How to Dissect a Muppet: The Structure of Transformer Embedding Spaces

Timothee Mickus[*]
University of Helsinki, Finland
timothee.mickus
@helsinki.fi

Denis Paperno
Utrecht University,
The Netherlands
d.paperno@uu.nl

Mathieu Constant
Université de Lorraine,
CNRS,ATILF, France
Mathieu.Constant
@univ-lorraine.fr

Abstract

Pretrained embeddings based on the Transformer architecture have taken the NLP community by storm. We show that they can mathematically be reframed as a sum of vector factors and showcase how to use this reframing to study the impact of each component. We provide evidence that multi-head attentions and feed-forwards are not equally useful in all downstream applications, as well as a quantitative overview of the effects of finetuning on the overall embedding space. This approach allows us to draw connections to a wide range of previous studies, from vector space anisotropy to attention weights.

1 Introduction

The Transformer architecture (Vaswani et al., 2017) has taken the NLP community by storm. Based on the attention mechanism (Bahdanau et al., 2015; Luong et al., 2015), it was shown to outperform recurrent architectures on a wide variety of tasks. Another step was taken with pretrained language models derived from this architecture (BERT, Devlin et al., 2019, among others): they now embody the default approach to a vast swath of NLP applications. Success breeds scrutiny; likewise the popularity of these models has fostered research in explainable NLP interested in the behavior and explainability of pretrained language models (Rogers et al., 2020).

In this paper, we develop a novel decomposition of Transformer output embeddings. Our approach consists in quantifying the contribution of each network submodule to the output contextual embedding, and grouping those into four terms: (i) what relates to the input for a given position, (ii) what pertains to feed-forward submodules, (iii) what corresponds to multi-head attention, and (iv) what is due to vector biases.

This allows us to investigate Transformer embeddings without relying on attention weights or treating the entire model as a black box, as is most often done in the literature. The usefulness of our method is demonstrated on BERT: Our case study yields enlightening connections to state-of-the-art work on Transformer explainability, evidence that multi-head attentions and feed-forwards are not equally useful in all downstream applications, as well as an overview of the effects of finetuning on the embedding space. We also provide a simple and intuitive measurement of the importance of any term in this decomposition with respect to the whole embedding.

We will provide insights on the Transformer architecture in Section 2, and showcase how these insights can translate into experimental investigations in Sections 3 to 6. We will conclude with connections to other relevant works in Section 7 and discuss future perspectives in Section 8.

2 Additive Structure in Transformers

We show that the Transformer embedding $\mathbf{e}_t$ for a token t is as a sum of four *terms*:

$$\mathbf{e}_t = \mathbf{i}_t + \mathbf{h}_t + \mathbf{f}_t + \mathbf{c}_t \tag{1}$$

where $\mathbf{i}_t$ can be thought of as a classical static embedding, $\mathbf{f}_t$ and $\mathbf{h}_t$ are the cumulative contributions at every layer of the feed-forward submodules and the MHAs respectively, and $\mathbf{c}_t$ corresponds to biases accumulated across the model.

Equation (1) provides interpretable and quantifiable terms that can explain the behavior of specific components of the Transformer architecture. More precisely, it characterizes what is the impact of adding another sublayer on top of what was previously computed: the terms in Equation (1) are defined as sums across (sub)layers;

[*]The work described in the present paper was conducted chiefly while at ATILF.

Transactions of the Association for Computational Linguistics, vol. 10, pp. 981–996, 2022. https://doi.org/10.1162/tacl_a_00501
Action Editor: Dani Yogatama. Submission batch: 3/2022; Revision batch: 5/2022; Published 9/2022.

$\mathbf{A}$	matrix
$(\mathbf{A})_{t,\cdot}$	t^{th} row of $\mathbf{A}$
$\mathbf{a}$	(row) vector
a, α	scalars
$\mathbf{W}^{(\mathrm{M})}$	item linked to submodule M
$\mathbf{a}\oplus\mathbf{b}$	concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$
$\bigoplus\limits_{n}\mathbf{a}_n$	$\mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \cdots \oplus \mathbf{a}_n$
$\mathbf{a}\odot\mathbf{b}$	element-wise multiplication of $\mathbf{a}$ and $\mathbf{b}$
$\bigodot\limits_{n}\mathbf{a}_n$	$\mathbf{a}_1 \odot \mathbf{a}_2 \odot \cdots \odot \mathbf{a}_n$
$\vec{1}$	vector with all components set to 1
$\mathbf{0}_{m,n}$	null matrix of shape $m \times n$
$\mathbf{I}_n$	identity matrix of shape $n \times n$

Table 1: Notation.

hence we can track how a given sublayer transforms its input, and show that this effect can be thought of as adding another vector to a previous sum. This layer-wise sum of submodule outputs also allows us to provide a first estimate of which parameters are most relevant to the overall embedding space: a submodule whose output is systematically negligible has its parameters set so that its influence on subsequent computations is minimal.

The formulation in Equation (1) more generally relies on the additive structure of Transformer embedding spaces. We start by reviewing the Transformer architecture in Section 2.1, before discussing our decomposition in greater detail in Section 2.2 and known limitations in Section 2.3.

2.1 Transformer Encoder Architecture

Let's start by characterizing the Transformer architecture of Vaswani et al. (2017) in the notation described in Table 1.

Transformers are often defined using three hyperparameters: the number of layers L, the dimensionality of the hidden representations d, and H, the number of attention heads in multi-head attentions. Formally, a Transformer model is a stack of *sublayers*. A visual representation is shown in Figure 1. Two sublayers are stacked to form a single Transformer *layer*: The first corresponds to a multi-head attention mechanism (MHA), and the second to a feed-forward (FF). A Transformer with L layers contains $\Lambda = 2L$ sublayers. In Figure 1 two sublayers (in blue) are grouped into one layer, and L layers are stacked one after the other.

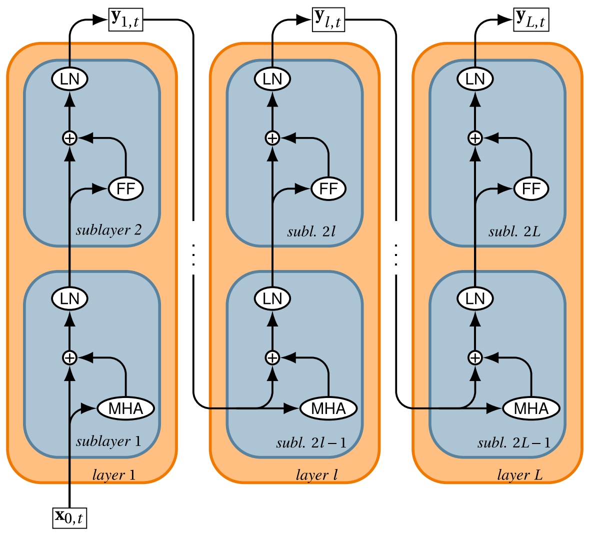


Figure 1: Overview of a Transformer encoder.

Each sublayer is centered around a specific sublayer function. Sublayer functions map an input $\mathbf{x}$ to an output $\mathbf{y}$, and can either be *feed-forward* submodules or *multi-head attention* submodules.

FFs are subnets of the form:

$$\mathbf{y}_t^{(\mathrm{FF})} = \phi\left(\mathbf{x}_t\mathbf{W}^{(\mathrm{FF,I})} + \mathbf{b}^{(\mathrm{FF,I})}\right)\mathbf{W}^{(\mathrm{FF,O})} + \mathbf{b}^{(\mathrm{FF,O})}$$

where ϕ is a non-linear function, such as ReLU or GELU (Hendrycks and Gimpel, 2016). Here, $^{(\ldots,\mathrm{I})}$ and $^{(\ldots,\mathrm{O})}$ distinguish the input and output linear projections, and the index t corresponds to the token position. Input and output dimensions are equal, whereas the intermediary layer dimension (i.e., the size of the hidden representations to which the non-linear function ϕ will be applied) is larger, typically of $b = 1024$ or 2048. In other words, $\mathbf{W}^{(\mathrm{FF,I})}$ is of shape $d \times b$, $\mathbf{b}^{(\mathrm{FF,I})}$ of size b, $\mathbf{W}^{(\mathrm{FF,O})}$ is of shape $b \times d$, and $\mathbf{b}^{(\mathrm{FF,O})}$ of size d.

MHAs are concatenations of scaled-dot *attention heads*:

$$\mathbf{y}_t^{(\mathrm{MHA})} = \left(\bigoplus_{h=1}^{H}(\mathbf{A}_h)_{t,\cdot}\right)\mathbf{W}^{(\mathrm{MHA,O})} + \mathbf{b}^{(\mathrm{MHA,O})}$$

where $(\mathbf{A}_h)_{t,\cdot}$ is the t^{th} row vector of the following $n \times d/H$ matrix $\mathbf{A}_h$:

$$\mathbf{A}_h = \mathrm{softmax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^T}{\sqrt{d/H}}\right)\mathbf{V}_h$$

with h an index tracking attention heads. The parameter matrix $\mathbf{W}^{(\mathrm{MHA,O})}$ of shape $d \times d$, and the bias $\mathbf{b}^{(\mathrm{MHA,O})}$ of size d. The queries $\mathbf{Q}_h$, keys $\mathbf{K}_h$ and values $\mathbf{V}_h$ are simple linear projections of shape $n \times (d/H)$, computed from all inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$(\mathbf{Q}_h)_{t,\cdot} = \mathbf{x}_t \mathbf{W}_h^{(\mathrm{Q})} + \mathbf{b}_h^{(\mathrm{Q})}$$
$$(\mathbf{K}_h)_{t,\cdot} = \mathbf{x}_t \mathbf{W}_h^{(\mathrm{K})} + \mathbf{b}_h^{(\mathrm{K})}$$
$$(\mathbf{V}_h)_{t,\cdot} = \mathbf{x}_t \mathbf{W}_h^{(\mathrm{V})} + \mathbf{b}_h^{(\mathrm{V})}$$

where the weight matrices $\mathbf{W}_h^{(\mathrm{Q})}$, $\mathbf{W}_h^{(\mathrm{K})}$ and $\mathbf{W}_h^{(\mathrm{V})}$ are of the shape $d \times (d/H)$, with H the number of attention heads, and biases $\mathbf{b}_h^{(\mathrm{Q})}$, $\mathbf{b}_h^{(\mathrm{K})}$ and $\mathbf{b}_h^{(\mathrm{V})}$ are of size d/H. This component is often analyzed in terms of *attention weights* α_h, which correspond to the softmax dot-product between keys and queries. In other words, the product $\mathrm{softmax}(\mathbf{Q}_h \mathbf{K}_h^T / \sqrt{d/H})$ can be thought of as $n \times n$ matrix of weights in an average over the transformed input vectors $\mathbf{x}_{t'} \mathbf{W}_h^{(\mathrm{V})} + \mathbf{b}_h^{(\mathrm{V})}$ (Kobayashi et al., 2020, Eqs. (1) to (4)): multiplying these weights with the value projection $\mathbf{V}_h$ yields a weighted sum of value projections:

$$(\mathbf{A}_h)_{t,\cdot} = \sum_{t'=1}^{n} \alpha_{h,t,t'} (\mathbf{V}_h)_{t',\cdot}$$

where $\alpha_{h,t,t'}$ is the component at row t and column t' of this attention weights matrix.

Lastly, after each sublayer function S, a *residual connection* and a *layer normalization* (LN, Ba et al., 2016) are applied:

$$\mathbf{y}_t^{(\mathrm{LN})} = \mathbf{g} \odot \left[\frac{S(\mathbf{x}_t) + \mathbf{x}_t - \left(m_t \cdot \vec{1} \right)}{s_t} \right] + \mathbf{b}^{(\mathrm{LN})}$$

The gain $\mathbf{g}$ and bias $\mathbf{b}^{(\mathrm{LN})}$ are learned parameters with d components each; $m_t \cdot \vec{1}$ is the vector $(1, \cdots, 1)$ scaled by the mean component value m_t of the input vector $S(\mathbf{x}_t) + \mathbf{x}_t$; s_t is the standard deviation of the component values of this input. As such, a LN performs a z-scaling, followed by the application of the gain $\mathbf{g}$ and the bias $\mathbf{b}^{(\mathrm{LN})}$.

To kick-start computations, a sequence of static vector representations $\mathbf{x}_{0,1} \ldots \mathbf{x}_{0,n}$ with d components each is fed into the first layer. This initial input corresponds to the sum of a static lookup word embedding and a positional encoding.[1]

[1] In BERT (Devlin et al., 2019), additional terms to this static input encode the segment a token belongs to, and a LN is added before the very first sublayer. Other variants also encode positions by means of an offset in attention heads (Huang et al., 2018; Shaw et al., 2018).

2.2 Mathematical Re-framing

We now turn to the decomposition proposed in Equation (1): $\mathbf{e}_t = \mathbf{i}_t + \mathbf{f}_t + \mathbf{h}_t + \mathbf{c}_t$.[2] We provide a derivation in Appendix A.

The term $\mathbf{i}_t$ corresponds to the input embedding (i.e., the positional encoding, the input word-type embedding, and the segment encoding in BERT-like models), after having gone through all the LN gains and rescaling:

$$\mathbf{i}_t = \frac{\bigodot_{\lambda=1}^{\Lambda} \mathbf{g}_\lambda}{\prod_{\lambda=1}^{\Lambda} s_{\lambda,t}} \odot \mathbf{x}_{0,t} \qquad (2)$$

where $\Lambda = 2L$ ranges over all sublayers. Here, the $\mathbf{g}_\lambda$ correspond to the learned gain parameters of the LNs, whereas the $s_{\lambda,t}$ scalar derive from the z-scaling performed in the λ^{th} LN, as defined above. The input $\mathbf{x}_{0,t}$ consists of the sum of a static lookup embedding and a positional encoding—as such, it resembles an uncontextualized embedding.

The next two terms capture the outputs of specific submodules, either FFs or MHAs. As such, their importance and usefulness will differ from task to task. The term $\mathbf{f}_t$ is the sum of the outputs of the FF submodules. Submodule outputs pass through LNs of all the layers above, hence:

$$\mathbf{f}_t = \sum_{l=1}^{L} \frac{\bigodot_{\lambda=2l}^{\Lambda} \mathbf{g}_\lambda}{\prod_{\lambda=2l}^{\Lambda} s_{\lambda,t}} \odot \tilde{\mathbf{f}}_{l,t} \qquad (3)$$

where $\tilde{\mathbf{f}}_{l,t} = \phi\left(\mathbf{x}_{l,t}^{(\mathrm{FF})} \mathbf{W}_l^{(\mathrm{FF,I})} + \mathbf{b}_l^{(\mathrm{FF,I})} \right) \mathbf{W}_l^{(\mathrm{FF,O})}$ is the unbiased output at the position t of the FF submodule for this layer l.

The term $\mathbf{h}_t$ corresponds to the sum across layers of each MHA output, having passed through the relevant LNs. As MHAs are entirely linear, we can further describe each output as a sum over all H heads of a weighted bag-of-words of the input representations to that submodule. Or:

$$\mathbf{h}_t = \sum_{l=1}^{L} \left(\frac{\bigodot_{\lambda=2l-1}^{\Lambda} \mathbf{g}_\lambda}{\prod_{\lambda=2l-1}^{\Lambda} s_{\lambda,t}} \odot \left[\sum_{h=1}^{H} \sum_{t'=1}^{n} \alpha_{l,h,t,t'} \mathbf{x}_{l,t'} \mathbf{Z}_{l,h} \right] \right)$$
$$\mathbf{Z}_{l,h} = \mathbf{W}_{l,h}^{(\mathrm{V})} \mathbf{M}_h \mathbf{W}_l^{(\mathrm{MHA,O})} \qquad (4)$$

[2] We empirically verified that components from attested embeddings $\mathbf{e}_t$ and those derived from Eq. (1) are systematically equal up to $\pm 10^{-7}$.

where $\mathbf{Z}_{l,h}$ corresponds to passing an input embedding through the unbiased values projection $\mathbf{W}_{l,h}^{(\mathrm{V})}$ of the head h, then projecting it from a d/H-dimensional subspace onto a d-dimensional space using a zero-padded identity matrix:

$$\mathbf{M}_h = \left[\begin{array}{ccc} \mathbf{0}_{d/H,(h-1)\times d/H} & \mathbf{I}_{d/H} & \mathbf{0}_{d/H,(H-h)\times d/H} \end{array} \right]$$

and finally passing it through the unbiased outer projection $\mathbf{W}_l^{(\mathrm{MHA,O})}$ of the relevant MHA.

In the last term $\mathbf{c}_t$, we collect all the biases. We don't expect these offsets to be meaningful but rather to depict a side-effect of the architecture:

$$\begin{aligned}\mathbf{c}_t = &\sum_{\lambda=1}^{\Lambda} \left(\frac{\bigodot\limits_{\lambda'=\lambda+1}^{\Lambda} \mathbf{g}_{\lambda'}}{\prod\limits_{\lambda'=\lambda+1}^{\Lambda} s_{\lambda',t}} \odot \mathbf{b}_{\lambda}^{(\mathrm{LN})} - \frac{\bigodot\limits_{\lambda'=\lambda}^{\Lambda} \mathbf{g}_{\lambda'}}{\prod\limits_{\lambda'=\lambda}^{\Lambda} s_t^{\lambda'}} \odot \left(m_{\lambda,t} \cdot \vec{1} \right) \right) \\ &+ \sum_{l=1}^{L} \frac{\bigodot\limits_{\lambda=2l-1}^{\Lambda} \mathbf{g}_{\lambda}}{\prod\limits_{\lambda=2l-1}^{\Lambda} s_{\lambda,t}} \odot \left[\mathbf{b}_l^{(\mathrm{MHA,O})} + \left(\bigoplus_{h=1}^{H} \mathbf{b}_{l,h}^{(\mathrm{V})} \right) \mathbf{W}_l^{(\mathrm{MHA,O})} \right] \\ &+ \sum_{l=1}^{L} \frac{\bigodot\limits_{\lambda=2l}^{\Lambda} \mathbf{g}_{\lambda}}{\prod\limits_{\lambda=2l}^{\Lambda} s_{\lambda,t}} \odot \mathbf{b}_l^{(\mathrm{FF,O})} \end{aligned} \quad (5)$$

The concatenation $\bigoplus_h \mathbf{b}_{l,h}^{(\mathrm{V})}$ here is equivalent to a sum of zero-padded identity matrices: $\sum_h \mathbf{b}_{l,h}^{(\mathrm{V})} \mathbf{M}_h$. This term $\mathbf{c}_t$ includes the biases $\mathbf{b}_{\lambda}^{(\mathrm{LN})}$ and mean-shifts $m_{\lambda,t} \cdot \vec{1}$ of the LNs, the outer projection biases of the FF submodules $\mathbf{b}_l^{(\mathrm{FF,O})}$, the outer projection bias in each MHA submodule $\mathbf{b}_l^{(\mathrm{MHA,O})}$ and the value projection biases, mapped through the outer MHA projection $\left(\bigoplus_h \mathbf{b}_{l,h}^{(\mathrm{V})} \right) \mathbf{W}_l^{(\mathrm{MHA,O})}$.[3]

2.3 Limitations of Equation(1)

The decomposition proposed in Equation (1) comes with a few caveats that are worth addressing explicitly. Most importantly, Equation (1) does not entail that the terms are independent from one another. For instance, the scaling factor $1/\prod s_{\lambda,t}$ systematically depends on the magnitude of earlier hidden representations. Equation (1) only stresses

[3]In the case of relative positional embeddings applied to value projections (Shaw et al., 2018), it is rather straightforward to follow the same logic so as to include relative positional offset in the most appropriate term.

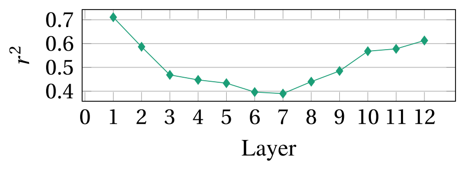


Figure 2: Fitting the $\mathbf{f}_t$ term: r^2 across layers.

that a Transformer embedding can be decomposed as a sum of the outputs of its submodules: It does not fully disentangle computations. We leave the precise definition of computation disentanglement and its elaboration for the Transformer to future research, and focus here on the decomposition proposed in Equation (1).

In all, the major issue at hand is the $\mathbf{f}_t$ term: It is the only term that cannot be derived as a linear composition of vectors, due to the non-linear function used in the FFs. Aside from the $\mathbf{f}_t$ term, non-linear computations all devolve into scalar corrections (namely, the LN z-scaling factors $s_{\lambda,t}$ and $m_{\lambda,t}$ and the attention weights $\alpha_{l,h}$). As such, $\mathbf{f}_t$ is the single bottleneck that prevents us from entirely decomposing a Transformer embedding as a linear combination of sub-terms.

As the non-linear functions used in Transformers are generally either ReLU or GELU, which both behave almost linearly for a high enough input value, it is in principle possible that the FF submodules can be approximated by a purely linear transformation, depending on the exact set of parameters they converged onto. It is worth assessing this possibility. Here, we learn a least-squares linear regression mapping the z-scaled inputs of every FF to its corresponding z-scaled output. We use the BERT base uncased model of Devlin et al. (2019) and a random sample of 10,000 sentences from the Europarl English section (Koehn, 2005), or almost 900,000 word-piece tokens, and fit the regressions using all 900,000 embeddings.

Figure 2 displays the quality of these linear approximations, as measured by a r^2 score. We see some variation across layers but never observe a perfect fit: 30% to 60% of the observed variance is not explained by a linear map, suggesting BERT actively exploits the non-linearity. That the model doesn't simply circumvent the non-linear function to adopt a linear behavior intuitively makes sense: Adding the feed-forward terms is what prevents the model from devolving into a sum of bag-of-words and static embeddings. While such approaches have been successful (Mikolov et al., 2013; Mitchell and Lapata, 2010), a non-linearity ought to make the model more expressive.

In all, the sanity check in Figure 2 highlights that the interpretation of the $\mathbf{f}_t$ term is the major ‘‘black box’’ unanalyzable component remaining under Equation (1). As such, the recent interest in analyzing these modules (e.g., Geva et al., 2021; Zhao et al., 2021; Geva et al., 2022) is likely to have direct implications for the relevance of the present work. When adopting the linear decomposition approach we advocate, this problem can be further simplified: We only require a computationally efficient algorithm to map an input weighted sum of vectors through the non-linearity to an output weighted sum of vectors.[4]

Also remark that previous research stressed that Transformer layers exhibit a certain degree of commutativity (Zhao et al., 2021) and that additional computation can be injected between contiguous sublayers (Pfeiffer et al., 2020). This can be thought of as evidence pointing towards a certain independence of the computations done in each layers: If we can shuffle and add layers, then it seems reasonable to characterize sublayers based on what their outputs add to the total embedding, as we do in Equation (1).

Beyond the expectations we may have, it remains to be seen whether our proposed methodology is of actual use, that is, whether is conducive to further research. The remainder of this article presents some analyses that our decomposition enables us to conduct.[5]

3 Visualizing the Contents of Embeddings

One major question is that of the relative relevance of the different submodules of the architecture with respect to the overall output embedding. Studying the four terms $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{h}_t$, and $\mathbf{c}_t$ can prove helpful in this endeavor. Given that Equations (2) to (5) are defined as sums across layers or sublayers, it is straightforward to adapt them to derive the decomposition for intermediate representations. Hence, we can study how relevant are each of the four terms to intermediary representations, and plot how this relevance evolves across layers.

To that end, we propose an *importance metric* to compare one of the terms $\mathbf{t}_t$ to the total $\mathbf{e}_t$. We require it to be sensitive to co-directionality (i.e., whether $\mathbf{t}_t$ and $\mathbf{e}_t$ have similar directions) and relative magnitude (whether $\mathbf{t}_t$ is a major component of $\mathbf{e}_t$). A normalized dot-product of the form:

$$\mu(\mathbf{e}_t, \mathbf{t}_t) = \mathbf{e}_t^T \mathbf{t}_t / \|\mathbf{e}_t\|_2^2 \tag{6}$$

satisfies both of these requirements. As dot-product distributes over addition (i.e., $\mathbf{a}^T \sum_i \mathbf{b}_i = \sum_i \mathbf{a}^T \mathbf{b}_i$) and the dot-product of a vector with itself is its magnitude squared (i.e., $\mathbf{a}^T \mathbf{a} = \|\mathbf{a}\|_2^2$):

$$\mu(\mathbf{e}_t, \mathbf{i}_t) + \mu(\mathbf{e}_t, \mathbf{f}_t) + \mu(\mathbf{e}_t, \mathbf{h}_t) + \mu(\mathbf{e}_t, \mathbf{c}_t) = 1$$

Hence this function intuitively measures the importance of a term relative to the total.

We use the same Europarl sample as in Section 2.3. We contrast embeddings from three related models: The BERT base uncased model and fine-tuned variants on CONLL 2003 NER (Tjong Kim Sang and De Meulder, 2003)[6] and SQuAD v2 (Rajpurkar et al., 2018).[7]

Figure 3 summarizes the relative importance of the four terms of Eq. (1), as measured by the normalized dot-product defined in Eq. (6); ticks on the x-axis correspond to different layers. Figures 3a to 3c display the evolution of our proportion metric across layers for all three BERT models, and Figures 3d to 3f display how our normalized dot-product measurements correlate across pairs of models using Spearman’s ρ.[8]

Looking at Figure 3a, we can make a few important observations. The input term $\mathbf{i}_t$, which corresponds to a static embedding, initially dominates the full output, but quickly decreases in prominence, until it reaches 0.045 at the last layer. This should explain why lower layers of Transformers generally give better performances on static word-type tasks (Vulić et al., 2020, among others). The $\mathbf{h}_t$ term is not as prominent as one could expect from the vast literature that focuses

[4]One could simply treat the effect of a non-linear activation as if it were an offset. For instance, in the case of ReLU:

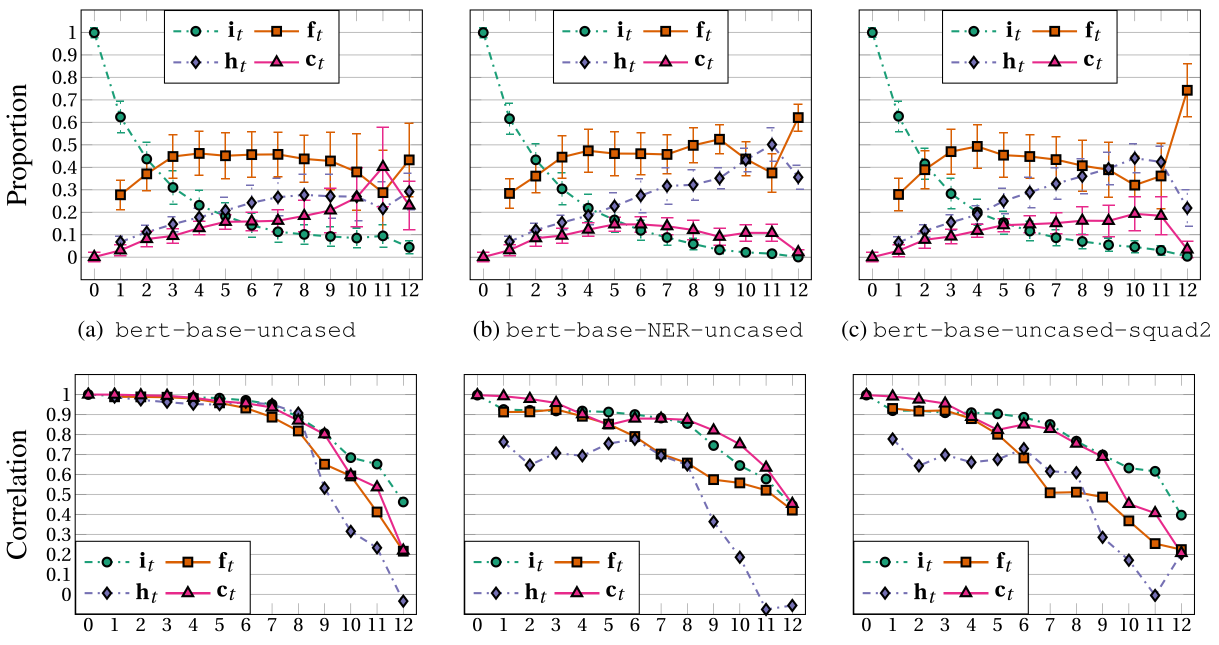
$$\text{ReLU}(\mathbf{v}) = \mathbf{v} + \mathbf{z} \quad \text{where } \mathbf{z} = \text{ReLU}(\mathbf{v}) - \mathbf{v} = \text{ReLU}(-\mathbf{v})$$

[5]Code for our experiments is available at the following URL: `https://github.com/TimotheeMickus/bert-splat`.

[6]`https://huggingface.co/dslim/bert-base-NER-uncased`.

[7]`https://huggingface.co/twmkn9/bert-base-uncased-squad2`.

[8]Layer 0 is the layer normalization conducted before the first sublayer, hence $\mathbf{f}_t$ and $\mathbf{h}_t$ are undefined here.



(a) `bert-base-uncased` (b) `bert-base-NER-uncased` (c) `bert-base-uncased-squad2`

(d) correlation BERT /NER (e) correlation BERT / QA (f) correlation NER / QA

Figure 3: Relative importance of main terms.

on MHA. Its normalized dot-product is barely above what we observe for $\mathbf{c}_t$, and never averages above 0.3 across any layer. This can be partly pinned down on the prominence of $\mathbf{f}_t$ and its normalized dot-product of 0.4 or above across most layers. As FF submodules are always the last component added to each hidden state, the sub-terms of $\mathbf{f}_t$ go through fewer LNs than those of $\mathbf{h}_t$, and thus undergo fewer scalar multiplications—which likely affects their magnitude. Lastly, the term $\mathbf{c}_t$ is far from negligible: At layer 11, it is the most prominent term, and in the output embedding it makes up for up to 23%. Note that $\mathbf{c}_t$ defines a set of offsets embedded in a 2Λ-dimensional hyperplane (cf. Appendix B). In BERT base, 23% of the output can be expressed using a 50-dimensional vector, or 6.5% of the 768 dimensions of the model. This likely induces part of the anisotropy of Transformer embeddings (e.g., Ethayarajh, 2019; Timkey and van Schijndel, 2021), as the $\mathbf{c}_t$ term pushes the embedding towards a specific region of the space.

The fine-tuned models in Figures 3b and 3c are found to impart a much lower proportion of the contextual embeddings to the $\mathbf{i}_t$ and $\mathbf{c}_t$ terms. While $\mathbf{f}_t$ seems to dominate in the final embedding, looking at the correlations in Figures 3d and 3e suggest that the $\mathbf{h}_t$ terms are those that undergo the most modifications. Proportions assigned to the terms correlate with those assigned in the non-finetuned model more in the case of lower layers than higher layers (Figures 3d and 3e). The required adaptations seem task-specific as the two fine-tuned models do not correlate highly with each other (Figure 3f). Lastly, updates in the NER model impact mostly layer 8 and upwards (Figure 3d), whereas the QA model (Figure 3e) sees important modifications to the $\mathbf{h}_t$ term at the first layer, suggesting that SQuAD requires more drastic adaptations than CONLL 2003.

4 The MLM Objective

An interesting follow-up question concerns which of the four terms allow us to retrieve the target word-piece. We consider two approaches: (a) Using the actual projection learned by the non-fine-tuned BERT model, or (b) learning a simple categorical regression for a specific term. We randomly select 15% of the word-pieces in our Europarl sample. As in the work of Devlin et al. (2019), 80% of these items are masked, 10% are replaced by a random word-piece, and 10% are left as is. Selected embeddings are then split between train (80%), validation (10%), and test (10%).

Results are displayed in Table 2. The first row (''Default'') details predictions using the default output projection on the vocabulary, that is, we test the performances of combinations sub-terms

Using	$\mathbf{i}_t$	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗	✓
	$\mathbf{h}_t$	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✓
	$\mathbf{f}_t$	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✓
	$\mathbf{c}_t$	✗	✗	✗	✓	✗	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓
Default		3.08	39.64	1.36	4.44	40.55	2.99	4.40	22.20	35.69	48.78	32.45	35.23	49.41	57.17	57.33
Learned	runs	14.51	48.33	53.92	9.24	49.72	53.66	13.18	55.45	48.10	**<u>53.97</u>**	55.58	49.08	54.05	**<u>55.79</u>**	55.53
		<u>15.54</u>	**<u>49.74</u>**	53.74	9.63	47.79	**<u>53.96</u>**	13.56	55.53	48.92	53.42	55.58	49.43	54.03	55.33	**<u>55.91</u>**
		15.09	49.60	53.69	9.86	49.71	53.61	13.72	55.54	48.67	53.57	54.76	48.46	54.07	55.26	55.75
		14.95	49.23	53.56	9.66	49.78	53.86	11.52	**<u>55.61</u>**	**<u>49.83</u>**	53.66	55.72	49.83	53.90	55.48	55.73
		15.22	49.52	**<u>53.95</u>**	**<u>10.21</u>**	**<u>49.86</u>**	53.85	**<u>14.29</u>**	55.40	49.68	53.85	**<u>55.81</u>**	**<u>49.86</u>**	**<u>54.08</u>**	55.79	55.52
	μ	15.06	49.28	53.77	9.72	49.37	53.79	13.26	55.51	49.04	53.69	55.49	49.33	54.03	55.53	55.69
	σ	±0.34	±0.51	±0.15	±0.32	±0.79	±0.13	±0.94	±0.07	±0.64	±0.20	±0.38	±0.52	±0.06	±0.22	±0.15

Table 2: Masked language model accuracy (in %). Cells in **<u>underlined bold</u>** font indicate best performance per setup across runs. Cell color indicates the ranking of setups within a run. Rows marked μ contain average performance; rows marked σ contains the standard deviation across runs.

under the circumstances encountered by the model during training.[9] The rows below (''Learned'') correspond to learned linear projections; the row marked μ display the average performance across all 5 runs. Columns display the results of using the sum of 1, 2, 3, or 4 of the terms $\mathbf{i}_t$, $\mathbf{h}_t$, $\mathbf{f}_t$ and $\mathbf{c}_t$ to derive representations; for example, the rightmost corresponds to $\mathbf{i}_t + \mathbf{h}_t + \mathbf{f}_t + \mathbf{c}_t$ (i.e., the full embedding), whereas the leftmost corresponds to predicting based on $\mathbf{i}_t$ alone. Focusing on the default projection first, we see that it benefits from a more extensive training: When using all four terms, it is almost 2% more accurate than learning one from scratch. On the other hand, learning a regression allows us to consider more specifically what can be retrieved from individual terms, as is apparent from the behavior of the $\mathbf{f}_t$: When using the default output projection, we get 1.36% accuracy, whereas a learned regression yields 53.77%.

The default projection matrix is also highly dependent on the normalization offsets $\mathbf{c}_t$ and the FF terms $\mathbf{f}_t$ being added together: Removing this $\mathbf{c}_t$ term from any experiment using $\mathbf{f}_t$ is highly detrimental to the accuracy. On the other hand, combining the two produces the highest accuracy scores. Our logistic regressions show that most of this performance can be imputed to the $\mathbf{f}_t$ term. Learning a projection from the $\mathbf{f}_t$ term already yields an accuracy of almost 54%. On the other hand, a regression learned from $\mathbf{c}_t$ only has a limited performance of 9.72% on average. Interestingly, this is still above what one would observe if the model always predicted the most frequent word-piece (viz. `the`, 6% of the test targets): even these very semantically bare items can be exploited by a classifier. As $\mathbf{c}_t$ is tied to the LN z-scaling, this suggests that the magnitude of Transformer embeddings is not wholly meaningless.

In all, do FFs make the model more effective? The $\mathbf{f}_t$ term is necessary to achieve the highest accuracy on the training objective of BERT. On its own, it doesn't achieve the highest performances: for that we also need to add the MHA outputs $\mathbf{h}_t$. However, the performances we can associate to $\mathbf{f}_t$ on its own are higher than what we observe for $\mathbf{h}_t$, suggesting that FFs make the Transformer architecture more effective on the MLM objective. This result connects with the work of Geva et al. (2021, 2022) who argue that FFs update the distribution over the full vocabulary, hence it makes sense that $\mathbf{f}_t$ would be most useful to the MLM task.

5 Lexical Contents and WSD

We now turn to look at how the vector spaces are organized, and which term yields the most linguistically appropriate space. We rely on WSD, as distinct senses should yield different representations.

We consider an intrinsic KNN-based setup and an extrinsic probe-based setup. The former is inspired from Wiedemann et al. (2019): We assign to a target the most common label in its neighborhood. We restrict neighborhoods to words with the same annotated lemma and use the 5 nearest

[9] We thank an anonymous reviewer for pointing out that the BERT model ties input and output embeddings; we leave investigating the implications of this fact for future work.

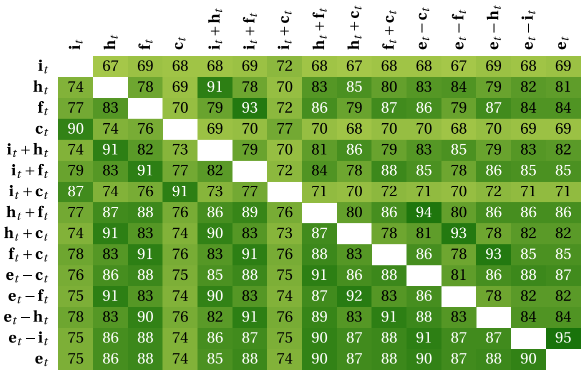
Using	$\mathbf{i}_t$	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗	✓
	$\mathbf{h}_t$	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✓
	$\mathbf{f}_t$	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✓
	$\mathbf{c}_t$	✗	✗	✗	✓	✗	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓
KNN		54.36	64.07	62.45	55.40	64.22	62.22	56.34	63.37	64.40	62.43	63.56	64.44	62.18	64.10	63.94
Classifiers	runs	**58.42**	66.84	64.46	57.32	66.65	63.88	**57.83**	65.71	**66.95**	64.46	65.76	**66.77**	64.45	65.88	65.99
		57.91	66.52	64.74	57.40	66.69	64.23	57.66	65.96	66.81	64.39	**65.94**	66.73	64.56	65.99	66.03
		58.31	66.39	64.67	**57.41**	66.56	**64.59**	57.55	**66.14**	66.51	64.35	65.89	66.74	64.17	65.80	65.83
		57.74	**67.06**	64.44	57.28	66.41	64.42	57.83	66.13	66.52	64.64	65.89	66.46	**64.58**	65.84	65.83
		58.10	66.59	**64.78**	57.20	**67.09**	64.31	57.34	65.86	66.80	**64.70**	65.73	66.74	64.57	**66.12**	**66.08**
	μ	58.09	66.68	64.62	57.32	66.68	64.29	57.64	65.96	66.72	64.51	65.84	66.69	64.47	65.92	65.95
	σ	±0.25	±0.24	±0.14	±0.08	±0.23	±0.23	±0.18	±0.16	±0.17	±0.14	±0.08	±0.12	±0.16	±0.12	±0.10

Table 3: Accuracy on SemCor WSD (in %).

neighbors using cosine distance. The latter is a 2-layer MLP similar to Du et al. (2019), where the first layer is shared for all items and the second layer is lemma-specific. We use the NLTK Semcor dataset (Landes et al., 1998; Bird et al., 2009), with an 80%–10%–10% split. We drop monosemous or OOV lemmas and sum over word-pieces to convert them into single word representations. Table 3 shows accuracy results. Selecting the most frequent sense would yield an accuracy of 57%; picking a sense at random, 24%. The terms $\mathbf{i}_t$ and $\mathbf{c}_t$ struggle to outperform the former baseline: relevant KNN accuracy scores are lower, and corresponding probe accuracy scores are barely above.

Overall, the same picture emerges from the KNN setup and all 5 runs of the classifier setup. The $\mathbf{f}_t$ term does not yield the highest performances in our experiment—instead, the $\mathbf{h}_t$ term systematically dominates. In single term models, $\mathbf{h}_t$ is ranked first and $\mathbf{f}_t$ second. As for sums of two terms, the setups ranked 1st, 2nd, and 3rd are those that include $\mathbf{h}_t$; setups ranked 3rd to 5th, those that include $\mathbf{f}_t$. Even more surprisingly, when summing three of the terms, the highest ranked setup is the one where we exclude $\mathbf{f}_t$, and the lowest corresponds to excluding $\mathbf{h}_t$. Removing $\mathbf{f}_t$ systematically yields better performances than using the full embedding. This suggests that $\mathbf{f}_t$ is not necessarily helpful to the final representation for WSD. This contrast with what we observed for MLM, where $\mathbf{h}_t$ was found to be less useful then $\mathbf{f}_t$.

One argument that could be made here would be to posit that the predictions derived from the different sums of terms are intrinsically different, hence a purely quantitative ranking might not capture this important distinction. To verify whether this holds, we can look at the proportion of predictions that agree for any two models. Because



	$\mathbf{i}_t$	$\mathbf{h}_t$	$\mathbf{f}_t$	$\mathbf{c}_t$	$\mathbf{i}_t+\mathbf{h}_t$	$\mathbf{i}_t+\mathbf{f}_t$	$\mathbf{i}_t+\mathbf{c}_t$	$\mathbf{h}_t+\mathbf{f}_t$	$\mathbf{h}_t+\mathbf{c}_t$	$\mathbf{f}_t+\mathbf{c}_t$	$\mathbf{e}_t-\mathbf{c}_t$	$\mathbf{e}_t-\mathbf{f}_t$	$\mathbf{e}_t-\mathbf{h}_t$	$\mathbf{e}_t-\mathbf{i}_t$	$\mathbf{e}_t$
$\mathbf{i}_t$		67	69	68	68	69	72	68	67	68	68	67	69	68	69
$\mathbf{h}_t$	74		78	69	91	78	70	83	85	80	83	84	79	82	81
$\mathbf{f}_t$	77	83		70	79	93	72	86	79	87	86	79	87	84	84
$\mathbf{c}_t$	90	74	76		69	70	77	70	68	70	70	68	70	69	69
$\mathbf{i}_t+\mathbf{h}_t$	74	91	82	73		79	70	81	86	79	83	85	79	83	82
$\mathbf{i}_t+\mathbf{f}_t$	79	83	91	77	82		72	84	78	88	85	78	86	85	85
$\mathbf{i}_t+\mathbf{c}_t$	87	74	76	91	73	77		71	70	72	71	70	72	71	71
$\mathbf{h}_t+\mathbf{f}_t$	77	87	88	76	86	89	76		80	86	94	80	86	86	86
$\mathbf{h}_t+\mathbf{c}_t$	74	91	83	74	90	83	73	87		78	81	93	78	82	82
$\mathbf{f}_t+\mathbf{c}_t$	78	83	91	76	83	91	76	88	83		86	78	93	85	85
$\mathbf{e}_t-\mathbf{c}_t$	76	86	88	75	85	88	75	91	86	88		81	86	88	87
$\mathbf{e}_t-\mathbf{f}_t$	75	91	83	74	90	83	74	87	92	83	86		78	82	82
$\mathbf{e}_t-\mathbf{h}_t$	78	83	90	76	82	91	76	89	83	91	88	83		84	84
$\mathbf{e}_t-\mathbf{i}_t$	75	86	88	74	86	87	75	90	87	88	91	87	87		95
$\mathbf{e}_t$	75	86	88	74	85	88	74	90	87	88	90	87	88	90	

Figure 4: Prediction agreement for WSD models (in %). Upper triangle: agreement for KNNs; lower triangle: for learned classifiers.

our intent is to see what can be retrieved from specific subterms of the embedding, we focus solely on the most efficient classifiers across runs. This is summarized in Figure 4: An individual cell will detail the proportion of the assigned labels shared by the models for that row and that column. In short, we see that model predictions tend to a high degree of overlap. For both KNN and classifier setups, the three models that appear to make the most distinct predictions turn out to be computed from the $\mathbf{i}_t$ term, the $\mathbf{c}_t$ term or their sum: that is, the models that struggle to perform better than the MFS baseline and are derived from static representations.

6 Effects of Fine-tuning and NER

Downstream application can also be achieved through fine-tuning, that is, restarting a model's training to derive better predictions on a narrower task. As we saw from Figures 3b and 3c, the modifications brought upon this second round

Using	$\mathbf{i}_t$	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗	✓
	$\mathbf{h}_t$	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✓
	$\mathbf{f}_t$	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✓
	$\mathbf{c}_t$	✗	✗	✗	✓	✗	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓
Base	runs	14.93	32.89	31.12	4.92	20.95	30.62	12.81	31.77	**34.62**	**32.43**	34.03	32.52	31.45	33.85	32.95
		15.79	**34.16**	31.05	4.88	**34.38**	31.08	10.26	34.31	33.12	30.14	31.84	32.57	30.48	**33.99**	30.82
		15.46	29.94	28.92	4.92	33.73	30.49	**13.72**	31.48	32.18	17.35	**34.68**	**34.46**	**32.72**	32.82	32.42
		15.55	33.95	31.36	**5.32**	33.16	**32.03**	12.88	**34.32**	33.37	31.09	31.80	32.91	31.42	31.72	**34.01**
		4.52	33.62	**31.38**	5.28	28.22	28.00	12.20	31.99	33.70	30.45	31.19	33.80	32.71	13.63	31.33
	μ	13.25	32.91	30.76	5.07	30.09	30.44	12.37	32.77	33.40	28.29	32.71	33.25	31.76	29.20	32.30
	σ	±4.37	±1.55	±0.93	±0.20	±5.06	±1.34	±1.16	±1.27	±0.79	±5.53	±1.38	±0.76	±0.86	±7.83	±1.14
Tuned	runs	**16.28**	39.24	36.96	12.23	**39.92**	38.65	15.70	37.39	38.67	38.25	38.67	**40.17**	39.01	37.30	37.46
		5.07	39.62	38.80	8.69	38.58	**39.50**	6.36	37.67	38.86	37.85	38.77	38.01	39.31	37.65	38.60
		16.06	39.06	38.62	11.81	38.93	37.53	4.69	**39.92**	**39.78**	37.92	38.03	39.33	35.83	37.18	**39.27**
		15.08	38.24	**39.81**	4.83	39.76	38.74	**16.07**	37.92	37.23	**39.94**	37.49	39.00	38.43	36.37	38.98
		11.68	**40.25**	39.23	**12.36**	38.87	37.43	15.74	35.48	39.32	38.83	**38.89**	38.80	**39.47**	**38.26**	38.30
	μ	12.83	39.28	38.68	9.98	39.21	38.37	11.71	37.68	38.77	38.56	38.37	39.06	38.41	37.35	38.52
	σ	±4.22	±0.66	±0.95	±2.91	±0.53	±0.78	±5.08	±1.41	±0.86	±0.77	±0.53	±0.70	±1.34	±0.62	±0.62

Table 4: Macro-f_1 on WNUT 2016 (in %).

of training are task-specific, meaning that an exhaustive experimental survey is out of our reach.

We consider the task of Named Entity Recognition, using the WNUT 2016 shared task dataset (Strauss et al., 2016). We contrast the performance of the non-finetuned BERT model to that of the aforementioned variant fine-tuned on the CONLL 2003 NER dataset using shallow probes.

Results are presented in Table 4. The very high variance we observe across is likely due to the smaller size of this dataset (46,469 training examples, as compared to the 142,642 of Section 5 or the 107,815 in Section 4). Fine-tuning BERT on another NER dataset unsurprisingly has a systematic positive impact: Average performance jumps up by 5% or more. More interesting is the impact this fine-tuning has on the $\mathbf{f}_t$ term: When used as sole input, the highest observed performance increases by over 8%, and similar improvements are observed consistently across all setups involving $\mathbf{f}_t$. Yet, the best average performance for fine-tuned and base embeddings correspond to $\mathbf{h}_t$ (39.28% in tuned), $\mathbf{i}_t+\mathbf{h}_t$ (39.21%), and $\mathbf{i}_t+\mathbf{h}_t+\mathbf{c}_t$ (39.06%); in the base setting the highest average performance are reached with $\mathbf{h}_t + \mathbf{c}_t$ (33.40%), $\mathbf{i}_t + \mathbf{h}_t + \mathbf{c}_t$ (33.25%) and $\mathbf{h}_t$ (32.91%)—suggesting that $\mathbf{f}_t$ might be superfluous for this task.

We can also look at whether the highest scoring classifiers across runs classifiers produce different outputs. Given the high class imbalance of the dataset at hand, we macro-average the prediction overlaps by label. The result is shown in Figure 5; the upper triangle details the behavior of the untuned model, and the lower triangle details

	$\mathbf{i}_t$	$\mathbf{h}_t$	$\mathbf{f}_t$	$\mathbf{c}_t$	$\mathbf{i}_t+\mathbf{h}_t$	$\mathbf{i}_t+\mathbf{f}_t$	$\mathbf{i}_t+\mathbf{c}_t$	$\mathbf{h}_t+\mathbf{f}_t$	$\mathbf{h}_t+\mathbf{c}_t$	$\mathbf{f}_t+\mathbf{c}_t$	$\mathbf{e}_t-\mathbf{c}_t$	$\mathbf{e}_t-\mathbf{f}_t$	$\mathbf{e}_t-\mathbf{h}_t$	$\mathbf{e}_t-\mathbf{i}_t$	$\mathbf{e}_t$
$\mathbf{i}_t$		17	21	5	18	23	22	21	16	21	20	17	22	21	21
$\mathbf{h}_t$	18		45	5	70	48	15	55	69	46	55	69	49	53	57
$\mathbf{f}_t$	19	72		5	46	66	17	54	41	68	56	44	65	55	59
$\mathbf{c}_t$	11	15	14		5	5	7	5	5	5	5	5	5	6	5
$\mathbf{i}_t+\mathbf{h}_t$	19	83	71	15		46	16	53	66	49	55	71	49	53	58
$\mathbf{i}_t+\mathbf{f}_t$	19	72	81	14	71		18	57	42	65	60	46	69	55	59
$\mathbf{i}_t+\mathbf{c}_t$	17	22	22	29	21	21		17	15	17	17	15	17	17	16
$\mathbf{h}_t+\mathbf{f}_t$	18	72	75	15	72	76	21		52	56	67	54	58	64	67
$\mathbf{h}_t+\mathbf{c}_t$	19	83	71	15	82	70	22	69		44	53	65	44	51	52
$\mathbf{f}_t+\mathbf{c}_t$	18	72	79	15	71	80	21	76	70		56	45	68	55	60
$\mathbf{e}_t-\mathbf{c}_t$	18	75	78	15	75	81	22	77	73	80		56	59	68	72
$\mathbf{e}_t-\mathbf{f}_t$	19	82	69	14	83	70	21	70	80	70	71		47	54	57
$\mathbf{e}_t-\mathbf{h}_t$	19	69	77	14	70	79	21	73	68	77	79	66		55	59
$\mathbf{e}_t-\mathbf{i}_t$	18	71	74	15	71	75	21	79	67	73	78	70	73		68
$\mathbf{e}_t$	18	73	77	15	73	79	22	76	74	78	82	71	76	76	

Figure 5: NER prediction agreement (macro-average, in %). Upper triangle: agreement for untuned models; lower triangle: for tuned models.

that of the NER-fine-tuned model. In this round of experiments, we see much more distinctly that the $\mathbf{i}_t$ model, the $\mathbf{c}_t$ model, and the $\mathbf{i}_t + \mathbf{c}_t$ model behave markedly different from the rest, with $\mathbf{c}_t$ yielding the most distinct predictions. As for the NER-fine-tuned model (lower triangle), aside from the aforementioned static representations, most predictions display a degree of overlap much higher than what we observe for the non-finetuned model: Both FFs and MHAs are skewed towards producing outputs more adapted to NER tasks.

7 Relevant Work

The derivation we provide in Section 2 ties in well with other studies setting out to explain how Transformers embedding spaces are structured (Voita et al., 2019; Mickus et al., 2020; Vázquez et al., 2021, among others) and more broadly how they behave (Rogers et al., 2020). For instance, lower layers tend to yield higher performance on surface tasks (e.g., predicting the presence of a word, Jawahar et al. 2019) or static benchmarks (e.g., analogy, Vulić et al. 2020): This ties in with the vanishing prominence of $\mathbf{i}_t$ across layers. Likewise, probe-based approaches to unearth a linear structure matching with the syntactic structure of the input sentence (Raganato and Tiedemann, 2018; Hewitt and Manning, 2019, among others) can be construed as relying on the explicit linear dependence that we highlight here.

Another connection is with studies on embedding space anisotropy (Ethayarajh, 2019; Timkey and van Schijndel, 2021): Our derivation provides a means of circumscribing which neural components are likely to cause it. Also relevant is the study on sparsifying Transformer representations of Yun et al. (2021): The linearly dependent nature of Transformer embeddings has some implications when it comes to dictionary coding.

Also relevant are the works focusing on the interpretation of specific Transformer components, and feed-forward sublayers in particular (Geva et al., 2021; Zhao et al., 2021; Geva et al., 2022). Lastly, our approach provides some quantitative argument for the validity of attention-based studies (Serrano and Smith, 2019; Jain and Wallace, 2019; Wiegreffe and Pinter, 2019; Pruthi et al., 2020) and expands on earlier works looking beyond attention weights (Kobayashi et al., 2020).

8 Conclusions and Future Work

In this paper, we stress how Transformer embeddings can be decomposed linearly to describe the impact of each network component. We showcased how this additive structure can be used to investigate Transformers. Our approach suggests a less central place for attention-based studies: If multi-head attention only accounts for 30% of embeddings, can we possibly explain what Transformers do by looking solely at these submodules? The crux of our methodology lies in that we decompose the output embedding by submodule instead of layer or head. These approaches are not mutually exclusive (cf. Section 3), hence our approach can easily be combined with other probing protocols, providing the means to narrow in on specific network components.

The experiments we have conducted in Sections 3 to 6 were designed so as to showcase whether our decomposition in Equation (1) could yield useful results—or, as we put it earlier in Section 2.3, whether this approach could be conducive to future research. We were able to use the proposed approach to draw insightful connections. The noticeable anisotropy of contextual embeddings can be connected to the prominent trace of the biases in the output embedding: As model biases make up an important part of the whole embedding, they push it towards a specific sub-region of the embedding. The diminishing importance of $\mathbf{i}_t$ links back to earlier results on word-type semantic benchmarks. We also report novel findings, showcasing how some submodules outputs may be detrimental in specific scenarios: The output trace of FF modules was found to be extremely useful for MLM, whereas the $\mathbf{h}_t$ term was found to be crucial for WSD. Our methodology also allows for an overview of the impact of finetuning (cf. Section 6): It skews components towards more task-specific outputs, and its effect are especially noticeable in upper layers (Figures 3d and 3e).

Analyses in Sections 3 to 6 demonstrate the immediate insight that our Transformer decomposition can help achieve. This work therefore opens a number of research perspectives, of which we name three. First, as mentioned in Section 2.3, our approach can be extended further to more thoroughly disentangle computations. Second, while we focused here more on feed-forward and multi-head attention components, extracting the static component embeddings from $\mathbf{i}_t$ would allow for a principled comparison of contextual and static distributional semantics models. Last but not least, because our analysis highlights the different relative importance of Transformer components in different tasks, it can be used to help choose the most appropriate tools for further interpretation of trained models among the wealth of alternatives.

Acknowledgments

We are highly indebted to Marianne Clausel for her significant help with how best to present the mathematical aspects of this work. Our thanks also go to Aman Sinha, as well as three anonymous reviewers for their substantial comments towards bettering this work.

This work was supported by a public grant overseen by the French National Research

Agency (ANR) as part of the ''Investissements d'Avenir'' program: Idex *Lorraine Université d'Excellence* (reference: ANR-15-IDEX-0004). We also acknowledge the support by the FoTran project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 771113).

A Step-by-step Derivation of Eq. (1)

Given that a Transformer layer consists of a stack of L layers, each comprising two sublayers, we can treat a Transformer as a stack of $\Lambda = 2L$ sublayers. For notation simplicity, we link the sublayer index λ to the layer index l: The first sublayer of layer l is the $(2l-1)^{\text{th}}$ sublayer, and the second is the $(2l)^{\text{th}}$ sublayer.[10] All sublayers include a residual connection before the final LN:

$$\mathbf{y}_{\lambda,t} = \mathbf{g}_\lambda \odot \left[\frac{(S(\mathbf{x}) + \mathbf{x}) - m_{\lambda,t} \cdot \vec{1}}{s_{\lambda,t}}\right] + \mathbf{b}_\lambda^{(\text{LN})}$$

We can model the effects of the gain $\mathbf{g}_\lambda$ and the scaling $1/s_{\lambda,t}$ as the $d \times d$ square matrix:

$$\mathbf{T}_\lambda = \frac{1}{s_{\lambda,t}} \begin{bmatrix} (\mathbf{g}_\lambda)_1 & 0 & \cdots & 0 \\ 0 & (\mathbf{g}_\lambda)_2 & & \\ \vdots & & \ddots & \\ 0 & & & (\mathbf{g}_\lambda)_d \end{bmatrix}$$

which we use to rewrite a sublayer output $\mathbf{y}_{\lambda,t}$ as:

$$\begin{aligned}\mathbf{y}_{\lambda,t} &= \left(S_\lambda(\mathbf{x}) + \mathbf{x} - \left(m_{\lambda,t} \cdot \vec{1}\right)\right)\mathbf{T}_\lambda + \mathbf{b}_\lambda^{(\text{LN})} \\ &= S_\lambda(\mathbf{x})\mathbf{T}_\lambda + \mathbf{x}\mathbf{T}_\lambda - \left(m_{\lambda,t} \cdot \vec{1}\right)\mathbf{T}_\lambda + \mathbf{b}_\lambda^{(\text{LN})}\end{aligned}$$

We can then consider what happens to this additive structure in the next sublayer. We first define $\mathbf{T}_{\lambda+1}$ as previously and remark that, as both $\mathbf{T}_\lambda$ and $\mathbf{T}_{\lambda+1}$ only contain diagonal entries:

$$\mathbf{T}_\lambda \mathbf{T}_{\lambda+1} = \frac{1}{s_{\lambda,t}s_{\lambda+1,t}} \times \begin{bmatrix} (\mathbf{g}_\lambda \odot \mathbf{g}_{\lambda+1})_1 & \cdots & 0 \\ \vdots & \ddots & \\ 0 & & (\mathbf{g}_\lambda \odot \mathbf{g}_{\lambda+1})_d \end{bmatrix}$$

[10] In the case of BERT, we also need to include a LN before the first layer, which is straightforward if we index it as $\lambda = 0$.

This generalizes for any sequence of LNs as:

$$\prod_\lambda \mathbf{T}_\lambda = \frac{1}{\prod_\lambda s_{\lambda,t}} \times \begin{bmatrix} \left(\bigodot_\lambda \mathbf{g}_\lambda\right)_1 & \cdots & 0 \\ \vdots & \ddots & \\ 0 & & \left(\bigodot_\lambda \mathbf{g}_\lambda\right)_d \end{bmatrix}$$

Let us now pass the input $\mathbf{x}$ through a complete layer, that is, through sublayers λ and $\lambda+1$:

$$\begin{aligned}\mathbf{y}_{\lambda+1,t} &= S_{\lambda+1}(\mathbf{y}_{\lambda,t})\mathbf{T}_{\lambda+1} \\ &\quad + \mathbf{y}_\lambda \mathbf{T}_{\lambda+1} - \left(m_{\lambda+1,t} \cdot \vec{1}\right)\mathbf{T}_{\lambda+1} + \mathbf{b}_{\lambda+1}^{(\text{LN})}\end{aligned}$$

Substituting in the expression for $\mathbf{y}_\lambda$ from above:

$$\begin{aligned}\mathbf{y}_{\lambda+1,t} &= S_{\lambda+1}\Big(S_\lambda(\mathbf{x})\mathbf{T}_\lambda + \mathbf{x}\mathbf{T}_\lambda \\ &\quad - \left(m_{\lambda,t} \cdot \vec{1}\right)\mathbf{T}_\lambda + \mathbf{b}_\lambda^{(\text{LN})}\Big)\mathbf{T}_{\lambda+1} \\ &\quad + S_\lambda(\mathbf{x})\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) + \mathbf{x}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ &\quad - \left(m_{\lambda,t} \cdot \vec{1}\right)\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ &\quad + \mathbf{b}_\lambda^{(\text{LN})}\mathbf{T}_{\lambda+1} - \left(m_{\lambda+1,t} \cdot \vec{1}\right)\mathbf{T}_{\lambda+1} \\ &\quad + \mathbf{b}_{\lambda+1}^{(\text{LN})}\end{aligned}$$

As we are interested in the combined effects of a layer, we only consider the case where S_λ is a MHA mechanism and $S^{\lambda+1}$ a FF. We start by reformulating the output of a MHA. Recall that attention heads can be seen as weighted sums of value vectors (Kobayashi et al., 2020). Due to the softmax normalization, attention weights $\alpha_{t,1}, \ldots \alpha_{t,n}$ sum to 1 for any position t. Hence:

$$\begin{aligned}(\mathbf{A}_h)_{t,\cdot} &= \sum_{t'=1}^{n} \alpha_{h,t,t'}(\mathbf{V}_h)_{t',\cdot} \\ &= \sum_{t'=1}^{n}\left[\alpha_{h,t,t'}\mathbf{x}_{t'}\mathbf{W}_h^{(\text{V})} + \alpha_{h,t,t'}\mathbf{b}_h^{(\text{V})}\right] \\ &= \left(\sum_{t'}^{n}\alpha_{h,t,t'}\mathbf{x}_{t'}\mathbf{W}_h^{(\text{V})}\right) + \mathbf{b}_h^{(\text{V})}\end{aligned}$$

To account for all H heads in a MHA, we concatenate these head-specific sums and pass them through the output projection $\mathbf{W}^{(\text{MHA,O})}$. As such, we can denote the unbiased output of the MHA and the associated bias as:

$$\tilde{\mathbf{h}}_{l,t} = \sum_h \sum_{t'} \alpha_{l,h,t,t'} \mathbf{x}_{l,t'} \mathbf{Z}_{l,h}$$

$$\mathbf{b}_l^{(\text{MHA})} = \mathbf{b}_l^{(\text{MHA,O})} + \left(\bigoplus_h^H \mathbf{b}_{l,h}^{(\text{V})} \right) \mathbf{W}_l^{(\text{MHA,O})}$$

with $\mathbf{Z}_{l,h}$ as introduced in (4). By substituting the actual sublayer functions in our previous equation:

$$\begin{aligned}\mathbf{y}_{l,t} = {} & \tilde{\mathbf{f}}_{l,t}\mathbf{T}_{\lambda+1} + \mathbf{b}_l^{(\text{FF,O})}\mathbf{T}_{\lambda+1} + \tilde{\mathbf{h}}_{l,t}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ & + \mathbf{b}_l^{(\text{MHA})}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) + \mathbf{x}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ & - \left(m_{\lambda,t}\cdot\vec{1}\right)\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ & + \mathbf{b}_\lambda^{(\text{LN})}\mathbf{T}_{\lambda+1} - \left(m_{\lambda+1,t}\cdot\vec{1}\right)\mathbf{T}_{\lambda+1} + \mathbf{b}_{\lambda+1}^{(\text{LN})}\end{aligned}$$

Here, given that there is only one FF for this layer, the output of sublayer function at $\lambda + 1$ will correspond to the output of the FF for layer l, i.e., $\tilde{\mathbf{f}}_{l,t} + \mathbf{b}_l^{(\text{FF,O})}$, and similarly the output for sublayer λ should be that of the MHA of layer l, or $\tilde{\mathbf{h}}_{l,t} + \mathbf{b}_l^{(\text{MHA})}$. To match Eq. (1), rewrite as:

$$\begin{aligned}\mathbf{y}_{l,t} = {} & \mathbf{i}_{\lambda+1,t} + \mathbf{h}_{\lambda+1,t} + \mathbf{f}_{\lambda+1,t} + \mathbf{c}_{\lambda+1,t} \\ \mathbf{i}_{\lambda+1,t} = {} & \mathbf{x}_{\lambda,t}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ \mathbf{h}_{\lambda+1,t} = {} & \tilde{\mathbf{h}}_{l,t}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ \mathbf{f}_{\lambda+1,t} = {} & \tilde{\mathbf{f}}_{l,t}\mathbf{T}_{\lambda+1} \\ \mathbf{c}_{\lambda+1,t} = {} & \mathbf{b}_l^{(\text{FF,O})}\mathbf{T}_{\lambda+1} + \mathbf{b}_l^{(\text{MHA})}\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ & - \left(m_{\lambda,t}\cdot\vec{1}\right)\left(\prod_{\lambda'=\lambda}^{\lambda+1}\mathbf{T}_{\lambda'}\right) \\ & - \left(m_{\lambda+1,t}\cdot\vec{1}\right)\mathbf{T}_{\lambda+1} \\ & + \mathbf{b}_\lambda^{(\text{LN})}\mathbf{T}_{\lambda+1} + \mathbf{b}_{\lambda+1}^{(\text{LN})}\end{aligned}$$

where $\mathbf{x}_{\lambda,t}$ is the t^{th} input for sublayer λ; that is, the above characterizes the output of sublayer $\lambda + 1$ with respect to the input of sublayer λ. Passing the output $\mathbf{y}_{l,t}$ into the next layer $l + 1$ (i.e., through sublayers $\lambda+2$ and $\lambda+3$) then gives:

$$\begin{aligned}\mathbf{y}_{l+1,t} = {} & \mathbf{i}_{\lambda+3,t} + \mathbf{h}_{\lambda+3,t} + \mathbf{f}_{\lambda+3,t} + \mathbf{c}_{\lambda+3,t} \\ \mathbf{i}_{\lambda+3,t} = {} & \mathbf{i}_{\lambda+1,t}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) \\ \mathbf{h}_{\lambda+3,t} = {} & \mathbf{h}_{\lambda+1,t}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) \\ & + \tilde{\mathbf{h}}_{l+1,t}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) \\ \mathbf{f}_{\lambda+3,t} = {} & \mathbf{f}_{\lambda+1,t}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) + \tilde{\mathbf{f}}_{l+1,t}\mathbf{T}_{\lambda+3} \\ \mathbf{c}_{\lambda+3,t} = {} & \mathbf{c}_{\lambda+1,t}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) \\ & + \mathbf{b}_l^{(\text{MHA})}\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) + \mathbf{b}_l^{(\text{FF,O})}\mathbf{T}_{\lambda+3} \\ & - \left(m_{\lambda+2,t}\cdot\vec{1}\right)\left(\prod_{\lambda'=\lambda+2}^{\lambda+3}\mathbf{T}_{\lambda'}\right) \\ & - \left(m_{\lambda+3,t}\cdot\vec{1}\right)\mathbf{T}_{\lambda+3} \\ & + \mathbf{b}_{\lambda+2}^{(\text{LN})}\mathbf{T}_{\lambda+3} + \mathbf{b}_{\lambda+3}^{(\text{LN})}\end{aligned}$$

This logic carries on across layers: Adding a layer corresponds to (i) mapping the existing terms through the two new LNs, (ii) adding new terms for the MHA and the FF, (iii) tallying up biases introduced in the current layer. Hence, the above generalizes to any number of layers $k \geq 1$ as:[11]

$$\begin{aligned}\mathbf{y}_{l+k,t} = {} & \mathbf{i}_{\lambda+2k-1,t} + \mathbf{h}_{\lambda+2k-1,t} \\ & + \mathbf{f}_{\lambda+2k-1,t} + \mathbf{c}_{\lambda+2k-1,t} \\ \mathbf{i}_{\lambda+2k-1,t} = {} & \mathbf{x}_{\lambda,t}\left(\prod_{\lambda'=\lambda}^{\lambda+2k-1}\mathbf{T}_{\lambda'}\right) \\ \mathbf{h}_{\lambda+2k-1,t} = {} & \sum_{l'=l}^{l+k}\tilde{\mathbf{h}}_{l',t}\left(\prod_{\lambda'=2l'-1}^{2(l+k)}\mathbf{T}_{\lambda'}\right) \\ \mathbf{f}_{\lambda+2k-1,t} = {} & \sum_{l'=l}^{l+k}\tilde{\mathbf{f}}_{l',t}\left(\prod_{\lambda'=2l'}^{2(l+k)}\mathbf{T}_{\lambda'}\right)\end{aligned}$$

[11] The edge case $\prod_{\lambda'=\lambda+1}^{\lambda}\mathbf{T}_{\lambda'}$ is taken to be the identity matrix $\mathbf{I}_d$, for notation simplicity.

$$\mathbf{c}_{\lambda+2k-1,t} = \sum_{\lambda'=\lambda}^{\lambda+2k-1} \left[\mathbf{b}^{(\text{LN})}_{\lambda'} \left(\prod_{\lambda''=\lambda'+1}^{\lambda+2k-1} \mathbf{T}_{\lambda''} \right) - \left(m_{\lambda',t} \cdot \vec{1} \right) \left(\prod_{\lambda''=\lambda'}^{\lambda+2k-1} \mathbf{T}_{\lambda''} \right) \right] + \sum_{l'=l}^{l+k} \left[\mathbf{b}^{(\text{MHA})}_{l'} \left(\prod_{\lambda'=2l-1}^{\lambda+2k-1} \mathbf{T}_{\lambda'} \right) + \mathbf{b}^{(\text{FF,O})}_{l'} \left(\prod_{\lambda'=2l-1}^{\lambda+2k-1} \mathbf{T}_{\lambda'} \right) \right]$$

Lastly, recall that by construction, we have:

$$\mathbf{v} \left(\prod_{\lambda} \mathbf{T}_{\lambda} \right) = \frac{\bigodot_{\lambda} \mathbf{g}_{\lambda}}{\prod_{\lambda} s_{\lambda,t}} \odot \mathbf{v}$$

By recurrence over all layers and providing the initial input $\mathbf{x}_{0,t}$, we obtain Eqs. (1) to (5).

B Hyperplane Bounds of $\mathbf{c}_t$

We can re-write Eq. (5) to highlight that is composed only of scalar multiplications applied to constant vectors. Let:

$$\mathbf{b}^{(\text{S})}_{\lambda} = \begin{cases} \mathbf{b}^{(\text{MHA,O})}_{l} + \left(\bigoplus_{h} \mathbf{b}^{(\text{V})}_{l,h} \right) \mathbf{W}^{(\text{MHA,O})}_{l} & \textit{if } \lambda = 2l - 1 \\ \mathbf{b}^{(\text{FF,O})}_{l} & \textit{if } \lambda = 2l \end{cases}$$

$$\mathbf{p}_{\lambda} = \left(\bigodot_{\lambda'=\lambda+1}^{\Lambda} \mathbf{g}_{\lambda'} \right) \odot \left(\mathbf{b}^{(\text{LN})}_{\lambda} + \mathbf{b}^{(\text{S})}_{\lambda} \right)$$

$$\mathbf{q}_{\lambda} = \bigodot_{\lambda'=\lambda+1}^{\Lambda} \mathbf{g}_{\lambda'}$$

Using the above, Eq. (5) is equivalent to:

$$\mathbf{c}_t = \sum_{\lambda}^{\Lambda} \left(\frac{1}{\prod_{\lambda'=\lambda+1}^{\Lambda} s^{\lambda'}_{t}} \cdot \mathbf{p}_{\lambda} \right) + \sum_{\lambda}^{\Lambda} \left(\frac{-m_{\lambda,t}}{\prod_{\lambda'=\lambda+1}^{\Lambda} s_{\lambda',t}} \cdot \mathbf{q}_{\lambda} \right)$$

Note that $\mathbf{p}_{\lambda}$ and $\mathbf{q}_{\lambda}$ are constant across all inputs. Assuming their linear independence puts an upper bound of 2Λ vectors necessary to express $\mathbf{c}_t$.

C Computational Details

In Section 2.3, we use the default hyperparameters of `scikit-learn` (Pedregosa et al., 2011). In Section 4, we learn categorical regressions using an AdamW optimizer (Loshchilov and Hutter, 2019) and iterate 20 times over the train set; hyperparameters (learning rate, weight decay, dropout, and the β_1 and β_2 AdamW hyperparameters) are set using Bayes Optimization (Snoek et al., 2012), with 50 hyperparameter samples and accuracy as objective. In Section 5, learning rate, dropout, weight decay, β_1 and β_2, learning rate scheduling are selected with Bayes Optimization, using 100 samples and accuracy as objective. In Section 6, we learn shallow logistic regressions, setting hyperparameters with Bayes Optimization, using 100 samples and macro-f_1 as the objective. Experiments were run on a 4GB NVIDIA GPU.

D Ethical Considerations

The offset method of Mikolov et al. (2013) is known to also model social stereotypes (Bolukbasi et al., 2016, among others). Some of the sub-representations of our decomposition may exhibit stronger biases than the whole embedding $\mathbf{e}_t$, and can yield higher performances than focusing on the whole embedding (e.g., Table 3). This could provide an undesirable incentive to deploy NLP models with higher performances and stronger systemic biases.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc.

Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jiaju Du, Fanchao Qi, and Maosong Sun. 2019. Using BERT for word sense disambiguation. *CoRR*, abs/1909.08358.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, Hong Kong, China. Association for Computational Linguistics. `https://doi.org/10.18653/v1/D19-1006`

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. `https://doi.org/10.18653/v1/2021.emnlp-main.446`

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELU). *arXiv preprint arXiv:1606.08415*.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. 2018. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics. `https://doi.org/10.18653/v1/P19-1356`

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2020. Attention is not only a weight: Analyzing transformers with vector norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7057–7075, Online. Association for Computational Linguistics. `https://doi.org/10.18653/v1/2020.emnlp-main.574`

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86. Phuket, Thailand.

Shari Landes, Claudia Leacock, and Randee I. Tengi. 1998. Building semantic concordances. *WordNet: An Electronic Lexical Database*, chapter 8, pages 199–216. Bradford Books.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1166

Timothee Mickus, Denis Paperno, Mathieu Constant, and Kees van Deemter. 2020. What do you mean, BERT? In *Proceedings of the Society for Computation in Linguistics 2020*, pages 279–290, New York, New York. Association for Computational Linguistics.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429. https://doi.org/10.1111/j.1551-6709.2010.01106.x

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pages 46–54, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.7

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online. Association for Computational Linguistics. -.1pthttps://doi.org/10.18653/v1/2020.acl-main.432

Alessandro Raganato and Jörg Tiedemann. 2018. An analysis of encoder representations in transformer-based machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 287–297, Brussels, Belgium. Association for Computational Linguistics. https://doi.org/10.18653/v1/W18-5431

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-2124

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866. https://doi.org/10.1162/tacl_a_00349

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2931–2951, Florence, Italy. Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1282

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *CoRR*, abs/1803.02155.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Benjamin Strauss, Bethany Toma, Alan Ritter, Marie-Catherine de Marneffe, and Wei Xu. 2016. Results of the WNUT16 named entity recognition shared task. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 138–144, Osaka, Japan. The COLING 2016 Organizing Committee.

William Timkey and Marten van Schijndel. 2021. All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4527–4546, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.372

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147. https://doi.org/10.3115/1119176.1119195

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Raúl Vázquez, Hande Celikkanat, Mathias Creutz, and Jörg Tiedemann. 2021. On the differences between BERT and MT encoder spaces and how to address them in translation tasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 337–347, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-srw.35

Elena Voita, Rico Sennrich, and Ivan Titov. 2019. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4396–4406, Hong Kong, China. Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1448

Ivan Vulić, Edoardo Maria Ponti, Robert Litschko, Goran Glavaš, and Anna Korhonen. 2020. Probing pretrained language models for lexical semantics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7222–7240, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.586

Gregor Wiedemann, Steffen Remus, Avi Chawla, and Chris Biemann. 2019. Does BERT make any sense? Interpretable word sense disambiguation with contextualized embeddings. *ArXiv*, abs/1909.10430.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20, Hong Kong, China. Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1002

Zeyu Yun, Yubei Chen, Bruno Olshausen, and Yann LeCun. 2021. Transformer visualization via dictionary learning: Contextualized embedding as a linear superposition of transformer factors. In *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 1–10, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.deelio-1.1

Sumu Zhao, Damián Pascual, Gino Brunner, and Roger Wattenhofer. 2021. Of non-linearity and commutativity in bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533563