$$\mathbb{E}[x \cdot y_2] = k\frac{y_1 \cdot y_2}{\|y_1\|^2}$$

and the maximum and minimum values of $x \cdot y_2$ are given by

$$\frac{\|y_2\|}{\|y_1\|}\left(k\cos(\theta) \pm |\sin(\theta)|\sqrt{s^2\|y_1\|^2 - k^2}\right)$$

where θ is the angle between y_1 and y_2.

We denote the value $\frac{y_1 \cdot y_2}{\|y_1^2\|}$ by $C(y_1, y_2)$, and call it the "coupling coefficient from y_1 to y_2", because it measures the extent to which the two feature vectors are coupled. Intuitively, $C(y_1, y_2)$ measures the expected dot product between a vector and y_2, assuming that that vector has a dot product of 1 with y_1.

Note that $C(y_1, y_2) \neq C(y_2, y_1)$. In fact, $C(y_1, y_2)C(y_2, y_1) = \cos\theta$, so the coupling coefficient from y_1 to y_2 is inversely proportional to the coupling coefficient from y_2 to y_1 – and the constant of proportionality is the cosine similarity between the two vectors. Additionally, note that Theorem 2 also implies that the coupling coefficient becomes a more accurate estimator as the cosine similarity between y_1 and y_2 increases. This is borne out experimentally: we observe that the coupling coefficient very accurately estimates the ratio between the dot products of embedding vectors with various pairs of features, particularly when those pairs of features have high cosine similarity.

6 Case study: gendered pronouns prediction

Armed with our "observable propagation" toolkit for obtaining and analyzing feature vectors, we ran a series of experiments in order to determine the extent to which these tools reflect the behavior of an actual large language model. Our experiments demonstrate the utility of observable propagation for both understanding and steering large language models.

6.1 Problem setting

As a case study, we considered the problem of understanding how a large language model predicts gendered pronouns. Specifically, given a sentence prefix including a gendered name (for example, "Mike" would be a male name; "Jane" would be a female name), how does the model predict whether the next token after the sentence prefix will be a male pronoun or a female pronoun?

This problem was previously considered by Mathwin et al. (2023) during a hackathon. Specifically, the authors used the "Automated Circuit Discovery" tool presented by Conmy et al. (2023) to investigate the flow of information between different components of GPT-2-small (Radford et al., 2019) in predicting subject pronouns (i.e. "he", "she").

We use this problem as a jumping-off point for testing our feature vector methods, extending the problem setting in various ways. We investigate both the subject pronoun case (in which the model is to predict the token "she" versus "he") and the object pronoun case (in which the model is to predict "her" versus "him"), in order to demonstrate the utility of our feature vector methods in comparing features across tasks. In particular, our initial motivation was to determine if the model refers to the same underlying features to compute its output with respect to both the subject pronoun task and the object pronoun task.

6.2 Model

We investigate the model GPT-Neo-1.3B (Black et al., 2021), which has approximately 1.3 billion parameters, 24 layers, 16 attention heads per attention sublayer, an embedding space dimension of 2048, and an MLP hidden layer dimension of 8192.

Note that this is significantly larger than GPT-2-small (Radford et al., 2019), the model investigated in much recent interpretability work such as Wang et al. (2022); GPT-2-small has approximately 117 million parameters, 12 layers, 12 attention heads, and an embedding space dimension of 768. By choosing to investigate a larger language model, we hope to demonstrate that the interpretability methods presented here can scale.

6.3 Prompts

Our prompts are inspired by those used by Mathwin et al. (2023). For the subject pronoun prediction task, we generate prompts using the template `"<|endoftext|>So, [NAME] really is a great friend, isn't"`, where `"[NAME]"` is replaced by a one-token name. For the object pronoun prediction task, the prompt template is `"<|endoftext|>What do I think about [NAME]? Well, to be honest, I love"`. The idea is that the most likely next token for the subject pronoun prompts is either `" she"` or `" he"`, depending on the gender of the name. Similarly, for the object pronoun prompts, the most likely next token is either `" her"` or `" him"`.

We create a total of forty prompts, by applying the two prompt templates to ten male names and ten female names. Male names used are “John”, “David”, “Mark”, “Paul”, “Ryan”, “Gary”, “Jack”, “Arnold”, “Joe”, “Andy”; female names used are “Jennifer”, “Jane”, “Annie”, “Chloe”, “Amy”, “Judy”, “Lisa”, “Carol”, “Clara”, “Sarah”. Note that all names are one token long.

6.4 Observables

We considered two observables, corresponding to the subject pronoun prediction task and the object pronoun prediction task. The observable for the subject pronoun task, n_{subj}, is given by $e_{\text{`` she''}} - e_{\text{`` he''}}$, where e_{token} is the one-hot vector with a one in the position corresponding to the token *token*[3]. This corresponds to the logit difference between the tokens “ she” and “ he”, and indicates how likely the model predicts the next token to be “ she” versus “ he”. Similarly, the observable for the object pronoun task, n_{obj}, is given by $e_{\text{`` her''}} - e_{\text{`` him''}}$.

6.5 Feature vector norms

We begin by analyzing the norms for the feature vectors corresponding to n_{subj} and n_{obj} for each attention head in the model. We will analyze these values both ignoring and taking into account LayerNorms; the purpose of ignoring LayerNorms is to demonstrate the utility of the method even without performing any forward pass on the model.

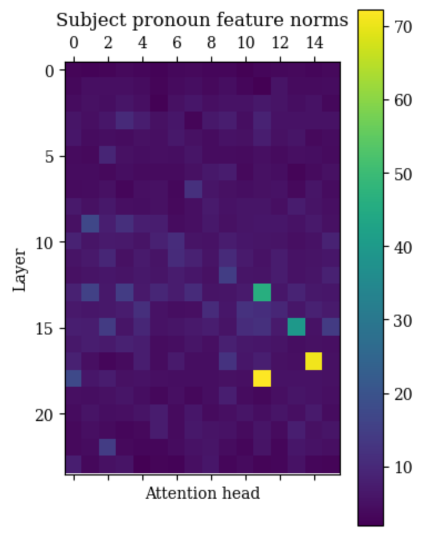
The feature vector norms calculated without taking into account LayerNorm are given in Figure 1. We can see that the same four heads have the highest feature norms for both n_{subj} and n_{obj}: 18.11, 17.14, 13.11, and 15.13. The norms of the n_{subj} feature vectors for these heads are $72.2168, 70.7219, 45.9200, 39.4049$ respectively; the norms of the n_{obj} feature vectors for these heads are $69.2137, 68.9342, 51.8211, 44.0450$ respectively.

We then calculated the feature vector norms taking into account LayerNorms as described in Section 5.1.1. As our estimator of $\widetilde{\|x\|}$, the average embedding norm before the pre-logits LayerNorm, we used the mean norm of the pre-logits embedding of the last token in each prompt for the respective task type (i.e. subject versus object). (We looked at the last token because the logits for this token are what predict the next token, and as such, what we measure with our observables.) Recall that here, we do not divide the feature vector by the pre-attention embedding norm; this is principled, as explained in Section 5.1.1.

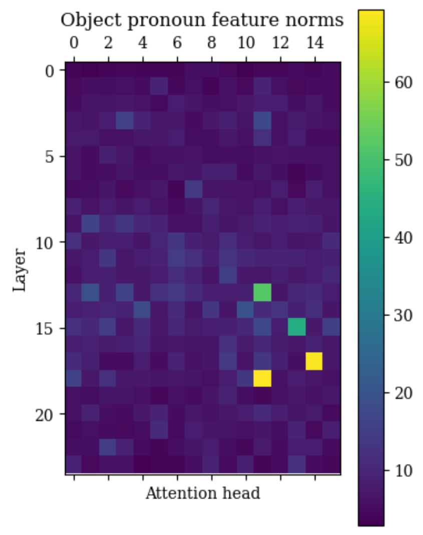
After taking LayerNorms into account, the attention heads with the highest feature vector norms for n_{subj} are 18.11, 17.14, 13.11, and 15.13, with norms $237.3204, 236.2457, 186.3744, 145.4124$ respectively. The attention heads with the highest feature vector norms for n_{obj} are 17.14, 18.11, 13.11, and 15.13, with norms $159.1822, 156.9978, 144.9379, 112.3253$; note that these are the same attention heads found in the LayerNorm-less case, although the ordering of 17.14 and 18.11 is switched.

We then used path patching (Goldowsky-Dill et al., 2023) to measure the degree to which each attention head contributes to the output. The results can be seen in Figure 2. The four attention heads with the highest attributions for n_{subj} are 17.14, 13.11, 15.13, and 13.3, with corrupted-clean logit differences of $4.7859, 4.5473, 1.0804, 0.6352$ respectively. The four attention heads with the highest attributions for n_{obj} are 17.14, 15.13, 13.11, and 22.2, with corrupted-clean logit differences

[3]Note the leading space in the token strings: this is intentional. If the space were not present, then the tokens would correspond to the strings “she” and “he” used in the middle of words, or as suffixes.

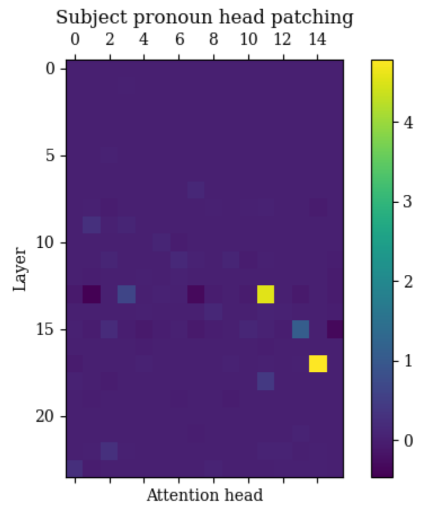


(a) Subject pronoun feature vector norms

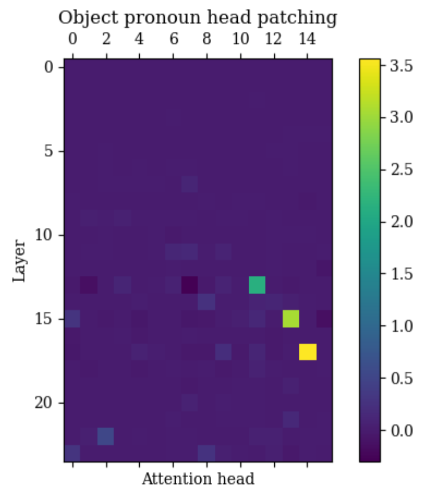


(b) Object pronoun feature vector norms

Figure 1: Feature vector norms, *ignoring LayerNorms*, by attention head and layer for the subject pronoun observable n_{subj} and the object pronoun observable n_{obj}.



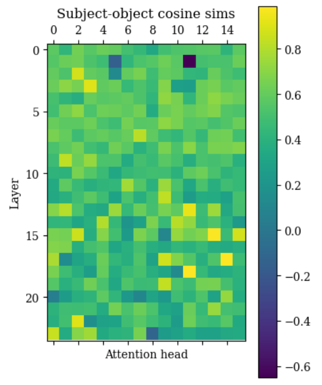
(a) Subject pronoun head attributions



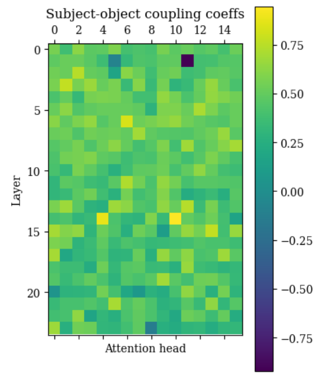
(b) Object pronoun head attributions

Figure 2: Path patching attributions for each attention head and layer for the subject pronoun observable n_{subj} and the object pronoun observable n_{obj}.

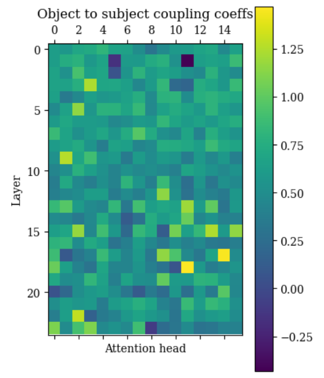
of $3.5636, 3.0652, 2.1328, 0.5218$ respectively. We see that three of the four attention heads with the highest feature norms – that is, 17.14, 15.13, and 13.11 – also have very high attributions for both the subject and object pronoun cases. Interestingly, head 18.11, despite having a large feature norm, does not have a high attribution in either case. This may be due to effects involving the attention head's QK circuit not attending to tokens even when the output of its OV circuit for those tokens has a high dot product. Overall, however, we found it very impressive that simply looking at the feature vectors' norms was able to correctly predict three of the four attention heads with the highest attribution scores.



(a) Cosine similarities between feature vectors for n_{subj} and n_{obj}.



(b) Couplings from feature vectors for n_{subj} to feature vectors for n_{obj}.



(c) Couplings from feature vectors for n_{obj} to feature vectors for n_{subj}.

Figure 3: Cosine similarities and coupling coefficients between feature vectors for n_{subj} and n_{obj}.

6.6 Cosine similarities

Next, we investigated the cosine similarities between the feature vectors for n_{subj} and n_{obj}. The cosine similarities are found in Figure 3a. The four heads with the highest cosine similarities between its n_{subj} feature vector and its n_{obj} feature vector are 17.14, 18.11, 15.13, and 13.11, with cosine similarities of 0.9882, 0.9831, 0.9816, 0.9352. The high cosine similarities of these feature vectors indicates that the model uses the same underlying features for both the task of predicting subject pronoun genders and the task of prediction object pronoun genders.

However, it could possibly be the case that these high cosine similarities are an artifact of the observables themselves being very similar – rather than evidence of the model using similar information to contribute to different observables; if so, then these results would be less interesting. We thus then considered the cosine similarities of $W_U^T n_{\text{subj}}$ and $W_U^T n_{\text{obj}}$, where W_U^T is the token unembedding matrix. (These vectors represent the observables as mapped into embedding space; in token space, $n_{\text{subj}} \cdot n_{\text{obj}} = 0$ trivially because the two observables measure different tokens.)

But, we found that the cosine similarity between these two vectors is 0.5685. While this is not low, it is certainly not high. As such, this supports the idea that the high cosine similarities between the feature vectors reflect the model using shared features for diverse tasks, rather than merely using shared features for the same task.

Also, we found it interesting that these four heads are the same as the four heads with the highest feature norms. That said, even though there is a high cosine similarity between the n_{subj} and n_{obj} feature vectors for attention head 15.13, this head has a much higher attribution value in the n_{obj} case than the n_{subj} case: 2.1328 logits versus 1.0804 logits. We suspect that this is due to effects in the QK circuit of the model.

6.7 Coupling coefficients

We then computed the coupling coefficients between these feature vectors. The coupling coefficients from the n_{subj} feature vectors to the n_{obj} feature vectors can be found in Figure 3b; the coupling coefficients from n_{obj} to n_{subj} can be found in Figure 3c.

In particular, we investigated the coupling coefficients for heads 17.14, 15.13, and 13.11. This is because these heads were present among the heads with the highest cosine similarities, highest feature norms, and highest patching attributions, for both the n_{subj} and n_{obj} cases. The coupling coefficients from n_{subj} to n_{obj} of these heads are 0.67086, 0.77306, and 0.73552 respectively. This means that, on average, if an input causes attention head 17.14 to contribute k logits to the observable n_{subj}, then that input will cause it to contribute $0.67k$ logits to n_{obj}, and the same with the other heads and their respective coupling coefficients.

Head	Coupling coefficient	Cosine similarity	Best-fit slope	r^2
17.14	0.67086	0.9882	0.67774	0.99811
15.13	0.77306	0.9816	0.79353	0.98146
13.11	0.73552	0.9352	0.75580	0.9274281

Table 1: Cosine similarity and coupling coefficients from n_{subj} to n_{obj}, versus the slope of the best-fit line of dot products with n_{subj} feature vectors versus dot products with n_{obj} feature vectors.

We then tested the extent to which the coupling coefficient accurately predicts the constant of proportionality between the dot products of different feature vectors with their inputs. We collected embeddings for all of the prompts described in 6.3 and, for each token's embedding in each prompt, collected the dot product of the post-LayerNorm attention input with the n_{subj} and n_{obj} feature vectors for the three attention heads listed in this section. We then computed the least-squares best fit line to predict the n_{obj} dot product of each head from the n_{subj} dot product. The slope and r^2 correlation coefficient of each line is given in Table 1. Note that the coupling coefficients are all very close to the slopes of the best-fit lines, indicating that the coupling coefficient is accurate. Additionally, note that the r^2 values measuring the strength of the linear correlation are higher for attention heads whose cosine similarities are higher; this is related to the prediction of Theorem 2 that coupling coefficients are more accurate estimators when the cosine similarity between the two vectors is higher.

6.8 Singular value decomposition

Millidge & Black (2023) investigated taking the SVD of weight matrices for attention heads and MLP layers in language models, in order to automatically find a set of generally important feature vectors. We wanted to see if observable propagation could yield feature vectors relevant to specific tasks that SVD might miss.

As such, for the n_{subj} and n_{obj} feature vectors for heads 17.14, 15.13, and 13.11, we computed the SVD of the W_{OV} matrices associated with those heads and looked at which right singular vectors had the greatest cosine similarities with the feature vectors obtained via observable propagation.[4] Intuitively, if the feature vectors do not have high cosine similarity with the right singular vectors of the weight matrices, then this implies that SVD alone is not enough to accurately capture all of the feature vectors that one might be interested in.

The five right singular vectors for 17.14 with the highest cosine similarities with the n_{subj} feature were vectors 1, 0, 2, 7, and 26, with cosine similarities $0.8204, 0.2875, 0.0865, 0.0277, 0.0176$ respectively. The five right singular vectors with the highest similarities with the n_{obj} feature were vectors 1, 0, 2, 22, and 42; their cosine similarities were $0.7955, 0.2849, 0.0773, 0.0308, 0.0263$ respectively. In both cases, there was a rather high cosine similarity with right singular vector 1. We applied the logit lens technique (nostalgebraist, 2020) to this singular vectors in order to see which tokens they are most heavily associated with; the top five tokens were `" she"`, `" her"`, `" herself"`, `" hers"`, and `" itself"`, while the bottom five tokens were `" himself"`, `" his"`, `"his"`, `" HIS"`, and `" His"`. In this case, it is clear that SVD yields a singular vector that does indeed capture the nature of the observables that we are considering. (It is worth noting, though, that the cosine similarity with singular vector 1 is less than even 0.8 – indicating that there are still some aspects of this feature vector that are not captured by the right singular vector.)

However, SVD yields less success for attention heads 15.13 and 13.11. For 15.13, the five most similar right singular vectors for the n_{subj} feature had cosine similarities $0.1257, 0.0864, 0.0703, 0.0644, 0.0592$; the five most similar ones for the n_{obj} feature had cosine similarities of $0.1659, 0.0608, 0.0560, 0.0559, 0.0545$. For 13.11, the five highest cosine similarities for n_{subj} were $0.5802, 0.3474, 0.2169, 0.1389, 0.0500$; the five highest cosine similarities for n_{obj} were $0.5986, 0.4873, 0.0818, 0.0616, 0.0579$.

Thus, although SVD was useful in largely recovering a relevant right singular vector for attention head 17.14, it was not able to recover any one specific singular vector that aligned well with the

[4]The reason that we consider the *right* singular vectors is because these are the singular vectors that correspond to directions in the input space of each attention head, in the same way that our feature vectors do.

Head	n_{subj} vs n_{obj}	Token	$-50v$ mean difference	$+50v$ mean difference
17.14	n_{subj}	15	-100.780	87.517
15.13	n_{subj}	6	-13.238	26.132
13.11	n_{subj}	6	-4.111	12.659
17.14	n_{obj}	15	-87.038	70.865
15.13	n_{obj}	6	-22.681	23.150
13.11	n_{obj}	6	-16.343	5.992

Table 2: Activation steering results for n_{subj} and n_{obj}. The "Token" column denotes the token to which the feature vector v was added.

feature vectors for 15.13 and 13.11. This suggests that observable propagation has utility in finding feature vectors that SVD alone might miss.

6.9 Activation steering

Recently, work has been done on investigating "activation steering" for large language models: finding vectors that can be added to embeddings in order to change the model's behavior in a desired way. For instance, Rimsky & Hubinger (2023) use activation steering to induce a language model to yield harmful outputs; Li et al. (2023) use activation steering to cause a language model to output more truthful answers. Both works found vectors by utilizing labeled datasets.

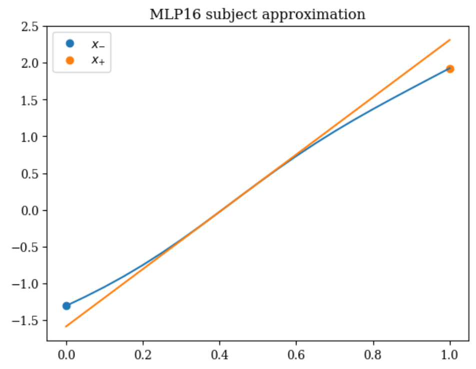
We therefore were interested in seeing whether the data-free feature vectors obtained via observable propagation could be used to effectively steer the model's outputs for different observables. If so, then this supports the idea that observable propagation can be used to quickly find steering vectors for any given observable. Now, if Transformers were wholly linear, then this would clearly be the case, given how observable propagation was derived in Section 4.1. But Transformers are not linear, as was discussed in Section 4.3; it is therefore possible that these linear feature vectors do not adequately capture non-linear behavior in the model. Thus, these activation steering experiments also served as a way to investigate the fidelity of observable propagation.

We evaluated activation steering for n_{subj} and n_{obj} on the set of prompts corresponding to the respective observable (see Section 6.3). For the three heads 17.14, 15.13, and 13.11, we first performed logit attribution (Nanda, 2022) on the attention heads to determine which token was the most important. Then, we added $\pm 50v$ to the pre-LayerNorm activations for that token, where v is the feature vector for that attention head (without taking into account LayerNorms)[5]. We recorded the mean change in output, with respect to each observable, between the clean run and the run with the added feature vector, over all of the prompts in the dataset for that observable.

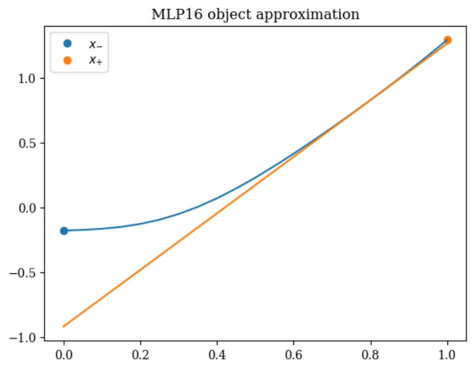
The results are summarized in Table 2. We find that generally, activation steering works: adding these feature vectors to certain tokens can very much change the predictions of the model. However, it is worth noting that the effects of activation steering are not as great as they would be if a Transformer was truly a linear model. If this were the case, then the change in the output logits would be proportional to the squared norm of the steering vector, but this is not what we see in the results. Additionally, we see asymmetries between the effect of subtracting the feature vector and the effect of adding the feature vector; sometimes, adding produces more of an effect than subtracting, and other times, the opposite is true.

We suspect that these asymmetries and unexpected outcomes are due to two causes: interference between the feature vectors and the QK circuits, and attention head compositionality. The former means that even if adding a feature vector to a token increases the dot product of the output of the attention head's OV matrix with the observable, the added feature vector might cause the attention head to shift attention away from this token, thus counteracting the effect of adding the feature vector. The latter means that adding a feature vector to an attention head's input does not solely effect that attention head's output with respect to the logits – but rather, the inputs of later attention heads are also affected.

[5]There is nothing principled about multiplying the feature vector v by 50.



(a) An approximation of the dot product of MLP16's output with the 17.14 feature vector for n_{subj}.



(b) An approximation of the dot product of MLP16's output with the 17.14 feature vector for n_{obj}.

Figure 4: Approximations of MLP16's output, in both the n_{subj} and n_{obj} cases, with respect to the 17.14 feature vector.

6.10 Recursion and MLP feature vectors

As an initial test of our method's capacity to handle MLPs and virtual attention heads, we recursed down a few layers in the computational graph in order to obtain feature vectors corresponding to virtual attention heads and MLPs. Direct logit attribution (Nanda, 2022) suggested that the computational paths 13.3 $\to$ 17.14 and 13.3 $\to$ MLP16 $\to$ 17.14 were important for both n_{subj} and n_{obj}. We approximated MLP16 by choosing x_+ and x_- as follows. We used pre-MLP embeddings from the token to which direct logit attribution ascribed the most importance. For x_+, we used embeddings from a single female prompt; for x_-, we used embeddings from a single male prompt. Visualizations of our MLP16 approximations for the n_{subj} and n_{obj} cases can be found in Figure 4. Figure 4b in particular demonstrates the importance of taking the linear approximation at the point in between x_- and x_+ on the decision boundary, rather than at x_- or x_+ themselves: because the MLP had already saturated at x_-, taking the gradient there would have been grossly inaccurate.

We obtained the feature vectors for 13.3 $\to$ 17.14 and 13.3 $\to$ MLP16 $\to$ 17.14 for both n_{subj} and n_{obj}; we then computed the cosine similarities between these vectors.

What we found was that while the cosine similarity between the feature vectors for 13.3 for n_{subj} and n_{obj} was only 0.3712, the cosine similarity between the feature vectors for 13.3 $\to$ 17.14 for n_{subj} and n_{obj} was 0.9965 – almost 1! This strongly indicates the importance of looking at virtual attention heads.

We also measured the cosine similarity between the 13.3 $\to$ 17.14 feature vectors and the 13.3 $\to$ MLP16 $\to$ 17.14 feature vectors. For n_{subj}, the cosine similarity between these feature vectors was 0.9457, indicating that MLP16 acted largely as the identity for the n_{subj} case. But things were different in the n_{obj} case: there, the cosine similarity was only 0.5423, indicating that the MLP was playing a larger role in transforming the output. (This might be what we see reflected in the more nonlinear graph visible in Figure 4b.)

Finally, we measured the cosine similarity between the 13.3 $\to$ MLP16 $\to$ 17.14 feature vectors for n_{subj} and n_{obj}. We found that the cosine similarity was 0.6072. This is not a low value, but it is certainly not high either, further suggesting that the role played by MLP16 in the n_{obj} case is different from that of the n_{subj} case. However, in order to make any definitive claims, further experiments on much more data are required.

7 Preliminary gender bias experiments

7.1 Setting

We now demonstrate how our method can be used to find activation steering vectors that increase the model's output on one observable while decreasing it on another. As a case study, we consider the setting of occupational gender bias. We consider the prompt template `"<|endoftext|>My friend [NAME] is an excellent"`, where `[NAME]` is replaced by the names listed in 6.3. Now, if the name in the prompt is female, then we would expect the model to predict that the next token is more likely to be `" actress"` than `" actor"` – the two tokens refer to the same occupation, but the former is grammatically female, while the latter is conventionally grammatically male. This is sensible behavior from the language model.

However, the model also predicts that for a female name, the next token is more likely to be `" nurse"` than `" programmer"`. In this case, we say that the model is biased, because it is making a prediction on the basis of gender, even though the difference between the occupations `" nurse"` and `" programmer"` is not one of grammatical gender.

This motivates us to consider the observables $n_{\text{same}} = e_{\texttt{" actress"}} - e_{\texttt{" actor"}}$ and $n_{\text{diff}} = e_{\texttt{" nurse"}} - e_{\texttt{" programmer"}}$. The goal is to find feature vectors for n_{same} that are not feature vectors for n_{diff}, and vice-versa. If this is the case, then we can potentially steer the model to be less biased in the n_{diff} case, while retaining the desired behavior in the n_{same} case.

7.2 Confirming bias

In order to confirm that the model is actually biased on this task, we ran the model on the full set of prompts, recorded the mean difference in the model's outputs for n_{same} between female prompts and male prompts, and recorded the mean difference in the model's outputs for n_{diff} between female prompts and male prompts.

For the n_{same} case, the model displayed a mean logit difference of 5.6809 logits. In other words, the mean outputs with respect to n_{same} on the female inputs and with respect to $-n_{\text{same}}$ on the male inputs was 5.6809 logits. This reflects expected behavior of the model: it makes sense for the model to be more likely to predict `" actress"` than `" actor"` for a female name, and vice versa for a male name.

On the other hand, for the n_{diff} case, the model displayed a mean logit difference of 4.5169 logits. This means that if all other tokens had logits of $-\infty$ (i.e. were irrelevant), then on average, the model would predict that a female name is $e^{4.5169} \approx 91.55$ times more likely to correspond to the token `" nurse"` than the token `" programmer"`, and vice versa for a male name.

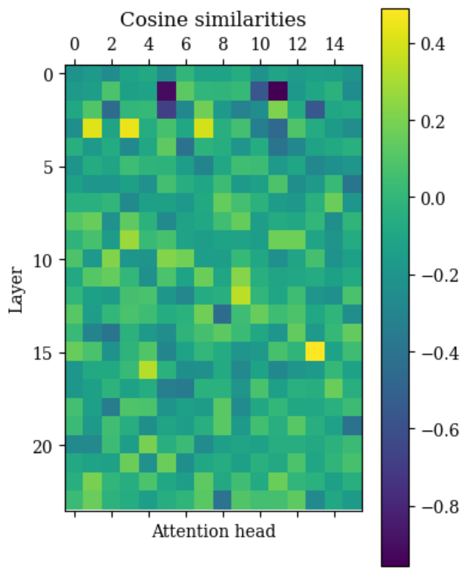
7.3 Cosine similarities and coupling coefficients

In order to investigate the cause of this, we examined the cosine similarities between the feature vectors for the n_{same} and n_{diff} cases. The results can be found in Figure 5a. Interestingly, the highest cosine similarity among these vectors is 0.4896 for head 15.13 – not high at all. In contrast, there are many heads with cosine similarities less than -0.75.

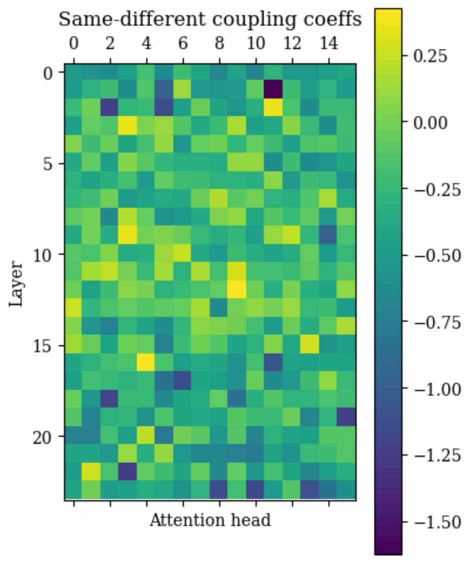
However, upon further investigation, we found that there were some feature vectors corresponding to virtual attention heads with high cosine similarity across the n_{same} and n_{diff} cases. As one example, the feature vector for n_{same} corresponding to virtual head $6.6 \to 15.13 \to 17.14$ had a cosine similarity of 0.9541 with the feature vector for n_{diff} corresponding to virtual head $6.6 \to 15.13 \to 21.13$. Note that the cosine similarity for head 6.6, without considering any attention head composition, is -0.0640.

What this indicates is that some of the same underlying features are being used by the model in both the n_{same} and n_{diff} cases – but these features are then mediated in different ways, depending on the observable. This is rather interesting, as it gives us a glimpse of how the same information is transformed throughout the model's computation.

Additionally, we looked at the coupling coefficients from feature vectors for n_{same} to feature vectors for n_{diff}; the results can be found in Figure 5b.



(a) Cosine similarities between feature vectors for n_{same} and n_{diff}.



(b) Couplings from feature vectors for n_{same} to feature vectors for n_{diff}.

Figure 5: Cosine similarities and coupling coefficients between feature vectors for n_{same} and n_{diff}.

Head	Token	n_{diff} or n_{same}	Coupling	Feature multiplier	n_{diff} effect	n_{same} effect
17.6	5	n_{diff}	-0.5230	-300	-3.6119	2.7614
18.2	0	n_{diff}	-0.2298	-1000	-0.4604	0.4694
19.15	0	n_{diff}	-0.4957	-300	-0.8513	1.8351
23.10	4	n_{same}	-0.5665	-120	-1.2027	0.1038

Table 3: Results of activation steering for bias mitigation. The "Head" column denotes the attention head whose feature vector was being considered. "Token" denotes the token to which the steering vector was added. "n_{diff} or n_{same}" denotes whether the steering vector used was associated with n_{diff} or n_{same}. "Coupling" denotes the coupling coefficient from the n_{diff} feature vector to the n_{same} feature vector. "Feature multiplier" denotes the scalar by which the feature vector was multiplied before being added to the embeddings. "n_{diff} effect" and "n_{same} effect" denote the mean difference, in logits, between the clean and patched runs, when measured using n_{diff} and n_{same} respectively. Note that a more negative n_{diff} effect is good, because it means that the model is yielding a less gender-biased output; in contrast, a more positive n_{same} effect is good.

7.4 ACTIVATION STEERING FOR BIAS MITIGATION

Once we obtained the coupling coefficients for the feature vectors, we tried to find vectors with negative coupling coefficients. The hope was that using such a vector as an activation steering vector could decrease the model's logit difference for n_{diff} but increase it for n_{same}; this corresponds to decreasing biased behavior while increasing desired behavior.

We tried using the feature vectors from a number of different attention heads whose coupling coefficients were the most negative. Although not every head worked, we found a number of heads where adding the feature vectors for those heads did yield the desired effects. We evaluated activation steering results over the dataset described in 7.1, using the methodology described in 6.9. The results for these heads are given in Table 3.

In particular, when we subtracted -300 times the feature vector for n_{diff} for attention head 17.6 from the embeddings of token 5, this impressively led to a mean effect of -3.6119 logits for n_{diff} and 2.7614 logits for n_{same}. This means that after this vector was applied to the embeddings, *the average prediction of the biased output was* -3.6119 *logits less than before*, while the average prediction of the grammatically-correct output was 2.7614 logits more than before.

Note that some trial-and-error was involved in finding these heads, along with the ideal feature multipliers to use. This was because, as can be seen in the results, the effect of the steering vectors on the different observables was not always in accordance with what the coupling coefficients predicted. This, in turn, is due to the various non-linearities inherent in the structure of the model; for further discussion, refer back to Section 6.9.

In future work, we plan to investigate more principled approaches to finding feature vectors that yield higher outputs for one given observable and lower outputs for another given observable. For now, though, we believe that this work is useful by making a first step towards demonstrating how interpretability methods can be used to directly address bias in large language models.

8 Discussion

We believe that we have demonstrated the utility of observable propagation for a variety of tasks, involving both the analysis of models and the manipulation of models. In our experiments, we found that looking at the norms of feature vectors obtained via observable propagation could be used to predict relevant attention heads for a task without actually running the model on any data; that observable propagation can be used to understand when two different tasks utilize the same feature; that coupling coefficients can be used to show the extent to which a high output for one observable implies a high output for another on a general distribution of data. From the perspective of wanting to learn more about the internal workings of large language models, I personally find this quite exciting. This method is a step towards not just knowing where in the language model processing is going on, but what processing is actually taking place.

But beyond merely the joy of knowledge, the preliminary results regarding activation steering for mitigating gender bias gives us hope that methods similar to observable propagation can be used to find steering vectors to cause models to yield desired outputs – where the observable paradigm allows for more precise definitions of what "desired outputs" might mean.

However, there is still a long way to go before mechanistic interpretability of large language models could be said to be "solved". It is thus worth noting the following limitations of observable propagation in its current form, so that we can better understand how to proceed forward.

The largest limitations of observable propagation have to do with the complexity inherent in the non-linearities of Transformers. To begin with, observable propagation only addresses the OV circuits of Transformers, ignoring computations involving QK circuits. As mentioned in Section 4.3.1, this means that observable propagation cannot capture the essence of "induction heads", which rely on QK circuits and are thought to be responsible for much in-context learning in Transformers (Elhage et al., 2021). Dealing with QK circuits in particular is quite difficult, because unlike MLPs and LayerNorms, QK circuits are nonlinearities that involve multiple tokens. In order for us to have a full understanding of Transformers, though, QK circuits must be tackled: there cannot be any *ignorabimus*. Additionally, it remains to be seen whether the linear approximation approach for extending observable propagation to work with MLPs can yield a tractable set of features corresponding to different nearly-linear subregions in the MLP (*a la* Black et al. (2022)), or whether the nonlinearities in MLPs are so great as to make such an effort infeasible. We plan to run more experiments on larger datasets, with a focus on MLP features, in order to answer this question.

There are some other limitations of observable propagation that are related to the definition of an observable as a linear functional over the logits. For example, the current formulation of observables does not support tasks that involve more than one token; this is problematic not only when attempting to deal with more complex tasks, but even when attempting to deal with simple tasks which involve multi-token words. Additionally, the observable formulation starts with a task and then finds feature vectors related to the task, but this requires a human to come up with a task to investigate in the first place. In future work, we plan to develop a method of automatically finding relevant tasks based on a dataset.

Despite these limitations, however, we believe that observable propagation represents an important first step towards deeply understanding the feature vectors that language models utilize in their computations, and how these feature vectors can be used to change the behavior of models. Given the power that the current formulation of observable propagation has demonstrated already in our case studies, we are very excited about the potential for this method, and methods building upon it, to yield even greater insights in the near future.

REFERENCES

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. Leace: Perfect linear concept erasure in closed form. *arXiv preprint arXiv:2306.03819*, 2023.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large scale autoregressive language modeling with mesh-tensorflow, 2021. URL `http://github.com/eleutherai/gpt-neo`.

Sid Black, Lee Sharkey, Leo Grinsztajn, Eric Winsor, Dan Braun, Jacob Merizian, Kip Parker, Carlos Ramón Guevara, Beren Millidge, Gabriel Alfour, and Connor Leahy. Interpreting neural networks through the polytope lens, 2022. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru`.

Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*, 2023.

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, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*, 2023.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing, 2023.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model, 2023.

Chris Mathwin, Guillaume Corlouer, Esben Kran, Fazl Barez, and Neel Nanda. Identifying a preliminary circuit for predicting gendered pronouns in gpt-2 small. *Apart Research Alignment Jam 4 (Mechanistic Interpretability)*, 2023. URL `https://cmathw.itch.io/identifying-a-preliminary-circuit-for-predicting-gendered-pronouns-in-gpt-2-smal`.

Joseph Miller and Clement Neo. We found an neuron in gpt-2. 2023. URL `https://www.lesswrong.com/posts/cgqh99SHsCv3jJYDS/we-found-an-neuron-in-gpt-2`.

Beren Millidge and Sid Black. The singular value decompositions of transformer weight matrices are highly interpretable. 2023. URL `www.lesswrong.com/posts/mkbGjzxD8d8XqKHzA/the-singular-value-decompositions-of-transformer-weight`.

Neel Nanda. A comprehensive mechanistic interpretability explainer glossary, 2022. URL `https://www.neelnanda.io/mechanistic-interpretability/glossary`.

Neel Nanda, Chris Olah, Catherine Olsson, Nelson Elhage, and Hume Tristan. Attribution patching: Activation patching at industrial scale. 2023. URL `https://www.neelnanda.io/mechanistic-interpretability/attribution-patching`.

nostalgebraist. Interpreting gpt: the logit lens, 2020. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru`.

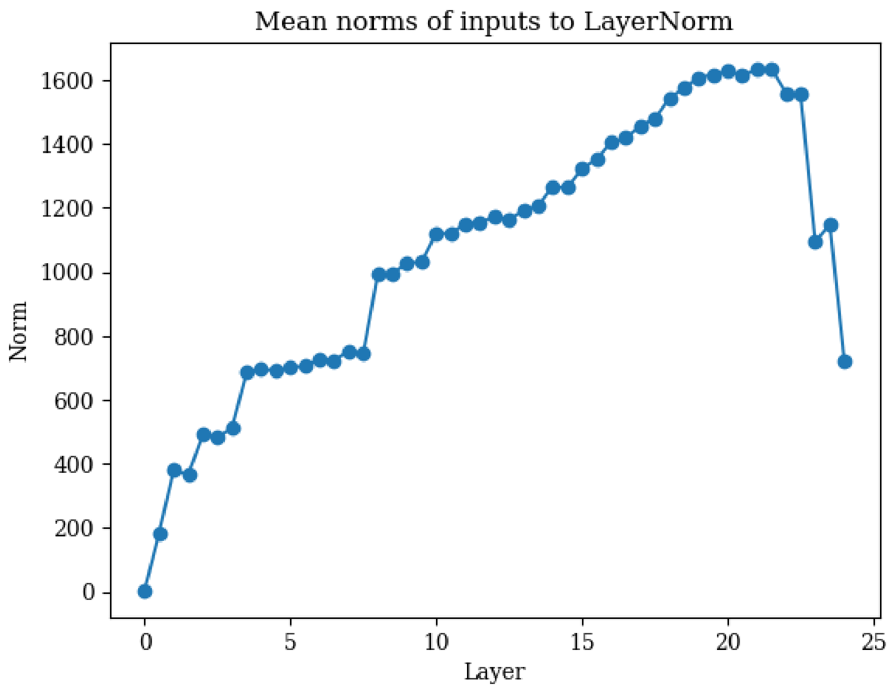


Figure 6: Mean norms per layer at each LayerNorm

Chris Olah. Mechanistic interpretability, variables, and the importance of interpretable bases, 2022. URL `https://transformer-circuits.pub/2022/mech-interp-essay/index.html`.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

KB Petersen and MS Pedersen. The matrix cookbook, version 2012/11/15. *Technical Univ. Denmark, Kongens Lyngby, Denmark, Tech. Rep*, 3274, 2012.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Nina Rimsky and Evan Hubinger. Red-teaming language models via activation engineering, 2023. URL `https://www.lesswrong.com/posts/iHmsJdxgMEWmAfNne`.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022.

A EMPIRICAL LAYERNORM GRADIENT INVESTIGATIONS

In this section, we put forth various empirical results relevant for the discussion of LayerNorm gradients in Section 4.3.3.

A.1 LAYERNORM INPUT NORMS PER LAYER

We calculated the average norms of inputs to each LayerNorm sublayer in the model, over all of the embeddings obtained from the prompts described in Section 6.3. The results can be found in Figure 6. The wide variation in the input norms across different layers implies that input norms must be taken into account in any approximation of LayerNorm gradients.

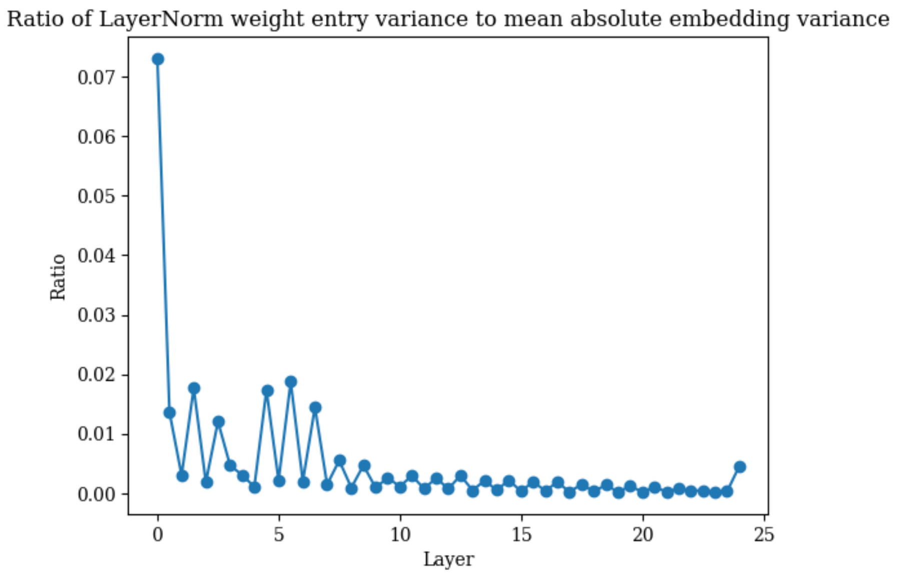


Figure 7: Ratio between LayerNorm weight matrix variances and mean absolute entries of each layer's embeddings

A.2 LayerNorm weight values are very similar

In Section 4.3.3, we state that the entries in the LayerNorm scaling matrices tend to be very close together, and use this as justification for treating weight matrices as scalars. Specifically, we found that the average variance of scaling matrix entries across all LayerNorms in GPT-Neo-1.3B is 0.007827. To determine the extent to which this variance is large, we calculated the ratio of the variance of each LayerNorm's weight matrix's entries to the mean absolute value of each layer's embeddings' entries. The results can be found in Figure 7. Note that the highest value found was 0.0731 at Layer 0 – meaning that the average entry in Layer 0 embeddings was over 13.67 times larger than the variance between entries in that layer's `ln_1` LayerNorm weight. This supports our assertion that LayerNorm scaling matrices can be largely treated as constants.

One possible guess as to why this behavior might be occurring is this: much of the computation taking place in the model does not occur with respect to basis directions in embedding space. However, the diagonal LayerNorm weight matrices can only act on these very basis directions. Therefore, the weight matrices end up "settling" on the same nearly-constant value in all entries.

A.3 Gradient accuracy on diverse prompts

In order to investigate the degree to which the linear approximations induced by LayerNorm gradients were accurate on a diverse data distribution, we carried out the following experiment. We considered the n_{subj} observables described in Section 6.1, and constructed prompts by inputting the names given in Section 6.3 into prompt templates. The difference here is that now, we considered three different prompt templates for n_{subj} instead of just one.

The n_{subj} prompt templates are as follows:

- `"<|endoftext|>So, [NAME] really is a great friend, isn't"`
- `"<|endoftext|>Man, [NAME] is so funny, isn't"`
- `"<|endoftext|>Really, [NAME] always works so hard, doesn't"`

As a result, we had three groups of twenty prompts each: ten for male names and ten for female names in each group.

Source prompt group	Prompt group 1	Prompt group 2	Prompt group 3
1	0.0500	0.2947	0.2519
2	0.1448	0.0949	0.1313
3	0.1803	0.0565	0.0545

Table 4: Mean absolute error, in logits, when the pre-unembedding-LayerNorm approximation from each source prompt group is used to approximate outputs from each destination prompt group.

We ran the model on all prompts and recorded the embeddings. Then, we use the procedure explained in Section 4.3.4 to find linear approximations for the pre-unembedding LayerNorm. We did the following for each prompt group. Using the terminology from Section 4.3.4, we calculated x_+ by taking the mean of the last-token pre-unembedding LayerNorm embeddings from all of the female prompts in the prompt group; we calculated x_- in the same way, using all of the male prompts in the prompt group. The resulting point at which we calculated our linear approximation thus is different for each prompt group.

The question was now thus: to what extent are the linear approximations from one prompt group accurate when applied to inputs from other prompt groups? For each prompt group, we calculated the mean absolute error in logits between the linear approximation for that prompt group and the model's actual output, when run over inputs from each of the three prompt groups. The results are given in Table 4. As expected, each prompt group's approximation is best at approximating inputs from its own prompt group. Interestingly, we do see asymmetries between prompt groups – for instance, prompt group 2 approximates prompt group 3 much worse than prompt group 3 approximates prompt group 2.

It is clear from this experiment that LayerNorm linear approximations are not uniformly accurate on all data. But the extent to which this matters depends on the degree of inaccuracy that can be tolerated.

B PROOF OF THEOREM 1

Theorem 1. *Define* $f(x;n) = n \cdot \mathrm{LayerNorm}(x)$. *Define*

$$\theta(x;n) = \arccos\left(\frac{n \cdot \nabla_x f(x;n)}{\|n\| \, \|\nabla_x f(x;n)\|}\right)$$

– that is, $\theta(x;n)$ *is the angle between* n *and* $\nabla_x f(x;n)$. *Then if* x *and* n *are i.i.d.* $\mathcal{N}(0,I)$ *in* $\mathbb{R}^d$*, and* $d \geq 8$ *then*

$$\mathbb{E}\left[\theta(x;n)\right] < 2\arccos\left(\sqrt{1 - \frac{1}{d-1}}\right)$$

To prove this, we will introduce a lemma:

Lemma 1. *Let* y *be an arbitrary vector. Let* $A = I - \frac{vv^T}{\|v\|^2}$ *be the orthogonal projection onto the hyperplane normal to* v*. Then the cosine similarity between* y *and* Ay *is given by* $\sqrt{1 - \cos(\theta)^2}$*, where* $\cos(\theta)$ *is the cosine similarity between* y *and* v*.*

Proof. Assume without loss of generality that y is a unit vector. (Otherwise, we could rescale it without affecting the angle between y and v, or the angle between y and Ay.)

We have $Ay = y - \frac{y \cdot v}{\|v\|^2} v$. Then,

$$\begin{aligned} y \cdot Ay &= y \cdot (y - \frac{y \cdot v}{\|v\|^2} v) \\ &= \|y\|^2 - \frac{(y \cdot v)^2}{\|v\|^2} \\ &= 1 - \frac{(y \cdot v)^2}{\|v\|^2} \end{aligned}$$

and

$$\begin{aligned} \|Ay\|^2 &= (y - \frac{y \cdot v}{\|v\|^2} v) \cdot (y - \frac{y \cdot v}{\|v\|^2} v) \\ &= y \cdot (y - \frac{y \cdot v}{\|v\|^2} v) - \frac{y \cdot v}{\|v\|^2} v \cdot (y - \frac{y \cdot v}{\|v\|^2} v) \\ &= y \cdot Ay - \frac{y \cdot v}{\|v\|^2} v \cdot (y - \frac{y \cdot v}{\|v\|^2} v) \\ &= y \cdot Ay - \frac{(y \cdot v)^2}{\|v\|^2} + \left\| \frac{y \cdot v}{\|v\|^2} v \right\|^2 \\ &= y \cdot Ay - \frac{(y \cdot v)^2}{\|v\|^2} + \frac{(y \cdot v)^2}{\|v\|^4} \|v\|^2 \\ &= y \cdot Ay - \frac{(y \cdot v)^2}{\|v\|^2} + \frac{(y \cdot v)^2}{\|v\|^2} \\ &= y \cdot Ay \end{aligned}$$

Now, the cosine similarity between y and Ay is given by

$$\begin{aligned} \frac{y \cdot Ay}{\|y\|\|Ay\|} &= \frac{y \cdot Ay}{\|Ay\|} \\ &= \frac{\|Ay\|^2}{\|Ay\|} \\ &= \|Ay\| \end{aligned}$$

At this point, note that $\|Ay\| = \sqrt{y \cdot Ay} = \sqrt{1 - \frac{(y \cdot v)^2}{\|v\|^2}}$. But $\frac{y \cdot v}{\|v\|}$ is just the cosine similarity between y and v. Now, if we denote the angle between y and v by θ, we thus have

$$\|Ay\| = \sqrt{1 - \frac{(y \cdot v)^2}{\|v\|^2}} = \sqrt{1 - \cos(\theta)^2}.$$

□

Now, we are ready to prove Theorem 1.

Proof. First, observe that $\mathrm{LayerNorm}(x) = \frac{Px}{\|Px\|}$, where $P = I - \frac{1}{d}\vec{1}\vec{1}^T$ is the orthogonal projection onto the hyperplane normal to $\vec{1}$, the vector of all ones. Thus, we have

$$f(x; n) = n^T \left(\frac{Px}{\|Px\|} \right)$$

Using the multivariate chain rule along with the rule that the derivative of $\frac{x}{\|x\|}$ is given by $\frac{I}{\|x\|} - \frac{xx^T}{\|x\|^3}$ (see section 2.6.1 of Petersen & Pedersen (2012)), we thus have that

$$\begin{aligned}\nabla_x f(x;n) &= \left(n^T \left(\frac{I}{\|Px\|} - \frac{(Px)(Px)^T}{\|Px\|^3}\right) P\right)^T \\ &= \left(\frac{1}{\|Px\|} n^T \left(I - \frac{(Px)(Px)^T}{\|Px\|^2}\right) P\right)^T \\ &= \frac{1}{\|Px\|} P \left(I - \frac{(Px)(Px)^T}{\|Px\|^2}\right) n \qquad \text{because } P \text{ is symmetric}\end{aligned}$$

Denote $Q = I - \frac{(Px)(Px)^T}{\|Px\|^2}$. Note that this is an orthogonal projection onto the hyperplane normal to Px. We now have that $\nabla_x f(x;n) = \frac{1}{\|Px\|} PQn$. Because we only care about the angle between n and $\nabla_x f(x;n)$, it suffices to look at the angle between n and PQn, ignoring the $\frac{1}{\|Px\|}$ term.

Denote the angle between n and PQn as $\theta(x,n)$. (Note that θ is also a function of x because Q is a function of x.) Then if $\theta_Q(x,n)$ is the angle between n and Qn, and $\theta_P(x,n)$ is the angle between Qn and PQn, then $\theta(x,n) \le \theta_Q(x,n) + \theta_P(x,n)$, so $\mathbb{E}[\theta(x,n)] \le \mathbb{E}[\theta_Q(x,n)] + \mathbb{E}[\theta_P(x,n)]$.

Using Lemma 1, we have that $\theta_Q(x,n) = \arccos\left(\sqrt{1-\cos(\phi(n,Px))^2}\right)$, where $\phi(n,Px)$ is the angle between n and Px. Now, because $n \sim \mathcal{N}(0,I)$, we have $\mathbb{E}[\cos(\phi(n,Px))^2] = 1/d$, using the well-known fact that the expected squared dot product between a uniformly distributed unit vector in $\mathbb{R}^d$ and a given unit vector in $\mathbb{R}^d$ is $1/d$.

At this point, define $g(t) = \arccos\left(\sqrt{1-t}\right)$, $h(t) = g'\left(\frac{1}{d-1}\right)\left(t - \frac{1}{d-1}\right) + g\left(\frac{1}{d-1}\right)$. Then if $\frac{1}{d-1} < c$, where c is the least solution to $g'(c) = \frac{\pi - 2g(c)}{2(1-c)}$, then $h(t) \ge g(t)$. (Note that $g(t)$ is convex on $(0, 0.5]$ and concave on $[0.5, 1)$. Therefore, there are exactly two solutions to $g'(c) = \frac{\pi-2g(c)}{2(1-c)}$. The lesser of the two solutions is the value at which $g'(c)$ equals the slope of the line between $(c, g(c))$ and $(1, \pi/2)$ – the latter point being the maximum of g – at the same time that $g''(c) \ge 0$.) One can compute $c \approx 0.155241\ldots$, so if $d \ge 8$, then $1/(d-1) < c$ is satisfied, so $h(t) \ge g(t)$. Thus, we have the following inequality:

$$\begin{aligned}h(1/(d-1)) &> h(1/d) \\ &= h(\mathbb{E}[\cos(\phi(n,Px))^2]) \\ &= \mathbb{E}[h(\cos(\phi(n,Px))^2)] \text{ due to linearity} \\ &\ge \mathbb{E}[g(\cos(\phi(n,Px))^2)] \text{ because } h(t) \ge g(t) \text{ for all } t \\ &= \mathbb{E}[\theta_Q(x,n)]\end{aligned}$$

Now, $h(1/(d-1)) = g(1/(d-1)) = \arccos\left(\sqrt{1-\frac{1}{d-1}}\right)$. Thus, we have that $\arccos\left(\sqrt{1-\frac{1}{d-1}}\right) > \mathbb{E}[\theta_Q(x,n)]$.

The next step is to determine an upper bound for $\mathbb{E}[\theta_P(x,n)]$. By Lemma 1, we have that $\theta_P(x,n) = \arccos\left(\sqrt{1-\cos(\phi(Qn,\vec{1}))^2}\right)$. Now, note that because $n \sim \mathcal{N}(0,I)$, then Qn is distributed according to a unit Gaussian in $\operatorname{Im} Q$, the $(d-1)$-dimensional hyperplane orthogonal to Px. Note that because $\vec{1}$ is orthogonal to Px (by the definition of P) and Px is orthogonal to $\operatorname{Im} Q$, this means that $\vec{1} \in \operatorname{Im} Q$. Now, let us apply the same fact from earlier: that the expected squared dot product between a uniformly distributed unit vector in $\mathbb{R}^{d-1}$ and a given unit vector in $\mathbb{R}^{d-1}$ is $1/(d-1)$. Thus, we have that $\mathbb{E}[\cos(\phi(Qn,\vec{1}))^2] = 1/(d-1)$.

From this, by the same logic as in the previous case, $\arccos\left(\sqrt{1-\frac{1}{d-1}}\right) \ge \mathbb{E}[\theta_P(x,n)]$.

Adding this inequality to the inequality for $\mathbb{E}[\theta_Q(x,n)]$, we have

$$2\arccos\left(\sqrt{1-\frac{1}{d-1}}\right) > \mathbb{E}[\theta_Q(x,n)] + \mathbb{E}[\theta_P(x,n)] \geq \mathbb{E}[\theta(x,n)]$$

. □

C PROOF OF THEOREM 2

Theorem 2. *Let $y_1, y_2 \in \mathbb{R}^d$. Let x be uniformly distributed on the hypersphere defined by the constraints $\|x\| = s$ and $x \cdot y_1 = k$. Then we have*

$$\mathbb{E}[x \cdot y_2] = k\frac{y_1 \cdot y_2}{\|y_1\|^2}$$

and the maximum and minimum values of $x \cdot y_2$ are given by

$$\frac{\|y_2\|}{\|y_1\|}\left(k\cos(\theta) \pm \sin(\theta)\sqrt{s^2\|y_1\|^2 - k^2}\right)$$

where θ is the angle between y_1 and y_2.

Before proving Theorem 2, we will prove a quick lemma.

Lemma 2. *Let $\mathcal{S}$ be a hypersphere with radius r and center c. Then for a given vector y, the mean squared distance from y to the sphere, $\mathbb{E}_{s\in\mathcal{S}}[\|y-c\|^2]$, is given by $\|y-c\|^2 + r^2$.*

Proof. Without loss of generality, assume that $\mathcal{S}$ is centered at the origin (so $\|y-c\|^2 = \|y\|^2$). Induct on the dimension of the $\mathcal{S}$. As our base case, let $\mathcal{S}$ be the 0-sphere consisting of a point in $\mathbb{R}^1$ at $-r$ and a point at r. Then $\mathbb{E}_{s\in\mathcal{S}}[|y-s|^2] = \frac{(y-r)^2+(y-(-r))^2}{2} = y^2 + r^2$.

For our inductive step, assume the inductive hypothesis for spheres of dimension $d-2$; we will prove the theorem of spheres of dimension $d-1$ in an ambient space of dimension d. Without loss of generality, let y lie on the x-axis, so that we have $y = [y_1 \quad 0 \quad 0 \quad \ldots]^T$. Next, divide $\mathcal{S}$ into slices along the x-axis. Denote the slice at position $x = x_0$ as $\mathcal{S}_{x_0}$. Then $\mathcal{S}_{x_0}$ is a $(d-2)$-sphere centered at $[x_0 \quad 0 \quad 0 \quad \ldots]^T$, and has radius $\sqrt{r^2 - x_0^2}$. Now, by the law of total expectation, $\mathbb{E}_{s\in\mathcal{S}}[\|y-s\|^2] = \mathbb{E}_{-r\leq x\leq r}\left[\mathbb{E}_{s'\in\mathcal{S}_x}\left[\|y-s'\|^2\right]\right]$. We then have that

$$\begin{aligned}\mathbb{E}_{s'\in\mathcal{S}_x}\left[\|y-s'\|^2\right] &= \mathbb{E}\left[(y_1-x)^2 + s_2^2 + s_3^2 + \cdots\right] \\ &= (y_1-x)^2 + \mathbb{E}\left[s_2^2 + s_3^2 + \cdots\right]\end{aligned}$$

Once again, $\mathcal{S}_x$ is a $(d-2)$-sphere defined by $s_2^2 + s_3^2 + \cdots = r^2 - x^2$. This means that by the inductive hypothesis, we have $\mathbb{E}\left[s_2^2 + s_3^2 + \cdots\right] = r^2 - x^2$. Thus, we have

$$\begin{aligned}\mathbb{E}_{s'\in\mathcal{S}_x}\left[\|y-s'\|^2\right] &= (y_1-x)^2 + r^2 - x^2 \\ \mathbb{E}_{s'\in\mathcal{S}_x}\left[\|y-s'\|^2\right] &= (y_1-x)^2 + r^2 - x^2 \\ \mathbb{E}_{s\in\mathcal{S}}[\|y-s\|^2] &= \mathbb{E}_{-r\leq x\leq r}\left[(y_1-x)^2 + r^2 - x^2\right] \\ &= \frac{1}{2r}\int_{-r}^{r}(y_1-x)^2 + r^2 - x^2 dx \\ &= r^2 + y_1^2\end{aligned}$$

□

We are now ready to begin the main proof.

Proof. First, assume that $\|x\| = 1$. Now, the intersection of the $(d-1)$-sphere defined by $\|x\| = 1$ and the hyperplane $x \cdot y_1 = k$ is a unit hypersphere of dimension $(d-2)$, oriented in the hyperplane $x \cdot y_1 = k$, and centered at $c_1 y_1$ where $c_1 = k / \|y_1\|^2$. Denote this $(d-2)$-sphere as $\mathcal{S}$, and denote its radius by r.

Next, define $c_2 = \frac{k}{y_2 \cdot y_1}$. Then $cy_2 \cdot y_1 = k$, so $c_2 y_2$ lies in the same hyperplane as $\mathcal{S}$. Additionally, because $c_1 y_1$ is in this hyperplane, and $c_1 y_1$ is also the normal vector for this hyperplane, we have that the vectors $c_1 y_1$, $c_2 y_2$, and $c_1 y_1 - c_2 y_2$ form a right triangle, where $c_2 y_2$ is the hypotenuse and $c_1 y_1 - c_2 y_2$ is the leg opposite of the angle θ between y_1 and y_2. As such, we have that $\|c_1 y_1 - c_2 y_2\| = \sin(\theta) \|c_2 y_2\|$.

Furthermore, we have that $c_1 y_1 \cdot c_2 y_2 = \frac{k^2}{\|y_1\|^2}$, that $\|c_1 y_1\| = \frac{|k|}{\|y_1\|^2}$, and that $\|c_2 y_2\| = \frac{|k|}{\|y_1\| |\cos\theta|}$

We will now begin to prove that the maximum and minimum values of $y_2 \cdot x$ are given by $\frac{\|y_2\|}{\|y_1\|}\left(k\cos(\theta) \pm |\sin(\theta)|\sqrt{s^2\|y_1\|^2 - k^2}\right)$.

To start, note that the nearest point on $\mathcal{S}$ to $c_2 y_2$ and the farthest point on $\mathcal{S}$ from $c_2 y_2$ are located at the intersection of $\mathcal{S}$ with the line between $c_2 y_2$ and $c_1 y_1$.

To see this, let x_+ be the at the intersection of $\mathcal{S}$ and the line between $c_2 y_2$ and $c_1 y_1$. We will show that x_+ is the nearest point on $\mathcal{S}$ to $c_2 y_2$. Let $x'_+ \in \mathcal{S} \neq x_+$. Then we have the following cases:

- Case 1: $c_2 y_2$ is outside of $\mathcal{S}$. Then $\|c_2 y_2 - c_1 y_1\| = \|c_2 y_2 - x_+\| + \|x_+ - c_1 y_1\|$, because $c_2 y_2$, x_+, and $c_1 y_1$ are collinear – so $\|c_2 y_2 - c_1 y_1\| = \|c_2 y_2 - x_+\| + r$ (because $x_+ \in \mathcal{S}$). By the triangle inequality, we have $\|c_2 y_2 - c_1 y_1\| \le \|c_2 y_2 - x'_+\| + \|x'_+ - c_1 y_1\| = \|c_2 y_2 - x'_+\| + r$. But this means that $\|c_2 y_2 - x_+\| \le \|c_2 y_2 - x'_+\|$.
- Case 2: $c_2 y_2$ is inside of $\mathcal{S}$. Then $\|c_2 y_2 - c_1 y_1\| = \|x_+ - c_1 y_1\| - \|c_2 y_2 - x_+\|$, because $c_2 y_2$, x_+, and $c_1 y_1$ are collinear – so $\|c_2 y_2 - c_1 y_1\| = r - \|c_2 y_2 - x_+\|$. By the triangle inequality, we have $\|x'_+ - c_1 y_1\| \le \|c_2 y_2 - x'_+\| + \|c_2 y_2 - c_1 y_1\|$, so $\|x'_+ - c_1 y_1\| \le \|c_2 y_2 - x'_+\| + r - \|c_2 y_2 - x_+\|$. But since $\|x'_+ - c_1 y_1\| = r$, this means that $\|c_2 y_2 - x_+\| \le \|c_2 y_2 - x'_+\|$.

A similar argument will show that x_-, the farthest point on $\mathcal{S}$ from $c_2 y_2$, is also located at the intersection of $\mathcal{S}$ with the line between $c_2 y_2$ and $c_1 y_1$.

Now, let us find the values of x_+ and x_-. The line between $c_2 y_2$ and $c_1 y_1$ can be parameterized by a scalar t as $c_1 y_1 + t(c_2 y_2 - c_1 y_1)$. Then x_+ and x_- are given by $c_1 y_1 + t^*(c_2 y_2 - c_1 y_1)$, where t^* are the solutions to the equation $\|c_1 y_1 + t(c_2 y_2 - c_1 y_1)\| = 1$.

We have the following:

$$\begin{aligned}
1 &= \|c_1 y_1 + t(c_2 y_2 - c_1 y_1)\| \\
&= \|c_1 y_1\|^2 + 2t(c_1 y_1 \cdot (c_2 y_2 - c_1 y_1)) + t^2 \|c_2 y_2 - c_1 y_1\|^2 \\
&= \|c_1 y_1\|^2 + 2t((c_1 y_1 \cdot c_2 y_2) - \|c_1 y_1\|^2) + t^2 \|c_2 y_2\|^2 \sin^2\theta \\
&= \frac{k^2}{\|y_1\|^2} + 2t\left(\frac{k^2}{\|y_1\|^2} - \frac{k^2}{\|y_1\|^2}\right) + t^2 \frac{k^2}{\|y_1\|^2 \cos^2\theta} \sin^2\theta \\
&= \frac{k^2}{\|y_1\|^2}(t^2 \tan^2\theta + 1)
\end{aligned}$$

Thus, solving for t, we have that $t^* = \frac{\pm\sqrt{\|y_1\|^2 - k^2}}{|k| \tan\theta}$. Therefore, we have that

$$\begin{aligned}
x_+, x_- &= c_1 y_1 + t^*(c_2 y_2 - c_1 y_1) \\
&= c_1 y_1 + \left(\frac{k^2}{\|y_1\|^2} (t^2 \tan^2 \theta + 1) \right) (c_2 y_2 - c_1 y_1) \\
&= \frac{k y_1}{\|y_1\|^2} + \left(\frac{\pm\sqrt{\|y_1\|^2 - k^2}}{|k| \tan\theta} \right) \left(\frac{k y_2}{y_1 \cdot y_2} - \frac{k y_1}{\|y_1\|^2} \right) \\
&= k \left[\frac{y_1}{\|y_1\|^2} \pm \left(\frac{\sqrt{\|y_1\|^2 - k^2}}{|k| \tan\theta} \right) \left(\frac{y_2}{y_1 \cdot y_2} - \frac{y_1}{\|y_1\|^2} \right) \right] \\
y_2 \cdot x_+, y_2 \cdot x_- &= y_2 \cdot k \left[\frac{y_1}{\|y_1\|^2} \pm \left(\frac{\sqrt{\|y_1\|^2 - k^2}}{|k| \tan\theta} \right) \left(\frac{y_2}{y_1 \cdot y_2} - \frac{y_1}{\|y_1\|^2} \right) \right] \\
&= \frac{k y_1 \cdot y_2}{\|y_1\|^2} \pm \left(\cot\theta \sqrt{\|y_1\|^2 - k^2} \right) \left(\frac{y_2 \cdot y_2}{y_1 \cdot y_2} - \frac{y_1 \cdot y_1}{\|y_1\|^2} \right) \\
&= \frac{k y_1 \cdot y_2}{\|y_1\|^2} \pm \left(\cot\theta \sqrt{\|y_1\|^2 - k^2} \right) \left(\frac{\|y_2\|}{\|y_1\| \cos\theta} - \frac{\|y_2\| \cos\theta}{\|y_1\|} \right) \\
&= \left[\frac{k y_1 \cdot y_2}{\|y_1\|^2} \pm \left(\cot\theta \sqrt{\|y_1\|^2 - k^2} \right) \frac{\|y_2\|}{\|y_1\|} \left(\frac{1}{\cos\theta} - \cos\theta \right) \right] \\
&= \frac{k y_1 \cdot y_2}{\|y_1\|^2} \pm \left(\cot\theta \sqrt{\|y_1\|^2 - k^2} \right) \frac{\|y_2\|}{\|y_1\|} \sin\theta \tan\theta \\
&= \frac{k y_1 \cdot y_2}{\|y_1\|^2} \pm \frac{\|y_2\|}{\|y_1\|} \sin\theta \sqrt{\|y_1\|^2 - k^2} \\
&= \frac{\|y_2\|}{\|y_1\|} \left(k \cos(\theta) \pm \sin(\theta) \sqrt{\|y_1\|^2 - k^2} \right)
\end{aligned}$$

We will now prove that $\mathbb{E}\left[y_2 \cdot x\right] = \frac{y_1 \cdot y_2}{\|y_1\|^2}$. Before we do, note that we can also use our value of t^* to determine the squared radius of $\mathcal{S}$. We have that the squared radius of $\mathcal{S}$ is given by

$$\begin{aligned}
r^2 &= \|t^*(c_2 y_2 - c_1 y_1)\|^2 \\
&= (t^*)^2 \|(c_2 y_2 - c_1 y_1)\|^2 \\
&= (t^*)^2 \sin^2\theta \|c_2 y_2\|^2 \\
&= \frac{\sin^2(\theta) k^2 / \left(\|y_1\|^2 \cos^2\theta \right)}{k^2 \tan\theta} \left(\|y_1\|^2 - k^2 \right) \\
&= 1 - \frac{k^2}{\|y_1\|^2}
\end{aligned}$$

We will use this result soon. Now, on to the main event. Begin by noting that $y_2 \cdot x = \|y_2\| \|x\| \cos(y_2, x) = \|y_2\| \cos(y_2, x)$, where $\cos(y_2, x)$ is the cosine of the angle between y_2 and x. Now, $\cos(y_2, x) = \text{signum}(c_2) \cos(c_2 y_2, x)$. And we have that $\|x - c y_2\|^2 = \|x\|^2 + \|c y_2\|^2 -$

$2\left\|x\right\|\left\|c_2y_2\right\|\cos(cy_2,x) = 1+\left\|c_2y_2\right\|^2 - 2\left\|c_2y_2\right\|\cos(c_2y_2,x)$. Therefore, we have

$$\begin{aligned}
\cos(y_2,x) &= \operatorname{signum}(c_2)\cos(c_2y_2,x)\\
&= \operatorname{signum}(c_2)\frac{\left\|x-c_2y_2\right\|^2 - 1 - \left\|c_2y_2\right\|^2}{-2\left\|c_2y_2\right\|}\\
&= \operatorname{signum}(c_2)\frac{1+\left\|c_2y_2\right\|^2 - \left\|x-c_2y_2\right\|^2}{2\left\|c_2y_2\right\|}
\end{aligned}$$

$$\begin{aligned}
y_2\cdot x &= \left\|y_2\right\|\cos(y_2,x)\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1+\left\|c_2y_2\right\|^2 - \left\|x-c_2y_2\right\|^2}{2\left\|c_2y_2\right\|}
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}\left[y_2\cdot x\right] &= \mathbb{E}\left[\operatorname{signum}(c_2)\left\|y_2\right\|\frac{1+\left\|c_2y_2\right\|^2 - \left\|x-c_2y_2\right\|^2}{2\left\|c_2y_2\right\|}\right]\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1+\left\|c_2y_2\right\|^2 - \mathbb{E}\left[\left\|x-c_2y_2\right\|^2\right]}{2\left\|c_2y_2\right\|}\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1+\left\|c_2y_2\right\|^2 - \left(1-\frac{k^2}{\left\|y_1\right\|^2}+\left\|c_1y_1-c_2y_2\right\|^2\right)}{2\left\|c_2y_2\right\|}
\end{aligned}$$

This last line uses Lemma 2: c_1y_1 is the center of $\mathcal{S}$, so the expected squared distance between c_2y_2 and a point on $\mathcal{S}$ is given by $1-\frac{k^2}{\left\|y_1\right\|^2}+\left\|c_1y_1-c_2y_2\right\|^2$, where $1-\frac{k^2}{\left\|y_1\right\|^2}$ is the squared radius of $\mathcal{S}$ and $\left\|c_1y_1-c_2y_2\right\|^2$ is the squared distance from c_2y_2 to the center. We can use this lemma because c_2y_2 is in the same hyperplane as $\mathcal{S}$, so we can treat this situation as being set in a space of dimension $d-1$.

Now, continue to simplify:

$$\begin{aligned}
\mathbb{E}\left[y_2\cdot x\right] &= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1+\left\|c_2y_2\right\|^2 - \left(1-\frac{k^2}{\left\|y_1\right\|^2}+\left\|c_1y_1-c_2y_2\right\|^2\right)}{2\left\|c_2y_2\right\|}\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{\left\|c_2y_2\right\|^2+\frac{k^2}{\left\|y_1\right\|^2}-\sin^2\theta\left\|c_2y_2\right\|^2}{2\left\|c_2y_2\right\|}\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{\left\|c_2y_2\right\|^2\cos^2\theta+\frac{k^2}{\left\|y_1\right\|^2}}{2\left\|c_2y_2\right\|}\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1}{2}\left(\left\|c_2y_2\right\|\cos^2\theta+\frac{|k|\cos\theta}{\left\|y_1\right\|}\right)\\
&= \operatorname{signum}(c_2)\left\|y_2\right\|\frac{1}{2}\left(\frac{|k|\left|\cos\theta\right|}{\left\|y_1\right\|}+\frac{|k|\left|\cos\theta\right|}{\left\|y_1\right\|}\right)\\
&= \operatorname{signum}(c_2)\,|k|\frac{\left\|y_2\right\|}{\left\|y_1\right\|}\left|\cos\theta\right|\\
&= k\frac{\left\|y_2\right\|}{\left\|y_1\right\|}\cos\theta\\
&= k\frac{y_1\cdot y_2}{\left\|y_1\right\|^2}
\end{aligned}$$

The last thing to do is to note that the above formulas are only valid when $\left\|x\right\|=1$. But if $\left\|x\right\|=s$, this is equivalent to the case when $\left\|x\right\|=1$ if we scale y_1 and y_2 by s. Scaling those two vectors by s gives us the final formulas in Theorem 2. $\square$