Weight-based Decomposition: A Case for Bilinear MLPs

Michael T. Pearce [1] **Thomas Dooms** [1] **Alice Rigg** [1]

Abstract

Gated Linear Units (GLUs) have become a common building block in modern foundation models. Bilinear layers drop the non-linearity in the "gate" but still have comparable performance to other GLUs. An attractive quality of bilinear layers is that they can be fully expressed in terms of a third-order tensor and linear operations. Leveraging this, we develop a method to decompose the bilinear tensor into a set of sparsely interacting eigenvectors that show promising interpretability properties in preliminary experiments for shallow image classifiers (MNIST) and small language models (Tiny Stories). Since the decomposition is fully equivalent to the model's original computations, bilinear layers may be an interpretability-friendly architecture that helps connect features to the model weights. Application of our method may not be limited to pretrained bilinear models since we find that language models such as TinyLlama-1.1B can be finetuned into bilinear variants.

1. Introduction

Multi-Layer Perceptrons (MLP) are ubiquitous components of large language models (LLMs) and deep learning models generally. Because of their nonlinear activation function, MLPs have historically been treated as irreducible components in mechanistic interpretability (Elhage et al., 2021; Black et al., 2022). Nonlinear activations are a major limitation because 1) they introduce complicated interactions (typically of all orders) between input features and 2) the neurons act as a privileged basis (Elhage et al., 2022; Brown et al., 2023), so we cannot transform to a feature basis and cannot easily work with linear combinations of neurons. MLPs have only been well-understood in special cases such as small networks trained on math problems (Nanda et al., 2022; Chughtai et al., 2023; Michaud et al., 2024).

Recent interpretability work does not rely on parsing model weights. Instead, sparse autoencoders (SAEs) are used to derive interpretable features based on a model's internal activations over a dataset (Cunningham et al., 2024; Bricken et al., 2023b; Marks et al., 2024). Training transcoders to bridge features from one layer to those in the next ends up circumventing MLPs entirely (Jacob Dunefsky & Nanda, 2024; Sharkey, 2024). Although SAE-based approaches have been successful, there remain questions about their data dependency and their out-of-distribution behavior. Ideally, there would be a way to connect the SAE-derived features to the model's computations.

Another approach is to use a more interpretable architecture. Sharkey (2023) suggested that bilinear MLP layers of the form $g(x) = (W\mathbf{x}) \odot (V\mathbf{x})$ have advantages for interpretability because their computations can be expressed in terms of linear operations with a third order tensor. This allows us to leverage tensor or matrix decompositions to understand the weights. A comparison of activations found that bilinear activations outperformed ReLU and GELU in transformer models (Shazeer, 2020), and have performance only slightly below SwiGLU, which is prevalent in competitive open-source transformers today.

This paper aims to provide a demonstration of how bilinear MLP layers can be decomposed into a set of functionally relevant features. We use the word "feature" loosely to describe linear directions in activation space that appear interpretable, even if they are not necessarily sparsely activating. Our contributions are as follows:

1. We introduce a method to decompose bilinear MLPs into a set of eigenvector features with sparse interactions. This decomposition is fully equivalent to the layer's original computations. It is possible to chain these decompositions together to decompile a deep MLP-only model.
2. In shallow image classification (MNIST) models, we show that the top eigenvectors are interpretable, and smaller eigenvalue terms can be truncated while preserving performance. Weight decay and noisy training produce qualitatively more interpretable features.
3. Preliminary results on small language models (Tiny Stories) suggest that language eigenvectors can also be interpretable. We give evidence that larger pretrained

*Equal contribution [1]Independent. Correspondence to: Michael Pearce <michaelttpearce@gmail.com>.

language models can be finetuned into bilinear models using little data, to achieve loss similar to that expected from pretraining.

We open-source our code and provide notebooks to replicate all experiments on GitHub.

2. Related Work

Weight-based Mechanistic Interpretability. Reverse-engineering a model's inner workings from its weights has been a central goal of circuits-style research (Elhage et al., 2021; Chughtai et al., 2023; Olsson et al., 2022; Wang et al., 2022; Michaud et al., 2024). Our work is most similar in spirit to Cammarata et al. (2020; 2021), which extracted simple curve features in CNNs from model weights and described how high-order features are constructed from simpler ones. Our work shares this basic approach but for an MLP-only model.

Transcoders. Recently, transcoders have been used to derive sparse feature interactions by predicting an MLP layer's outputs from its inputs, effectively learning how to skip over the layer (Jacob Dunefsky & Nanda, 2024; Sharkey, 2024). Our work has a similar goal of describing an MLP layer's outputs in terms of a sparse set of interactions, but we derive the interactions and input features directly from the weights, although our features do not necessarily have sparse activations.

Matrix Decompositions. Various matrix decompositions have been used previously to understand neural network internals. In deep linear models, singular value decomposition (SVD) provides a nearly full description of how hierarchical semantic concepts are learned (Saxe et al., 2019). Non-negative matrix factorization has been used to cluster neuron activations effectively (Olah et al., 2018). For transformers, it's been noticed that SVD of model weights often yields interpretable feature directions (Beren & Black, 2022). More recently, SVD has been used to decompose gradients and identify a sparse interaction basis (Bushnaq et al., 2024a;b).

3. Background

We use a standardized notation as presented by Kolda & Bader (2009) throughout this paper. Scalars are denoted by a normal character s, vectors are denoted in bold $\mathbf{v}$, matrices as capital letters M, and third-order tensors as calligraphic script $\mathcal{T}$. The entry in row i and column j of a matrix M is a scalar and therefore denoted as m_{ij}. As is common in many computing libraries, we denote taking row i or column j of a matrix by $\mathbf{m}_{i:}$ and $\mathbf{m}_{:j}$ respectively. Additionally, for brevity, we use the term tensor to encompass all orders, wherever possible, we disambiguate. Lastly, we use $\odot$ to denote an element-wise product and $\cdot_{\text{axis}}$ to denote a product of tensors along the specified axis.

3.1. Gated Linear Units

Gated Linear Units (GLU), introduced in Dauphin et al. (2017), use three weight matrices, as opposed to two for an ordinary MLP. The output of an MLP layer using a GLU has the form $W_{\text{out}}g(\mathbf{x})$, such that

$$g(\mathbf{x}) = (W\mathbf{x}) \odot \sigma(V\mathbf{x}), \tag{1}$$

where σ is an activation function applied pointwise. Common choices for σ include Gaussian Error Linear Units, $\text{GELU}(x) = x\Phi(x)$, and $\text{Swish}_\beta(x) = x\text{sigmoid}(\beta x)$. Bilinear layers are the result of omitting the nonlinear activations σ.

3.2. Interaction Matrices

To illustrate the behavior of bilinear layers, we show how a single neuron activation $g(\mathbf{x})_a$ is computed.

$$\begin{aligned} g(\mathbf{x}) &= (W\mathbf{x}) \odot (V\mathbf{x}) \\ g(\mathbf{x})_a &= (\mathbf{w}_{a:}^T\mathbf{x})\ (\mathbf{v}_{a:}^T\mathbf{x}) \\ &= \mathbf{x}^T \left(\mathbf{w}_{a:}\mathbf{v}_{a:}^T\right) \mathbf{x} \end{aligned}$$

We call the term $B_{a::} = \mathbf{w}_{a:}\mathbf{v}_{a:}^T$ an *interaction* matrix. This matrix defines how each pair of inputs interact for a given output. There is an interaction matrix for each output neuron, resulting in a third-order tensor $\mathcal{B}_{\text{out,in,in}}$ given by $b_{aij} = w_{ai}v_{aj}$.

3.3. Symmetry

Since $g(\mathbf{x})_a = \mathbf{x}^T B_{a::}\mathbf{x}$ is a scalar, we can write

$$\begin{aligned} g(\mathbf{x})_a &= g(\mathbf{x})_a^T \\ &= \mathbf{x}^T B_{a::}^T \mathbf{x}. \end{aligned}$$

Consequently, only the symmetric part of the interaction matrix, defined as $B_{a::}^{sym} = \frac{1}{2}(B_{a::} + B_{a::}^T)$, contributes to the output of $g(\mathbf{x})_a$. The symmetric form will be useful because it has a nice eigenvalue decomposition. From here on, any interaction matrix is assumed to be in symmetric form, dropping the superscript.

Theorem 3.1 (Spectral Theorem)**.** *If $Q : \mathbb{R}^d \to \mathbb{R}^d$ is a real, symmetric matrix, then there exists an orthonormal basis of $\mathbb{R}^d$ consisting of eigenvectors of Q. Each eigenvalue is real. That is, $Q = P^T\Lambda P$, where Λ is a real diagonal matrix, and P is a real orthogonal matrix.*

3.4. Biases and Residuals

Both biases and residuals can be included in the weights in a way that requires no change to our analysis method.

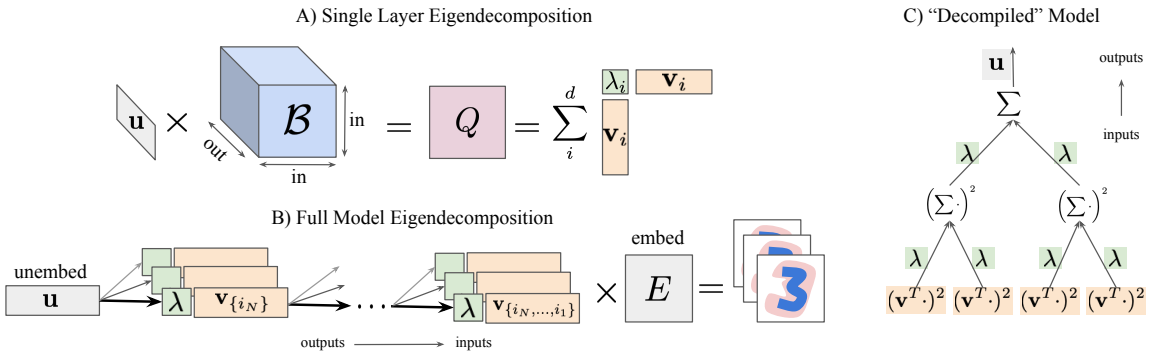

Figure 1. Illustration of the eigenvalue decomposition. **A)** For a single layer, we start with a vector $\mathbf{u}$ in the output space of the third-order tensor $\mathcal{B}$. Their dot product gives a symmetric interaction matrix Q. Eigendecomposition of Q gives an orthonormal set of eigenvectors $\mathbf{v}_i$. In this basis, the interactions are sparse since an eigenvector only interacts with itself. **B)** For a fully bilinear model, we start with an unembedding vector $\mathbf{u}$ (e.g., the logit directions for the digit "3") and repeatedly apply the single-layer eigendecomposition. Each layer k eigenvector $\mathbf{v}_{\{i_N,\ldots,i_k\}}$ acts as the output vector that determines a set of layer k-1 eigenvectors $\mathbf{v}_{\{i_N,\ldots,i_k,i_{k-1}\}}$. Using the embedding weights, the layer-1 eigenvectors can be transformed into input features. **C)** Schematic of a 2-layer model after full decomposition showing the tree-like computational graph, with a branching factor of 2 (instead of d_model) for simplicity. The model is *decompiled* in the sense that interactions have been sparsified and made explicit through the graph. Only the layer-1 eigenvectors are needed to get the initial activations from the post-embedding inputs. The eigenvalue magnitudes parameterize the importance of the edges.

Biases can be incorporated as an additional column: $W' = [W; \mathbf{b}]$ and $\mathbf{x}' = [\mathbf{x}, 1]$. Residuals are slightly trickier but can be included by setting $\mathbf{x}' = [\mathbf{x}, \mathbf{x}]$, $W' = [W, I]$ and $V' = [V, 1]$ where I is an identity matrix and 1 is fully zero, except for ones on the bias column. Finally, since this doubles the output dimensions, we append a doubled identity matrix, whose role is to sum both terms. We have found both techniques to yield interpretable results, however, for simplicity, this paper focuses on the MLP-only case without biases.

4. Analysis Techniques

4.1. Single-Layer Eigendecomposition

Directly studying the third-order tensor $\mathcal{B}$ is possible through higher-order tensor decompositions (De Lathauwer et al., 2000; Kolda & Bader, 2009), which we describe in subsection B.2.

Higher-order techniques typically rely on first flattening the tensor into a matrix. Instead of flattening, we can reduce $\mathcal{B}$ using a vector $\mathbf{u}$ in the output space of $\mathcal{B}$. This yields a matrix $Q = \mathbf{u} \cdot_{\text{out}} \mathcal{B}$ that describes the interactions between input pairs and determines the output along the $\mathbf{u}$-direction. An advantage of reduction over flattening is that we can use meaningful $\mathbf{u}$ vectors, such as unembedding vectors that give the logits for specific output classes (such as the MNIST digits) or downstream feature vectors.

As illustrated in Figure 1, we can diagonalize the interaction matrix by taking its eigenvalue decomposition. Since Q can be taken as symmetric (see subsection 3.3), the spectral theorem provides a decomposition of the form

$$Q = \sum_i^d \lambda_i \, \mathbf{v}_i \mathbf{v}_i^T \tag{2}$$

with a set of d (model dim) orthonormal eigenvectors $\mathbf{v}_i$ and real-valued eigenvalues λ_i.

Since an eigenvector only interacts with itself, the eigenvectors form a basis with sparse interactions, and the eigenvalues λ_i parameterize the interaction strengths. In this basis, the output in the $\mathbf{u}$-direction is

$$\mathbf{x}^T Q \mathbf{x} = \sum_i^d \underbrace{\lambda_i \, (\mathbf{v}_i^T \mathbf{x})^2}_{\text{activation for } \mathbf{v}_i} \tag{3}$$

where each term can be considered the activation for the eigenvector $\mathbf{v}_i$.

If we have a set of $\mathbf{u}$-vectors, it's possible to re-express $\mathcal{B}$ in terms of the eigenvectors, which amounts to a change of basis, as described in subsection B.1.

4.2. Full Model Eigendecomposition

Figure 1 (B) shows how a deep MLP-only model with bilinear layers can be decomposed by starting with the unembedding vectors $\mathbf{u}$ and repeatedly applying the single-layer eigendecomposition, moving in the direction of outputs to inputs. Each layer k eigenvector $\mathbf{v}_{\{i_N,\ldots,i_k\}}$ acts as the output vector that determines a set of layer k-1 eigenvectors $\mathbf{v}_{\{i_N,\ldots,i_k,i_{k-1}\}}$. Finally, the layer 1 eigenvectors can be

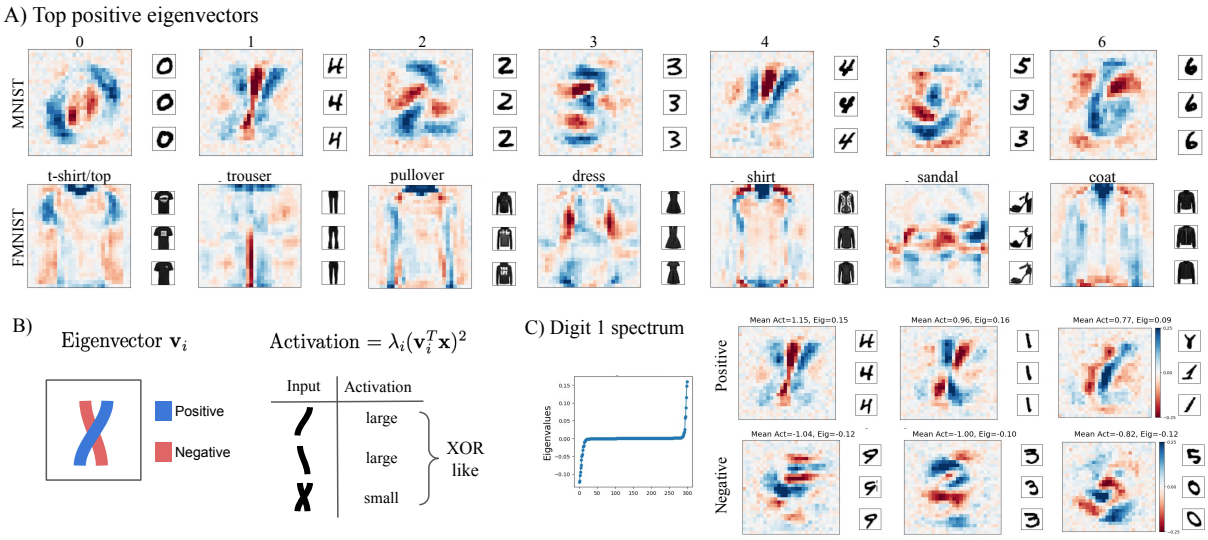

Figure 2. Eigenvectors for single-layer MNIST and Fashion-MNIST models. **A)** Top positive eigenvectors by mean activation for the first seven output classes. To the right of each eigenvector are the 3 inputs with the highest activations. Ordering by activation favors eigenvectors with more "complete" images that overlap more with the inputs. Often the top activating eigenvector has the top or second top eigenvalue. **B)** Because of the square dot product in an eigenvector's activation, $\lambda_i(\mathbf{v}_i^T\mathbf{x})^2$, its overall sign is arbitrary, so we choose the sign so that the positive component (blue) is similar to the top inputs. If an input separately has a high overlap with the positive (blue) or negative (red) component then the activation can be large, but a high overlap with both components cancels out. This behavior is similar to an XOR. **C)** The top three positive and negative eigenvectors for digit 1. The top positive eigenvalue has strong activations for digit 4, but these likely cancel out with activations for the top negative eigenvalue which detects horizontal lines in the center of an image.

transformed into input features using the embedding weights and then interpreted.

The decomposition process produces a tree-like computational graph for each starting unembedding vector, shown schematically in Figure 1 (C). Only the input features (layer-1 eigenvectors) are needed to get initial activations from the post-embedding inputs. We call this model *decompiled* because the interactions between input features are explicitly captured in the graph, for example, two input features interact only in the layer where their branches join. The eigenvalue magnitudes parameterize the importance of edges and would allow the most important subgraphs to be identified.

For each layer-1 eigenvector, it's possible to define an effective eigenvalue describing its overall importance. Because an eigenvalue at a deeper layer (an upper branch in Figure 1 C) is shared by all the layer-1 eigenvectors at the leaves of its tree, we can pass its value through the computational graph to shallower layers. For a specific layer-1 eigenvector, let λ_ℓ be its eigenvalue from layer ℓ. Then the effective eigenvalue is

$$\tilde{\lambda} = \lambda_1 \prod_{\ell>1} |\lambda_\ell|^{\left(\frac{1}{2}\right)^{\ell-1}} \tag{4}$$

where the square roots compensate for the squaring operations in the decompiled computational graph. In general, the number of layer-1 eigenvectors grows exponentially as $(\text{d\_model})^{\text{n\_layer}}$ but it's unclear how the fraction of significant effective eigenvalues scales. In section 7 we discuss ideas for finding shared features among the eigenvectors so that repeated decomposition does not lead to an exponential number.

5. Image Classification

We demonstrate our approach on the MNIST dataset of handwritten digits and the Fashion-MNIST dataset of clothing images. We use a model consisting of one or two bilinear layers with linear embedding and unembedding layers. Unless specified otherwise, the model dimension is 300. Early experiments showed that biases and normalization layers can be incorporated into the decomposition with similar results, but for simplicity, we have not included them in the models used for the results presented here. Details of the training setup can be found in Appendix A.

5.1. Single-Layer Eigenvectors

Figure 2 shows the top positive eigenvectors for the output classes in each dataset. Since the activation of an eigenvector, $\lambda_i(\mathbf{v}_i^T\mathbf{x})^2$, contains a squared dot product, both positive and negative components can contribute to large activations. Cancellations between them lead to small activations. So

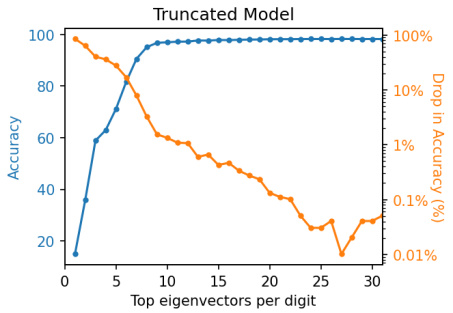

Figure 3. The validation accuracy for an MNIST model truncated to the top k eigenvectors by eigenvalue magnitude.

the form of the activation gives an XOR-like structure, as illustrated in (B). The squared dot product ensures that the activations for positive eigenvectors are always positive, and those for negative eigenvectors are always negative.

Many of the MNIST eigenvectors resemble the full digit (0,2,3,6) while others emphasize certain parts (4,5). The FMNIST eigenvectors focus much more on specific parts, for example, the neck and sleeve areas on shirts/pullovers/coats and the leg gap for trousers.

A portion of the eigenvalue spectrum is shown for digit 1 in Figure 2(C). Only a small fraction of eigenvalues have significant magnitude. For the first positive eigenvector, the top three inputs by activation are all 4s. The first negative eigenvector activates on horizontal lines in the middle, so we expect it to cancel out the activations on 4s for the first positive eigenvector. The other positive eigenvectors contain 1-like features at different slants. Similar plots for the other digits can be found in Appendix E.

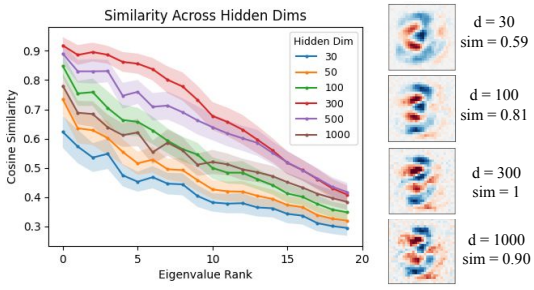

Figure 4. Cosine similarities for eigenvectors that best match the eigenvectors from a $d = 300$ model at each rank. Results are averaged over 20 initializations per model size with the shaded region showing the p95 confidence interval. The eigenvectors for digit 3 show that moderate similarities can correspond to visually similar features. Results shown are only for positive eigenvectors and similarities are computed for eigenvectors transformed to the input basis using the embedding weights.

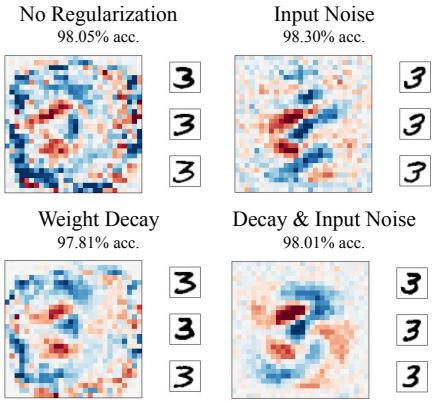

Figure 5. Comparison of different regularizations. Input noise regularization adds dense Gaussian noise to the inputs. Input noise is found to remove the large values in the periphery of the image and improve validation accuracy. Weight decay (parameter=0.5) reduces the fine-scale noisiness in the images and reduces performance. The paper results are from models with weight decay and input noise.

Figure 3 shows the validation accuracy for an MNIST model truncated to the top-k eigenvectors. Nearly full performance (less than 1% drop in accuracy) is recovered when only 10 eigenvectors (in total, both positive and negative) per digit are kept. Generally, the drop in accuracy decreases exponentially with the number of eigenvectors, supporting the idea that the eigenvalues usefully parameterize the importance of a feature.

Across different model sizes and initialization, it is possible to find very similar input features. Figure 4 looks at the cosine similarity of the best match to the positive eigenvectors in a $d = 300$ model. The best matches are for models of the same size, with around 0.9 similarity for the top five or so eigenvectors per digit (roughly corresponding to a truncated model with only a 1% drop in accuracy). Across model sizes, similar sizes have more similar eigenvectors but even a small $d = 30$ model has similarities between 0.5-0.6 for the top eigenvectors. In high dimensions, such moderate cosine similarities can still correspond to visually similar features, as shown for digit 3 in Figure 4.

5.2. Regularization

Regularizing the model is important for getting more visually interpretable input features. We use a combination of weight decay and input noise regularization, based on dense Gaussian noise. The motivation for input noise is to penalise the over-reliance on spurious correlations [1]. Additionally,

[1]Within MNIST, some pixels are only active on a handful of samples and are therefore highly indicative of a prediction, without being "robust"

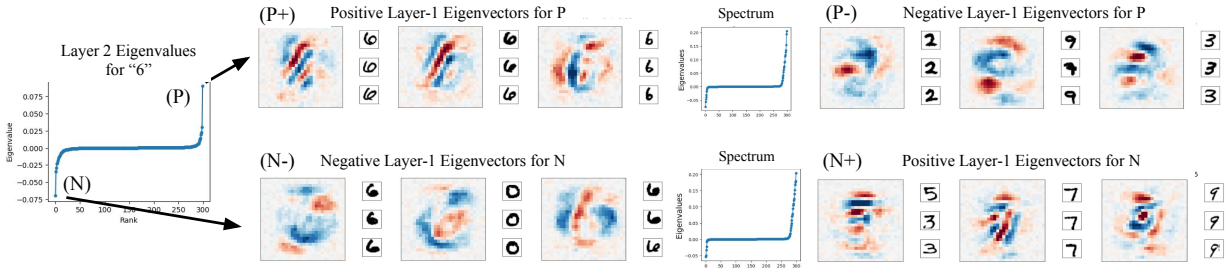

Figure 6. Eigendecomposition of a two-layer MNIST model. Across digits, there is typically one prominent positive (P) and one prominent negative (N) layer-2 eigenvalue. For the example of digit 6, we show the top eigenvectors for (P) and (N), showing a pattern where (P+) eigenvectors have sharp lines specific to the output digit 6 while (N-) has vague shapes resembling 6. It's important to note that (N-) contributes negatively to the logits for 6 since the sign is determined solely by the layer-2 eigenvalue. We hypothesize that non-6 inputs that resemble 6 will activate the (N-) eigenvectors and cancel out any overlap with the (P+) eigenvectors but true 6 inputs will activate (P+) much more strongly than (N-). The other eigenvectors (P-) and (N+) show a similar but reversed pattern for non-6 digits.

learned representations become sparser, without using an L1 norm (Bricken et al., 2023b).

Adding input noise alone improved validation accuracy from 98.05% to 98.30%, likely indicating better generalization. Figure 5 shows that the features without regularization are noisy and have large values on the periphery where pixels are rarely "on" which may be due to overfitting to outlier images. Training with input noise removes the peripheral values and noisiness and adding weight decay further cleans up the features. With both types of regularization, the performance is the same as without regularization. These results suggest that regularization during training may be important for finding more interpretable weights among the large degeneracy of weights with the same performance.

5.3. Two-Layer Eigendecomposition

Following the procedure outlined in subsection 4.2, we decompose a two-layer MNIST model. The spectra of the layer-2 eigenvalues typically have one prominent positive eigenvalue (P) and one negative one (N), as seen in the spectrum for 6 in Figure 6. The layer-1 eigenvectors for (P) and (N) have an interesting pattern. Focusing on the example of digit 6, the positive layer-1 eigenvectors (P+) have sharp lines specific to 6 while the (P-) eigenvectors have vague non-6 shapes. Likewise, (N+) has sharp non-6 lines while (N-) has vague 6 shapes. This same pattern is seen across digits.

It's important to note that both the negative (N-) and positive (N+) eigenvectors of (N) contribute negatively to the logits of 6 since the sign of the contribution is determined by the sign of the layer-2 eigenvalue. So it's interesting that (N-) resemble the digit 6. We hypothesize that this pattern is a way to better differentiate true 6s from similar non-6s. A faux-6 input will activate the vague 6-like shapes in (N-) and cancel out any activation of (P+) eigenvectors. A true-6 will activate some of the (P+) eigenvectors strongly, overcoming any activation of the (N-) eigenvectors.

5.4. Limitations to interpretability

Even though the top eigenvectors appear largely interpretable, we expect there to be potential limitations to interpretability that arise in other contexts. One issue is polysemanticity within eigenvectors. For example, in the spectrum for digit 1 in Figure 2, it is clear that the positive and negative components, particularly for the second positive eigenvector, can represent 1's with different slopes. This polysemanticity is possible due to the XOR-like behavior of the squared dot product in an activation of the eigenvector. More generally, sparsity in the inputs as seen for LLMs (due to vocab size and attention) will likely encourage polysemanticity in the eigenvectors as well.

6. Language Models

In this section, we present the preliminary results of our techniques on a language modeling task. We use a single-layer transformer model in which the MLP is replaced with a bilinear version. We train this model on the Tiny Stories dataset (Eldan & Li, 2023). We make several simplifications for ease of analysis; importantly, these modifications are not required. First, we use a simple uncased WordPiece (Wu et al., 2016) tokenizer with a vocabulary size of 4096, created specifically for this dataset. Second, we omit all normalization layers and biases. The full details of our architecture and training setup are located in Appendix C.

6.1. Bigrams & Skip-Trigrams

Since the bilinear layer parameterizes interactions between pairs of inputs, it's easy to read off important interactions. For example, we can find how the MLP contributes to the

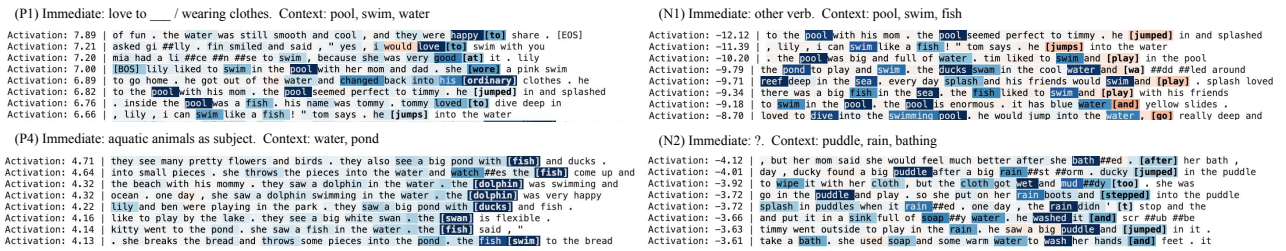

Figure 7. Top input activations for eigenvectors of a single-layer Tiny Stories model. The output direction used is the unembedding of the "swim" token minus the average unembedding for other verbs ["run", "climb", "eat", "see", "smell", "walk", "fly", "sit", "sleep"], which helps remove grammatical features that apply to any verb. We show the first (P1) and fourth (P4) top positive eigenvectors and the first (N1) and second (N2) top negative ones. The focal token is indicated by brackets ("[token]") and each token's coloring indicates its contribution to the eigenvector's activation, either positive (blue) or negative (red), as mediated through attention. The feature descriptions are divided into immediate (previous few tokens) and broader contexts. See Figure 13 for additional eigenvectors and see Figure 12 for the eigenvectors for only "swim"

Preceding		Following
little, the, young, small, poor, good, birthday, another, first, kind	**girl**	named, who, nodded, called, hugged, touched, names, cried, s, name

Table 1. The 10 highest weights corresponding to preceding and following tokens for "girl". This considers both the direct path and the path through the MLP.

learned bigrams. In ordinary models, bigrams are approximated by the direct path between the embedding and unembedding. In a bilinear layer, the direct path token through attention interacts with itself, which also contributes to the bigram. This interaction is captured in the diagonal of each interaction matrix, $B_{:ii}$, which combined with the residual path represents a more complete picture of the bigrams the model has learned. Given that no normalization is present, adding this diagonal to the residual requires no approximation. Additionally, this matrix can be transposed to show the most likely preceding tokens, as seen in Table 1.

Generally, the embeddings learn the clearest bigrams while the MLP focuses on disambiguating bigrams related to context. For instance, it strongly impacts the token "pre", which can have multiple sensible continuations ("##cisely", "##pare", "##fer") along with other compound tokens as shown in Table 6. Furthermore, it is also strongly involved in punctuation, which can also be seen as a form of context-aware bigrams. We find that the MLP diagonal can sometimes contain strange bigrams with high values, possibly due to noise or the inherent low rank of the interaction matrices. We defer a deeper study of this phenomenon to further work.

In a single-layer model, tokens passing through attention followed by the unembeddings will capture skip-trigrams of the form "A...BC", as described in Elhage et al. (2021). In bilinear models, we can find additional contributions to skip-trigrams from the MLP by analyzing the interaction of a previous token "A" that passes through attention, and the direct path "B". The "A" token will be transformed by the OV matrix for the specific head it passes through. For a specific output token "C", we can find the top interactions between "A" and "B" in the interaction matrix.

We show the effectiveness of this approach by uncovering skip trigrams ending in a quote token. In this simple setup, quotes are exclusively predicted to open or close reported speech. This reveals that 99 out of the top 100 weights in head 0.2 correspond to skip trigrams of the form talk-related verb (say, shout, whisper) followed by a comma or a colon. The closing of the quote is performed by several heads, most notably by head 0.5, which fires on trigrams of the form quote, punctuation. We show comprehensive tables in the appendix (Appendix F)

6.2. Eigenvectors

In a single-layer model, the MLP goes beyond skip trigrams by capturing interactions between the tokens that pass through the attention layer. The interactions help pick up on higher n-gram statistics, for example, the broader context of a given token. With a bilinear layer, we can read off specific token interactions, but it's difficult to describe its overall behavior. The eigenvector decomposition described in subsection 4.1 can be a starting place for a more general understanding.

A preliminary investigation of the eigenvectors for the single-layer Tiny Stories model shows that the eigenvectors generally capture relevant aspects for predicting a given output token. Figure 7 shows the top input activations for the eigenvectors for the output "swim" and each token's contributions to the activation. Initially, we found that the

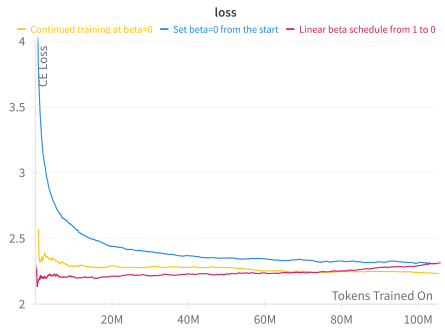

Figure 8. The result of two training runs, both trained on approximately 105 million tokens of FineWeb data. Blue: setting β to 0 (bilinear activation) from the start. Red: Linearly scheduling β over the course of the same training to transition from SiLU to bilinear. Both runs converge to the same loss. Yellow: Further training on the endpoint of the Red run with $\beta = 0$ fixed. The decreasing loss shows the model had not full converged by the end of the linear β scheduling.

eigenvectors picked up on grammatical features that could apply to any verb, shown in Figure 12, so we instead show results for the unembedding of "swim" minus the average unembedding of some other verbs.

The eigenvectors' top activations appear mostly monosemantic. For example, having aquatic animals as subjects (P4) or non-swimming contexts for water such as rain, puddles, and baths (N2). There is a strong negative eigenvector (N1) that inhibits an output of "swim" based on the context of "swim"-related words. In fact, this pattern is common across different output tokens. We hypothesize that the strong negative eigenvector cancels out skip-trigram (or perhaps skip-bigram) predictions already in the residual stream, in favor of higher-order predictions from the MLP. We find similar results for the eigenvectors for outputting quotation marks, shown in Figure 11.

The top positive eigenvector for "swim" (P1) is somewhat polysemantic, since it activates for the immediate context of "love to ___" which is appropriate for verbs and the context of putting on clothes, appropriate for "swim" as the start of "swimsuit". Both rely on a broader context of being at the pool or near water.

6.3. Finetuning

An interpretable architecture can only see widespread adoption if it is both comparably performant and efficient to train. In this section, we provide evidence that bilinear transformers may satisfy both conditions, as pretrained models can be finetuned into bilinear variants while using significantly less compute compared to pretraining.

Many state-of-the-art open-source models use a gated SiLU activation.[2] Since Swish_β interpolates between common activation functions including SiLU ($\beta = 1$) and linear ($\beta = 0$) we can use a gated Swish, as in Equation 1, to finetune between the gated SiLU and bilinear activations.

We finetuned TinyLlama-1.1B, a 1.1 billion parameter transformer model pretrained on 3 trillion tokens of data, using a single A40 GPU. We finetuned the model in two ways. The first approach is to set β to 0 at the start and perform standard finetuning. For the second approach, we implemented a linear scheduler to the β parameter, gradually decreasing it from 1 to 0 over the course of training.

Figure 8 shows the results of both training strategies. Throughout training on the same number of tokens, both runs converge to the same end loss. In yellow, we see that we benefit from continued training. More details on training can be found in Appendix D.

7. Discussion

We have presented a novel method to decompose bilinear MLPs into sets of eigenvectors by leveraging their close-to-linear structure. In the case of MNIST digit classification the top eigenvectors appear largely interpretable and help reveal interesting structure in a two-layer model. Preliminary results for a Tiny Stories language model suggest the top eigenvectors may be interpretable also.

The method has several attractive properties for mechanistic interpretability:

1. The eigenvectors are derived entirely from the model weights and are fully equivalent to the model's original computations. So there are no concerns about data dependence or unexplained error terms, as is currently the case for sparse autoencoders.

2. The eigenvector basis effectively sparsifies interactions since an eigenvector only interacts with itself.

3. The eigenvalues naturally parameterize each eigenvector's importance and show that MNIST models with small eigenvalues truncated retain good performance.

Our method, however, requires an architecture change since it only works for bilinear MLPs. The change may be minor though, since Shazeer (2020) showed that transformers with bilinear activations perform on par with other activations: slightly worse than SwiGLU but better than ReLU. We show that LLMs can be finetuned from a SiLU to bilinear and achieve the expected loss for a pretrained bilinear model. This opens up the possibility of applying our methods to

[2] LLaMA, Mistral, Mixtral, PaLM all use SiLU. Gemma uses a GELU, which can be approximated by SwiGLU with $\beta = 1.702$.

existing models by first finetuning them to have bilinear activations.

Along the lines of Bricken et al. (2023a), we found that adding Gaussian input noise during training helped improve the interpretability of the eigenvectors and improved model performance. Regularization may be an under-explored avenue for improving weight-based interpretability.

A limitation of the eigenvector decomposition is that the eigenvectors are not expected to be fully monosemantic, as discussed in subsection 5.4. We see evidence that MNIST eigenvectors contain different orientations of a digit in their positive and negative components. The eigenvector decomposition may be a good starting place for applying other interpretability methods, such as sparse autoencoders (SAEs). Since eigenvectors have sparse interactions, combining them with SAEs may effectively produce a transcoder with sparse interactions between sparse features, similar to those used in Jacob Dunefsky & Nanda (2024). Explorations in this direction are left to future work.

Application of our method is limited to shallow models because the number of eigenvectors grows as $(\text{d\_model})^{\text{n\_layers}}$. Currently, each output direction used in the decomposition leads to a distinct set of eigenvectors. Future work into clustering eigenvectors in order to find shared features among different eigenvector sets would help prevent the growth of features and enable the analysis of deeper models. Given that the eigenvalues parameterize the importance of feature directions another direction is to use search algorithms to identify the most important computational paths through the model.

Acknowledgements

We are grateful to Lee Sharkey whose paper on bilinear models inspired this work and who offered helpful comments on an early version of this work. We would also like to thank José Oramas, Narmeen Oozeer and Nora Belrose for useful feedback on the draft. We are grateful to the AI Safety Camp program where this work first started. We thank CoreWeave for providing compute for the finetuning experiments. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

Contributions

Michael performed the bulk of the work on the MNIST analysis and provided valuable insights across all presented topics. Thomas worked on the Language Models section and was responsible for code infrastructure. Alice conducted the fine-tuning project. The paper was written in tandem, each focussing on their respective section.

References

Beren and Black, S. The singular value decompositions of transformer weight matrices are highly interpretable, 2022. URL https://www.lesswrong.com/posts/mkbGjzxD8d8XqKHzA/the-singular-value-decompositions-of-transformer-weight.

Black, S., Sharkey, L., Grinsztajn, L., Winsor, E., Braun, D., Merizian, J., Parker, K., Guevara, C. R., Millidge, B., Alfour, G., and Leahy, C. Interpreting Neural Networks through the Polytope Lens. *CoRR*, November 2022. URL https://arxiv.org/abs/2211.12312. Publisher: arXiv Version Number: 1.

Bricken, T., Schaeffer, R., Olshausen, B., and Kreiman, G. Emergence of sparse representations from noise. In *International Conference on Machine Learning*, pp. 3148–3191. PMLR, 2023a.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N. L., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread*, October 2023b. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Brown, D., Vyas, N., and Bansal, Y. On privileged and convergent bases in neural network representations. *arXiv preprint arXiv:2307.12941*, 2023.

Bushnaq, L., Goldowsky-Dill, S. H. N., Braun, D., Mendel, J., Hänni, K., Griffin, A., Stöhler, J., Wache, M., and Hobbhahn, M. The local interaction basis: Identifying computationally-relevant and sparsely interacting features in neural networks. *arXiv preprint arXiv:2405.10928*, 2024a.

Bushnaq, L., Mendel, J., Heimersheim, S., Braun, D., Goldowsky-Dill, N., Hänni, K., Wu, C., and Hobbhahn, M. Using degeneracy in the loss landscape for mechanistic interpretability. *arXiv preprint arXiv:2405.10927*, 2024b.

Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M., and Olah, C. Curve detectors. *Distill*, 2020. doi: 10.23915/distill.00024.003. https://distill.pub/2020/circuits/curve-detectors.

Cammarata, N., Goh, G., Carter, S., Voss, C., Schubert, L., and Olah, C. Curve circuits. *Distill*, 2021. doi: 10.23915/distill.00024.006. https://distill.pub/2020/circuits/curve-circuits.

Chughtai, B., Chan, L., and Nanda, N. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pp. 6243–6267. PMLR, 2023.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. Sparse Autoencoders Find Highly Interpretable Features in Language Models. *ICLR*, January 2024. doi: 10.48550/arXiv.2309.08600. URL http://arxiv.org/abs/2309.08600. arXiv:2309.08600 [cs].

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks, 2017.

De Lathauwer, L., De Moor, B., and Vandewalle, J. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000. doi: 10.1137/S0895479896305696. URL https://doi.org/10.1137/S0895479896305696.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english?, 2023.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., and others. Toy Models of Superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Jacob Dunefsky, P. C. and Nanda, N. Transcoders enable fine-grained interpretable circuit analysis for language models, 2024. URL https://www.alignmentforum.org/posts/YmkjnWtZGLbHRbzrP/transcoders-enable-fine-grained-interpretable-circuit.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Marks, S., Rager, C., Michaud, E. J., Belinkov, Y., Bau, D., and Mueller, A. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024.

Michaud, E. J., Liao, I., Lad, V., Liu, Z., Mudide, A., Loughridge, C., Guo, Z. C., Kheirkhah, T. R., Vukelić, M., and Tegmark, M. Opening the ai black box: program synthesis via mechanistic interpretability. *arXiv preprint arXiv:2402.05110*, 2024.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2022.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Saxe, A. M., McClelland, J. L., and Ganguli, S. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116(23):11537–11546, 2019.

Sharkey, L. A technical note on bilinear layers for interpretability. 2023.

Sharkey, L. Sparsify: A mechanistic interpretability research agenda, 2024. URL https://www.alignmentforum.org/posts/64MizJXzyvrYpeKqm/sparsify-a-mechanistic-interpretability-research-agenda.

Shazeer, N. Glu variants improve transformer, 2020.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

A. MNIST Setup

Architecture	
d_model	300
d_hidden	300
n_layer	1-2
Training Parameters	
dropout	0.0
weight decay	0.5 (MNIST), 1.0 (F-MNIST)
input noise	1.0 **batch size**
100	
learning rate	0.001
optimizer	AdamW
schedule	exponential decay
epochs	20-100

Table 2. Model architecture and training setup for the MNIST models, unless otherwise stated in the text.

Architecture. Our model consists of a linear embedding layer, the bilinear MLP layers, and a linear unembedding layer. For ease of analysis we removed biases in the embedding and bilinear MLP layers and did not include normalization layers. Early experiments showed that these features did not improve model performance in our shallow models.

Training. For training we used the MNIST dataset [site] consisting of 60,000 handwritten digits. We did not use ordinary data augmentation to increase the dataset size. Instead, we added Gaussian input noise to the input.

B. Tensor Decompositions

B.1. Change of basis

Given a complete or over-complete set of m $\mathbf{u}$-vectors, we can re-express $\mathcal{B}$ in terms of the eigenvectors, which amounts to a change of basis. To avoid multiple contributions from similar $\mathbf{u}$-vectors, we have to use the pseudo-inverse which generalizes the inverse for non-square matrices. Taking the $\mathbf{u}$-vectors as the columns of U, the pseudo-inverse U^+ satisfies $UU^+ = I$, as long as U is full rank (equal to d). Then

$$\mathcal{B} = \sum_k^m \mathbf{u}^+_{:k} \otimes Q_k \tag{5}$$

$$= \sum_k^m \sum_i^d \lambda_{\{k,i\}}\ \mathbf{u}^+_{:k} \otimes \mathbf{v}_{\{k,i\}} \otimes \mathbf{v}_{\{k,i\}} \tag{6}$$

where $\mathbf{u}^+_{:k}$ are the rows of U^+. We can then recover the interaction matrices from $Q_k = \mathbf{u}_{k:} \cdot_{\text{out}} \mathcal{B}$ using the fact that $\mathbf{u}_{k:} \cdot \mathbf{u}^+_{:k'} = \delta_{kk'}$ (Kronecker delta).

B.2. Higher order SVD

Instead of working backwards from the outputs, we can use higher-order tensor decompositions on $\mathcal{B}$ for a given layer in isolation. The simplest approach that takes advantage of the symmetry between the inputs of $\mathcal{B}$ is to reshape the tensor, flattening the two input dimensions to produce a $d_{\text{out}} \times d_{\text{in}}^2$ shape matrix, and then do a standard singular value decomposition (SVD). Schematically, this gives $B_{\text{out,in}\times\text{in}} = \sum_i \sigma_i\ \mathbf{r}^{(i)}_{\text{out}} \otimes Q^{(i)}_{\text{in}\times\text{in}}$, where Q can still be treated as an interaction matrix and further decomposed into eigenvectors as described in subsection 4.1.

C. Tiny Stories Setup

Given that the main goal of this project is to create interpretable (not capable) models. This section delves into all architectural and training tradeoffs in designing our language models for maximal interpretability. Note that all claims in this section are based on superficial ablations. Lastly, we provide anecdotal insights that may be of interest.

C.1. Hyperparameters

Architecture	
d_model	1024
d_hidden	3072
d_head	128
n_head	8
n_layer	1
n_ctx	256
n_vocab	4096
parameters	22,020,096
Training Parameters	
dropout	0.0
weight decay	0.1
batch size	128
learning rate	0.001
optimizer	AdamW
schedule	linear decay
epochs	5
tokens	$\pm$ 2B
Miscellaneous	
pos_emb	rotary
initialisation	gpt2

Table 3. Model architecture and training setup. Omitted parameters are the HuggingFace Trainer defaults.

C.2. Architecture

The field of mechanistic interpretability is converging towards the limits of circuit-based analysis. This has prompted the use of leaky abstractions, such as sparse autoencoders, which may conceal noteworthy behavior. Therefore, this paper takes a different approach and modifies the architecture to obviate approximations.

MLP. Ordinarily, the most complex object to interpret in a transformer model is the MLP. Within our architecture, we replace this with a similar bilinear variant. Specifically, with a bilinear layer, followed by a linear out projection: $P(W\mathbf{x} \odot V\mathbf{x})$. Notably, keeping all else equal, this change results in 33% more parameters in the model. As a trade-off, the expansion factor can be set to 3 (instead of 4), resulting in only an 8% increase. From limited experiments, we found that bilinear MLPs outperform ReLU-based versions with equal parameter count, in line with Shazeer (2020).

Normalization. Empirically, we found that using normalization in our single-layer model has no impact on validation loss. Therefore, this component is removed in all locations. This strengthens the evidence that normalization is not necessary for the model but rather acts as a training regularize. However, we wish to note that including and ignoring it did not change our interpretation results significantly.

Biases. There seems to be little consensus on whether biases are useful in language models. In our setup, we found this not to be the case, consequently, they are excluded.

Width. Increasing the model width seems to improve both the accuracy and the interpretability.

C.3. Dataset

The TinyStories dataset (Eldan & Li, 2023) is great for interpretability purposes. It narrows down the scope of language modeling to its bare necessities. This allows the use of tiny models, which can coherently complete this task. Unfortunately, the dataset has several undesirable properties, we outline these and discuss our solution.

Noise. Many stories are littered with random symbols between words. While the story itself is coherent, this unnecessarily complicates the task. We remedy this by applying Unicode normalization, stripping all accents, and removing all remaining non-ASCII symbols.

Vocabulary. Stories are automatically generated by GPT-3.5 and GPT-4. Unfortunately, the instruction to use a simple vocabulary is occasionally violated. Consequently, the dataset contains a considerable amount of complex words (additionally, it also contains spelling mistakes). We did not make efforts to clean this.

Contamination. For unknown reasons, the TinyStories dataset has a high degree of data contamination. About 15% of training samples are equivalent and about 30% of the validation set is also found in the training set. We solve this by combining both training and validation, filtering out identical entries, reshuffling, and resplitting.

C.4. Tokenizer

Most language models use tokenizers designed to achieve a good balance between accuracy and throughput. This commonly leads to complexities in terms of interpretability; specific words can be tokenized differently depending on a preceding space, capital letter, and so on. Again, we adjust our approach to maximize interpretability. Specifically, this encompasses training and using an interpretability-first tokenizer, this is achieved as follows.

1. Split all non-alphabetic characters (punctuation, numbers, etc...) into separate tokens.
2. Split between all whitespaces (whitespace is implicit between each token).
3. Lowercase all characters (to avoid different tokenization on capital letters).
4. Use WordPiece instead of BPE (the former results in cleaner splits).

We optimize the tokenizer specifically on the TinyStories dataset. We find that using a vocabulary size of 4096 results in most common words being represented as a single token.

C.5. Capability Analysis

Given all the modifications to the model, architecture, tokenizer, and dataset, providing a meaningful quantitative analysis of model performance compared to Eldan & Li (2023) is challenging. We consider the automated analysis of creativity and coherence out of the scope of this work. However, to provide insight into the capabilities of the model, we share some generated text snippets. Note that this is generated by a single-layer model. Deeper models with fewer parameters easily outperform this.

> once upon a time, there was a little girl named lily. she loved to play outside in the sunshine. one day, she saw a big, red ball in the sky. she wanted to play with it, but it was too high up. lily asked her mom, ” can i play with the ball? ” her mom said, ” no, lily. it’s too high up in the sky. ” lily was sad, but she didn’t give up. she went to her mom and said, ” mommy, can you help me get the ball? ” her mom said, ” sure, lily. let’s go get the ball. ” they went to the store and found a big, red ball. lily was so happy and said, ” thank you, mommy! ” they played with the ball all day long. when it was time to go home, lily said, ” mommy, i had so much fun playing with the ball. ” her mom smiled and said, ” i’m glad you had fun, lily. ” and they went home, happy and tired from playing with the ball. the end.

> once upon a time, there was a little boy who was very curious. he wanted to know what was inside, so he asked his mom. his mom said, ” let’s go to the park and find out! ” so they went to the park. when they got there, they saw a big tree. the tree had lots of leaves and branches. the boy was so excited! he ran over to it and started to play. he was having so much fun. but then, he saw something else. it was a big, green leaf! the boy was so happy. he ran

over to the tree and started to play. he had so much fun! he played with his new toy all day long. when it was time to go home, the boy was sad to leave the park. he was so happy to have a new friend. he couldn't wait to come back and play with his new toy again. he was so glad he was able to have a fun day with his new toy. the end.

D. Bilinear Finetuning

Architecture	
d_model	2048
d_hidden	5632
d_head	64
n_heads	32
n_key_value_heads	4
n_layer	22
n_ctx	2048
n_vocab	32000
Finetuning Parameters	
dropout	0.0
weight decay	0.0
batch size	8
sequence length	512
learning rate	$3 \cdot 10^{-6}$
optimizer	AdamW
schedule	None
tokens	105M-500M
Miscellaneous	
pos_emb	rotary

Table 4. Model architecture and training setup. Omitted parameters are the HuggingFace Trainer defaults.

Training Details We used a sequence length of 512, a batch size of 8, a fixed learning rate of $3 \cdot 10^{-6}$, no gradient accumulation, the AdamW optimizer with default settings, and trained with standard cross-entropy loss. We trained on sequences of FineWeb data filtered between 384 and 512 tokens. An oversight in our work is that the base model TinyLlama-1.1B was pretrained on a mix of RefinedWeb and StarCoder data, which has a different distribution than FineWeb. We finetuned the base model on 105M tokens of FineWeb data and included the result in the table below. Note that the base model finetuning had plateaued after 105M tokens, while the bilinear tuning had not plateaued yet at 500M tokens. While we cannot expect bilinear models to match SwiGLU performance, these results suggest that the gap between the bilinear tuned models and SwiGLU base models can be closed further. We emphasize that while bilinear tuning doesn't damage performance much, it may alter knowledge and circuits in a way we cannot predict. The ideal direction moving forward would be to use and study the bilinear variant.

	Train Loss	Val. Loss
Base Model	N/A	2.559
Bilinear Tuned (105M tokens)	2.310	2.450
Bilinear Tuned (500M tokens)	2.222	2.354
Finetuned Base Model (105M tokens)	2.199	2.294

Table 5. Train and Validation loss for TinyLlama-1.1B models on FineWeb data.

E. MNIST Eigenvectors

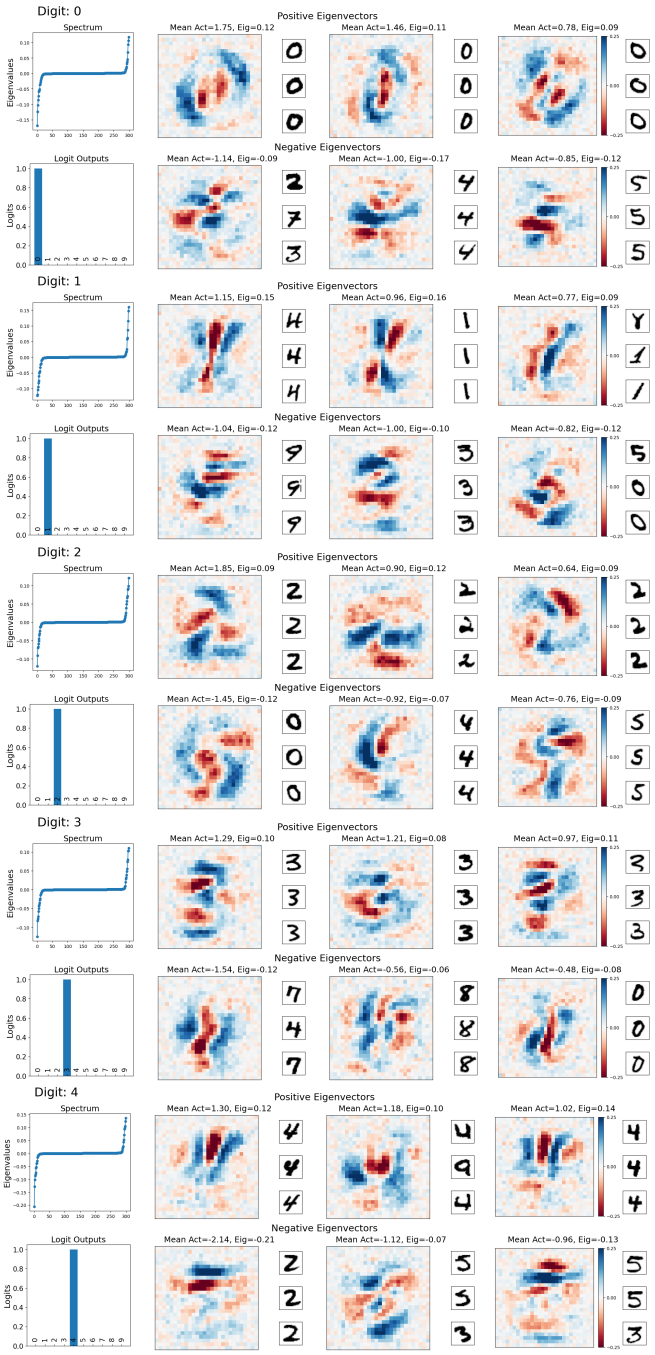

Figure 9. Eigenvectors for MNIST digits 0-4

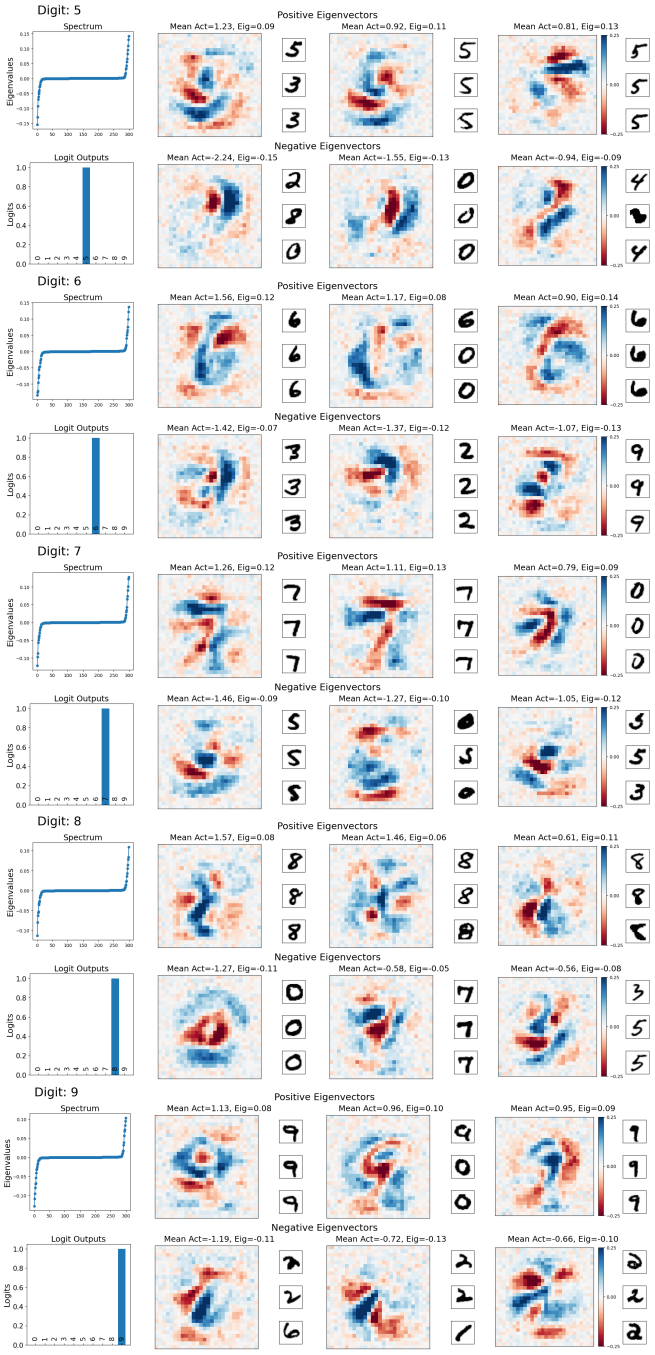

Figure 10. Eigenvectors for MNIST digits 5-9

F. Bigrams & Trigrams

Residual			MLP		
input	**output**	**value**	**input**	**output**	**value**
each	other	2.41	capt	##ured	1.14
exper	##ience	2.16	capt	##ained	1.08
amaz	##em	2.07	st	##alk	1.02
res	##ult	1.94	st	##ident	0.99
re	##que	1.93	st	##ace	0.97
e	##ff	1.90	st	##ale	0.90
un	##fort	1.85	pre	##ci	0.90
st	##itch	1.85	resp	##ond	0.88
bec	##om	1.85	pre	##par	0.85
incl	##ud	1.84	st	##ield	0.83
thank	you	1.82	con	##cer	0.82
bu	##dge	1.81	st	##ained	0.82
bran	##d	1.81	disag	##reed	0.81
’	s	1.81	gl	##itter	0.81
pl	##ucked	1.78	des	##ire	0.78
stra	##ight	1.76	con	##ut	0.77
purp	##ose	1.75	bl	##ouse	0.76
fa	##shion	1.75	bl	##ush	0.76
dir	##ect	1.75	capt	##ain	0.76
fl	##ight	1.74	somet	##ac	0.76

Table 6. Comparison of most salient bigrams in the weights of the Residual and MLP measured in a non-regularized model with d_model=512. We note that the MLP generally learns bigrams that are dependent on context, such as completions of very general tokens: st, pre, ... but tends to overcompensate towards that.

	Head 0.2			**Head 0.5**		
	virtual	direct	value	virtual	direct	value
0	saying	,	3.79	”	,	18.67
1	shouts	,	3.76	”	!	15.27
2	say	,	3.69	”	:	14.89
3	says	,	3.64	”	?	13.18
4	shouted	,	3.60	”	;	11.98
5	yelled	,	3.45	”	.	11.23
6	yell	,	3.40	”	-	10.58
7	shout	,	3.27	”	say	8.12
8	##laimed	,	3.25	”	\	7.19
9	asks	,	3.24	”	’	6.78
10	said	,	3.20	”	exclaimed	6.71
11	##laimed	:	3.20	”	/	6.32
12	said	:	3.18	”	when	6.29
13	shouted	:	3.18	.	,	5.75
14	shouting	,	3.17	”	says	5.71
15	says	:	3.15	”	wasn	5.36
16	whispered	,	3.11	”	shouted	5.32
17	saying	:	3.08	”	saying	4.98
18	asked	,	3.04	”	because	4.92
19	told	,	3.02	”	if	4.88

Table 7. Strongest weights for predicting a quote for both Head 0.2 and Head 0.5. Head 0.2 detects when it should open a quote from its context. Head 0.5 is much stronger and detects when it should close a quote but also encourages the opening of a new quote if there already has been one.

G. Tiny Stories Eigenvectors

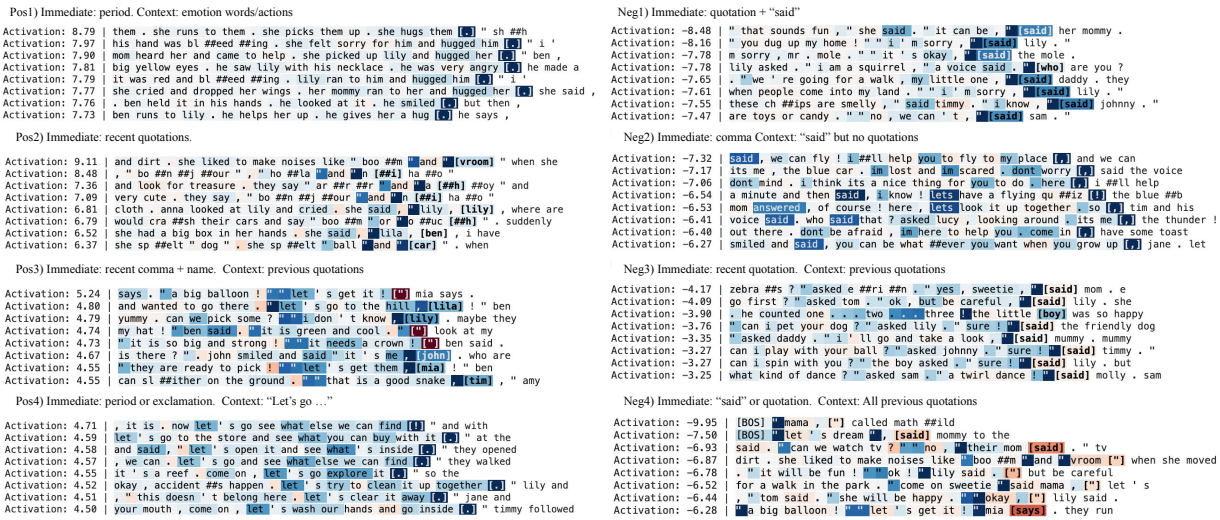

Figure 11. Eigenvectors for the quotation output token (") in a single-layer Tiny Stories model.

Figure 12. Eigenvectors for the output token "swim" in a single-layer Tiny Stories model. Note that some eigenvectors capture features that grammatically apply to most verbs, such as completing the phrase "love to _____" (Pos2) or completing the contraction "let's" (Neg2) where no verb would work.

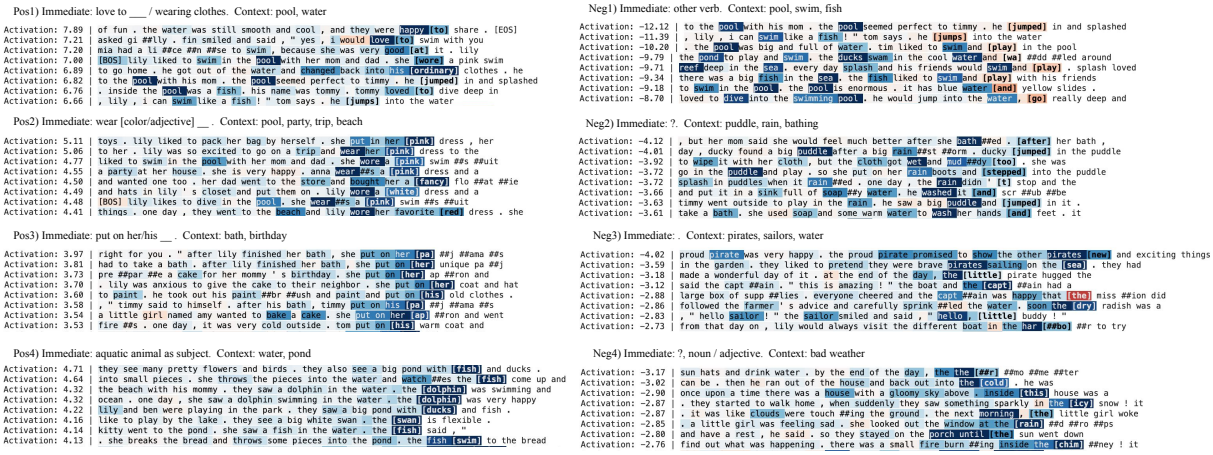

Figure 13. Eigenvectors in a single-layer Tiny Stories model for the "swim" unembedding minus the average unembedding of ["run", "climb", "eat", "see", "smell", "walk", "fly", "sit", "sleep"].